H2020 FETHPC-1-2014

Enabling Exascale Fluid Dynamics Simulations

Project Number 671571

D2.1 – Initial Report on Exascale Technology State-of-the-Art

WP2: Efficiency Improvements Towards Exascale

Document Information

Deliverable Number	D2.1
Deliverable Name	Initial Report on Exascale Technology State-of-the-Art
Due Date	31/05/2016 (PM8)
Deliverable Lead	UEDIN
Authors	N. Johnson, UEDIN
Responsible Author	N. Johnson, UEDIN
Keywords	Exascale, Technologies, Programming
WP	WP2
Nature	R
Dissemination Level	PU
Final Version Date	31/05/2016
Reviewed by	Philipp Schlatter, KTH Ulrich Rist, Univ. Stuttgart
MGT Board Approval	31/05/2016

Document History

Partner	Date	Comments	Version
UEDIN	29/04/2016	First draft	0.1
UEDIN	02/05/2016	Second draft	0.2
UEDIN	18/05/2016	Integrate comments from USTUTT	0.5
UEDIN	26/05/2016	Final edits	0.6
KTH	31/05/2016	Finalization after PMB review	1.0

Executive Summary

Current HPC technology is unable to scale to a sufficient degree to allow construction of a feasible exascale machine. Using current technology, such a machine would consume at least 7 times the 20 MW target power limit. If this is ignored, then it would be possible to construct such a machine. However no job is likely able to make efficient use of a whole machine for two reasons: faults, and software scaling. With respect to this, scaling current technology would lead to a high number of faults, i.e., where a processor fails in some manner that is unrecoverable by the hardware alone. The application would have to be robust to this, most likely by using fault tolerant methods, as check pointing a computation on such a machine may prove prohibitively expensive.

Looking forward, emerging technology will develop to help reduce current bottlenecks: latencies will fall, both between memory and processor and between nodes; energy efficiency will increase, bringing the power target within reach, and programming models will evolve to either provide or allow simpler construction of scalable fault tolerant methods.

This deliverable examines some current technologies and projects working towards the Exascale to see how a machine implementing them or their successors might look, and where the bottlenecks might lie in such a machine. For example, using die-stacked memory (where the memory and processor inhabit the same physical package, if not the same die) reduces the energy required to move data, along with the latency incurred. This, however, implies that off-package communication is relatively more expensive than at present, unless interconnect technology can keep pace. Consequently, algorithm design may have to focus more on communications avoidance to reduce the impact of these penalties.

Contents

1 Acronyms & Abbreviations

Acronym	Meaning
CAF	Co-array Fortran
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
DDR4	Double Data Rate 4th Generation
DRAM	Dynamic Random Access Memory
DSM	Die Stacked Memory
ECC	Error Correcting Code
Flop	Floating Point Operation
FPGA	Field Programmable Gate Array
FT	Fault tolerant / fault tolerance
GPGPU	General Purpose Graphics Processing Unit
HBM	High Bandwidth Memory
HMC	Hybrid Memory Cube
HPC	High Performance Computing
L1/L2	Level 1 / Level 2 Cache
LLC	Last Level Cache
MPI	Message Passing Interface
OpenMP	Open Multi-Processing
PCIe	Peripheral Component Interconnect Express
PE	Processing Element
PGAS	Partitioned Global Address Space
SoC	System on Chip
TSV	Through Silicon Via
UPC	Unified Parallel C

2 Introduction

The purpose of this deliverable is to describe some of the current technology trends in HPC and project these trends forward to imagine what an exascale system based on these technologies would look like, what performance characteristics it might have, and what bottlenecks it may display.

3 Building an exascale system with current technology

In order to provide a meaningful comparison, a fictitious exascale system named Fictision, which is based upon current COTS technologies, is described. This allows us to consider how far current technology needs to advance to reach the exascale.

Fictision is based on the same technology as ARCHER (ARCHER, 2016), the UK National Facility operated by EPCC. ARCHER has 4920 nodes, each comprising two Intel Xeon CPUs. It has a theoretical maximum performance of 2.55 PFlop/s and a measured maximum of 1.64 PFlop/s - thus, it has a ratio of theoretical to measured performance of 1.55. It consumes 1,200 kW of power at full load, giving a delivered energy efficiency of 1.368 GFlop/W.

Based on these characteristics, if Fictision were to be built using this technology, in order to achieve a measured performance of 1 EFlop/s, it would require hardware capable of 1.55 EFlop/s. A single node has a theoretical performance of 518.4 GFlop/s[1] and, therefore, Fictision requires 2,989,970 nodes. Using ARCHER's power figures, Fictision requires 729 MW of power at full load.

As a comparison, Shoubu, the leader of the Green 500 list (a list of the most efficient supercomputers), has a power efficiency of 7.031 GFlops/W. Using this efficiency, an exascale machine would draw 142 MW. Whilst this is less than Fictision, it is still greater than the 20 MW target with a per flop energy-efficiency deficit of 7x.

Given sufficient financial resources, it would be feasible to manufacture the hardware required to build an exascale machine. The same would be true for the cooling and space required to host such a machine, and for the electricity required to power it. What is less clear is if it would be of practical use. Current codes are unlikely to be able to exploit the full width of the machine, i.e. they would not be able to scale sufficiently well to make efficient use of the available compute power. If a code could scale to the whole machine, it would likely see hardware and software faults during each run. Unless fault tolerance was implemented, either by the programming model, program or algorithm, this would likely result in a very low rate of successful job completion. More likely is that such a machine would be a capacity machine rather than a capability

[1] We assume the CPU runs fixed at its maximum speed of 2.7 GHz and executes 8 instructions per cycle.

machine, running a selection of jobs of different sizes, much like current machines - except that no job would be run using all available resources.

4 Emergent technologies

In this section, we consider some emerging technologies and consider how they will help to overcome some of the limitations of current technology, which will be required to reach a practical and useable exascale system.

4.1 Die Stacked Memory

Die Stacked Memory (DSM) is the process whereby Dynamic Random Access Memory (DRAM) is placed as close as possible to the processing elements (PE) in a chip. There are some variations in technology: one where the DRAM is on the same die as the PE with the inter-connect via the substrate; and another where they are separate wafers packaged together, with a through-silicon via (TSV) as the interconnect. The former gives a higher bandwidth and lower latency but requires a design freeze earlier in the process, as the whole chip must be laid out as a single part. The latter allows for design changes to a single layer but has a (comparatively) lower bandwidth and higher latency, as communication still occurs as if between chips (as it technically is), although with an improved speed compared to that of current PE to DRAM.

The implication is that the latency seen between PE and DRAM decreases and moves much closer to that of PE and last-level cache (LLC). The latency penalty for a data or instruction miss to DRAM on either a read or write is much lower, which implies PEs should have a lower stall time. With a subsequently higher memory bandwidth, memory bound codes should perform better where the memory restriction is to local memory. Any off-chip communication, i.e. via the PE interconnect, will be comparatively more costly. Should the ratio of computation to data movement (with respect to DRAM) be suitable, it may be possible to avoid the on-die cache hierarchy entirely and use the DRAM as a flat memory hierarchy. This may offer an advantage where the dataset is too large to hold in cache and the hardware pre-fetcher is ineffective with respect to cachelines.

4.2 High Bandwidth Memory

High bandwidth memory (JEDEC, 2015) (HBM), is a memory interface specification and natural consequence of DSM. When 4th generation double data rate memory (DDR4) is integrated into the package or die (via DSM) the additional memory bandwidth can be exploited to reduce the limitations of the DRAM bus.

4.3 Hybrid Memory Cube

Hybrid memory cube (Hybrid Memory Cube Consortium, 2012) (HMC) is the competing interface to HBM from Intel which follows the same principles of using fast modern DDR4 coupled to the CPU via TSVs.

4.4 System on Chip processors

In-package heterogeneity gives rise to heterogeneous architectures, i.e. different types of processing element such as GPU cores, CPU cores and FPGA fabric being integrated into the same package, if not the same die. Current models for compute offload, where certain computational kernels are passed to a PE that has better performance for the kernel than the CPU core, must be wary about the time taken to transfer data, and the kernel, to and from the PE. Even at the full speed of a 3rd generation Peripheral Component Interconnect Express (PCIe) bus, the effect is non-negligible. Bringing these elements closer to the CPU cores allows more efficient offload, as transfer time to and from the PE is reduced. The disadvantage is that the package must be larger in order to accommodate the extra elements. This leads to either a smaller area available for CPU cores or a higher density, which may increase power consumption and heat generation. Furthermore, as the manufacturing process may be more complex to accommodate the increased chip complexity, the likely yield from production will drop. This could increase costs or result in variability between chips, with some having different performance than others. The gamble played is that the efficiency gained by using a PE better matched to the computational kernel negates the loss of the generalist CPU cores, and consequently the overall electrical efficiency is better than a chip of pure CPU cores.

An exascale machine built using this technology would have a high potential computational density (flops/mm^2), but would be able to realise this only where both the algorithm and programming model were able make efficient use of the non-CPU cores.

4.5 Fault tolerant algorithms

As the number of PEs increases - regardless of their type - the likelihood of a physical, hardware fault occurring in a machine increases. An exascale machine will see regular hardware faults, some of which can be corrected by the hardware itself (e.g. error correcting code (ECC) DRAM), and the rest, which cannot be corrected, will percolate up through the software stack and may become evident at the application layer. A small-scale fault, such as a non-correctable bit-flip in memory, may lead to a bad value being fed to a PE on load. A large fault may be one in which a PE or node fails and powers down. The part of the calculation being run on this element is lost and the work must be re-done.

Some algorithms may require restarting from the most recent checkpoint; this may incur a significant time penalty if checkpoints are widely spaced in time, which is not unreasonable if the checkpoint itself incurs a penalty to create. However, it is possible for some algorithms to continue and gracefully recover, perhaps taking longer to complete; these algorithms are collectively called fault

tolerant (FT) algorithms. Any machine built at the exascale level will require a degree of FT in the programming model, runtime or algorithm to cope with the unavoidable faults which will occur due to the scale of the hardware.

4.6 Scalable programming models

It is highly unlikely that any of the programming models used on current petascale machines will not run on exascale machines. The difficulty will come from ensuring they make efficient use of the available hardware. Shared memory models such as OpenMP and Pthreads will have to be able to handle on-die heterogeneity. OpenMP and OpenACC currently have a model for this type of working which may extend to meet the hardware. There is some element of chicken-and-egg here, where the model's extension will adapt to meet the hardware as it becomes available. Distributed memory approaches, most notably the message passing interface (MPI), will have to be able to scale to $O(10^6)$ nodes, assuming that any intra-node threading is handled by another model, or possibly $O(10^7)$ processes if a code is written using purely MPI. Partitioned Global Address Space (PGAS) implementations such as Chapel, Unified Parallel C (UPC), GASPI and Co-array Fortran (CAF) will face the same limitations. The advantage they may have over the purely distributed memory models is that they already include a notion of memory affinity, which may be adapted or extended to be aware of the difference between on-die and off-die/off-node memory.

5 What would a system look like if it used all such technologies?

Based on the technologies above, an exascale machine could look quite different to current architectures. The bottlenecks which dominate the system will be the energy required and latency incurred in moving data between nodes. Nodes will be a collection of heterogeneous processors: some general purpose, like today's CPUs, perhaps of varying sizes; some more specialised (such as SIMD units) like todays GPGPUs; and some configurable cores akin to FPGA fabrics. Closely coupled to this will be very high bandwidth, low latency DRAM which acts as a memory for data storage via the normal cache mechanism and a buffer for I/O. Configuration of this may be dynamic and flexible, allowing a runtime to best manage the resource.

The programming models will be more complex. In order to manage such a diverse set of processors available on each node, smart runtime systems will offload the computational kernels to the most suitable available processor. They may manage scheduling of such kernels either automatically, or with some level of direction from the user. The choice will largely depend on which metrics the user ranks most highly: energy efficiency, time to solution, peak power, or best utilisation of nodes.

6 Conclusion and Future Work

This deliverable has highlighted some of the currently emerging technologies which will go some way to delivering a feasible, usable and efficient exascale machine in the next 10 to 15 year timescale. The exact construction, with respect to hardware, software, programming model and algorithms is likely to be an evolution of that used in current systems with different bottlenecks, and thus different targets for optimisation at each level.

7 Bibliography

ARCHER. (2016, May 2). *ARCHER hardware guide*. Retrieved from ARCHER website: http://www.archer.ac.uk/about-archer/hardware/

Hybrid Memory Cube Consortium. (2012). Hybrid Memory Cube Specification 1.0.

JEDEC. (2015, Novermber 1). HIGH BANDWIDTH MEMORY (HBM) DRAM.