Many-core Acceleration of a Discrete Ordinates Transport Mini-app at Extreme Scale

Tom Deakin[1], Simon McIntosh-Smith[1], and Wayne Gaudin[2]

[1] Department of Computer Science, University of Bristol, UK
tom.deakin@bristol.ac.uk, simonm@cs.bris.ac.uk
[2] High Performance Computing, UK Atomic Weapons Establishment, UK
Wayne.Gaudin@awe.co.uk

Abstract. Time-dependent deterministic discrete ordinates transport codes are an important class of application which provide significant challenges for large, many-core systems. One such challenge is the large memory capacity needed by the solve step, which requires us to have a scalable solution in order to have enough node-level memory to store all the data. In our previous work, we demonstrated the first implementation which showed a significant performance benefit for single node solves using GPUs. In this paper we extend our work to large problems and demonstrate the scalability of our solution on two Petascale GPU-based supercomputers: Titan at Oak Ridge and Piz Daint at CSCS. Our results show that our improved node-level parallelism scheme scales just as well across large systems as previous approaches when using the tried and tested KBA domain decomposition technique. We validate our results against an improved performance model which predicts the runtime of the main 'sweep' routine when running on different hardware, including CPUs or GPUs.

1 Introduction

Deterministic discrete ordinates neutral particle transport is a balance equation which describes the movement of neutral particles through materials of varying properties. As the particles move through the material, they can collide with material atoms causing a change in direction of movement and/or a change in energy level of the particle. Additionally, the particle may be absorbed and potentially a fission reaction can occur, resulting in the loss or gain of particles. The governing equation for this balance of particles, the *transport equation*, is solved via a numerical method resulting in an approximate solution due to the complexity of finding an analytic solution in all but the simplest cases. Therefore this is a computationally intensive problem which requires High Performance Computing in order to find solutions in a tractable time. Indeed, it is estimated that 50–80% of simulation time is devoted to transport codes on United States Department of Energy systems [14] making this a very important problem.

The solution of the transport equation involves a matrix-free inversion of an operator which forms a sweep across the grid; it is this part of the algorithm

which takes up the majority of the computation time in transport codes. In this paper we investigate applying our previous attempts to accelerate the intra-node updates to run a transport mini-app at large scale using many-core processors such as GPUs. A mini-app removes the complexity of a production application whilst also providing a proxy for performance. We use the KBA algorithm for spatial decomposition, the de facto algorithm to provide the benchmark for the performance of transport codes [15], see Sec. 2.1. We show weak scaling results for our accelerated scheme requiring storage of the angular flux in device memory, and discuss the challenges of strong scaling.

In particular we make the following contributions:

1. We present weak scaling results using an improved node level scheme running on GPUs. The results show that the KBA algorithm is still a solution at scale.
2. We improve on a performance model to predict the running time of our accelerated transport sweep at scale.
3. Results are presented for two leading GPU enabled supercomputers, the two largest in the world at the time of writing. We demonstrate up to a $4\times$ speedup of a GPU accelerated time-dependent deterministic transport application at scale compared to an optimised CPU version.

The rest of the paper is structured as follows. In Sec. 2 we introduce the SNAP mini-app and the KBA algorithm used for spatial decomposition. Sec. 3 introduces the current state of the art. Our implementation is discussed in Sec. 4. Our performance model is introduced in Sec. 5 and we present weak and strong scaling results in Sec. 6 before concluding in Sec. 7.

