21 Nordic Seminar on Computational Mechanics
NSCM-21
T. Kvamsdal, K. M. Mathisen and B. Pettersen (Eds)

IMPACT OF HETEROGENEOUS COMPUTERS ON COMPUTATIONAL METHODS

NSCM-21

TROND RUNAR HAGEN AND JOHAN SELAND

SINTEF ICT, Department of Applied Mathematics
P.O. Box 124 Blindern, 0314 Oslo, Norway
e-mail: Trond.R.Hagen@sintef.no, Johan.Seland@sintef.no, web page: http://www.sintef.no

Key words: Heterogeneous Computing, Massive Parallelization, Computational Methods.

Summary. We present a brief overview of heterogeneous computing, and develop a model for an idealized heterogeneous computer based on expected technology trends. Furthermore we discuss the impact of this model on computational and numerical algorithms.

1 INTRODUCTION

A *heterogeneous computer* (HC) is a tightly coupled system of processing units with distinct characteristics. A modern desktop or laptop computer is an example of such a system, as most systems include both a task-parallel, multi-core CPU (Central Processing Unit) and one or more data-parallel processors in the form of programmable GPUs (Graphical Processing Unit). These different units communicate over a fast bus. As of 2008 such a desktop-based HC can deliver around five *teraFLOPS* of single-precision floating point performance, compared to around 100 g*igaFLOPS* for a high-end, quad-core CPU. *Heterogeneous computing* is the strategy of deploying all of these processors within a single workflow. This allows each processor to perform the task to which it is best suited, thereby utilizing all resources in the system and increasing performance.

The HC has emerged as a result of the increased complexity of CPUs. Relatively few of the transistors on a CPU die are dedicated to numerical computing, but are reserved for tasks such as instruction reordering, branch prediction, and memory cache. This makes CPUs suitable for irregular workloads, such as those issued by a general-purpose operating system. Conversely, a data-parallel architecture dedicates most transistors to floating point computations, and handles regular workloads efficiently, such as those encountered in most numerical and graphical computations. Consequently, the highest performance on a given transistor and power budget is achieved by combining both of these approaches in one system, and distributing the workload accordingly.

Future computer systems are expected to couple different types of processing units even more tightly than modern computers. The Cell BE[1] already integrates both task- and data-parallel processors on a single chip, and AMD is expected to follow suit with their *Fusion* processor in 2009. There are also initiatives, such as AMD's *Torrenza* and IBM and Intel's

Geneseo, that will provide very high bandwidth between different types of processing units.

Heterogeneous computers provide high performance at an acceptable price. However, existing mainstream programming languages such as C(++) and Java lack support for heterogeneous computers. Therefore, existing software can not automatically utilize HCs. However, several new languages have been developed, most notably Nvidia's *CUDA*[2], and Apple's *OpenCL*. The latter is currently being standardized by the Khronos Group. Both of these languages are extensions of C(++), and existing software can therefore gradually be migrated to them.

2 THE IDEALIZED HETEROGENEOUS COMPUTER

There is currently a rapid ongoing development of computer architecture, and as described above, the different vendors have not yet converged to a common platform. We now present an idealized HC which will be used as a basis for discussing its impact on computational algorithms. This platform incorporates both processor technology and a memory model based on expected technology trends.

We assume such an HC consist of several (4-10s) general-purpose processors, alongside numerous (100s-10000s) data-parallel processors. Both can process single-precision floating point numbers at twice the speed of double-precision. Our generalized machine has a NUMA (Non-Uniform Memory Architecture) where memory access times depends highly on the memory location relative to a processor. Furthermore, the architecture has a very efficient hardware thread scheduler, making it possible to manage thousands of threads at the same time. This architecture is illustrated in Figure 1.

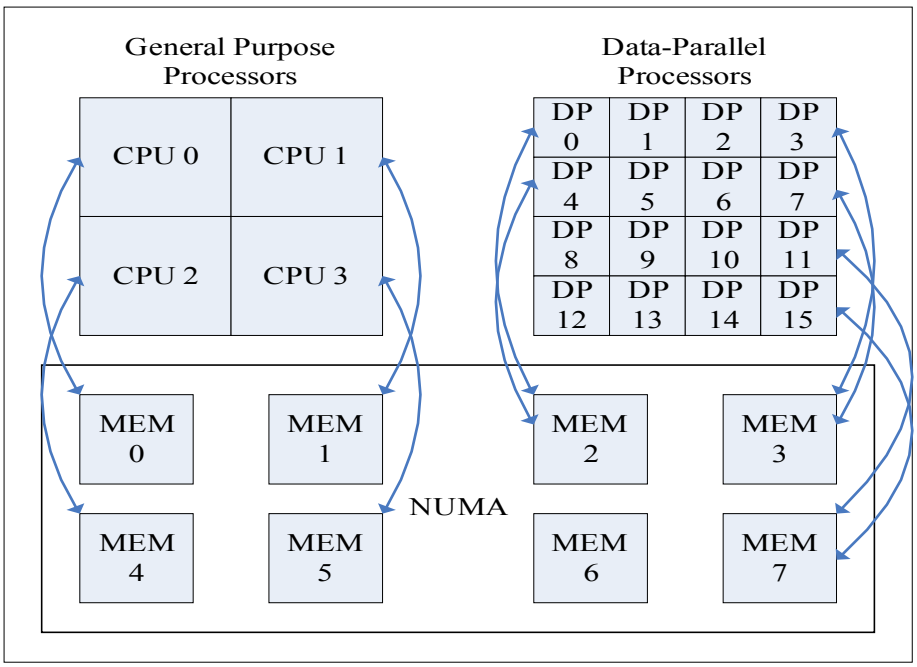

Figure 1: The idealized heterogeneous computer is equipped with several general-purpose processors and numerous data-parallel processors. Each processor has fast access to a local part of the global memory.

