Università degli Studi di Pisa
Facoltà di Ingegneria
Corso di Laurea in Elettronica

Valutazione delle Prestazioni di Sistemi Multiprocessore Basata sull'Analisi di Tracce Reali

Tesi di Laurea

Roberto Giorgi

Relatori:
Prof. Cosimo Antonio Prete

Prof. Roberto Saletti

Anno Accademico 1994-95

Sommario

Indice delle Figure

Indice delle Tabelle

Obiettivo principale di questa tesi è stato quello di analizzare il comportamento di applicazioni in esecuzione su una workstation (realizzata con architettura multiprocessore a memoria condivisa e bus condiviso), per trarne informazioni che consentissero sia di guidare la progettazione di hardware e software, sia di valutare le prestazioni del sistema.

Il metodo utilizzato per fare questo è stato di procurarsi e/o produrre ciò che, a livello di bus, costituisce lo stimolo (o input) logico dell'elaborazione ovvero la sequenza di operazioni, detta *traccia*, generata da un insieme di applicazioni realmente in esecuzione sulla macchina (esistente o simulata) oggetto dello studio; questo tipo di tracce sono anche dette *tracce reali.* La traccia è 'un'impronta' del carico del sistema e disporre di un ampio spettro di tracce consente di valutare le prestazioni dell'architettura sotto studio al variare di uno qualsiasi dei parametri che la caratterizzano.

Da un lato è stato, quindi, creato un vero e proprio database di tracce, e dall'altro è stato messo a punto un *ambiente operativo* che consentisse non solo di tenere conto di tracce esistenti e prodotte dalle fonti più disparate (Carnegie Mellon University e MIT, per citarne alcune), ma anche di produrre, a partire dal software applicativo, la corrispondente traccia (grazie al tool TangoLite, fornito dall'Università di Stanford e adattato alle esigenze specifiche del presente lavoro).

Tale ambiente operativo, denominato ***Trace Factory***, consente di produrre la traccia di una data applicazione, analizzare tracce ed, eventualmente, inserire le caratteristiche necessarie ad ottenere una traccia che contenga tutte le informazioni normalmente presenti in una traccia prelevata dal bus con metodi hardware notevolmente più costosi e di minore flessibilità.

Ambiente di sviluppo

Il software è stato sviluppato in ambiente UNIX e portato sotto AIX 3.2.5, SunOS 4.1, Ultrix 3.2, Linux 1.2.3, FreeBSD 2.0. L'interfaccia grafica è stata sviluppata con il toolkit Tcl/Tk 3.2/7.1.

La scelta di operare in ambiente UNIX è dovuta principalmente alla notevole quantità di memoria allocata per analizzare i riferimenti di una traccia. Punto di svolta nella implementazione degli strumenti è stata la presa di coscienza del fatto che, sebbene le tracce - in alcuni casi - contenessero centinaia di milioni di riferimenti, pari a oltre 400MB di dati, gli spazi di indirizzamento effettivamente utilizzati erano dell'ordine di qualche MB. Nonostante questo, l'unico sistema in grado di: gestire queste quantità di memoria, operare in rete, fornire tutta una serie di strumenti di sviluppo del software - dai compilatori ai programmi di utilità per il reperimento delle informazioni nei file di testo - è risultato UNIX.

Inoltre, i tool forniti dalle altre università erano sviluppati ancora per questo ambiente. In particolare il software TangoLite è stato implementato solo su workstation con processore MIPS R3000. UNIX è risultato, almeno in ambito accademico, davvero il sistema operativo 'universale'.

Il compilatore preferenziale è stato il `gcc`, (GNU C Compiler) di cui è stata utilizzata l'ultima versione disponibile, la 2.6.3; esso è disponibile su ognuna delle piattaforme UNIX elencate ed ha un alto grado di affidabilità e di rispetto delle specifiche degli standard ANSI.

È stato, inoltre, utilizzato Tcl/Tk, un recente pacchetto software di notevole successo, che costituisce un sistema di programmazione per sviluppare e usare applicazioni con interfaccia grafica, avente il 'look and feel' di Motif e che poggia interamente sulla librerie intrinseche di X-Window, senza quindi dipendere dal toolkit di Motif stesso. Il pacchetto consente di sviluppare in tempi rapidi interfacce grafiche e di integrare in maniera naturale, attraverso uno script-language, vari tool software che compongono un'applicazione complessa; esso è quindi risultato ottimale per lo sviluppo dell'ambiente integrato di Trace Factory.

Database di tracce

Sebbene la produzione di software ben progettato, scritto accuratamente e ben testato sia stata una delle principali attività svolte durante il periodo di tesi, questa *non è* una tesi di ingegneria del software.

Al contrario il software è stato *lo strumento* con cui *implementare le idee* che convalidassero le scelte architetturali concernenti la categoria di sistemi multiprocessore presa in esame.

Da un punto di vista quantitativo, uno dei risultati macroscopicamente evidenti è stato l'allestimento di un data base di tracce, contenente oltre due GigaByte di dati, di cui la maggior parte in formato compresso; l'uso di tecniche di compressione, gestite anche dall'interno dei tool che compongono Trace Factory, ha ridotto l'occupazione di spazio e

di conseguenza il costo dei dispositivi di memorizzazione di massa, altrimenti necessari, di un fattore 10.

Il data base, di cui è allegata documentazione dettagliata, traccia per traccia (in appendice), consente di selezionare il carico per le simulazioni, che vengono effettuate, in generale, per valutare le prestazioni dell'architettura in esame; nel caso particolare queste tracce sono state ampiamente utilizzate per valutare e comparare le prestazioni di vari protocolli di coerenza per cache di sistemi multiprocessore fra cui il protocollo UCR3, sviluppato presso questo dipartimento [Prete95b].

Tracce-target

Il database è composto da tracce che possono soddisfare solo parzialmente i requisiti che ha una traccia rilevata direttamente dal bus di una macchina; nel caso peggiore può quindi essere necessario: 1) tracciare un numero di applicazioni sufficiente a rappresentare l'esecuzione di *n* processi; 2) estrarre la traccia dei riferimenti del nucleo del sistema operativo da tracce che la contengano; 3) rischedulare i processi e il nucleo sul numero di processori della macchina sotto studio. Le tracce così generate sono chiamate *tracce target* e gli strumenti da utilizzare per ottenerle saranno discussi in dettaglio nei capitoli 3 e 4.

Valutazione delle prestazioni delle architetture multiprocessore

I grandi nomi dell'elettronica e i maggiori esperti mondiali affermano che (fonte: International Symposium on Computer Architecture del giugno 1995), entro cinque anni, la tecnologia microelettronica dell'anno 2000 permetterà di integrare su un unico chip oltre 15 milioni di transistor su una superficie di circa 400 mm^2; questo consentirà di sfruttare non solo il parallelismo a livello di istruzioni, come già avviene nell'esecuzione 'fuori ordine' degli attuali microprocessori, ma anche di sfruttare il parallelismo a livello di thread (o processo-leggero), utilizzando 5-6 degli attuali processori su un unico chip.

La concentrazione di notevoli sforzi nell'evitare i conflitti nell'accesso alla memoria e ridurre il più possibile l'utilizzazione del bus comune da parte dei singoli processori della macchina multiprocessore appare, quindi, ampiamente giustificata. Se si considera, inoltre, che grosse memorie cache non possono in ogni caso incrementare le prestazioni dei multiprocessori oltre un certo limite, si rende necessaria una accorta gestione del bus e una conseguente ricodifica delle applicazioni per ridurre la condivisione e per tenere conto di eventuali nuove possibilità offerte dall'hardware.

Considerando la materia del dibattere, non c'è oggi più spazio per mettere a punto nuove architetture semplicemente disegnando uno 'schemino con carta e penna'. Ogni valutazione seria ed obiettiva delle scelte implementative di una macchina deve passare per la valutazione basata sull'effettivo carico della macchina stessa, costituito dalle applicazioni reali che questa dovrà eseguire e di cui le tracce (reali) sono la fedele traduzione in termini di segnali sul bus comune.

Roberto Giorgi, 19 luglio 1995.

1 I Sistemi Multiprocessore

I sistemi multiprocessore rappresentano la naturale evoluzione degli attuali sistemi monoprocessore, oggi diffusi come qualsiasi altro prodotto dell'elettronica sotto il nome di *Personal Computer*.

In questo capitolo verrà presentata la categoria dei sistemi multiprocessore, introducendo alcuni dei concetti ad essa correlati, in quanto oggetto principale di studio in questa tesi.

1.1 Perché il multiprocessore

A differenza di altre architetture, basate su più processori interconnessi fra di loro, *il sistema multiprocessore*, costituito da un unico bus che collega i vari processori e da una memoria comune, presenta il notevole vantaggio di *poter* essere programmato 'come se' fosse una macchina monoprocessore; il sistema operativo può facilmente effettuare il bilanciamento del carico sulle risorse disponibili nella macchina.

Il multiprocessore a bus comune ha il vantaggio di avere il metodo più semplice per interconnettere processori diversi con un ottima velocità di comunicazione.

Gli attuali sistemi operativi, anche commerciali, stanno facendo un uso intensivo della multiprogrammazione; un'applicazione si presenta spesso 'naturalmente' suddivisa in processi e i processi suddivisi in thread (o processi leggeri). Questi ultimi divengono, quindi, le unità elementari del parallelismo che possono essere assegnate ad uno qualsiasi dei processori del sistema per poter agire sui dati privati o comuni senza bisogno di particolari accorgimenti, salvo le dovute sincronizzazioni presenti anche in un sistema monoprocessore.

I multiprocessori si comportano meglio di ad altre architetture basate su più processori quali i *multicomputer*, almeno quando il numero di processori in gioco non è estremamente elevato (quantitativamente inferiore a qualche decina); ma, se si utilizza un gran numero di processori, il costo della macchina può giustificare anche un notevole sforzo per programmarla con un paradigma diverso da quello utilizzato nelle macchine monoprocessore; in questo caso però si entra in una categoria di sistemi ad altissime prestazioni e anche ad altissimo costo e diffusione limitata, ovvero una categoria assai diversa da quella in esame.

Infine, fra 5 anni la tecnologia microelettronica consentirà di integrare, su un unico chip di 400 mm^2, oltre 15 milioni di transistor rendendo possibile mettere insieme 5-6 degli attuali processori e ottenere così il 'multiprocessore su singolo chip'.

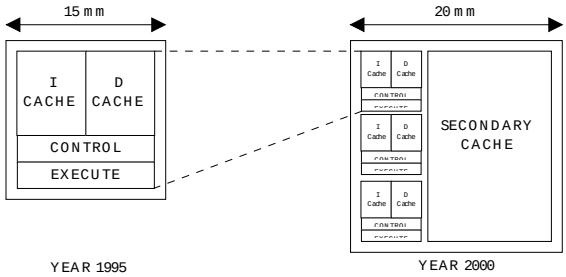


Figura 1–1. Evoluzione della tecnologia microelettronica dal 1995 al 2000.

I processori dell'attuale generazione sfruttano il cosiddetto parallelismo a livello di istruzione; ad esempio, MIPS-R10000 e Intel-P6 utilizzano l'esecuzione 'fuori ordine' delle istruzioni macchina per aumentare le prestazioni. Nel futuro la possibilità di avere più processori sullo stesso chip consentirà di sfruttare il parallelismo a livello di thread [Mudge95].

Già oggi personal computer o workstation, a costo relativamente basso, con 2 processori (il *biprocessore*) sono implementati e commercialmente disponibili. Sistemi operativi che già supportano questa possibilità sono Windows NT e Sun Solaris.

1.2 Evoluzione dell'architettura dei computer

Lo studio dell'architettura dei computer comprende sia l'organizzazione hardware che i requisiti del software o del sistema di programmazione.
Il programmatore assembler, interagisce con l'architettura del computer attraverso l'astrazione fornita dal set di istruzioni che consta di codici delle operazioni (opcode), modi di indirizzamento, registri, memoria virtuale e così via. Da un punto di vista dell'implementazione hardware la macchina astratta è organizzata in termini di processore, cache, bus, microcodice, pipeline, memoria fisica e così via. Dunque, sebbene lo studio architetturale riguardi anche la struttura del set di istruzioni, in questa tesi l'architettura verrà presa in considerazione soprattutto per quanto riguarda l'organizzazione delle unità che compongono la macchina.

Durante gli ultimi 40 anni l'architettura dei computer non ha avuto veri e propri sconvolgimenti; si è osservata piuttosto una continua evoluzione.

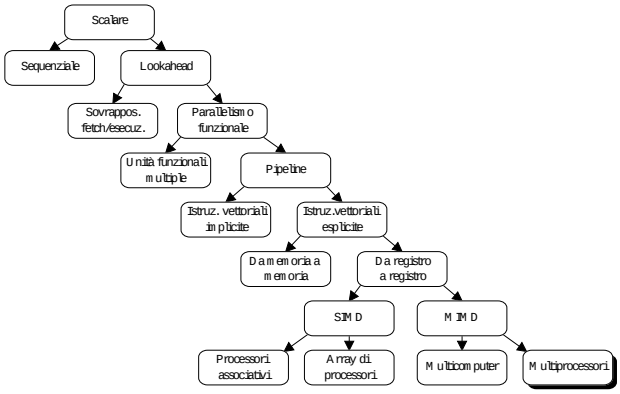


Figura 1–2. Evoluzione dell'architettura dai computer sequenziali scalari ai multiprocessori.

Si parte dall'architettura di Von Neumann basata sull'esecuzione sequenziale di dati scalari. Questa architettura è però lentissima perché si deve *prima* prelevare una istruzione della memoria e *poi* eseguirla: sono state così introdotte le tecniche lookahead, quali il prefetch, che consentono di sovrapporre la fase di fetch e quella di esecuzione.

Lo stadio successivo è stato quello di disporre di più unità funzionali che potessero eseguire simultaneamente le operazioni.

Più unità funzionali possono essere utilizzate sia per realizzare un *parallelismo funzionale,* che un parallelismo di tipo *pipeline* in cui si effettua, ad esempio, esecuzione dell'istruzione, calcoli aritmetici, accessi in memoria.

Alcuni hanno poi consentito l'uso esplicito di *istruzioni vettoriali* avendo nello stesso processore varie pipeline che possono eseguire concorrentemente (*processori vettoriali*).

Ci sono due tipi di processori vettoriali: quelli che mantengono un costante flusso di dati *da memoria a memoria* e quelli che usano una serie di *registri vettoriali* per interfacciare le pipeline funzionali alla memoria.

Quest'ultima architettura può essere suddivisa in due grandi categorie quella delle *macchine SIMD* e delle *macchine MIMD*, secondo la classificazione introdotta da Flynn nel 1972 [Hwang93].

Macchine SIMD e MIMD

Le macchine SIMD sfruttano il *parallelismo spaziale* dei dati piuttosto che il *parallelismo funzionale* come nei computer basati su pipeline; nella macchina SIMD, più elementi di elaborazione (o PE, processing elements) sono sincronizzati dallo stesso controller. Tipicamente le macchine SIMD sono suddivise in macchine costituite da processori associativi e macchine con parallelismo vettoriale.

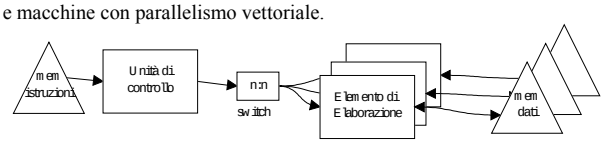


Figura 1–3. Architettura SIMD.

Le macchine MIMD sono i computer a parallelismo intrinseco e ci sono due grandi categorie, quella delle macchine a *memoria condivisa* o *multiprocessore* e quella delle macchine a *memoria distribuita* o *multicomputer*; nelle prime la

comunicazione fra i processori avviene attraverso *variabili condivise*, nelle seconde la comunicazione avviene tramite lo *scambio di messaggi* (ed è proprio questo il motivo della minore efficienza di queste ultime macchine, per un numero di processori non elevato).

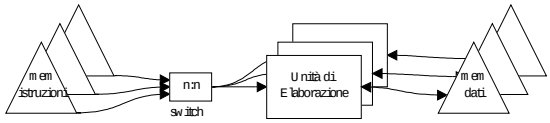


Figura 1–4. Architettura MIMD a memoria condivisa o multiprocessore.

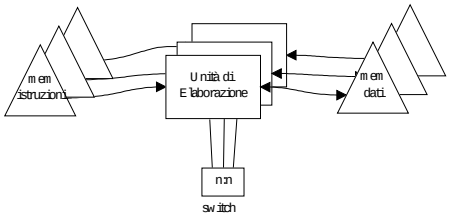


Figura 1–5. Architettura MIMD a memoria distribuita o multicomputer.

Il *switch* che collega le unità di elaborazione (o PU, Processing Unit) può avere le topologie più disparate; nei multiprocessori è generalmente un *bus condiviso*; ed è questo il caso analizzato in particolare in questa tesi.

1.3 Le macchine multiprocessore

Tre tipi di architetture multiprocessore sono oggi utilizzate a seconda del modello di accesso alla memoria condivisa:

- UMA, Uniform Memory Access, in cui tutti i processori hanno uguale tempo di accesso alla memoria;
- NUMA, Non-Uniform Memory Access, in cui il tempo di accesso ad un dato varia a seconda della sua posizione in memoria;
- COMA, Cache Only Memory Architecture, in cui la memoria distribuita, ma virtualmente condivisa, è costituita da sola memoria cache.

Il modello UMA

Nella seguente figura è mostrata la struttura tipica di una macchina multiprocessore con modello UMA.

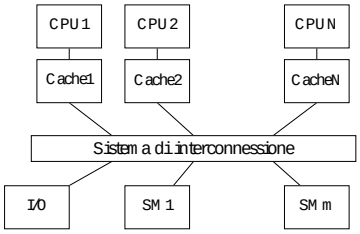


Figura 1–6. Multiprocessore con modello UMA; SM = Shared Memory.

Tutti i processori hanno lo stesso tempo di accesso a tutti gli indirizzi della memoria comune globale. Ogni processore può utilizzare una cache privata. Anche le periferiche vengono in qualche maniera condivise.

I sistemi multiprocessore di questo tipo vengono anche chiamati 'ad accoppiamento stretto' a causa dell'alto grado di condivisione delle risorse. Il sistema di interconnessione può essere un *bus comune*, un *crossbar-switch* o, in generale, una *rete multistadio*.

La maggior parte dei costruttori di computer hanno una estensione multiprocessore (MP) della loro linea di prodotti monoprocessore (UP, o UniProcessor).

Il modello UMA si presta ad applicazioni di tipo generale (general-purpose) e time-sharing da parte di più utenti. Può essere utilizzata per aumentare la velocità di esecuzione di un singolo grande programma nelle applicazioni in cui il tempo è un parametro critico. Per coordinare il parallelismo, la sincronizzazione e la comunicazione fra i processori vengono effettuate tramite l'uso di variabili condivise nella memoria comune.

Quando tutti i processori hanno anche uguale tempo di accesso a tutti i dispositivi periferici, il sistema è chiamato *multiprocessore simmetrico*. In questo caso tutti i processori hanno la stessa capacità di eseguire i programmi e il

nucleo del sistema operativo, così come le routine di servizio dell'I/O.

Nei multiprocessori *asimmetrici,* solo un sottoinsieme dei processori ha la possibilità di eseguire programmi. Un processore principale si occupa invece di eseguire il nucleo e le routine di I/O; i restanti processori non hanno possibilità di effettuare l'I/O e per questo sono chiamati *processori attaccati* (AP o Attached Processor) o secondari. I processori secondari eseguono il codice utente sotto la supervisione del processore principale. Sia nella configurazione MP che AP si ha la condivisione della memoria fra tutti i processori del sistema.

Nella presente tesi si è considerato in particolare un modello UMA in cui la rete di interconnessione è un bus-condiviso.

Il modello NUMA

La macchina NUMA, in cui il tempo di accesso ad un dato dipende dalla posizione del dato stesso all'interno della memoria condivisa, può essere implementata in due diverse maniere.

Un caso è quello in cui tutti i processori hanno accesso a tutte le varie memorie locali, che sono viste come se fossero un'unica memoria virtuale condivisa; ogni memoria ha le proprie caratteristiche e tempi di accesso.

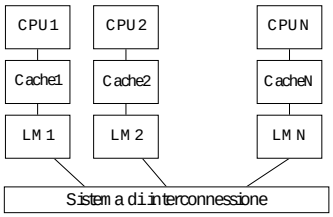


Figura 1–7. Primo tipo di multiprocessore con modello NUMA; LM = Local Memory,

L'accesso a memorie pertinenti nodi remoti è tipicamente maggiore dell'accesso alla memoria locale. Un esempio di macchina esistente basata su questo modello è il multiprocessore Butterfly BBM TC-2000.

Oltre a questa memoria condivisa distribuita può essere aggiunta della memoria globalmente condivisa. In questo caso esistono *tre* cammini di accesso alla memoria: il più veloce è quello verso la memoria locale; il più lento è quello verso le memorie locali remote; l'intermedio è quello verso la memoria globalmente condivisa. Esiste poi la possibilità di modificare i due modelli principali ora analizzati per ottenere un qualsivoglia 'mescolamento' di memoria condivisa e privata.

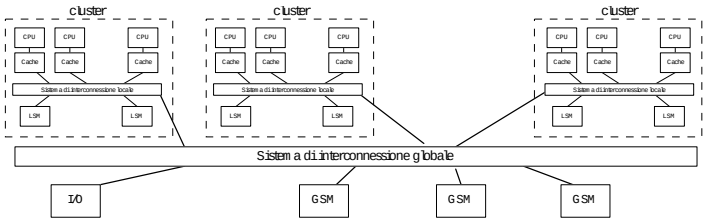


Figura 1–8. Secondo tipo di multiprocessore con modello NUMA;
LSM = Local Shared Memory, GSM = Global Shared Memory

Il modello riportato in figura è relativo ad un multiprocessore con struttura gerarchica. I processori sono raggruppati in *cluster*; ogni cluster è di per sé un multiprocessore di tipo UMA o NUMA. L'intero sistema può essere considerato un multiprocessore NUMA. Tutti i processori appartenenti allo stesso cluster possono uniformemente accedere alla memoria locale condivisa LSM. Tutti i cluster hanno uguale di tempo accesso alla memoria globale condivisa; però, il tempo di accesso alla memoria condivisa all'interno dei cluster è minore di quello alla memoria condivisa globale.

Un esempio di macchina di questo tipo è il multiprocessore Cedar costruito all'Università dell'Illinois, in cui ogni cluster è un multiprocessore Alliant FX/80.

Il modello COMA

Esempi di macchine COMA sono la Data Diffusion Machine dell'Istituto di Scienza dell'Informazione Svedese e la KSR-1 della Kendall Square Research.

Il modello COMA è un particolare caso di modello NUMA in cui le memorie distribuite principali sono state convertite in memoria cache. Non c'è gerarchia fra le memorie nei nodi di ogni processore. Tutte le cache formano un unico spazio di indirizzamento globale. Le cache remote sono assistite dalle directory distribuite delle cache per

accedere alle locazioni di memoria non appartenenti al nodo ove si origina una richiesta di accesso.

A seconda della rete di interconnessione utilizzata, possono essere presenti delle directory gerarchiche per facilitare l'individuazione della copia richiesta.

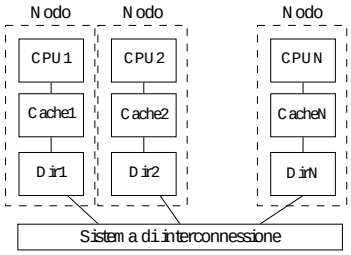


Figura 1–9. Multiprocessore con modello COMA; Dir = directory della cache.

La posizione iniziale dei dati non è critica, perché i dati possono eventualmente migrare dove devono essere usati più frequentemente.

Altri modelli

Oltre ai basilari modelli UMA, NUMA e COMA esistono altre varianti di sistemi multiprocessore. Per esempio, esiste un modello CC-NUMA, Cache-Coherent Non-Uniform Memory Access che può essere utilizzato quando si disponga di directory delle cache e memoria condivisa distribuita; questo modello è stato utilizzato nella macchina DASH sviluppata a Stanford e nella macchina ALEWIFE, sviluppata al Massachusetts Institute of Technology.

1.4 Limiti dei sistemi multiprocessore

I multiprocessori sono adatti per applicazioni multiprogrammate e multiutente, di tipo generale, dove il requisito principale è la programmabilità. Un difetto dei multiprocessori è però la mancanza di scalabilità: è piuttosto difficile costruire macchine multiprocessore con tutta la memoria centralizzata e d'altra parte, se ci sono nodi remoti, la latenza dell'accesso alla memoria limita notevolmente le operazioni. Il bus, inoltre, diventa presto il 'collo di bottiglia del sistema'.

Anche la difficoltà di inserire in una sola macchina tutto l'hardware, con necessità di ricorrere ad opportuni sistemi di raffreddamento, limita la scalabilità della macchina. In futuro questi problemi saranno però superati dall'introduzione di chip capaci di integrare *vari* processori e memoria.

Resta un punto, sul quale lavorare per aumentare le prestazioni dei sistemi multiprocessore, ed è quello di diminuire gli accessi alla memoria condivisa, in parte ricodificando le applicazioni e in parte eliminando le transazioni inutili che possono essere causate dalle operazione che mantengono la coerenza delle cache; in questa direzione si rivolgono protocolli di coerenza come UCR3 sviluppato presso questo dipartimento [Prete95b].

Neppure grosse memorie cache riescono a far aumentare le prestazioni dei multiprocessore quanto la riduzione del traffico sulla rete di interconnessione [Gee93]. Finché la macchina non è portata ad avere grossi carichi sul bus sembra che tutti i protocolli di coerenza abbiano lo stesso comportamento. Alcuni protocolli cominciano a saturare il bus in corrispondenza di valori molto più bassi del numero di processori di cui è costituita la macchina.

Una cifra di merito, che può essere utilizzata per valutare l'effettiva capacità che può fornire un sistema multiprocessore è il Global System Power (GSP), definito come:

$$GSP = \sum_{k=1}^{N} U_k$$

essendo N il numero di processori di cui è costituito il sistema multiprocessore e U_k il fattore di utilizzazione del k-esimo processore, espresso in percentuale (come numero da 0 a 100).

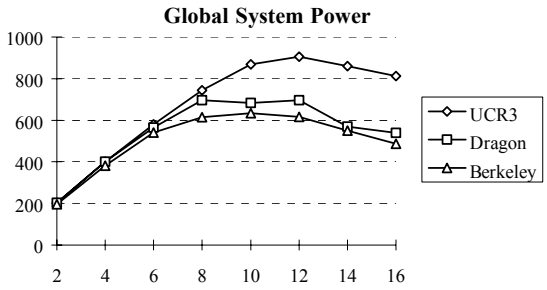


Figura 1–10. Global System Power ottenuto utilizzando diversi protocolli per mantenere la coerenza delle cache nei sistemi multiprocessore.

Per migliorare l'architettura di un sistema multiprocessore bisogna, quindi, andare ad operare ad un livello necessariamente fine, osservando cosa succede veramente sulla rete di interconnessione, quando la macchina sta eseguendo dei programmi applicativi realmente utilizzati e non semplici benchmark.

Il metodo universalmente adottato per analizzare il comportamento di una macchina, con un carico di questo tipo, consiste nel prelevare la sequenza di riferimenti generati dai vari processori che accedono alla rete di interconnessione; tale sequenza, detta *traccia* (o *traccia reale* perché riguardante la reale esecuzione di programmi sulla macchina sotto studio) è composta dalla lista degli indirizzi acceduti con il relativo tipo di operazione e l'identificatore del processore che ha effettuato l'accesso.

Nel capitolo successivo verrà spiegato cos'è una traccia e come si possono valutare le prestazioni utilizzando tracce. Nei capitoli seguenti, verrà presentato l'ambiente operativo Trace Factory, che consente di operare sulle tracce attraverso vari strumenti, e discussi in dettaglio i vari strumenti. Infine, saranno presentati alcuni risultati delle analisi effettuate.

2 Le Tecniche di Tracciamento e la Valutazione delle Prestazioni

Nel seguito di questa tesi verrà considerata, in particolare, la classe dei multiprocessori UMA, ovvero con bus comune, memoria condivisa ed uguali tempi di accesso alla memoria.
La domanda di potenza di calcolo fornita da un Personal Computer o da una Workstation si sposta sempre verso livelli più elevati. I sistemi multiprocessore risultano la naturale evoluzione delle macchine ad alto rapporto prestazioni/prezzo. D'altra parte, un sistema multiprocessore non viene realmente sfruttato per quelle che sono le sue potenzialità, senza un'accurata progettazione delle modalità di interazione dei vari processori che lo compongono.

Figura 2–1. Le prestazioni dei computer non soddisfano mai...

Per capire come migliorare le prestazioni di un multiprocessore, uno dei metodi più utilizzati consiste nell'esaminare *la lista dei riferimenti che vengono generati sul bus comune* ovvero *la traccia* dell'esecuzione di applicazioni rappresentative del reale carico del sistema. In questo capitolo verranno discussi pregi e difetti della tecnica di tracciamento per valutare le prestazioni di una suddetta architettura di sistema multiprocessore.

2.1 Cos'è una traccia

La *traccia* è una *lista dei riferimenti generati sul bus* della macchina sotto studio. Un *riferimento* è l'insieme delle informazioni che caratterizzano la *posizione* del dato in memoria (ovvero il suo *indirizzo*), il *tipo* di operazione (read, write, lock) e *l'identificatore del processore* che l'ha generata.

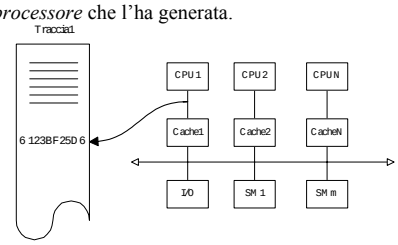


Figura 2–2. Traccia del processore 1.

Il sistema da cui si prelevano le tracce è denominato *sistema tracciato*.
La traccia può essere prelevata direttamente sul bus oppure fra processore e cache; se interessa simulare anche le cache è auspicabile che la traccia si riferisca agli indirizzi generati direttamente dal processore.
Gli odierni processori contengono una unità di gestione della memoria virtuale (MMU), memoria cache ed effettuano un'esecuzione che spesso non rispetta l'ordine originario delle istruzioni macchina di cui è composto il codice binario del programma in esecuzione. Quindi, per avere delle tracce da riutilizzare in svariate simulazioni architetturali, è auspicabile che queste siano relative agli *indirizzi virtuali* generati dal programma applicativo; esistono metodi *basati sull'instrumentazione software* che consentono questa possibilità e che verranno discussi successivamente.
Le tracce possono anche essere generate *sinteticamente* utilizzando dei generatori che modellino il comportamento delle applicazioni.

2.2 Valutazione di prestazioni basata su analisi di tracce reali

Le tracce consentono di:

- predirre il comportamento e le prestazioni di un sistema;
- simulare un sistema, anche non esistente, utilizzandole come carico.

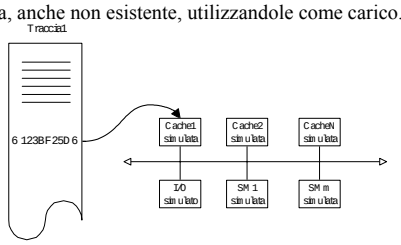


Figura 2–3. Utilizzazione della traccia del processore per la simulazione.

I tre metodi più usati - in ordine crescente di accuratezza, costo e difficoltà di utilizzazione - per predirre le prestazioni di una macchina sono:

- i modelli matematici;
- la simulazione basata su tracce (trace-driven);
- la costruzione di prototipi hardware.

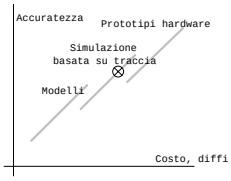


Figura 2–4. Comparazione dei metodi di predizione delle prestazioni.

I *modelli matematici* vanno da semplici calcoli sulla struttura delle applicazioni, a sistemi estremamente complessi che tengono conto di moltissime variabili. Questi possono essere utilizzati per una previsione molto teorica e per comparare varie architetture a parità di condizioni. L'attendibilità può essere, tuttavia, piuttosto scarsa.

I *prototipi hardware* predicono accuratamente i parametri hardware, ma il software che gira su questi prototipi può essere sviluppato solo in fase embrionale o essere costituito da programmi giocattolo che girano in assenza di sistema operativo. I prototipi, inoltre, hanno lo svantaggio di avere poca flessibilità nel consentire la messa a punto.

La qualità delle *simulazioni basate sulle tracce* (o trace-driven) dipende moltissimo dall'accuratezza della traccia e dalla fedeltà della simulazione. Le simulazioni possono variare da semplici modelli dell'hardware a modelli molto dettagliati e complessi.

Le simulazioni possono essere pilotate anche da tracce sintetiche, che ricalchino il comportamento delle tracce reali del sistema oggetto dello studio. Spesso non è molto chiaro come generare le tracce sintetiche e ci si affida parzialmente all'aleatorietà. In particolare, i dati su cui si modellano le tracce sintetiche non riescono a tenere in considerazione o la località spaziale o quella temporale.

La simulazione basata sulle tracce reali, invece, *consente di rappresentare accuratamente ciò che verrà eseguito sulla macchina sotto studio*. Ad esempio, le tracce fornite dalla Carnegie Mellon University (CMU) possono essere classificate col punto indicato con una crocetta nel precedente grafico [Vashow93]. Queste tracce, in particolare, sono fra le più accurate fra quelle utilizzate durante il lavoro relativo a questa tesi: contengono molti più riferimenti di tracce precedentemente prodotte da altri gruppi, contengono i riferimenti del sistema operativo, riguardano sistemi multiprocessore e contengono le sincronizzazioni fra i processori.

Un metodo ancora non esplorato è quello di utilizzare tracce prodotte da un dato sistema reale per produrre un secondo set di tracce relative ad un sistema anche non esistente con una architettura anche profondamente diversa. Questo è l'argomento principale di questa tesi e per esplorarlo è stato realizzato l'ambiente operativo ***Trace Factory***, la *fabbrica delle tracce*.

2.3 Validità e distorsioni di una traccia

Una traccia è *distorta* se gli indirizzi in essa contenuti differiscono da quelli generati dall'esecuzione del programma.

Una traccia è *valida* se l'insieme dei possibili comportamenti che sono riflessi dalla traccia del programma non differisce dall'insieme dei possibili comportamenti del programma originario.

Sfortunatamente, questa definizione di validità è quasi impossibile da verificare in quanto la validità varia con il comportamento che la traccia dovrebbe riflettere. Si cerca allora di mantenere minima la distorsione e massimizzare la validità.

Le distorsioni possono essere classificate in tre categorie [Stunkel91]:

- distorsioni *del pattern di esecuzione*;
- distorsione *del tempo di attesa*;
- distorsione *dell'ordine degli accessi*.

La *distorsione del pattern di esecuzione* si ha quando ci sono dei task schedulati dinamicamente e viene modificato l'ordine dei punti di sincronizzazione dei task in modo che risulta diverso da quello del programma non instrumentato.

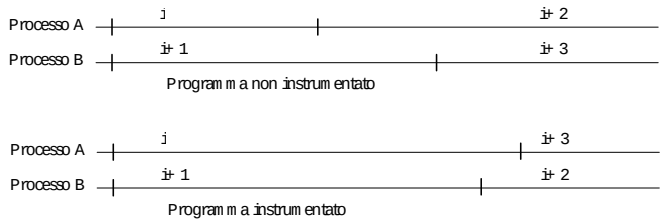


Figura 2–5. Distorsione dei pattern di esecuzione; le istanze i+2 e i+3 si sono scambiate di posto nel programma instrumentato; il risultato è una traccia 'modificata'.

I modelli di programmazione che usano *processi leggeri* (thread) a *grana fine* sono particolarmente sensibili a questa distorsione. Se la validità della traccia è influenzata da questa distorsione si può mantenere al minimo la dilatazione dovuta all'instrumentazione oppure rallentare l'esecuzione dell'intero sistema per conservare la temporizzazione originaria.

La *distorsione del tempo di attesa* si ha quando l'instrumentazione modifica il numero di iterazioni del ciclo di attesa attiva di un processo che aspetta una sincronizzazione. Il tempo di attesa al punto di sincronizzazione può variare con le caratteristiche della particolare macchina, quali il tipo di cache o il tipo di rete di interconnessione.

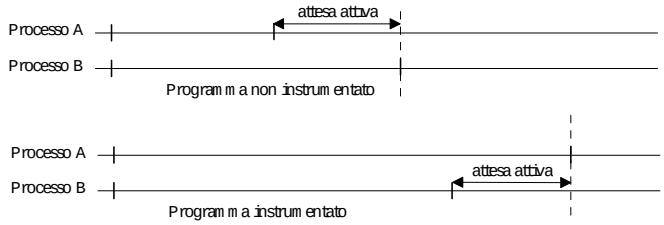


Figura 2–6. Distorsione del tempo di attesa; nel programma instrumentato il processo B raggiunge per primo la barriera; risulta una modifica del numero di accessi in memoria durante i loop di attesa attiva dei processi A e B.

La soluzione migliore consiste nel reimplementare la sincronizzazione come parte del simulatore che leggerà la traccia; ciò significa che la traccia deve includere dei record di sincronizzazione. In questo modo l'attesa attiva può essere correttamente rigenerata dal simulatore.

La *distorsione dell'ordine degli accessi* è la distorsione di grana più fine che può essere introdotta in una traccia. Il comportamento di un sistema multiprocessore è determinato dall'esatto ordine degli accessi in memoria. Una distorsione anche piccola come lo scambio di temporizzazione fra due accessi può incidere sulle prestazione del sistema.

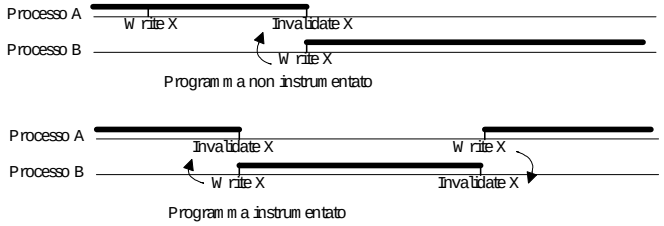


Figura 2–7. Distorsione dell'ordine degli accessi. Si supponga di utilizzare un protocollo di coerenza basato sull'invalidazione, con il processo A che inizialmente possiede il blocco X; nel programma instrumentato la scrittura del blocco X nella cache di A avviene dopo la scrittura del blocco X nella cache di B, causando una invalidazione in più del caso reale.

Fortunatamente, questa distorsione non influenza significativamente la validità di una traccia; gli accessi influenzati sono solo quelli relativi a variabili condivise che non devono seguire l'ordine imposto dalle barriere di sincronizzazione, ma possono avvenire in un ordine qualsiasi nei modelli di programmazione normalmente usati.

2.3.1 Come convivere con le distorsioni

Prima di tutto si cerca, ovviamente, di mantenere al minimo la distorsione. In seconda istanza, si possono inserire nella traccia delle informazioni temporali che possono essere utilizzate dal simulatore per ricreare una traccia valida. Le tracce possono così essere mantenute in file separati ognuno relativo ad un processore e saranno rielaborati nel corretto ordine dal simulatore. La maggior parte dei sistemi di estrazione delle tracce utilizza questa tecnica.

In Trace Factory il sistema di tracciamento usato per produrre le tracce è basato sul software TangoLite [Herrod95]. TangoLite è basato, a sua volta, sull'instrumentazione del software: le tracce sono generate su una macchina monoprocessore ma possono essere relative ad un sistema multiprocessore anche astratto. TangoLite, di cui si parlerà più in dettaglio in un capitolo successivo, consente di specificare quali eventi tracciare e i modelli di memoria e di sincronizzazioni che devono essere usati nel simulatore incorporato. L'uso che se ne è fatto in questa tesi prescinde dal simulatore incorporato; si è sfruttato, invece, il sistema di tracciamento, modificandolo in base alle esigenze specifiche del presente lavoro. TangoLite fornisce un ambiente flessibile per effettuare il tracciamento con una bassa distorsione e un basso costo.

Altre tracce di cui si dispone sono state ricavate con metodi di estrazione diretta dall'hardware e sono state fornite da altre università, come le tracce CMU.

Nel capitolo successivo si discuteranno i sistemi di tracciamento e si analizzerà più in dettaglio l'ambiente operativo di Trace Factory.

3 Trace Factory
La Fabbrica delle Tracce

Ottenere tracce reali che godano delle proprietà di accuratezza, completezza di copertura, facilità di estrazione, flessibilità di variazione dei parametri, contenenti i riferimenti del sistema operativo e delle applicazioni multiprogrammate è un obiettivo altamente auspicato in ogni ambiente di ricerca o progettazione che si proponga di effettuare una valutazione scientifica delle prestazioni di un sistema di elaborazione senza dover costruire e testare prototipi hardware.

Trace Factory rappresenta il tentativo di mettere a punto un ambiente di produzione delle tracce che soddisfino i precedenti requisiti.

3.1 Definizione dei requisiti delle tracce

Si analizzano ora, in dettaglio, le caratteristiche che si vogliono ottenere nelle tracce prodotte con Trace Factory.

3.1.1 Accuratezza della traccia prodotta

Il primo e più importante obiettivo da considerare nella produzione della traccia è che il sistema di estrazione non alteri qualche aspetto dell'esecuzione dell'applicazione; in altri termini la traccia deve rappresentare *fedelmente* il comportamento del relativo programma.

Poiché nel tracciamento si desiderano ricavare informazioni ad un livello di dettaglio estremamente fine (le operazioni in memoria rispetto al numero di cicli di clock costituiscono una percentuale significativa), il problema del disturbo provocato dalla sonda non è assolutamente banale.

Soprattutto nella categoria di sistemi presa qui in esame – i sistemi multiprocessore a memoria condivisa – il non rispetto dell'ordinamento originario degli accessi in memoria *può* cambiare la sequenza di esecuzione di uno o più processori.

Ogni motivo per il quale la traccia raccolta differisca dalla traccia effettiva è stato chiamato, nel precedente capitolo, *distorsione* della traccia. Spesso le distorsioni sono volutamente introdotte per diminuire il costo di tracciamento; la traccia può ritenersi comunque *valida* se la distorsione introdotta non altera le informazioni di interesse inerenti il comportamento dell'applicazione che si desidera tracciare.

3.1.2 Completezza di copertura

Spesso non è sufficiente campionare i riferimenti ad intervalli regolari, poiché si possono perdere accessi significativi per la completezza logica di un'operazione.

Un'altra forma di perdita di copertura si ha qualora il sistema di rilevamento possa registrare solo un sottoinsieme degli indirizzi effettivamente generati da un processore (ad esempio possono essere esclusi gli indirizzi di I/O).

La completezza di copertura va anche considerata in termini di numero di riferimenti che costituiscono la traccia: certi aspetti dell'applicazione possono emergere solo su un tempo di esecuzione sufficientemente lungo e certi aspetti dell'hardware oggi disponibile – come cache di ampie dimensioni – possono essere analizzati in condizioni limite solo con una grossa quantità o densità di riferimenti [Borg90].

3.1.3 Facilità di estrazione della traccia o costo

Questo requisito può essere interpretato sia in termini di costo della messa a punto del sistema di tracciamento, che in termini di accessibilità delle informazioni che si desiderano raccogliere. I metodi basati sull'hardware vincolano la raccolta dei dati alla sola macchina sotto test e consentono di tracciare solo gli indirizzi fisici, senza la possibilità di valutare il sistema a prescindere da eventuali dispositivi quali cache on-chip, buffers, code di prefetch; un altro inconveniente è la limitazione nell'analizzare architetture diverse da quella per cui l'hardware è stato costruito.

3.1.4 Flessibilità di variazione dei parametri

Nella valutazione delle architetture, può essere necessario esaminare il comportamento del sistema al variare del numero dei processi, al variare del numero di processori, al variare delle caratteristiche hardware o dei protocolli utilizzati nelle memorie cache, al variare del costo delle operazioni o al variare delle politiche di scheduling del sistema operativo. Tutto questo non è possibile – o è solo parzialmente possibile – nei sistemi di tracciamento basati su tecniche hardware.

