A Brief Overview on Runtime-Aware Architectures

Marc Casas, Miquel Moretó, Eduard Ayguadé, Jesus Labarta, Mateo Valero
Barcelona Supercomputing Center
Jordi Girona, 29, Nexus II Building, Barcelona, Spain

When uniprocessors were the norm, Instruction Level Parallelism (ILP) and Data Level Parallelism (DLP) were widely exploited to increase the number of instructions executed per cycle. The main hardware designs that were used to exploit ILP were superscalar and Very Long Instruction Word (VLIW) processors. The VLIW approach implies statically figuring out dependencies between instructions and scheduling them. However, since it is not possible in general to obtain optimal scheduling's at compile time, VLIW does not fully exploit the potential ILP that many workloads have. Superscalar designs try to overcome the increasing memory latencies, the so called *Memory Wall* [8], by using Out of Order (OoO) and speculative executions [3]. Additionally, techniques such as prefetching, to start fetching data from the memory ahead of time, deep memory hierarchies, to exploit the locality that many programs have, and large reorder buffers, to increase the number of speculative instructions exposed to the hardware, have been also used to enhance superscalar processors performance. DLP is typically expressed explicitly at the software layer and it consisted in a parallel operation on multiple data performed by multiple independent instructions, or by multiple independent threads. In uniprocessors, the Instruction Set Architecture (ISA) was in charge of decoupling the application, written in a high-level programming language, and the hardware. In this context, the architecture improvements were applied at the pipeline level without changing the ISA.

Some years ago, the traditional ways to keep increasing hardware performance at the rate predicted by Moore's Law vanished. Additionally to the memory wall, the processor clock frequency stagnated because, when it increased beyond a threshold, the power per unit of area (power density) could not be dissipated. That problem was called the *Power Wall* [5]. A study made by the International Technology Roadmap for Semiconductors predicts an annual frequency increase of 5% for the next 15 years [4]. That means that we are left with parallelism alone in order to further increase performance.

To overcome the stagnation of the processor clock frequency, vendors started to release devices with multiple cores over a decade ago. Multi-core designs can potentially provide the desired performance gains by exploiting Task Level Parallelism (TLP). However, multi-cores, rather than fixing the problems associated with the memory and power walls, exacerbate them. The ratio cache storage per operation stagnates or decreases in multi-core designs, as well as the memory bandwidth per operation does, making it very hard to fully exploit the throughput that multi-core designs have. Another major concern is energy consumption, since if it keeps growing with the same rate as today, some major technological challenges like designing exascale supercomputers or developing petaflop mobiles will become chimeras. This set of challenges related to power consumption issues constitutes a new power wall.

Additionally, multi-core systems might have a heterogeneous set of processors with different ISA's, connected through several layers of shared resources with variable access latencies and distributed memory regions. To manage data motion among this deep and heterogeneous memory hierarchy while properly handling Non-Uniform Memory Access (NUMA) effects and respecting stringent power budget in data movements is going to be a major challenge in future multi-core machines. All these problems regarding programmability and data management across the memory hierarchy are commonly referred as the *Programmability Wall* [1]. Multi-core architectures can theoretically achieve significant performance with low voltages and frequencies. However, as the voltage supply scales relative to the transistor threshold voltage, the sensitivity of circuit delays to transistor parameter variations increases remarkably, which implies that processor faults will become more frequent in future designs. Additionally, the fact that the total number of cores in future designs will increase in several orders of magnitude only makes the fault prevalence problem more dramatic. In addition to the current challenges in parallelism, memory and power management, we are moving towards a *Reliability Wall* [9].

With the irruption of multi-cores and parallel applications, the simple interface between the hardware and the application started to leak. As a consequence, the role of decoupling again applications from the hardware was moved to the runtime system. This runtime layer is also in charge of efficiently using the underlying hardware without exposing its complexities to the application. In fact, the collaboration between the heterogeneous parallel hardware and the runtime layer becomes the only way to keep the programmability hardship that we are anticipating within acceptable levels while dealing with the memory, power and resilience walls.