2 SNAP: Sn Application Proxy

The Discrete Ordinates (S_N) Application Proxy (SNAP) [24] is a proxy application (mini-app) from Los Alamos National Laboratory based on their deterministic discrete ordinates neutral particle transport code, PARTISN. It maintains the performance characteristics of PARTISN, without the complexity of a production application; in particular the input data is deliberately arbitrary and non-physical. Deterministic transport models the movement and interaction of neutral particles, such as neutrons or photons, through a mesh with varying material properties. As the particles move in straight lines within the material, they may interact with the material and change energy and/or direction. It is the net balance of these neutral particles that is governed by the Boltzmann transport equation (1) solved in the proxy application; the authors would recommend that readers unfamiliar with the solution of the transport equation consult the Lewis and Miller textbook for an introduction [16]. Whilst the transport equation can in general include fission terms, the equation solved in SNAP does not include this.

$$\left[\frac{1}{v}\frac{\partial}{\partial t} + \hat{\Omega}\cdot\boldsymbol{\nabla} + \sigma(\boldsymbol{r}, E)\right]\psi(\boldsymbol{r}, \hat{\Omega}, E, t) = \tag{1a}$$

$$q_{\text{ex}}(\boldsymbol{r}, \hat{\Omega}, E, t) + \tag{1b}$$

$$\int dE' \int d\Omega' \sigma_{\text{s}}(\boldsymbol{r}, E' \to E, \hat{\Omega}' \cdot \hat{\Omega})\psi(\boldsymbol{r}, \hat{\Omega}', E', t) \tag{1c}$$

The transport equation operates over seven dimensions: time (t), three-dimensional space ($\boldsymbol{r}$), two angular ($\hat{\Omega}$) and energy (E). The net movement of the neutral particles in these dimensions is called the *angular flux* ψ. Integrating over the angular dimensions yields the *scalar flux* ϕ. q_{ex} defines an external source of particles. A large number of particles are assumed so that each dimension can be considered as a continuum. However in order to solve the equation using numerical methods they must be discretised; SNAP uses finite difference in space and time, multi-group in energy and discrete ordinates in angle.

The application seeks to solve numerically the equation for the unknown ψ. This is done by iteration on the scattering source: a simple iteration where the value of ψ for the previous iteration is used in (1c) in order to update the values for ψ in (1a). Jacobi is used in energy groups [6]. This results in two implicit solve iteration loops for each time-step, typically called *outer* and *inner* iterations. The update of the value on the right hand size of (1) is partially updated in the outer and inner iterations. The outer updates the group-to-group scattering whilst the inner updates only the within group scattering.

It is the inversion of the streaming-collision operator (1a) that results in a *sweep* across the mesh and consists of the vast majority of the runtime of the SNAP proxy application. The discrete ordinates and spatial discretisation results in a data dependence between cells, as pictured in Fig. 1. In order to calculate the angular flux at the centre of each cell, the incoming face values are required from upwind neighbours. Outgoing fluxes are then calculated to satisfy downwind neighbour dependencies.

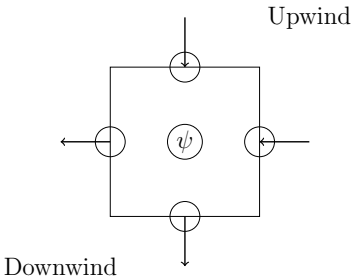


Fig. 1: Data dependency of the sweep algorithm

The original implementation of SNAP uses OpenMP threads to run in parallel over energy groups, and relies on automatic SIMD compiler vectorisation over angles. The spatial domain is decomposed across MPI ranks according to the KBA algorithm, discussed in more detail in Sect. 2.1. The sweep within each MPI rank is conducted in serial.

SNAP is designed to perform both time-dependent and time-independent deterministic transport solves. For the purposes of this paper we consider a time-dependent solution and therefore require storage of the full angular flux, rather than just the scalar flux and face angular fluxes. This imposes a significant memory capacity requirement and it is important that this is taken into account for discrete accelerators. The angular flux in a time-dependent solution requires storage for two time-steps each containing all points in the six-dimensional space/energy/angle domain.

2.1 The KBA Schedule

The data dependency between cells during the sweep requires some degree of sequential computation. Given that the angular flux is spatially decomposed across multiple compute nodes we require an algorithm or schedule to orchestrate the computation which adheres to the data dependency whilst maximising the concurrency in the wavefront.