3.1.5 Riferimenti del sistema operativo e delle applicazioni multiprogrammate

Molti sistemi di tracciamento esistenti consentono di raccogliere solo i riferimenti utente; ma studi sulle prestazioni delle cache [Agarwal86] hanno dimostrato la presenza di errori quando si ignorino i riferimenti prodotti dal nucleo del sistema operativo.

La presenza di carichi multiprogrammati è altresì importante proprio negli studi sulle cache dove le sincronizzazioni esistenti fra i vari processi determinano il traffico dovuto ai riferimenti a dati condivisi. Il rispetto della reale sequenza di serializzazione influisce sulla corretta individuazione dei pattern di accesso ai dati condivisi.

3.2 Le principali tecniche di tracciamento

Le tecniche di tracciamento possono essere raggruppate in cinque classi:

- *tecniche basate su hardware dedicato* [Flanagan92, Grimsrud92, Vashow93] che, oltre a richiedere la messa a punto di hardware dedicato, non godono di flessibilità e non consentono di correlare facilmente le tracce ai sorgenti.
- *tecniche basate sugli interrupt* [Eggers88, Wiecek82] che richiedono la presenza nel processore del cosiddetto *trap-flag*; anche se di costo minore questa tecnica soffre, come la precedente, di scarsa flessibilità. Inoltre, poiché il sistema operativo gira tipicamente ad interrupt disabilitati, non è possibile tracciare i riferimenti da esso generati.
- *tecniche basate sulla sintesi artificiale della traccia* [So88, Ricciardi95] che hanno il difetto di non potersi considerare un carico reale, anche se potenzialmente godono di alta flessibilità ed accuratezza.
- *tecniche ATUM basate sull'alterazione del microcodice* [Agarwal86] pure queste strettamente legate all'hardware; i processori commerciali tendono, poi, o ad essere integrati su un unico chip (con la ROM del microcodice integrata) o a non usare per niente microcodice.
- *tecniche basate sull'instrumentazione dei programmi sorgenti* [Stunkel89, Eggers90, Goldschmidt90] che hanno l'inconveniente di necessitare di un nucleo instrumentato del sistema operativo o della disponibilità dei sorgenti del nucleo stesso, ma godono di alta flessibilità e offrono la possibilità di tracciare una qualsiasi architettura, esistente o simulata (*macchina target*), senza essere vincolati dalla macchina usata per l'instrumentazione (*macchina host*). Alcune di queste tecniche modificano direttamente il codice binario delle applicazioni [Borg90], altre inseriscono modifiche a tempo di compilazione [Larus90]. Altre tecniche si basano sulla modifica del codice assembler [Goldschmidt90]; questa tecnica è stata riconosciuta come quella più completa [Stunkel 91] fra le varie tecniche presentate.

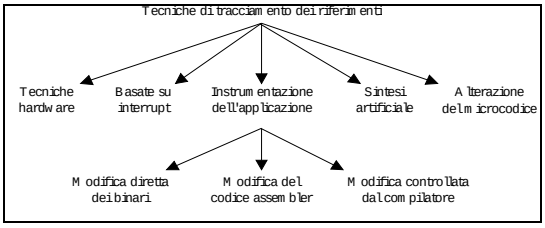


Figura 3–1. Tassonomia delle tecniche di tracciamento dei riferimenti.

Trace Factory può utilizzare questa tecnica col tool *TangoLite* [Herrod95].

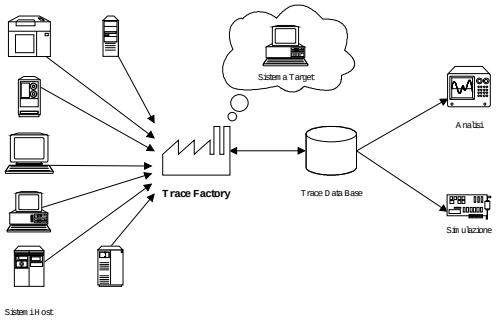


Figura 3–2. Ambiente operativo di Trace Factory

Trace Factory consente di gestire vari tipi di carico, sia sequenziale che multiprogrammato, contenenti o meno riferimenti del nucleo del sistema operativo; consente di inserire i riferimenti del nucleo stesso; consente di utilizzare vari algoritmi di scheduling definiti dall'utente, per stabilire quali processi gireranno sui processori disponibili; consente di rimappare gli spazi di indirizzamento di ciascun processo in aree private o condivise dello spazio di indirizzamento globale; consente di effettuare tutta una serie di analisi sulle tracce prodotte: dal semplice conteggio statistico degli eventi, all'analisi di densità di blocchi distinti, all'analisi delle metriche di condivisione (lista blocchi condivisi e write-run [Eggers89]).

Trace Factory sopperisce alle carenze dei vari metodi di produzione di tracce precedentemente elencati semplicemente cercando di mettere insieme tutti gli elementi base che ci si può procurare facilmente.

3.3.1 Il carico

In questo lavoro con il termine *carico* si intenderà l'insieme di una o più *tracce-base*; esso verrà utilizzato quale fonte di domanda del tempo macchina offerto dal sistema sotto studio (*sistema target*). Con il termine *traccia-base* si intenderà la traccia di una ben determinata applicazione, traccia che può essere stata prodotta sia con metodi hardware-based, che col metodo dell'instrumentazione del codice assembler.

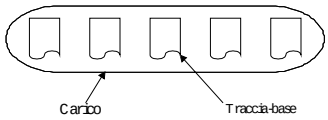


Figura 3–3. Relazione fra carico e tracce-base.

Poiché al Dipartimento di Ingegneria della Informazione di Pisa non ci sono attualmente sistemi del primo tipo, le tracce di questo genere sono state fornite da altre università, fra cui la Carnegie Mellon University di Pittsburgh e il Massachusetts Institute of Technology. Il tool di instrumentazione TangoLite è stato fornito dal Computer Systems Laboratory dell'Università di Stanford; grazie ad esso, si è avuta una notevole flessibilità nella produzione delle tracce-base, utilizzando una workstation DEC5000 del Dipartimento; la disponibilità dei sorgenti di TangoLite ha consentito inoltre di adattare il tool alle esigenze specifiche dell'ambiente operativo Trace Factory.

In particolare, le tracce-base prodotte con TangoLite possono essere generate a partire da una generica applicazione parallela o monoprocesso. Per le applicazioni parallele la definizione della creazione e comunicazione dei processi viene effettuata utilizzando un set di macro di parallelizzazione definito dall'Argonne National Laboratory che consentono la specifica della concorrenza in maniera totalmente indipendente dall'hardware [Lusk87] (per le applicazioni monoprocesso si può altresì procedere direttamente all'instrumentazione).

Nel caso di TangoLite però si presenta un'ulteriore "problema", che non si ha per le tracce direttamente fornite da altri, semplicemente per il fatto che tale questione è già stata risolta da chi ha prodotto la traccia-base: il problema della *scelta delle applicazioni*. La questione verrà esaminata in dettaglio in un paragrafo successivo.

3.3.2 Il sistema host

Con questo termine verrà inteso sia, in senso stretto, il sistema che ospita il software di instrumentazione (che nel caso di TangoLite è una macchina basata sul processore MIPS R3000), sia le varie macchine da cui, con metodi hardware-based sono state estratte tracce-base.

3.3.3 Il sistema target

È questo il sistema che si intende studiare attraverso l'analisi basata su tracce. Come già evidenziato, questo può essere sia coincidente con una macchina realmente esistente o con uno dei sistemi host, sia consistere in una macchina completamente astratta.
Con il termine *traccia-target* verrà indicata la traccia come verrebbe prodotta dal sistema target se la si estraesse direttamente, al pari di quanto si fa coi metodi hardware-based.

3.3.4 Trace Data Base

Come scelta progettuale, le tracce generate vengono memorizzate su disco allo scopo di poter essere sia analizzate da più strumenti, sia utilizzate come ingresso di vari simulatori, sia usate da uno stesso simulatore al variare dei parametri architetturali, senza bisogno di doverle rigenerare ogni volta. La collocazione in memoria stabile garantisce, inoltre, la assoluta ripetibilità degli esperimenti compiuti con una traccia o set di tracce. L'insieme delle tracce-base costituisce il Trace Data Base.

Una singola traccia-base può essere costituita da vari file: questo è tipicamente il caso di una traccia che rappresenta un carico multiprogrammato, in cui per ogni processore del sistema target esiste un file o *file-traccia* contente il flusso di esecuzione di quel processore; un ulteriore file specifica, poi, come vanno intercalati i vari file-traccia per rispettare la sincronizzazione esistente nell'applicazione.

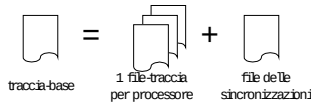


Figura 3–4. Relazione fra traccia-base e file di cui è composta.

Un problema non banale è quello del *formato di memorizzazione* delle tracce; una processore con una potenza di calcolo di 10 MIPS produce fino a 70 MB di traccia per secondo [Larus93]; ad un tale ritmo anche esecuzioni relativamente corte riempiono ben presto il disco. Alcuni hanno utilizzato metodi di compressione dedicati [Vashow93, Johnson94], non raggiungendo comunque gli elevati rapporti di compressione ottenibili con le ben note tecniche basate sull'algoritmo di Lempel-Ziv. La compressione della traccia, inoltre, è utile non solo per diminuire l'occupazione dello spazio su disco, ma anche per ridurre la quantità di dati trasferita da dispositivi relativamente lenti, quali il disco, alla memoria centrale. Trace Factory utilizza appunto la tecnica Lempel-Ziv[1].

3.3.5 Analisi della traccia

L'analisi è compiuta dal tool *ttf* (Trace to Trace Filter), che verrà descritto in dettaglio successivamente e che consente anche di convertire la traccia dal formato originario al formato `IIP` (v. appendice per una descrizione dei vari formati di traccia).

3.3.6 Simulazione

Le tracce prodotte da Trace Factory sono state validate anche tramite l'uso intensivo che ne è stato fatto come ingresso di un simulatore [Ricciardi95] ideato per la valutazione delle prestazioni di sistemi multiprocessore basati su cache. Le tracce generate possono essere usate anche per: studiare le strategie di paginazione della memoria; validare risultati derivanti da modelli analitici; valutare diverse soluzioni per la rete di interconnessione verso la memoria comune; valutare l'impatto del software reale sulle prestazioni di un sistema monoprocessore o multiprocessore.

[1] Invocando direttamente, tramite la procedura di libreria standard `popen`, che biforca un processo, il software *gzip,* basato su una variante di tale tecnica (LZ77), con la quale si raggiunge una compressione generalmente più alta di quella ottenibile col metodo LZW (*compress*) o con la codifica di Huffman normale (*pack*) o adattiva (*compact*).

3.4 La 'catena di montaggio' di Trace Factory

Trace Factory è costituito da una serie di strumenti software che sono volutamente stati ideati *semplici* e *modulari* per poter essere utilizzati in varie fasi della 'lavorazione' della traccia del sistema target.

Questo tipo di filosofia progettuale consente anche di intercambiare o riprogettare uno degli strumenti senza dover per questo effettuare delle modifiche agli altri. Inoltre, è possibile in futuro ideare nuovi strumenti in grado di operare nell'ambiente integrato, che effettuino nuove operazioni di 'messa a punto' della traccia-target, o integrare (come per TangoLite) strumenti di varia origine o provenienza. Poiché la sincronizzazione fra i vari strumenti avviene tramite file, è possibile, poi, combinare gli strumenti in molteplici sequenze. Il paradigma utilizzato è quello della catena di montaggio: le tracce-base sono sottoposte ad una serie di operazioni al fine di ottenere la traccia-target.

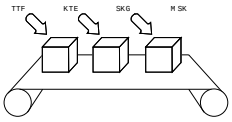


Figura 3–5. Trace Factory consente di operare sulle tracce usando il paradigma della catena di montaggio.

Gli strumenti che attualmente costituiscono Trace Factory sono *TangoLite* (software di instrumentazione), *ttf* (Trace to Trace Filter), *kte* (Kernel Trace Extractor), *skg* (Scheduling Generator) e *msk* (Multiprocessor Scheduler).

3.4.1 Sequenza delle operazioni della catena di montaggio.

Sono di seguito riportati i passi che generalmente si effettuano per ottenere 'da zero' la traccia-target.

1. Procurarsi i sorgenti della applicazione che si desidera tracciare.
2. Effettuare l'instrumentazione e la produzione della traccia-base con TangoLite (in alternativa a questi due primi passi ci si può direttamente procurare una traccia-base).
3. Convertire la traccia dal formato originario al formato utilizzato dai vari strumenti (i.e. al formato `IIP`) con il tool `ttf` (vengono in questa fase prodotte informazioni generali inerenti la traccia-base che possono essere utili per la scelta delle tracce-base da usare nei passi successivi).
4. Procurarsi una traccia contente riferimenti kernel ed estrarli con `kte`.
5. Generare la tabella di scheduling, che si desidera venga usata dalla macchina target, col tool `skg`.
6. Schedulare e rimappare i riferimenti contenuti nelle tracce-base usando il tool `msk` ottenendo così la traccia-target.

A questo punto si può analizzare la traccia base nuovamente con il tool `ttf` (se lo si desidera) e/o utilizzarla direttamente.

3.4.2 Presentazione dell'interfaccia grafica

L'ambiente operativo viene mandato in esecuzione con il comando `TF`; notare la facilità di selezione dei vari strumenti, disposti in una toolbar nella stessa sequenza logica in cui vengono utilizzati.

Figura 3–6. La toolbar di Trace Factory presenta gli strumenti nello stesso ordine in cui si usano.

3.5 La scelta delle applicazioni da tracciare

Le applicazioni scelte devono costituire un *set rappresentativo* dell'insieme totale dei programmi che si prevede di eseguire sul sistema di elaborazione sotto studio. Tracce sintetiche o 'programmi giocattolo' hanno solo una lontana rassomiglianza ,o rappresentano solo una parte, delle applicazioni che vengono fatte realmente girare sulla classe di sistemi che si sta considerando [Singh92].

'Rappresentativo', nell'accezione più generale significa che l'insieme deve comprendere *vari algoritmi* per risolvere *vari problemi* in *vari domini* e con *vari sistemi operativi e strumenti di sviluppo del software*. Nell'ambito di questo lavoro, 'rappresentativo' avrà un significato leggermente più ristretto, in quanto si spazierà solo in alcune delle classi appena elencate.

In ogni caso i set di applicazioni sono costituiti da applicazioni ben note, di cui sono disponibli i sorgenti (con i problemi di porting già risolti) per poter essere compilati sulla piattaforma di interesse.

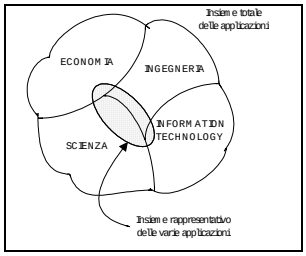


Figura 3–7. Rappresentatività del set di applicazioni scelto.

3.5.1 Il set di applicazioni SPLASH

Soprattutto nell'ambito della programmazione parallela esistono pochi programmi rappresentativi delle applicazioni reali. Per sopperire a questa carenza, l'Università di Stanford ha raccolto questo set di applicazioni [Singh92] che ha subito riscosso un notevole successo negli ambienti di ricerca. Il set comprende le applicazioni `MP3D`, `Ocean`, `Water`, `Barnes-Hut`, `LocusRoute`, `PTHOR`, `Cholesky`[1]. Recentemente questo set è stato esteso con nuove applicazioni (SPLASH2): `FFM`, `Radiosity`, `Raytrace`, `Volrend`, `FFT`, `LU`, `RADIX`.

Considerazioni di carattere generale sulle applicazioni parallele

La tecnologia software per scrivere programmi paralleli è immatura. Non è chiaro se i programmi scritti con gli odierni costrutti saranno ancora rappresentativi di una certa categoria di applicazioni, in futuro. Gli studi hanno una validità limitata a ciò che oggi è disponibile.

I programmi possono non essere parallelizzati nella miglior maniera per il problema che devono risolvere; il parallelismo su larga scala può, inoltre, richiedere algoritmi molto differenti da quelli usati su macchine piccole.

Le relazioni fra le architetture e le applicazioni si sviluppano in una nuova dimensione con il parallelismo. Ad esempio, gestire la località dei dati sta diventando sempre più complesso, così come le interazioni fra la dimensione delle applicazioni e la dimensione della macchina multiprocessore. Il numero di parametri architetturali è molto elevato ed è quindi necessario riguardare con cura le varie ipotesi, essendo difficile valutare l'impatto di una certa modifica.

Difetti delle applicazioni di SPLASH

Il parallelismo di queste applicazioni è principalmente limitato dalla natura e dalla dimensione dei dati di ingresso. Le applicazioni sono, inoltre, scritte per macchine di piccola o media dimensione e può essere necessario ristrutturarle per farle girare su multiprocessori di dimensione più grossa.

È quindi necessario *prestare attenzione nell'usare propriamente queste applicazioni* per valutare le prestazioni di una determinata categoria di macchine o per dimostrare che un sistema è migliore di un altro.

Pregi delle applicazioni di SPLASH

Nonostante i precedenti difetti le applicazioni *sono* rappresentative e reali, *scritte in maniera totalmente indipendente dall'architettura*, utilizzando lo *stesso modello di programmazione*.

Sono state usate ampiamente nella comunità scientifica e ciò consente di confrontare realmente i risultati di un gruppo con quelli di un altro.

3.5.2 Il set di applicazioni CMU

Il set è stato messo insieme da B. Vashow [Vashow93], come lavoro svolto nella sua tesi di dottorato presso la Carnegie Mellon University di Pittsburgh. Il set comprende 10 applicazioni: `MP3D`†, `QuickSort`, `LocusRoute`†, `Notoy`, `ECAS`, `FFT`, `PDE`, `HartStone`, `Pile_Tracer`, `MS_Tracer`. Quasi tutte le applicazioni sono state scritte nella locale università.

Difetti del set CMU

Valgono le stesse considerazioni di carattere generale fatte per il set di applicazioni parallele SPLASH. Di questo set sono disponibili soltanto le tracce: è qui che entra in gioco Trace Factory, estraendo e ricombinando opportunamente le tracce-base è possibile studiare il comportamento della macchina utilizzando il solito tipo di carico e al variare dei parametri architetturali.

---

[1] Le applicazioni sono disponibili via rete all'indirizzo *ftp://mojave.stanford.edu/splash*.

† Anche queste prelevate, in effetti, dal set di SPLASH.

Pregi del set CMU
È costituito da uno dei set di tracce più completi ed accurati che siano stati messi a disposizione della comunità scientifica. Il set di applicazioni è adeguatamente rappresentativo, le tracce contengono sia i riferimenti generati dai programmi utente che dal kernel e contengono le dovute sincronizzazioni per rielaborare gli accessi conservando l'ordinamento originario; le tracce sono complete, in quanto prelevate direttamente con metodi hardware; il sistema non è perturbato dal sistema di estrazione.

Dati i vari pregi queste tracce sono state analizzate, più in dettaglio di altre, in un successivo capitolo ed utilizzate anche localmente [Ricciardi95].

3.5.3 Il set di applicazioni MIT
Il set comprende delle applicazioni che sono state usate per valutare le prestazioni dei protocolli di coerenza per cache basate sullo schema a directory [Chaiken90]. Parte di queste applicazioni sono state generate utilizzando un metodo di scheduling postmortem e sono scritte in Fortran: `Wthr64`, `Simp64`, `FFT64`; un'altra è stata ricavata con un metodo basato sull'instrumentazione ed è scritta in MultiLisp: `Speech64`.

Wthr, o Weather, partiziona l'atmosfera in una griglia tridimensionale e usa il metodo degli elementi finiti per risolvere un sistema di equazioni differenziali alle derivate parziali che descrivono lo stato del sistema.

Simp, o Simple, modella il comportamento dei fluidi e usa il metodo delle differenze finite per risolvere le equazioni del modello idrodinamico.

FFT è la trasformata di Fourier veloce in base 2.

Speech effettua la decodifica lessicale per un sistema di compressione fonetica del linguaggio sviluppato dal MIT. Essa usa un dizionario di circa 300 parole, rappresentate da un grafo direzionale con 3500 nodi. L'ingresso del decoder lessicale è un altro grafo direzionale che rappresenta le possibili sequenze dei fonemi nel discorso.

Difetti del set MIT
Come per il set CMU, sono disponibili le sole tracce; entra, quindi, in gioco Trace Factory per utilizzare queste ultime come carico della macchina target. Le tracce sono, inoltre, piuttosto 'corte' relativamente alle altre disponibili (non si supera il mezzo milione di riferimenti per processore) e non contengono i riferimenti del nucleo. Per questi motivi dopo una prima analisi le tracce sono state 'accantonate'.

Pregi del set MIT
Sono tracce relative a multiprocessori su larga scala (64 processori), quindi, il numero *totale* di riferimenti è tale da potersi confrontare con quello delle altre tracce disponibili. Tali tracce sono abbastanza citate in letteratura.

3.5.4 Il set di applicazioni GNU
Buona parte delle applicazioni che fanno parte di questo set sono relative a comandi di utilità in ambiente UNIX. Il software GNU è di completo pubblico dominio: il 'marchio' serve per garantire che nessuno possa appropriarsi di questo software per poterlo rivendere e lasciare così la libertà ad ognuno di utilizzarlo, modificarlo e condividerlo nuovamente.
Sono tutte applicazioni monoprocesso, rappresentative dei programmi di utilità in ambiente UNIX; alcune tracce riguardano anche comandi che operano in ambiente distribuito (e.g. Telnet). Il set comprende: `awk`, `cp`, `dd`, `du`, `gzip`, `lex`, `ls`, `rm`, `telnet`.

Pregi e difetti del set GNU
Nel presente lavoro di tesi è stata scelta questa categoria di programmi, non solo per l'ampia disponibilità in rete dei sorgenti, ma anche per la notevole cura con cui il software stesso è stato scritto e per la portabilità del codice, problema non sempre scontato in ambiente UNIX.

3.5.5 Il set di applicazioni SPEC92

Un altro set, diventato anche standard commerciale, è SPEC92. Questo è costituito da programmi non banali e reali, con lo scopo di standardizzare l'operazione di benchmarking. Il set comprende varie applicazioni aventi una accettabile portabilità e ampiamente supportate dall'industria della tecnologia dell'informazione; 6 applicazioni sono in C e riguardano soprattutto gli interi: `compress`, `eqntott`, `espresso`, `gcc`, `sc`, `xlisp`; altre 14 sono scritte anche in Fortran e riguardano soprattutto i floating-point `alvinn`, `doduc`, `ear`, `fpppp`, `hydro2s`, `mdljdp2`, `mdljsp2`, `nasa7`, `ora`, `spice`, `su2cor`, `swm256`, `tomcatv`, `wave5`. Il primo gruppo valuta il numero chiamato SPECINT92; il secondo gruppo valuta il numero chiamato SPECFP92.

Pregi e Difetti del set SPEC92

Non avendo disponibilità di questo set, esso viene qua semplicemente citato.

3.5.6 Riepilogo delle applicazioni tracciate

In appendice sono riportate le schede delle applicazioni tracciate, che sono state quasi tutte prelevate dai precedenti set, tranne qualcuna proveniente da altra fonte ma sempre di pubblico dominio. Le applicazioni sono state raggruppate a seconda del metodo utilizzato per ricavare le relative tracce.

Nome traccia	#CPU	Ling	Descrizione
CJPEG	1	C	Compressione di immagini
DHRY	1	C	Benchmark

Tabella 3–1. Tracce analizzate in formato APT.

Nome traccia	#CPU	Ling	Descrizione
ECAS	8	C	Simulatore di architettura di computer
HART	8	Ada	Hard-Real-Time Benchmark
LOCUS	8	C	VLSI standard cell router
MP3D	8	C	Simulazione di fluidi rarefatti
MS_TRACER	8	C	Ray tracing
PDE	8	C	Risolutore di equaz. differenziali d.parziali

Tabella 3–2. Tracce analizzate in formato CMU.

Nome traccia	#CPU	Ling	Descrizione
AWK	1	C	Elaborazione di file di testo
BARNES	1,8,24	C	Simulazione di evoluzione di galassie
CHOLESKY	1,8,24	C	Fattorizzazione di una matrice sparsa
CJPEG	1	C	Compressione di immagini
CP	1	C	Copia da un file a un altro
CSIM16	1	C	Simulatore di cache per multiprocessore
DDRMT0	1,	C	Conversione di file durante la copia
DJPEG	1	C	Conversione di immagine (piccola)
DJPEGA	1	C	Conversione di immagine (grande)
DU	1	C	Utilizzazione del disco
GZIPA	1	C	Compressione di file con algoritmo LZ77
LEX	1	C	Generatore di scanner lessicale
LS-R	1	C	Elenco ricorsivo del contenuto di directory
LS-AR	1	C	Elenco ricorsivo e completo del cont di dir.
LS-LTR	1	C	Elenco ricorsivo e ordinato del cont. di dir.
MP3D	1,8,24	C	Simulazione del flusso di fluidi rarefatti
RM	1	C	Rimozione di file
TELNET	1	C	Collegamento ad un host remoto

Tabella 3–3. Tracce analizzate e prodotte localmente in formato SCL.

3.6 Generazione di traccia-target a partire dalle tracce-base

Una volta scelte le applicazioni da tracciare, il passo successivo consiste nel produrre le tracce-base ed, eventualmente, combinare queste ultime per ottenere la traccia-target,

La scelta delle applicazioni è di fondamentale importanza per ottenere la traccia rappresentativa di un determinato carico. Ad esempio, per simulare un carico di una workstation UNIX, sono state scelte una serie di applicazioni monoprocesso tipicamente utilizzate durante una sessione UNIX e combinate opportunamente sfruttando come traccia del kernel quella presente in una delle applicazioni CMU; per effettuare la scelta sono state utilizzate le schede delle applicazioni riportate in appendice.

Nel capitolo successivo, verranno descritte le funzionalità dei vari strumenti di Trace Factory e verrà, infine, mostrato come ottenere la traccia del carico di una workstation UNIX.

4 Gli Strumenti di Trace Factory

L'ambiente operativo di Trace Factory è composto da vari strumenti che intervengono sulle tracce e da un interfaccia grafica che permette di integrare le funzionalità degli strumenti stessi, selezionarli e configurarli.
In questo capitolo verranno descritte in dettaglio le possibilità offerte da ogni strumento rimandando alle appendici i dettagli implementativi e il manuale d'uso del software.

4.1 Il tool SOURCE

Se le tracce-base non sono già fornite da qualche fonte, e devono essere prodotte, questo tool consente di compilare i sorgenti dell'applicazione prescelta e lanciarne l'esecuzione in modo da verificare il corretto funzionamento *prima* dell'instrumentazione.

In questa fase vengono preparati i makefile di base, selezionate le opzioni dell'esecuzione, in modo tale che nella fase successiva - l'instrumentazione con TangoLite - siano già stati risolti tutti gli eventuali problemi di porting e di scelta del compilatore.

L'esperienza dimostra che la maggior parte dei problemi nella instrumentazione delle applicazioni derivano da una scelta inadeguata del codice dell'applicazione che si intende tracciare. Anche per questo nel precedente capitolo è stato dato ampio spazio al problema della scelta delle applicazioni, selezionando quelle applicazioni di cui si disponga di sorgenti ampiamente collaudati e portabili.

4.2 Il tool TangoLite

TangoLite è un ambiente software, per la simulazione di sistemi multiprocessore, messo a punto dal Computer System Laboratory dell'Università di Stanford, utilizzabile nello studio e nella progettazione sia dell'hardware che del software per queste macchine [Herrod95, Goldschmidt93]. Questo software è stato adattato alle esigenze specifiche di tracciamento relative alla presente tesi.

Il *sistema host* per TangoLite è una macchina basata sul processore MIPS R3000 (o compatibile), il *sistema target* può essere definito all'interno del simulatore/tracciatore incorporato, utilizzando il linguaggio C.

La caratteristica di maggior rilievo per il presente lavoro è la possibilità di produrre la traccia di una applicazione monoprocesso o multiprocesso, partendo dai sorgenti dell'applicazione stessa. Il tracciamento avviene in maniera automatica utilizzando un apposito tool (`aug`) che inserisce all'interno del codice assembler, corrispondente all'applicazione in esame, delle *sonde software*, le quali estraggono le informazioni di interesse; questa operazione è chiamata *instrumentazione* dell'applicazione.

Mentre un'applicazione monoprocesso può essere instrumentata così come sta, un'applicazione multiprogrammata richiede un passo di preprocessing per tradurre le primitive ANL [Lusk87] di creazione, sincronizzazione e comunicazione fra i processi in normale linguaggio C.

4.2.1 Peculiarità di TangoLite

- Diversamente dai simulatori che agiscono *interpretando* istruzione per istruzione, TangoLite aggiunge codice di instrumentazione solo nei punti desiderati; si può ottenere la traccia dei soli punti di sincronizzazione, della chiamata/ritorno a procedura, fino a scendere ai cosiddetti *basic-block* le unità di codice che vengono eseguite in sequenza, ovvero non contenenti più di una istruzione di diramazione; i basic-block possono essere ulteriormente suddivisi in *blocchi,* non contenenti più di una istruzione di tipo *load* o *store* (vedi esempio al paragrafo successivo).
- L'instrumentazione è effettuata a livello di codice assembler, consentendo così di instrumentare carichi scritti in vari linguaggi di programmazione e compilati con vari compilatori.
- I processi del carico sono rappresentati come *processi leggeri* (light-weight processes) che eseguono in uno spazio di indirizzamento comune. Ogni processo ha associato un clock che indica a che punto si trova l'esecuzione del relativo processo; gli eventi generati da processi diversi sono così eseguiti in stretto ordine cronologico. In effetti, esiste una dilatazione dei tempi di esecuzione, ma questa risulta uniforme, dando così una risultante nulla per quanto riguarda l'accuratezza della simulazione.
- TangoLite non è un simulatore di una particolare macchina; in particolare non deve trarre in inganno il fatto che gira su macchine MIPS. TangoLite consente di estrarre l'impronta del programma applicativo a prescindere da eventuali riorganizzazioni del codice (che devono essere disabilitate) che possono potenzialmente essere effettuate da un compilatore per macchine RISC.

Chi e perché ha già usato TangoLite

TangoLite è stato usato per investigare l'efficacia di alcuni modelli nel mascherare le latenze della memoria e nel gestire lo scheduling dinamico di processori [Gharachorl92], per valutare processori a contesto multiplo sia per applicazioni parallele che multiprogrammate [Laudon92], per studiare il comportamento delle cache e i pattern di invalidazione nei multiprocessori a memoria condivisa e per comparare le prestazioni di differenti protocolli di coerenza basati sullo schema a directory [Gupta92a], per valutare le architetture COMA non gerarchiche [Gupta92b], per investigare modi di migliorare la località dei dati e il bilanciamento del carico nei programmi paralleli [Chandra94], per implementare MemSpy, un tool di monitoring che consente di interagire con il codice sorgente per migliorare le prestazioni delle cache [Martonosi95].

4.2.2 Procedura base di TangoLite

La procedura base di TangoLite è illustrata nella seguente figura:

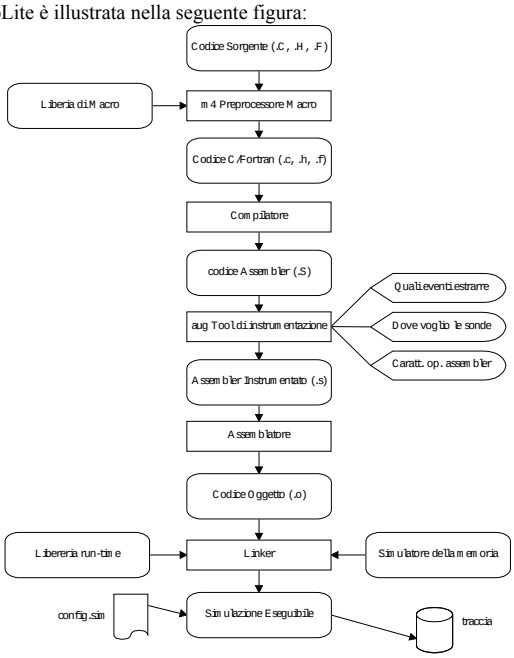


Figura 4–1. Procedura base di TangoLite.

Ci sono cinque passi: espansione delle macro, compilazione, instrumentazione, assemblatura, collegamento.

- Tipicamente il codice iniziale è stato pensato per una macchina parallela ed è stato parallelizzato a mano usando il set di macro ANL [Lusk87]; l'espansione delle macro fornisce codice C o Fortran 'puro' usando il preprocessore `m4` (standard in quasi tutti i sistemi UNIX); è possibile usare varie librerie di macro: per questa tesi è stata usata la libreria `c.m4.tds` che implementa la sincronizzazione ad alto livello per il C.
- Il secondo passo è la compilazione usando un qualsiasi compilatore C o Fortran che produca del codice assembler MIPS.
- Il terzo passo è la instrumentazione attraverso il tool `aug` al quale vanno specificati gli eventi che si desidera simulare e il file di inizializzazione. I tipi di eventi sono:
  - *eventi di sincronizzazione* (generati dalle macro ANL);
  - *eventi basic-block* (generati dai blocchi base dell'applicazione);
  - *eventi frame_reference* (riferimenti ai dati all'interno dello stack);
  - *eventi normal_reference* (tutti gli altri riferimenti che non coinvolgono lo stack).

  Il file di inizializzazione descrive ogni codice mnemonico assembler classificandolo per tipo di operazione (load, store, branch, ...), numero di cicli di default e quali registri macchina usa o modifica.
- Il quarto passo è la compilazione utilizzando un normale assemblatore MIPS. I primi 4 passi vengono ripetuti per ognuno dei file che compongono il programma applicativo.

- Il quinto passo è infine il collegamento di tutti i file che compongono l'applicazione, dei file di libreria e del simulatore.

Una volta preparata la simulazione eseguibile, con il file `config.sim` si può configurare a piacere il comportamento del simulatore per raccogliere nella traccia gli eventi desiderati, per ottenere statistiche o comportamenti particolari o per specificare il numero di processi/processori[1].

4.2.3 Instrumentazione e generazione della traccia base

Più in dettaglio il tool di instrumentazione effettua un parsing del codice assembler MIPS, inserendo le cosiddette *sonde software* ovvero delle porzioni di codice assembler che provvedono a:

- memorizzare l'indirizzo dell'evento,
- memorizzare il tipo di evento e altre informazioni,
- chiamare la `sys_call_routine`.

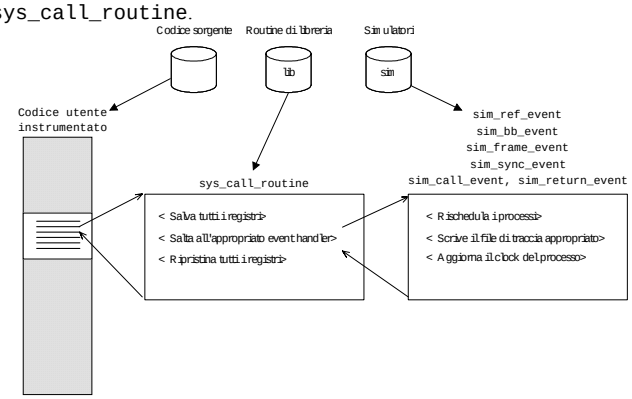


Figura 4–2. Dettaglio dell'operazione di chiamata alla routine di tracciamento.

La `sys_call_routine` non fa altro che chiamare l'appropriato event-handler, avendo cura che nessun registro macchina venga sporcato.
L'event-handler provvede a:

- rischedulare i processi[2] prelevando quello con il clock più basso;
- scrivere nel file di traccia tutte le informazioni che si desiderano (v. in appendice il formato di traccia `SCL`, per ulteriori dettagli);
- aggiornare il clock del processo.

4.2.4 Indirizzi delle istruzioni

Mentre per quanto riguarda gli indirizzi dei dati la precedente procedura funziona correttamente, per gli indirizzi delle istruzioni verrebbero introdotte grosse distorsioni. Infatti, le sonde fanno variare la posizione originaria delle istruzioni del codice applicativo.

TangoLite consente di effettuare il corretto tracciamento degli indirizzi delle istruzioni utilizzando una procedura a doppia passata, leggermente più complessa della procedura base precedentemente illustrata.

Durante la prima passata, il codice assembler viene anche etichettato, in modo tale da far comparire nella *symbol table* gli indirizzi di ogni basic-block. Nella seconda passata si usa il lavoro effettuato nella prima passata per inserire il codice della sonda (che memorizzerà l'indirizzo delle istruzioni).

[1] Ad ogni processore TangoLite associa un solo processo o thread.

[2] Questa operazione consente di conservare l'ordinamento degli eventi: un processo si blocca se il suo clock è andato oltre quello di ogni altro processo.

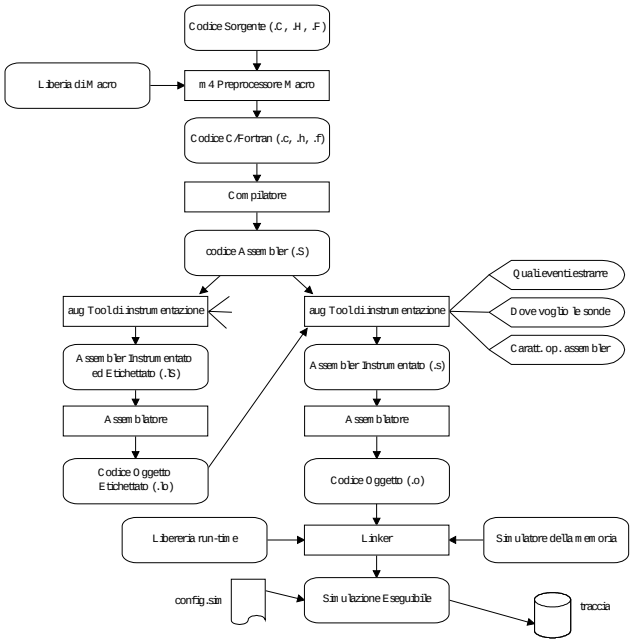


Figura 4–3. Procedura per evitare la distorsione degli indirizzi delle istruzioni.

4.2.5 Limiti di TangoLite

Mentre risulta relativamente semplice prendere del software applicativo e tracciarne l'esecuzione, non è affatto banale fare altrettanto col sistema operativo, operazione che richiederebbe l'instrumentazione dell'intero nucleo. Questa mancanza di flessibilità ha orientato ad effettuare un'opera di modularizzazione, utilizzando TangoLite *per il solo tracciamento* (senza considerare la possibilità di effettuare simulazioni con quanto offre lo strumento); d'altra parte le simulazioni di interesse possono essere effettuate con i simulatori disponibili usando, come punto di sincronizzazione col software di instrumentazione, i file di traccia; questo metodo di sincronizzazione è ampiamente utilizzato in Trace Factory.
Un altro limite di TangoLite riguarda la completezza di copertura: esso non può instrumentare quelle porzioni della libreria "C" che sono chiamate dallo stesso supporto a run-time. Le chiamate a funzioni non instrumentate sono trattate come operazioni atomiche. Le parti di codice affette da questo problema includono le procedure di output (`printf`), la allocazione della memoria (`malloc`) e l'I/O di file (`fopen`). Per applicazioni CPU-intensive il codice non instrumentato ha però scarso impatto sulla accuratezza della traccia e d'altra parte l'intera libreria matematica (`libm`) e la libreria run-time del C (`libc`) sono state instrumentate riducendo così la portata del problema.

Barnes	Water	Ocean	MP3D	Cholesky	PThor	Locus
99.84%	99.66%	99.51%	98.44%	95.01%	90.94%	85.53%

Tabella 4–1. Completezza di copertura per alcune applicazioni del set SPLASH [Herrod95].

Spesso le routine precedentemente considerate compaiono solo in fase di inizializzazione del programma; è quindi possibile esaminare la traccia solo nella zona ove la completezza di copertura è maggiore.

Un inconveniente, di minore impatto, è il fatto che il simulatore/tracciatore e l'applicazione condividono un unico spazio di memoria e quindi gli indirizzi dei dati dell'applicazione sono intercalati dai dati del simulatore e dall'espansione del codice causato dall'instrumentazione.

4.3 Esempio di produzione di traccia-base utilizzando TangoLite

Viene mostrato in dettaglio come agisce TangoLite per produrre tracce. Vale la pena di ricordare che TangoLite non è stato ideato solo per generare tracce ma per creare una intera *running-simulation*; Trace Factory ne utilizza invece solo la parte di generazione di tracce.

4.3.1 Dal sorgente in linguaggio C al codice assembler

Il seguente frammento di codice sorgente (sulla sinistra) è l'inizio dell'applicazione monoprocesso `ls` (v. appendice per la relativa scheda descrittiva); il primo passo consiste nel tradurlo in assembler (come mostrato sulla destra) usando l'opzione `-S` del compilatore C (`cc`).

FILE ls.c

```
# 527 void main (int argc, char **argv)
# 528 {
# 529   register int i;
# 530   register struct pending *thispend;
# 531
# 532   exit_status = 0;
# 533   dir_defaulted = 1;
# 534   print_dir_name = 1;
# 535   pending_dirs = 0;
# 536   current_time = time ((time_t *) 0);
# 537
# 538   program_name = argv[0];
# 539   i = decode_switches (argc, argv);
# 540
# 541   if (show_version)
# 542     {
# 543       printf ("%s\n", version_string);
# 544       exit (0);
# 545     }
# 546
# 547   if (show_help)
# 548     usage (0);
# 549
# 550   format_needs_stat = sort_type == sort_time
# 551     || sort_type == sort_size
# 552     || format == long_format
# 553     || trace_links || trace_dirs
# 554     || indicator_style != none
# 555     || print_block_size || print_inode;
```

Legenda:

- preceduti da '#': sono i numeri di linea nel file del codice sorgente
- precedute da '$': sono le etichette di salto del codice assembler
- il codice assembler si riferisce al processore MIPS R3000

FILE ls.S

```
 # 527   void main (int argc, char **argv)
 # 528   {
       .ent      main 2
main:
       .option   O2
       subu      $sp, 80
       sw        $31, 44($sp)
       sw        $16, 40($sp)
       .mask     0x80010000, -36
       .frame    $sp, 80, $31
       sw        $4, 80($sp)
       sw        $5, 84($sp)
       .loc      2 532
 # 529     register int i;
 # 530     register struct pending *thispend;
 # 531
 # 532     exit_status = 0;
       sw        $0, $$295
       .loc      2 533
 # 533     dir_defaulted = 1;
       li        $14, 1
       sw        $14, $$291
       .loc      2 534
 # 534     print_dir_name = 1;
       li        $15, 1
       sw        $15, $$292
       .loc      2 535
 # 535     pending_dirs = 0;
       move      $2, $0
       .loc      2 536
 # 536     current_time = time ((time_t *) 0);
       move      $4, $0
       sw        $2, $$266
       .livereg  0x800ff00e,0xfff
       jal       time
       lw        $5, 84($sp)
       sw        $2, $$267
       .loc      2 538
 # 537
 # 538     program_name = argv[0];
       lw        $24, 0($5)
       sw        $24, program_name
       .loc      2 539
 # 539     i = decode_switches (argc, argv);
       lw        $4, 80($sp)
       .livereg  0x800ff00e,0xfff
       jal       $$239
       sw        $2, 76($sp)
       .loc      2 541
 # 540
 # 541     if (show_version)
       lw        $25, $$297
       beq       $25, 0, $44
       .loc      2 543
 # 542       {
 # 543         printf ("%s\n", version_string);
       la        $4, $$357
       lw        $5, version_string
       .livereg  0xc00ff00e,0xfff
       jal       printf
       .loc      2 544
 # 544         exit (0);
       move      $4, $0
       .livereg  0x800ff00e,0xfff
       jal       exit
       .loc      2 545
 # 545       }
$44:
       .loc      2 547
 # 546
 # 547     if (show_help)
       lw        $8, $$296
       beq       $8, 0, $45
       .loc      2 548
 # 548       usage (0);
       move      $4, $0
       .livereg  0x800ff00e,0xfff
       jal       $$261
$45:
       lw        $4, $$272
       .loc      2 553
 # 549
 # 550   format_needs_stat= sort_type == sort_time
 # 551       || sort_type == sort_size
 # 552       || format == long_format
 # 553       || trace_links || trace_dirs
 # 554       || indicator_style != none
 # 555       || print_block_size || print_inode;
       seq       $2, $4, 3
       move      $3, $2
       bne       $2, $0, $46
       seq       $2, $4, 4
       move      $3, $2
       bne       $2, $0, $46
       lw        $2, $$269
       seq       $2, $2, 0
       move      $3, $2
       bne       $2, $0, $46
       lw        $2, $$281
       sne       $2, $2, 0
       move      $3, $2
       bne       $2, $0, $46
       lw        $2, $$282
       sne       $2, $2, 0
       move      $3, $2
       bne       $2, $0, $46
       lw        $2, $$279
       sne       $2, $2, 0
       move      $3, $2
       bne       $2, $0, $46
       lw        $2, $$276
       sne       $2, $2, 0
       move      $3, $2
       bne       $2, $0, $46
       lw        $3, $$280
       sne       $3, $3, 0
$46:
       sw        $3, $$294
       .loc      2 555
```

4.3.2 Dal codice assembler al codice assembler etichettato

A questo punto interviene la procedura di instrumentazione (tool `aug`), che suddivide il codice assembler in *blocchi,* contenenti al più una istruzione con operandi in memoria dati (`load` e `store`). In corrispondenza di ogni blocco vengono poi inserite delle etichette che, pur non alterando assolutamente il codice originario, genereranno nuove entry nella symbol table, corrispondenti agli indirizzi dei blocchi in memoria istruzioni; questo metodo consente di non introdurre distorsioni sui riferimenti di tipo fetch all'interno della traccia che verrà prodotta.

