High performance computing challenges for research and industrial simulation codes

Raphaël Poncet

YIC2013
Second ECCOMAS Young Investigators Conference
2–6 September 2013, Bordeaux, France

High performance computing challenges for research and industrial simulation codes

R. Poncet [a]

[a] CEA-DAM-DIF
F-91297, Arpajon, France

*raphael.poncet@cea.fr

Abstract. *Computing hardware, from mobile devices to supercomputer clusters, is undergoing a paradigm shift : with the advent of multicore CPUs and accelerators (GPUs, Intel Xeon Phi), parallelism is becoming hierarchical, ubiquitous, heterogeneous, and more and more complex. Performance critical applications need new programming models to exploit efficiently such architectures. Guided by Herb Sutter analysis ([1], [2]) we draw practical conclusions for the design of future simulation codes, and the optimization of legacy codes.*

Keywords: High Performance Computing, GPGPU, parallel computing, software development, legacy codes

1 EVOLUTION OF COMPUTING HARDWARE

In a seminal paper, *The free lunch is over* (2004, [1]), Herb Sutter, a leading authority on software development, concurrency and parallelism, analyzed the evolution of mainstream CPUs towards multicore, and discussed its main consequence for performance oriented application development : the need for parallelism, to be able to exploit efficiently this new hardware. A few years later, in the follow-up paper *Welcome to the jungle* (2011, [2]), he predicted the democratization of distributed heterogeneous parallel computing hardware, with the advent, along multicore CPUs, of specialized processors for mainstream computing (e.g. accelerators such as graphical processing units — GPUs —, or Intel Xeon Phi), and the rapid development of cloud computing. His predictions have been remarkably accurate. As a matter of fact, today, every smartphone, tablet, desktop computer is a — potentially distributed — parallel heterogeneous machine. He showed that these three hardware trends (multicore, accelerators, cloud computing) are three aspects of a single trend, which is a consequence of diminishing returns from Moore's Law ([4]). This helps to put into context three phenomena that are not, at first sight, connected to each other, and proves that these changes towards ubiquitous distributed heterogeneous parallelism are permanent.
The High Performance Computing landscape is a bit different : indeed, parallelism has been used to program supercomputers since the very beginning. Distributed memory parallelism has been in wide use since the nineties (mainly with the BSP — Bulk Synchronous Parallelism — paradigm [3], domain decomposition, and message passing using the `MPI` library), and is still the norm today. The development of multicore CPUs did not immmediately cause the democratization of shared memory parallelism (e.g. multithread), since multicore CPUs can be programmed with the message passing paradigm, as long as their number of cores remains modest. Yet, it is clear that Sutter's diagnosis applies readily to present and future supercomputer architectures. Indeed, in today's most powerful supercomputers, a compute node is generally constituted with several dozens of cores, a number that will grow significantly in the years to come. Scalability of `MPI`-only parallelism is not guaranteed anymore, not only for CPU performance, but also — especially for industrial codes — for memory consumption (because distributed parallelism on a shared memory node yield memory overhead). In several supercomputer designs, specialised processors (e.g. GPUs or Intel Xeon Phi) are used, mainly because they provide better compute performance per watt (they are more energy efficient). Specific care is needed for scalability on such processors : for instance, in order to use efficiently one Intel Xeon Phi, application need shared memory parallelism with O(100) scalability, along with SIMD vectorization. For GPUs, specific programming language extensions (`NVIDIA CUDA`, `OpenCL`) or directive

(`HMPP`, `OpenACC`) must be used.

2 CHALLENGES FOR HIGH PERFORMANCE SIMULATION CODES

Having established Sutter's analysis interest for High Performance Computing applications, the consequence he draws for mainstream application also holds : *" a single compute-intensive application will need to harness different kinds of cores, in immense numbers, to get its job done."*. This in mind, we turn on practical consequences for the development of future simulation codes, and the optimization of legacy codes.
It is now clear that, the end of the "free performance lunch" is near for simulation codes. The fact that these codes generally exhibit one level of parallelism (`MPI`) allowed them to sustain the development of mainstream multicore CPUs in the last ten years, but, to efficiently use future processors, they will need, in the years to come, to exhibit several levels of parallelism (e.g. distributed memory, shared memory, and SIMD/vectorization). This is a significant technical problem, especially if we take into account the necessity of portability and sustainability. Using hierarchical domain decomposition (such as nested computational grids) allows to express multiple parallelism at the algorithmic level, which is probably the key to portability.
Applications will also need to handle heterogeneity, e.g. different kinds of cores (a mix of generalist "slow" cores and specialised "fast" cores). This is a challenge for the repartition of workloads between processors. Technical solutions exist (at the runtime level such as StarPU, Dague, or at the library level such as Microsoft AMP, Intel TBB). Again, to provide portability, it is probably necessary to express the structure of a computer program as a graph, e.g. using a task-based programming model.
Finally, load unbalance limits the scalability of a lot of simulation codes even today, a problem which will be magnified in the future. It often occurs when two (or more) numerical methods with contradictory parallel partitioning requirements cohabit in the same application, which is the norm for multiphysics codes. Practical examples include coupled mesh algorithms (stencil codes...) and particle methods (Spherical Particle Harmonics, Particle In Cell...), contact-impact simulations, or multi-mesh Lagrangian (or Arbitrary Lagrangian Eulerian) methods coupled with sliding between meshes. Cures have been proposed (such as, for instance, dual domain decomposition for contact-impact simulations), but they will probably need to be revisited, to improve their scalability.

3 CONCLUSIONS

High performance simulation codes need to face huge challenges, in order to cope with the ever growing complexity of computer hardware. These challenges are transverse, and concern software development, scientific computing, numerical analysis and physical modeling. They need to be tackled now, and prompt interesting problems for research. The problems faced by industrial multiphysics codes, for which load unbalance often causes the main performance bottlenecks, are in general less studied, despite their practical relevance.
In the words of Sutter — which are intended for mainstream applications, but apply perfectly to the high performance computing landscape : *"Mainstream hardware is becoming permanently parallel, heterogeneous, and distributed. These changes are permanent, and so will permanently affect the way we have to write performance-intensive code on mainstream architectures."*

ACKNOWLEDGEMENT

REFERENCES

[1] Sutter, H., The free lunch is over, `http://www.gotw.ca/publications/concurrency-ddj.htm`
[2] Sutter, H., Welcome to the jungle, `http://www.herbsutter.com/welcome-to-the-jungle/`
[3] Valiant, L.G., A bridging model for parallel computation, *Communications of the ACM*, Volume 33 Issue 8, Aug. 1990
[4] Moores law. (2010). In Encyclopdia Britannica. Retrieved August 13, 2010, from Encyclopdia Britannica Online: `http://www.britannica.com/EBchecked/topic/705881/Moores-law`