Such a schedule was developed by Koch, Baker and Alcouffe (KBA) [15, 5, 7]. This method decomposes the spatial domain in one dimension less than the total dimension of the problem: a 1D decomposition for a 2D problem and a 2D decomposition for a 3D problem. Each node therefore contains the full domain of a single spatial dimension, and a portion of the remaining spatial dimensions. This is visualised in three-dimensions in Fig. 2. Each coloured $2 \times 2 \times 6$ block of the spatial domain is placed onto a separate compute node. Due to their long, thin, shape these sub-domains are often described as *pencils*.

A sweep begins from a corner of the grid, with a single angle. The schedule performs the sweep over all angles in this octant before sweeping all angles in the octant starting from the cell at the opposite end of the pencil. The next corner is then chosen until all angles are swept.

The communication occurs when a calculation reaches a sub-domain boundary; once a value of the angular flux is calculated on a boundary edge for a given angle, this value is communicated to the neighbouring node so that it may begin or continue with the sweep. This results in a natural over-decomposition or tiling of the spatial domain. Note that cells within a pencil on a given node are operated on in serial in the original algorithm.

The data dependency of the sweep gives a simple task dependency graph for the sweep of each angle. By combining the graphs of each angle, the workload can be *pipelined* so that the number of idle stages is reduced; as soon as one processor finishes their portion of work for one angle they begin the graph for the next angle.

This algorithm has been extended to consider more modern architectures with vector units on each processor [6]. Vector units allow for angles within an octant to be solved concurrently in a data parallel fashion.

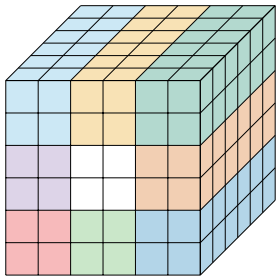


Fig. 2: KBA spatial decomposition in three dimensions

3 Related Work

Denovo is a three-dimensional steady state discrete ordinates code from Oak Ridge National Laboratory [12]. It is designed to solve the time independent version of the Boltzmann transport equation. Denovo uses the KBA algorithm for spatial decomposition (see Sec. 2.1) and a GMRES solve for within-group solves in contrast to the traditional iteration on a scattering source. The sweep routine occupies over 80% of typical Denovo execution time. As the Denovo code solves only the steady state version of the transport equation, only the scalar flux need be stored.

The Denovo code has also been ported to GPUs using CUDA, in particular targeted at the Titan supercomputer [4]. Octants and energy groups are decoupled so that they may be solved on different portions of the machine. The spatial domain is decomposed according to the KBA algorithm using MPI. The KBA algorithm is also applied to thread blocks to exploit spatial parallelism within the sub-domains. Each moment in the scalar flux is assigned to a warp.

The reference [4] shows scaling results on the early upgrade of the Jaguar supercomputer in preparation for the full upgrade to Titan when the system was installed with NVIDIA Fermi X2090 GPUs. Weak scaling results on this system show that the GPU version of the sweep was up to 3.5× faster than the CPU sweep. The same Denovo code was later run on Titan, which contains NVIDIA Tesla K20X GPUs. The sweep routine on the GPU was 4–6× faster than the CPU sweep running on 8 (of 16) cores per node, however the total code obtained an overall 2× speedup for the benchmark problems [11].

Whilst these results show good speedup, Denovo does not require storage of the angular flux as it is not time-dependent, and so is not constrained by this challenging memory capacity requirement. Due to the limited memory capacity

of the many-core devices compared to CPU DRAM capacity, a solution which requires storage of the angular flux requires a solution with better scalability in order to provide the total memory footprint by using many more nodes.

Additionally, because energy groups are decomposed into separate sets in Denovo they are solved on separate parts of the machine and so a fixed problem size artificially requires a large node count. Because of the speedup in the sweep routine from the GPU code, the majority of computation time shifts to other portions of the code which require all-to-all communication to distribute updated vectors.

