H2020 FETHPC-1-2014

Enabling Exascale Fluid Dynamics Simulations
Project Number 671571

D2.3 – Final report on exascale technology state-of-the-art

WP2: Efficiency improvements towards exascale

Document Information

Deliverable Number	D2.3
Deliverable Name	Final report on exascale technology state-of-the-art
Due Date	30/04/2018 (PM31)
Deliverable Lead	UEDIN
Authors	N. Johnson, UEDIN N. Jansson, KTH A. Peplinksi, KTH B. Dick, USTUTT J. Gong, KTH M. Bareford, UEDIN M. Vymazal, IMPERIAL P. Vogler, USTUTT U. Rist, USTUTT
Responsible Author	N. Johnson, UEDIN, n.johnson@epcc.ed.ac.uk
Keywords	Exascale, CFD
WP	WP2
Nature	R|Other
Dissemination Level	PU
Final Version Date	30/04/2018
Reviewed by	E. Laure, KTH & P. Schlatter, KTH
MGT Board Approval	30/04/2018

Document History

Partner	Date	Comments	Version
UEDIN	31/01/2018	First draft, skeleton outline	0.1
UEDIN	26/02/2018	Second draft, add partner contributions + begin to combine.	0.2
UEDIN	12/03/2018	Add conclusions + exec summary	0.3
UEDIN	20/04/2018	Add supplementary material from EASC conference outputs.	0.4
UEDIN	26/04/2018	Add comments from EL.	0.5
UEDIN	30/04/2018	Add comments from PS	0.6

Executive Summary

This deliverable considers the performance which we might see from an exascale HPC system running our co-design applications. We first recap the current state of the art with respect to our available HPC systems and find that since the beginning of ExaFLOW, there has been only a small change in performance of hardware. Further, we find that our co-design applications are limited not by the absolute performance of the CPUs or memory but by the efficiency of the algorithms used with respect to the hardware.

We also discuss improvements being made to make more efficient usage of existing hardware.

In particular:

- We examine the communications library, GS, used by both co-design applications and find that in cases of high complexity meshes, an approach based on PGAS methods can reduce communication latency. We expect CFD problems to use more complex meshes as we advance towards the exascale, and more efficient communications between computing elements (whatever the format of those elements), to be to our advantage.
- We examine the I/O performance of Nektar++ with respect the library used to perform the reading and writing of data. We find that for large (multi-gigabyte) reads the SIONlib library gives good performance and is constant across core-count. For large writes, file-per-process XML gives the best performance and is second best for reads, so is the best aggregate choice.
- We also see how offline compression methods can reduce the volume of data generated by a CFD simulation output, to enable efficient disk usage and reduce the time taken for check points and restarts.
- We comment on improvements to algorithms and other technologies which will be required to achieve good performance. For example, we are exploring GPUs via porting Nek5000 to OpenACC to explore performance on accelerators.
- We briefly comment on weak boundary conditions. Using weak boundary conditions allows us to begin iterations of a solver routine with loose boundary conditions and thus control the build-up of errors which could lead to solver instability.
- We also comment on work done to improve the resilience of algorithms to machine failure. By considering solver iterations as tasks within the simulation, when a node fails, the task can be migrated to another node to complete so that the complete simulation need not be started again.

Finally, we make some comments on how close current codes are to being able to achieve good performance should an exascale system become available.

Contents

1 Introduction

This deliverable reports on the current state of the art with respect to Computational Fluid Dynamics (CFD) and considers how our current software would perform on a future exascale system, what that system may look like and what obstacles we need to overcome to reach efficient performance on said system.

2 Current state-of-the-art hardware

The current state-of-the-art with respect to hardware that is used to run large scale CFD applications can be seen through the three HPC machines used with ExaFLOW - ARCHER[1] at UEDIN, Hazel Hen[3] at USTUTT and Beskow[2] at KTH, and with three leading machines - Piz Daint[8] at CSCS, TaihuLight at the National Supercomputing Centre at Wuxi[6] and Summit[7] at OLCF. Whilst all three ExaFLOW machines are supplied by a single vendor, Cray Inc, they have different specifications and represent the differences between current machines that current CFD users will see in production.

ARCHER is a Cray XC30 system with a total of 4920 compute nodes (118,080 cores). Each node contains two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors. Each of the cores in these processors can support 2 hardware threads (hyperthreads). Within the node, the two processors are connected by two QuickPath Interconnect (QPI) links. Each node has at least 64 GB of memory shared between the two processors, a small number have 128GB. The memory is arranged in a non-uniform access (NUMA) form: each 12-core processor is a single NUMA region with local memory of 32 GB (or 64 GB for high-memory nodes). Access to the local memory by cores within a NUMA region has a lower latency than accessing memory on the other NUMA region. At zero load, the compute nodes on ARCHER draw approximately 400 kW of power and at full load they draw approximately 1.2 MW of power.

Hazel Hen is a Cray XC40 system with a total of 7712 nodes (185,088 cores) with 2x Each node contains two 2.5 GHz 12 core E5-2680 v3 (Haswell) series processors. Each of the cores in these processors can support 2 hardware threads (hyperthreads). Within the node, the two processors are connected by two QuickPath Interconnect (QPI) links. Each node has at least 128 GB of memory shared between the two processors. As with ARCHER, each processor is a single NUMA region. The complete system, including disk, draws approximately 2.8 MW of power in production mode.