`FILE ls.lS`

```
          .ent      main 2
main:
sys_b65:
          subu      $sp , 80
sys_e65:
sys_b66:
          sw        $31 , 44($sp)
sys_e66:
sys_b67:
          sw        $16 , 40($sp)
sys_e67:
          .mask     0x80010000 , -36
          .frame    $sp , 80 , $31
sys_b68:
          sw        $4 , 80($sp)
sys_e68:
sys_b69:
          sw        $5 , 84($sp)
sys_e69:
          .loc      2 532
sys_b70:
          sw        $0 , $$295
          .loc      2 533
          li        $14 , 1
sys_e70:
sys_b71:
          sw        $14 , $$291
          .loc      2 534
          li        $15 , 1
sys_e71:
sys_b72:
          sw        $15 , $$292
          .loc      2 535
          move      $2 , $0
          .loc      2 536
          move      $4 , $0
sys_e72:
sys_b73:
          sw        $2 , $$266
          .livereg  0x800ff0e , 0xfff
          jal       time
sys_e73:
sys_b74:
          lw        $5 , 84($sp)
sys_e74:
sys_b75:
          sw        $2 , $$267
sys_e75:
          .loc      2 538
sys_b76:
          lw        $24 , 0($5)
sys_e76:
sys_b77:
          sw        $24 , program_name
sys_e77:
          .loc      2 539
sys_b78:
          lw        $4 , 80($sp)
          .livereg  0x800ff0e , 0xfff
          jal       $$239
sys_e78:
sys_b79:
          sw        $2 , 76($sp)
sys_e79:
          .loc      2 541
sys_b80:
          lw        $25 , $$297
          beq       $25 , 0 , $44
sys_e80:
          .loc      2 543
sys_b81:
          la        $4 , $$357
sys_e81:
sys_b82:
          lw        $5 , version_string
          .livereg  0xc00ff0e , 0xfff
          jal       printf
sys_e82:
          .loc      2 544
sys_b83:
          move      $4 , $0
          .livereg  0x800ff0e , 0xfff
          jal       exit
sys_e83:
          .loc      2 545
sys_b84:
sys_e84:
$44:
          .loc      2 547
sys_b85:
sys_e85:
sys_b86:
          lw        $8 , $$296
          beq       $8 , 0 , $45
sys_e86:
          .loc      2 548
sys_b87:
          move      $4 , $0
          .livereg  0x800ff0e , 0xfff
          jal       $$261
sys_e87:  $45:
sys_b88:
sys_e88:
sys_b89:
          lw        $4 , $$272
          .loc      2 553
          seq       $2 , $4 , 3
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e89:
sys_b90:
          seq       $2 , $4 , 4
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e90:
sys_b91:
          lw        $2 , $$269
          seq       $2 , $2 , 0
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e91:
sys_b92:
          lw        $2 , $$281
          sne       $2 , $2 , 0
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e92:
sys_b93:
          lw        $2 , $$282
          sne       $2 , $2 , 0
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e93:
sys_b94:
          lw        $2 , $$279
          sne       $2 , $2 , 0
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e94:
sys_b95:
          lw        $2 , $$276
          sne       $2 , $2 , 0
          move      $3 , $2
          bne       $2 , $0 , $46
sys_e95:
sys_b96:
          lw        $3 , $$280
          sne       $3 , $3 , 0
sys_e96:
$46:
sys_b97:
sys_e97:
sys_b98:
          sw        $3 , $$294
sys_e98:
          .loc      2 555
```

Legenda:

- '`sys_bXX`': sono le etichette che marcano l'inizio del blocco XX
- '`sys_eXX`': sono le etichette che marcano la fine del blocco XX

4.3.3 Dal codice assembler etichettato al codice oggetto etichettato

L'assemblatore (`as`) effettua la traduzione, da codice assembler-etichettato (`.lS`) a codice oggetto-etichettato (`.lo`); di quest'ultimo file verranno utilizzate, nella successiva passata attraverso il tool di instrumentazione (`aug`), soltanto le informazioni contenute nella symbol table, per stabilire l'indirizzo effettivo di ciascun blocco.

4.3.4 Dai codici assembler e oggetto etichettati al codice assembler instrumentato

Viene poi richiamato il tool `aug` (in questa seconda passata genera il file `.s`), che provvede a raggruppare i blocchi in *basic-block*: infatti non è necessario conoscere l'indirizzo di ogni blocco ma solo del gruppo di blocchi che contengono al più un'istruzione che modifica il flusso del programma (tipicamente un salto); tale insieme di blocchi è chiamato appunto basic-block. Inoltre `aug` inserisce in ogni blocco delle istruzioni che provvedono a memorizzare in un'opportuna struttura dati (che verrà utilizzata in fase di tracciamento) informazioni quali:

- l'indirizzo di ogni *blocco* e il numero di riferimenti che contiene;
- i riferimenti di tipo dati (letti dinamicamente);
- il numero di linea del codice sorgente C che ha generato quel blocco.

`FILE ls.s`

```
        .ent      main 2
main:
 # BLOCK 65: basic = 1
          sw       $31, -88($sp)
          jal      sys_init_routine
          sw       $15, -100($sp)
          lw       $15, sys_current_thread
          la       $31, sim_bb_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x244
          sw       $31, 140($15)
          li       $31, 528
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         subu      $sp , 80
 # BLOCK 66: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 44($sp)
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x248
          sw       $31, 140($15)
          li       $31, 528
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $31 , 44($sp)
 # BLOCK 67: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 40($sp)
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x24c
          sw       $31, 140($15)
          li       $31, 528
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $16 , 40($sp)
         .mask     0x80010000 , -36
         .frame    $sp , 80 , $31
 # BLOCK 68: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 80($sp)
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x250
          sw       $31, 140($15)
          li       $31, 528
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $4 , 80($sp)
 # BLOCK 69: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 84($sp)
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x254
          sw       $31, 140($15)
          li       $31, 528
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $5 , 84($sp)
         .loc      2 532
 # BLOCK 70: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, $$295
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x258
          sw       $31, 140($15)
          li       $31, 532
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 2
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $0 , $$295
         .loc      2 533
          li       $14 , 1
 # BLOCK 71: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, $$291
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x260
          sw       $31, 140($15)
          li       $31, 533
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 2
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $14 , $$291
         .loc      2 534
          li       $15 , 1
 # BLOCK 72: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, $$292
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x268
          sw       $31, 140($15)
          li       $31, 534
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 3
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $15 , $$292
         .loc      2 535
         move       $2 , $0
         .loc      2 536
         move       $4 , $0
 # BLOCK 73: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, $$266
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x274
          sw       $31, 140($15)
          li       $31, 536
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 2
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $2 , $$266
         .livereg  0x800ff0e , 0xfff
          jal      time
 # BLOCK 74: basic = 1
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 84($sp)
          sw       $31, 120($15)
          la       $31, sim_2_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x27c
          sw       $31, 140($15)
          li       $31, 536
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 2
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         lw        $5 , 84($sp)
 # BLOCK 75: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, $$267
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x284
          sw       $31, 140($15)
          li       $31, 536
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $2 , $$267
         .loc      2 538
 # BLOCK 76: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 0($5)
          sw       $31, 120($15)
          la       $31, sim_2_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x288
          sw       $31, 140($15)
          li       $31, 538
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 2
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         lw        $24 , 0($5)
 # BLOCK 77: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, program_name
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x290
          sw       $31, 140($15)
          li       $31, 538
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $24 , program_name
         .loc      2 539
 # BLOCK 78: basic = 0
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 80($sp)
          sw       $31, 120($15)
          la       $31, sim_2_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x294
          sw       $31, 140($15)
          li       $31, 539
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 3
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         lw        $4 , 80($sp)
         .livereg  0x800ff0e , 0xfff
          jal      $$239
 # BLOCK 79: basic = 1
          sw       $15, -100($sp)
          sw       $31, -88($sp)
          lw       $15, sys_current_thread
          la       $31, 76($sp)
          sw       $31, 120($15)
          la       $31, sim_10_event
          sw       $31, 108($15)
          la       $31, sys_b0+0x2a0
          sw       $31, 140($15)
          li       $31, 539
          sw       $31, 136($15)
          la       $31, sys_a0
          sw       $31, 124($15)
          li       $31, 1
          sw       $31, 144($15)
          jal      sys_call_routine
          lw       $15, -100($sp)
          lw       $31, -88($sp)
         sw        $2 , 76($sp)
         .loc      2 541
```

```
# BLOCK 80: basic = 0
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$297
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2a4
          sw        $31, 140($15)
          li        $31, 541
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 4
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $25 , $$297
        beq       $25 , 0 , $44
          .loc      2 543
 # BLOCK 81: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, sim_bb_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2b4
          sw        $31, 140($15)
          li        $31, 543
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 1
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        la        $4 , $$357
 # BLOCK 82: basic = 0
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, version_string
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2b8
          sw        $31, 140($15)
          li        $31, 543
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 3
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $5 , version_string
        .livereg  0xc00ff0e , 0xfff
        jal       printf
          .loc      2 544
 # BLOCK 83: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, sim_bb_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2c4
          sw        $31, 140($15)
          li        $31, 544
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 2
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        move      $4 , $0
        .livereg  0x800ff0e , 0xfff
        jal       exit
          .loc      2 545
 # BLOCK 84: basic = 1
$44:
          .loc      2 547
 # BLOCK 85: basic = 1
 # BLOCK 86: basic = 0
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$296
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2cc
          sw        $31, 140($15)
          li        $31, 547
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 4
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $8 , $$296
        beq       $8 , 0 , $45
          .loc      2 548
 # BLOCK 87: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, sim_bb_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2dc
          sw        $31, 140($15)
          li        $31, 548
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 2
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
          move      $4 , $0
        .livereg  0x800ff0e , 0xfff
        jal       $$261
$45:
 # BLOCK 88: basic = 1
 # BLOCK 89: basic = 0
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$272
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2e4
          sw        $31, 140($15)
          li        $31, 549
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 6
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $4 , $$272
          .loc      2 553
          seq       $2 , $4 , 3
          move      $3 , $2
          bne       $2 , $0 , $46
 # BLOCK 90: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, sim_bb_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x2fc
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 4
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $31, -88($sp)
        seq       $2 , $4 , 4
        move      $3 , $2
        bne       $2 , $0 , $46
 # BLOCK 91: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$269
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x30c
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 5
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $2 , $$269
        seq       $2 , $2 , 0
        move      $3 , $2
        bne       $2 , $0 , $46
 # BLOCK 92: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$281
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x320
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 5
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $2 , $$281
        sne       $2 , $2 , 0
        move      $3 , $2
        bne       $2 , $0 , $46
 # BLOCK 93: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$282
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x334
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 5
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $2 , $$282
        sne       $2 , $2 , 0
        move      $3 , $2
        bne       $2 , $0 , $46
 # BLOCK 94: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$279
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x348
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 5
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $2 , $$279
        sne       $2 , $2 , 0
        move      $3 , $2
        bne       $2 , $0 , $46
# BLOCK 95: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$276
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x35c
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 5
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $2 , $$276
        sne       $2 , $2 , 0
        move      $3 , $2
        bne       $2 , $0 , $46
 # BLOCK 96: basic = 1
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$280
          sw        $31, 120($15)
          la        $31, sim_2_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x370
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 3
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        lw        $3 , $$280
        sne       $3 , $3 , 0
$46:
 # BLOCK 97: basic = 1
 # BLOCK 98: basic = 0
          sw        $15, -100($sp)
          sw        $31, -88($sp)
          lw        $15, sys_current_thread
          la        $31, $$294
          sw        $31, 120($15)
          la        $31, sim_10_event
          sw        $31, 108($15)
          la        $31, sys_b0+0x37c
          sw        $31, 140($15)
          li        $31, 553
          sw        $31, 136($15)
          la        $31, sys_a0
          sw        $31, 124($15)
          li        $31, 1
          sw        $31, 144($15)
          jal       sys_call_routine
          lw        $15, -100($sp)
          lw        $31, -88($sp)
        sw        $3 , $$294
          .loc      2 555
```

Legenda:

- indentate di un carattere: sono le istruzioni del codice assembler non-instrumentato;
- il parametro "basic" vale 1 se il blocco è il primo di un basic-block, vale 0 altrimenti.

4.3.5 Dettaglio dell'instrumentazione di un blocco

Codice di instrumentazione inserito, es. nel blocco 92 (anche basic-block):

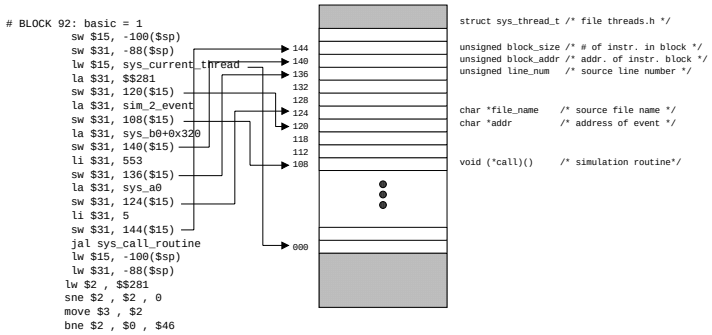


Figura 4–4. I dati raccolti in ciascun basic-block

Il tipo di informazioni raccolte in ciascun blocco viene stabilito con le opzioni sulla linea di comando di `aug` (l'esempio è relativo alle opzioni `-bcnrf`). Notare che dopo la memorizzazione delle informazioni viene chiamata la `sys_call_routine` che a sua volta richiama le opportune procedure di simulazione e tracciamento. Il file `.s` viene, infine, compilato per produrre il file oggetto `.o` che verrà collegato agli altri file dell'applicazione e, in particolare, alle librerie con le funzioni di simulazione e tracciamento.

4.3.6 Traccia prodotta dal codice in esame

Per completare l'esempio, ecco la traccia generata dal frammento di codice in esame, in formato nativo `SCL` sulla sinistra e in formato `IIP` sulla destra (v. appendice per i dettagli sui vari formati di traccia); si può facilmente verificare che la sequenza di blocchi eseguita (non riportando le chiamate a procedure di libreria) è ... , 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, ... , 79, 80, 85, 86, 88, 89, 90, 91, 92, 93, 97, 98.

```
FILE ls-ar000.dat                                                FILE ls-ar000.iip

THREAD_START    (47)  FFFFFFFF FFFFFFFF 0007DFFF 0007FFFF      6 0043AC24     0 10008880
BASIC_BLOCK     (E1)  0043AC24                                  6 0043AC28     6 0043AC34
STORE           (0A)  00000001 0043AC28 7FFFBBCC                1 7FFFBBCC     6 0043AC38
LOAD            (02)  00000002 0043AC2C 10008880                6 0043AC2C     6 0043AC3C
LOAD            (02)  00000003 0043AC34 100031C8                6 0043AC30     0 100031C8
  .
  .
  .
BASIC_BLOCK     (E1)  00400424                  ;line 528       6 00400424     1 10009188     6 0040046C
STORE           (0A)  00000001 00400428 7FFFBC74 ;line 528      6 00400428     6 00400448     0 7FFFBCB4
STORE           (0A)  00000001 0040042C 7FFFBC70 ;line 528      1 7FFFBC74     6 0040044C     6 00400470
STORE           (0A)  00000001 00400430 7FFFBC98 ;line 528      6 0040042C     6 00400450     1 10009360
STORE           (0A)  00000001 00400434 7FFFBC9C ;line 528      1 7FFFBC70     1 1000918C     6 00400474
STORE           (0A)  00000002 00400438 10009198 ;line 532      6 00400430     6 00400454     6 00400478
STORE           (0A)  00000002 00400440 10009188 ;line 533      1 7FFFBC98     6 00400458     6 0040047C
STORE           (0A)  00000003 0040044B 1000918C ;line 534      6 00400434     1 1000912C     0 7FFFBC98
STORE           (0A)  00000002 00400454 1000912C ;line 536      1 7FFFBC9C     6 0040045C
LOAD            (02)  00000002 0040045C 7FFFBC9C ;line 536      6 00400438     6 00400460
STORE           (0A)  00000001 00400464 10009130 ;line 536      6 0040043C     0 7FFFBC9C
LOAD            (02)  00000002 00400468 7FFFBCB4 ;line 538      1 10009198     6 00400464
STORE           (0A)  00000001 00400470 10009360 ;line 538      6 00400440     1 10009130
LOAD            (02)  00000003 00400474 7FFFBC98 ;line 539      6 00400444     6 00400468
  .
  .
  .
STORE           (0A)  00000001 00400480 7FFFBC94 ;line 539      6 00400480     6 004004D0     6 00400504
LOAD            (02)  00000004 00400484 100091A0 ;line 541      1 7FFFBC94     6 004004D4     6 00400508
LOAD            (02)  00000004 004004AC 1000919C ;line 547      6 00400484     6 004004D8     6 0040050C
LOAD            (02)  00000006 004004C4 10009140 ;line 548      6 00400488     0 10009140     6 00400510
BASIC_BLOCK     (E4)  004004DC                  ;line 553       6 0040048C     6 004004DC     0 10009160
LOAD            (02)  00000005 004004EC 10009138 ;line 553      6 00400490     6 004004E0     6 00400514
LOAD            (02)  00000005 00400500 10009160 ;line 553      0 100091A0     6 004004E4     6 00400518
LOAD            (02)  00000005 00400514 10009164 ;line 553      6 004004AC     6 004004E8     6 0040051C
STORE           (0A)  00000001 0040055C 10009194 ;line 553      6 004004B0     6 004004EC     6 00400520
                                                                6 004004B4     6 004004F0     6 00400524
                                                                6 004004B8     6 004004F4     0 10009164
                                                                0 1000919C     6 004004F8     6 0040055C
                                                                6 004004C4     6 004004FC     1 10009194
                                                                6 004004C8     0 10009138
                                                                6 004004CC     6 00400500
```

4.4 Tool ttf*, Trace to Trace Filter*

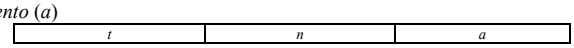
Lo strumento `ttf` (Trace to Trace Filter) ha due funzionalità principali:

- trasformare una traccia da un formato ad un altro;
- raccogliere dati statistici sulla traccia.

4.4.1 Trasformazione del formato di traccia

A prescindere dal formato fisico (*sequenza di byte*), una funzione[1] effettua una trasformazione iniziale, in modo che l'organizzazione logica della traccia sia la *sequenza di record*; ciascun record contiene i seguenti campi:

- *tipo del riferimento* (*t*);
- *n. di identificazione del processore* (*n*) che ha generato il riferimento;
- *indirizzo del riferimento* (*a*)



Mentre per i campi *n* ed *a* non esiste ambiguità sulla codifica, il campo t contiene un numero che deve, normalmente, essere semplicemente trasformato da un valore ad un altro. Tipicamente il grosso del lavoro sta, quindi, nella traduzione da formato fisico a formato logico (e viceversa durante la scrittura della traccia); l'ottimizzazione di questa operazione ha importanza fondamentale nella determinazione del costo di elaborazione della traccia, dovendo essere ripetuta (decine di) milioni di volte.

I formati attualmente letti da `ttf` sono cinque:

- `CMU`, Carnegie Mellon University;
- `MIT`, Massachusetts Institute of Technology;
- `SCL`, Stanford Computer Laboratory;
- `APT`, Arm Processor Trace;
- `IIP`, Ingegneria della Informazione a Pisa.

L'uscita è per il momento implementata solo in formato `IIP`.

Nella traduzione può aversi perdita di informazioni: il problema è correlato con quello della distorsione della traccia; ogni cura viene posta affinché la traccia di uscita non ne abbia, altrimenti avrebbe poco senso procurarsi tracce inizialmente prive di distorsione. In particolare, per tracce contenenti operazioni di sincronizzazione (e. g. `CMU`) occorre generare il *file delle sincronizzazioni* (file `.syn`).

4.4.2 Raccolta dei dati statistici

Oltre alla semplice traduzione della traccia (che può essere disabilitata) `ttf` può compiere una serie di analisi od operazioni:

- conteggio degli eventi;
- lettura non-sincronizzata;
- marcatura dei blocchi staticamente condivisi;
- analisi dello write-run.

I risultati degli ultimi due punti dipendono dalla *dimensione del blocco di cache*; infatti, questi punti riguardano non tanto gli indirizzi, ma piuttosto i blocchi in cui si può pensare partizionata la memoria; la dimensione dei blocchi, nelle architetture UMA (Uniform Memory Access), è normalmente pari alla dimensione del blocco delle cache.

4.4.3 Conteggio degli eventi

Ogni volta che viene letto un riferimento di un certo tipo viene aggiornato il contatore associato; i tipi conteggiati sono riportati in tabella:

lettura dati privati	scrittura dati privati
lettura dati condivisi	scrittura dati condivisi

lettura codice privato
lettura codice condiviso

operazioni di lock	operazioni di unlock

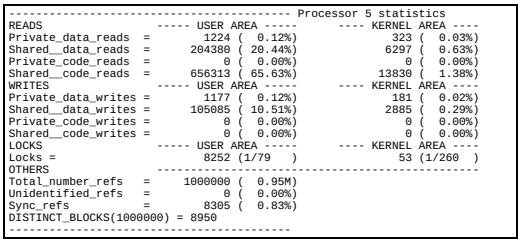
Tabella 4–2. Principali eventi conteggiati sia in area utente che in area kernel.

I riferimenti vengono inoltre classificati per area di appartenenza ovvero a seconda dello spazio di indirizzamento (utente o kernel).

I conteggi sono effettuati separatamente per ogni processore del sistema ed è riportato, per comodità, un conteggio cumulativo di tutti i riferimenti elaborati. I contatori possono essere campionati ad intervalli regolari della durata

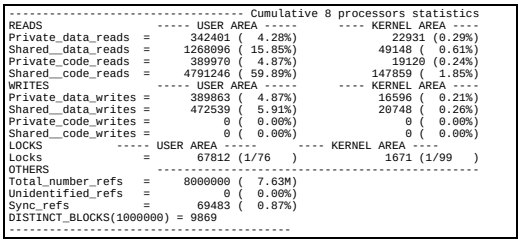
[1] Normalmente scritta da chi ha raccolto la traccia stessa o fornita con la documentazione relativa al formato di traccia.

desiderata; ciò consente di produrre grafici relativi all'andamento delle statistiche di conteggio durante l'evoluzione della traccia.
Sono contati anche i blocchi distinti; questo dato è indice della dimensione degli spazi di indirizzamento effettivamente utilizzati dall'applicazione ed ha particolare importanza per caratterizzare la traccia nelle simulazioni di cache, ove un basso numero di blocchi distinti può essere insufficiente per esaminare il sistema quando è in condizioni di massimo carico.



```
---------------------------------------- Processor 5 statistics
READS                    ----- USER AREA -----       ---- KERNEL AREA ----
Private_data_reads  =       1224 (  0.12%)              323 (  0.03%)
Shared__data_reads  =     204380 ( 20.44%)             6297 (  0.63%)
Private_code_reads  =          0 (  0.00%)                0 (  0.00%)
Shared__code_reads  =     656313 ( 65.63%)            13830 (  1.38%)
WRITES                   ----- USER AREA -----       ---- KERNEL AREA ----
Private_data_writes =       1177 (  0.12%)              181 (  0.02%)
Shared__data_writes =     105085 ( 10.51%)             2885 (  0.29%)
Private_code_writes =          0 (  0.00%)                0 (  0.00%)
Shared__code_writes =          0 (  0.00%)                0 (  0.00%)
LOCKS                    ----- USER AREA -----       ---- KERNEL AREA ----
Locks =                     8252 (1/79   )               53 (1/260  )
OTHERS                   -------------------------------------------------
Total_number_refs   =    1000000 (  0.95M)
Unidentified_refs   =          0 (  0.00%)
Sync_refs           =       8305 (  0.83%)
DISTINCT_BLOCKS(1000000) = 8950
----------------------------------------
```

Figura 4–5. Esempio di output delle statistiche di conteggio del processore 5 per 1'000'000 di riferimenti su una traccia relativa ad un sistema con 8 processori.



```
------------------------------------ Cumulative 8 processors statistics
READS                    ----- USER AREA -----       ---- KERNEL AREA ----
Private_data_reads  =     342401 (  4.28%)             22931 (0.29%)
Shared__data_reads  =    1268096 ( 15.85%)             49148 (  0.61%)
Private_code_reads  =     389970 (  4.87%)             19120 (0.24%)
Shared__code_reads  =    4791246 ( 59.89%)            147859 (  1.85%)
WRITES                   ----- USER AREA -----       ---- KERNEL AREA ----
Private_data_writes =     389863 (  4.87%)             16596 (  0.21%)
Shared__data_writes =     472539 (  5.91%)             20748 (  0.26%)
Private_code_writes =          0 (  0.00%)                0 (  0.00%)
Shared__code_writes =          0 (  0.00%)                0 (  0.00%)
LOCKS              ----- USER AREA -----       ---- KERNEL AREA ----
Locks               =      67812 (1/76   )              1671 (1/99   )
OTHERS             -------------------------------------------------------
Total_number_refs   =    8000000 (  7.63M)
Unidentified_refs   =          0 (  0.00%)
Sync_refs           =      69483 (  0.87%)
DISTINCT_BLOCKS(1000000) = 9869
----------------------------------------
```

Figura 4–6. Esempio di output delle statistiche di conteggio cumulativo per 1'000'000 di riferimenti su una traccia relativa ad un sistema con 8 processori.

In appendice sono riportati, sia in forma grafica che tabulare, i dati relativi alle probabilità di lettura, scrittura, fetch e numero di blocchi distinti (nel caso di blocco di cache di dimensione pari a 32 byte).

4.4.4 Lettura non-sincronizzata

È possibile effettuare la lettura non-sincronizzata delle tracce. Questo significa che, invece di prelevare riferimenti dal processore correntemente non bloccato, si preleva ciclicamente un riferimento per processore. È essenziale che la lettura sincronizzata sia utilizzata nella simulazioni più particolareggiate - come nel caso delle simulazioni di cache - ma anche durante l'analisi dello write-run. Tralasciando le sincronizzazioni si possono infatti introdurre distorsioni sull'ordine degli accessi [Stunkel91].
È inoltre opportuno abilitare la lettura sincronizzata durante la traduzione delle tracce contenti sincronizzazioni, per mantenere nella traccia prodotta tutte le informazioni presenti nella traccia di ingresso.
Il formato della traccia di sincronizzazione (`syn`) è riportato in appendice.

4.4.5 Marcatura dei blocchi staticamente condivisi

Qualora il campo *tipo* dei record della traccia non contenga informazioni riguardo alla appartenenza a spazi di indirizzamento condivisi da tutti i processori, si può rendere necessaria un'identificazione dei blocchi *staticamente condivisi*; con questo termine si intendono blocchi contenenti riferimenti che sono acceduti da più di un processore durante l'elaborazione di una traccia [Agarwal88, Gee93].

La marcatura è effettuata creando, dinamicamente, una lista di blocchi acceduti, il cui stato viene variato quando un diverso processore accede a quel blocco. Per non confondere accessi di processori diversi, con accessi diversi di uno stesso processore - non memorizzando l'identificatore del processore, ma solo i bit che codificano lo stato del blocco - vengono elaborati prima tutti gli accessi relativi ad un dato processore per poi passare agli accessi del processore successivo.

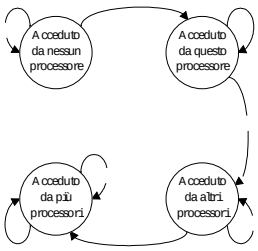


Figura 4–7. Transizioni dello stato di un blocco durante la marcatura degli indirizzi staticamente condivisi. Con linea continua sono rappresentate le transizioni che avvengono mentre si elaborano riferimenti dello stesso processore. Con linea tratteggiata sono rappresentate le transizioni che avvengono mentre si commuta da un processore a un altro.

Al contrario, altri [Eggers88] considerano condivisi i riferimenti appartenenti a zone di memoria staticamente allocate in spazi di indirizzamento condivisi, individuando i riferimenti condivisi grazie alle informazioni contenute negli header dei file dei codici oggetto.

L'individuazione dei blocchi condivisi ha importanza, in particolare, per i riferimenti a dati, sui quali si possono avere accessi in scrittura da parte di diversi processori, dando origine al *problema della coerenza delle cache*.

`ttf` permette di ottenere una lista dei blocchi condivisi, che può essere utilizzata in successive analisi.

4.4.6 Analisi dello write-run

Lo *write-run* è una metrica che caratterizza i pattern di accesso a dati condivisi. Una discussione più approfondita, insieme ai risultati delle analisi compiute su varie tracce, verrà fatta successivamente.

La metrica dello write-run è definita attraverso due funzioni di distribuzione: la *write-run length WRL(L)* e la *external re-reads XRR(L)*; l'uso delle funzioni di distribuzione caratterizza da un lato la lunghezza delle sequenze (*L*), dall'altro la frequenza di tali sequenze.

- Si dice che un blocco *ha subito uno write-run di lunghezza L*, se un processore effettua L accessi in scrittura a quel blocco, eventualmente intercalati da letture, prima che un altro processore acceda allo stesso blocco (in scrittura o in lettura). Lo write-run length dà un'indicazione sulla località di accesso a quel blocco da parte di un dato processore.
- Si dice che un blocco *ha subito L external re-reads*, se ci sono L accessi in lettura al blocco da parte di processori diversi da quello che ha appena finito uno write-run sullo stesso blocco e prima che inizi un altro write-run. Dà una indicazione del numero di processori che condividono attivamente quel blocco.

Lo strumento `ttf` consente di rilevare le funzioni di distribuzione *WRL(L)* e *XRR(L)* per i riferimenti sia a dati kernel che a dati utente e per una ampiezza di distribuzione variabile a piacere (asse x).

4.4.7 Presentazione del pannello di controllo di *ttf*

In Trace Factory lo strumento `ttf` può essere configurato attraverso un pannello di controllo; qui, è mostrato il pannello con tutte le opzioni (modalità avanzata) ma è anche possibile visualizzare un set di opzioni principali per non disperdere l'attenzione sulle varie particolarità.

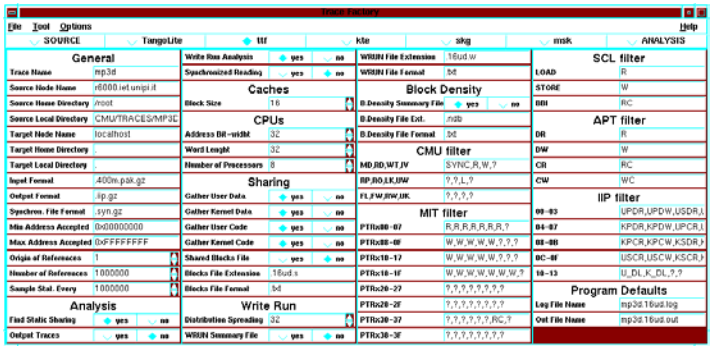


Figura 4–8. Il pannello di controllo di ttf.

4.5 Tool kte*, Kernel Trace Extractor*

Lo strumento `kte` (Kernel Trace Extractor) estrae, da una traccia contenente riferimenti ad aree di kernel/sistema, una traccia chiamata *traccia di kernel*, composta da:

- un file contente la sequenza di riferimenti del kernel,
- vari file, quanti sono i processori della traccia originaria, chiamati *tracce di posizione dei riferimenti kernel* (reference position trace, file `.rpt`).

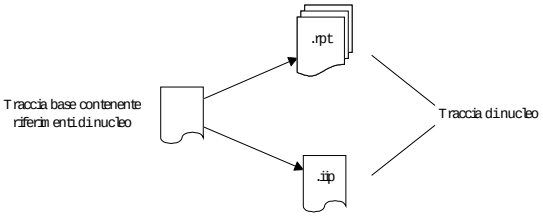


Figura 4–9. Generazione della traccia di nucleo a partire da una traccia con riferimenti di nucleo.

Se la lettura della traccia originaria non viene fatta rispettando le sincronizzazioni, la traccia di nucleo può essere diversa rispetto al caso di lettura sincronizzata, in quanto la sequenza dei riferimenti può essere riletta in un ordine diverso da quello originario.

La traccia di posizione viene associata, poi, ad ognuno dei file costituenti le tracce-base del carico.

```
K       100
U       200
K       100
U       250
K       102
U       700
K        98
U       300
K       200
U       200
K       150
U       300
K       300
U       400
K       150
U       216
K       207
U       290
K       300
U       150
```

Figura 4–10. Esempio di traccia di posizione dei riferimenti (file .rpt).

4.5.1 Presentazione del pannello di controllo di *kte*

In Trace Factory lo strumento `kte` può essere configurato attraverso un pannello di controllo; qui, è mostrato il pannello con tutte le opzioni (modalità avanzata) ma è anche possibile visualizzare un set di opzioni principali per non disperdere l'attenzione sulle varie particolarità.

Figura 4–11. Il pannello di controllo di kte.

4.6 Tool skg*, Scheduling Generator*

Lo strumento `skg` (Scheduling Generator) consente di allegare al carico della macchina target una tabella di scheduling che specifica quali sono i processi che, ad ogni quanto temporale, eseguiranno sui processori disponibili.
Il quanto temporale è in realtà specificato in termini di numero di riferimenti, essendo la nozione di tempo del tutto trascurata nell'implementazione di questo strumento.
Lo scheduling viene generato tenendo presente il classico diagramma degli stati possibili per un processo in un sistema operativo multitasking [Tanenbaum92, cap 2]. Un processo può trovarsi in stato pronto (READY), bloccato (BLOCKED) o in-esecuzione (EXE). Inizialmente, quando un processo viene creato, si trova in stato bloccato.

- Le transizioni da stato bloccato a stato pronto sono regolate dall'*algoritmo di attivazione*.

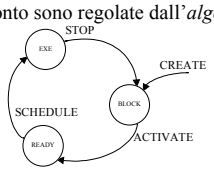


Figura 4–12. Stati di un processo e transizioni fra gli stessi.

- Le transizioni da stato pronto a in-esecuzione sono regolate dall'*algoritmo di scheduling*.
- Le transizioni da stato in-esecuzione a stato bloccato sono regolate dall'*algoritmo di arresto*.

Ogni stato ha associata una coda nella quale vengono inseriti i processi creati. Ad ogni quanto temporale, lo scheduler esamina se ci sono processori in attesa di un processo; se il numero dei processi in esecuzione è inferiore al numero di processori disponibili e ci sono processi in coda pronti o in coda bloccati, vengono chiamati in sequenza gli algoritmi di attivazione e di schedulazione; si esamina, quindi, se ci sono processi che hanno terminato e che devono essere bloccati e, successivamente, si invoca l'algoritmo di context-switch che determina l'istante iniziale e il tempo di calcolo[1] di ciascuno dei processi che devono eseguire; infine, tutti i processi vengono assegnati ad un processore ed eseguono per un quanto; dopodiché si ripete il ciclo a meno che non siano terminati i quanti di tempo che si volevano considerare.

4.6.1 Algoritmo di attivazione

Sono implementati i seguenti due algoritmi:

- *two-phase*: prima di attivare i processi bloccati, si attende che non ci siano più processi pronti;
- *noblocking*: i processi bloccati vengono subito attivati.

Col primo metodo si assicura che, prima che un processo ritorni ad eseguire, tutti gli altri abbiano eseguito una volta; si evita così la *starvation*, in cui un processo potrebbe essere scavalcato da altri senza mai riuscire ad andare in esecuzione.

4.6.2 Algoritmo di scheduling

È implementato il seguente algoritmo:

- *random*: si sceglie in maniera casuale il processo da mandare in esecuzione fra quelli che sono in stato READY; la densità di probabilità della funzione casuale è uniforme (si utilizza, in realtà, il generatore di numeri pseudo-casuali, ma con periodo 2^{32} di D. Knuth [Knuth73]).
- *least recently scheduled (n*): si esamina la storia dei processi che hanno girato su un dato processore (ultime *n* posizioni) e viene selezionato il processo meno recentemente schedulato.
- *affinity*: si esamina la storia dei processi che hanno girato su un dato processore (ultime *n* posizioni) e viene selezionato il processo più recentemente schedulato.

4.6.3 Algoritmo di arresto

Questo algoritmo si occupa semplicemente di esaminare se qualche processo ha terminato il suo tempo di calcolo; nel qual caso tale processo viene inserito nella coda dei processi bloccati.

4.6.4 Algoritmo di context-switch

Effettua la modellazione degli istanti in cui si ha context-switch, determinando il valore del tempo di calcolo del processo prima che questo vada in esecuzione.
Sono implementati i seguenti algoritmi:

- *simultaneous*: al termine di un quanto temporale, tutti i processori compiono un context-switch;
- *linear*: ad ogni quanto temporale, il context-switch avviene su uno ed un solo processore che viene scelto ciclicamente e linearmente fra i processori del sistema (il processo che subisce il context-switch è comunque casuale, se casuale è stata la scelta del processo schedulato su quel processore).

[1] Sempre in termini di numero di riferimenti da eseguire.

4.6.5 Tabella di scheduling

Il file di uscita di `skg` è rappresentato da una tabella dello scheduling (process scheduling table, file `.pst`), in cui ogni colonna corrisponde ad un processore e ogni riga corrisponde ad un quanto; i numeri indicati sono invece gli identificatori dei processi.

```
0    5    9    8
7    5    9    8
7    2    9    8
7    2    4    8
7    2    4    3
1    2    4    3
1    6    4    3
1    6    2    3
1    6    2    9
7    6    2    9
```

Figura 4–13. Tabella di scheduling per un sistema con 4 processori e 10 processi, context-switch lineare, scheduling random, attivazione two-phase.

4.6.6 Presentazione del pannello di controllo di *skg*

In Trace Factory lo strumento `skg` può essere configurato attraverso un pannello di controllo; qui, è mostrato il pannello con tutte le opzioni (modalità avanzata) ma è anche possibile visualizzare un set di opzioni principali per non disperdere l'attenzione sulle varie particolarità.

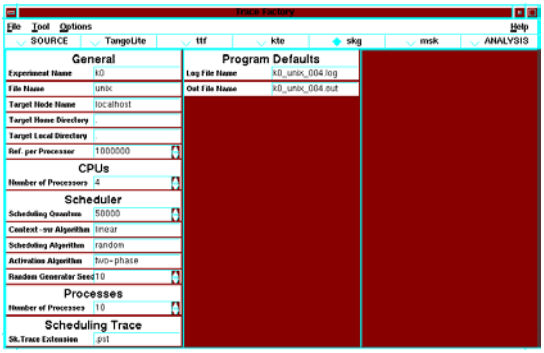


Figura 4–14. Il pannello di controllo di skg.

4.7 Tool msk*, Multiprocessor Scheduler*

Lo strumento `msk` (Multiprocessor Scheduler) consente di ottenere una traccia soddisfacente i requisiti che sono stati precedentemente elencati. La traccia prodotta può essere memorizzata nel Trace Data Base per essere oggetto di varie simulazioni in cui il carico del sistema target è dato dalla traccia stessa.

Le principali funzionalità dello strumento sono:

- schedulazione, su un numero qualsiasi di processori, di un carico rappresentato da applicazioni monoprocesso e/o multiprogrammate;
- rimappatura dei riferimenti in una porzione dello spazio di indirizzamento, secondo un dato algoritmo di paginazione;
- inserimento dei riferimenti del nucleo provenienti da un altro carico che li contiene (traccia di nucleo, prodotta da kte);
- generazione della mappa delle pagine riferite con i relativi processi che le hanno utilizzate.

Lo scheduling è di tipo *table-driven,* ovvero guidato dalla tabella di scheduling prodotta con `skg`. Questo consente di esaminare lo scheduling *prima* che venga fisicamente effettuato usando la reale durata del quanto di schedulazione.

4.7.1 Scheduling di carichi monoprocesso e/o multiprogrammati

Il *carico* è stato precedentemente definito come l'insieme di una o più tracce-base, ognuna delle quali costituisce la traccia di una applicazione monoprocesso o multiprogrammata.

La traccia-base, però, contiene indirizzi che possono trovarsi teoricamente ovunque all'interno dello spazio di indirizzamento dell'applicazione. Il problema della collisione degli spazi di indirizzamento viene risolto sia *selezionando* una zona contigua dello spazio di indirizzamento della traccia base e scartando, quindi, gli indirizzi al di fuori di tale zona, sia *rimappando* gli indirizzi con una granularità a piacere, data dalla dimensione della pagina nell'algoritmo di paginazione. Le tracce di applicazioni multiprogrammate contengono, inoltre, riferimenti a zone di memoria allocate in modo condiviso ai processi di cui l'applicazione è costituita; tali spazi condivisi sono diversi da

applicazione ad applicazione.

La lista complessiva dei processi da schedulare viene generata semplicemente mettendo in sequenza i processi che costituiscono la lista delle varie tracce-base e l'eventuale traccia di nucleo.

4.7.2 Rimappatura degli indirizzi

È basata sul concetto di *paginazione* [Tanenbaum92, cap. 3]. Si immagina che gli indirizzi contenuti in un file-traccia siano relativi ad un processo a cui è associata una *tabella delle pagine privata* per i riferimenti privati e una *tabella delle pagine condivisa* per i riferimenti condivisi; la tabella delle pagine condivisa è la stessa per tutti i processi costituenti una stessa applicazione multiprogrammata.

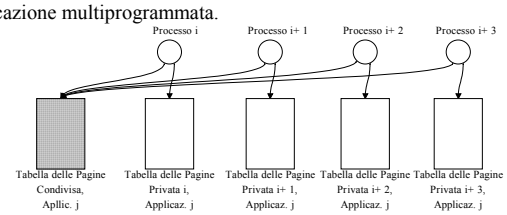


Figura 4–15. Tabelle delle pagine private e tabella delle pagine condivisa di un'applicazione multiprogrammata.

Lo schema di paginazione è *su domanda*, *senza prepaging* e *a due livelli*: non appena un processo produce un riferimento il cui indirizzo non appartiene a una pagina logica allocata (*page-fault*), viene allocata la corrispondente pagina logica. Il paginatore (*pager*) richiede ogni nuova pagina al gestore della memoria, il quale alloca le pagine fisiche secondo l'*algoritmo di allocazione* prescelto. Due livelli di paginazione consentono di ridurre l'occupazione di memoria delle tabelle delle pagine.

Così facendo, si ottiene 'gratuitamente' la separazione degli spazi di indirizzamento privati dei processi e l'accomunamento degli spazi di indirizzamento condivisi di ciascuna applicazione multiprogrammata.