Current multi-cores are designed as simple symmetric multiprocessors (SMP) on a chip. However, we believe that this is not enough to overcome all the problems that multi-

cores already have to face. To properly take advantage of their potential, tight hardware-software collaboration is required. It is our position that the runtime has to drive the design of future multi-cores to overcome the challenges of the above mentioned walls. We envision a **Runtime-Aware Architecture (RAA)** [7], a holistic approach where the parallel architecture is partially implemented as a software runtime management layer, and the remainder in hardware. In this architecture, TLP and DLP are managed by the runtime and are transparent to the programmer. The idea is to have a task-based representation of parallel programs and handle the tasks in the same way as superscalar processors manage ILP, since tasks have data dependencies between them and a Task Dependency Graph (TDG) can be built at runtime or statically. In this context, the runtime drives the design of new architecture components to support activities like the construction of the TDG [2], among other things.

In RAA's, there is a great margin for mitigating *Memory* and *Power Walls* by exploiting information encoded in the TDG. For example, the scheduler can detect and exploit temporal **data reuse** and avoid data movement by reordering tasks that reuse the same input data, or that use the outputs of the previous task. Also, it can make decisions about data distribution, **prefetching** [6] data close to where the task will be executed and creating explicit copies for **increased locality** if the same data is required in multiple locations at the same time. That would be done by relying on software coherence or on specific architectural support to propagate updates across the multiple copies of data. Task-based workloads typically have complex critical paths that can be exploited by RAA's as the workload segments that are out of it can tolerate some degree of extra latency without hurting the performance of the whole application, allowing more improvements in terms of power consumption. In general, we expect the runtime to guide and impact the design of future memory hierarchies, which may have hybrid designs composed by caches and local memories. Some work has already been done on runtime-aware prefetching mechanisms and simplified coherence protocols have already been proposed in the literature.

To deal with the *Reliability Wall* in RAA's, an analysis of tasks' resilience and criticality in the face of hard and soft faults can be performed at runtime. That analysis can be based on fault injection experiments or, simpler than that, can be based just in the information contained in the TDG. That would make possible to schedule more critical and less resilient tasks to hardware components with support to carry out some re-computation, to correct wrong computations, and also some ECC integrated in the memory, to make sure that the stored data is not corrupted. Additionally, algorithmic recovery techniques can exploit the natural asynchrony of task-based workloads to reduce the impact they typically have over the performance of the application. This kind of solutions would make possible to develop resilience solutions at the runtime level and, thus, integrate them into the whole RAA approach.

Under the experience of the current ending age and equipped with a mature vision of what a productive future can be, many good ideas that disappeared during the RISC clock frequency boom of the 80's and 90's can be reshaped and applied with unforeseen scales or scope, resulting in an innovative vision of how to address the current embroilment where hardware technology has taken us.

Our approach towards parallel architectures offers a single solution that could alleviate most of the problems we encounter in the current approaches: handling parallelism, the memory wall, the power wall, the programmability wall, and the upcoming reliability wall in a wide range of application domains from mobile up to supercomputers. Altogether, this novel approach towards future parallel architectures is the way to ensure continued performance improvements, getting us out of the technological hardship that computers have turned into, once more riding on Moore's Law.

REFERENCES

[1] B. Chapman. The Multicore Programming Challenge, volume 4847 of Lecture Notes in Computer Science, pages 3-3. Springer Berlin Heidelberg, 2007.

[2] Y. Etsion, F. Cabarcas, A. Rico, A. Ramírez, R. M. Badia, E. Ayguadé, J. Labarta, and M. Valero. Task superscalar: An out-of-order task pipeline. In MICRO, pages 89-100, 2010.

[3] J. L. Hennessy and D. A. Patterson. Computer Architecture - A Quantitative Approach (5. ed.). Morgan Kaufmann, 2012.

[4] International technology roadmap for semiconductors (ITRS), system drivers. In ITRS, 2011.

[5] T. Mudge. Power: A first-class architectural design constraint. Computer, 34(4):52-58, Apr. 2001.

[6] D. Prat, C. Ortega, M. Casas, M. Moretó and M. Valero. Adaptive and application dependent runtime guided hardware prefetcher reconfiguration on the IBM POWER7. In ADAPT 2015.

[7] M. Valero, M. Moreto, M. Casas, E. Ayguade, J. Labarta. Runtime-Aware Architectures: A First Approach. Journal of Supercomputing Frontiers and Innovations, 1(1), 2014.

[8] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. ACM SIGARCH Computer Architecture News, 23(1):20-24, 1995.

[9] X. Yang, Z. Wang, J. Xue, and Y. Zhou. The reliability wall for exascale supercomputing. IEEE Trans. Comput., 61(6):767-779, June 2012.