GPGPU-Accelerated Instruction Accurate and Fast Simulation of Thousand-core Platforms

Shivani Raghav‡ Martino Ruggiero†‡, David Atienza ‡,Christian Pinto†, Andrea Marongiu†, Luca Benini†
ESL - EPFL, Lausanne, CH. (‡) {shivani.raghav, martino.ruggiero, david.atienza}@epfl.ch
DEIS - University of Bologna, Bologna, IT. (†)
{christian.pinto, a.marongiu, luca.benini}@unibo.it

***Abstract*— Future architectures will feature hundreds to thousands of simple processors and on-chip memories connected through a network-on-chip. Architectural simulators will remain primary tools for design space exploration, performance (and power) evaluation of these massively parallel architectures. However, architectural simulation performance is a serious concern, as virtual platforms and simulation technology are not able to tackle the complexity of 1,000-core future scenarios. The main contribution of this paper is the development of a simulator for 1,000-core processors which exploits the enormous parallel processing capability of low-cost and widely available General Purpose Graphic Processing Units (GPGPU). We demonstrate our GPGPU simulator on a target architecture composed by several cores (i.e. ARM ISA based), with instruction and data caches, connected trough a Network-on-Chip (NoC). Our experiments confirm the feasibility of our approach. Currently, our ongoing work is focused on developing the power models within the simulation engine.**

I. Introduction and Related Work

During last decade the design of integrated architectures has indeed been characterized by a paradigm shift: boosting clock frequencies of monolithic processor cores has clearly reached its limits, and designers are turning to multicore architectures to satisfy the growing computational needs of applications within a reasonable power envelope.

Hardware designers will be soon capable to create integrated circuits with thousands of cores and a huge amount of on-chip fast memory. Future architectures will expose a massive battery of parallel processors and large on-chip memories connected through a network-on-chip, which speed is more than hundred times faster than the off-chip one [1].

It is clear that current virtual platform technologies are not able to tackle the possible issues coming by the complexity derived by simulating this future scenario, because they suffer problems of either performance or accuracy. Simulators for architectural explorations are quite accurate [2] [3] [4], but they are not adequate for simulating large systems as they are slow. On the contrary, high level simulation technologies can provide good performance for software development [5] [6], but can not enable accurate design space explorations because they are lacking of low level architectural details. Another simulation technique makes usage of parallel machines [7] [8], but they require multiple processing nodes to increase the simulation rate.

Moreover, none of the current simulators takes advantage of the computational power provided by modern manycores, like General Purpose Graphic Processing Units (GPGPU) [9]. The development of computer technology brought an unprecedented performance increase along with these new architectures. They provide both scalable computation power and flexibility, and they have already being adopted for many computational intensive applications. However, in order to obtain the highest performances on such a machine, the programmer has to write programs that best exploit the hardware architecture.

The main novelty of this paper is the development of fast and accurate simulation (i.e. at instruction level) technology targeting extremely parallel embedded systems (i.e. composed by 1000-cores) by specifically taking advantage of the inherent parallel processing power available in modern GPGPUs. The simulation of manycore architectures exhibits indeed a high level of parallelism and is inherently parallelizable. The large number of threads that can be computed in parallel on a GPGPU can be employed to perform such a large number of core simulations in parallel. Clearly, we developed a new simulation technology to deploy the parallel simulation of 1000-core full systems on top of GPGPUs. The simulated architecture is composed by several cores (i.e. ARM ISA based), with instruction and data caches, connected trough a Network-on-Chip (NoC). Our experiments indicate the feasibility and goodness of our idea and approach, since our simulator can simulate architectures composed by thousand of cores and provide fast simulation time and good scalability. Current effort is devoted to the modelling of the power consumption of the target simulated architecture.

II. Target Architecture

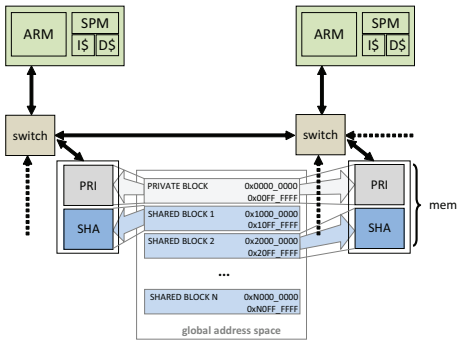

Fig. 1: Target simulated architecture.

The platform template targeted by this work and our simulator is the manycore depicted in Fig.1. The platform consists of a scalable number of homogeneous processing cores, a shared communication infrastructure and a shared memory for inter-tile communication. The main architecture is made by several computational tiles composed by a ARM-based CPU. Processing cores embed instruction and data caches and are directly connected to tightly coupled software controlled scratch-pad memories. Each computational tile

also features a bank of private memory, only accessible by the local processor, and a bank of shared memory. Interaction between CPUs and memories takes place through a Network-on-Chip communication network (NoC).