È selezionabile il seguente algoritmo di allocazione della memoria fisica:

- *sequential*: lo spazio di indirizzamento fisico viene suddiviso in partizioni di uguale dimensione; il numero delle partizioni è pari al numero di processi del sistema (compreso l'eventuale processo-kernel); ogni partizione contiene, quindi, lo stesso numero di pagine fisiche; le pagine vengono allocate sequenzialmente in ciascuna partizione; le pagine logiche condivise vengono allocate nella partizione relativa al primo processo che provoca il page-fault su quella pagina[1].

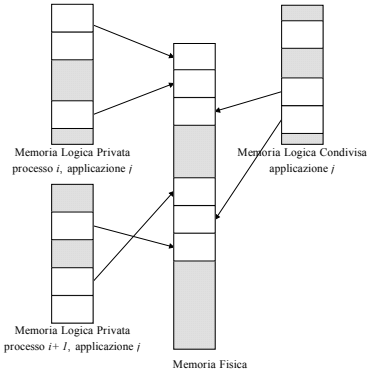


Figura 4–16. Paginazione della memoria virtuale privata e condivisa di ciascun processo.

4.7.3 Inserimento dei riferimenti della traccia di kernel

L'inserimento della traccia di kernel avviene utilizzando i file di posizione dei riferimenti (`.pst`); mentre vengono elaborati i riferimenti relativi ad un dato processo *i*, si analizza il corrispondente file `.pst` per determinare se i riferimenti di ingresso devono essere prelevati dal file-traccia relativo al processo *i* o dal file-traccia della traccia di kernel.

[1] Notare che indirizzi relativi ad una pagina condivisa, anche generati da processi diversi da quello che ha provocato il page-fault, ma aventi a comune la stessa tabella delle pagine, vengono mappati sulla stessa pagina fisica, ovviamente.

4.7.4 Mappa delle pagine riferite

Può essere interessante, per studiare le diverse strategie di allocazione della memoria o le tecniche di paginazione, disporre della lista delle pagine riferite durante ciascun quanto di scheduling, insieme all'identificatore dell'ultimo processo che ha acceduto a quella pagina.

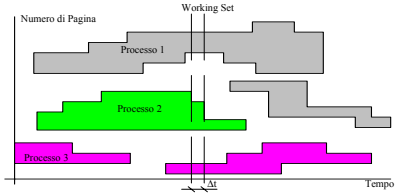


Figura 4–17. Mappa delle pagine riferite (fonte [Hwang93, fig.4.19]).

I file che si ottengono (page reference map, `.prm`) hanno il formato riportato nella seguente figura.

```
0 P0  00000000        0 P5  0003A2E8        0 P9  00068BA2        0 P10 000745D1
0 P0  00000001        0 P5  0003A2E9        0 P9  00068BA3        0 P10 000745D2
0 P0  00000002        0 P5  0003A2EA        0 P9  00068BA4        0 P10 000745D3
0 P10 000745D1                                                    0 P10 000745D4
0 P10 000745D2                                                    0 P10 000745D5
0 P10 000745D3                                                    0 P10 000745D6
                                                                  0 P10 000745D7
                                                                  0 P10 000745D8
                                                                  0 P10 000745D9
1 P7  00051745        1 P5  0003A2E8        1 P9  00068BA2        1 P8  0005D174
1 P7  00051746        1 P5  0003A2EB        1 P9  00068BA4        1 P8  0005D175
1 P7  00051747        1 P10 000745D2        1 P9  00068BA5        1 P10 000745D3
                      1 P10 000745D3                              1 P10 000745D4
                      1 P10 000745D8                              1 P10 000745D5
                      1 P10 000745D9                              1 P10 000745D7
                      1 P10 000745DA                              1 P10 000745D8
                      1 P10 000745DB                              1 P10 000745DB
                                                                  1 P10 000745DC
                                                                  1 P10 000745DD
2 P7  00051745        2 P2  0001745D        2 P10 000745D5        2 P8  0005D174
2 P7  00051747        2 P2  0001745E        2 P10 000745D6        2 P8  0005D175
2 P7  00051748        2 P2  0001745F        2 P10 000745D8        2 P8  0005D176
2 P10 000745D2        2 P10 000745D2        2 P10 000745DB
2 P10 000745D3        2 P10 000745D6        2 P10 000745DD
2 P10 000745D4        2 P10 000745D8        2 P10 000745E0
2 P10 000745D5        2 P10 000745D9        2 P10 000745E1
2 P10 000745D8        2 P10 000745DA        2 P10 000745E2
2 P10 000745DD        2 P10 000745DB        2 P10 000745E3
2 P10 000745DE        2 P10 000745DD
                      2 P10 000745DE
                      2 P10 000745DF
                      2 P10 000745E0
3 P10 000745D3        3 P2  0001745D        3 P10 000745D3        3 P8  0005D174
3 P10 000745D5        3 P2  0001745E        3 P10 000745D5        3 P8  0005D175
3 P10 000745D8        3 P2  0001745F        3 P10 000745DC        3 P8  0005D176
3 P10 000745DC        3 P10 000745D3        3 P10 000745DD        3 P10 000745D3
3 P10 000745DD        3 P10 000745DD        3 P10 000745E1        3 P10 000745DD
3 P10 000745DE        3 P10 000745E0        3 P10 000745E3        3 P10 000745E6
3 P10 000745DF        3 P10 000745E1        3 P10 000745E4
3 P10 000745E0        3 P10 000745E3        3 P10 000745E6
3 P10 000745E3        3 P10 000745E4
3 P10 000745E4        3 P10 000745E5
```

Figura 4–18. Mappa delle pagine riferite nel caso di 10 processi e 4 processori, per i quanti 0..3. Il processo kernel ha identificatore '10', mentre i processi utente hanno identificatori 0..9.

4.7.5 Presentazione del pannello di controllo di *msk*

In Trace Factory lo strumento `msk` può essere configurato attraverso un pannello di controllo; qui è mostrato il pannello con tutte le opzioni (modalità avanzata) ma è anche possibile visualizzare un set di opzioni principali per non disperdere l'attenzione sulle varie particolarità.

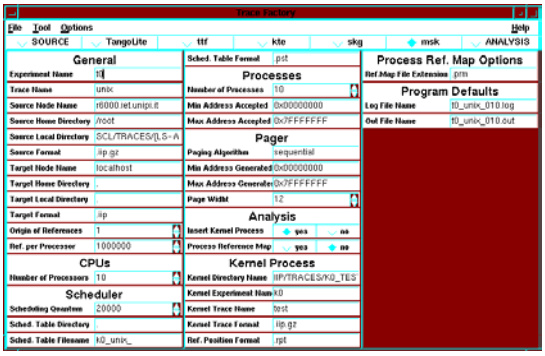


Figura 4–19. Il pannello di controllo di msk.

4.8 Un caso concreto: traccia di workstation UNIX multiprocessore.

L'esperimento preso come modello è quello riportato in [Vashow93]. Vashow produce una traccia di Encore-Multimax mentre sta eseguendo del carico a divisione di tempo di tipo general-purpose; la situazione è tipica di una macchina UNIX: 10 utenti sono in stato idle, altri 10 stanno eseguendo uno script composto da comandi di utilità fra cui:

ar -t	awk	banner "x"	cat	cd	chmod
cmp	compress	cp	date	df	diff
du	echo	egrep	expr	false	find
finger	file	host	hostid	hostname	make
mkdir	mv	ln	ln -s	ls	od
ps	pwd	rm	ruptime	rwho	size
sort	split -10	strip	sync	tail	tee
time ls -FCa	touch	true	uncompress	who	wc

Tabella 4–3. Insieme rappresentativo di comandi usati durante una sessione UNIX.

I comandi sono stati scelti in modo da rappresentare un'insieme dei comandi tipicamente usati durante una sessione UNIX. Contrariamente ad altre tracce descritte nello stesso lavoro, questa traccia è ovviamente prelevata mentre il Multimax è in modo multi-user.

L'inizio dell'esecuzione dei vari script viene ritardata di un'opportuno Δt e la traccia è prelevata dopo che tutti gli script sono in esecuzione; se un'utente termina il proprio script, inizia a rieseguirlo (esecuzione ciclica).

4.8.1 Realizzazione tramite Trace Factory

Dal paniere delle applicazioni - tracciate con TangoLite e provenienti da altre fonti - sono state selezionate delle 'fette' di traccia-base della lunghezza di 1 milione di riferimenti, in base alle schede riportate in appendice. Dai grafici della densità di blocchi si sono scelte le applicazioni con *densità maggiore* e con *maggiore probabilità di scrittura* per poter subito individuare, nei test iniziali, quali applicazioni avrebbero potuto generare un traffico sufficientemente sostenuto sul bus.

Il carico è stato composto utilizzando diverse tracce, ognuna delle quali rappresentasse l'attività di un utente durante la sessione UNIX.

Il carico risulta composto da 24 processi. Le tracce sono state filtrate con il tool `ttf` per riportarle dal formato originario al formato `IIP`.

Con il tool `kte` si sono quindi estratti i riferimenti del nucleo, estraendo un numero di riferimenti sufficienti a ricoprire la lunghezza dei 24 file traccia.

Con il tool `skg` è stato generato uno scheduling considerando una durata del quanto pari a 1'000'000/24 riferimenti.

Con `msk` si sono fabbricate le tracce per il caso di 2, 4, 6, 8, 10, 12, 14, 16 processori.

Queste nove carichi sono, poi, stati utilizzati per confrontare le prestazioni del sistema mentre le cache usano i protocolli di coerenza Dragon, Berkeley, UCR3 [Gee93, Prete95]; per completezza sono riportati due dei grafici più significativi dei risultati prodotti col simulatore CSIM [Ricciardi95].

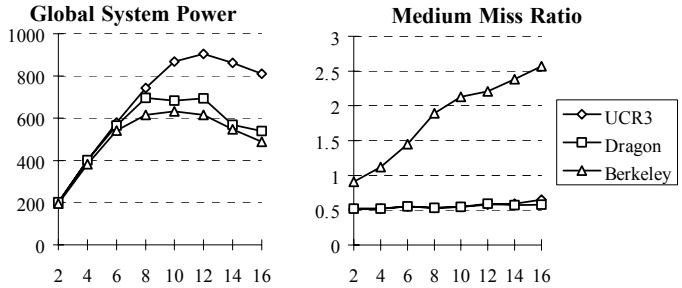


Figura 4–20. Grafici del Global System Power e del Miss Ratio Medio, ricavabili con CSIM (i grafici sono relativi alla prova "p70e3" in cui i 24 processi sono costituiti da tre applicazioni multiprogrammate a granularità fine (i.e. ECAS) ognuna costituita da 8 file-traccia).

5 Metodologie e Risultati delle Analisi condotte

Gli strumenti di Trace Factory consentono di operare su una enorme quantità di dati, costituiti dai riferimenti di cui le tracce sono composte. La grande mole di informazioni non può che essere gestita, in termini statistici, attraverso valori medi e funzioni di distribuzione; ad esempio, una delle metriche proposte in letteratura per valutare i pattern di accesso ai dati condivisi è lo write-run.
In questo capitolo sarà analizzato in dettaglio uno dei set di tracce già introdotti -il set CMU- riguardante applicazioni parallele; in particolare, saranno discussi i metodi utilizzati per ricavare informazioni statistiche e presentati i risultati sulla condivisione dei dati - la cui gestione ottimale è un parametro critico per le prestazioni dei multiprocessori - attraverso la metrica dello write-run.

5.1 Write-run

Lo *write-run* è una metrica che caratterizza i pattern di accesso a dati condivisi attraverso due funzioni di distribuzione: *write-run length WRL(L)* e *external re-reads XRR(L)*; l'uso di funzioni di distribuzione caratterizza da un lato la lunghezza delle sequenze (*L*), dall'altro la frequenza di tali sequenze.

- Si dice che un blocco *ha subito uno write-run di lunghezza L*, se un processore effettua L accessi in scrittura a quel blocco, eventualmente intercalati da letture, prima che un altro processore acceda allo stesso blocco (in scrittura o in lettura). Lo write-run length dà un'indicazione sulla località di accesso a quel blocco da parte di un dato processore.

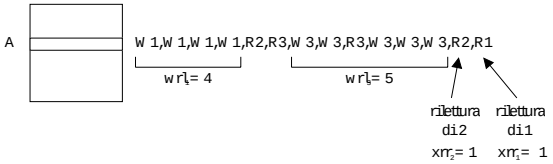


Figura 5–1. `Write-run` *ed* `external re-read` *all'indirizzo A; Wx = scrittura da parte del processore x, Rx = lettura da parte del processore x. Nell'esempio si sono verificati 1 write-run di lunghezza 4 e 1 di lunghezza 5; inoltre si hanno 2 riletture esterne di lunghezza 1.*

Nell'esempio mostrato in figura 1, la lettura R2 termina lo write-run del processore 1.

- Si dice che un blocco *ha subito L external re-reads*, se ci sono L accessi in lettura al blocco da parte di processori diversi da quello che ha appena finito uno write-run sullo stesso blocco e prima che inizi un altro write-run. Dà una indicazione del numero di processori che condividono attivamente quel blocco.

Nell'esempio di figura 1, le read che si trovano al di fuori di write-run sono letture esterne, ma *solo le due letture finali - R2 e R1 - sono riletture* in quanto, nella sequenza degli accessi a quell'indirizzo, il processore corrispondente aveva già acceduto all'indirizzo stesso.
Alcune osservazioni:

- gli indirizzi che si stanno considerando appartengono *a dati condivisi*;
- gli write-run sono unità non sovrapposte;
- ogni indirizzo ha il proprio pattern di accessi in scrittura ovvero la propria sequenza di write-run.

5.1.1 Uso dello write-run

Oltre a caratterizzare i pattern di accesso ai dati condivisi lo write-run consente di [Eggers88]:

- evidenziare le differenze fra i protocolli di coerenza, spiegando come questi vengono influenzati dai diversi pattern di accesso;
- dare un valore approssimato del costo di mantenimento della coerenza.

Valutazione di protocolli di coerenza

Un uso 'naturale' dello write-run è per valutare quale delle due principali classi di protocolli di coerenza - write-update o write-invalidate - risulta più vantaggiosa, quando nel sistema multiprocessore è presente un determinato carico.
Write-run lunghi suggeriscono di usare un protocollo che invalida: infatti, il costo della invalidazione iniziale è ammortizzato durante lo write-run. Viceversa write-run corti (in particolare di lunghezza 1) suggeriscono di adottare un protocollo che aggiorna la memoria[1]: infatti il costo della prima write è uguale per le due classi di protocolli ma, in write-invalidate, c'è *in più* il costo relativo all'invalidazione.

I vari protocolli che invalidano trattano in maniera pressoché identica le letture; per questo, le analisi del costo del mantenimento della coerenza si concentrano soprattutto sulle operazioni di scrittura. Le operazioni di lettura hanno importanza nella valutazione delle transazioni inutili, nei protocolli di tipo write-invalidate, quando sono *riletture*

[1] A patto di ignorare gli effetti della falsa condivisione.

esterne: infatti, il numero di letture esterne, da parte di processori diversi, dà un indicazione del numero di processori che attivamente[1] condividono un dato blocco e, se il blocco era stato invalidato, tali letture provocano transazioni che si sarebbero potute evitare in un protocollo write-update. Parecchie riletture esterne danno altresì un'indicazione di quanto può essere vantaggioso un protocollo write-update.
Ricapitolando:

- un protocollo write-invalidate può risultare vantaggioso se si ha: alta lunghezza degli write-run e basso numero di external-reread;
- un protocollo write-update può risultare vantaggioso se si ha: bassa lunghezza degli write-run e alto numero di external-reread.

È importante la *misura combinata* di questi due valori; infatti dal solo write-run non è possibile dire quale classe di protocolli è più vantaggiosa: ad esempio, anche se si ha bassa lunghezza degli write-run, ma non si hanno riletture esterne si possono avere migliori prestazioni in uno schema write-invalidate.

Caratterizzazione della condivisione

Si può quantificare la *condivisione* di un indirizzo dal:

- numero totale di write-run che si verificano a quell'indirizzo, ovvero dal numero di volte che cambia il processore che scrive a quell'indirizzo.

Si può quantificare la *competizione* su un indirizzo condiviso in diversi modi:

- numero di riletture esterne; supponendo che un processore non rilegga più di una volta lo stesso indirizzo, fra due write-run, questo valore è pari al n. di processori che condividono l'indirizzo;
- numero di write-run rispetto al numero totale di indirizzi condivisi in scrittura;
- numero di processori che sta per effettuare una lock su un indirizzo unlocked (questa situazione è rilevabile se ci sono molte più letture che scritture su un indirizzo di lock; un numero di scritture esattamente pari alle letture significa totale assenza di processori in attesa attiva[2]).

Metrica	Aspetto della condivisione misurato
Lunghezza dello write-run	Uso consecutivo da parte dello stesso processore di un dato condiviso
Numero di write-run	Condivisione
Numero di riletture esterne	Competizione
Rapporto fra write-run e numero totale di indirizzi condivisi in scrittura	Competizione
Numero di processori in attesa attiva per una lock	Competizione

Tabella 5–1. Metriche di condivisione basate sullo write-run.

5.1.2 Valutazione delle funzioni di distribuzione *WRL(L)* e *XRR(L)*

Ancora seguendo il lavoro della Eggers [Eggers88, Eggers90], le funzioni di distribuzione WRL(L) e XRR(L) sono state valutate implementando una macchina a stati finiti il cui comportamento è descritto dal seguente diagramma.

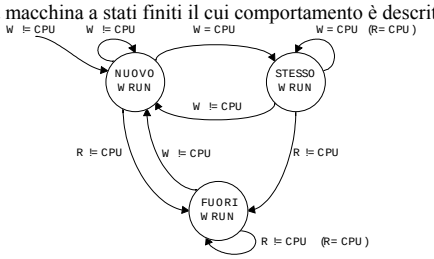


Figura 5–2. Transizioni che determinano le funzioni WRL(L) e XRR(L); R e W indicano operazioni di lettura e scrittura, rispettivamente; !=CPU e =CPU indicano che l'operazione è stata effettuata da una diversa CPU e dalla stessa CPU, rispettivamente.

In particolare va osservato che le riletture esterne sono valutate in maniera approssimata, assumendo cioè che tutte le letture esterne siano riletture; questo semplifica l'implementazione in quanto non è necessario conservare una struttura dati per ogni indirizzo, che indichi se quell'indirizzo è già stato acceduto o meno da un particolare processore. L'ipotesi è realistica perché assume di porsi in una condizione di regime, in cui ogni indirizzo è almeno stato 'inizializzato'.

---

[1] Si dice, invece, che dei processori condividono *passivamente* un blocco quando questo rimane nella cache di un processore anche se non verrà più acceduto dallo stesso processore; la condivisione passiva può essere causata dalla migrazione dei processi e dall'esecuzione di routine di interrupt su differenti processori [Prete95].

[2] Supponendo un algoritmo di lock del tipo `test-and-set`; se l'algoritmo fosse `test-and-test-and-set` l'assenza di attesa attiva si avrebbe quando il numero di scritture è pari al doppio di quello di letture sull'indirizzo di lock [Eggers88].

La Eggers ha utilizzato questo semplice modello per assegnare ad ogni arco del diagramma a stati un costo di mantenimento della coerenza; in questo modo ha costruito un simulatore per valutare le prestazioni di due protocolli rappresentativi delle classi write-update e write-invalidate che sono rispettivamente Firefly e Berkeley OwnerShip.

L'uso che se ne è fatto in questa tesi è, invece, più limitato e consiste nella caratterizzazione di tracce; nei precedenti capitoli si è evidenziato come la scelta delle tracce sia uno dei problemi principali per costruire il carico della macchina di cui si intendono valutare le prestazioni.

*** ZONE USER_DATA ***
mp3d - 1000000 r/cpu - cbs=32 - TWRL=484 TXRR=494 TSW=1101 TSR=2700
E[WRL]= 2.21 - DStd[WRL]= 1.38 - E[XRR]= 1.69 - DStd[XRR]= 1.15

L	WRL	XRR	%WRL	%XRR	%SW	%SR
0	-	0	-	0.00	-	0.00
1	181	235	37.40	47.57	16.90	27.07
2	144	236	29.75	47.77	26.89	54.38
3	84	2	17.36	0.40	23.53	0.69
4	47	9	9.71	1.82	17.55	4.15
5	19	1	3.93	0.20	8.87	0.58
6	6	4	1.24	0.81	3.36	2.76
7	0	0	0.00	0.00	0.00	0.00
8	0	1	0.00	0.20	0.00	0.92
9	2	0	0.41	0.00	1.68	0.00
10	0	5	0.00	1.01	0.00	5.76
11	0	0	0.00	0.00	0.00	0.00
12	0	0	0.00	0.00	0.00	0.00
13	1	0	0.21	0.00	1.21	0.00
14	0	0	0.00	0.00	0.00	0.00
15	0	0	0.00	0.00	0.00	0.00
16	0	0	0.00	0.00	0.00	0.00
17	0	0	0.00	0.00	0.00	0.00
18	0	0	0.00	0.00	0.00	0.00
19	0	0	0.00	0.00	0.00	0.00
20	0	0	0.00	0.00	0.00	0.00
21	0	0	0.00	0.00	0.00	0.00
22	0	0	0.00	0.00	0.00	0.00
23	0	0	0.00	0.00	0.00	0.00
24	0	0	0.00	0.00	0.00	0.00
25	0	0	0.00	0.00	0.00	0.00
26	0	0	0.00	0.00	0.00	0.00
27	0	0	0.00	0.00	0.00	0.00
28	0	0	0.00	0.00	0.00	0.00
29	0	0	0.00	0.00	0.00	0.00
30	0	0	0.00	0.00	0.00	0.00
31	0	0	0.00	0.00	0.00	0.00
>31	0	1	0.00	0.20	0.00	3.69

Tabella 5–2. WRL(l) e XRR(L) per l'applicazione MP3D.8, primo milione di riferimenti, dim. del blocco di cache 16B, TWRL = totale write-run, TXRR = totale external reread, TSW = totale shared write, TRW = totale shared read, ZONA DATI UTENTE.

*** ZONE KERNEL_DATA ***
mp3d - 1000000 r/cpu - cbs=32 - TWRL=2052 TXRR=2045 TSW=7440 TSR=30713
E[WRL]= 2.20 - DStd[WRL]= 3.00 - E[XRR]= 1.77 - DStd[XRR]= 3.01

L	WRL	XRR	%WRL	%XRR	%SW	%SR
0	-	9	-	0.44	-	0.18
1	1218	1570	59.36	76.77	20.35	31.01
2	391	245	19.05	11.98	13.06	9.68
3	158	41	7.70	2.00	7.92	2.43
4	72	29	3.51	1.42	4.81	2.29
5	7	35	0.34	1.71	0.58	3.46
6	47	1	2.29	0.05	4.71	0.12
7	3	3	0.15	0.15	0.35	0.41
8	10	2	0.49	0.10	1.34	0.32
9	2	2	0.10	0.10	0.30	0.36
10	42	10	2.05	0.49	7.02	1.98
11	9	4	0.44	0.20	1.65	0.87
12	22	1	1.07	0.05	4.41	0.24
13	0	1	0.00	0.05	0.00	0.26
14	2	3	0.10	0.15	0.47	0.83
15	4	7	0.19	0.34	1.00	2.07
16	0	0	0.00	0.00	0.00	0.00
17	0	3	0.00	0.15	0.00	1.01
18	0	7	0.00	0.34	0.00	2.49
19	0	1	0.00	0.05	0.00	0.38
20	5	2	0.24	0.10	1.67	0.79
21	6	12	0.29	0.59	2.10	4.98
22	0	4	0.00	0.20	0.00	1.74
23	0	5	0.00	0.24	0.00	2.27
24	1	0	0.05	0.00	0.40	0.00
25	3	0	0.15	0.00	1.25	0.00
26	0	2	0.00	0.10	0.00	1.03
27	0	0	0.00	0.00	0.00	0.00
28	0	0	0.00	0.00	0.00	0.00
29	0	1	0.00	0.05	0.00	0.57
30	4	0	0.19	0.00	2.00	0.00
31	0	0	0.00	0.00	0.00	0.00
>31	46	45	2.24	2.20	24.59	28.44

Tabella 5–3. WRL(l) e XRR(L) per l'applicazione MP3D.8, primo milione di riferimenti, dim. del blocco di cache 16B, TWRL = totale write-run, TXRR = totale external reread, TSW = totale shared write, TRW = totale shared read, ZONA DATI KERNEL.

5.2 Probabilità di scrittura, lettura, fetch

Il conteggio degli eventi fornisce il numero di letture dati n_r, scritture dati n_w, letture codice (fetch) n_f per ognuna delle tracce analizzate; questo numero, rapportato al numero totale degli eventi N fornisce, con buona stima, la probabilità che un dato accesso in memoria risulti rispettivamente una lettura, una scrittura o fetch. In altri termini se,

$$N = n_r + n_w + n_f \quad \text{allora:} \quad P(DR) = \lim_{N\to\infty} \frac{n_r}{N} \qquad P(DW) = \lim_{N\to\infty} \frac{n_w}{N} \qquad P(CR) = \lim_{N\to\infty} \frac{n_f}{N}$$

avendo indicato con *P(DR)*, *P(DW)*, *P(CR)* rispettivamente le probabilità di lettura, scrittura e fetch.

Per verificare la correttezza del conteggio, vengono ricavati anche i dati relativi alla *scrittura di codice* (operazione, ovviamente, normalmente non consentita). Nel set di tracce CMU sono peraltro stati riscontrati degli indirizzi che vengono dapprima scritti come dati e successivamente riletti, più volte, come codice (e.g. traccia MP3D).

In appendice, sono riportati i grafici che rappresentano l'andamento delle probabilità suddette lungo le tracce; generalmente, queste probabilità si mantengono abbastanza costanti, così che si possono assumere questi dati come una caratteristica che contraddistingue l'applicazione o, talvolta, la categoria di cui essa è rappresentativa.

Per comodità di lettura queste probabilità (relative a 6 applicazioni del set CMU) vengono riportate nella pagina seguente (oltre che in appendice). Sulla colonna di sinistra sono collocate le applicazioni a granularità fine mentre sulla colonna di destra le applicazioni a granularità grossa e media.

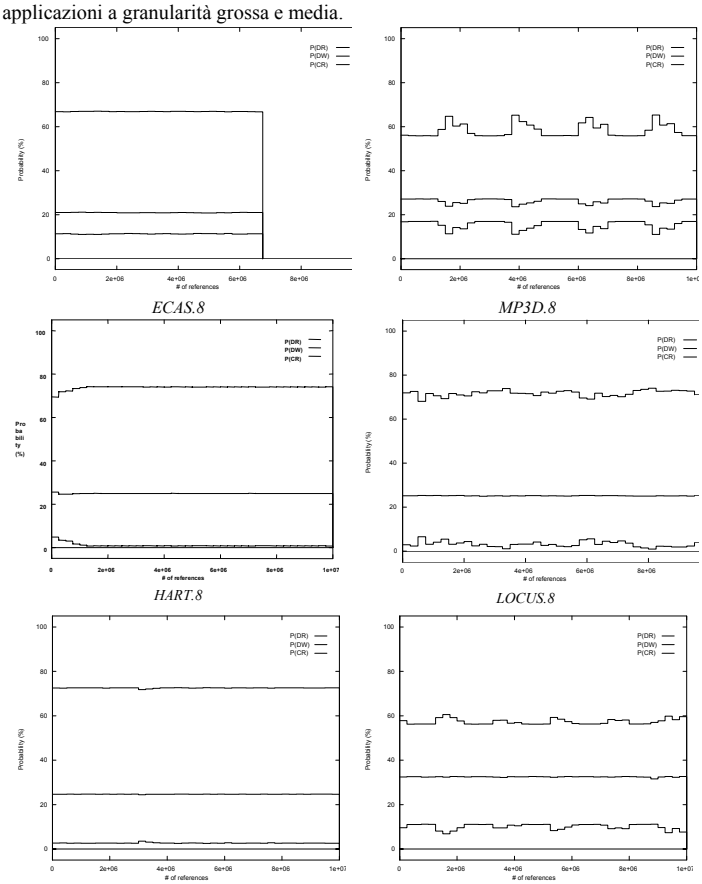


Figura 5–3. I 3 'oscillogrammi' che compaiono in ogni figura sono relativi (nell'ordine dall'alto verso il basso) alla probabilità di fetch, lettura, scrittura per i primi 10 milioni di riferimenti del set di tracce CMU.

I valori medi su 10 milioni di riferimenti dei precedenti grafici sono riportati nella seguente tabella.

Applicazione	P(DR)%	P(DW)%	P(CR)%
ECAS.8	21.0	11.2	66.8
HART.8	25.0	0.9	74.1
PDE.8	24.7	2.7	72.6

Applicazione	P(DR)%	P(DW)%	P(CR)%
MP3D.8	26.8	15.3	57.9
LOCUS.8	25.1	3.1	71.8
MS_TRACER.8	32.5	10.1	57.4

Tabella 5–4. Valori medi su 10 milioni di riferimenti delle probabilità di lettura scrittura e fetch nel set CMU.

5.3 Blocchi distinti

Per caratterizzare le tracce in termini di effettiva memoria utilizzata dall'applicazione, sono stati contati i blocchi distinti; il numero di blocchi distinti *DBN* (Distinct Blocks Number) moltiplicato la dimensione del blocco di cache *CBS* (Cache Block Size) può fornire una 'misura' dello spazio di indirizzamento *ASS* (Address Space Size) di un'applicazione: $ASS = DBN \cdot CBS$.

Nella successiva figura è riportata la densità incrementale di blocchi; anziché analizzare la traccia 'a fette', è più significativo vedere il numero totale di blocchi all'aumentare del numero di riferimenti elaborati (notare che i blocchi distinti nelle varie fette potrebbero non essere indipendenti); la densità è riportata in forma di percentuale incrementale, per evidenziare la *variazione* di blocchi da una fetta all'altra: questo è anche la percentuale *minima* di blocchi distinti in quella fetta; la percentuale è relativa al rapporto fra incremento di blocchi distinti e numero di blocchi distinti che si avrebbero nel caso di *accessi in memoria unicamente consecutivi* (un valore scelto arbitrariamente come riferimento).

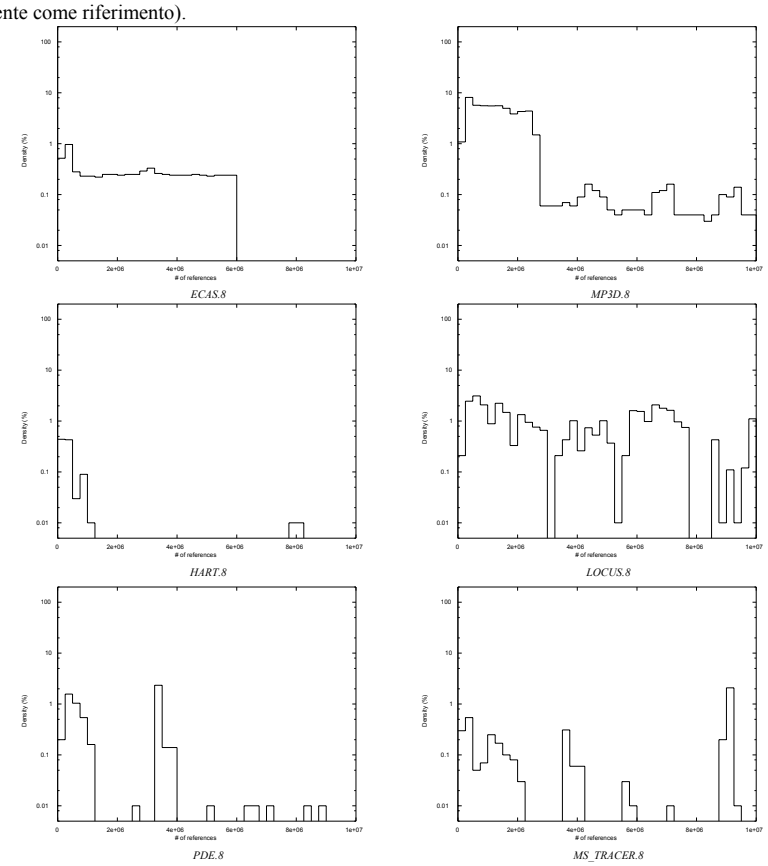


Figura 5–4. Densità incrementale di blocchi distinti da 32B nel set di tracce CMU.

Applicazione	DBN	DBN%	ASS(KB)
ECAS.8	2902	2.32	907
HART.8	256	0.20	80
PDE.8	1546	1.23	483

Applicazione	DBN	DBN%	ASS(KB)
MP3D.8	13174	10.50	4116
LOCUS.8	8627	6.90	2696
MS_TRACER.8	1999	1.60	625

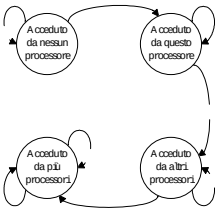
Tabella 5–5. DBN = numero massimo di blocchi distinti (normalizzato per milione di riferimenti); DBN% = numero massimo di blocchi distinti (normalizzato per milione di riferimenti) rapportato al numero massimo di blocchi distinti in caso di accessi unicamente consecutivi; ASS = dimensione complessiva dello spazio di indirizzamento (in KB e senza normalizzazioni) nel set CMU.

I blocchi distinti sono calcolati associando ad ogni blocco acceduto un bit d'accesso'[1].

Figura 5–5. Stati di un blocco durante il conteggio dei blocchi distinti.

[1] Poiché, potenzialmente, il numero di blocchi distinti può essere molto alto è stato fatto uso di uno schema di memorizzazione basato su un vettore a due livelli: il primo livello memorizza i puntatori a gruppi consecutivi di indirizzi in cui è partizionato lo spazio di indirizzamento; il secondo livello è costituito dai bit che codificano lo stato. Per ridurre ulteriormente l'occupazione di spazio ogni byte è stato suddiviso in modo da sfruttare anche i singoli bit di cui esso è costituito. Infine, per non far pesare l'overhead legato alla gestione di questa struttura, sono state utilizzate delle tabelle che 'al volo' forniscono lo stato successivo.

5.4 Blocchi condivisi



Il numero di blocchi condivisi *SBN* (Shared Blocks Number) fornisce una misura dello spazio di indirizzamento condiviso *SSS* (Shared Space Size) dell'applicazione, in maniera analoga a quanto visto per i blocchi distinti:

$$SSS = SBN \cdot CBS.$$

Il numero di blocchi condivisi è stato ricavato utilizzando la seguente macchina a stati finiti, in cui ogni stato è codificato con 2 soli bit[1]:

Figura 5–6. Transizioni dello stato di un blocco durante la marcatura degli indirizzi staticamente condivisi. Con linea continua sono rappresentate le transizioni che avvengono mentre si elaborano riferimenti dello stesso processore. Con linea tratteggiata sono rappresentate le transizioni che avvengono mentre si commuta da un processore a un altro.

L'analisi dei blocchi condivisi è stata effettuata *solo sul primo milione di riferimenti*; in questo caso, però, si è analizzato cosa succedeva al variare delle dimensioni del blocco di cache.
Si può osservare che, per quasi tutte le tracce, il numero dei blocchi condivisi tende circa a dimezzare; questo si può spiegare col *principio di inclusione* [Gee93], in base a cui eventi che accadono su blocchi di dimensione più piccola si verificano anche su blocchi di dimensione più grande. In questo caso, coppie di blocchi adiacenti condivisi risultano in un unico blocco condiviso di dimensione doppia e un blocco 'sparisce'. C'è però un fenomeno che contrasta questa diminuzione dei blocchi condivisi e la traccia MP3D.8 lo dimostra palesemente.

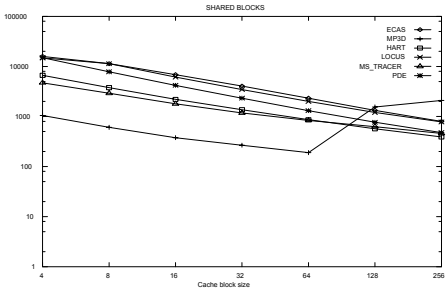


Figura 5–7. N. di blocchi condivisi al variare della dim. del blocco di cache per il set CMU.

Blocco	ECAS.8	HART.8	PDE.8	MP3D.8	LOCUS.8	MS_TRACER.8
4	15850	6651	14910	1048	14838	4714
8	11362	3776	7813	607	11364	2912
16	6846	2199	4188	378	6140	1788
32	4050	1369	2324	267	3457	1175
64	2328	865	1310	190	1997	834
128	1329	567	767	1550	1205	627
256	796	390	476	2088	777	459

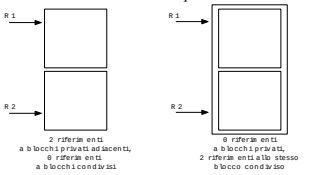
Tabella 5–6. Blocchi condivisi al variare del blocco per il set CMU.

ECAS.8	HART.8	PDE.8	MP3D.8	LOCUS.8	MS_TRACER.8
fine	non definita	asincrono	grossa	grossa	media

Tabella 5–7. Granularità delle applicazioni del set CMU [Vashow93].

ECAS.8	HART.8	PDE.8	MP3D.8	LOCUS.8	MS_TRACER.8
199KB	97.5KB	119KB	522KB	195KB	114K

*Tabella 5–8. Dimensione degli spazi di indirizzamento condivisi stimati in base SSS = SBN*CBS, utilizzando la massima dimensione del blocco.*



L'aumento di blocchi condivisi è dovuto al fatto che riferimenti privati in blocchi adiacenti vengono a trovarsi nello stesso blocco condiviso, quando si raddoppia la dimensione del blocco stesso; si può anche osservare che si ha una certa percentuale 'di crescita' rispetto al teorico dimezzamento.

Figura 5–8. Generazione di blocchi condivisi fra CPU 1 e CPU 2, per coalescenza.

5.5 Analisi dello write-run

Sono già state introdotte le funzioni *WRL(L)* e *XRR(L)*, è stata definita la loro interpretazione e mostrato come vengono ricavate. In questo paragrafo vengono mostrati i risultati rilevati sul set di tracce CMU[2].
Nel caso di blocco di cache pari a 16B e a 32B sono stati rilevati i seguenti grafici per *WRL(L)* normalizzata rispetto al numero totale di write-run,

$$WRUN\%(L) = \frac{WRL(L)}{\sum_{k=1}^{\infty} WRL(k)} \cdot 100$$

e per la percentuale di scritture condivise *SH.W%(L)*.
Si può osservare, ad esempio per PDE.8, in cui il fenomeno è più pronunciato, uno spostamento dei grafici verso lunghezze di write-run più elevate, indicativo di un aumento del valor medio.
Questa serie di grafici ha lo svantaggio di presentare poca selettività nell'individuazione delle applicazioni che hanno uno write-run più alto.

[1] Anche per il conteggio dei blocchi condivisi è stato adottato lo stesso schema di memorizzazione delle informazioni visto nella nota del paragrafo precedente; l'unica variazione è dovuta al fatto che gli stati possibili sono stavolta 4 invece di 2 e occorrono quindi due bit per memorizzare lo stato.

[2] Tutti i dati che seguono sono relativi al caso di lettura sincronizzata delle tracce.

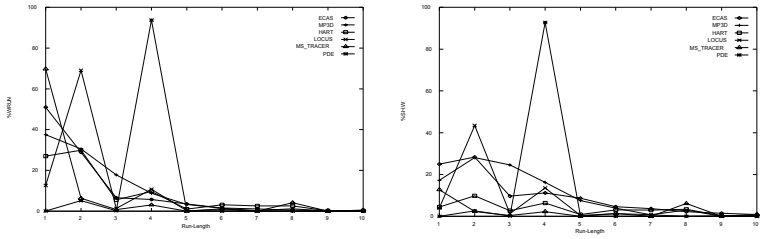


Figura 5–9. WRL(L) normalizzata rispetto al n. totale di write-run e scritture condivise, riportate in percentuale e blocco=16B.

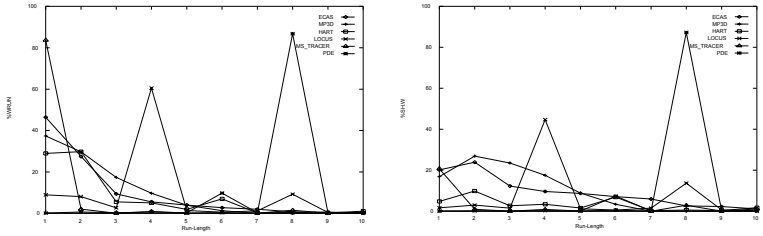


Figura 5–10. WRL(L) normalizzata rispetto al n. totale di write-run e scritture condivise, riportate in percentuale e blocco=32B.

Per ottenere selettività è stata effettuata la seguente trasformazione dei precedenti grafici:

$$P_{WRUN}(L) = 100 - \sum_{k=1}^{L} WRUN\%(k) \qquad P_{SH.W}(L) = 100 - \sum_{k=1}^{L} SH.W\%(k)$$

Le quantità $P_{WRUN}(L)$ e $P_{SH.W}(L)$ sono le funzioni cumulative delle rispettive funzioni *WRUN%* e *SH.W%*.

Nel caso di dimensione del blocco di cache di 16B e 32B si hanno I seguenti grafici. Si può osservare come, sia nel caso di blocco di 16B che di 32B, le applicazioni ECAS.8 e LOCUS.8 siano abbastanza rappresentative delle situazioni estreme con write-run rispettivamente più basso e più alto; da un confronto che le schede riportate in appendice si ricava che ECAS.8 ha una granularità fine, mentre LOCUS.8 ha una granularità grossa: questo è pienamente riscontrabile dai precedenti grafici, in quanto una granularità maggiore indica una maggior probabilità di scrivere sullo stesso dato.

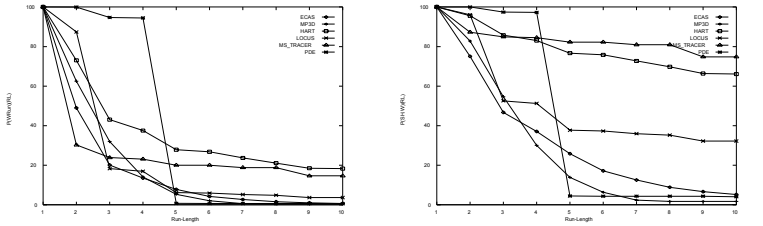


Figura 5–11. WRL(L) cumulativa normalizzata rispetto al n. totale di write-run e scritture condivise, riportate in % e (blocco=16B).

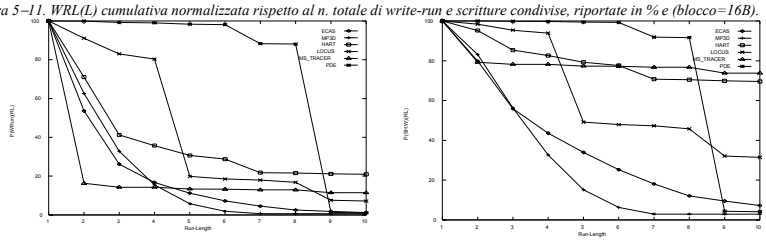


Figura 5–12. WRL(L) cumulativa normalizzata rispetto al n. totale di write-run e scritture condivise, riportate in % (blocco=32B).

5.5.1 Analisi del valor medio e della deviazione standard

I precedenti grafici rappresentano *WRL(L)* ma non sono facilmente leggibili; *WRL(L)* e *XRR(L)* possono essere caratterizzate più propriamente ricorrendo a parametri globali che quantifichino la funzione di distribuzione; data la loro natura statistica è naturale ricorrere al *valor medio* e alla *deviazione standard*.

Il valor medio e la varianza sono (detto $t = \sum_{1}^{\infty} WRL(k) \quad \Rightarrow \quad F_k^{WRL} = \frac{WRL(k)}{t}$ e analogamente per XRR):

$$\mu_k = E[k] = \sum_{k=1}^{\infty} k \cdot F_k(k) \qquad \sigma_k^2 = E[(k-\mu_k)^2] = \sum_{k=1}^{\infty} (k-\mu_k)^2 \cdot F_k(k)$$

dove $F_k(k)$ è la funzione di distribuzione (*WRL(L)* o *XRR(L)*). La deviazione standard non è altro che la radice della varianza: $\sigma_k = \sqrt{\sigma_k^2}$.

