Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Siena

Tiled Architectures & Recent Proposals for Chip Multiprocessors

Sandro Bartolini, Roberto Giorgi, Enrico Martinelli, Zdravko Popovic
{bartolini, giorgi, enrico, popovic}@dii.unisi.it

Siena, 13. May 2005

Abstract – How to effectively use the increasing number of transistors available on a single chip while avoiding the wire delay problem? This is one of the most interesting research questions for the microarchitecture community. We have finally arrived at the point where the time needed for signals to reach the opposite edge of a chip is becoming longer than one cycle. This leads to the impossibility of gaining performance improvements via the scaling of superscalar architectures. One possible solution for using the available transistors efficiently and effectively, while hiding wire delay as much as possible is to parallelize resource usage through resource clustering and decoupling. For example, using on chip multiprocessor architectures is the most natural way to increase performance beyond what we can obtain from a single processor core. A generalization of this concept has led to several solutions for chip multiprocessors. The focus of this paper is to review some recent proposals that employ the clusterization/tiling paradigm, at different extents, in a comparative fashion, and highlight their main features and advantages.

Recently, a good number of tiled/clustered architectures have been proposed, indicating that this field is gathering high interest from both academia and industry: WaveScalar (University of Washington), TRIPS (University of Texas at Austin), Smart Memories (Stanford University), Synchroscalar (University of California, Davis and California Polytechnic State University, San Luis Obispo), Raw (MIT), CODE (Stanford University), SCALE (MIT). Even if such proposals adopt a resource tiling approach to implement chip multiprocessors, several other approaches make use of multithreading, dataflow ISA, vectorization and clustering to overcome the limitations of simple symmetric multiprocessor (SMP) design.

Keywords: multiprocessor architectures, scalability, wire delay, chip multiprocessors

1. The main idea

First of all, let's focus on the potentials of the main idea behind the architectures we are considering. For example,

WaveScalar architecture is trying to overcome low scalability in current superscalar architectures aiming to exploit the untapped *dataflow locality* through static and dynamic prediction of instruction dependencies in the dynamic trace of an application, and proper allocation of instructions.

TRIPS uses a flexible approach to adapt its architecture in order to exploit different types of application parallelism, like instruction, thread or data-level parallelism, achieving a better utilization of microarchitectural resources. Processing cores and on-chip memory can be configured to achieve this.

Smart Memories is a modular reconfigurable architecture made up of an array of tiles, with the aim to efficiently execute many different types of applications. At design time, each of the tiles can be configured either as a processing or memory element, and the internal memory organization of each processing element can be dynamically reconfigured.

Synchroscalar addresses communication and multimedia applications, and it is designed to provide the flexibility of DSPs while approaching the power efficiency of ASICs through non-homogeneous voltage and frequency scaling of different tile sets. In this case, the design goal is to meet performance targets with the lowest possible power consumption.

Raw architecture is proposed as a solution for the emerging problem of wire delay by tiling resources on the chip. Its ISA allows programmer to have an effective control over the communication hardware between tiles and towards off-chip modules.

The designer of CODE architecture tried to eliminate three main limitations of standard vector processors (centralized vector register file, precise exceptions, on-chip memory requirement) with extensive use of clusterization of vector resources and decoupling between operand transfers and execution.

SCALE is an implementation of vector-threaded architectural paradigm, with the goal of unifying vector and multithreaded execution.

2. Architectural characteristics and organization – Main characteristics of all examined architectures are summarized in Table 1. In the effort to clarify the main structure of the examined architectures, we tried to classify processing tiles as T1-type tiles and other tiles (mostly memory tiles) as T2-type tiles.