3 CONSEQUENCES FOR COMPUTATIONAL METHODS

The idealized HC described above has large implications for algorithm design. Notably, the vast computational resources available make it crucial to design algorithms that are parallel from the ground up. The classical tenets of parallel programming: identify separate tasks, data-parallelism within each task, and synchronization points, still apply. Different sequential tasks are issued to the general-purpose processors. Inheritably parallel tasks, such as processing every element of a data set, is executed on the data-parallel processors. For an overview of successful data-parallel algorithms, see Luebke et. al.[3]

The NUMA memory layout of HCs has probably an even bigger impact on algorithm design. The expected growth in processor performance seems to outpace the growth of memory bandwidth, making future HCs even more memory starved than they are today. The flow of data through the computation must be organized so that for the most part processors access data that is close to it. The biggest challenge may be to identify and minimize the dependency between data, even if it involves performing extra computations to avoid data synchronization. This requires developers to understand and incorporate the processor and memory topology into their algorithms.

Even if developers rely on high-quality libraries to perform standard algorithms such as linear algebra and Fourier-transforms, they can not help with data dependency and locality. Hence, they can not solve the problem of global data-flow through a large computation. Developers have to a certain degree been spoiled by automatic cache hierarchies that have hidden much of the complexity of memory management. Analysis of data flow is hard to perform at compile-time, and in the long term we might see systems with a runtime-environment that speculatively moves data around based on profiling information.

Another aspect of the architecture is the added cost of performing double-precision calculations. Since such numbers requires twice the bandwidth and compute at half the speed of single-precision, mixed-precision algorithms will play a key role. Several such algorithms has already been demonstrated, see Göddeke et. al.[4] Furthermore it provides ample opportunities for theoretical developments within stability and error analysis.

Even if the above suggests a new approach to developing algorithms, there is still hope for existing programs. Cluster simulations based on message passing, such as MPI, has already identified synchronization points between data, and can be modified to execute on a HC. Existing sequential methods provide a greater challenge, but numerical software can often gain dramatic increases in performance by refactoring just a few select parts[5]. However, in the end, such approaches will most likely reach the limits of Amdahl's law: *The theoretical maximum speedup is limited by the relative time needed for the non-parallel part of the algorithm.*

We expect a plethora of tools will emerge with the aim to help fully utilize HCs. However, even after several decades of compiler design, automated parallelization is still in its infancy. Automatically utilizing hundreds of processors with different capabilities is believed to be an even harder task. Such tools will therefore probably take the form of profilers and debuggers, aiding developers in identifying bottlenecks and data dependencies. There will be no tool that acts as a *silver-bullet,* automatically making sequential algorithm perform well on HCs.

For an in-depth analysis of research challenges as we move to HC architectures, see Asanovic et. al.[6]

4 CONCLUSION

The discussion above highlights many issues and challenges that heterogeneous computing presents. However, they have already demonstrated that they can provide an enormous amount of floating-point performance, which ultimately can lead to new insight and understanding. Already, the world's fastest computer, IBM RoadRunner, is built using heterogeneous principles. We expect languages and tools to mature, and different hardware platforms to converge over time, ultimately making HCs easier to utilize. As an analogue, it took a decade from the first shared memory machines was available, until OpenMP became an industry standard. In the short- to mid term it is necessary for algorithms designers to have deep knowledge of the underlying hardware. We expect the most successful algorithms will be designed by cross-disciplinary teams, consisting of both domain specialists and computer scientists.

The move to heterogeneous computing might prove to be a radical change in the way algorithms are developed. The mental model of a computation as a composition of functions could be a fallacy, instead it might be better to see computations as a largely disconnected flow of data. This provides an ample opportunity for new research and novel algorithms, bringing computational methods into the era of heterogeneous computing.

REFERENCES

[1] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe and T. Yamazaki, *Synergistic Processing in Cell's Multicore Architecture*, IEEE Micro, (March 2006).

[2] http://www.nvidia.com/cuda

[3] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos and I. Buck. *GPGPU: General-Purpose Computation on Graphics Hardware,* ACM/IEEE Conference on Supercomputing, Tampa, Florida, November 11 - 17, (2006).

[4] D. Göddeke, R. Strzodka and S. Turek, *Performance and Accuracy of Hardware-Oriented Native-, Emulated- and Mixed-Precision Solvers in FEM Simulations*, IJPEDS, Special issue: Applied parallel computing, Taylor & Francis, Volume 22, Pages 221-256, (2007).

[5] J. Michalakes and M. Vachharajani, *GPU Acceleration of Numerical Weather Psrediction,* IEEE International Symposium on Parallel and Distributed Processing Symposium, Pages 1-7, (2008)

[6] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams and K. Yelic. *The Landscape of Parallel Computing Research: A View From Berkeley,* EECS Deoartment, Universityof California, Berkeley. Technical Report NO. UCB/EECS-2996-183, (2006)