TERAFLUX: Exploiting Dataflow Parallelism in Teradevices

Roberto Giorgi
Universita' degli Studi di Siena
Via Roma 56, Siena, ITALY
http://www.dii.unisi.it/~giorgi

ABSTRACT

The TERAFLUX project is a Future and Emerging Technologies (FET) Large-Scale Project funded by the European Union. TERAFLUX is at the forefront of major research challenges such as programmability, manageable architecture design, reliability of many-core or 1000+ core chips. In the near future, new computing systems will consist of a huge number of transistors - probably 1 Tera or 1000 billions by 2020: we name such systems as "Teradevices". In this project, the aim is to solve the three challenges at once by using the dataflow principles wherever they are applicable or make sense in the general economy of the system. An Instruction Set Extension (ISE) for the x86-64 is illustrated. This ISE supports the dataflow execution of threads.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures---Teradevices; C.1.3 [**Other Architecture Style**]: Dataflow; D.1.3 [**Programming Techniques**]: Concurrent Programming; B.8.0 [**Performance and Reliability**]: General; D.3.4 [**Processors**]: Compilers.

General Terms

Performance, Design, Reliability, Languages.

Keywords

Keywords are your own designated keywords.

1. INTRODUCTION

Most recent updates in the worldwide scenario include the availability of a new type of transistor (3D transistor), which marks the biggest change in the semiconductor industry since 1948 with the introduction of the transistor itself. New materials like Graphene may allow even greater power saving. The technology-node scaling has reached 22nm, with 14nm silicon foundries to be operative by 2013, and it seems the pace will continue at least until 8nm. The 3D layering gives new lymph to the Moore's law too. In this scenario, and in perspective beyond the year 2020, the TERAFLUX project]2] brings together 10 industrial and academic partners to give their best contribution in order to find a common ground to solve at once all the above three challenges.

CF'12, May 15–17, 2012, Cagliari, Italy.
ACM 978-1-4503-1215-8/12/05….$10.00.

The research in this project is inspired by the Dataflow principle. As recalled by Jack Dennis [1], dataflow is "a Scheme of Computation in which an activity is initiated by presence of the data it needs to perform its function". We believe that, if properly exploited, dataflow can enable parallelism which is orders of magnitude greater than what is achievable by control-flow dominated execution models. To investigate our concepts, we are studying dataflow principles at any level of a complete transformation hierarchy, starting from general and complex applications able to load properly a Teradevice through programming models, compilation tools, reliability techniques and architecture.

One key point it is also the evaluation of this system: our choice has been to rely on an existing simulation infrastructure (HPLabs COTSon) that immediately enabled us to start from a nowadays Teradevice (i.e., a 1000+ cluster of nodes, where each node consists of tens of cores) and progressively evolve such system into a more ambitious system where we can gradually remove major bottlenecks. While relying on solid and well-known reference points such as the x86-64 ISA, GCC tools, StarSs programming model and applications, we wish to demonstrate the validity of our research in such common evaluation infrastructure.

Below, we focus on some part of dataflow execution model as proposed by the partner University of Siena with the author's guidance.

2. THE DATAFLOW THREADS

The TERAFLUX system is not forced to follow entirely the dataflow paradigm: in fact, we distinguish among legacy and system-threads (L-, S-threads) and dataflow threads (DF-threads): this will allow for a progressive migration of programs to the new "dataflow paradigm", while accelerating the available DF-threads on the more dataflow-friendly cores [4]. One other important choice is the exploration of synchronization mechanisms such as transactional memory, and the repetition of a thread on a different core by using the dataflow principles [5] in cases when the cores are detected as failing. We can currently afford to run with an acceptable slowdown and accuracy, parallel, scalable, full-system (with unmodified Linux) simulations of 1000+ x86-64 cores [6] while experimenting with very ambitious changes in the execution model implying a major effort to support the execution model based on dataflow threads [3], especially from the compiler point of view.

Moreover, in recent experiments we were able to boot a kind of "datacenter-in-a-box", thanks to a simulation host HP DL-585-G7 with 64 AMD cores and 1 TB of shared memory. The simulation environment has been able to boot about 7000 guest cores, thanks to the off-the-shelf capabilities of COTSon.

3. THE T-STAR ISA EXTENSION

In order to support the execution of DF-Threads, we designed a minimalistic extension of the x86-64 ISA, that we call T-Star (or T-*) [3] shown in Table 1. The key-points of this ISE are: i) it enables an asynchronous execution of threads, that will execute not under the control-flow of the program but under the data-flow of it; ii) the execution of a DF-thread is decided by an core-external component that we call DTS (or Distributed Thread Scheduler) [3]; iii) the types of memory that are used are distinguished in 4 main types (1-to-1 communication or Thread Local Storage, N-to-1 or Frame Memory, 1-to-N or Owner Writable Memory, and N-to-N or Transactional Memory. More details are available in the public deliverables of the project (see http://teraflux.eu).