Table 1. – Architectural characteristics

	WaveScalar [1], [2]	TRIPS [3]	Smart Memories [4]	Synchroscalar [5]	Raw [6]	CODE [8]	SCALE [9]
Processing elements per chip - number & organization	~ 2K	64	up to 64 (1)	16 organized in columns of 4 PE	16	4	16 organized in 4 lanes, with 4 clusters in each lane
Brief description of PE	1ALU	1ALU + 1FPU	1ALU + 2FPU	6ALU + 2MUL	1ALU + 1FPU	Clusters including a VFU (2)	Specialized clusters (3)
Memory associated to each PE	Instruction storage ~0.75KB (4)	Instruction storage ~1152B (5)	2 integer register files + 1 shared and 4 local FP register files	32x32bits register file	L1.I$ (32KB) + L1.D$(32KB)	8 vector registers (32x64bits)	I$ data + registers (6)
Total number of tiles	~ (32T1 + 24T2)	4T1 + 5T2	64(T1 or T2)	16T1 + 4T2	16T1	4T1+1T2	4T1+1T2
Number of PEs in the processing tile	64	16	1	1	1	1	4
Memory associated to processing tile (T1)	4L1.D$ (16KB)	4 banks L1.I$ (64KB) + 4 banks L1.D$ (64KB) + 4 register file banks (128 registers)	8KB x 16 banks	L1.D$(32KB)	Instruction cache for static router (64KB)	N/A	I$ tags
Other tiles (T2)	Memory tiles - L2$	Memory tiles (32KB x 16 banks)	Memory tiles (up to 64) (1)	Control tile (with 2KB I$)	N/A	Control tile (with L1$)	Control tile (with L1$)
Interconnection	Dynamically routed grid-based network	2D switched interconnection network	Mesh	Segmented buses (8 x 32-bit)	4 point-to-point mash 32-bit networks	N/A	Unidirectional ring between lanes

* These characteristics are mostly taken from the basic configurations evaluated in the referenced papers

(1) A tile can be either a processing tile or a memory tile.

(2) VFU = vector functional unit. This VFU is specialized in each cluster: two cluster have an integer VFU, one has a load/store VFU and the fourth doesn't have VFU, but has more vector registers.

(3) All clusters support basic integer operations. And additionally, one cluster supports memory access instructions, a second one supports fetch instructions, and a third supports integer multiply/divide.

(4) NI * (SI + 3*SO*SQ) with NI = number of instructions that can be stored = 8; SI = size of instruction = 1B; SO = size of operand = 3B; SQ = operand queue size = 8; 8*(1+3*4B*8) ~ 768B = 0.75KB; all these values, except number of instruction, are not found in literature but were assumed as a reasonable values.

(5) NI * (SI + 2*SO) = 128*(1+2*4B) = 1152B; values for SI and SO are assumed.

WaveScalar – It is the first dataflow architecture able to run programs written in any language, because it can provide traditional memory semantics [1]. WaveCache (figure 1) is one possible implementation of the WaveScalar architecture. It consists of a grid of simple **P**rocessing **E**lements (PE), each one having buffering and storage for up to 8 instructions (only one can fire each cycle). Each PE also contains logic for instruction placement and execution, input and output queues for instruction operands, communicational logic and a functional unit (ALU) [2]. PEs are grouped into domains, and four domains are grouped into a cluster, which contains shared L1 data cache, store buffer and a 4-ported bi-directional network switch. This structure can be replicated several times, so we consider this as the processing tile (T1, figure 1) of WaveScalar architecture. Instructions are mapped to some number of processing elements through a simple greedy strategy that tries to place dependent instructions into the same domain (there is work in progress on a dynamic placement algorithm), and each PE executes the set of instructions locally mapped. Unified L2 cache is distributed along edges of the grid. We name these tiles as memory tiles (T2, figure 1). For communication within a domain there is a set of shared buses and communication among domains is done through a dynamically routed grid-based on-chip network. In this grid, each network "hop" takes a single cycle. Domain size is a tradeoff parameter because large domains require more wires and more area for intra-domain communication, and small domains increase inter-domain communication costs.

TRIPS – In this architecture, the program is partitioned into blocks by the compiler [3]. A block is a portion of code with no loops, a single entry point and possibly multiple exit points. Compiler statically schedules each block onto the computational engine. Within a block,

instructions execute in dataflow order. Processing tile (T1, figure 2) consists of an array of homogenous processing nodes, each containing an integer ALU, floating point unit, frames[1] that store instructions and operands waiting to execute, and router to deliver input and output to all nodes, not just to neighboring (communication takes 0.5 cycles per "hop" in the array of processing nodes). Nearby this array, there are banked instruction caches and data caches, banked register files, and block control logic. L1 caches are connected to on-chip memory tiles (T2 tiles) through chip-wide 2D interconnection network. A TRIPS mode is a configuration of hardware resources used to exploit different types of parallelism. The following three TRIPS modes are possible: D-morph (for ILP), T-morph (TLP) and S-morph (DLP). For achieving high ILP, TRIPS D-morph uses frames as large, distributed issue window and its ISA to allow out-of-order execution without associative issue window lookups. T-morph is used to map multiple threads on processing tile, when available single thread parallelism is low. S-morph is useful for data-parallel applications, where frames are filled with unrolled loops instead of using frames for speculation or multithreading. In general, resources can be fixed (operate in same manner in every mode, for example processing nodes), specialized (not needed for every mode) or polymorphous which means that they can be reconfigured. Polymorphous resources are frames, register file banks, block sequencing control, and memory tiles. Memory tiles are banks of memory that can be configured in different ways like NUCA L2 cache banks [12], scratchpad memory, and synchronization buffers. Prototype chip is expected to have four processing cores,

[1] Actually, a frame spans over all the processing elements of the tile; a PE includes storage for N instructions with two operands; each of this storage is a portion of the whole frame space [3].

an array of 32KB memory tiles (connected by routed network) and a set of distributed memory controllers with channels to external memory.

Smart memories – A smart memory chip contains a 2D array of tiles (which can be processing or memory) connected by dynamically-routed network (figure 3) [4]. This network is also accessible from the external of the chip, so this chip multiprocessor system can be part of a wider multiprocessor system. The size of tiles is chosen so that a signal is able to propagate for a length equivalent to half of the tile perimeter in less than one clock cycle. Four processing tiles are grouped together into a "quad" connected with the network described below. In this way, there are less global network interfaces (one per quad), and efficiency of global network is increased. Each processing tile has its memory system, crossbar interconnect, processor and quad interface. Memory system is built of sixteen 8KB banks, organized as 1024x64-bit array which can perform reads, writes, compares and read-modify-writes operations. Four of the banks are fully dual-ported. Memory system can be organized in many different ways like a direct-mapped cache, set-associative cache, scratchpad memories, vector/stream register files. Processor contains two integer clusters and one floating-point cluster. Integer clusters have an ALU, a register file and a load/store unit, while the floating-point cluster contains two adders, one multiplier, one divider/sqrt unit, shared register file and each unit has its local register file to provide high operand bandwidth (figure 3). Quad interconnection network consists of nine 64-bit buses that can be configured as half-word buses for intra-quad communication, and it also connects the quad to the global network. There are three execution modes in Smart Memories: VLIW (each cycle two integer and two FP instructions are issued), multi-threaded (two

asymmetric threads per tile, one with just integer instructions, and the other with up to two instructions per cycle, both integer and FP) and streaming execution mode (four tiles in the quad are running in SIMD mode) [16].

Synchroscalar – Main targets of this architecture are signal processing applications [5]. Processing tiles form a 2D array structure and are grouped into columns (figure 4). Every column of processing tiles (T1 in figure 4) is assigned to a single thread of control, with a single SIMD controller (T2 in figure 4) and 2KB program memory. Processing element contains two 40-bit ALUs, four 8-bit video ALUs, two 40-bit accumulators, two 16x16 multipliers, one 40-bit barrel shifter, 32x32 bits register file and 32KB data memory. Each column has a specific clock generator and voltage that can be reconfigured at startup. It is possible to map different parts of the application on each column. Then, both voltage and clock frequency of each column can be tuned to the lowest possible values meeting the application constraints. All control instructions are executed into T2 tiles and only computation instructions are sent to T1 tiles. T1 and T2 tiles are interconnected with eight 32-bit separable buses (256-bit wide), that are divided into segments linking different sets of tiles. With proper control over the segments, the bus can be organized as a unique bus (broadcast message to all the tiles) or a segmented one allowing different message transfer in each section at the same time. Each column also contains a Data Orchestration Unit (DOU) controller, which is responsible to control the bus segments. They provide "zero-overhead" communication between tiles. "Zero-overhead" means buffered statically-scheduled communication.

Raw – The Raw chip is divided into n^2 identical programmable T1 tiles (current value of n is 4) [6]. Tile size is chosen so that one clock cycle is needed for signals to travel across the tile and through interconnection logic. Each T1 tile (figure 5) contains computational resources (8 stage in-order single-issue MIPS like processor and 4 stage pipelined FPU), three programmable routers (a static one having routes specified at compile time, and two dynamic routers having routes specified at run time), and one on-chip L1 cache memory (instruction and data cache for computation, 32KB each, and instruction cache for static router of 64KB). Each tile is connected to its four neighbors through channels containing two static and two dynamic 32-bit full duplex networks. At the borders of each tile, the on-chip network has a register that retains sent or received data. These registers are included directly into computational resources. Because of this, no wire is longer than the length or width of a tile. In this way, high clock speeds and further scalability can be exploited. Moreover, the programmer can program these on-chip networks through the ISA to achieve carefully orchestrated transfers of data between tiles. Also, two types of network are available, static and dynamic networks: static networks are used for operand transport between tiles, and dynamic for all other traffic (memory, interrupts, I/O, message passing). On the edges of the chip, network buses are multiplexed in hardware onto pins of the chip and they can be used for DRAM access or external I/O device.