A direct port of the SNAP mini-app using the CUDA framework by P. Wang et al. exists and uses a similar parallelisation scheme to the original code [23]. In our previous work we showed that on a single node this approach did not improve the performance over our benchmark CPU despite the use of GPUs [9].

4 A Many-core Implementation

The KBA schedule discussed in Sec. 2.1 describes the spatial decomposition between compute nodes but stipulates little about the computation within a node. It was shown that a hybrid approach where each MPI rank should utilise a number of cores to improve the scalability of the transport application [6], and this approach is implemented in the original SNAP implementation.

The authors have previously demonstrated that the intra-node solve of SNAP can be accelerated using many-core devices, such as GPUs [9, 10] providing single node speedups in line with the improved memory bandwidth of such devices. Our previous work was, to the best of our knowledge, the first time that such a significant speedup was achieved for a SNAP-like transport code using a many-core processor. The within-cell concurrency of angles within an octant and of energy groups via Jacobi was combined with the spatial concurrency of the wavefront within a node's spatial domain (as pictured in Fig. 3) in order to provide an effective scheme to exploit the increased resources of the many-core device over the multi-core CPU. However these improvements will not necessarily transfer to a multi-node domain.

KBA is used as a baseline for sweep scheduling algorithms, and so it is important to begin with combining this schedule with our accelerated intra-node scheme. However, this does not necessarily yield equivalent scaling results for many-core. Sweep algorithms rely on pipelining the work to maximise the number of busy processors while minimising the start up and tear down costs associated with the data dependency across the decomposed spatial domain; not all processors can begin at the start of the simulation and must wait until some internal boundary data reaches them before they can begin. The approaches to obtain good intra-node performance on accelerated devices require large aggregation factors in the angular and energy dimensions, and so if the length of the pipeline relies on these dimensions then the number of sweeps available to overlap is severely reduced and hence predicted scaling is poor; this is the case for the algorithms of Adams et al. [1, 2]. The KBA algorithm was originally designed to

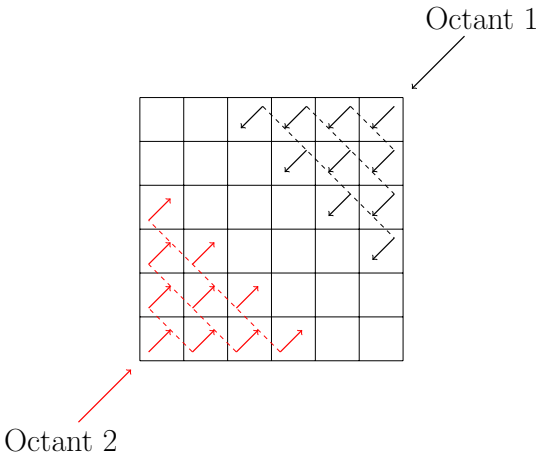


Fig. 3: Parallel sweep

allow for aggregating all angles in one octant [15]; the original authors of KBA found that while this reduced the theoretical parallel efficiency it performed better in practice. The SNAP mini-app uses KBA and so a fair comparison with the same spatial decomposition is required to test how our accelerated intra-node scheme performs at scale.

Our implementation of KBA and accelerated intra-node transport sweep uses MPI and OpenCL. OpenCL [18] is a parallel programming framework for heterogeneous platforms. The basic model consists of a host and a device such as a GPU, CPU, FPGA, etc. The host CPU and device have separate memory regions and memory must be explicitly managed by the programmer.

Units of work are defined, called work-items. Similar to OpenMP, this unit of work is typically the body of a loop. The work-items are mapped onto vector-lanes of the device architecture. In general these lanes are called processing elements, and operate in groups called compute units.