Table 1: T* Instruction Set Extension (ISE) for the x86-64 ISA

Synopsis	**TSCHEDULE *RS1, RS2, RD***	**TSCHEDULE(<*IP*>, <*SC*>, &<*frame_pointer*>)**
Description	This instruction allocates the resources (a DF-frame of size ***RS2*** words and a corresponding entry in the Distributed Thread Scheduler – or DTS) for a new DF-thread and returns its Frame Pointer (FP) in ***RD***. ***RS1*** specifies the Instruction Pointer (IP) of the first instruction of the code of this DF-thread and ***RS2*** specifies the Synchronization Count (SC).	
Notes	The allocated DF-thread is not executed until its SC reaches 0. The TSCHEDULE can be conditional or non-conditional based on the value stored in the zero flag. If the zero flag is set to 1 then the TSCHEDULE will take effect, otherwise it is ignored.	
Synopsis	**TDESTROY**	**TDESTROY**
Description	The thread that invokes TDESTROY finishes and its DF-frame is freed, (the corresponding entry in the Distributed Thread Scheduler is also freed).	
Synopsis	**TWRITE *RS, RD, offset***	***(<frame_pointer> + <offset>) = (<source_register>)**
Description	The data in ***RS*** is stored into the DF-frame pointed to by ***RD*** at the specified offset.	
Notes	***Side Effect***: The Distributed Thread Scheduler decrements the SC of the corresponding DF-thread entry (located through the FP): $SC_{FP} = SC_{FP}-1$	
Synopsis	**TREAD *offset, RD***	**(<destination_register>) = *(<self_frame_pointer> + <offset>)**
Description	Loads the data indexed by 'offset' from the self (current thread) DF-frame into ***RD.***	
Notes	***Assumption***: the DTS has to load into the register implicitly used by TREAD the value <self_frame_pointer>. In a x86-64 implementation, we can reserve RAX for this purpose.	
Synopsis	**TALLOC RS1, RS2, RD**	**<pointer> = TALLOC (<size>, <type>)**
Description	Allocates a block of memory of ***RS1*** words. The pointer to it is stored in ***RD***. ***RS2*** specifies the special purpose memory type.	
Notes	The Distributed Thread Scheduler tracks the memory allocated. An implementation can code <type> in the 2 LSBs of <size>	
Synopsis	**TFREE *RS***	**TFREE(<pointer>)**
Description	Frees memory pointed to by ***RS***.	
Notes	The Distributed Thread Scheduler tracks the memory deallocated.	

4. ACKNOWLEDGMENTS

This work was partly funded by the European FP7 projects TERAFLUX id. 249013 http://www.teraflux.eu.

5. REFERENCES

[1] J. Dennis, "The Data Flow Concept Past, Present and Future", DFM-2011: Data-Flow Execution Models for Extreme Scale Computing, Oct. 2011

[2] R. Giorgi, "TERAFLUX: Ideas for the Future Many-Cores", ODES: Workshop on Optimizations for DSP and Embedded Systems, Apr. 2011, pp. 38-38.

[3] A. Portero Z. Yu, R. Giorgi, "T-Star (T*): An x86-64 ISA Extension to support thread execution on many cores", HiPEAC ACACES-2011, ISBN:978 90 382 17987, Fiuggi, Italy, July 2011, pp. 277-280.

[4] Z. Yu, A. Righi, R. Giorgi, "A Case Study on the Design Trade-off of a Thread Level Data Flow based Many-core Architecture", Future Computing, ISBN:978-1-61208-154-0, Rome, Italy, Sept. 2011, pp. 100-106, Best paper award.

[5] S. Weis, A.Garbade, J. Wolf, B. Fechner, A. Mendelson R. Giorgi, T. Ungerer, "A Fault Detection and Recovery Architecture for a Teradevice Dataflow System", DFM-2011: Data-Flow Execution Models for Extreme Scale Computing, Oct. 2011, pp. 39-45.

[6] A. Portero, A. Scionti, Z. Yu, P. Faraboschi, C. Concatto, L. Carro, A. Garbade, S. Weis, T. Ungerer, R. Giorgi, "Simulating the Future kilo-x86-64 core Processors and their Infrastructure", 45th Annual Simulation Symp. (ANSS12), Orlando, FL, Mar 2012.