CODE (**C**lustered **O**rganization for **D**ecoupled **E**xecution) – CODE contains a scalar core (a MIPS like processor with L1 I and D caches), vector issue logic (T2 tile, figure 6), clustered vector processors and a communication network (T1 tile, figure 6) [8]. T2 tile controls several T1 tiles named “clusters”. Each cluster contains a **V**ector **F**unctional **U**nit (VFU), a portion of the

vector register file (**CL**ustered **V**ector **R**egister **F**ile - CLVRF), one instruction queue and one input and one output interface for vector register transfers between clusters. VFU can be an arithmetic or a load-store unit. Some of the clusters (not all) can have more vector registers replacing the VFU. To execute each instruction on a large number of elements in each cycle, multiple lane organization can be applied, where each lane is a group of clusters and includes additional support for parallel datapaths and address generators (equal to number of lanes). The use of CLVRF allows us to separate the tasks of delivering operands to functional units from operand communication between functional units. In this way, the classical problem of scalability in a centralized vector register file is solved. Moreover, area, power consumption, latency and complexity of each portion of CLVRF are constant. For transferring vector operands among clusters there is a communication network. Since exchange of vector operands is separated from VRF, the type of communication network can be chosen at design time, depending on the performance that is needed. All instructions for moving vector registers are generated automatically by vector issue logic, which also selects one cluster to execute each vector instruction. CODE uses renaming vector registers to allow us to access all clustered registers globally, instead of the 32 seen by the programmer. Renaming table is also maintained in vector issue logic. CODE addresses the problem of sensitivity to memory latency through execution decoupling. In fact, network interfaces are allowed to look ahead in instruction queue and, if there are no dependencies, start vector register transfers before the start of a certain instruction that needs these operands. Special case is considered for precise exceptions through the use of a history buffer in vector issue logic. With this buffer, it is possible to restore previous

state of the machine, if some instruction causes an exception. History buffer holds just renaming information, not values of vector registers.

SCALE – In this architecture, a conventional MIPS based scalar processor is extended with a vector-thread unit, divided into some lanes (T1 tiles, figure 7) [9]. Tile T2 contains the control processor. Each lane contains physical registers and functional units. In this case, lanes execute decoupled from each other, which is different from other vector processor architectures. By means of time-multiplexing functional units the vector-thread unit can provide different numbers of virtual processors (VP), depending on the number of registers needed to each processor for holding its state and data. Each lane has the same number of VPs. There are two interacting instruction sets – one for control processor and one for VPs. VP executes strings of RISC-like instructions grouped into **A**tomic **I**nstruction **B**locks (AIBs). Control processor directs execution on VPs by delivering the same AIB to all VPs (for data-parallel code), or one AIB to a single VP, with *vector-fetch* and *VP-fetch* instructions respectively. Each VP can also direct its own execution with *thread-fetch* instructions that requests to execute a new AIB once active AIB has finished. When a thread starts, it executes completely before that another command from control processor is executed. In this way, threaded execution in VPs is supported. A T1 tile (lane) includes one **C**ommand **M**anagement **U**nit (CMU) and one **E**xecution **U**nit (EU) (figure 7). For transfers between VPs, there is a unidirectional ring that connects lanes (it can be used for loops with cross-iteration dependencies). The CMU decides which fetch command to process. The EU contains multiple heterogeneous clusters with independent register sets, and a small cache for holding AIBs (big enough to hold at least one AIB of maximal size). The cluster provides basic

integer operations and memory access instructions, fetch instructions or integer multiply/divide instructions in particular cluster. Clusters that perform some specialized operations can be added too. Registers can be allocated as private (preserve their values among AIBs) or shared. Additionally, there are also chain registers for two ALU input operands to avoid reading and writing the register file. Depending on how many shared and private registers are required in each VP, control processor configures the total number of VPs. This is usually done once outside each loop.

3. Benchmarks and simulation framework – In Table 2, we highlight more information about the evaluation methodology, including simulators, tools, compilers and benchmark applications that are used in the reference papers evaluating the discussed architectures ([1], [3], [4], [5], [7], [8], [9]). SPEC (92, 95 or 2000) [19] is one of the mostly used benchmark. EEMBC [20] suite is used for evaluating performance of vector architectures, because those benchmarks are highly vectorizable. For evaluating computational power and ability to exploit ILP, both SPEC and Mediabench [13] are used. In the column other benchmarks, we put all applications that were used as separate applications or algorithms, apart from other standard benchmark-suites.

Table 2. – Benchmarks, simulators, compilers and tools

	SPECint SPECfp	Media-Bench	EEMBC	Other benchmaks	Simulation framework
WaveScalar [1]	vpr, twolf, mcf (2000) Equake, art (2000)	adpcm, mpeg2encode	-	fft	tool for converting Alpha binaries into WaveScalar ISA; execution-driven simulator
TRIPS (D morph) [3]	bzip2, gzip, mcf, parser, twolf, vortex, m88ksim, compress Ammp, art, equake, swim, mgrid, hydro2d, tomcatv, turbo3d	adpcm, mpeg2	-	dct	Compiling with Trimaran tool set; scheduling with custom scheduler/rewriter
TRIPS (T morph) [3]	Mcf, bzip2, parser, m88ksim art, equake, tomcatv, compress	-	-	-	same as D – morph
TRIPS (S morph) [3]	-	Convert, dct, fft8, fir16, idea, transform (extracted from)	-	-	hand-coding in TRIPS meta-assembly language; mapping with custom scheduler; event-driven simulator
Smart Memories [4]	Compress, ijpeg (95) M88ksim(95), alvin(92)	mpeg	-	fft, fir, dct convolve (Imagine); grep, wc, simplex (Hydra)	Adaptation of Imagine and Hydra compilation and simulation tools
Synchro-scalar [5]	-	-	-	DDC, Stereo Vision, 802.11a, MPEG4	Adapted variant of SimpleScalar; ISA retargeted to BlackfinISA; compilation to assembly, hand-optimizing inner loops, hand parallelized, hand-scheduling communication and insertion of NOPS
RAW [7]	vpr, mcf, parser, bzip2, twolf (2000) Fppp, tomcatv, nasa7(92), swim(95), mgrid, applu, mesa, equake, ammp, apsi (2000)	-	-	jacobi, life (*), SHA, AES decode; six StreamIt benchmarks; linear algebra algorithms; 8b/10b Encoder, 802.11a ConvEnc: STREAM	Validated simulator; Rawcc compiler
CODE [8]	-	-	rgb2cmyk, rgb2yiq, filter, cjpeg, djpeg, autocor, conven, bital, fft, viterbi (**)	-	Traces generation with VIRAM vectorizing compiler and ISA simulator; trace-driven simulator
SCALE [9]	-	adpcm	rgbcmy, rgbyiq, hpg, text, dither, rotate, lookup, ospf, pktflow, pntrch, fir, fbital, fft, viterb, autocor, conven	rijndael, sha, qsort, li	VTU code hand-written, execution-driven simulator; C compiler under development

(*) Two applications from Raw benchmark suite

(**) No floating point operations in these benchmarks

4. Evaluation and metrics

WaveScalar – In the evaluation of the WaveScalar architecture, a WaveCache with 16 element domain was used (~2K PEs, unbounded input queues, perfect L1 data caches) [1]. WaveCache performance is measured relative to a superscalar (15 pipeline stages, 16-wide out-of-order processing core, 1024-entry issue window, g-share branch predictor, store-buffer, perfect memory system) and TRIPS processors. Results are reported through AIPC metric (Alpha-equivalent Instructions Per Cycle), which doesn't include instructions added by Alpha-to-WaveScalar binary-rewriter, specific for WaveScalar ISA – just instructions from original Alpha binary are counted in order to have a fair comparison. WaveCache outperforms the superscalar in every benchmark application (average factor 3.1). Compared to TRIPS, WaveCache in some applications has better and in some worse results, ranging from factor 2.5 in IPCs in favor of WaveCache to factor 2 better results in favor of TRIPS. Also, the effects of domain size, cache capacity, input queue size, and control memory speculations on performance of WaveCache are evaluated.

TRIPS – The three supported modes (D-morph, T-morph, S-morph) are evaluated separately [3]. D-morph performance is measured in IPCs, and configurations with different number of A-frames (1 to 32) were used (A-frames are several frames in which one block is mapped). Deeper speculation is possible with a higher number of A-frames. For some benchmarks, IPC number has its peak with 8 or 16 A-frames. Almost every of these configurations outperform Alpha 21264 (from factor 2 to factor 7 on average). In T-morph, several benchmarks are executed together as a multithreaded workload (in D-morph they are executed as single threads). It is

considered that T-morph has same efficiency as D-morph (100%) when it gets the same IPC. From the experiments, T-morph has an efficiency ranging from 80-100% for 2 threads to 39% for 8 threads, due to inter-thread contention, pollution and reduced number of available frames. Still T-morph estimated speedup is from 1.4 to 2.9, because it executes more applications in parallel with the assumption is that each application has approximately the same running time. S-morph has higher IPC by factor 2.4 on average than D-morph. Technique of revitalization for execution of loops is used in this evaluation. In this technique, the instructions of the iteration block are not removed from the A-frame until all the iterations are completed (without it performance of S-morph drops by factor 5 on average).

In the early stages of the project it wasn't feasible to test large number of applications directly, so the evaluation of **Smart Memories** [4] architecture is presented through the evaluation of two of its mappings, Imagine [10] and Hydra [11]. These two architectures were chosen because they have very different memory systems and arrangement of computational resources, and thus show reconfigurable power of Smart Memories. For Imagine mapping, performance degradation is from 20% to 80%, mostly because of fewer functional units. In Hydra mapping, performance degradation is from 10% to nearly 80%, mostly because cache latencies and algorithmic modifications (some hardware structures doesn't exists in Smart Memories).

Synchroscalar – This architecture is evaluated for its capability to reduce power consumption given a certain performance targets of chosen applications [5]. Depending on the application, configurations from 16 to 64 PEs are used. Power consumption ranges from 8 to 30 times worse then ASICs and from 10 to 60 times better than DSPs.

Raw – This processor is compared to Intel Pentium III (P3) [7]. Die areas are 331mm^2 and 106mm^2 for Raw and P3 respectively, while frequencies are 425MHz for Raw and 600MHz for P3. The reference processor has 16 tiles and it outperforms P3 for almost all benchmark applications (ILP computation – 2x for low ILP and 2-9x for high ILP; stream computation, bit-level computation and server tasks 10-100x better results). Speedup is measured in both numbers of cycles and time spent. In some benchmarks, P3 is outperformed even by a Raw with a smaller number of tiles.

CODE – The designers of this architecture compared it with a previous architecture (VIRAM, vector processor with centralized vector register file) [8]. The same scalar cores, memory system, computational capabilities, die areas and frequencies are used in both architectures. CODE has worse performance in benchmarks with frequent inter-cluster transfers (filter, see Table 2) or no decoupling chances across clusters (bital, see Table 2), but for most other benchmarks it outperforms VIRAM. Depending on number of lanes used in both architectures, performance growth is from 42% for 1 lane to 21% for 8 lanes. Support of precise exception degrade performance of less than 5% on average with 8 registers per clusters. The proposed techniques for memory latency tolerance reduce (32 cycles latency) lead to performance loss less than 25%, while VIRAM demands 8 cycles latency.

SCALE – various configurations (with 1, 2, 4, 8 lanes) are compared to several processors (AMD Au1100, Philips TM 1300, Motorola PowerPC, VIRAM, BOPS Manta, TI TMS). The metric for EEMBC benchmarks is the iterations per second, and for others benchmarks the total number of cycles. Overall SCALE configuration with 8 lanes has very good results for every

benchmark. In many applications even a configuration with 4 lanes gives also better results than other processors.

5. Advantages and disadvantages – In Table 3 major advantages and disadvantages are described. These characteristics are just those mentioned in papers regarding that particular architecture. Common advantage of all architectures is good scalability. Some architectures are in the early phase of development, so it is hard to look at all possible disadvantages.

Table 3. – Advantages and disadvantages

	Advantages	Disadvantages
WaveScalar [1]	good initial performance characteristics and parallelism exploiting	many open questions (interrupts handling, I/O)
TRIPS [3]	Exploiting three types of parallelism efficiently	open questions about interface between software and reconfigurable hardware
Smart Memories [4]	general model; architecture can be configured to match the structure of applications	performance will always be worse than on machines optimized for special applications
Synchroscalar [5]	power saving	tied to performance targets, not highest performance possible; for signal processing applications
RAW [6]	good scalability; high bandwidth; good performance for many different applications	
CODE [8]	good scalability; support for precise exceptions; possible multi-lane organization	inter-cluster transfers can reduce performance; not all VFUs can perform all operations
SCALE [9]	Efficient execution of all kinds of loops and multithreaded execution	

6. References:

[1] Steven Swanson, Ken Michelson, Andrew Schwerin, Mark Oskin "WaveScalar", in the 36th International Symposium on Microarchitecture (MICRO-36 2003)

[2] Steve Swanson, Andrew Schwerin, Andrew Petersen, Mark Oskin and Susan Eggers "Threads on the Cheap: Multithreaded Execution in a WaveCache Processor", in the Workshop on Complexity-effective Design (WCED) held in conjunction with the 31st Annual International Symposium on Computer Architecture (ISCA), June 2004

[3] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler Charles R. Moore "Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture", in the 30th Annual International Symposium on Computer Architecture

[4] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, Mark Horowitz, "Smart Memories: A Modular Reconfigurable Architecture", in International Symposium on Computer Architecture, June 2000

[5] John Oliver, Ravishankar Rao, Paul Sultana, Jedidiah Crandall, Erik Czemikowski, Leslie W. Jones IV, Diana Franklin, Venkatesh Akella, Frederic T. Chong, "Synchroscalar: A Multiple Clock Domain, Power-aware, Tile-based Embedded Processor", in 31st International Symposium on Computer Architecture, Munich, Germany, June 2004

[6] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman

Amarasinghe, Anant Agarwal "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", IEEE Micro, Mar/Apr 2002.

[7] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, Anant Agarwal, "Evaluation of the Raw Microprocessor: An Exposed Wire Delay Architecture for ILP and Streams", in Proceedings of the International Symposium on Computer Architecture, June 2004

[8] Christos Kozyrakis, David Patterson, "Overcoming the Limitations of Conventional Vector Processors" in Proceedings of the 30th International Symposium on Computer Architecture (ISCA), June 2003

[9] Ronny Krashinsky, Christopher Batten, Mark Hampton, Steve Gerding, Brian Pharis, Jared Casper, Krste Asanovic, "The Vector-Thread Architecture", in 31st International Symposium on Computer Architecture (ISCA-31), Munich, Germany, June 2004

[10] S. Rixner, et al. "A Bandwidth-Efficient Architecture for Media Processing", in Proceedings of the 31st Annual International Symposium on Microarchitecture, pages 3-13, Nov.-Dec. 1998

[11] K. Olukuton, et al. "Improving the Performance of Speculatively Parallel Applications on the Hydra CMP", in Proceedings of the 1999 ACM International Conference on Supercomputing, June 1999

[12] C. Kim, D. Burger, and S.W. Keckler, ”An Adaptive, non-uniform cache structure for wire-delay dominated on-chip caches”, In 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 211-222, October 2002.

[13] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “Mediabench: A tool for evaluating and synthesizing multimedia and communication systems”, in International Symposium on Microarchitecture, 1997.

[14] http://wavescalar.cs.washington.edu/index.html

[15] http://www.cs.utexas.edu/users/cart/trips/

[16] http://velox.stanford.edu/smart_memories/

[17] http://cag.csail.mit.edu/raw/

[18] http://www.cag.lcs.mit.edu/scale/

[19] http://www.spec.org

[20] http://www.eembc.org

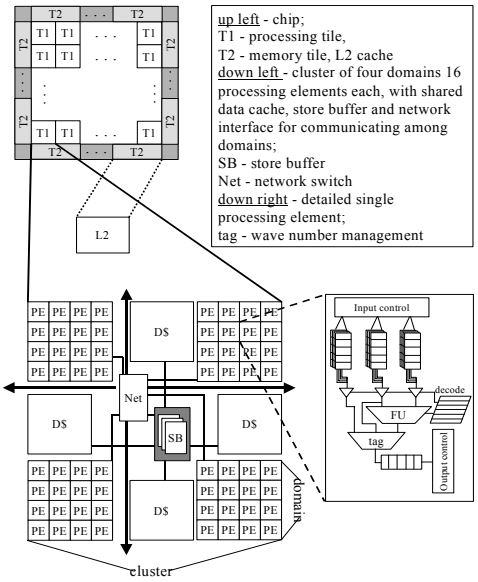

Figure 1. – WaveScalar architecture

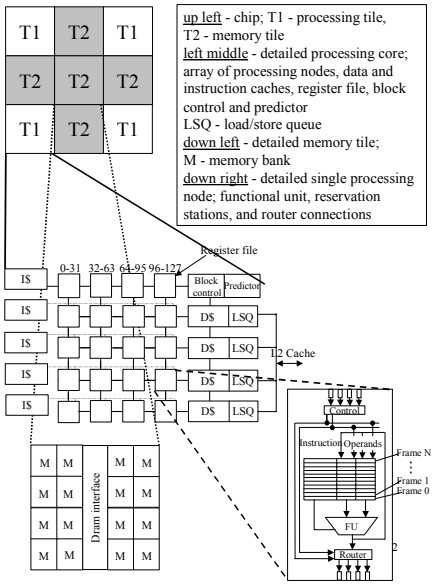

Figure 2. – TRIPS architecture

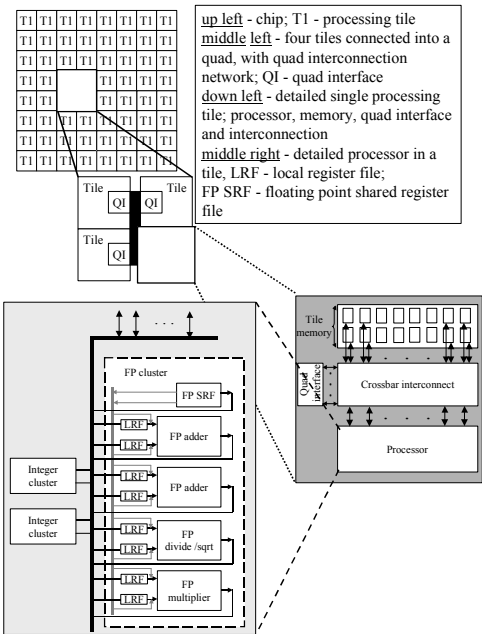

Figure 3. – Smart Memories architecture

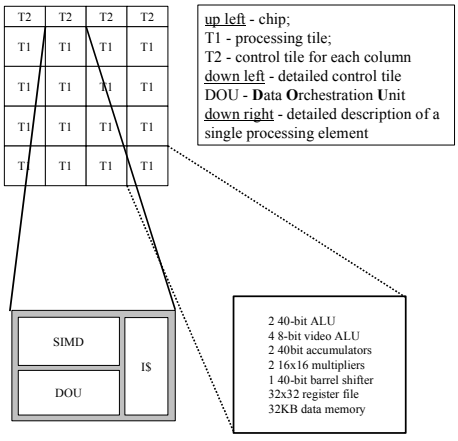

Figure 4. – Synchroscalar architecture

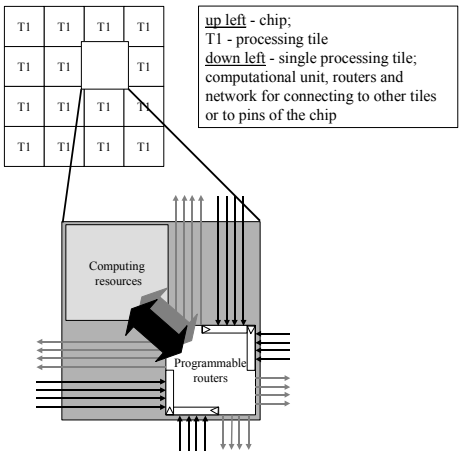

Figure 5. – Raw architecture

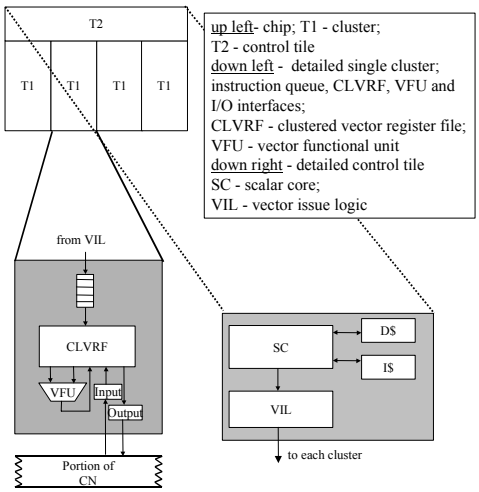

Figure 6. – CODE architecture

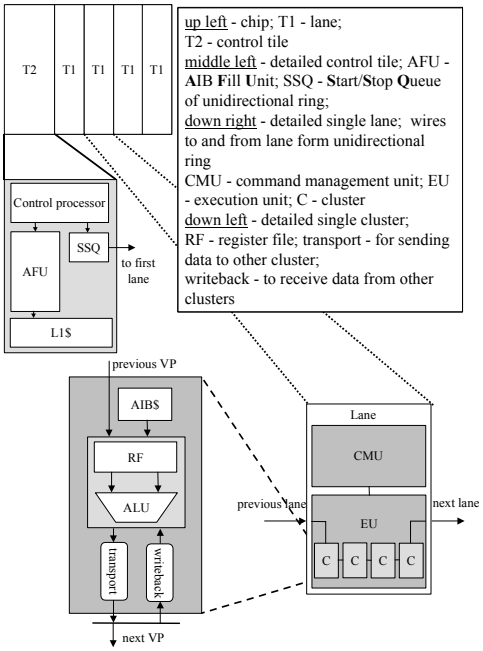

Figure 7. – SCALE architecture