Beskow is a Cray XC40 system with a total of 2060 compute nodes, split across 11 cabinets. Nine of the cabinets have nodes comprising two 2.3 GHz 16 core E5-2698 v3 (Haswell) processors and 64GB of memory (per node), and the remaining two cabinets have nodes consisting of two 2.1 GHz 18 core E5-2695 v4 Broadwell CPUs and 128GB of memory (per node). Each of the cores in these processors can support 2 hardware threads (Hyperthreads). Within the node, the two processors are connected by two QuickPath Interconnect (QPI) links. As with ARCHER, each processor is a single NUMA region. The compute part of the system draws approximately 842 kW of power in production mode.

Whilst these machines differ in age and size, they are similar in offering multi-core CPUs with hyper-threading and between 2 and 5 GB of RAM per physical core. Processor speeds vary slightly, but are generally between 2 and 2.5 GHz base frequency. Current CPUs from the same vendor (Intel) are the same, so there is little expectation that this will change in the next iteration. The number of cores per CPU varies between iterations of the technology (and choice of processor) but the RAM per CPU thread stays roughly similar.

Not used in ExaFLOW, but similar in that it is a hybrid Cray XC40/XC50 is Piz Daint at CSCS in Lugano. Piz Daint has an XC40 partition comprising 1813 nodes comprising two Xeon 2.10GHz 18 core E5-2695 v4 (Broadwell) processors with 64 or 128 GB RAM of RAM per node. It has an XC50 partition comprising 5320 nodes which each have a single 2.60GHz 12 core E5-2690 v3 (Haswell) processor with 64GB RAM coupled to a NVIDIA Tesla P100 HPU with 16GB of RAM. The inclusion of GPUs in the XC50 partition allows the running of codes which can make efficient use of heterogenous hardware, such as OpenSBLI.

Again, using a GPU plus CPU model is Summit, at Oak Ridge Leadership Computing Facility. Summit is the replacement to the Titan supercomputer and is currently being installed. Summit will be comprised of approximately 4,600 nodes and will have a hybrid architecture, with each node will containing multiple IBM POWER9 CPUs and NVIDIA Volta GPUs connected with NVIDIA's high-speed NVLink. Each node will have over half a terabyte of coherent memory (high bandwidth memory + DDR4) addressable by all CPUs and GPUs plus 800GB of non-volatile RAM that can be used as a burst buffer or as extended memory. To provide a high rate of I/O throughput, the nodes will be connected in a non-blocking fat-tree using a dual-rail Mellanox EDR InfiniBand interconnect. This further pushes the idea of using GPUs plus CPUs in a single node, like the XC50 but with more attention paid to fast interconnect on-node to move data between the CPU and GPU devices. Also, this machine contains node-local storage designed specifically to act a burst buffer, amenable to offline compression methods such as those using wavelet compression.

Finally, there is the current number one system from the top 500 list of powerful supercomputers, TaihuLight. This machine comprises a total of 40,960 Chinese-designed SW26010 manycore 64-bit RISC processors based on the Sunway architecture. Each processor contains 256 processing cores (much more than the 12 or 18 seen with the other systems), and an additional four auxiliary cores for system management (also RISC cores, just more fully featured) for a total of 10,649,600 CPU cores across the entire system. Interestingly, this system does not have a traditional cache hierarchy seen in the x86 Xeon chips, but uses a network on chip to connect the processing cores for communication rather than via the cache. This machine supports the use of a custom version of OpenACC v2.0 to take advantage of the processing power available and aid in the parallelization of codes. This validates the use of our experiments with OpenACC to parallelize key kernels.

There are also some systems based on the now discontinued XeonPhi architecture such as Tianhe-2 based at NUDT in China which operated a similar model to Summit, combining Xeon processors and XeonPhi accelerators in nodes. This technology will not see any updates or upgrades and indeed, Tianhe-2 is deprecated in favour of TiahuLight. The Sunway processor, which is based on a multi-core architecture much like XeonPhi, but with a greater number of cores/die may be a good replacement for those codes or kernels previously targeted to a XeonPhi multicore architecture

3 Software advances to co-design applications.

Since the beginning of the ExaFLOW project in October 2015, we have worked to close the performance gap of our co-design applications to make more efficient usage of available hardware. As hardware advances, further work will be required to tailor the applications (and algorithms) to make the most efficient use of resources.

3.1 Improvements in communication libraries

We have developed new communication kernels for Nek5000 based on the one-sided communication model, provided by the partitioned global address space (PGAS) abstraction.

Communication in both Nektar++ and Nek500 is provided by the GS library. This offers both pairwise communications between processing elements and the crystal router, a hypercube exchange method well suited to swapping data for CFD meshes. GS lib is currently written in single sided, basic MPI which requires message matching.

Using the one-sided model, we reduce latency and synchronization costs by avoiding costly message matching, thus allowing for more fine-grained parallelism, with less necessary local data per processing element to balance communication costs, resulting in an improved strong scalability of Nek5000. Furthermore, with more fine-grained parallelism we also anticipated that Nek5000 will be able to efficiently utilize the extreme parallelism offered by new and anticipated exascale hardware.

Figure 1 shows the performance of a simple test case (a round trip latency, or ping-pong test) for both the existing MPI implementation and ExaGS, our UPC PGAS implementation. We see an improvement as core count increases and therefore a reduced latency in communications at scale. Given that Nek5000, Nekbone & Nektar++ can use the same communications routines, we expect this to be reflected in the co-design applications, where we expect to see a reduction in wait times on data exchange between nodes or when performing synchronisation steps.

However, the underlying structure of the crystal router is non-optimal for the programming model requiring some inelegant designs to maintain functionality and interoperability with both application codes.

One of the limitations in ExaGS is the crystal router data exchange method. It is suited to the MPI programming model (and implementations thereof) but not best suited to UPC implementations as the algorithm assumes the programming model. To become efficient at exascale node counts, it will require rework at the algorithmic level to decouple from message passing type of programming model. This is likely to be true for most communication libraries, especially where there is reliance on communication having completed before an iteration of a solver begins. Reliance on barriers and reliable global functions (such as all-reduce operations) will have to be lessened within application codes.

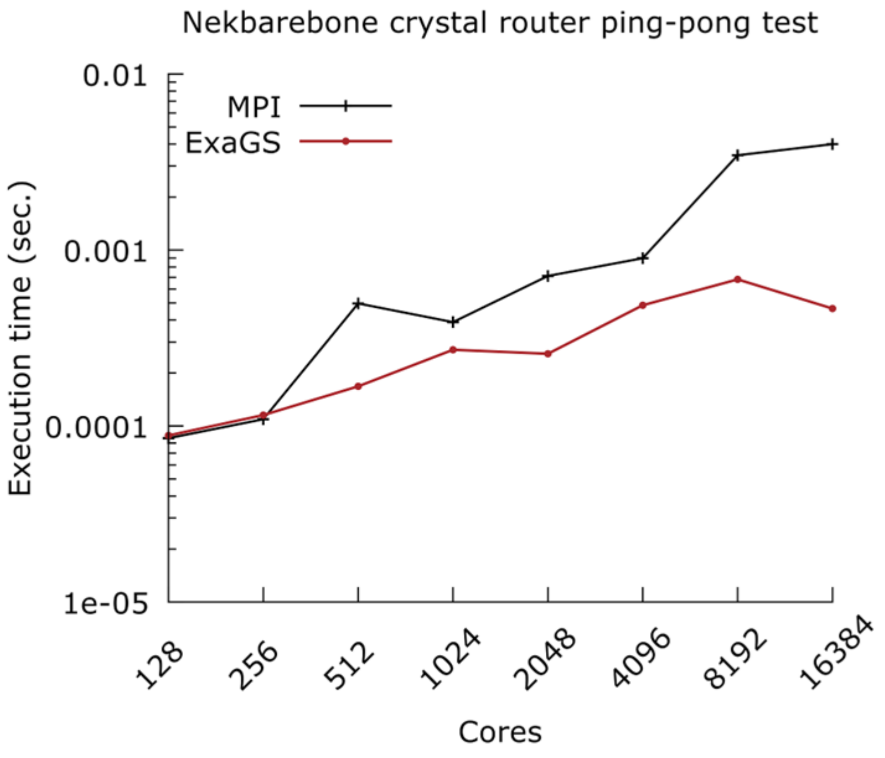

Figure 1: Comparison of round-trip latencies for MPI-based and PGAS-based GS library.

3.2 Exploration of GPU architecture

Within the project, we expanded on our previously developed work and take advantage of the optimized results to port the full version of Nek5000 to GPU-accelerated systems, especially regarding the $P_N–P_{N-2}$ algorithm. This latter algorithm is a way to decouple the momentum from the pressure equations that does not lead to spurious pressure modes. It is more efficient than other methods, and it involves different approximation spaces for velocity (order N) and pressure (order N-2). The project focuses on the technology watch of heterogeneous modelling and its impact on the exascale architectures (e.g. GPU accelerators system). GPU accelerators can strongly speed up the most consuming parts of the code, running efficiently in parallel on thousands of cores. The goal of this sub-project is to implement a full version of the $P_N–P_{N-2}$ algorithm which can take advantage of hybrid architectures and be used in Nek5000 to improve its scalability to exascale. The current performance of the matrix-free fast tensor product kernel is shown in Figure 2. However, this is a single kernel running on a single GPU; to increase the usage of GPUs, memory swapping (for halo exchange) must be implemented via MPI with GPUdirect to avoid the overhead of copying complete memory buffers between GPU and CPU host.

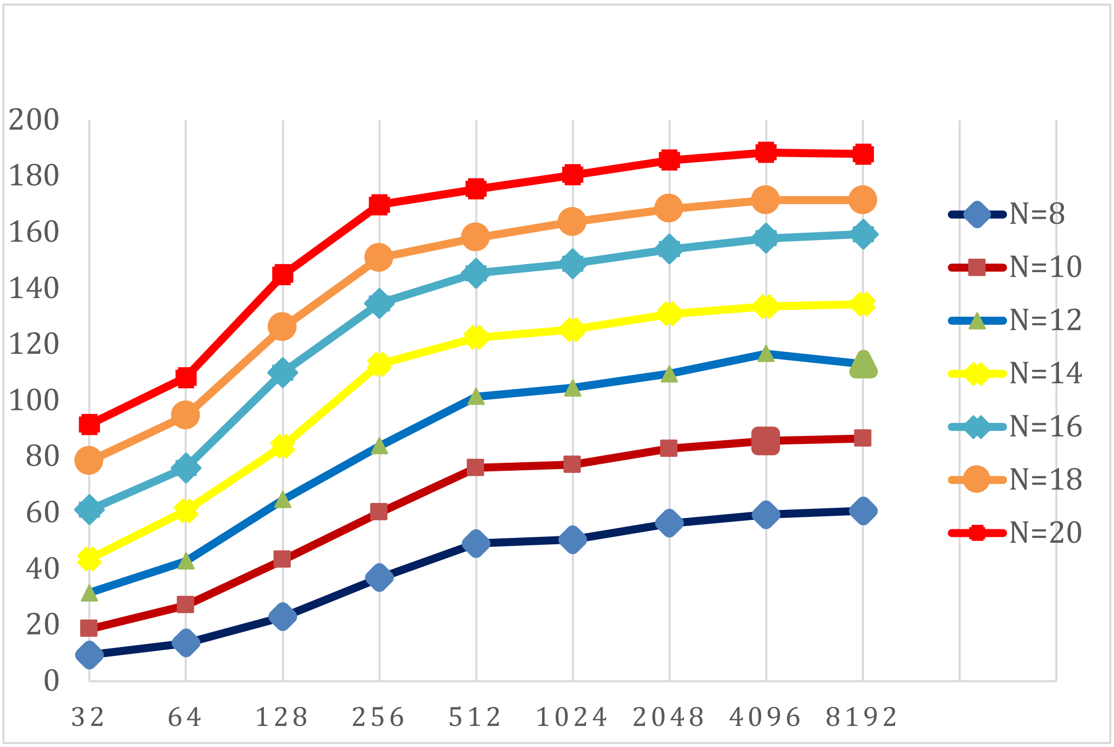

Figure 2: Performance of tensor product kernel running a single P100 GPU with OpenACC.

3.3 Fault Tolerance

The reliability of running CFD simulations poses one of the major challenges when one considers using exascale machines, i.e. achieving 10^{18} floating-point operations per second. Given the necessary number of components that an exascale system consists of, based on the failure rates of current hardware we can extrapolate that the mean time to interrupt (MTTI, the time before a failure occurs) is of the order of only a few minutes. This makes exascale resources unusable for the context of extremely large-scale simulations, requiring novel strategies to deal with in-situ hardware failures.

Our work in Nektar++ as part of ExaFLOW has been to design algorithms tolerant of failures, without adversely affecting the performance and scalability of the code. Most importantly, we have been seeking solutions which provide resilience in memory-conservative manner. The prototype approach, which is now available in Nektar++, uses a proposed fault-tolerance extension to the MPI standard called User-Level Failure Mitigation (ULFM), combined with a memory-efficient approach to represent and duplicate static data (i.e. data that does not change during simulation). ULFM provides functionality to detect failures among the processes participating in simulation and to recover from them through moving the task associated with the failed node onto a spare node in the system. The greater the number of spare nodes, the greater the resilience to additional failures. Memory efficiency and performant recovery is achieved through recording the result of each MPI communication during simulation initialization; this information can be used to reconstruct the failed process on the spare node independently of other processes, thus avoiding collective communication. Currently we can run flow simulations which recover from as many sequential failures as there are spare nodes, with a memory overhead of less than 5% on large-scale runs.

3.4 I/O

To reduce performance costs associated with file I/O, the capability to write HDF5 files was added to Nektar++. The original approach of writing checkpoint field files by Nektar++ in an XML format generated individual files for each process, thus increasing disk contention with growing number of cores, and presented a significant barrier to the use of the code at extremely high core counts anticipated at exascale.

The HDF5 library uses a file format that allows for a parallel file access that allows for concurrent writing by all MPI processes at the same time. To complement this, we have explored the use of the SIONlib library to attempt to further reduce IO times at scale. Figure 3 & Figure 4 show the results of a large read and write done as part of an industrial test case, using XML, HDF5 and SIONlib routines as the core count is scaled.

Of note for the read is that for XML and SIONlib, the runtime is constant across core counts, implying that this is a limit of the combination of system and library. For the HDF5 case, the runtime increases with high core count as there is contention when each core must read some common parts of the file. For write, the situation is different. Both HDF5 and XML are consistent across core counts, but the SIONlib runtime improves (decreases) as the core count increases.

Coupled with increased efficiency in data I/O methods, compression can be used to reduce the volume of the data that is stored, allowing sufficient recovery. Current work has shown that using wavelet based compression algorithms can result in efficient lossless and lossy compression of CFD. This is currently run as a a post-processing method due to speed, but further performance improvement coupled with emerging technology, such as burst-buffers could enable use as an online method in feasible time. It could also result in a decrease in the volume of data moved across the network to disk storage, which would decrease both checkpoint and restart times for simulations, and the time taken to store snapshots for post-processing.

Table 1 below shows the compression ratio, compression time (in seconds) and Peak Signal to Noise Ratio (PSNR) for the compression algorithms JP3D, ZFP and 7-zip. As 7-zip is a lossless method, the PSNR is infinite.

Timestep	**Compression Ratio**			**Compression Time**			**PSNR**		
	JP3D	*ZFP*	*7-Zip*	*JP3D*	*ZFP*	*7-Zip*	*JP3D*	*ZFP*	*7-Zip*
0	64.66	65	41.72	6.86	1.66	114.91	181.90	61.21	∞
2.5	62.18	62.1	6.03	4.40	1.98	160.37	181.80	60.02	∞
5	61.55	61.1	5.84	7.15	1.92	173.20	153.20	48.16	∞
7.5	61.02	61.1	5.57	8.81	1.86	164.00	112.40	44.26	∞
10	60.77	61.1	5.07	10.95	1.83	172.91	88.0	37.83	∞
12.5	60.78	61.1	4.63	11.35	1.81	172.80	90.7	40.64	∞
15	60.84	61.1	4.21	11.07	1.82	186.38	90.4	40.48	∞
17.5	60.80	61.1	3.92	10.70	1.84	183.88	95.2	43.0	∞
20	60.87	61.1	3.72	11.07	1.87	181.45	99.6	45.2	∞

Table 1: Compression ratio, time and Peak Signal to Noise Ratio for JP3D, ZPF & 7-Zip compression algorithms.

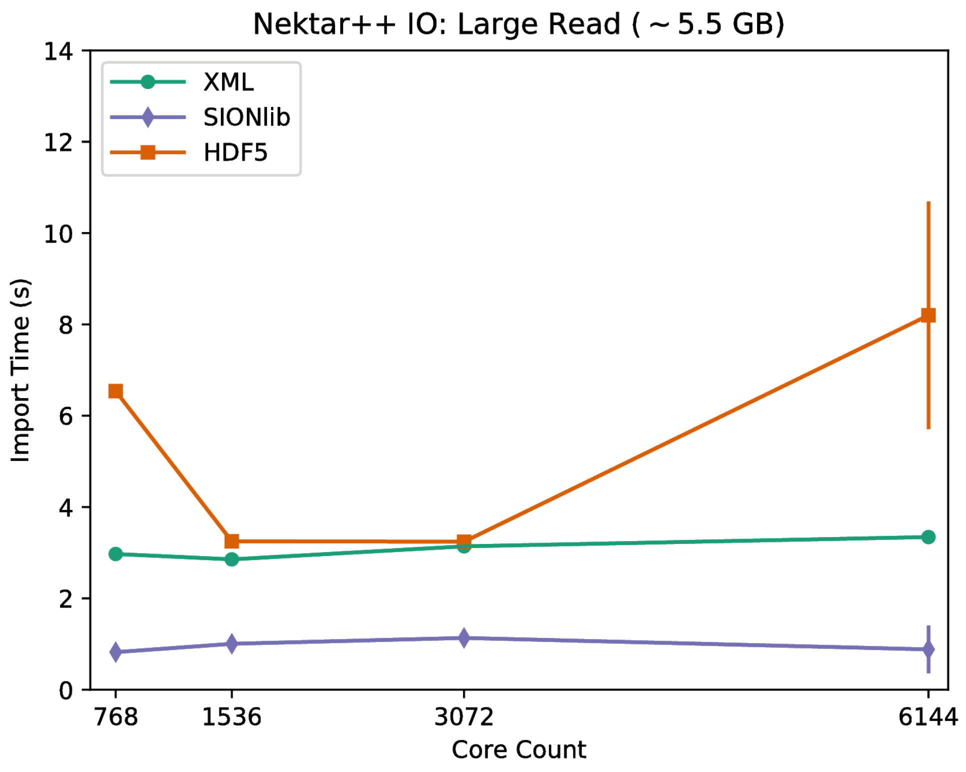

Figure 3: Nektar++ import times using XML, HDF5 and SIONlib read routines.

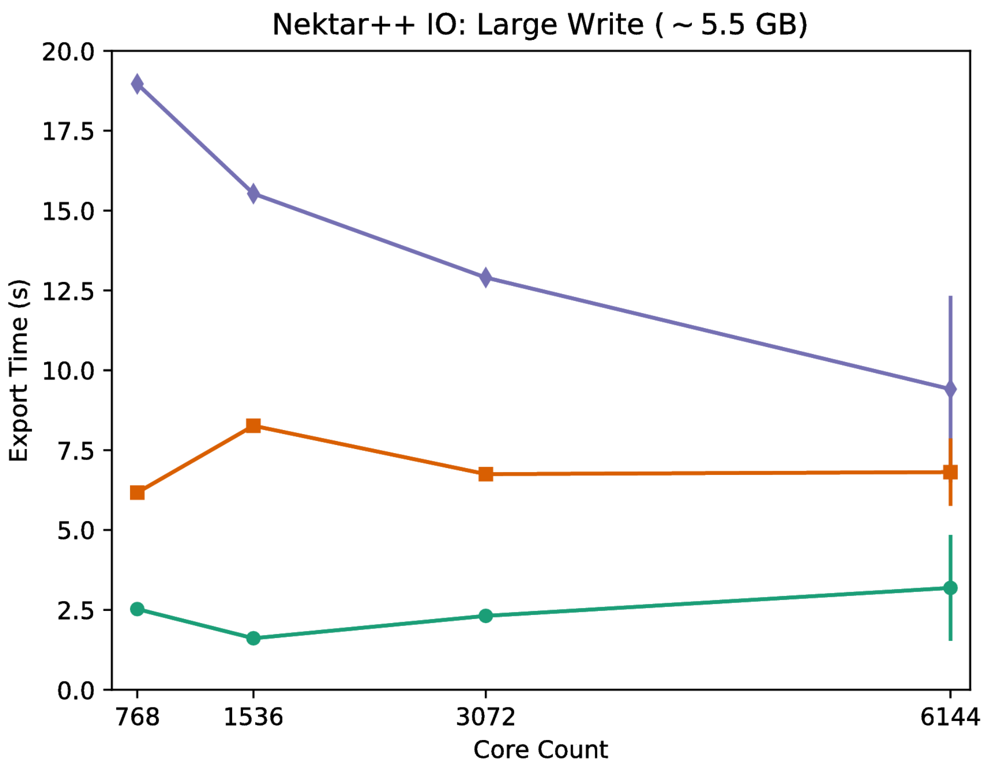

Figure 4: Nektar++ export times using XML, HDF5 and SIONlib write routines.

These results are echoed from an industrial automotive use case (Opel car) run on Hazel Hen between 96 and 786 cores. The XML-I/O-implementation produces a very high load on the meta-data server of the file system because of writing a single file per MPI-process. Unfortunately, the HDF5-I/O-implementation performed worse when compared to XML. One possible reason for the unexpected low I/O-performance is the overhead of the data management caused by the rather complex usage of HDF5 in Nektar++. Further investigations are necessary to identify the best strategy to optimize the HDF5-I/O. Maybe some adjustments in the domain-decomposition are necessary since efficiently writing HDF5 data requires large contiguous chunks of data to be written which in turn requires data to be hold as large contiguous chunks in memory.

3.5 Weak boundary conditions

At the core of the test cases considered in ExaFLOW are the incompressible Navier-Stokes equations, which are efficiently solved numerically both in Nektar++ and Nek5000 using the spectral/hp element method and a time-splitting scheme. Although the splitting of the equations involves the solution of four scalar elliptic equations for pressure and velocity components, the cost of one step largely depends on the amount of work need to obtain the pressure field, which is defined as a solution to a (frequently ill-conditioned) Poisson equation.

The difficulty in solving the governing equation for pressure is further increased when one considers complex flow features, particularly for the extremely large simulations expected at exascale, for example the formation and evolution of a wingtip vortex simulated by a high-order method. Under-integration of nonlinear terms in Navier-Stokes equations introduces an aliasing error which may compromise the stability of the simulation. This is usually not problematic when the flow features are adequately resolved. Once the build-up of aliasing errors becomes an issue, however, the stability of the solver is often compromised in vicinity of solid wall boundaries, where the introduction of boundary constraints into the discrete problem further degrades the conditioning of the stiffness matrix.

To mitigate these issues, we formulated and implemented an algorithm for weak imposition of boundary conditions, which modifies the underlying weak form by adding penalty terms. The advantage is that the boundary condition is not imposed exactly (within machine accuracy) from the very first iteration, but only at converged state and the rate of resolving flow features in the interior of the domain and on the boundary, is thus consistent. In addition, numerical experiments show that the method is more robust than the state-of the-art technique for imposing boundary conditions weakly - Nitsche's method. This will allow simulations to scale to larger core counts, on the path to exascale, whilst retaining numerical stability.

3.6 Power efficiency

To gauge the effective efficiency achievable with current state of the art systems, we investigated the energy consumption of Nektar++ running an automotive use case (Opel car) at all the available CPU clock frequencies. With respect to the computation phase, a sweet spot of runtime and energy consumption has been detected at 1.7 GHz, allowing for 19.2% reduction of the energy demand while increasing runtime by *only* 12.6% (see Figure 5). However, a subsequent analysis of identical runs revealed a variance of the used energy-delay-product in the size of the effect achievable by reducing the clock frequency. This may be due to (known) node interconnect issues. The exascale system target is to use less than 20MW to produce 1 exaflop. By optimizing the processor usage in this manner, independent of the processor hardware, we can reduce the power required to generate 1 exaflop.

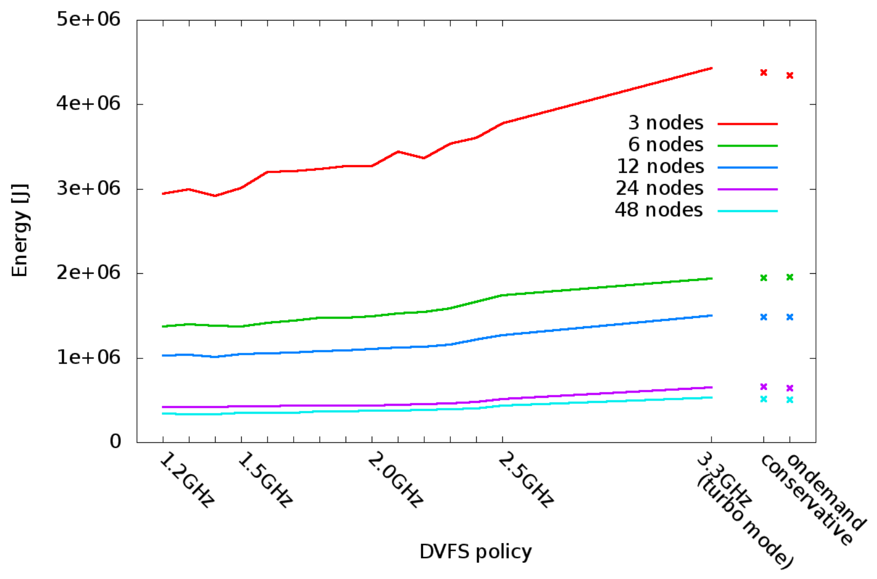

Figure 5: Energy consumed versus processor frequency for Opel test case - computation phase.

4 Future predictions

In this section, we consider the effects of our work to improve on the state of art and the progress to readiness for exascale simulations.

Since the beginning of the project in 2015, HPC hardware has seen relatively modest advancement and has mostly been centred around Intel processors. Clock speeds remain roughly constant as can be seen by comparing the base frequency of Beskow (the newest of our three HPC machines) with ARCHER (the oldest). However, during the period, the processors have undergone a process geometry shrink (between the 22nm of Ivy Bridge and 14nm of Broadwell micro-architectures) and with consequent reduction in operating power. This translates as an increase in useful work (iterations of a CFD code) per kWh. There has also been a gentle increase in memory bandwidth, as the upper limit on memory speed increases from 1.8 GHz to 2.4 GHz. The increase in memory bandwidth benefits kernels that routinely stall waiting for data to be brought into the CPU and for which the hardware pre-fetcher has been unable to optimally hide this latency. Perhaps the most interesting advance, not echoed in ExaFLOWs machines, has been the introduction of ARM-based systems to the HPC landscape. Unfortunately, these machines are not yet production ready and will not be installed and made available, for any serious testing, during the lifetime of the ExaFLOW project.

Future advances in hardware seem unlikely to result in a dramatic increase in the base clock speed of the CPU. CPUs will likely increase in core-count following a multi-core route to scaling with *fatter* chips, and some of these cores may be heterogeneous. For example, different core types on-die would allow computational kernels to be handled by cores (or collections of cores) which are well matched to the type of computation; the core types may be GPU, FPGA or some other type of offload device. As seen with our work to offload kernels via OpenACC, this may require some re-architecting of the application to achieve a more complete port of the code to accelerator, however it is dependent on the hardware available. Using the code generation features in OpenSBLI, the third of our project application codes, should go some way to helping with this point. The ability to generate code which is then compiled for the target hardware gives good flexibility across targets and may help to make quick use of a system such as TaihuLight if access were available. The separation of concerns between algorithm development, code generation and efficient compilation on a target platform is likely to be seen with more the complex heterogenous systems which may emerge.

Memory speed is unlikely to dramatically increase for external (to the CPU) memory. It is highly likely is for high-bandwidth memory to appear on-die with the CPU as part of the same package, if not sharing silicon. Further, memory hierarchies are likely to deepen and increase in size, with fast per-node storage which could be implemented using NVRAM. NVRAM is non-volatile and allows data to persist between node reboots and is flexible in its usage, being either a level 4 cache, or as a fast node-local disk. This may reduce some of the disk IO requirements and allow data to be stored node-local between jobs, reducing restart time. It may also allow on-node compression to run using node-local storage reducing the requirement for it to be on-line and act as a buffer for write phases. This technology, in the form of burst-buffers is available in a system such as in the upcoming Summit machine at OLCF.

Aside from architecture considerations, exploiting such systems with reasonable efficiently will require algorithmic approaches that make better use of available parallelism. This implies using exascale-capable programming models and techniques, including flexible use of heterogeneous hardware where available, such as in the JUWELS modular supercomputer[4] or hardware being considered for the Post-K project[5]. Considering that the current codes are unable to achieve a performance where they are constrained by the arithmetic and memory bandwidth bounds of the current machines, further work is required to make use of an exaflop capable machine.

To illustrate low arithmetic intensity, we use the roofline models as generated by the Intel Advisor toolset running on Cirrus, a cluster at UEDIN which has the same hyperthread capable processors as the smaller part of Beskow (2.1 GHz 18 core E5-2695 Broadwell) arranged similarly into dual CPU nodes. It has 280 nodes in total, giving 10,080 cores. Cirrus has 256 GB of RAM per node (~7 GB per core) giving it the highest RAM to thread-count of the machines used.

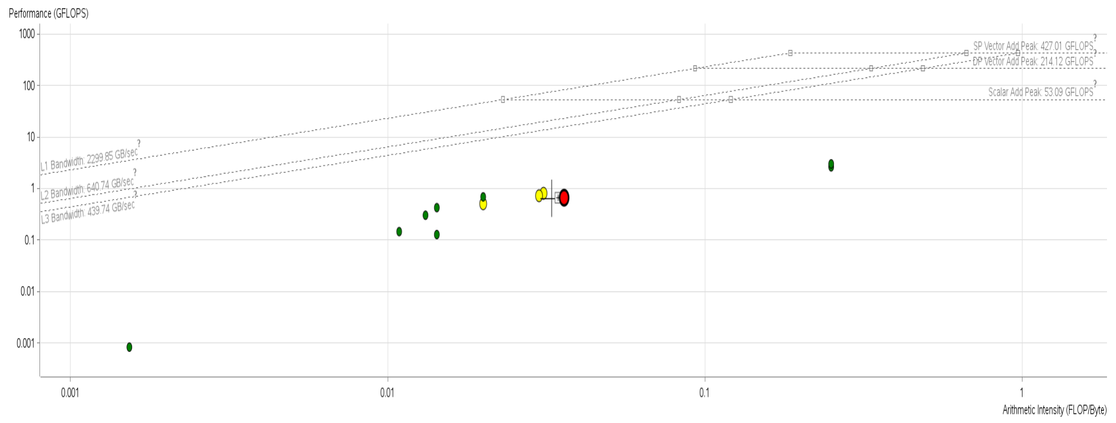

Figure 6: Roofline model of the Nekbone application showing performance limits for memory bandwidth and computation intensity.

Figure 6 shows an example of a roofline model for the Nekbone application. Nekbone is a small test/benchmark application used with Nek5000 to judge the performance of matrix multiplication operations for different sizes and shapes of matrix. It is used here to judge the performance of Nekbone, which, unencumbered by complex processing used in the Nek5000 application, should achieve excellent results.

Of primary interest is the large red circle in the centre of the graph. It shows the performance of the core matrix multiplication kernel. The y-axis shows performance in flop/s and the x-axis the arithmetic intensity, the number of operations carried out per byte of data. The dashed lines indicate the limiting factors: the bandwidth of the various memory types (on-die caches and DRAM) and the peak arithmetic performance of CPU units. From the example, the kernel is neither limited by memory bandwidth (if it was, it would appear closer to the L3 bandwidth line) nor arithmetic limit (it doesn't approach even the scalar add limit).

Thus, if memory bandwidth or available operations per second were to increase, this code would not be able to take advantage of it, without further optimization.

To gain exascale capable performance, our codes need to continue to develop:

- Better support for accelerators. Currently accelerators can achieve high arithmetic intensity and GPUs tend to have high memory bandwidth per processing unit. Whilst OpenSBLI can currently target a variety of hardware, only some kernels from Nek500 and Nektar++ can make use of this type of hardware. Efficient performance is required from all kernels and modules, and taking advantage of multiple accelerators will be required to make use of a machine of and architecture like Summit.
- Good scaling of all components. Currently there is a range of scalability across the components of our codes. For example, the Adaptive Mesh Refinement (AMR) in Nek5000 does not scale as well as the solver components. However, this is balanced by the usefulness of the method in increasing the resolution of the grid where it is needed as opposed to having a high-resolution grid at all points and doing expensive work in regions with no interesting characteristics. Efficient use of an exascale capable system will require all components to have good scaling behavior so efficiency is not lost by a weak part of the workflow.
- Further I/O improvements. We have seen that both compression and new I/O libraries can reduce the data volume and make transfer more efficient. However, more work is required to minimize the data generated, stored and, most importantly, transferred. Exascale capable machines will have likely have millions of cores (based on the scaling of TaihuLight) which implies low per-core disk bandwidth. Even with burst buffers and node-local storage, the primary bottleneck for CFD applications will be checkpoint data used to both restart simulations and generate flow over time movies for visualization. This work will require co-operation and co-design of the I/O and compression methods to balance the I/O regimes for best performance on each system.

5 Conclusion: Performance at exascale

Based on a move towards multi-core systems, and on our observations that CPU cores have not increased dramatically in clock speed or memory bandwidth, we can begin to consider performance on a future exascale machine.

Our most basic example of a CFD code, Nekbone, is not currently able to achieve a performance which is limited by the arithmetic performance or memory bandwidth available on any of ExaFLOWs current systems. Therefore, we can surmise that future performance, using current codes, is unlikely to be limited by processor or memory performance. However, it will be affected by limited IO performance, as the number of cores available increases faster than the IO bandwidth per core.

Investigations into different IO libraries have shown that there is no single best solution for both reading and writing of large data files. The most hopeful outcome here is node-local storage (NVRAM, burst-buffer or some other method) to reduce the IO wait time, assuming a checkpoint + restart scenario, combined with on-line data compression.

With respect to software limitations, investigation has shown that offloading key kernels by recoding with OpenACC can achieve a higher arithmetic intensity on GPU hardware than possible on a traditional, x86 CPU. Whilst this is a change of hardware target, it is the move from MPI-based code to OpenACC which enables the offload. It is noted that this is the programming model used for TaihuLight which utilises highly multi-core CPUs which appear more like GPU streaming multiprocessors than traditional cores in this regard.

Finally, algorithmic, rather than implementation changes are required such as: weak boundary conditions for solver improvements, AMR for reduction in work done in regions of little interest and fault tolerance to recover from hardware errors.

All the above point towards current codes exhibiting similar scaling limitations as seen on current machines unless they take an approach to engineering applications at all levels, from efficient use of hardware through compilers, programming models and communications libraries, to algorithms through methods such as AMR to reduce computational load.

6 References

1. ARCHER hardware specifications: https://www.archer.ac.uk/about-archer/hardware/
2. Beskow hardware specifications: https://www.pdc.kth.se/hpc-services/computing-systems/beskow-1.737436
3. Hazel Hen hardware specifications: https://www.hlrs.de/systems/cray-xc40-hazel-hen/
4. JUWELS modular supercomputer: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUWELS/Configuration/Configuration_node.html
5. Post-K project: http://www.r-ccs.riken.jp/en/postk/project
6. Sunway TaihuLight: http://www.nsccwx.cn/wxcyw/soft1.php?word=soft&i=46
7. Summit Supercomputer: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
8. Piz Daint: https://www.cscs.ch/computers/piz-daint/