A kernel defines the problem size (number of work-items) and if required the work-group size. A work-group is a collection of work-items. Synchronisation is only possible within a work-group on the device and between work-groups via the host at the end of kernels. On a CPU, each work-group would be assigned to a core, and the work-items to vector lanes within that core. The kernel is enqueued on (offloaded to) the device as a whole and the OpenCL runtime schedules the computation of work-groups. The host controls the compute offload and memory transfer through the use of a command queue.

OpenCL therefore allows a very fine grained parallel programming notation of concurrent work. This means that it is very portable and can run on a variety of supported architectures from different vendors. In other work we have compared the performance of OpenCL and CUDA versions of this code and they are within a few percent of each other.

The computation of the angular flux kernel is designed so that each work-item computes the angular flux for one angle and energy group in one cell. The number of work-items is the number of angles in the octant multiplied by the number of energy groups, multiplied by the number of cells in the wavefront.

The original SNAP mini-app is written in Fortran, however OpenCL provides a C API, and so the required parts of the mini-app were rewritten in C. This includes the MPI communication routines so that the energy groups can be included in the kernel solve. The original SNAP calls the MPI routines from within the OpenMP parallel region, something which is not possible when using an offload model such as that in OpenCL. Moving the MPI calls outside of the group iteration loop allows us to send a single (larger) message containing data from all energy groups, rather than one message per energy group reducing the number times the penalty of message latency is paid.

Within a pencil sub-domain of the array according to the KBA decomposition, the many-core implementation runs in parallel over all cells within the local wavefront. We are free to choose how many XY-planes, or *chunks*, we compute before communicating to neighbouring nodes. Figure 4 shows a 2D pencil shape where six planes numbered 1–6 are computed before communication of downwind fluxes occurs. The numbers show which wavefront the cells are in. Notice that the widest (and thus most parallel) wavefront in this example occurs at the third and fourth wavefront. There is a start-up time until this maximum wavefront size is reached, and also a tear down cost returning back to a single cell just before the communication.

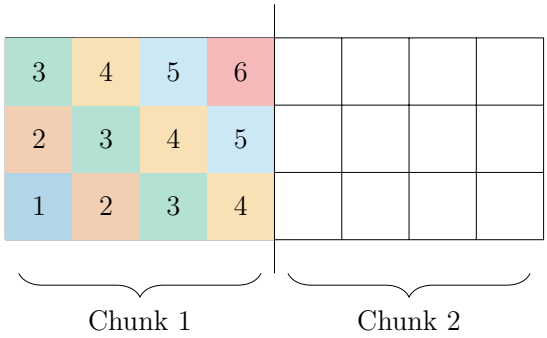


Fig. 4: Chunking of KBA communication

The original SNAP implementation operates in serial over cells in the chunks and so the start-up/tear-down costs do not appear there. For the many-core version the number of cells in the wavefront helps to generate enough independent work-items to keep the GPU saturated with enough work; we are running just a single cell's worth of work-items at the start and end of every chunk. Kernel invocations of wavefronts consisting of just a few cells will likely under-utilise the GPU, however generates a simpler communication pattern than an approach

suggested by Pennycook et al. for non-perpendicular chunking which avoids this repeated start-up cost [20].

As we are performing a time-dependent solve of the transport equation we require storage of two instances the angular flux. Our implementation requires that this data is stored resident on the accelerator device. We will therefore be limited by the memory capacity of the device. It is the subject of a future study to stream the data from host memory to the device memory. However, note that future self-hosting many-core systems, such as Intel's upcoming Knights Landing, may require a fully resident scheme such as presented in this paper.

5 Performance Model

The performance model proposed by Bailey and Falgout [3] predicts the runtime of transport sweeps derived from the number of stages in the sweep multiplied by the time to complete a stage. The time per stage is defined as the sum of the computational time per stage C, the communication time per stage B and the latency time per stage L. Whilst the model is just for the sweep time rather than total application time, given that the sweeps form the majority of the runtime of the application it should provide a reasonable approximation to runtime.

The latency time per stage L can be given as αK, for K messages per stage and a machine dependent value α representing the time to send a message (network latency). Likewise the communication time per stage B can be given as $\beta m \Gamma$, where Γ is the total number of cells in the communication faces, m is the number of bytes to communicate per cell, and β is a machine dependent value representing the time to send a byte (inverse network bandwidth). Both α and β are running micro-benchmarks on the machine in question.

We define the number of cells in a given three dimensional problem by the values N_x, N_y, N_z, the number of energy groups by N_g and the number of angles per octant N_m. We employ a two dimensional decomposition across $P_x \times P_y$ processors. We also define the number of chunks (XY-planes per communication) as η.

The number of stages is derived from the KBA scheduling algorithm. A stage consists of calculating a chunk of the KBA pencil and communicating the values to the neighbouring processors. Octants are pipelined in pairs where the sweeps start from the same processor in the four corners of the processor grid. Each one of the pairs behaves symmetrically. The processor in the opposite corner of the processor grid must wait to receive its first message. This is $P_x + P_y - 2$ stages before the final processor can begin. Once the processor in the opposite corner to the starting processor starts we assume that it can complete its work uninterrupted as messages will arrive on time (this fact was demonstrated by Adams et al. [1]). The workload is the number of chunks in the pencil for each of the two octants.

Therefore the total time T for S stages of the sweep is given in the model (2):

$$T = S(C + B + L) \tag{2a}$$
$$B = \beta m \Gamma \tag{2b}$$
$$L = \alpha K \tag{2c}$$
$$S = 4\left(P_x + P_y - 2 + \frac{2N_z}{\eta}\right) \tag{2d}$$

Finally we must define the computational time per stage C. For a code which uses the CPU for the sweeps we use the metric suggested by Bailey and Falgout (3). This is the product of a 'grind time', γ, per value of the angular flux to update in a stage. This γ gives us a tuning parameter to the model that is related to machine characteristics. This value allows us to use regression analysis to assist fitting the model to the data.

$$C_{\text{CPU}} = \gamma N_m N_g \eta \frac{N_x}{P_x} \frac{N_y}{P_y} \tag{3}$$

We alter this model for our GPU accelerated version. We hope to capture the idea that running a kernel on a GPU with few work-items takes similar time to a kernel with enough work-items to saturate a GPU device. Therefore the work per stage is related to the number of kernel enqueues, rather than the amount of work per cell, as specified in (4). Notice that the number of kernel enqueues is the number of wavefronts in each chunk, and γ represents an average kernel execution time.

$$C_{\text{GPU}} = \gamma\left(\frac{N_x}{P_x} + \frac{N_y}{P_y} + \eta - 2\right) \tag{4}$$

The model requires three machine dependent parameters: network latency (α), inverse bandwidth (β) (time to send a byte) and an estimate of the compute cost (γ). In order to obtain the first two metrics the benchmark b_eff was run on 2048 nodes of Titan, with one MPI rank per node [21]. The benchmark reported a network latency of $\alpha = 3.327\mu$s and an effective bandwidth of 575 MBytes/s per node, and so $\beta = 1/(575 \times 10^6)$.

We also ran the benchmark on 64 nodes of Piz Daint, with one MPI rank per node and achieved a network latency of $\alpha = 1.735\mu$s and an effective bandwidth of 6354 MBytes/s per node, and so $\beta = 1/(6354 \times 10^6)$. These network benchmarks show that Piz Daint has a higher performing network than Titan.

6 Results

6.1 Supercomputers

The Swiss National Supercomputing Centre supercomputer, *Piz Daint*, is a Cray XC30 supercomputer consisting of 5,272 nodes, each containing one 8-core Intel Xeon E5-2670 CPU and one NVIDIA Tesla K20X GPU. The nodes are connected

with the Aries interconnect according to a Dragonfly topology. The machine has an RMAX of 6.3 PFLOPS/s [22]. Piz Daint has CUDA 6.5 with OpenCL driver version 340.87.

The Oak Ridge National Laboratory supercomputer, *Titan*, is a Cray XK7 consisting of 18,688 nodes, each containing one 16-core AMD Opteron 6274 CPU and one NVIDIA Tesla K20X GPU. The nodes are connected with the Gemini interconnect according to a three-dimensional torus topology. The machine has an RMAX of 17.6 PFLOPS/s [22]. Titan has CUDA 7.0 with OpenCL driver version 346.99.

6.2 Weak Scaling

In order to assess the weak scaling of our implementation we drew inspiration from previous weak scaling transport studies [6]. The problem is chosen with sub-domains representing pins in a reactor, and so by weak-scaling the pencil-shaped blocks of work per node by increasing the number of pencils one can begin to build a model of a full reactor core. This means that both the total amount of work and the nature of that work per node remains constant as the problem is scaled up. We therefore use the following parameters:

- $4 \times 4 \times 400$ cells per MPI rank with mesh size of 1.0cm $\times$ 1.0cm $\times$ 100.0cm
- S_{32}: 136 angles per octant and 32 energy groups
- 1 time-step of 0.01 seconds with inner convergence criteria of 1.0E-5
- 2 orders of scattering expansion
- Computation of 4 XY planes before communication

We found that this problem requires four outer iterations, with between four and two inner iterations per outer. Each MPI rank requires 3.6 GBytes for storage of the angular flux. Note that we always perform an inner for each energy group in our implementation. Each NVIDIA Tesla K20X GPU has 6 GBytes of memory, of which 5.6 GBytes is usable by OpenCL. Our current implementation requires the data be fully resident on the GPU and so we are limited by the memory capacity of the GPU; our experiments are chosen to remain within this limit.

The scaling studies begin at 4 MPI ranks, and we assign one GPU per MPI rank. This requires both vertical and horizontal communication across the interconnect in the 2D MPI rank layout according to KBA. We also run the original implementation of SNAP on the CPUs in an appropriate configuration given the memory capacity per compute node. We found that two MPI ranks per socket/NUMA region provided the fastest run time on the CPU. On Titan this is four MPI ranks per node, and on Piz Daint this is two MPI ranks per node. This will allow some threading over the energy groups whilst also providing more spatial parallelism than a layout with fewer MPI ranks.

Titan The results from Titan can be seen in Fig. 5. It is clear that the GPU implementation provides a speedup of around 4$\times$ over the CPU implementation. The STREAM benchmark [17] achieves a memory bandwidth of 32 GBytes/s

on the Opteron CPUs, whilst GPU-STREAM [8] achieves 182 GBytes/s on the K20X GPUs, a 5.7× improvement in memory bandwidth of the GPU over the CPU. These benchmarks have no communication costs associated with them as they are simply run on a single node. We previously showed that we obtained a 4× speedup of single node performance of our intra-node accelerated scheme on Titan [9], and we can see here that this speedup is maintained as we weak-scale up to thousands of GPUs.