Sono riportate due tabelle:

- la prima è relativa al val. medio (*mu*) e alla dev. standard (*si*) di *WRL(L)* e di *SH.W(L),* al variare della dimensione del blocco di cache, sia per la zona dati utente (ud) che per la zona dati kernel (kd)
- la seconda è relativa al val. medio (*mu*) e alla dev. standard (*si*) di *XRR(L)* e di[1] *SH.R(L),* al variare della dimensione del blocco di cache, sia per la zona dati utente (ud) che per la zona dati kernel (kd).

Per ECAS.8, MP3D.8 e MS_TRACER.8 si possono osservare, in corrispondenza di 16B e 32B i tre più bassi valori per *WRL*-medio e valori superiori ad 1 di *XRR*-medio: queste tre applicazioni sono anche quelle con maggior Global System Power nello studio effettuato in [Prete95b].

TRACE	CBS	muWRud	siWRud	muSWud	SiSWud	muWRkd	siWRkd	muSWkd	siSWkd
ecas	4	1.42	0.92	2.04	2.33	3.07	4.68	8.76	8.87
ecas	8	1.68	1.56	2.83	3.14	2.85	4.47	8.31	8.59
ecas	16	2.07	2.03	3.71	4.04	2.81	4.47	8.17	8.82
ecas	32	2.34	2.31	4.32	4.10	2.75	4.20	7.02	8.30
ecas	64	2.62	2.78	5.22	5.19	2.58	3.95	6.46	7.83
ecas	128	2.87	2.90	5.78	5.47	2.68	3.88	6.80	7.72
ecas	256	3.03	3.20	6.29	6.01	3.04	4.08	7.63	7.79
mp3d	4	2.06	1.18	2.69	1.33	1.89	2.34	3.63	4.15
mp3d	8	2.11	1.18	2.73	1.32	1.81	2.32	3.70	4.71
mp3d	16	2.14	1.21	2.78	1.37	1.96	2.64	4.25	5.72
mp3d	32	2.18	1.31	2.92	1.73	2.16	2.86	4.47	5.92
mp3d	64	4.40	2.63	5.97	3.28	2.13	3.05	4.78	6.48
mp3d	128	6.63	2.10	7.30	2.44	2.46	3.35	5.66	6.48
mp3d	256	10.67	6.35	14.20	7.01	2.95	3.55	6.24	5.81
hart	4	3.02	4.85	7.67	9.07	2.93	4.00	6.60	7.47
hart	8	2.98	3.74	4.42	6.04	3.34	4.54	7.46	7.66
hart	16	3.54	4.23	5.17	6.26	3.83	5.26	8.86	8.09
hart	32	4.04	5.13	7.26	7.22	3.57	5.04	7.28	7.89
hart	64	4.20	5.66	8.98	7.93	3.44	4.98	6.84	7.84
hart	128	3.43	4.20	4.91	6.19	3.76	5.10	7.31	7.63
hart	256	4.16	5.11	6.55	6.96	4.06	5.50	8.24	8.09
locus	4	1.49	2.59	3.48	6.43	3.87	5.29	9.12	7.80
locus	8	1.61	2.63	2.79	5.43	3.54	5.10	8.50	8.03
locus	16	2.45	2.49	3.47	5.12	3.45	4.91	8.15	7.84
locus	32	4.38	3.16	5.23	4.61	3.46	4.95	8.19	8.05
locus	64	7.33	4.67	8.88	5.35	3.36	4.94	8.02	8.18
locus	128	8.47	6.69	10.10	5.80	3.24	4.72	7.72	8.27
locus	256	5.77	7.89	6.73	8.83	3.50	4.72	8.03	7.97
ms_tracer	4	2.92	4.41	4.34	5.94	2.43	4.14	6.37	8.22
ms_tracer	8	2.07	2.92	1.81	4.25	2.38	3.84	5.74	7.84
ms_tracer	16	1.90	3.10	2.49	5.01	2.43	3.80	5.78	7.67
ms_tracer	32	2.08	4.25	5.04	7.77	2.72	4.20	6.72	8.42
ms_tracer	64	1.41	2.61	2.69	5.85	2.64	3.94	6.29	7.85
ms_tracer	128	1.18	1.90	2.27	5.56	2.85	4.15	7.06	8.18
ms_tracer	256	1.29	1.48	1.65	4.91	3.11	4.32	7.43	8.03
pde	4	1.00	0.06	0.96	0.12	3.08	4.65	8.10	8.22
pde	8	2.00	0.12	1.93	0.22	2.86	4.14	6.47	7.20
pde	16	3.88	0.53	3.81	0.57	2.79	3.90	5.89	6.71
pde	32	7.64	0.91	7.51	0.75	2.88	4.23	6.71	7.46
pde	64	14.87	2.19	14.86	1.49	2.79	4.23	7.04	7.97
pde	128	11.87	11.11	11.53	11.27	2.87	4.29	7.24	8.10
pde	256	1.05	4.14	0.63	3.69	3.17	4.30	7.27	7.88

Tabella 5–9. Valor medio e deviazione standard di WRL(L) e SH.W(L) per dati utente e dati kernel (set CMU).

TRACE	CBS	muXRud	siXRud	muSRud	siSRud	muXRkd	siXRkd	muSRkd	siSRkd
ecas	4	2.00	3.38	5.07	7.50	1.73	3.67	3.50	6.15
ecas	8	2.38	3.59	5.19	7.33	1.99	3.55	5.76	7.84
ecas	16	2.83	3.95	5.62	7.42	2.07	3.71	5.47	7.64
ecas	32	3.21	4.27	5.98	7.51	2.22	3.85	5.74	7.56
ecas	64	3.44	4.42	5.99	7.34	2.03	3.33	5.67	7.39
ecas	128	3.75	4.85	6.66	7.84	2.30	4.06	5.81	7.39
ecas	256	3.87	5.07	6.90	7.96	2.73	4.54	6.03	7.01
mp3d	4	1.73	1.11	1.77	0.57	1.98	4.17	4.09	7.01
mp3d	8	1.78	1.24	2.01	1.14	2.32	4.48	6.40	7.85
mp3d	16	1.81	1.31	2.16	1.38	2.12	4.22	5.20	7.28
mp3d	32	1.84	1.49	2.39	2.18	2.10	3.97	4.92	7.16
mp3d	64	2.33	1.48	3.11	2.73	1.83	3.07	5.34	7.10
mp3d	128	4.82	1.01	5.03	0.95	2.05	3.62	6.83	8.17
mp3d	256	4.46	1.20	4.78	1.27	3.37	5.61	6.17	7.19
hart	4	2.78	5.98	4.77	6.43	1.91	3.57	5.09	6.64
hart	8	2.46	4.19	6.75	8.23	2.27	3.10	6.15	6.85
hart	16	3.06	4.38	6.59	7.60	2.60	3.64	7.22	7.25
hart	32	3.69	4.84	6.58	7.55	2.70	3.81	7.44	7.57
hart	64	3.82	4.93	7.01	7.55	2.70	3.82	7.69	7.50
hart	128	4.32	5.34	6.78	7.50	2.54	3.45	6.82	7.05
hart	256	4.62	5.72	6.88	7.54	2.71	3.45	6.72	6.89
locus	4	2.92	5.43	4.15	7.83	1.31	1.79	3.50	5.57
locus	8	3.46	5.86	3.98	7.72	1.90	2.73	5.25	6.66
locus	16	3.49	5.52	3.30	6.97	2.36	3.82	5.41	6.66
locus	32	3.50	4.93	2.49	5.56	2.45	3.89	5.41	6.49
locus	64	3.82	5.12	3.14	5.74	2.23	3.34	6.14	7.13
locus	128	4.83	7.36	5.15	8.22	2.55	4.10	5.81	6.68
locus	256	3.54	6.86	3.51	7.96	2.86	4.08	5.72	6.13
ms_tracer	4	2.08	4.23	8.18	6.22	1.44	1.94	3.55	5.33
ms_tracer	8	3.16	6.78	16.89	10.39	1.96	3.18	5.30	6.67
ms_tracer	16	2.62	5.56	9.11	8.51	2.18	3.13	5.62	6.71
ms_tracer	32	2.44	5.01	4.53	6.10	2.52	3.83	5.79	6.28
ms_tracer	64	3.38	5.99	6.88	8.24	2.53	3.51	6.39	6.97
ms_tracer	128	2.28	4.79	4.66	7.50	2.73	3.69	6.01	6.50
ms_tracer	256	1.40	4.31	3.38	6.56	3.00	4.01	6.38	6.57
pde	4	3.00	0.37	3.04	0.37	1.36	1.64	3.22	4.74
pde	8	5.00	0.70	5.09	0.69	1.89	2.76	5.27	6.92
pde	16	6.99	1.34	7.23	1.34	2.06	2.96	5.71	6.94
pde	32	10.81	2.70	11.47	2.66	2.30	3.10	5.82	6.71
pde	64	16.87	4.11	17.21	2.79	2.42	3.44	6.04	7.00
pde	128	0.56	2.96	0.31	2.72	2.49	3.40	6.12	6.97
pde	256	0.52	2.64	0.26	2.46	2.65	3.64	6.51	7.19

Tabella 5–10. Valor medio e deviazione standard di XRR(L) e SH.R(L) per dati utente e dati kernel (set CMU).

[1] *SH.R(L)* rappresenta la percentuale di letture condivise, in maniera analoga a *SH.W(L)* per le scritture.

Riportando in forma grafica i *valori medi* di *WRL(L)* e *XRR(L),* si ha un migliore visibilità della situazione al variare del blocco di cache per le varie applicazioni del set.

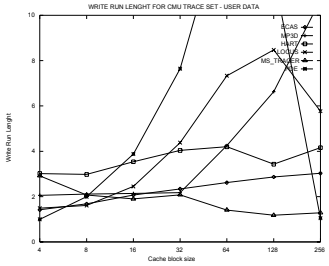


Figura 5–13. Lunghezza di write-run al variare della dim. del blocco di cache per il set CMU.

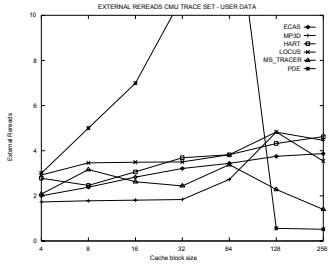


Figura 5–14. Numero di reread esterne al variare della dim. del blocco di cache per il set CMU.

Per quanto riguarda la zona kernel i risultati sono molto più 'piatti'; ciò è parzialmente spiegabile col fatto che, mentre nella zona dati utente le strutture dati possono variare moltissimo a seconda dell'applicazione, il kernel ha strutture dati che sono sempre più o meno le stesse: descrittori di processo, buffer, code, tabelle hash e così via.

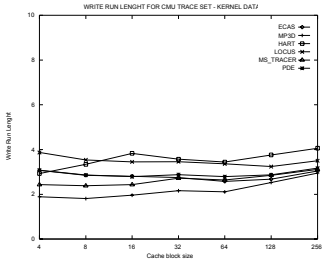


Figura 5–15. Lunghezza di write-run al variare della dim. del blocco di cache per il set CMU.

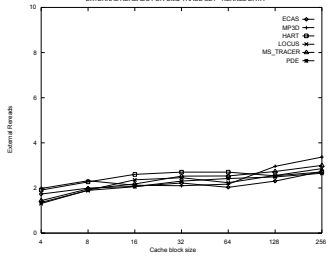


Figura 5–16. Numero di reread esterne al variare della dim. del blocco di cache per il set CMU.

Per avere un'idea complessiva del comportamento delle applicazioni del set di tracce CMU, nei confronti della metrica dello write-run, sono infine riportati due grafici riassuntivi nei quali, per ciascuna dimensione del blocco di cache, è stata fatta la media, sulle 6 applicazioni, dei valori medi di *WRL(L)* (analogamente a quanto fatto in [Gee93]) e di *XRR(L)* (trascurato, invece, nello studio appena citato).

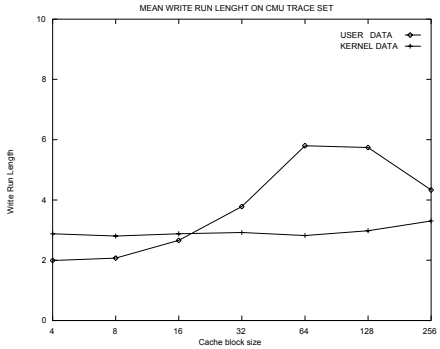


Figura 5–17. Media del valori medi di WRL(L) per le 6 applicazioni del set CMU.

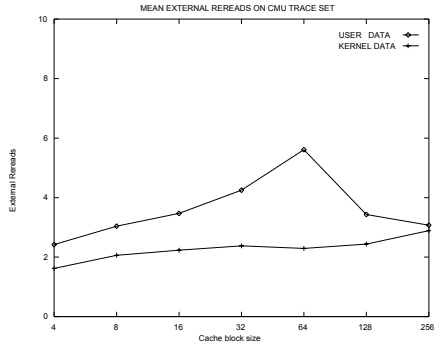


Figura 5–18. Media del valori medi di XRR(L) per le 6 applicazioni del set CMU.

5.6 Conclusioni

La messa a punto di un sistema per generare tracce, contenenti le informazioni desiderate, per macchine multiprocessore costituite da un qualsiasi numero di processori ha consentito di *elevare il lavoro di ricerca nell'ambito delle architetture dei sistemi di elaborazione*, al Dipartimento di Ingegneria della Informazione di Pisa, fornendo una dimensione in più in cui spaziare per analizzare il comportamento dei sistemi al variare dei parametri e per documentare il lavoro prodotto.
Le analisi effettuate in questo ultimo capitolo possono essere ripetute anche al variare di parametri, come il numero di processori, aumentando le possibilità di analisi.

Oltre due GigaByte di file di tracce prodotti sono ora disponibili per ogni tipo di analisi o simulazione; ogni grossa compagnia commerciale che effettua ricerca e sviluppo nel settore dell'informazione ha investito, soprattutto nell'ultimo decennio, cospicue energie per produrre ed elaborare tracce reali delle applicazioni; nell'ambito universitario, università quali Stanford, il MIT e la Carnegie Mellon spendono da diversi anni energie in questo settore e hanno fornito il materiale di base per questo lavoro di tesi; altre università - quali la New Mexico State University - hanno recentemente costituito un Trace Data Base disponibile attraverso rete (*http://tracebase.nmsu.edu/*).

Grazie a questo lavoro, anche qui al Dipartimento di Ingegneria della Informazione di Pisa c'è ora la possibilità di generare e manipolare tracce reali correlandole direttamente ai sorgenti delle applicazioni.

Lo strumento di Trace Factory denominato `ttf` (Trace to Trace Filter) legge e scrive tracce di vario formato; per non far dipendere l'implementazione dello strumento dal formato letto, viene utilizzato uno 'stadio di disaccoppiamento' che consente di avere da un lato una visione uniforme del formato di traccia o *organizzazione logica* e dall'altro consente di interfacciarsi con la struttura fisica della traccia.

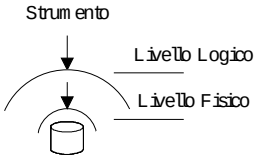
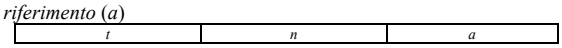


Figura A1–1. Relazione fra livello logico e livello fisico nell'accesso ad una traccia.

L'organizzazione logica della traccia è la *sequenza di record*, in cui ciascun record contiene i campi:

- *tipo del riferimento* (*t*);
- *n. di identificazione del processore* (*n*) che ha generato il riferimento;
- *indirizzo del riferimento* (*a*)



A livello fisico la traccia è invece una *sequenza di byte*. Alcuni formati fisici sono parzialmente compressi e richiedono routine di lettura particolarmente elaborate (e.g. `CMU`); altri, memorizzano direttamente le informazioni con codifica esadecimale (e.g. `SCL`, `MIT`); altri ancora, scrivono le informazioni in ASCII (e.g. `APT`, `IIP`) con il vantaggio di poter leggere direttamente le tracce senza bisogno di particolari strumenti.
Dato che oggi sono disponibili efficienti programmi di compressione (e in futuro le cose possono migliorare) quest'ultima codifica presenta il vantaggio della semplicità, lasciando la risoluzione del problema della compressione (d'altra parte importante, [Larus93]) ai livelli più bassi.

A1.1 Formato di traccia `CMU`

Il formato è stato ideato dal "Center for High Performance Computing" (CHCP) del Worcester Polytechnic Institute[1], con l'obiettivo di ottenere la maggior generalità possibile e allo stesso tempo una buona efficienza di memorizzazione. Alcuni campi non sono necessariamente usati, ma ciò non comporta un allungamento della traccia, proprio grazie al metodo di organizzazione fisica utilizzato. Le tracce sono fra le più complete, contenendo i riferimenti generati dal kernel e le sincronizzazioni fra i vari processori.

Nomenclatura dei file-traccia

Ogni file-traccia è relativo ad un determinato processore; questo evita di dover memorizzare lo stesso identificatore del processore per ogni riferimento dello stesso file; tale identificatore è contenuto nel nome del file-traccia.

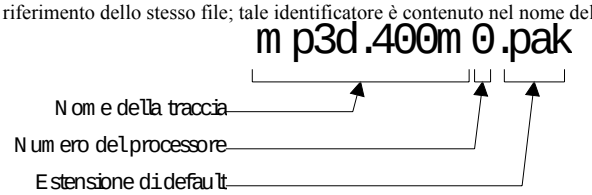


Figura A1–2. Nomenclatura dei file-traccia CMU.

Il nome della traccia è a sua volta composto da due campi: il primo (nell'esempio `mp3d`) è il nome (o una abbreviazione del nome) della applicazione che ha generato il file; il secondo (nell'esempio `400m`) indica il numero di byte occupato dai soli indirizzi dei riferimenti contenuti in tutti i file-traccia che compongono quella traccia-base (ad esempio, `400m` significa 400MB ovvero[2], essendo ogni indirizzo pari a 4 byte, 100M riferimenti, ovvero 12.5M riferimenti per file-traccia).

[1] Per ulteriori informazioni contattare il prof. Dan Siewiorek, *siewiorek@silvertip.edrc.cmu.edu*.

[2] 'M' o 'm' in questo contesto significa 1024K ovvero 1048576.

Codifica del formato

Il file-traccia inizia con un header lungo 4 byte che memorizza, in modalità *little-endian*[1], un identificatore della versione del formato.

Successivamente seguono, fino alla fine del file, vari record fisici di lunghezza variabile da 1 a 12 byte e aventi la seguente struttura:

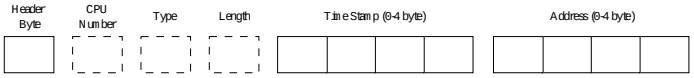


Figura A1–3. Struttura del record fisico del formato CMU.

I vari campi che seguono l'header-byte possono essere anche tutti assenti, in base a quanto specificato dai bit che compongono tale byte.

La struttura dell'header-byte è mostrata nella seguente figura, mentre la codifica dei vari campi è elencata nella successiva tabella.

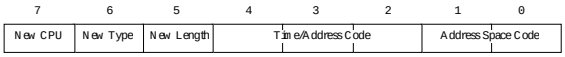


Figura A1–4. Struttura dell'header byte del record fisico.

New CPU	(bit 7)
0	Stessa CPU
1	È presente il byte che indica la nuova CPU

New Type	(bit 6)
0	Stesso tipo di riferimento
1	È presente il byte che indica il nuovo tipo del rif.

New Length	(bit 5)
0	Stessa lunghezza dell'istruzione
1	È presente il byte che indica la nuova lunghezza

Time/Address Code	(bit 4-2)
000	Nessun indirizzo, né timestamp, a_off=+4
001	1 byte indirizzo=a, nessun timestamp, a_off=a+4
010	2 byte indirizzo=a, nessun timestamp, a_off=a+4
011	Indirizzo assoluto, nessun timestamp
100	2 byte indirizzo=a, 2 byte timestamp=t_off, a_off=a+4
101	Indirizzo assoluto, 2 byte timestamp=t_off
110	2 byte indirizzo=a, timestamp assoluto
111	Indirizzo assoluto, timestamp assoluto

Address Space Code	(bit 1-0)
00	User Code
01	Kernel Data
10	User Data
11	Kernel Code

Tabella A1–1. Codifica del'header-byte del record fisico; a_off=offset dell'indirizzo, t_off=offset del timestamp.

Si può dimostrare che tecniche di questo tipo consentono di codificare la maggioranza dei riferimenti con un solo byte [Johnson94]; la riduzione effettiva delle dimensioni di file-traccia risulta circa pari a 2/3.
Il campo tipo contiene un valore che identifica il tipo dell'operazione; sono previsti i seguenti tipi:

TAG_NAME	VALORE	TIPO DELL'OPERAZIONE
MD	0	read/modify/write
RD	1	read public
WT	2	write invalidate
IV	3	invalidate
RP	4	read private
RO	5	read owner
LK	6	Lock Operation
UW	7	update write
FL	9	Flush
FW	10	flush write
RW	18	response write
UK	22	Unlock
END_OF	255	end of file type for common formats

Tabella A1–2. I tipi fisici del formato CMU.

Nelle tracce esaminate sono stati riscontrati i soli tipi `MD`, `RD`, `WT`, `LK`. Il campo `MD` è, inoltre, stato usato per codificare i timestamp, i quali non vengono inseriti come specificato precedentemente, bensì nel campo indirizzo

[1] *Little-endian* e *big-endian* sono le due modalità con cui vengono memorizzate le word composte di n byte: little-endian significa che la sequenza è composta dai byte *0, 1, 2, ... , (n-1)* essendo *0* il byte meno significativo e *(n-1)* quello più significativo della word; in big-endian invece si ha *(n-1), (n-2), (n-3), ... , 0.*

delle operazioni di tipo `MD`; tali timestamp non sono, inoltre, i valori reali, ma costituiscono una sequenza monotonicamente crescente che parte dal valore 0 e che viene via-via incrementata; i timestamp si trovano subito dopo ogni operazione di tipo `LK`, e consentono di ricostruire l'esatta sincronizzazione esistente fra i vari flussi che costituiscono i file-traccia.

Ricostruzione delle sincronizzazioni

Le operazioni che coinvolgono strutture dati condivise sono contenute nelle regioni critiche, l'accesso alle quali è garantito che avvenga in modo corretto grazie alle operazioni di lock/unlock.

I timestamp consentono di elaborare i riferimenti nell'esatta sequenza in cui vengono generati. Ogni volta che viene trovato un timestamp basta confrontarne il valore con quello precedentemente incontrato; se la differenza fra i due valori è pari ad 1 allora può continuare l'elaborazione del corrente file-traccia, altrimenti si sospende l'elaborazione di quel particolare file-traccia e si esamina ciclicamente lo stato degli altri file-traccia per determinare quello col timestamp desiderato e la cui elaborazione può riprendere (esiste senz'altro un file-traccia che contiene il timestamp cercato e, quindi, non c'è pericolo di deadlock).

Filtro verso il formato logico

Nella sezione '`CMU Filter`' di `ttf`, per le prove effettuate in questa tesi, è stata specificata la seguente traduzione da tipo fisico a tipo logico:

```
[CMU filter]
 MD,RD,WT,IV   = SYNC,R,W,?
 RP,RO,LK,UW   = ?,?,L,?
 FL,FW,RW,UK   = ?,?,?,?
```

Figura A1–5. Sezione riguardante il filtro CMU nello strumento `ttf`.

dove `SYNC` sta per *operazione di sincronizzazione*, `R` per *read*, `W` per *write*, `L` per *lock* e `?` per *riferimento non identificato*.

A1.2 Formato di traccia `MIT`

Il formato è stato messo a punto da A. Agarwal e D. Chaiken del "Laboratory for Computer Science" del Massachusetts Institute of Technology di Cambridge[1]. Consiste semplicemente nella memorizzazione in formato binario dei valori *n*, *t* ed *a* (in quest'ordine) del formato logico.

Le tracce non contengono riferimenti al kernel, sono state generate da simulatori di sistemi multiprocessore e all'inizio contengono una lunga sequenza di accessi allo stesso indirizzo su ciascun processore (modello SPMD, Single Program Multiple Data).

Nomenclatura dei file-traccia

Non esistono i file-traccia per ciascun processore, ma un unico file-traccia *composto* che rappresenta la sequenza di indirizzi che vengono inviati sul bus del sistema. La sincronizzazione è quindi implicita nella sequenza in cui i riferimenti sono scritti all'interno del file.

La traccia *composta* viene inizialmente spacchettata da, `ttf`, per poter utilizzare le operazioni logiche che vengono effettuate anche sulle tracce di altri formati.

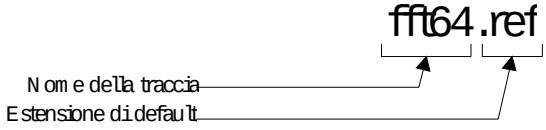


Figura A1–6. Nomenclatura dei file-traccia composto in formato MIT.

Il nome della traccia è a sua volta composto da due campi: il primo (nell'esempio `fft`) è il nome (o una abbreviazione del nome) della applicazione che ha generato il file; il secondo (nell'esempio 64) è il numero di processori totale dalla macchina da cui è stato estratta la traccia.

Per quanto riguarda le tracce separate è stata adoperata la seguente convenzione:

[1] Per ulteriori dettagli contattare D. Chaiken, *chaiken@hing.ls.mit.edu*.

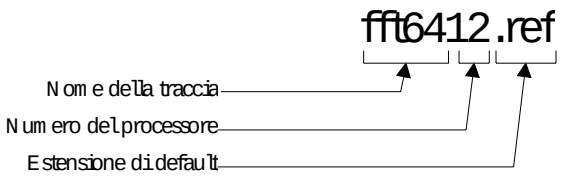


Figura A1–7. Nomenclatura dei file-traccia separato in formato MIT.

Codifica del formato

Il file è composto da una sequenza di record a lunghezza fissa pari a 6 byte e aventi la seguente struttura:

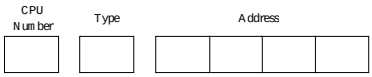


Figura A1–8. Struttura del record fisico del formato MIT.

Il campo tipo contiene un valore che identifica l'operazione; solo i valori da 0 a 63 sono effettivamente utilizzati.

La seguente tabella mostra le etichette associate a ciascun tipo:

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x00	rb	rw	rl	rq	ro	rp	rfa	x07
	mb	mw	ml	mq	mo	x0d	x0e	x0f
0x10	imb	imw	iml	imq	imo	x15	x16	x17
	wb	ww	wl	wq	wo	wp	wfa	x1f
0x20	scbb	sbr	isp	pcbb	p0br	p1br	scboff	psl
	RPTE	WPTE	TBMISS	x2b	x2c	x2d	s2e	x2f
0x30	ASPEC	DSPEC	x32	rmask	gprsp	IBYTE	ifetch	opcode
	tstart	tend	x3a	comm	x3c	x3d	dummy	filler

Tabella A1–3. Tipi fisici del formato MIT.

Per quanto riguarda la zona ombreggiata in chiaro:

- `r` = read
- `w` = write
- `im` = interlocked modify
- `m` = modify
- `b` = byte (8 bit)
- `w` = word (16 bit)
- `l` = long (32 bit)
- `q` = quad-word (64 bit)
- `o` = octal-word (128 bit)

Le operazioni `rp` e `wp` sono read e write *private*; `rfa` e `wfa` sono operazioni di tipo *fetch-and-add* (operazioni che servono per implementare una sincronizzazione atomica a N vie con complessità indipendente da N [Hwang93, par. 7.1.3]).

Un operazione importante è `ifetch` che significa *instruction fetch*.

Il codice `filler` serve per indicare che il sistema di tracciamento non riesce a memorizzare correttamente l'informazione nel formato di traccia. Le rimanenti operazioni sono usate per tracce del sistema VAX e possono essere ignorate nelle simulazioni trace-driven.

Filtro verso il formato logico

```
[MIT filter]
 PTRx00-07     = R,R,R,R,R,R,R,?
 PTRx08-0F     = W,W,W,W,W,?,?,?
 PTRx10-17     = W,W,W,W,W,?,?,?
 PTRx18-1F     = W,W,W,W,W,W,W,?
 PTRx20-27     = ?,?,?,?,?,?,?,?
 PTRx28-2F     = ?,?,?,?,?,?,?,?
 PTRx30-37     = ?,?,?,?,?,?,RC,?
 PTRx38-3F     = ?,?,?,?,?,?,?,?
```

Figura A1–9. Sezione riguardante il filtro MIT nello strumento `ttf; R = read, W = write, RC = Read Code.`

A1.3 Formato di traccia SCL

Il formato deriva dagli stessi ideatori di TangoLite, S.R. Goldschmidt e H. Davis dello "Stanford Computer Laboratory" dell'omonima Università[1]. Consiste nella semplice memorizzazione in formato binario delle informazioni ricavate dalle *sonde software* inserite all'interno del programma. Data la disponibilità del codice sorgente che genera questo formato, esso è facilmente modificabile secondo le proprie esigenze.

Nomenclatura dei file-traccia

Ogni file-traccia è relativo ad un determinato processore; questo evita di dover memorizzare lo stesso identificatore del processore per ogni riferimento dello stesso file; tale identificatore è contenuto nel nome del file-traccia.

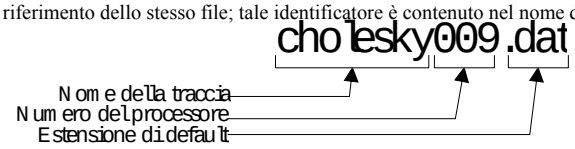


Figura A1–10. Nomenclatura dei file-traccia SCL.

Codifica del formato

Il file è composto da una sequenza di record a lunghezza variabile da 5 a 21 byte - a seconda delle opzioni prescelte nel simulatore/tracciatore (file `config.sim`) e nel tool di instrumentazione - con la seguente struttura:

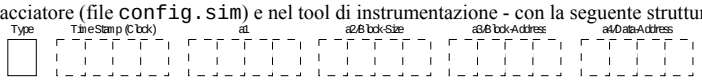


Figura A1–11. Struttura del record fisico del formato SCL.

Il primo byte indica il tipo o, per usare la terminologia SCL, l'*event-type*.
Il campo Clock è presente solo per gli eventi di tipo <224 mentre, contemporaneamente, si è messa l'opzione `trace_irefs=0` in `config.sim`.

Le successive 4 word, identificate anche con a1, a2, a3, a4, possono contenere varie informazioni:

- per valori *da 16 a 223*, sono presenti tutte e 4 le word; gli eventi relativi riguardano il rilevamento delle statistiche del simulatore, eventi di sincronizzazione, eventi legati ai processi (context-switch, send/receive, allocazione di memoria e così via); un evento particolare è quello di inizio thread per il quale a3 ed a4 contengono rispettivamente il numero della prima e dell'ultima pagina del range di pagine private da 12 bit assegnate al processo;

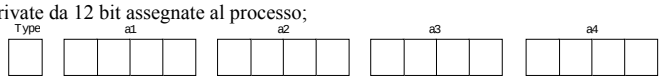


Figura A1–12. Formato SCL per event-type da 16 a 223.

- per valori *da 0 a 15*, gli eventi sono tracciati solo se `trace_drefs=1`; è sempre presente il campo a4/Data-Address in quanto gli eventi riguardano le operazioni di load e store in memoria per le varie lunghezze di parola (byte, half-word, word, double-word); sono, inoltre, presenti i campi a2/Block-Size ed a3/Block-Address, se `trace_irefs=1`;

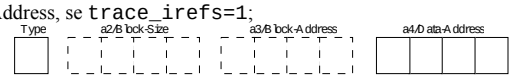


Figura A1–13. Formato SCL per event-type da 0 a 15.

- per valori *da 224 a 255*, gli eventi sono tracciati solo se `trace_irefs=1` ed è stata utilizzata l'opzione `b` del tool di instrumentazione `aug`; gli eventi riguardano i basic-block (ovviamente, se è stato richiesto il tracciamento anche degli eventi del punto precedente, la priorità va ad essi e questo tipo di evento viene tracciato solo nei basic-block privi di riferimenti in memoria); è sempre presente il campo a3/Block-Address mentre il campo a2/Block-Size compare solo se la dimensione del basic-block è maggiore di 31; per dimensioni del basic-block non-superiori a 31 il valore di Block-Size è codificato nei bit 4-0 del campo event-type.

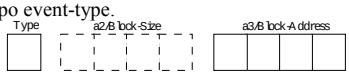


Figura A1–14. Formato SCL per event-type da 224 a 255.

[1] Per ulteriori informazioni contattare S. Goldschmidt, *goldschm@meadow.stanford.edu* oppure S.A. Herrod, *herrod@cs.stanford.edu*.

Per la codifica dettagliata degli eventi si rimanda ai file `reftypes.h` e `synctypes.h`.

```
/************************************************
 *  File:    reftypes.h
 *  Purpose: Define values of the type field
 *           of a trace record.
 *  Author:  Stephen Goldschmidt
 *           (goldschm@mojave.stanford.edu)
 *  Revisions: Steve Herrod
 *           (herrod@cs.stanford.edu
 *
 *  $Revision: 1.1 $
 ************************************************/
#define BLOAD 0   /*  byte load  */
#define HLOAD 1   /*  halfword load  */
#define LOAD 2    /*  word load  */
#define DLOAD 3   /*  double load  */
#define LOADL 4   /*  lwl */
#define LOADR 5   /*  lwr  */

#define BSTORE 8  /*  byte store  */
#define HSTORE 9  /*  halfword store  */
#define STORE 10  /*  word store  */
#define DSTORE 11 /*  double store  */
#define STOREL 12 /*  swl */
#define STORER 13 /*  swr */

#define REF_TYPES 16 /*max number of ref_types*/


/************************************************
 *  File:     tango/synctypes.h
 *  Purpose:  Miscellaneous event codes used
 *            by Tango Lite.
 *
 *  $Revision: 1.1 $
 ************************************************/

/************************************************
 *    0 -  15 are reserved for load/store
 *            events. <tango/reftypes.h>
 *   16 -  63 are used for memory statistics in
 *            the simulator. <memsys.h>
 *   64 - 138 are reserved for miscelaneous
 *            events. <tango/synctypes.h>
 *  140 - 159 are used for sync statistics in
 *            the simulator. <basic_sim.h>
 *  224 - 255 are used for tracing irefs in the
 *            simulator.
 ************************************************/

#define ST_PAUSE_INIT 64
#define ST_CLOCK 65
#define ST_GET_PAGETYPE 66
#define ST_TEST_AND_SET 67
#define ST_MONINIT 68
#define ST_SET_PAGETYPE 69

/*  thread start/end events  */
#define ST_THREAD_DONE 70
#define ST_THREAD_START 71

/*  scalar lock events  */
#define ST_LOCK_ENTER 72
#define ST_LOCK_EXIT 73
#define ST_UNLOCK 74

/*  barrier events  */
#define ST_BAR_ENTER 75
#define ST_BAR_EXIT 77
#define ST_BAR_EXIT_LAST 78

/*  self-scheduled loop macro  */
#define ST_GS_ENTER 79
#define ST_GS_EXIT 80
#define ST_GS_EXIT_LAST 81

#define ST_DELAY_ENTER 82
#define ST_DELAY_EXIT 83

#define ST_CONT 84
#define ST_CONT_LAST 85

#define ST_AS_ENTER 86
#define ST_AS_EXIT 87
#define ST_AS_PROBDONE 88
#define ST_AS_GOTTASK 89
#define ST_AS_WORKWAIT 90
#define ST_AS_PROGDONE 91
#define ST_AS_RESET 92
#define ST_PROGEND 93

#define ST_EVENT_ENTER 94
#define ST_EVENT_EXIT 95
#define ST_PAUSE_ENTER 96
#define ST_PAUSE_EXIT 97

/*  message-passing events  */
#define ST_SEND_ENTER 98
#define ST_SEND_EXIT 99
#define ST_RECEIVE_ENTER 100
#define ST_RECEIVE_EXIT 101
#define ST_HSEND_ENTER 102
#define ST_HSEND_EXIT 103

/*  array-of-locks events  */
#define ST_ALOCK_ENTER 104
#define ST_ALOCK_EXIT 105
#define ST_AULOCK 106

/*  awaiting thread completion  */
#define ST_WAIT_FOR 107

/*  shared memory (de)allocation
    events  */
#define ST_GMALLOC 108
#define ST_GFREE 109

/*  parallel DO events (Fortran only)  */
#define ST_PDO_ENTER 110
#define ST_PDO_EXIT 111
#define ST_PDOEND_ENTER 112
#define ST_PDOEND_EXIT 113

/*  more message-passing events  */
#define ST_HRECEIVE_ENTER 114
#define ST_HRECEIVE_EXIT 115

/*  user-level monitor events  */
#define ST_MENTER 116
#define ST_MEXIT 117
#define ST_CONTINUE 118
#define ST_DELAY 119

/*  initialization events  */
#define ST_LOCK_INIT 120
#define ST_ALOCK_INIT 121
#define ST_BAR_INIT 122
#define ST_GS_INIT 123
#define ST_A_INIT 124

/*  task-delimiting events -
    for performance monitoring  */
#define ST_BEGINTASK 125
#define ST_ENDTASK 126
#define ST_TASK_INIT 127

/*  procedure entry/exit events  */
#define ST_PROC_ENTER 128
#define ST_PROC_EXIT 129

/*  multiprogramming events  */
#define ST_CONTEXT_SWITCH 130
#define ST_FORK 131
#define ST_EXEC 132

/*  simulator control events  */
#define ST_RESET_STATS 133
#define ST_SPIN_ENTER 134
#define ST_SPIN_EXIT 135
#define ST_SPIN_RELEASE 136

#define ST_ENQUEUE 137
#define ST_DEQUEUE 138

#define ST_SEM_INIT 76
#define ST_SEM_OP 139

#define ST_MAX_CODE 192
```

Figura A1–15. Codifica degli eventi del formato fisico SCL (file `reftypes.h` e `synctypes.h`).

Filtro verso il formato logico

Nella sezione '`SCL Filter`' di `ttf`, per le prove effettuate in questa tesi, è stata specificata la seguente traduzione da tipo fisico a tipo logico:

```
[SCL filter]
 LOAD              = R
 STORE             = W
 BBI               = RC
```

Figura A1–16. Sezione relativa al filtro CMU nel tool `ttf; R = read, W = write, RC = Read Code`.

A1.4 Formato di traccia APT

Il formato è stato messo a punto da S. Thomas del gruppo 'Arm S/W Development' della ditta VLSI Technology di San Jose[1]. Consiste nella memorizzazione in formato ASCII delle informazioni rilevate dal debugger `armdb`, durante l'esecuzione di programmi per il processore ARM (da cui deriva l'acronimo Arm Processor Trace).
Le tracce non contengono riferimenti del kernel.

Nomenclatura dei file-traccia

Poiché la macchina tracciata è monoprocessore, non esiste il problema di identificare il numero di processore; per comodità è comunque stato aggiunto il prefisso '0' all'estensione di default.

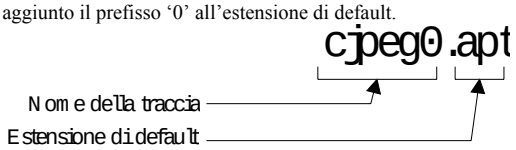


Figura A1–17. Nomenclatura dei file-traccia APT.

Il nome della traccia è (nell'esempio `cjpeg`) è il nome (o una abbreviazione del nome) della applicazione che ha generato il file.

Codifica del formato

Il file è composto da una sequenza di record separati dal carattere `newline` (`\n`); le parole chiave sono separate da uno o più spazi o tabulazioni; le linee di commento iniziano col carattere '#'.

```
# Purpose                           : ARM Debugger Low Level Trace
# Date                      : xxxxx
# Author                            : Stephane Thomas
# Program file              : hellow
#
# +---------------------------- Code
# |        +-------------------- Data
# |        |        +----------- Code / Data  access
# |        |        | +--------- Read / Write access
# |        |        | | +------- Word / Byte  access
# |        |        | | | +----- Seq  / Non seq
# |        |        | | | |  +-- Cycle count from
# |        |        | | | |  |    previous memory access
# |        |        | | | |  |  +------- Operation Mode
# |        |        | | | |  |  |     +- Is Instr exec ?
# v        v        v v v v  v  v     v
# -------- -------- - - - - --  ----- -
  00008008 eb00000c C R W N  0
  0000800c eb00006f C R W S  0
  00008010 ef000011 C R W S  0
# 00008008 eb00000c             USR32 Y : BL 0x8040
  00008040 e1a00000 C R W N  0
  00008044 e04ec00f C R W S  0
```

Figura A1–18. Struttura del record fisico del formato APT.

Le parole chiave di interesse sono la prima (Code), la quale non è altro che il campo *a* del record logico, la terza (Code/Data) e la quarta (Read/Write), che vengono tradotte nel campo *t* del record logico. Il campo *n* del record logico verrà, ovviamente, posto a 0. La combinazione dei campi Code/Data e Read/Write fornisce i quattro valori base del tipo fisico, cioè `DR = Data Read`, `DW = Data Write`, `CR = Code Read`, `CW = Code Write`.

Filtro verso il formato logico

Nella sezione '`APT Filter`' di `ttf`, per le prove effettuate in questa tesi, è stata specificata la seguente traduzione da tipo fisico a tipo logico:

```
[APT filter]
 DR                    = R
 DW                    = W
 CR                    = RC
 CW                    = WC
```

Figura A1–19. Sezione relativa al filtro APT nel tool `ttf; R = read, W = write, RC = Read Code, WC = Write Code`.

[1] Per ulteriori informazioni contattare S. Thomas, *stephane.thomas@vlsi.com*.

A1.5 Formato di traccia `IIP`

Il formato è stato ideato da L. Ricciardi [Ricciardi95] per l'utilizzazione nel simulatore di cache per sistemi multiprocessore - CSIM - da lui sviluppato presso il Dipartimento di Ingegneria della Informazione dell'Università di Pisa (da cui `IIP`). Il formato è stato successivamente esteso da R. Giorgi[1]. Consiste nella memorizzazione in formato ASCII delle informazioni di tipo dell'operazione e dell'indirizzo.

Questo è anche il formato di traccia comune a tutti i tool sviluppati nell'ambito di questa tesi, nonché il formato di uscita di default del tool `ttf`; la tracce possono essere generate sinteticamente utilizzando CSIM oppure, come tutte quelle discusse in questa dissertazione, derivare dallo stadio finale di lavorazione di Trace Factory.

Nomenclatura dei file-traccia

Ogni file-traccia è relativo ad un determinato processore; questo evita di dover memorizzare lo stesso identificatore del processore per ogni riferimento dello stesso file; tale identificatore è contenuto nel nome del file-traccia.

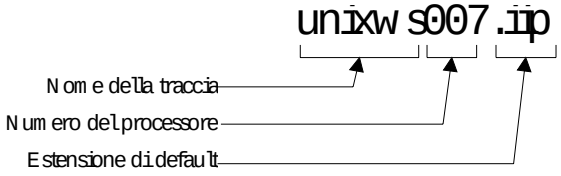


Figura A1–20. Nomenclatura dei file-traccia IIP.

Il nome della traccia è il nome (o una abbreviazione del nome) della applicazione che ha generato il file (nell'esempio `unixws`). Il numero di processore viene scritto utilizzando sempre 3 cifre.

Può essere presente anche un file di sincronizzazione, che indica l'ordine in cui vanno prelevati i riferimenti presenti nei vari file-traccia.

La nomenclatura del suddetto file utilizza come prefisso lo stesso nome di traccia utilizzato nei file `.iip` e come estensione il suffisso `.syn`.

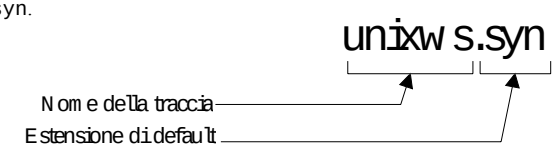


Figura A1–21. Nomenclatura del file di sincronizzazione `.syn`.

Codifica del formato

Il file-traccia è composto da una sequenza di record separati dal carattere `newline` (`\n`); le parole chiave sono separate da uno o più spazi: la prima parola chiave rappresenta il tipo di operazione, la seconda rappresenta l'indirizzo e viene tradotta nel campo *a* del record logico.

```
6 0043AC24
6 0043AC28
1 7FFFBBCC
6 0043AC2C
6 0043AC30
0 10008880
6 0043AC34
6 0043AC38
6 0043AC3C
0 100031C8
```

Figura A1–22. Struttura del record fisico del formato IIP.

[1] Per ulteriori informazioni contattare L. Ricciardi, *ricciard@pisolo.iet.unipi.it* oppure R. Giorgi, *giorgi@vega.iet.unipi.it*.

I tipi di operazione sono riportati nella seguente tabella

UPDR	0	User Private Data Read
UPDW	1	User Private Data Write
USDR	2	User Shared Data Read
USDW	3	User Shared Data Write

KSDR	4	Kernel Shared Data Read
KSDW	5	Kernel Shared Data Write
UPCR	6	User Private Code Read
UPCW	7	User Private Code Write

KPCR	8	Kernel Private Code Read
KPCW	9	Kernel Private Code Write
KSDR	10	Kernel Shared Data Read
KSDW	11	Kernel Shared Data Write

USCR	12	User Shared Code Read
USCW	13	User Shared Code Write
KPCR	14	Kernel Private Code Read
KPCW	15	User Private Code Write

U_DL	16	User Data Lock
K_DL	17	Kernel Data Lock

UNKR	250	Unknown Reference
SYNC	254	Synchronization Reference

Tabella A1–4. Codifica del tipo fisico nel formato IIP.

File di sincronizzazione

Per quanto riguarda il file di sincronizzazione, la struttura fisica è ancora una sequenza di record separati dal carattere `newline` (`\n`); le parole chiave sono separate da uno o più spazi: la prima parola chiave rappresenta il numero di identificazione del processore e la seconda parola chiave rappresenta il numero di riferimenti che deve essere prelevato dal file-traccia relativo a quel processore.

```
0 25
1 24
2 33
3 3
4 423
5 133
6 108533
0 177
6 99
0 73
6 49
0 44
6 44
0 44
6 88
0 44
6 44
```

Figura A1–23. Struttura del record fisico del file di sincronizzazione `.syn`.

Filtro verso il formato logico

Nella sezione '`IIP Filter`' di `ttf`, per le prove effettuate in questa tesi, è stata specificata la seguente traduzione da tipo fisico a tipo logico:

```
[IIP filter]
 00-03    = UPDR,UPDW,USDR,USDW
 04-07    = KPDR,KPDW,UPCR,UPCW
 08-0B    = KPCR,KPCW,KSDR,KSDW
 0C-0F    = USCR,USCW,KSCR,KSCW
 10-13    = U_DL,K_DL,?,?
```

Figura A1–24. Sezione riguardante il filtro IIP nello strumento `ttf`.

In questa appendice viene data una descrizione delle modalità d'uso dei vari tool di Trace Factory.
I tool possono essere richiamati, oltre che dall'ambiente integrato, anche direttamente dall'interfaccia a caratteri di UNIX e possono essere configurati interagendo sugli appositi file-script o makefile.

A2.1 Uso del tool TangoLite

Una volta installato TangoLite, la struttura delle varie directory si presenta come mostrata in figura.

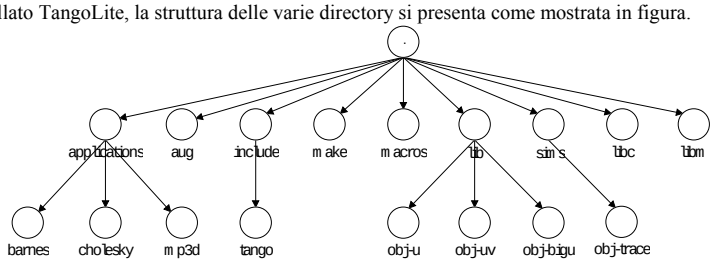


Figura A2–1. Struttura delle directory dopo l'installazione di TangoLite.

La directory capostipite "." può essere scelta a piacere, assegnando alla variabile di ambiente `LITEDIR` il path ad essa relativo. In questa tesi tale variabile è stata assegnata (in C-Shell) con:

```
setenv LITEDIR ~giorgi/SCL/TangoLite
```

Preparazione dell'ambiente di tracciamento

Se non è stato già fatto, occorre compilare il simulatore/tracciatore che si intende utilizzare, il cui `makefile` sta in una delle sottodirectory di `sims`.

I file di supporto al tracciamento, non presenti nella distribuzione 'ufficiale' di TangoLite, sono stati forniti da S. Herrod, (`herrod@cs.stanford.edu`) ed aggiunti nella directory `sims`.

Moltissime sono le opzioni che selezionabili in questo `makefile`; sono state scelte le seguenti:

- supporto al tracciamento su file: `traceout-f.o`;
- supporto alla gestione degli eventi riguardanti i riferimenti a istruzioni, sistema di memoria simulato con eventi ordinati e simulatore combinato ottimizzato per la gestione degli eventi e dei riferimenti: `comb_simmI.o`;
- simulatore della memoria semplificato: `memsim4.o`;
- simulatore del sistema (a livello superiore alla memoria): `memsys.o`.

Questi file vanno poi selezionati nel makefile di instrumentazione della particolare applicazione (di seguito è riportato l'esempio relativo alla instrumentazione del comando di utilità `ls`) nella linea "`SIMOBJS= ...`"; i relativi file oggetto sono posti nella directory `obj-trace`, specificata con la variabile "`SIMDIR=...`".

Se non è stato già fatto, occorre compilare la libreria di instrumentazione (`lib.`); è possibile scegliere, di nuovo, molte opzioni; è stata scelta:

- libreria con clock a 32 bit senza supporto di debugging `libliteu.a` (i cui makefile stanno nella sottodirectory `obj-u` di `lib`).

La libreria viene poi selezionata nel makefile dell'applicazione da instrumentare con la variabile "`LIB=...`".

Si devono, poi, selezionare le opzioni sulla linea di comando del tool di instrumentazione `aug`; nuovamente ci sono una miriade di opzioni per le quali si rimanda all'help della linea di comando; sono state scelte le opzioni:

- `-b`, per instrumentare tutti i basic-block;
- `-c`, per combinare i tipi dei riferimenti e i gestori degli eventi (comb_sim);
- `-n`, per rilevare i num. di linea dei sorgenti a cui è relativo un riferimento;
- `-r`, per instrumentare i riferimenti fuori dallo stack;
- `-f`, per instrumentare i riferimenti all'interno dello stack.

Questo opzioni vengono specificate con la variabile "`AUGFLAGS=...`" del makefile relativo all'applicazione da instrumentare.

Se si desidera che l'instrumentazione ricopra anche le routine della libreria standard del C, si possono sfruttare i sorgenti di queste, poste nella directory `libc` (e le routine della libreria matematica in `libm`) semplicemente elencandoli, di seguito ai file assembler prodotti dal ciclo di instrumentazione, nella variabile "`ASMS=...`" del solito

makefile.

Bisogna, poi, indicare il makefile 'base' da includere nel makefile dell'applicazione da instrumentare; ci sono ancora varie possibilità; è stato scelto il makefile `make/lite-objInst.make` per eseguire la instrumentazione in due passate, basata sull'etichettamento dei file assembler (per consentire il tracciamento dei riferimenti delle istruzioni senza provocare distorsioni della traccia).

Infine, si deve scegliere il set di macro ANL che si desidera adoperare; fra le varie possibilità sono state scelte quelle per la trace-driven-simulation (tds): `macros/c.m4.tds`.

Di seguito è riportato il makefile che è stato preparato per l'instrumentazione del comando di utilità `ls`:

```
#!/bin/make -f

SIMDIR    = $(LITEDIR)/sims/obj-trace

ASFLAGS   = -O2
ASMS      = ls.s ls-ls.s atod.s atof.s atoi.s doscan.s frexp.s ldexp.s modf.s random.s scanf.s strchr.s strcmp.s
AUGFLAGS  = -bcnrf
CFLAGS    = -O2 -DDASH -I../lib -I..
CS                = ls.c ls-ls.c
HS                = ls.h
LDFLAGS   = -O2
LIB       = liteu
MACS      = $(LITEDIR)/macros/c.m4.tds
OBJS      = ls.o ls-ls.o atod.o atof.o atoi.o doscan.o frexp.o ldexp.o modf.o random.o scanf.o strchr.o strcmp.o
SIMOBJS    $(SIMDIR)/comb_simmI.o $(SIMDIR)/memsys.o $(SIMDIR)/memsim4.o $(SIMDIR)/traceout-f.o
LOADLIBES = ../lib/libfu.a
LSS       = ls.lS ls-ls.lS atod.lS atof.lS atoi.lS doscan.lS frexp.lS ldexp.lS modf.lS \
random.lS scanf.lS strchr.lS strcmp.lS
SS        = ls.S ls-ls.S atod.S atof.S atoi.S doscan.S frexp.S ldexp.S modf.S random.S scanf.S strchr.S strcmp.S
TARGET    = ls

include $(LITEDIR)/make/lite-objInst.make

$(ASMS): $(HS)
```

Figura A2–2. Makefile relativo all'instrumentazione del comando `ls`.

Come si può osservare l'instrumentazione in due passate richiede di indicare anche i nomi dei file sorgenti C nella variabile "`CS=`...", i nomi dei file header (variabile "`HS=`..."), i nomi dei file oggetto (variabile "`OBJS=`..."), i nomi dei file tradotti in assembler (variabile "`LS=`..."), i nomi dei file assembler etichettati (variabile "`LSS=`...").
Eventuali opzioni del compilatore vengono indicate nella variabile ("`CFLAGS=`..."); questo è utile per indicare directory ove, eventualmente, risiedono i file header. Le librerie dell'applicazione sono specificate nella variabile "`LOADLIBES=`...".

Infine, le opzioni a tempo di link vengono specificate nella variabile "`LDFLAGS=`..." e il nome della simulazione-eseguibile nella variabile "`TARGET=`..." (questo coincide, normalmente, col nome dell'applicazione stessa).

Produzione della traccia

Una volta fatto il makefile, basta dare `make`. Se tutto è andato bene, si può lanciare l'applicazione instrumentata, avendo prima preparato il file di configurazione del simulatore `config.sim`.

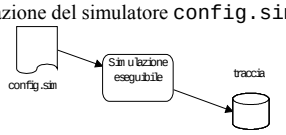


Figura A2–3. La fase di generazione della traccia.

Un esempio del file `config.sim`, ancora relativo alla instrumentazione del comando di utilità `ls` è:

```
1 pe
1 trace_irefs
1 trace_drefs
1 trace_syncs
-1
```

Figura A2–4. File di configurazione della simulazione-eseguibile.

Il file è composto da vari record il cui primo campo è un valore, mentre il secondo è un nome corrispondente ad un flag che si vuole modificare. L'ultimo record è contraddistinto dal valore "`-1`".
Il flag `pe` assegna il numero di processi/processori che si intendono utilizzare nella simulazione; *corrispondentemente verranno generati altrettanti file-traccia*.
I flag `trace_irefs`, `trace_drefs`, `trace_syncs` servono per abilitare il tracciamento rispettivamente dei riferimenti a dati, riferimenti a istruzioni, riferimenti a eventi di sincronizzazione.

Numerosi altri flag possono essere specificati in questo file; inoltre, è possibile intercettarne i valori, in maniera relativamente semplice, anche all'interno dei file preparati "ad hoc" dall'utente.

A2.2 Uso dei tool ttf, kte, skg, msk

Questi quattro tool sono stati progettati avendo in mente di integrarli in un unico ambiente operativo. Per questo motivo si programmano in maniera molto simile e presentano caratteristiche comuni.

In tutti i casi sono presenti almeno tre file:

- file `.ini`, di inizializzazione dello strumento; contiene tutti i dati necessari e sufficienti per ripetere l'esecuzione del tool;
- file `.out`, contiene tutti gli output più significativi per riassumere il comportamento del tool;
- file `.log`, contiene il registro di tutte le operazioni più importanti; è un soprainsieme del file .out.

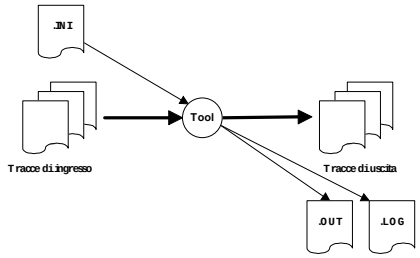


Figura A2–5. File di ingresso e di uscita dei tool ttf, kte, skg, msk.

I file elaborati in ingresso e prodotti in uscita sono, quasi in tutti i casi, tracce.

Preparazione dei file di inizializzazione

La maniera più semplice per creare un file `.ini` consiste nell'utilizzare l'interfaccia grafica `TF`, per configurare le opzioni di interesse a partire da un file di configurazione 'di default' caricato automaticamente alla partenza dell'ambiente operativo.

L'interfaccia grafica consente di visualizzare solo le opzioni principali dei vari tool, in modo da concentrare l'attenzione solo su quei parametri che variano più spesso durante un ciclo di simulazione o produzione di tracce.

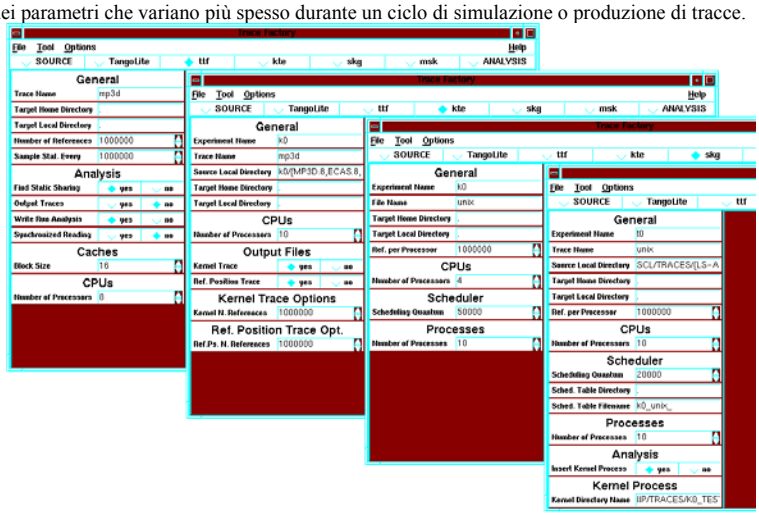


Figura A2–6. I pannelli di controllo con le opzioni principali dei 4 strumenti ttf, kte, skg, msk.

Attivazione degli strumenti

Dall'interfaccia grafica basta selezionare il menu Tool/Applicate. Durante l'esecuzione compare sullo schermo una finestra che mostra il contenuto dei file `.log` o `.out` ed è possibile commutare fra la visualizzazione dell'uno o dell'altro. L'esecuzione viene lanciata in background e viene sospesa se viene chiuso l'ambiente operativo `TF`, a meno che non venga selezionata l'opzione Options/Nohup, che consente di far proseguire l'esecuzione del tool anche dopo la chiusura dell'ambiente operativo e della shell da cui questa è stata lanciata.

È possibile selezionare un file di inizializzazione già esistente scegliendolo dal menu File/Load.
Dall'interfaccia a caratteri, invece, basta dare il comando:

```
ttf -i <nome del file di inizializzazione (.ini)>
```

(per `ttf`; per `kte`, `skg`, `msk` basta fare la stessa cosa utilizzando il nome del tool desiderato). Inoltre, sono presenti dei file script che mostrano come generare i file `.ini`, usando solo poche opzioni a partire da un file-modello (detto

file template, `.tem`): `ttfini`, `kteini`, `skgini`, `mskini`.

File di inizializzazione di ttf, kte, skg, msk

```
;Trace Filter initialization file

[General]                                ;
 Trace Name                 = mp3d               ;  Trace name-radix
 Source Node Name   = r6000.iet.unipi.it  ;; InterNET domain name
 Source Home Directory      = /home/giorgi       ;; User home directory
 Source Local Directory     = CMU/TRACES/MP3D.8  ;; Local directory
 Target Node Name   = localhost           ;; InterNET domain name
 Target Home Directory      = .                  ;  User home directory
 Target Local Directory     = .                  ;  Local directory
 Input Format               = .400m.pak.gz       ;; Compression format
 Output Format      = .iip.gz             ;; Compression format
 Synchron. File Format      = .syn.gz            ;; extension
 Min Address Accepted       = 0x00000000         ;; Hex long integer
 Max Address Accepted       = 0xFFFFFFFF         ;; Hex long integer
 Origin of References       = 1                  ;; Long integer
 Number of References       = 1000000            ;  Long integer
 Sample Stat. Every = 1000000            ;  Long integer

[Analysis]                               ;
 Find Static Sharing        = yes                ;  yes or no
 Output Traces      = no                  ;  yes or no
 Write Run Analysis = yes                 ;  yes or no
 Synchronized Reading       = no                 ;  yes or no

[CPUs]                                   ;
 Address Bit-widht  = 32                  ;; # of actual bits
 Word Lenght                = 32                 ;; 32, 16, 8
 Number of Processors       = 8                  ;  Integer

[Caches]                                 ;
 Block Size                 = 16                 ;  Bytes in a cache block

[Sharing]                                ;;
 Gather User Data   = yes                 ;; yes or no
 Gather Kernel Data = yes                 ;; yes or no
 Gather User Code   = yes                 ;; yes or no
 Gather Kernel Code = yes                 ;; yes or no
 Shared Blocks File = no                  ;; yes or no
 Blocks File Extension      = .16ud.s             ;; extension
 Blocks File Format = .txt                ;; Compression format

[Write Run]                                      ;;
 Distribution Spreading     = 32                 ;; integer
 WRUN Summary File  = no                  ;; yes or no
 WRUN File Extension        = .16ud.w             ;; extension
 WRUN File Format           = .txt                ;; Compression format

[Block Density]                          ;;
 B.Density Summary File     = no                 ;; yes or no
 B.Density File Ext.        = .ndb               ;; extension
 B.Density File Format      = .txt               ;; Compression format

[CMU filter]                                     ;;
 MD,RD,WT,IV                = SYNC,R,W,?          ;; R=Read, W=Write, L=Lock
 RP,RO,LK,UW                = ?,?,L,?             ;; R=Read, W=Write, L=Lock
 FL,FW,RW,UK                = ?,?,?,?             ;; R=Read, W=Write, L=Lock

[MIT filter]                                     ;;
 PTRx00-07                  = R,R,R,R,R,R,R,?     ;;
 PTRx08-0F                  = W,W,W,W,W,?,?,?     ;;
 PTRx10-17                  = W,W,W,W,W,?,?,?     ;;
 PTRx18-1F                  = W,W,W,W,W,W,W,?     ;;
 PTRx20-27                  = ?,?,?,?,?,?,?,?     ;;
 PTRx28-2F                  = ?,?,?,?,?,?,?,?     ;;
 PTRx30-37                  = ?,?,?,?,?,?,RC,?    ;;
 PTRx38-3F                  = ?,?,?,?,?,?,?,?     ;;

[SCL filter]                                     ;;
 LOAD               = R                   ;;
 STORE              = W                   ;;
 BBI                = RC                  ;;

[APT filter]                                     ;;
 DR                         = R                   ;;
 DW                         = W                   ;;
 CR                         = RC                  ;;
 CW                         = WC                  ;;

[IIP filter]                                     ;;
 00-03              = UPDR,UPDW,USDR,USDW ;;
 04-07              = KPDR,KPDW,UPCR,UPCW ;;
 08-0B              = KPCR,KPCW,KSDR,KSDW ;;
 0C-0F              = USCR,USCW,KSCR,KSCW ;;
 10-13              = U_DL,K_DL,?,?       ;;

[Program Defaults]                       ;;
 Log File Name      = mp3d.16ud.log       ;;
 Out File Name      = mp3d.16ud.out       ;;
```

Figura A2–7. File di inizializzazione (`.ini`) di `ttf`.

Ogni file `.ini`, è composto da varie sezioni, il cui nome è indicato fra parentesi quadre; le sezioni sono composte da record del tipo:

`<nome variabile> = <valore>`

I commenti sono preceduti da un “;”. Le variabili seguite da un solo “;” sono variabili principali e sono quelle che in `TF` vengono mostrate nei menu di configurazione semplificati. Le variabili seguite da due “;” sono variabili secondarie ovvero opzioni avanzate.

```
;Kernel Trace Extractor initialization file


[General]                                   ;
 Experiment Name    = k0                    ;  Name of experiment
 Trace Name                 = mp3d                      ;  Name of trace
 Source Node Name   = r6000.iet.unipi.it    ;; InterNET domain name
 Source Home Directory      = /home/giorgi              ;; User home directory
 Source Local Directory     = k0/{MP3D.8,ECAS.8,LOCUS.8,HART.8}     ; Local directory
 Source Format      = .iip.gz               ;; Trace format
 Target Node Name   = localhost                         ;; InterNET domain name
 Target Home Directory      = .                         ;  User home directory
 Target Local Directory     = .                         ;  Local directory

[CPUs]                                      ;
 Number of Processors       = 10                        ;  1..512

[Output Files]                              ;
 Kernel Trace               = yes                       ;  yes or no
 Ref. Position Trace        = yes                       ;  yes or no

[Kernel Trace Options]                                  ;
 Kernel N. References       = 1000000                   ;  Long integer
 Kernel Trace Extension     = .iip.gz                   ;; extension

[Ref. Position Trace Opt.]                  ;
 Ref.Ps. N. References      = 1000000                   ;  Long integer
 Ref.Ps. File Extension     = .rpt                      ;; extension

[Program Defaults]                          ;;
 Log File Name      = k0_mp3d_010.log       ;;
 Out File Name      = k0_mp3d_010.out       ;;
```

Figura A2–8. File di inizializzazione (`.ini`) di `kte`.

```
;Scheduling Generator Initialization File

[General]                                   ;
 Experiment Name    = k0                    ;  Name of experiment
 File Name                  = unix                      ;  Name of file
 Target Node Name   = localhost                         ;; InterNET domain name
 Target Home Directory      = .                         ;  User home directory
 Target Local Directory     = .                         ;  Local directory
 Ref. per Processor = 1000000               ;  Long integer

[CPUs]                                      ;
 Number of Processors       = 4                         ;  1..512

[Scheduler]                                             ;
 Scheduling Quantum = 50000                 ;  Number of references
 Context-sw Algorithm       = linear                    ;; Name of algorithm
 Scheduling Algorithm       = random                    ;; Name of algorithm
 Activation Algorithm       = two-phase                 ;; Name of algorithm
 Random Generator Seed      = 10                        ;; A long integer

[Processes]                                             ;
 Number of Processes        = 10                        ;

[Scheduling Trace]                          ;;
 Sk.Trace Extension = .pst                  ;;

[Program Defaults]                          ;;
 Log File Name      = k0_unix_004.log       ;;
 Out File Name      = k0_unix_004.out       ;;
```

Figura A2–9. File di inizializzazione (`.ini`) di `skg`.

```
;Multiprocessor Scheduler initialization file

[General]                                   ;
 Experiment Name    = t0                    ;  Name of experiment
 Trace Name                 = unix                      ;  Name of trace
 Source Node Name   = r6000.iet.unipi.it    ;; InterNET domain name
 Source Home Directory      = /home/giorgi              ;; User home directory
 Source Local Directory     =    SCL/TRACES/{LS-AR.1,RM.1,LS-LTR.1,CP.1,LS-R.1,LS-AR.1,RM.1,LS-LTR.1,CP.1,LS-R.1,LS-AR.1,RM.1,LS-LTR.1,CP.1,LS-
R.1,LS-AR.1,RM.1,LS-LTR.1,CP.1,LS-R.1}      ; Local directory
 Source Format      = .iip.gz               ;; Trace format
 Target Node Name   = localhost                         ;; InterNET domain name
 Target Home Directory      = .                         ;  User home directory
 Target Local Directory     = .                         ;  Local directory
 Target Format      = .iip                  ;; Trace format
 Origin of References       = 1                         ;; Long integer
 Ref. per Processor = 1000000               ;  Long integer

[CPUs]                                      ;
 Number of Processors       = 10                        ;  1..512

[Scheduler]                                 ;
 Scheduling Quantum = 20000                 ;  Number of references
 Sched. Table Directory     = .                         ;
 Sched. Table Filename      = k0_unix_                  ;
 Sched. Table Format        = .pst                      ;;

[Processes]                                 ;
 Number of Processes        = 10                        ;
 Min Address Accepted       = 0x00000000                ;; Hex long
 Max Address Accepted       = 0x7FFFFFFF                ;; Hex long

[Pager]                                     ;;
 Paging Algorithm   = sequential            ;; Name of algorithm
 Min Address Generated      = 0x00000000                ;; Hex long
 Max Address Generated      = 0x7FFFFFFF                ;; Hex long
 Page Widht                 = 12                        ;; Page width in bits

[Analysis]                                  ;
 Insert Kernel Process      = yes                       ;  yes or no
 Process Reference Map      = no                        ;;  yes or no

[Kernel Process]                            ;
 Kernel Directory Name      = IIP/TRACES/K0_TEST_.1     ;
 Kernel Experiment Name     = k0                        ;; Name of experiment
 Kernel Trace Name  = test                  ;; Name of trace
 Kernel Trace Format        = .iip.gz                   ;;
 Ref. Position Format       = .rpt                      ;;

[Process Ref. Map Options]                  ;;
 Ref.Map File Extension     = .prm                      ;;

[Program Defaults]                          ;;
 Log File Name      = t0_unix_010.log       ;;
 Out File Name      = t0_unix_010.out       ;;
```

Figura A2–10. File di inizializzazione (`.ini`) di `msk`.

Per la spiegazione della sintassi di `kte`, `skg`, `msk` si rimanda al tool `ttf`.

Altri tool

Per quanto riguarda gli altri tool (`SOURCE` e `ANALYSIS`), si rimanda ai file `README` allegati al software.

A3 Implementazione dell'ambiente operativo

In questa appendice verrà dato qualche cenno sugli strumenti utilizzati per sviluppare il software e sulle scelte di progetto.
In particolare, verrà presentato Tcl/Tk, un toolkit di recente sviluppo (che ha avuto notevole successo) e verrà mostrata l'implementazione di alcune strutture dati che hanno consentito l'ottimizzazione delle prestazioni del software nonché, talvolta, la fattibilità stessa del progetto.

A3.1 Ambiente di sviluppo

Il software è stato sviluppato in ambiente UNIX e portato sotto AIX 3.2.5, SunOS 4.1, Ultrix 3.2, Linux 1.2.3, FreeBSD 2.0. L'interfaccia grafica è stata sviluppata con il toolkit Tcl/Tk 3.2/7.1.
Il compilatore preferenziale è stato il `gcc`, (GNU C Compiler) di cui è stata utilizzata l'ultima versione disponibile, la 2.6.3; esso è disponibile su ognuna delle piattaforme UNIX elencate ed ha un alto grado di affidabilità e di rispetto delle specifiche degli standard ANSI.

A3.2 Tcl/Tk

Tcl/Tk (pr. tickle-tee-kay) è un pacchetto software che *costituisce un sistema di programmazione per sviluppare e usare applicazioni con interfaccia grafica* (GUI, Graphical User Interface); esso è basato su Tcl (Tool Command Interface), che non è altro che un semplice linguaggio di scripting[1], grazie al quale è possibile controllare ed estendere applicazioni; Tk è un toolkit che costituisce un'estensione di Tcl per il sistema X-Window [Ousterhout94].

A3.2.1 Tool Command Interface

Tcl in realtà è due cose: un *linguaggio di scripting* e *un interprete per tale linguaggio che può essere facilmente incorporato in una applicazione*. Come linguaggio di scripting [Ousterhout90], Tcl è simile ad altri linguaggi di shell disponibili in ambiente UNIX come la Bourne Shell, la C Shell, la Korn Shell e Perl, i quali forniscono costrutti di programmazione (variabili, controllo di flusso, procedure) per definire file script che, mettendo insieme programmi esistenti, consentono di generare nuove applicazioni. Questo requisito è il motivo principale della scelta di Tcl/Tk per integrare i vari strumenti di cui è costituito Trace Factory. Tcl è un ottimo 'linguaggio sinergico'.

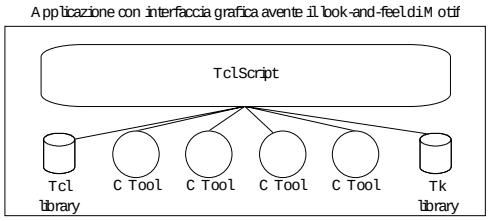


Figura A3–1. Generazione di un'applicazione grafica utilizzando il linguaggio Tcl e il toolkit Tk.

Un'altra possibilità fornita da Tcl è quella di inserire un'interprete di Tcl stesso all'interno dell'applicazione. Questo permette di evitare di inventarsi un nuovo linguaggio per fornire all'utente la programmabilità dell'applicazione. Tcl incoraggia a strutturare l'applicazione come un insieme di tool che aumentano le operazioni primitive fornite dal linguaggio stesso, operazioni che l'utente può comporre a piacimento in un file script. Tcl fornisce anche un metodo per lo scambio di informazioni fra applicazioni basate su Tcl.

A differenza di altri linguaggi analoghi - come Scheme, Elisp, Python - Tcl ha semplici costrutti e somiglia al C. Attraverso l'uso di un interfaccia uniforme, Tcl consente di accedere in modo unico a pacchetti software diversi, senza dover dipendere dalla particolarità di accesso alle funzionalità offerte dal pacchetto. I nuovi pacchetti software potenziano Tcl semplicemente aggiungendo alla classica libreria di funzioni C, fornita dal pacchetto, un'interfaccia Tcl alla libreria stessa.

Esistono diverse estensioni di Tcl che facilitano la risoluzione di determinati problemi: Tcl-DP, per l'accesso ai socket e alla programmazione di rete; Incr-Tcl, un sistema a oggetti; Expect, per la generazione di programmi

[1] Per 'linguaggio di scripting' si intende un linguaggio per scrivere file di tipo script ovvero eseguibili da un interprete di comandi.

interattivi; varie altre estensioni per l'accesso ai database, controllo telefonico, accesso a controllori MIDI. L'estensione più nota di Tcl è però senz'altro Tk.

A3.2.2 X-Window Toolkit

Tk implementa il look-and-feel di Motif™ come specificato dall'Open Software Foundation (OSF) utilizzando le funzioni di base fornite dalla libreria Xt, la libreria intrinseca del sistema X; Motif è uno standard che definisce l'aspetto tridimensionale (look) e molti dettagli del comportamento (feel) degli widget[1] di cui è costituita un'applicazione con interfaccia grafica. Rispetto ad altri toolkit, nei quali è necessario programmare interamente in C, come appunto Xt/Motif, la quantità di informazioni da apprendere per usare Tcl/Tk è notevolmente inferiore e il codice da scrivere è notevolmente meno. Questo è un beneficio tutt'altro che trascurabile, in quanto consente da un lato di ridurre drasticamente i tempi di sviluppo dell'interfaccia grafica e dall'altro di concentrarsi maggiormente sugli aspetti più significativi del comportamento dell'applicazione.
Tcl/Tk consente di definire un'applicazione anche senza scrivere una sola riga di codice C, usando interamente Tcl e la shell a finestre chiamata `wish`. Questo consente di programmare a un livello più alto di quello del C o C++, trascurando molti dei dettagli che un programmatore C deve invece tenere ben presenti. Grazie al fatto che Tcl è un linguaggio interpretato si possono generare ed eseguire file script 'al volo' senza bisogno di ricompilare o far ripartire l'applicazione. È possibile così provare nuove idee, trovare i bug, rifinire i dettagli in modo molto rapido.

`TF`, lo script che lancia l'ambiente operativo di Trace Factory è appunto basato sulla shell grafica `wish`.

A3.2.3 Archivio FTP del Tcl/Tk

Tcl/Tk è disponibile gratuitamente, insieme alle estensioni e alla documentazione, ai seguenti URL (Uniform Resource Location) InterNET:

```
ftp://sunsite.unc.edu/pub/languages/tcl
ftp://iskut.ucs.ubc.ca/pub/X11/tcl
```

A3.3 Implementazione e ottimizzazione del software

Sebbene le tracce contengano centinaia di milioni di riferimenti, pari a oltre 400MB di dati, gli spazi di indirizzamento effettivamente utilizzati sono dell'ordine di qualche MB. Associare delle informazioni ad ogni riferimento comporta, però, il mantenimento di una *lista dei riferimenti*.

Implementare la lista tramite una *coda* sarebbe però altamente inefficiente. La struttura utilizzata è un vettore (per la velocità di accesso alle inf.) paginato su 2 livelli (per ridurre l'occupazione di memoria).

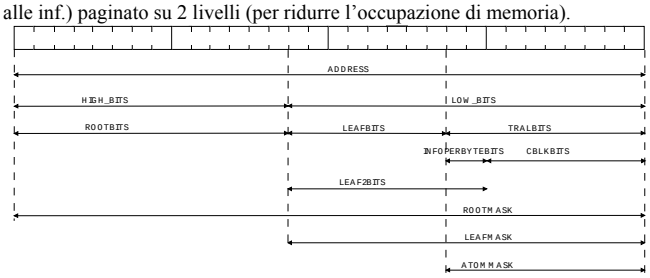


Figura A3–2. Suddivisione dei bit di un indirizzo, per implementare il vettore paginato.

Sostanzialmente i 32 bit di un indirizzo sono stati suddivisi in 3 parti: i bit che servono per indirizzare la pagina *radice* (contenente i puntatori al secondo livello di pagine, dette anche *foglie*); i bit che servono per indirizzare le foglie; i bit che possono essere buttati quando si accede, invece che al singolo indirizzo, ad un intero blocco.

La soluzione si è dimostrata *efficace*.
Sono riportati, qui di seguito, i *tempi di calcolo* e la *massima quantità di memoria allocata* durante l'elaborazione di un milione di riferimenti (per un totale di 8 milioni di riferimenti, avendo 8 file-traccia) delle tracce CMU. Le prove comprendevano la lettura sincronizzata, la raccolta delle statistiche e il calcolo dello write-run. Ogni prova è relativa a una diversa dimensione del blocco di cache.

[1] Uno widget (o anche window, in altri contesti) è l'elemento base di un'interfaccia utente; esempi di widget sono le etichette, i bottoni, i menu a discesa, le scrollbar, le caselle di ingresso per il testo.

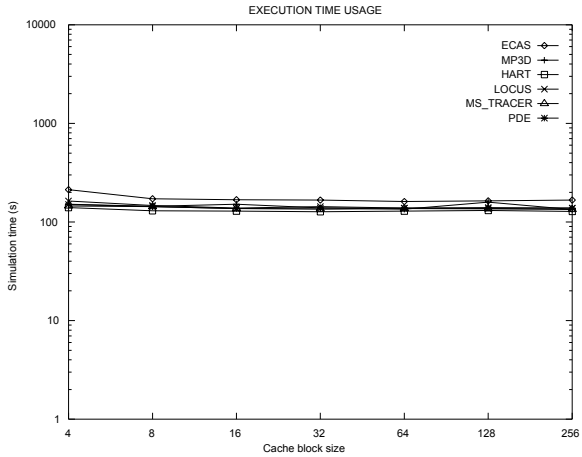


Figura A3–3. Tempo di calcolo al variare della dim. blocco di cache, per il set CMU.

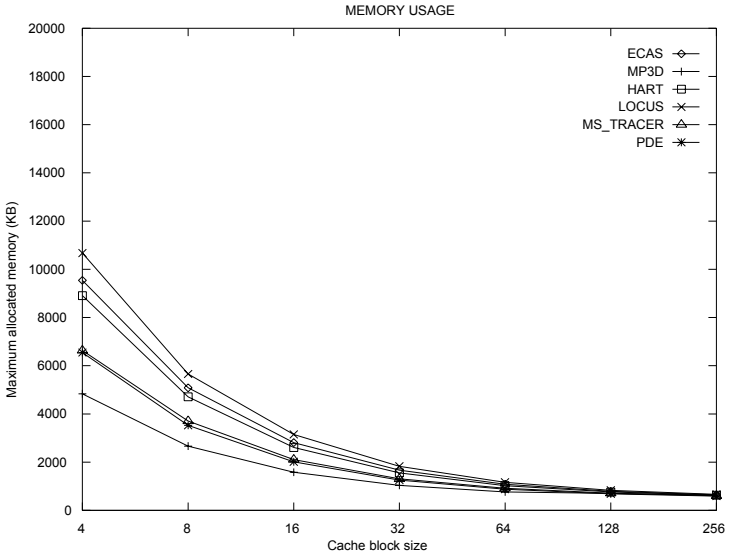


Figura A3–4. Massima memoria allocata al variare della dim. del blocco di cache, per il set CMU.

A4 Schede delle Applicazioni tracciate

Sono qua riportate delle schede che riassumono le principali caratteristiche delle applicazioni e delle relative tracce.

Profilo dell'applicazione

Dopo una *breve descrizione dello scopo* del programma, ne viene descritto il *funzionamento*, *le strutture dati principali*, la *categoria* di cui esso è rappresentativa, *a quale parte dell'esecuzione* è relativa la traccia; per le applicazioni parallele viene descritto il *metodo di parallelizzazione*, sono evidenziati il *tipo di sincronizzazioni* e la *granularità del parallelismo*.

Dati generali della traccia

Altri dati generali, più quantitativi sono: la *fonte dei sorgenti*, con il relativo sito di rete da cui provengono, il *linguaggio di programmazione* e le *linee di codice*, la *linea di comando* utilizzata nell'esecuzione che ha generato la traccia, la *directory* ove risiede la traccia (relativa all'host *vega.iet.unipi.it*), il *numero di riferimenti massimo* che si è voluto analizzare, il *numero di processori* della macchina target di queste tracce.

Statistiche d'accesso

Sono riportati sia in forma tabulare che grafica, le percentuali della *probabilità di lettura, scrittura e fetch* (indicate, rispettivamente, con `P(DR)`, `P(DW)`, `P(CR)`); i dati in tabella sono relativi ad ogni milione di riferimenti di cui è composta la traccia; i dati nel grafico sono invece campionati ogni 250'000 riferimenti. Per le tracce multiprocessore, i dati sono relativi ad una elaborazione non-sincronizzata delle tracce.

Densità incrementale di blocchi da 32B

Viene, infine, riportata la densità incrementale di blocchi distinti nel caso di un'architettura con blocco di cache da 32B. I dati sono riportati in forma analoga al punto precedente e sono stati normalizzati dividendo per il numero di processori, per avere un confronto con i dati monoprocessore. La densità di blocchi distinti indica l'effettiva memoria indirizzata dall'applicazione; piuttosto che analizzare la traccia 'a fette', è più significativo vedere il numero totale di blocchi presenti all'aumentare del numero di riferimenti elaborati (notare che i blocchi distinti nelle varie fette potrebbero non essere indipendenti); la densità è riportata in forma incrementale per evidenziare la *variazione* di blocchi da una fetta all'altra: questo è anche il n. *minimo* di blocchi distinti in quella fetta; la percentuale è relativa ad un valore di confronto: il n. di blocchi distinti che si avrebbero nel caso di *accessi in memoria unicamente consecutivi*.

- Profilo dell'applicazione

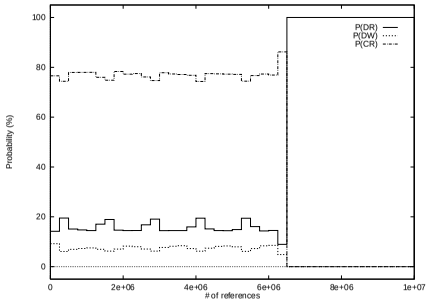
  Compressione di un file contenente un'immagine da un formato non-JPEG a formato JPEG. Vengono supportati vari formati di ingresso: PPM (PBMPLUS color format), PGM (PBMPLUS grey-scale format), BMP, GIF, Targa, RLE. JPEG è uno standard di compressione ideato per immagini del 'mondo reale' (cartoni animati o altre immagini non-realistiche non sono compresse altrettanto bene). JPEG comprime irreversibilmente, nel senso che l'immagine di uscita non è necessariamente identica all'immagine di ingresso; per immagini del mondo reale non si hanno però apprezzabili variazioni. Il programma utilizza un algoritmo di compressione chiamato DCT (per ulteriori informazioni vedere G. K. Wallace, "The JPEG Still Picture Compression Standard", Communications of ACM, April 1991, vol. 34, no.4, pp. 30-44). Il programma è rappresentativo delle applicazioni monoprocesso e fa parte delle utilità oggi normalmente disponibili nei sistemi UNIX più aggiornati. La traccia è stata prelevata a partire dall'inizio del programma fino al desiderato numero di riferimenti; è stata generata utilizzando un'apposita opzione del debugger 'armdb' fornito con l'ambiente di sviluppo per il processore ARM; il parametro 'int' significa che l'algoritmo DCT lavora solo sugli interi; l'immagine è fornita insieme ai sorgenti del programma ed ha una dimensione di circa 1KB.

- Dati generali della traccia

Fonte dei sorgenti:	Independent JPEG Group, ftp://ftp.uu.net/graphics/jpeg
Linguaggio e linee di codice:	C, 20000 linee
Linea di comando:	cjpeg -dct int -outfile testout.jpg testimg.ppm
Directory della traccia:	/home/giorgi/APT/TRACES/CJPEG.1
# di riferimenti per processore:	10000000
# di processori:	1

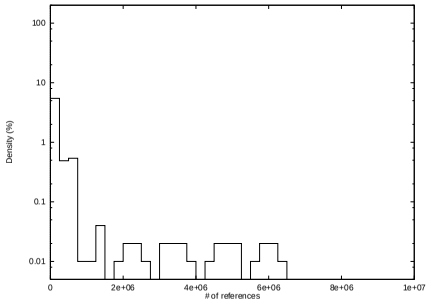
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	15.89	7.39	76.72
2000000	16.28	6.95	76.77
3000000	16.22	7.40	76.39
4000000	14.85	7.89	77.26
5000000	15.88	7.52	76.59
6000000	16.18	7.43	76.39
7000000	55.88	3.35	40.77
8000000	100.00	0.00	0.00
9000000	100.00	0.00	0.00
10000000	100.00	0.00	0.00



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2029	1.62
2000000	19	0.02
3000000	16	0.01
4000000	20	0.02
5000000	14	0.01
6000000	15	0.01
7000000	9	0.01
8000000	0	0.00
9000000	0	0.00
10000000	0	0.00



- Profilo dell'applicazione

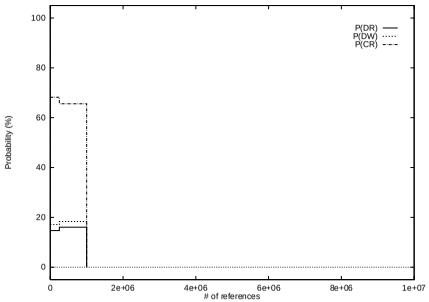
  Benchmark di tipo CPU-intensive. Consiste di un misto di circa 100 istruzioni e tipi di dati del linguaggio ad alto livello, disponibili nelle applicazioni in cui le operazioni floating-point non sono utilizzate. I comandi Dhrystone sono bilanciati fra le tre classi: comandi, dati, riferimenti locali; non ci sono chiamate al sistema operativo e non si fa uso di routine di libreria. Dhrystone fornisce quindi una misura delle prestazioni sugli interi nei moderni processori [Hwang93]. La traccia è stata prelevata utilizzando un'apposita opzione del debugger 'armdb' fornito con l'ambiente di sviluppo per il processore ARM; la traccia è relativa all'inizio del programma.

- Dati generali della traccia

Fonte dei sorgenti:	Public Domain Software
Linguaggio e linee di codice:	C, 4000 linee
Linea di comando:	dhrystone
Directory della traccia:	/home/giorgi/APT/TRACES/DHRY.1
# di riferimenti per processore:	10000000
# di processori:	1

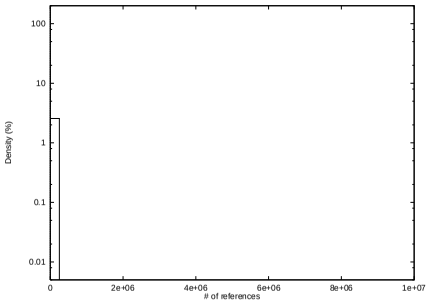
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	15.72	18.04	66.24



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	799	0.64



- Profilo dell'applicazione

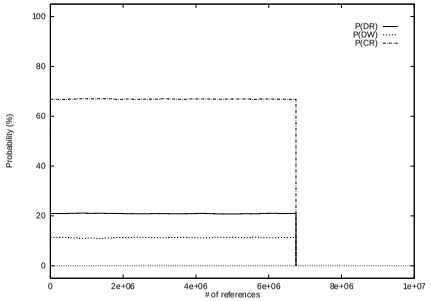
  Simulatore di architettura di computer. È utilizzato alla Encore per valutare nuove architetture; ogni blocco fondamentale viene modellato singolarmente e la comunicazione fra le parti viene fatta attraverso una coda di eventi che avvengono a istanti discreti. Le strutture simulate sono del tipo PMS (Processor-Memory-Switch) ma si può scendere al livello di dettaglio desiderato. Durante un time-slice la simulazione di ogni unità può procedere in parallelo; un processo è tipicamente un'istanza di un componente del sistema, sia esso a basso o ad alto livello. I processi comunicano utilizzando dei token inseriti o prelevati da buffer che connettono i processi. La sincronizzazione avviene attraverso send-bloccante e receive-bloccante sui buffer di comunicazione. Il programma è rappresentativo di simulazioni a eventi su scala temporale dettagliata; la granularità è fine. La memoria viene allocata staticamente e assegnata dinamicamente. La traccia è stata prelevata da un Encore Multimax [Vashow93], 30 secondi dopo il termine della fase di inizializzazione e mentre la macchina si trova in modalità single-user.

- Dati generali della traccia

Fonte dei sorgenti:	Encore (autore A.W. Wilson, DWilson@corp.adaptec.com)
Linguaggio e linee di codice:	C, 25000 linee
Linea di comando:	Programma lanciato dalla dir. /usr/ecas/testfiles
Directory della traccia:	/home/giorgi/CMU/TRACES/ECAS.8
# di riferimenti per processore:	10000000
# di processori:	8

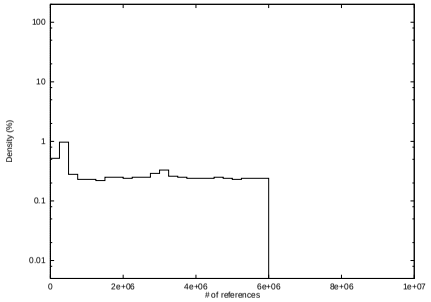
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	21.03	11.25	66.86
2000000	21.03	11.16	66.96
3000000	20.89	11.37	66.86
4000000	20.89	11.30	66.92
5000000	20.89	11.32	66.91
6000000	20.88	11.34	66.89



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	626	0.50
2000000	297	0.24
3000000	319	0.26
4000000	338	0.27
5000000	301	0.24
6000000	296	0.24



- Profilo dell'applicazione

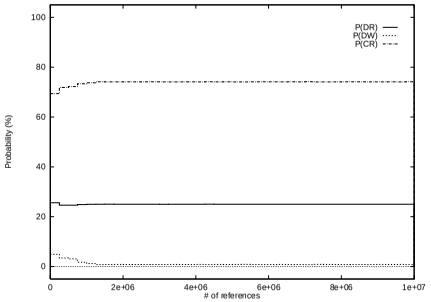
  Hard-Real-Time benchmark. Serve per caratterizzare le capacità real-time di un sistema; 'hard' significa che i processi devono tutti rispettare la propria deadline, pena il fallimento dell'intero sistema. L'implementazione è progettata per sistemi embedded e, in effetti, il Multimax, su cui è stato fatto girare, fallisce alcuni test, non essendo una macchina embedded e tantomeno hard-real-time. Inizialmente vengono creati 5 task periodici indipendenti; ogni task ha una frequenza, un corpo, una priorità e una deadline; le frequenze sono armoniche fra loro. Il corpo dei task è costituito da 1024 istruzioni (1K) Whetstone (un'altro benchmark da cui questo deriva). Ci sono 4 serie di esperimenti, ognuno dei quali stressa il sistema in maniera diversa. Questa traccia è stata prelevata mentre la macchina fa girare il quarto esperimento, in cui vengono creati nuovi task con la stessa frequenza e con corpo costituito da 1K Whetstone. Non ci sono dati condivisi. La sincronizzazione è costituita dal semplice process switching. Granularità non definita. La traccia è stata prelevata a partire da quando la macchina ha raggiunto il 20% di utilizzazione [Vashow93].

- Dati generali della traccia

Fonte dei sorgenti:	SW Eng. Institute, CMU (autore N. Weiderman)
Linguaggio e linee di codice:	Ada, 2500 linee
Linea di comando:	hs.ex4
Directory della traccia:	/home/giorgi/CMU/TRACES/HART.8
# di riferimenti per processore:	10000000
# di processori:	8

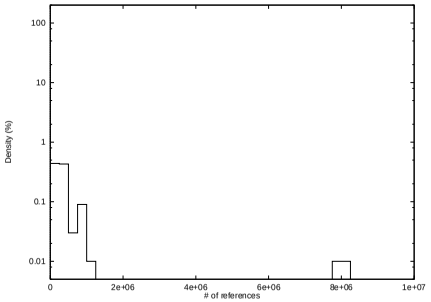
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	24.94	3.29	71.70
2000000	25.01	0.91	74.03
3000000	25.01	0.83	74.12
4000000	25.01	0.84	74.11
5000000	25.02	0.83	74.12
6000000	25.01	0.85	74.10
7000000	25.01	0.83	74.12
8000000	25.01	0.84	74.11
9000000	25.01	0.83	74.12
10000000	25.01	0.84	74.12



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	310	0.25
2000000	4	0.00
3000000	0	0.00
4000000	0	0.00
5000000	1	0.00
6000000	0	0.00
7000000	0	0.00
8000000	3	0.00
9000000	2	0.00
10000000	0	0.00



- Profilo dell'applicazione

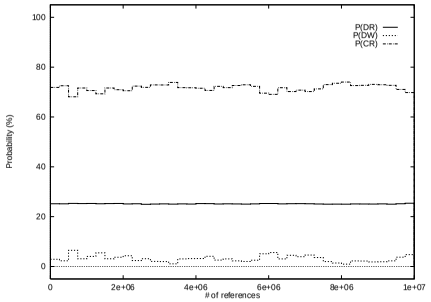
  VLSI standard cell router di qualità commerciale. LOCUS è uno dei programmi della suite di SPLASH (Stanford Parallel Applications for Shared Memory). Il programma cerca di far passare i fili di collegamento attraverso regioni (denominate 'routing cell') che contengono pochi fili (algoritmo iterativo). La principale struttura di Locus è il vettore di costo, che tiene conto dei fili che passano in ciascuna routing cell. I dati condivisi includono, oltre ai precedenti, le posizioni dei pin che, dopo l'inizializzazione, vengono soltanto lette e il cammino dei fili, ma questi dati sono acceduti meno frequentemente del vettore di costo. Il parallelismo può essere usato sull'intero filo, su un suo segmento o sul calcolo del costo dei vari percorsi per ogni segmento. Il file Primary2.grin è fornito col programma e specifica uno scheduling 'geografico': il circuito è suddiviso in varie regioni, ad ognuna delle quali è associata un coda di processi; i processori prelevano task da una coda preferenziale ma, se questa è vuota esplorano altre code; ciò migliora la località e le prestazioni delle cache. L'allocazione dei dati è semi-statica; la granularità è grossa; la sincronizzazione avviene per mezzo di barriere e code. La traccia è stata prelevata da un Encore Multimax [Vashow93], durante la prima iterazione, dopo la creazione di tutti i processi e per la durata di circa 1 minuto, mentre la macchina è in modalità single-user.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (autore J.Rose), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 6400 linee
Linea di comando:	locus inputs/Primary2.grin ParameterFile output
Directory della traccia:	/home/giorgi/CMU/TRACES/LOCUS.8
# di riferimenti per processore:	10000000
# di processori:	8

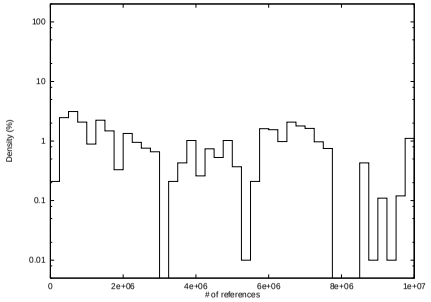
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	25.22	3.70	71.03
2000000	25.25	4.08	70.62
3000000	25.07	2.97	71.92
4000000	25.09	2.33	72.55
5000000	25.18	3.23	71.56
6000000	25.13	2.98	71.84
7000000	25.21	4.28	70.46
8000000	25.08	2.90	71.99
9000000	25.07	1.80	73.10
10000000	25.18	3.17	71.62



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2467	1.97
2000000	1545	1.24
3000000	1161	0.93
4000000	518	0.41
5000000	798	0.64
6000000	689	0.55
7000000	2001	1.60
8000000	1049	0.84
9000000	137	0.11
10000000	419	0.34



- Profilo dell'applicazione

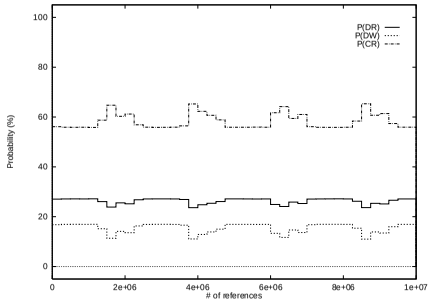
  Simulazione del flusso in fluidi rarefatti. MP3D è uno dei programmi della suite di SPLASH (Stanford Parallel Applications for Shared Memory). È un simulatore di particelle tridimensionali che viene usato per studiare le pressioni e i profili di temperatura degli aeroveivoli nell'alta atmosfera mentre viaggiano a velocità ultrasoniche. Il programma usa i metodi Monte Carlo per effettuare un'analisi statistica della traiettoria che sarà raggiunta nel fluido di natura discreta; la grande quantità di dati acceduti è funzione del numero di molecole simulate. Ad ogni time-step viene valutata la posizione e la velocità di ogni molecola, la quale viene spostata in accordo ad esse, alle altre molecole e al velivolo. Il programma ha granularità grossa; si presta bene ad essere parallelizzato perché ogni molecola può essere trattata indipendentemente, ad ogni time-step. Il principale tipo di sincronizzazione è la barriera fra time-step. La geometria del velivolo riguarda il 'flat sheet problem'. Questa traccia riguarda il caso di 100000 molecole, su 8 processori e per 20 time-steps. La traccia è stata prelevata da un Encore Multimax [Vashow93], dopo poche iterazioni e per la durata di 10.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J.D. McDonald), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 1500 linee
Linea di comando:	mp3d 100000 8 < run.in (20 q e)
Directory della traccia:	/home/giorgi/CMU/TRACES/MP3D.8
# di riferimenti per processore:	10000000
# di processori:	8

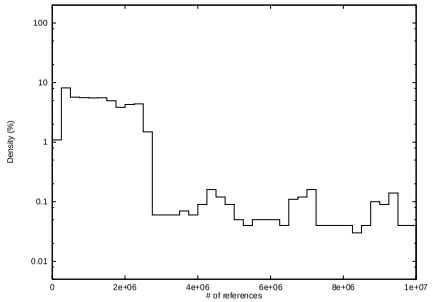
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	27.12	16.91	55.95
2000000	25.67	14.41	59.90
3000000	26.57	15.94	57.47
4000000	26.21	15.39	58.38
5000000	25.85	14.69	59.44
6000000	27.13	16.93	55.91
7000000	25.04	13.34	61.59
8000000	27.14	16.91	55.93
9000000	25.59	14.31	60.08
10000000	26.50	15.83	57.66



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	6411	5.13
2000000	6223	4.98
3000000	3197	2.56
4000000	76	0.06
5000000	144	0.12
6000000	57	0.05
7000000	99	0.08
8000000	87	0.07
9000000	68	0.05
10000000	96	0.08



- Profilo dell'applicazione

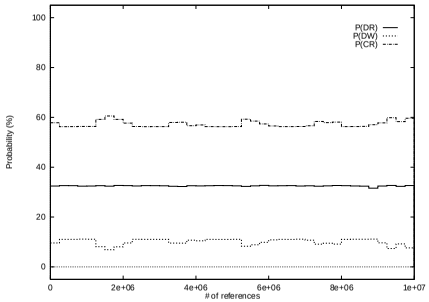
  Ray tracing. Il programma effettua il ray-tracing di un insieme di sfere poste al di sopra di una superficie avente un certo pattern. Lo schermo è suddiviso in gruppi di linee orizzontali. Il numero di iterazioni viene indicato sulla linea di comando (nella traccia in questione è 50). La suddivisione avviene staticamente e ogni processore è responsabile della elaborazione di un'unica zona dello schermo. Questo metodo ha il vantaggio di avere i dati strettamente associati con un unico processore, migliorando così le prestazioni delle cache. Non c'è alcuna variabile di particolare importanza; lo svantaggio principale è dovuto al fatto che non tutti i processori finiscono la loro elaborazione contemporaneamente, lasciando qualche processore in stato di attesa attiva. La granularità del programma è media; le sincronizzazioni usate sono le barriere. La traccia è stata prelevata dall'Encore Multimax [Vashow93] mentre si trovava in modalità multi-user (per qualche motivo sconosciuto la macchina non si comportava correttamente in modalità single-user); il primo parametro della linea di comando indica che sono utilizzati 8 processori; le 50 iterazioni richiedono circa 6 minuti mentre la traccia è relativa ad un solo minuto di esecuzione.

- Dati generali della traccia

Fonte dei sorgenti:	Frittz Graphics, CMU (autore M. Rao)
Linguaggio e linee di codice:	C, 1100 linee
Linea di comando:	tracer -p8 -n50
Directory della traccia:	/home/giorgi/CMU/TRACES/MS_TRACER.8
# di riferimenti per processore:	10000000
# di processori:	8

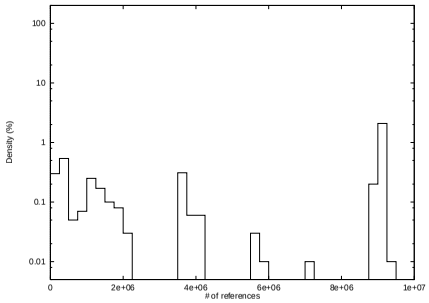
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	32.51	10.71	56.68
2000000	32.53	8.54	58.82
3000000	32.57	10.70	56.64
4000000	32.43	10.21	57.24
5000000	32.57	10.90	56.44
6000000	32.50	9.52	57.87
7000000	32.54	11.01	56.36
8000000	32.52	9.62	57.74
9000000	32.26	11.10	56.51
10000000	32.52	8.47	58.90



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	302	0.24
2000000	189	0.15
3000000	12	0.01
4000000	116	0.09
5000000	19	0.02
6000000	13	0.01
7000000	0	0.00
8000000	3	0.00
9000000	62	0.05
10000000	658	0.53



- Profilo dell'applicazione

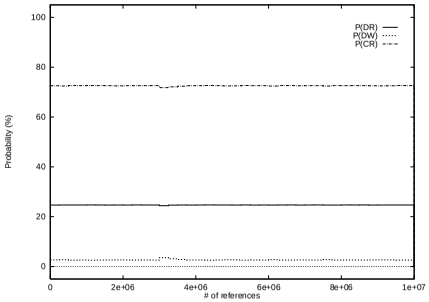
  Il programma usa una griglia con un set di condizioni iniziali e, facendo avanzare il tempo, effettua ripetutamente la media di ciascun elemento delle griglia con i suoi vicini; il fenomeno misurato è la variazione di temperatura nel tempo. La superficie è suddivisa in strisce, ognuna delle quali è completamente elaborata da un solo processo; poiché ogni punto può essere scritto da un unico processo non c'è alcun bisogno di sincronizzazione, salvo al termine del programma. L'allocazione dei dati è statica e i dati sono scritti da un solo processore; la granularità non dipende dai lock ma dalle frequenti comunicazioni; il programma è completamente asincrono salvo per la barriera finale. La traccia riguarda solo il kernel ed è prelevata dall'Encore Multimax [Vashow93] mentre si trova in modalità multi-user (perchè il programma non funziona correttamente in modalità single-user); in ogni caso durante l'esecuzione questo è l'unico programma utente. I parametri 8 e 30 indicano rispettivamente il numero di processori e un fattore di scala per produrre un'esecuzione adeguatamente lunga. Il programma dura circa 30 minuti e la traccia è relativa all' intervallo di tempo fra 20 e 21 minuti dall'inizio.

- Dati generali della traccia

Fonte dei sorgenti:	A. Wilson, DWilson@corp.adaptec.com
Linguaggio e linee di codice:	C, 300 linee
Linea di comando:	dispoff 8 30
Directory della traccia:	/home/giorgi/CMU/TRACES/PDE.8
# di riferimenti per processore:	10000000
# di processori:	8

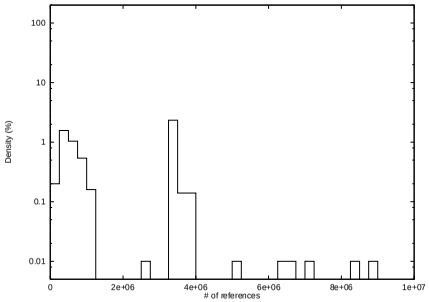
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	24.70	2.66	72.55
2000000	24.71	2.62	72.58
3000000	24.70	2.62	72.58
4000000	24.61	3.04	72.21
5000000	24.71	2.63	72.57
6000000	24.70	2.62	72.58
7000000	24.70	2.66	72.56
8000000	24.70	2.66	72.55
9000000	24.71	2.63	72.58
10000000	24.71	2.64	72.57



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1047	0.84
2000000	49	0.04
3000000	3	0.00
4000000	820	0.66
5000000	1	0.00
6000000	1	0.00
7000000	4	0.00
8000000	3	0.00
9000000	4	0.00
10000000	1	0.00



- Profilo dell'applicazione

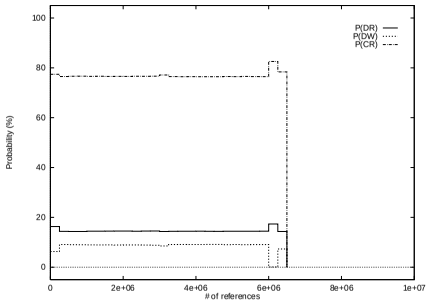
  Elaborazione di file di testo. AWK è un programma di utilità che interpreta uno speciale linguaggio di programmazione (molto simile al C), grazie al quale si possono specificare varie operazioni da compiere sui record del file di ingresso ogni volta che viene incontrato un determinato pattern. l file di test.awk è uno dei file degli esempi allegati ai sorgenti del software. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU awk (D. Barlow Close), ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 15000 linee
Linea di comando:	awk -f test.awk
Directory della traccia:	/home/giorgi/SCL/TRACES/AWK.1
# di riferimenti per processore:	10000000
# di processori:	1

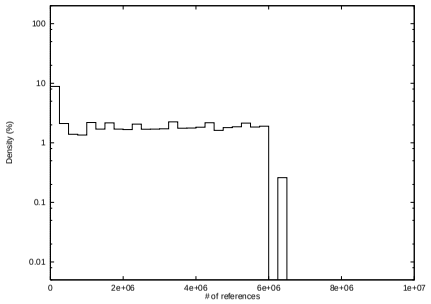
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	14.84	8.35	76.81
2000000	14.47	8.94	76.59
3000000	14.50	8.89	76.61
4000000	14.41	8.96	76.63
5000000	14.43	9.11	76.46
6000000	14.44	9.06	76.50



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	4279	3.42
2000000	2421	1.94
3000000	2227	1.78
4000000	2342	1.87
5000000	2318	1.85
6000000	2418	1.93



- Profilo dell'applicazione

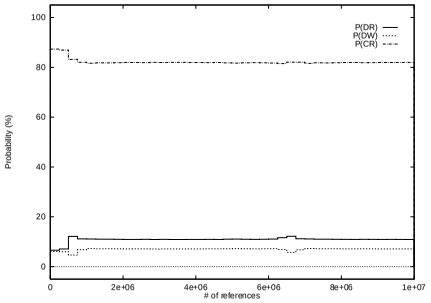
  Simulazione dell'evoluzione di galassie. BARNES è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). L'applicazione viene usata per studiare il classico problema gravitazionale di N corpi interagenti fra loro. L'algoritmo è basato su una rappresentazione gerarchica di tipo octree dello spazio tridimensionale suddiviso in celle; l'algoritmo è, quindi, adattivo e si estende nei dettagli ove la densità di particelle è più alta. Questa struttura gerarchica (sostanzialmente un'albero) rappresenta la principale struttura dati dell'applicazione; sono poi presenti degli array di puntatori ai corpi e di puntatori alle celle; ogni processore possiede un uguale pezzo di questi vettori di puntatori e ciò determina quali corpi e celle vengono elaborati da un dato processore. I dati utilizzati dal programma aumentano linearmente con il numero di corpi. La struttura del programma è iterativa; ogni fase di un time-step può essere eseguita in parallelo. La granularità del problema è grossa; ci sono fasi in cui c'è scarso bisogno di sincronizzazione; quando questa occorre, si utilizzano le barriere oppure, per proteggere dati globali la mutua esclusione tramite lock. La traccia relativa al caso di 120 corpi, seme 123, time-step di 0.025, 0.05 e 0.6 di accuratezza, 2.0 * 120 celle, 0.075 di tempo di simulazione (4 time-step), output dopo 0.10, 1 processore, no-debugging.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J. E. Barnes), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2750 linee
Linea di comando:	barnes < input.barnes (120 123 0.025 0.05 0.6 2.0 0.075 0.10 1 0)
Directory della traccia:	/home/giorgi/SCL/TRACES/BARNES.1
# di riferimenti per processore:	10000000
# di processori:	1

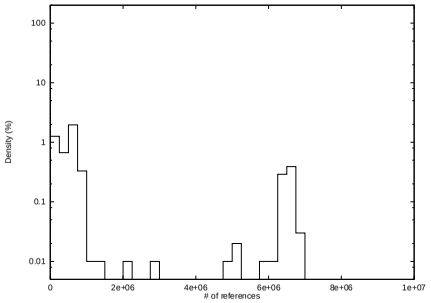
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	9.25	5.91	84.84
2000000	11.04	7.20	81.77
3000000	10.94	7.13	81.93
4000000	10.93	7.12	81.96
5000000	10.96	7.16	81.87
6000000	11.02	7.20	81.79
7000000	11.52	6.66	81.82
8000000	11.04	7.22	81.74
9000000	10.96	7.15	81.89
10000000	10.93	7.12	81.94



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1315	1.05
2000000	5	0.00
3000000	7	0.01
4000000	2	0.00
5000000	5	0.00
6000000	9	0.01
7000000	223	0.18
8000000	0	0.00
9000000	0	0.00
10000000	0	0.00



- Profilo dell'applicazione

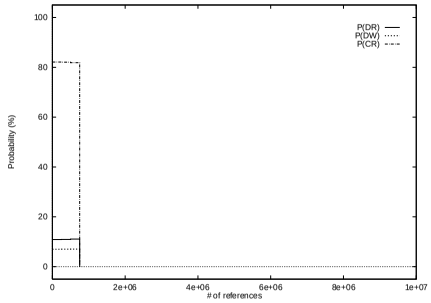
  Simulazione dell'evoluzione di galassie. BARNES è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). L'applicazione viene usata per studiare il classico problema gravitazionale di N corpi interagenti fra loro. L'algoritmo è basato su una rappresentazione gerarchica di tipo octree dello spazio tridimensionale suddiviso in celle; l'algoritmo è, quindi, adattivo e si estende nei dettagli ove la densità di particelle è più alta. Questa struttura gerarchica (sostanzialmente un'albero) rappresenta la principale struttura dati dell'applicazione; sono poi presenti degli array di puntatori ai corpi e di puntatori alle celle; ogni processore possiede un uguale pezzo di questi vettori di puntatori e ciò determina quali corpi e celle vengono elaborati da un dato processore. I dati utilizzati dal programma aumentano linearmente con il numero di corpi. La struttura del programma è iterativa; ogni fase di un time-step può essere eseguita in parallelo. La granularità del problema è grossa; ci sono fasi in cui c'è scarso bisogno di sincronizzazione; quando questa occorre, si utilizzano le barriere oppure, per proteggere dati globali la mutua esclusione tramite lock. La traccia relativa al caso di 120 corpi, seme 123, time-step di 0.025, 0.05 e 0.6 di accuratezza, 2.0 * 120 celle, 0.075 di tempo di simulazione (4 time-step), output dopo 0.10, 24 processori, no-debugging.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J. E. Barnes), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2750 linee
Linea di comando:	barnes < input.barnes (120 123 0.025 0.05 0.6 2.0 0.075 0.10 24 0)
Directory della traccia:	/home/giorgi/SCL/TRACES/BARNES.24
# di riferimenti per processore:	10000000
# di processori:	24

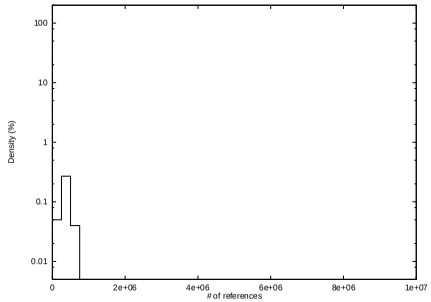
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*



- Profilo dell'applicazione

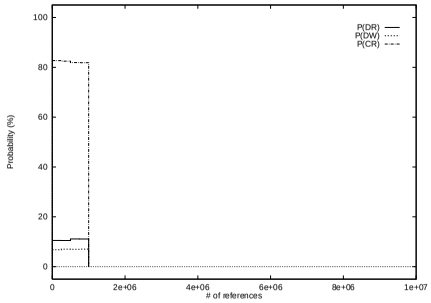
  Simulazione dell'evoluzione di galassie. BARNES è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). L'applicazione viene usata per studiare il classico problema gravitazionale di N corpi interagenti fra loro. L'algoritmo è basato su una rappresentazione gerarchica di tipo octree dello spazio tridimensionale suddiviso in celle; l'algoritmo è, quindi, adattivo e si estende nei dettagli ove la densità di particelle è più alta. Questa struttura gerarchica (sostanzialmente un'albero) rappresenta la principale struttura dati dell'applicazione; sono poi presenti degli array di puntatori ai corpi e di puntatori alle celle; ogni processore possiede un uguale pezzo di questi vettori di puntatori e ciò determina quali corpi e celle vengono elaborati da un dato processore. I dati utilizzati dal programma aumentano linearmente con il numero di corpi. La struttura del programma è iterativa; ogni fase di un time-step può essere eseguita in parallelo. La granularità del problema è grossa; ci sono fasi in cui c'è scarso bisogno di sincronizzazione; quando questa occorre, si utilizzano le barriere oppure, per proteggere dati globali la mutua esclusione tramite lock. La traccia relativa al caso di 120 corpi, seme 123, time-step di 0.025, 0.05 e 0.6 di accuratezza, 2.0 * 120 celle, 0.075 di tempo di simulazione (4 time-step), output dopo 0.10, 8 processori, no-debugging.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J. E. Barnes), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2750 linee
Linea di comando:	barnes < input.barnes (120 123 0.025 0.05 0.6 2.0 0.075 0.10 8 0)
Directory della traccia:	/home/giorgi/SCL/TRACES/BARNES.8
# di riferimenti per processore:	10000000
# di processori:	8

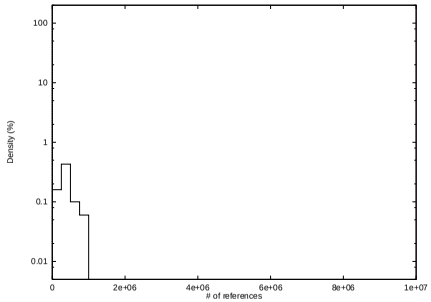
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	10.82	6.96	82.22



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	234	0.19



- Profilo dell'applicazione

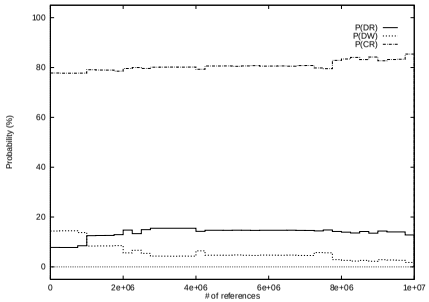
  Fattorizzazione di una matrice in forma sparsa. CHOLESKY è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). La matrice definita positiva A viene fattorizzata come $A = LL^T$, dove L è una matrice triangolare inferiore; matrici di questo tipo si incontrano in problemi di analisi strutturale, simulazione di processo, reti per l'energia elettrica. L'algoritmo effettua tre passi: ordinamento, fattorizzazione simbolica, fattorizzazione numerica. Nel presente programma non viene fatto il passo di ordinamento. La principale struttura dati è costituita dalla rappresentazione della matrice sparsa stessa. L'operazione primaria è la modifica supernodale di una colonna in cui una colonna viene modificata da tutte le colonne di un supernodo (ovvero un set di colonne con elementi diversi da zero circa nelle stesse posizioni); questa operazione coincide con la granularità. La sincronizzazione è ottenuta con dei lock, quando più processori tentano di accedere simultaneamente ad un supernodo in corso di modifica o quando tentano di prelevare supernodi momentaneamente accodati in una coda globale. Questa traccia è stata generata per 1 processore e utilizzando un file di test standard.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (E. Rothberg), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	cholesky -p1 Inputs/bcsstk14.O
Directory della traccia:	/home/giorgi/SCL/TRACES/CHOLESKY.1
# di riferimenti per processore:	10000000
# di processori:	1

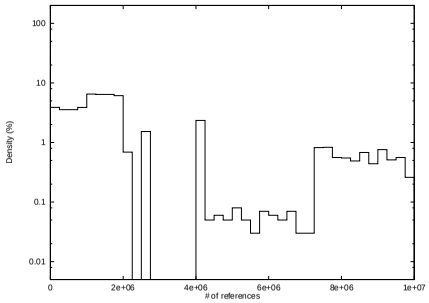
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	7.97	14.28	77.75
2000000	12.67	8.43	78.90
3000000	14.64	5.51	79.85
4000000	15.52	4.32	80.16
5000000	14.59	5.12	80.29
6000000	14.70	4.70	80.60
7000000	14.69	4.68	80.63
8000000	14.53	4.72	80.76
9000000	13.81	2.46	83.72
10000000	13.82	2.52	83.66



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	4655	3.72
2000000	7951	6.36
3000000	693	0.55
4000000	0	0.00
5000000	784	0.63
6000000	72	0.06
7000000	65	0.05
8000000	698	0.56
9000000	677	0.54
10000000	652	0.52



- Profilo dell'applicazione

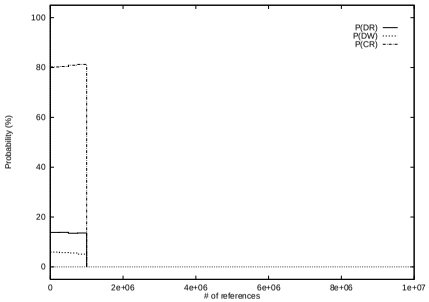
  Fattorizzazione di una matrice in forma sparsa. CHOLESKY è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). La matrice definita positiva A viene fattorizzata come $A = LL^T$, dove L è una matrice triangolare inferiore; matrici di questo tipo si incontrano in problemi di analisi strutturale, simulazione di processo, reti per l'energia elettrica. L'algoritmo effettua tre passi: ordinamento, fattorizzazione simbolica, fattorizzazione numerica. Nel presente programma non viene fatto il passo di ordinamento. La principale struttura dati è costituita dalla rappresentazione della matrice sparsa stessa. L'operazione primaria è la modifica supernodale di una colonna in cui una colonna viene modificata da tutte le colonne di un supernodo (ovvero un set di colonne con elementi diversi da zero circa nelle stesse posizioni); questa operazione coincide con la granularità. La sincronizzazione è ottenuta con dei lock, quando più processori tentano di accedere simultaneamente ad un supernodo in corso di modifica o quando tentano di prelevare supernodi momentaneamente accodati in una coda globale. Questa traccia è stata generata per 24 processori e utilizzando un file di test standard.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (E. Rothberg), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	cholesky -p24 Inputs/bcsstk14.O
Directory della traccia:	/home/giorgi/SCL/TRACES/CHOLESKY.24
# di riferimenti per processore:	10000000
# di processori:	24

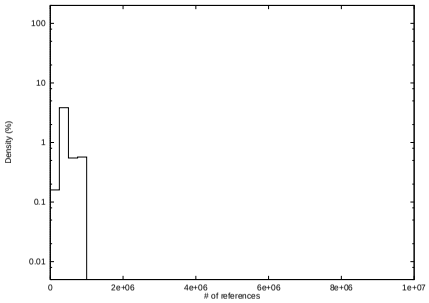
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.70	5.59	80.71



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1600	1.28



- Profilo dell'applicazione

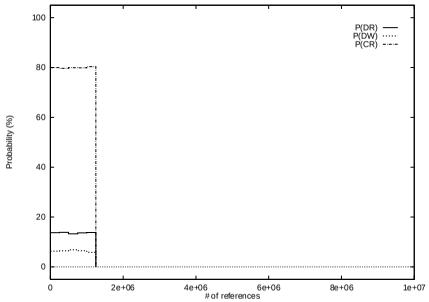
  Fattorizzazione di una matrice in forma sparsa. CHOLESKY è uno dei programmi distribuiti nella suite di SPLASH (Stanford Parallel Applications for Shared Memory). La matrice definita positiva A viene fattorizzata come $A = LL^T$, dove L è una matrice triangolare inferiore; matrici di questo tipo si incontrano in problemi di analisi strutturale, simulazione di processo, reti per l'energia elettrica. L'algoritmo effettua tre passi: ordinamento, fattorizzazione simbolica, fattorizzazione numerica. Nel presente programma non viene fatto il passo di ordinamento. La principale struttura dati è costituita dalla rappresentazione della matrice sparsa stessa. L'operazione primaria è la modifica supernodale di una colonna in cui una colonna viene modificata da tutte le colonne di un supernodo (ovvero un set di colonne con elementi diversi da zero circa nelle stesse posizioni); questa operazione coincide con la granularità. La sincronizzazione è ottenuta con dei lock, quando più processori tentano di accedere simultaneamente ad un supernodo in corso di modifica o quando tentano di prelevare supernodi momentaneamente accodati in una coda globale. Questa traccia è stata generata per 8 processori e utilizzando un file di test standard.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (E. Rothberg), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	cholesky -p8 Inputs/bcsstk14.O
Directory della traccia:	/home/giorgi/SCL/TRACES/CHOLESKY.8
# di riferimenti per processore:	10000000
# di processori:	8

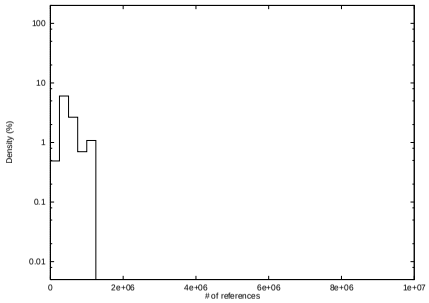
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.62	6.52	79.86



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	3087	2.47



- Profilo dell'applicazione

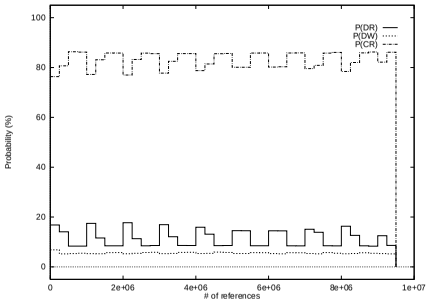
  Compressione di un file contenente un'immagine da un formato non-JPEG a formato JPEG. Vengono supportati vari formati di ingresso: PPM (PBMPLUS color format), PGM (PBMPLUS grey-scale format), BMP, GIF, Targa, RLE. JPEG è uno standard di compressione ideato per immagini del 'mondo reale' (cartoni animati o altre immagini non-realistiche non sono compresse altrettanto bene). JPEG comprime irreversibilmente, nel senso che l'immagine di uscita non è necessariamente identica all'immagine di ingresso; per immagini del mondo reale non si hanno però apprezzabili variazioni. Il programma utilizza un algoritmo di compressione chiamato DCT (per ulteriori informazioni vedere G. K. Wallace, "The JPEG Still Picture Compression Standard", Communications of ACM, April 1991, vol. 34, no.4, pp. 30-44). Il programma è rappresentativo delle applicazioni monoprocesso e fa parte delle utilità oggi normalmente disponibili nei sistemi UNIX più aggiornati. La traccia è stata prelevata a partire dall'inizio del programma fino al desiderato numero di riferimenti; il parametro 'int' significa che l'algoritmo DCT lavora solo sugli interi; l'immagine è fornita insieme ai sorgenti del programma ed ha una dimensione di circa 1KB.

- Dati generali della traccia

Fonte dei sorgenti:	Independent JPEG Group, ftp://ftp.uu.net/graphics/jpeg
Linguaggio e linee di codice:	C, 20000 linee
Linea di comando:	cjpeg -dct int -outfile testout.jpg testimg.ppm
Directory della traccia:	/home/giorgi/SCL/TRACES/CJPEG.1
# di riferimenti per processore:	10000000
# di processori:	1

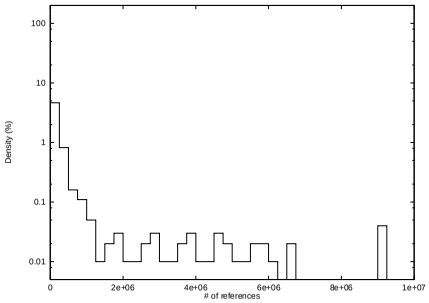
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	11.89	5.72	82.38
2000000	11.50	5.51	82.99
3000000	11.52	5.59	82.89
4000000	11.55	5.60	82.85
5000000	11.53	5.63	82.83
6000000	11.50	5.54	82.96
7000000	11.47	5.49	83.04
8000000	11.48	5.46	83.06
9000000	11.43	5.43	83.13



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1788	1.43
2000000	32	0.03
3000000	20	0.02
4000000	19	0.02
5000000	21	0.02
6000000	18	0.01
7000000	11	0.01
8000000	0	0.00
9000000	0	0.00



- Profilo dell'applicazione

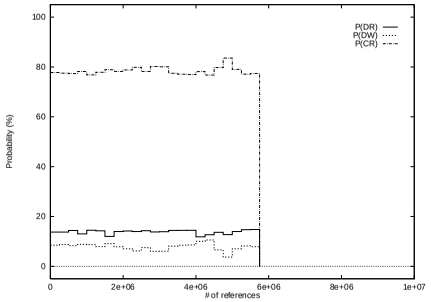
  Copia da un file ad un altro. In questo caso l'ultimo argomento è un particolare file, ovvero una directory, quindi tutti i file che si trovano sotto il filesystem '/', e che sono accessibili dall'utente, vengono trasferiti nella directory 'mia' mantenendo la stessa struttura gerarchica dell'albero (l'opzione -R significa, appunto, che vengono copiati ricorsivamente tutti i file del path sorgente). La copia di file comporta l'accesso al file system in lettura, da un lato, e in scrittura dall'altro utilizzando dei buffer per memorizzare i dati che vengono via-via trasferiti. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 1500 linee
Linea di comando:	cp -R / mia
Directory della traccia:	/home/giorgi/SCL/TRACES/CP.1
# di riferimenti per processore:	10000000
# di processori:	1

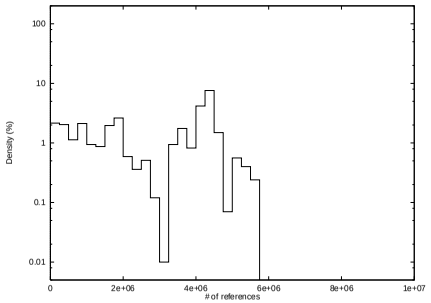
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.72	8.58	77.70
2000000	13.67	8.39	77.94
3000000	14.05	6.69	79.26
4000000	14.31	7.80	77.89
5000000	12.73	7.73	79.55



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2320	1.86
2000000	1998	1.60
3000000	493	0.39
4000000	1103	0.88
5000000	4159	3.33



- Profilo dell'applicazione

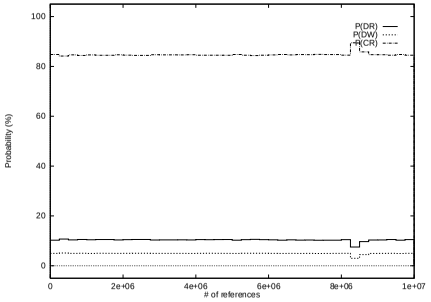
  Simulatore di cache per sistemi multiprocessore. Il programma simula un sistema multiprocessore a bus-comune e memoria condivisa con ogni processore avente una cache privata; per mantenere la coerenza delle cache si può adottare uno fra vari protocolli disponibili [Ricciardi95]. I protocolli possono essere poi valutati in base a vari parametri, fra cui il miss ratio e il Global System Power. Il programma è rappresentativo delle applicazioni monoprocesso che possono essere lanciate durante una sessione UNIX. Le strutture dati principali sono quelle che mantengono lo stato delle cache; tali strutture aumentano lineramente col numero dei processori della macchina simulata. L'elaborazione fondamentale è la gestione delle code di eventi e l'esecuzione delle operazioni relative a ciascun evento. La traccia è prelevata nella fase iniziale del programma fino ad ottenere il desiderato numero di riferimenti. I parametri della linea di comando significano, nell'ordine: protocollo UCR originario, 2 processori presenti nel sistema simulato, 3 processi presenti, altri parametri di inizializzazione prelevati dal file 'dati'.

- Dati generali della traccia

Fonte dei sorgenti:	Dipartimento Ingegneria della Informazione (Pisa)
Linguaggio e linee di codice:	C, 3500 linee
Linea di comando:	csim16 -u -c2 -p3 dati
Directory della traccia:	/home/giorgi/SCL/TRACES/CSIM16.1
# di riferimenti per processore:	10000000
# di processori:	1

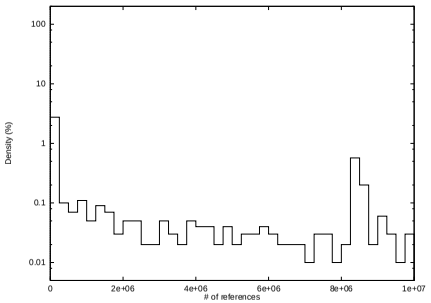
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	10.48	5.03	84.49
2000000	10.46	4.99	84.55
3000000	10.48	5.01	84.50
4000000	10.38	4.98	84.64
5000000	10.50	5.01	84.49
6000000	10.46	5.01	84.53
7000000	10.35	4.96	84.68
8000000	10.29	4.96	84.76
9000000	9.52	4.36	86.13
10000000	10.41	4.98	84.61



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	945	0.76
2000000	75	0.06
3000000	43	0.03
4000000	45	0.04
5000000	45	0.04
6000000	35	0.03
7000000	27	0.02
8000000	23	0.02
9000000	254	0.20
10000000	38	0.03



- Profilo dell'applicazione

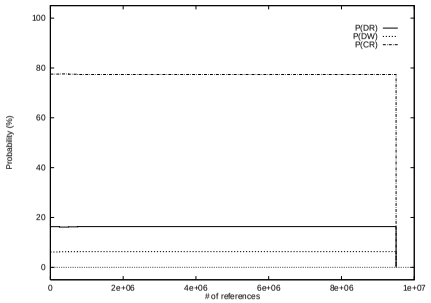
  Conversione di file durante la copia. Questo programma di utilità copia lo standard input nello standard output utilizzando una dimensione del blocco selezionabile dall'utente ed effettuando un'eventuale conversione. Nella esecuzione utilizzata per questa traccia il programma ha eseguito una semplice copia dell'output di una unità a nastro presente sull'host remoto 'r6000' verso lo standard input del comando 'tar', il quale, con le opzioni selezionate, non fa altro che mostrare il contenuto del nastro stesso. L'applicazione è molto semplice e non fa uso di particolari strutture dati; esso è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 1000 linee
Linea di comando:	rsh r6000 cat /dev/rmt0 | dd | tar -tvf -
Directory della traccia:	/home/giorgi/SCL/TRACES/DDRMT0.1
# di riferimenti per processore:	10000000
# di processori:	1

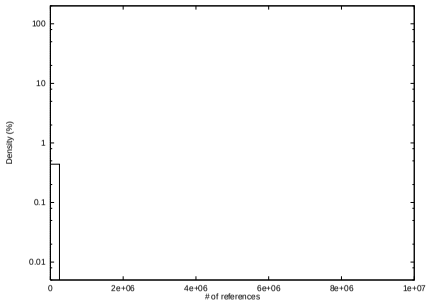
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	16.26	6.24	77.50
2000000	16.35	6.29	77.36
3000000	16.35	6.29	77.36
4000000	16.35	6.29	77.36
5000000	16.35	6.29	77.36
6000000	16.35	6.29	77.36
7000000	16.35	6.29	77.36
8000000	16.35	6.29	77.36
9000000	16.35	6.29	77.36



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	139	0.11
2000000	0	0.00
3000000	0	0.00
4000000	0	0.00
5000000	0	0.00
6000000	0	0.00
7000000	0	0.00
8000000	0	0.00
9000000	0	0.00



- Profilo dell'applicazione

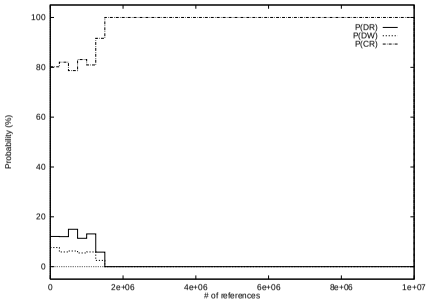
  Decompressione di un file contenente un'immagine JPEG in un formato non-JPEG. Vengono supportati vari formati di ingresso: PPM (PBMPLUS color format), PGM (PBMPLUS greyscale format), BMP, GIF, Targa, RLE. JPEG è uno standard di compressione ideato per immagini del 'mondo reale' (cartoni animati o altre immagini non-realistiche non sono compresse altrettanto bene). JPEG comprime irreversibilmente, nel senso che l'immagine di uscita non è necessariamente identica all'immagine di ingresso; per immagini del mondo reale non si hanno però apprezzabili variazioni. Il programma utilizza un algoritmo di compressione chiamato DCT (per ulteriori informazioni vedere G. K. Wallace, "The JPEG Still Picture Compression Standard", Communications of ACM, April 1991, vol. 34, no.4, pp. 30-44). Il programma è rappresentativo delle applicazioni monoprocesso e fa parte delle utilità oggi normalmente disponibili nei sistemi UNIX più aggiornati. La traccia è stata prelevata a partire dall'inizio del programma fino al desiderato numero di riferimenti; il parametro 'int' significa che l'algoritmo DCT lavora solo sugli interi; il parametro ppm significa che viene usato il formato di ingresso ppm; l'immagine è fornita insieme ai sorgenti del programma ed ha una dimensione di circa 1KB.

- Dati generali della traccia

Fonte dei sorgenti:	Independent JPEG Group, ftp://ftp.uu.net/graphics/jpeg
Linguaggio e linee di codice:	C, 20000 linee
Linea di comando:	djpeg -dct int -ppm -outfile testout.ppm testorig.jpg
Directory della traccia:	/home/giorgi/SCL/TRACES/DJPEG.1
# di riferimenti per processore:	10000000
# di processori:	1

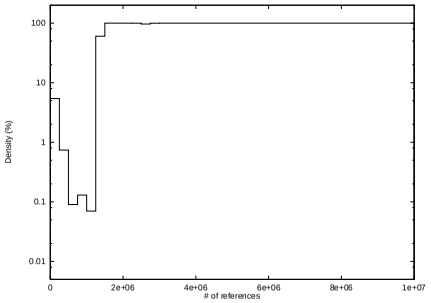
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	12.65	6.35	81.01
2000000	4.75	2.11	93.14
3000000	0.00	0.00	100.00
4000000	0.00	0.00	100.00
5000000	0.00	0.00	100.00
6000000	0.00	0.00	100.00
7000000	0.00	0.00	100.00
8000000	0.00	0.00	100.00
9000000	0.00	0.00	100.00
10000000	0.00	0.00	100.00



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1992	1.59
2000000	81388	65.11
3000000	123880	99.10
4000000	125000	100.00
5000000	125000	100.00
6000000	125000	100.00
7000000	125000	100.00
8000000	125000	100.00
9000000	125000	100.00
10000000	125000	100.00



- Profilo dell'applicazione

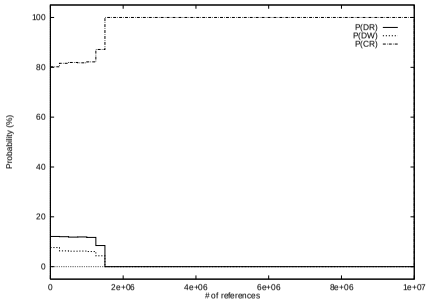
  Decompressione di un file contenente un'immagine JPEG in un formato non-JPEG. Vengono supportati vari formati di ingresso: PPM (PBMPLUS color format), PGM (PBMPLUS grey-scale format), BMP, GIF, Targa, RLE. JPEG è uno standard di compressione ideato per immagini del 'mondo reale' (cartoni animati o altre immagini non-realistiche non sono compresse altrettanto bene). JPEG comprime irreversibilmente, nel senso che l'immagine di uscita non è necessariamente identica all'immagine di ingresso; per immagini del mondo reale non si hanno però apprezzabili variazioni. Il programma utilizza un algoritmo di compressione chiamato DCT (per ulteriori informazioni vedere G. K. Wallace, "The JPEG Still Picture Compression Standard", Communications of ACM, April 1991, vol. 34, no.4, pp. 30-44). Il programma è rappresentativo delle applicazioni monoprocesso e fa parte delle utilità oggi normalmente disponibili nei sistemi UNIX più aggiornati. La traccia è stata prelevata a partire dall'inizio del programma fino al desiderato numero di riferimenti; il parametro 'int' significa che l'algoritmo DCT lavora solo sugli interi; il parametro ppm significa che viene usato il formato di ingresso ppm; l'immagine è di dimensioni più ampie rispetto al caso della traccia DJPEG.1; il file relativo ha una dimensione di circa 300KB.

- Dati generali della traccia

Fonte dei sorgenti:	Independent JPEG Group, ftp://ftp.uu.net/graphics/jpeg
Linguaggio e linee di codice:	C, 20000 linee
Linea di comando:	djpeg -dct int -ppm -outfile sc93-05.ppm sc93-05.jpg
Directory della traccia:	/home/giorgi/SCL/TRACES/DJPEGA.1
# di riferimenti per processore:	10000000
# di processori:	1

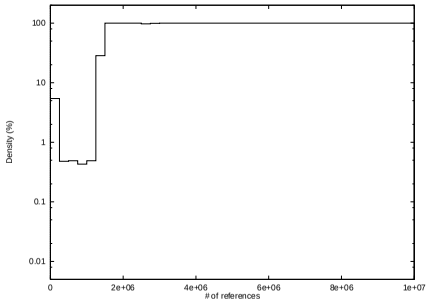
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	11.99	6.62	81.39
2000000	5.06	2.61	92.34
3000000	0.00	0.00	100.00
4000000	0.00	0.00	100.00
5000000	0.00	0.00	100.00
6000000	0.00	0.00	100.00
7000000	0.00	0.00	100.00
8000000	0.00	0.00	100.00
9000000	0.00	0.00	100.00
10000000	0.00	0.00	100.00



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2134	1.71
2000000	71540	57.23
3000000	123992	99.19
4000000	125000	100.00
5000000	125000	100.00
6000000	125000	100.00
7000000	125000	100.00
8000000	125000	100.00
9000000	125000	100.00
10000000	125000	100.00



- Profilo dell'applicazione

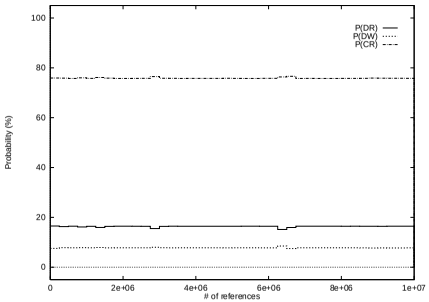
  Sommario dell'utilizzazione del disco. L'applicazione fa accesso agli i-node per ricavare le informazioni riguardanti la dimensione dei vari file. Lo spazio viene fornito normalmente in numero di blocchi di disco da 1K. Il programma utilizza delle tabelle hash, per tenere conto degli i-node visitati, le quali sono le principali strutture dati del programma e vengono allocate dinamicamente. Esso è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 700 linee
Linea di comando:	du /
Directory della traccia:	/home/giorgi/SCL/TRACES/DU.1
# di riferimenti per processore:	10000000
# di processori:	1

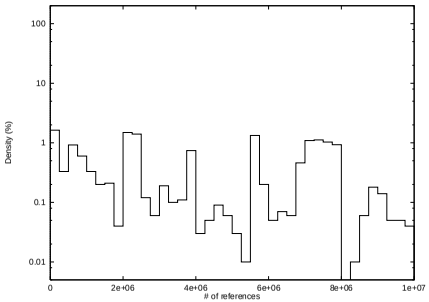
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	16.36	7.76	75.88
2000000	16.30	7.82	75.88
3000000	16.21	7.84	75.95
4000000	16.43	7.79	75.78
5000000	16.44	7.78	75.78
6000000	16.44	7.78	75.77
7000000	15.99	7.88	76.12
8000000	16.45	7.79	75.76
9000000	16.44	7.76	75.81
10000000	16.45	7.74	75.82



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1086	0.87
2000000	246	0.20
3000000	958	0.77
4000000	353	0.28
5000000	73	0.06
6000000	492	0.39
7000000	201	0.16
8000000	1303	1.04
9000000	78	0.06
10000000	90	0.07



- Profilo dell'applicazione

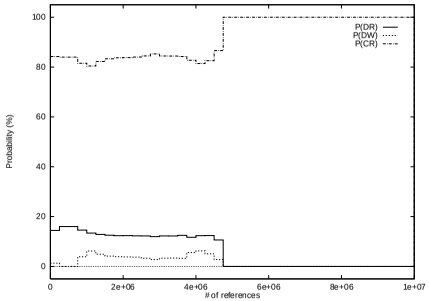
  Compressione di file. L'algoritmo utilizzato è basato sulla codifica Lempel-Ziv (LZ77). Ogni file di gzip viene compresso separatamente, non ci sono funzioni di archiviazione come in 'tar' di UNIX. Vengono trovate le stringhe duplicate entro una distanza pari a 32KB; la seconda occorrenza della stringa viene sostituita da un puntatore alla precedente stringa con una coppia del tipo (distanza, lunghezza). I letterali, ovvero le lunghezze riscontrate, vengono compresse usando un albero di Huffman, mentre le stringhe riscontrate sono compresse con un altro albero. L'elaborazione principale consiste, quindi, nell'analizzare ripetutamente un flusso di byte. Il programma è rappresentativo delle applicazioni monoprocesso in ambiente UNIX. Il ricorso alla compressione dei dati è oggi diventato un uso comune, per sfruttare al meglio le risorse di memorizzazione e diminuire il costo dei trasferimenti di dati. La traccia è stata prelevata nella fase iniziale del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 8000 linee
Linea di comando:	gzip TangoLite/thesis.ps
Directory della traccia:	/home/giorgi/SCL/TRACES/GZIPA.1
# di riferimenti per processore:	10000000
# di processori:	1

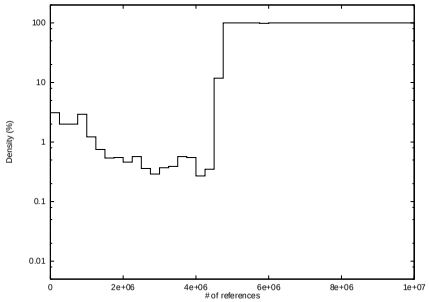
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	15.26	1.30	83.44
2000000	12.78	4.79	82.43
3000000	12.21	3.40	84.39
4000000	12.16	3.88	83.96
5000000	8.84	3.51	87.65
6000000	0.00	0.00	100.00
7000000	0.00	0.00	100.00
8000000	0.00	0.00	100.00
9000000	0.00	0.00	100.00
10000000	0.00	0.00	100.00



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	3138	2.51
2000000	954	0.76
3000000	527	0.42
4000000	588	0.47
5000000	35138	28.11
6000000	124514	99.61
7000000	125000	100.00
8000000	125000	100.00
9000000	125000	100.00
10000000	125000	100.00



- Profilo dell'applicazione

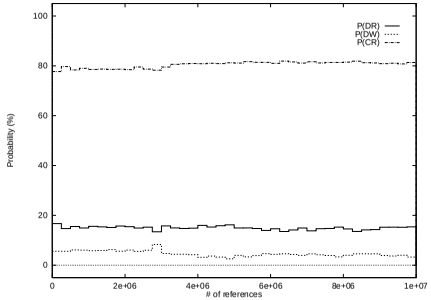
  Analisi lessicale di un file di testo. È un generatore di scanners, ovvero di programmi che riconoscono dei lessemi nel testo di un file. Il file di ingresso dà una descrizione dello scanner da generare, nella forma di coppie di espressioni regolari e routine C (dette regole). L'output di lex è il file lex.yy.c che definisce la routine 'yylex'. Il file di test è uno dei file degli esempi allegati ai sorgenti del software. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU flex (autore V. Paxson), ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 12000 linee
Linea di comando:	lex test
Directory della traccia:	/home/giorgi/SCL/TRACES/LEX.1
# di riferimenti per processore:	10000000
# di processori:	1

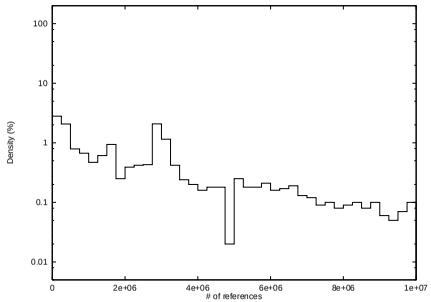
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	15.46	5.84	78.70
2000000	15.47	5.90	78.63
3000000	14.77	6.50	78.74
4000000	15.09	4.45	80.47
5000000	15.84	3.17	81.00
6000000	14.65	3.95	81.41
7000000	14.31	4.30	81.39
8000000	14.62	4.00	81.38
9000000	14.15	4.41	81.44
10000000	15.28	3.71	81.01



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1979	1.58
2000000	710	0.57
3000000	1041	0.83
4000000	626	0.50
5000000	165	0.13
6000000	259	0.21
7000000	200	0.16
8000000	119	0.10
9000000	115	0.09
10000000	91	0.07



- Profilo dell'applicazione

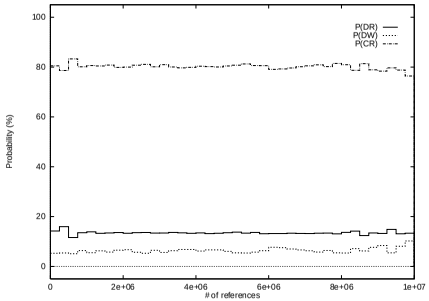
  Elenco del contenuto di directory. Il programma analizza le informazioni contenute negli i-node determinati dal path specificato sulla linea di comando e presenta in uscita un elenco dei file e delle directory secondo una varietà di formati possibili. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX; probabilmente, questo è il comando più utilizzato in assoluto. Viene dapprima costruita una lista delle directory che si intendono analizzare e poi per ogni directory vengono stampati i file in essa contenuti (ricordare che in UNIX sono file anche le directory, i link, i device). L'opzione -R significa analisi ricorsiva di ogni sotto-directory contenuta nella directory specificata nella linea di comando (in questo caso l'intero file-system). L'opzione -a significa che vengono elencati anche i file che iniziano con il carattere '.'. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	ls -aR /
Directory della traccia:	/home/giorgi/SCL/TRACES/LS-AR.1
# di riferimenti per processore:	10000000
# di processori:	1

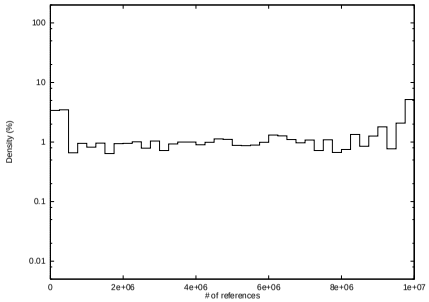
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.83	5.55	80.62
2000000	13.55	6.01	80.43
3000000	13.48	6.04	80.48
4000000	13.46	6.39	80.14
5000000	13.37	6.38	80.25
6000000	13.48	5.73	80.79
7000000	13.25	7.22	79.53
8000000	13.26	5.99	80.76
9000000	13.42	6.60	79.98
10000000	13.64	8.04	78.32



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2654	2.12
2000000	1052	0.84
3000000	1186	0.95
4000000	1142	0.91
5000000	1287	1.03
6000000	1134	0.91
7000000	1453	1.16
8000000	1113	0.89
9000000	1313	1.05
10000000	3070	2.46



- Profilo dell'applicazione

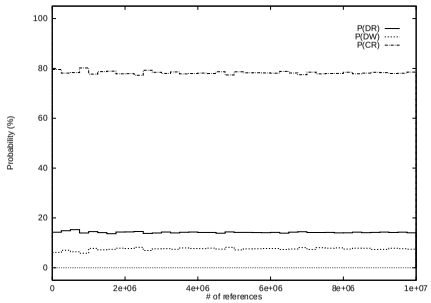
  Elenco del contenuto di directory. Il programma analizza le informazioni contenute negli i-node determinati dal path specificato sulla linea di comando e presenta in uscita un elenco dei file e delle directory secondo una varietà di formati possibili. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX; probabilmente, questo è il comando più utilizzato in assoluto. Viene dapprima costruita una lista delle directory che si intendono analizzare e poi per ogni directory vengono stampati i file in essa contenuti (ricordare che in UNIX sono file anche le directory, i link, i device). L'opzione -R significa analisi ricorsiva di ogni sotto-directory contenuta nella directory specificata nella linea di comando (in questo caso l'intero file-system). L'opzione -l fa in modo che siano stampate per esteso varie informazioni riguardanti i file (dimensione, data, diritti); l'opzione -t provoca l'ordinamento per data decrescente degli elenchi di file presentati. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	ls -ltR /
Directory della traccia:	/home/giorgi/SCL/TRACES/LS-LTR.1
# di riferimenti per processore:	10000000
# di processori:	1

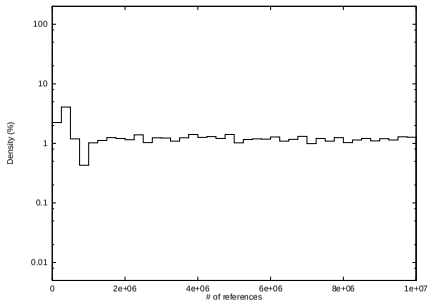
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	14.60	6.36	79.04
2000000	14.16	7.56	78.28
3000000	14.18	7.62	78.21
4000000	14.23	7.69	78.08
5000000	14.15	7.82	78.03
6000000	14.16	7.53	78.32
7000000	14.21	7.67	78.12
8000000	14.13	7.81	78.06
9000000	14.15	7.64	78.22
10000000	14.21	7.59	78.21



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2478	1.98
2000000	1437	1.15
3000000	1503	1.20
4000000	1555	1.24
5000000	1621	1.30
6000000	1422	1.14
7000000	1516	1.21
8000000	1419	1.14
9000000	1401	1.12
10000000	1531	1.22



- Profilo dell'applicazione

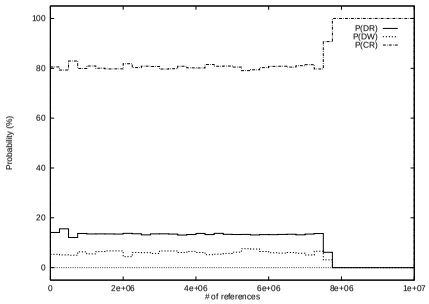
  Elenco del contenuto di directory. Il programma analizza le informazioni contenute negli i-node determinati dal path specificato sulla linea di comando e presenta in uscita un elenco dei file e delle directory secondo una varietà di formati possibili. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX; probabilmente, questo è il comando più utilizzato in assoluto. Viene dapprima costruita una lista delle directory che si intendono analizzare e poi per ogni directory vengono stampati i file in essa contenuti (ricordare che in UNIX sono file anche le directory, i link, i device). L'opzione -R significa analisi ricorsiva di ogni sotto-directory contenuta nella directory specificata nella linea di comando (in questo caso l'intero file-system). La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	ls -R /
Directory della traccia:	/home/giorgi/SCL/TRACES/LS-R.1
# di riferimenti per processore:	10000000
# di processori:	1

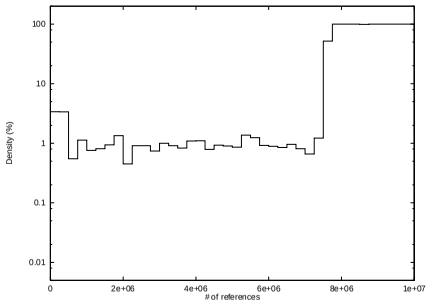
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.88	5.45	80.67
2000000	13.53	6.35	80.12
3000000	13.52	5.56	80.92
4000000	13.36	6.50	80.14
5000000	13.55	5.61	80.84
6000000	13.26	6.94	79.80
7000000	13.29	5.93	80.78
8000000	8.34	3.69	87.97
9000000	0.00	0.00	100.00
10000000	0.00	0.00	100.00



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	2634	2.11
2000000	1205	0.96
3000000	938	0.75
4000000	1195	0.96
5000000	1165	0.93
6000000	1375	1.10
7000000	1101	0.88
8000000	48049	38.44
9000000	124741	99.79
10000000	125000	100.00



- Profilo dell'applicazione

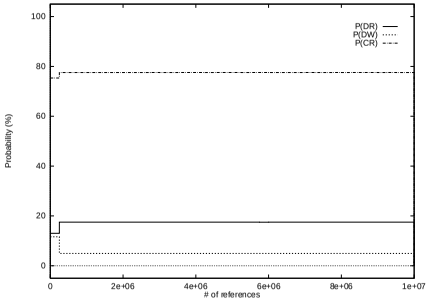
  Simulazione del flusso in fluidi rarefatti. MP3D è uno dei programmi della suite di SPLASH (Stanford Parallel Applications for Shared Memory). È un simulatore di particelle tridimensionali che viene usato per studiare le pressioni e i profili di temperatura degli aeroveivoli nell'alta atmosfera mentre viaggiano a velocità ultrasoniche. Il programma usa i metodi Monte Carlo per effettuare un'analisi statistica della traiettoria che sarà raggiunta nel fluido di natura discreta; la grande quantità di dati acceduti è funzione del numero di molecole simulate. Ad ogni time-step viene valutata la posizione e la velocità di ogni molecola, la quale viene spostata in accordo ad esse, alle altre molecole e al velivolo. Il programma ha granularità grossa; si presta bene ad essere parallelizzato perché ogni molecola può essere trattata indipendentemente, ad ogni time-step. Il principale tipo di sincronizzazione è la barriera fra time-step. La geometria del velivolo riguarda il 'flat sheet problem'. Questa traccia riguarda il caso di 10000 molecole, su 1 processore e per 20 time-steps.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J.D. McDonald), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 1500 linee
Linea di comando:	mp3d 10000 1 < run.in (20 q e)
Directory della traccia:	/home/giorgi/SCL/TRACES/MP3D.1
# di riferimenti per processore:	10000000
# di processori:	1

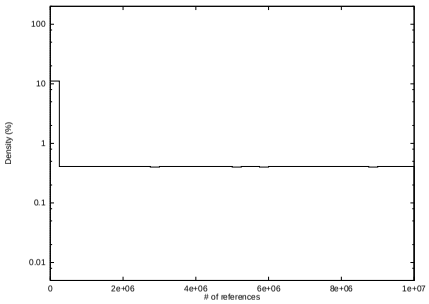
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	16.39	6.62	76.99
2000000	17.51	4.95	77.54
3000000	17.51	4.95	77.54
4000000	17.51	4.95	77.54
5000000	17.50	4.95	77.55
6000000	17.50	4.95	77.55
7000000	17.51	4.95	77.54
8000000	17.51	4.95	77.55
9000000	17.50	4.95	77.55
10000000	17.51	4.95	77.54



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	3859	3.09
2000000	509	0.41
3000000	509	0.41
4000000	508	0.41
5000000	509	0.41
6000000	506	0.40
7000000	510	0.41
8000000	508	0.41
9000000	507	0.41
10000000	509	0.41



- Profilo dell'applicazione

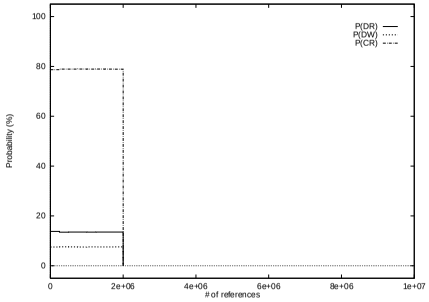
  Simulazione del flusso in fluidi rarefatti. MP3D è uno dei programmi della suite di SPLASH (Stanford Parallel Applications for Shared Memory). È un simulatore di particelle tridimensionali che viene usato per studiare le pressioni e i profili di temperatura degli aeroveivoli nell'alta atmosfera mentre viaggiano a velocità ultrasoniche. Il programma usa i metodi Monte Carlo per effettuare un'analisi statistica della traiettoria che sarà raggiunta nel fluido di natura discreta; la grande quantità di dati acceduti è funzione del numero di molecole simulate. Ad ogni time-step viene valutata la posizione e la velocità di ogni molecola, la quale viene spostata in accordo ad esse, alle altre molecole e al velivolo. Il programma ha granularità grossa; si presta bene ad essere parallelizzato perché ogni molecola può essere trattata indipendentemente, ad ogni time-step. Il principale tipo di sincronizzazione è la barriera fra time-step. La geometria del velivolo riguarda il 'flat sheet problem'. Questa traccia riguarda il caso di 10000 molecole, su 24 processori e per 20 time-steps.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J.D.McDonald), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 1500 linee
Linea di comando:	mp3d 10000 24 < run.in (20 q e)
Directory della traccia:	/home/giorgi/SCL/TRACES/MP3D.24
# di riferimenti per processore:	10000000
# di processori:	24

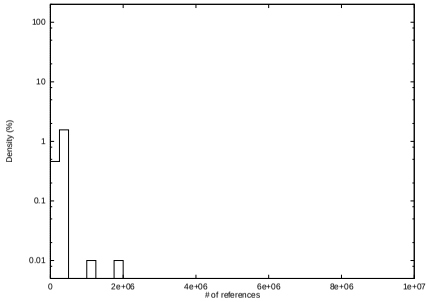
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.58	7.56	78.86
2000000	13.52	7.56	78.92



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	633	0.51
2000000	4	0.00



- Profilo dell'applicazione

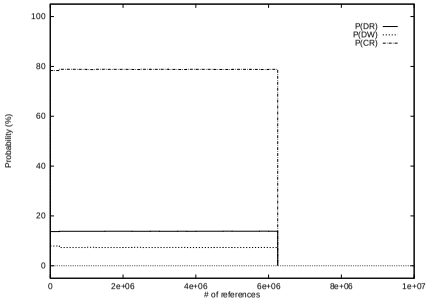
  Simulazione del flusso in fluidi rarefatti. MP3D è uno dei programmi della suite di SPLASH (Stanford Parallel Applications for Shared Memory). È un simulatore di particelle tridimensionali che viene usato per studiare le pressioni e i profili di temperatura degli aeroveivoli nell'alta atmosfera mentre viaggiano a velocità ultrasoniche. Il programma usa i metodi Monte Carlo per effettuare un'analisi statistica della traiettoria che sarà raggiunta nel fluido di natura discreta; la grande quantità di dati acceduti è funzione del numero di molecole simulate. Ad ogni time-step viene valutata la posizione e la velocità di ogni molecola, la quale viene spostata in accordo ad esse, alle altre molecole e al velivolo. Il programma ha granularità grossa; si presta bene ad essere parallelizzato perché ogni molecola può essere trattata indipendentemente, ad ogni time-step. Il principale tipo di sincronizzazione è la barriera fra time-step. La geometria del velivolo riguarda il 'flat sheet problem'. Questa traccia riguarda il caso di 10000 molecole, su 8 processori e per 20 time-steps.

- Dati generali della traccia

Fonte dei sorgenti:	SPLASH (J.D. McDonald), ftp://www-flash.stanford.edu/pub/splash2
Linguaggio e linee di codice:	C, 1500 linee
Linea di comando:	mp3d 10000 8 < run.in (20 q e)
Directory della traccia:	/home/giorgi/SCL/TRACES/MP3D.8
# di riferimenti per processore:	10000000
# di processori:	8

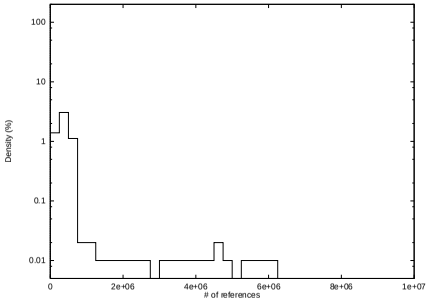
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.81	7.47	78.72
2000000	13.85	7.34	78.81
3000000	13.85	7.34	78.81
4000000	13.85	7.34	78.81
5000000	13.87	7.33	78.80
6000000	13.87	7.33	78.80



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1747	1.40
2000000	18	0.01
3000000	10	0.01
4000000	14	0.01
5000000	15	0.01
6000000	11	0.01



- Profilo dell'applicazione

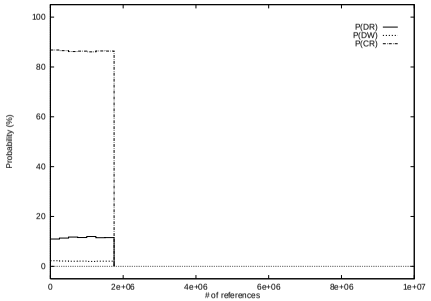
  Rimozione di file. Questo programma fa semplicemente accesso agli i-node per determinare quali entry deve marcare come inutilizzate. Il parametro -r significa che l'operazione avviene in maniera ricorsiva su tutti i nodi dell'albero facente capo alla directroy specificata; il flag -f forza la rimozione senza chiedere conferma dell'effettiva volontà dell'utente di cancellare il file. Questa esecuzione viene abbinata a quella del comando 'cp -R / mia' in modo tale che l'operazione di cancellazione coinvolga un numero sufficientemente alto di nodi. Il programma è rappresentativo delle applicazioni monoprocesso e dei comandi di utilità utilizzati in ambiente UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU file utilities 3.12, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 500 linee
Linea di comando:	rm -rf mia
Directory della traccia:	/home/giorgi/SCL/TRACES/RM.1
# di riferimenti per processore:	10000000
# di processori:	1

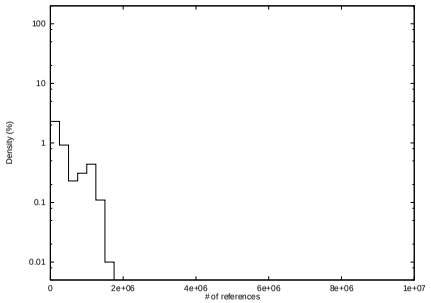
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	11.40	2.13	86.47



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	1176	0.94



- Profilo dell'applicazione

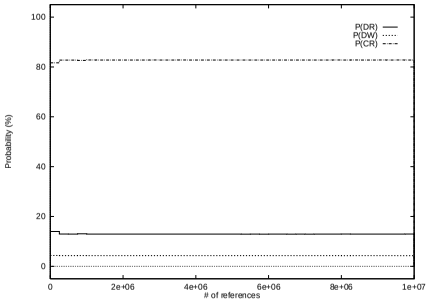
  Collegamento con un host remoto attraverso rete usando il protocollo TELNET. Esso è la parte client e prevede che sull'host remoto esista la parte server che, agendo da 'demone' ascolta e serve le richieste del client. Il flusso di dati consiste nei caratteri ricevuti e trasmessi, i quali corrispondono all'input che, l'utente che ha fatto telnet, immette da tastiera e nell'output presentato sullo schermo locale, inviato dallo standard output e dallo standard error dell'applicazione lanciata sull'host remoto (nel caso di questa traccia, il comando ls). Il programma è rappresentativo delle applicazioni monoprocesso e dei programmi utilizzati in ambienti 'distribuiti' sotto UNIX. La traccia è prelevata a partire dall'inizio del programma fino al raggiungimento del desiderato numero di riferimenti.

- Dati generali della traccia

Fonte dei sorgenti:	GNU, ftp://prep.ai.mit.edu/pub/gnu
Linguaggio e linee di codice:	C, 2000 linee
Linea di comando:	telnet r6000 (e poi, una volta fatto login, ls -R /)
Directory della traccia:	/home/giorgi/SCL/TRACES/TELNET.1
# di riferimenti per processore:	10000000
# di processori:	1

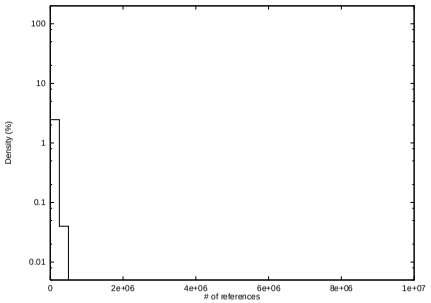
- Statistiche d'accesso

RIF	*P(DR)*	*P(DW)*	*P(CR)*
1000000	13.23	4.31	82.45
2000000	12.93	4.29	82.78
3000000	12.93	4.29	82.78
4000000	12.93	4.29	82.78
5000000	12.93	4.29	82.78
6000000	12.93	4.29	82.78
7000000	12.93	4.29	82.78
8000000	12.93	4.29	82.78
9000000	12.94	4.28	82.78
10000000	12.94	4.27	82.78



- Densità incrementale di blocchi da 32B (normalizzata per # di processori)

RIF	*Densità-i*	*Densità-i(%)*
1000000	781	0.62
2000000	0	0.00
3000000	0	0.00
4000000	0	0.00
5000000	0	0.00
6000000	0	0.00
7000000	0	0.00
8000000	0	0.00
9000000	0	0.00
10000000	0	0.00



Ringraziamenti

Desidero ringraziare il Computer Systems Laboratory dell'Università di Stanford, per aver gentilmente fornito il software TangoLite, senza il quale questo lavoro non avrebbe potuto evolversi allo stadio attuale in così breve tempo e, in particolare, Steve Herrod, per aver fornito il supporto al tracing da utilizzare con TangoLite.

Un ringraziamento particolare a Luigi Ricciardi, i colloqui col quale sono risultati di grande importanza per la comprensione del simulatore di cache per sistemi multiprocessore, di cui egli è autore, e per aver fornito, specialmente negli ultimi mesi, il necessario feedback per la messa a punto degli strumenti di cui è composto Trace Factory.

Un ringraziamento ai dottorandi del Dipartimento di Ingegneria della Informazione Adriana Maggiore, Guido Pazzaglia e Lorenzo Vicisano, per il fondamentale supporto logistico nella utilizzazione delle risorse.

Infine, un grazie di cuore a Filippo Pulighedu e a Andrea Gorelli per l'opera di revisione di questa tesi.

Bibliografia

[Agarwal86] A. Agarwal, R.L. Sites, M. Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode*", Proc. 13th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., OrderNo. 719, 1986, pp. 119-127, *agarwal@mit.edu*.

[Agarwal88] A. Agarwal, A. Gupta, "Memory Reference Characteristic of Multiprocessor Applications under Mach*", Proc. ACM SIGMetrics*, May 1988, Santa Fe, NM, pp. 215-225, *agarwal@mit.edu*.

[Borg90] A. Borg, R.E. Kessler, D.W. Wall, "Generation and Analysis of Very Long Address Traces", *Proc 17th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., OrderNo 2047, 1990, pp 270-281, *borg@decwrl.dec.com*.

[Chaiken90] D. Chaiken, C. Fields, K. Kurihara, A. Agarwal, "Directory-Based Cache Coherence in Large-Scale Multiprocessors", *IEEE Computer*, Vol.23, No.6, pp. 49-58, June 1990, *chaiken@hing.ls.mit.edu*

[Chandra94] R. Chandra, A. Gupta, J. L. Hennessy, "COOL: An Object-Based Language for Parallel Programming", IEEE Computer, Vol. 27, No. 8, pp. 13-26, Aug. 1994, *rohit@cool.stanford.edu*.

[Eggers88] S.J. Eggers, R.H. Katz, "A Characterization of Sharing in Parallel Programs and its Application to Coherency Protocol Evaluation", *Proc 15th Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., OrderNo. 861, pp. 373-382, *eggers@cs.washington.edu*.

[Eggers89] S.J. Eggers, "Simulation Analysis of Data Sharing in Shared Memory Multiprocessors", Ph.D. dissertation, Univ. of California, Berkeley, Apr. 1989, *eggers@cs.washington.edu*.

[Eggers90] S.J. Eggers et al., "Techniques for Efficient In-Line Tracing on a Shared-Memory Multiprocessor", *Proc. ACM SIGMetrics Int'l Conf. Measurement and Modeling of Computer Systems*, 1990, pp. 37-47, *eggers@cs.washington.edu*.

[Flanagan92] K. Flanagan, K.Grimsrud, J. Archibald, B. Nelson, "BACH: BYU Address Collection Hardware", Tech, Report TR-A150-92.1, Dept. of Electrical & Computer Engineering, Brigham Young Univ., Jan. 1992, *archibald@ee.byu.edu*

[Gee93] J.D. Gee, A.J. Smith, "Absolute and Comparative Performance of Cache Consistency Algorithms", Tech. Report, Univ. of California, 1993.

[Gharachorl92] K. Gharachorloo, A. Gupta, J. Hennessy. "Hiding memory latency using dynamic scheduling in shared-memory multiprocessors", *Proc 19th Int'l Symp. Computer Architecture*, pages 22-33, May 1992, *ag@cs.stanford.edu*.

[Goldschmidt90] S.R. Goldschmidt, H. Davis, "Tango Introduction and Tutorial", Tech. Report CSL-TR-90-410, Computer Systems Laboratory, Stanford University, Stanford, Calif., Jan. 1990, *goldschm@meadow.stanford.edu*.

[Goldschmidt93] S.R. Goldschmidt, "Simulation of Multiprocessors: Accuracy and performance", PhD. Thesis, Computer Systems Laboratory, Stanford University, Stanford, Calif., June 1993, *goldschm@meadow.stanford.edu*.

[Grimsrud92] K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, B. Nelson, "BACH: A Hardware Monitor for Tracing Multiprocessor-Based Systems", Dept. of Electrical & Computer Engineering, Brigham Young Univ., Nov. 1992, *bach@ee.byu.edu*.

[Gupta92a] A. Gupta, W. Weber, "Cache Invalidation patterns in Shared-Memory Multiprocessors", *IEEE Transactions on Computing*, Vol. 41, No. 7, July 1992 pp. 1794-810, *ag@cs.stanford.edu*.

[Gupta92b] A. Gupta, T. Joe, P. Stenstrom, "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA architectures", Technical Report CSL-TR-92-524, Stanford university Computer System Laboratory, May 1992, *ag@cs.stanford.edu*.

[Johnson94] E.E. Johnson, J. Ha, "PDATS, Lossless Address Trace Compression For Reducing File Size And Access Time", *Proceedings 1994 IEEE International Phoenix Conference on Computers and Communications*, pp. 213-219, *ejohnson@nmsu.edu*.

[Knuth73] D.E. Knuth, *The art of Computer Programming, Volume II: Semi-Numerical Algorithms*, Reading, MA, Addison-Wesley, 1973.

[Hwang93] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*,

McGraw-Hill, 1993.

[Herrod95] S.A. Herrod, "TangoLite: A Multiprocessor Simulation Environment", Computer Systems Laboratory, Stanfoed University, Stanford, CA 94305, *herrod@cs.stanford.edu*.

[Larus90] J.R. Larus, "Abstract Execution: A technique for Efficiently Tracing Programs", *Software Practice & Experience*, Vol. 20, No. 12, Dec. 1990, pp. 1241-1258, *larus@cs.wisc.edu*.

[Larus93] J.R. Larus, "Efficient Program Tracing", *IEEE Computer*, Vol. 26, No. 5, May 93, pp. 52-61, *larus@cs.wisc.edu*.

*[Laudon92] J. Laudon, A. Gupta, M. Horowitz, "Architectural and Implementation Tradeoffs in the design of multiple-context processors", Techinical Report CSL-TR-92-523, Stanford University Computer System Laboratory, May 1992, *ag@cs.stanford.edu*.

*[Lusk87] E.L. Lusk, R.A. Overbeek, et al., *Portable Programs for Parallel Processors*, Holt, Rinehart and Winston, Inc. 1987.

[Martonosi95] M. Martonosi, A. Gupta, T. E. Anderson, "Tuning memory Performance of Sequential and Parallel Programs", IEEE Computer, Vol. 28, No. 4, pp. 32-40, April 1995, *martonosi@princeton.edu*.

[Mudge95] T. Mudge, D. Nagle, R. Uhlig, M. Upton, "High-Performance Processors", *Tutorial in 22nd Int'l Symp. Computer Architecture*, June 1995, *tnm@eecs.umich.edu*.

[Ousterhout90] J.K. Ousterhout, "Tcl: An Embeddable Command Language", *Proceedings 1990 Winter USENIX Conference*, Jan. 1990, pp. 133-146, *ouster@cs.berkeley.edu*.

[Ousterhout94] J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, 1994, *ouster@cs.berkeley.edu*.

[Prete95] C.A. Prete, L. Ricciardi, G.P. Prina, "Reducing Coherence-Related Overhead in Multiprocessor Systems", Proceedings of EuroMicro Workshop on Parallel and Distributed Processing", San Remo, Italy, Jan. 95, *prete@iet.unipi.it*.

[Prete95b] C.A. Prete, "A Cache Coherence Solution Based on Selective Invalidation for Shared-Bus Multiprocessors", to appear, *prete@iet.unipi.it*

[Ricciardi95] L. Ricciardi, G.P. Prina, C.A. Prete, "A Trace-Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor Systems", *ricciard@pisolo.iet.unipi.it*.

[Singh92] J.P. Singh, W.D. Weber, A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory", *Computer Architecture News*, Mar. 1992, Vol. 20, No. 1, pp. 5-44, *jps@cs.stanford.edu*.

*[So88] K. So et al., "PSIMUL -- A System for Parallel Execution of Parallel Programs", in *Performance Evaluation of Supercomputers*, J.L. Martin, ed. Elsevier Science Publishers B. V., North Holland, 1988, pp. 187-213.

[Stunkel89] C.B. Stunkel, W.K. Fuchs, "TRAPEDS: Producing Traces for Multicomputer Via Execution-driven Simulation", *Proc. ACM SIGMetrics Int'l Conf. Measurement and Modeling of Computer Systems*, 1989, pp. 70-78.

[Stunkel91] C.B. Stunkel, B. Janssens, W.K. Fuchs, "Address Tracing for Parallel Machines", *IEEE Computer*, Vol. 24, No. 1, Jan. 1991, pp- 31-45.

[Tanenbaum92] A.S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992, *ast@cs.vu.nl*.

[Vashow93] B. Vashow, "Address Trace Collection an Trace Driven Simulation of Bus Based, shared Memory Multiprocessors", Research Report No. CMUCDS-93-4, Carnegie Mellon University, Pittsburgh, PA, March 1993, *dps+@cs.cmu.edu*.

*[Wiecek82] C.A. Wiecek, "A Case Study of VAX 11 Instruction Set Usage for Compiler Execution", *Proc. Symp. Architectural Support for Programming Languages and Operating Systems*, 1982, pp. 177-184.

[Ziv77] J. Ziv, A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, Vol. 23, No. 3, Mar. 1977, pp. 337-343.