III. Full Simulation Flow

The entire simulation flow is structured as a single CUDA kernel [9], whose simplified structure is depicted in Fig. 2. One physical GPU thread is used to simulate one single target machine processor, its cache subsystem and the NoC switch to which it is connected. The program is composed by a main loop – also depicted in the code snippet in Fig. 2 – which we refer to as a *simulation step*.

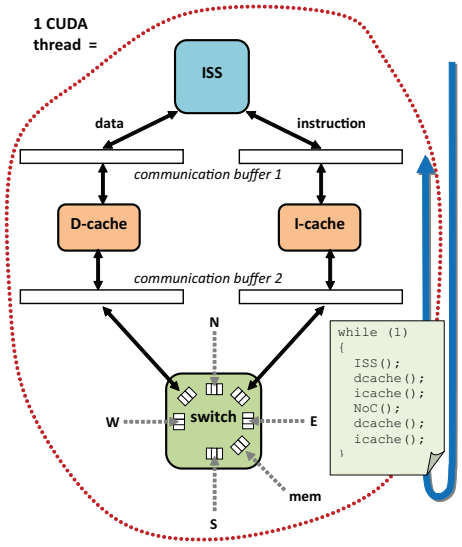

Fig. 2: Main simulation loop.

The `ISS` module is executed first. During the *fetch* phase and while executing `LOAD/STORE` instructions the core issues memory requests to the `Cache` module, which is executed immediately after. *Communication buffer 1* is used to exchange information such as target address and data.

The `Cache` module is in charge of managing data/instructions stored in the private memory of each core. The shared segment of each core's memory is globally visible through the entire system. Shared regions are not cacheable. The cache simulator is also responsible for forwarding access requests to shared memory segments to the `NoC` simulator. Upon cache miss there is also the necessity to communicate with the NoC. This is done through *communication buffer 2*. For a LOAD operation (that does not hit in cache) to complete there is the need to wait for the request to be propagated through the NoC and for the response to travel back.

IV. Experimental Results

We investigate the performance of our simulator with three real-world program kernels, which are widely adopted in several application from the embedded domain, namely *Matrix Multiplication*, *IDCT* and *FFT*. We adopt an OpenMP-like parallelization scheme to distribute work among available cores. An identical number of iterations is assigned to parallel threads. The dataset touched by each thread is differentiated based on the processor ID. The metric we adopt to test simulation speed is Million-Instructions Per Second

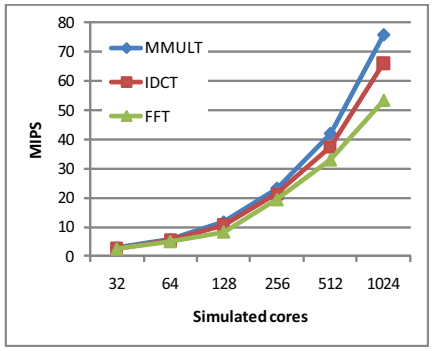

Fig. 3: Simulated MIPS for the three benchmarks.

(MIPS), calculated as total simulated instructions divided by wall clock time of the host.

We show in Fig. 3 the simulated MIPS for each benchmarks.It is possible to notice that our simulation engine scales well for all the considered programs. *IDCT* and *MMULT* exhibit a high degree of data parallelism, which results in a favorable use case for our simulator since a very low percentage of divergent branches takes place. *FFT*, on the other hand, features data-dependent conditional execution, which significantly increases control flow divergence.

V. Conclusion

In this paper, we present a novel approach that represents an important first step towards the simulation of manycore chip of an arbitrary number of cores. The presented simulation infrastructure for massive parallel embedded architectures exploits the high computational power of modern GPGPUs. Our experiments indicate that our approach can scale up to thousand of cores architecture providing fast simulation time and good accuracy. This work highlights important directions in building a comprehensive tool to simulate performance and power of many-core architectures that might be very helpful for the future research in computer architecture.

Acknowledgement

This research has been partly supported by the Swiss National Science Foundation (SNF), grant number 200021-130048 and by the PRO3D EU FP7-ICT-248776 project.

References

[1] Intel Corp., "Single-chip Cloud Computer ."
[2] T. Grotker, *System Design with SystemC*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.
[3] M. Ruggiero, F. Angiolini, F. Poletti, D. Bertozzi, L. Benini, and R. Zafalon, "Scalability analysis of evolving SoC interconnect protocols," in *In Int. Symp. on Systems-on-Chip*, 2004, pp. 169–172.
[4] The Open Virtual Platforms (OVP) portal, "http://www.ovpworld.org/."
[5] QEMU, "http://wiki.qemu.org."
[6] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.
[7] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1 –12.
[8] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, april 2004, p. 78.
[9] *NVIDIA CUDA Programming Guide*, 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf