Asymmetric Computation for Speculative Heterogeneous HPC

Lorenzo Altamura
DIAG
Sapienza, University of Rome
Rome, Italy
altamura.1538468@studenti.uniroma1.it

Stefano Conoci
DIAG
Sapienza, University of Rome
Rome, Italy
conoci@diag.uniroma1.it

Alessandro Pellegrini
DIAG
Sapienza, University of Rome
Rome, Italy
pellegrini@diag.uniroma1.it

***Abstract*—HPC applications on future exascale systems will demand for runtime environments able to transparently manage the complexity of the underlying heterogeneous hardware. In this abstract, we discuss a computation model for speculative HPC applications, able to deliver non-minimal performance increase and significant energy savings. This model can be easily adapted to multiple heterogeneous hardware families with minor effort, and it can autonomically and promptly reassign units of work to different hardware classes. Our design jointly targets performance and energy efficiency. We also provide a preliminary experimental evaluation of our design.**

***Index Terms*—Speculative Processing, High Performance Computing, Heterogeneous Computing, Runtime Environments**

I. Introduction

The international effort to target emerging pre-exascale and future exascale-class HPC systems has already identified fundamental aspects to make an effective usage of the computing resources from an application point of view [1]. Some of them relate to: i) autonomicity in the management of the resources provided by HPC systems; ii) the role of runtime environments, which should understand application workload to fine tune the usage of the underlying hardware to increase performance and efficiency; iii) speculation, as a means to capture the degree of parallelism of applications, and make a better usage of the available computational power.

Speculation has been shown in the literature to effectively enable a performance increase of a wide range parallel/ distributed HPC applications. In particular, it has been fruitfully used in the contexts of Parallel Discrete Event Simulation (PDES) [2], Transactional Memory [3], and task-based fine-grain parallel applications [4]. In these contexts, units of work (e.g., transactions, events, tasks) are carried out independently of their *safety*, i.e. they might temporarily bring the execution path to an inconsistent state, which is detected a-posteriori and undone (e.g., by means of transactional aborts or rollbacks).

Future exascale systems are expected to be based upon *heterogeneous* infrastructures, i.e., composed of different families of hardware such as CPUs, GPUs, FPGAs, and a wide range of coprocessors. Heterogeneous infrastructures are inherently *asymmetric*, in the sense that different hardware shows a different performance/energy tradeoff in the execution.Nowadays, this asymmetry is also present in CPU-only systems where ISA-diverse cores are employed, or differentiated per-core voltage and frequency regulations are necessary to keep the chip temperature in the safe operating range [5].

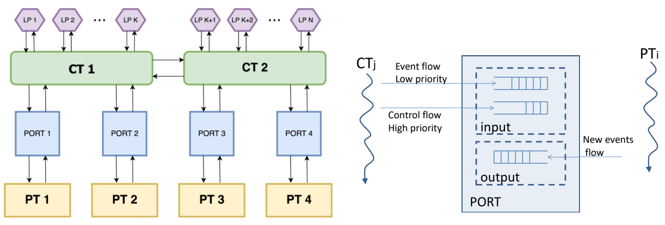
At the same time, most massively-parallel runtime environments used to support the execution of HPC applications are designed to be inherently *symmetric* (see, e.g., [6], [7]). Symmetry in their design refers to the threads of execution which carry out similar activities, independently of the hardware on which they run or the load. Given the various performance/ energy trade-offs provided by heterogeneous architectures, this approach could be sub-optimal.

In particular, there is still no consensus on a viable path to effectively exploit speculative computation on heterogeneous architectures, from a performance and energy efficiency standpoint. This is an important open question, because speculation creates additional and interesting opportunities to optimize the execution of HPC applications, especially in the case of autonomic self-tuning capabilities transparently offered by runtime environments.

We therefore envisage a completely different design of speculative runtime environments, able to autonomically capture and exploit the intrinsic capabilities of these emerging hardware paradigms. We present in this abstract a general software architecture for speculative applications on heterogeneous systems, which has been devised with portability as a central goal. We propose an *asymmetric* runtime environment inspired by the intrinsic asymmetrical structure of heterogeneous machines where threads are specialized to perform different operations. Housekeeping operations, i.e. those which are necessary only to enforce correctness in the speculative execution, are moved off the critical path of threads which carry out units of work associated with the applications. In this way, coprocessors, GPUs, and/or FPGAs could be used to carry out tasks associated with the HPC application in a more efficient way, while CPUs are dedicated to both the orchestration/autonomic management, and processing. This scheme allows to fine tune the allocation of hardware resources to the different contributors that affect the application progress, allowing to increase the overall performance, reduce the energy footprint, or both at the same time. We present preliminary results for PDES applications, but we plan in the near future to generalize our design and adapt it also for other application domains.

II. The Proposed System Model

In the context of speculative runtime environments for HPC, the de-facto architectural standard uses symmetric threads which carry out the same activities, yet related to two different classes of tasks. *Class-1* encloses forward mode processing of



(a) Runtime Architecture (b) Communication Port

Fig. 1: Asymmetric System Model

application-related units of work, while *Class-2* encompasses all other tasks, related to housekeeping operations and autonomic tuning of the application. In our asymmetric model, we map different classes to different *incarnations* of threads. Fig. 1a shows the overall re-organization of the runtime environment which we propose. Controller threads (CTs) are in charge of governing the lifetime of the application (in terms of scheduling and correctness control), from all perspectives: energy efficiency, performance, self-tuning, etc. Conversely, Processing threads (PTs) are much simpler and process units of work. Their simplicity makes them perfectly suitable to offload them also on specialized or extremely parallel hardware, such as FPGAs or GPUs. CTs and PTs can be in any number in the system, and their *rebalancing* is in charge of the CTs. A number of PTs can be bound to a single CT, which will dispatch to its PTs units of work.

A core aspects is related to the communication between CTs and PTs. We base our model on the concept of *communication ports* (as depicted in Fig. 1b) with multi-priority queues. The CT uses the low-priority queue to dispatch units of work to the PTs. The high-priority queue is used to notify PTs of priority inversions, leading PTs to discard no-longer consistent units of work. The output queue can be used to notify CTs of newly-generated units of work, which might be dispatched at a later time to different PTs. The decoupling between CTs and PTs based on communication ports make it extremely easy to migrate PTs to different hardware architectures, while leaving untouched the logic associated with the management of computation and resources.

We then enforce a *2-level autonomic reorganization*. On the one hand, we determine the proper number of CTs and PTs in the system, and the hardware on which PTs are run (long-term self-tuning). Possibly, we can decide not to use all the available cores, depending on the current workload. On the other hand, we control the degree of speculation by determining the best-suited number of units of work to be injected in every PT's port (short-term self-tuning). In this way, we can maximize performance while minimizing energy consumption, by identifying the best-suited trade-off between speculation and efficiency, i.e. minimizing the amount of work wasted by speculation. Therefore, this approach allows to fine-tune the amount of speculation based on the peculiarities of the different hardware families, such as the speed of forward mode processing or the degree of parallelism.

III. Preliminary Evaluation and Future Work

We present results associated with performance and energy efficiency when running speculative PDES simulations [2] on

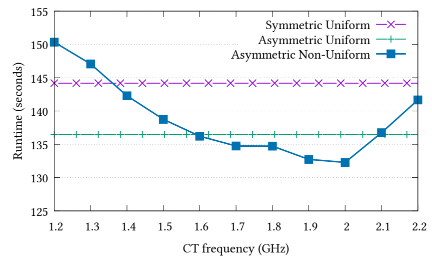


Fig. 2: Performance under power cap

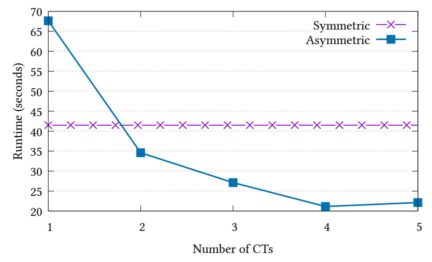


Fig. 3: Effect of thread rebalancing

top of heterogeneous (in terms of frequency) CPUs. Experiments have been run using the PHold benchmark on a 10 core machine equipped with Intel Xeon E5-2630 v4 and 256 GB of ECC memory. In Fig. 2 we show the variation of the execution time with power cap set to 30 Watt for the classical symmetric architecture, with cores uniformly slowed to meet the power budget, and our asymmetric architecture. The CPU frequency ranges from 1.2 GHz at P-state 11 to 2.2 GHz at P-state 1. This test was used to assess the efficiency gains achievable by the asymmetrical paradigm compared to a traditional symmetrical system. In Fig. 3 we show the effect on performance when relying on a different number of CTs, thus showing the importance of an autonomic rebalancing strategy to fine tune the resource allocation when running on heterogeneous systems. In both scenarios, our architecture is able to reduce the completion time in several configurations, also being able to meet the imposed power cap. We are currently finalizing the implementation of PTs on FPGAs and GPUs.

References

[1] J. Dongarra, P. Beckman *et al.*, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, 2011.

[2] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and System*, vol. 7, no. 3, pp. 404–425, 1985.

[3] N. Shavit and D. Touitou, "Software transactional memory," in *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, 1995.

[4] J. F. Martínez and J. Torrellas, "Speculative synchronization: Applying thread-level speculation to explicitly parallel applications," in *Operating Systems Review (ACM)*, 2002.

[5] H. Esmaeilzadeh, E. Blem *et al.*, "Dark Silicon and the End of Multicore Scaling," *IEEE Micro*, vol. 32, no. 3, pp. 122–134, may 2012.

[6] C. D. Carothers, D. W. Bauer, and S. Pearce, "ROSS: a High Performance Modular Time Warp System," in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, 2000, pp. 53–60.

[7] A. Pellegrini and F. Quaglia, "The ROme OpTimistic Simulator: A tutorial," in *Proceedings of the Euro-Par 2013: Parallel Processing Workshops*, ser. PADABS, D. an Mey, M. Alexander *et al.*, Eds. LNCS, Springer-Verlag, 2014, pp. 501–512.