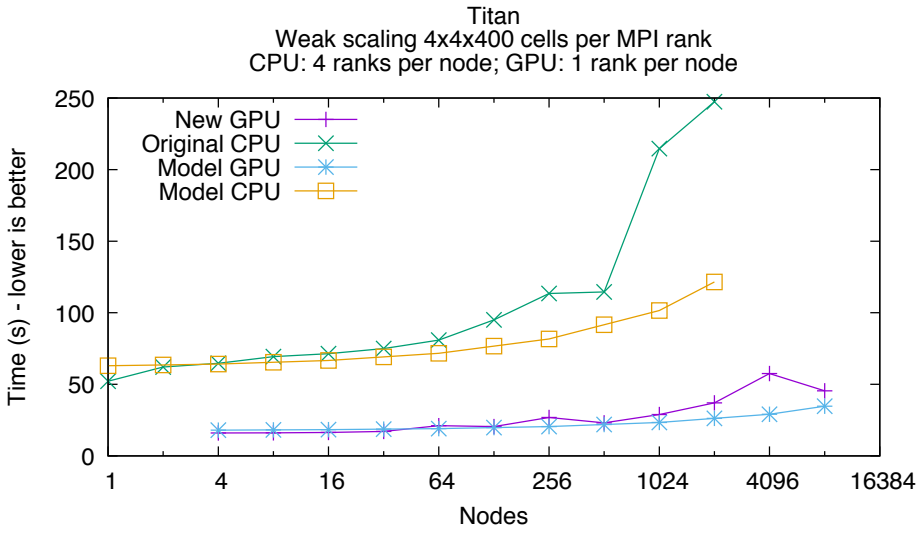


Fig. 5: Weak scaling SNAP on Titan

Our performance model can be used as a comparison to the results. For the CPU version of the model, we set $\gamma_{\text{CPU}} = 1.8 \times 10^{-8}$ in (3), and for the GPU version we set $\gamma_{\text{GPU}} = 8.0 \times 10^{-6}$ in (4). Note again that γ is a tuning parameter to the model that we cannot directly benchmark, and so we must choose a value with regression. We use the benchmarked values for network latency and bandwidth as given in Sec. 5.

The model predicts the runtime of eight octant sweeps, and so we multiply the prediction by the number of these as reported by the application; for the CPU this was 348 sweeps and for the GPU this was 11 sweeps. The number of sweeps is the same as the number of inners performed, noting that the CPU version sweeps (and counts) energy groups independently whereas the GPU code always sweeps all groups.

The model is plotted in Fig. 5 alongside the data collected from the implementation and it is clear that our scaling results are validated by the model. Given the choice of γ in the performance model, the scaling results are accurate on average to 17.6% for the GPU, and 18.8% for the CPU runs; these accuracies

are across a very wide range of scales, from 1 to 8,192 nodes. Note that we would *not* expect 'perfect' weak scaling of a horizontal line parallel to the x-axis in the figure. The achieved times of the CPU runs at large scale do start to deviate from the model, however this is likely due to network considerations on Titan, rather than being attributed to inaccuracies the model.

It should be noted that even on four nodes the accelerated implementation spends 55% of the sweep time in communication. At 8192 ranks with a total grid size of $512 \times 256 \times 400$, 84% of the sweep time is spent in inter-node communication. At this scale, we obtain 35% weak-scaling efficiency comparing the time for four nodes to 8192. This efficiency is similar to that obtained by the original implementation running on the CPUs. Therefore the benefits of our improved node-level scheme utilising GPUs have successfully carried across to the MPI version at scale, whilst running four times as fast.

The running times on Titan can be very variable, often up to a $2\times$ difference. Whilst the time taken to complete the sweeps increases this is only in the network time; the computation portion remains the same. There seems to be large variability in the network performance of the Titan supercomputer at scale, as has been shown previously for the Gemini network with the torus topology [19, 13]. In our results we have taken the minimum time across five runs. We see the variability even within consecutive runs within the same job allocation; node placement is not the major factor in the variability.

Piz Daint The results of the same weak scaling experiment run on Piz Daint can be seen in Fig. 6. The GPU implementation provides a speedup of up to $2\times$ over the original implementation running on the CPU. The STREAM benchmark [17] achieves a memory bandwidth of 41 GBytes/s on the single socket Xeon compared to 182 GBytes/s for GPU-STREAM on the K20X [8].

Piz Daint performs much better than Titan when it comes to time spent communicating. At the initial point of four nodes, 42% of the sweep time is spent in the communications compared to 55% on Titan. At 2048 nodes this has risen to 60% on Piz Daint compared to 80% on Titan demonstrating that Piz Daint has better network performance at scale. The communication time then doubles when doubling the number of nodes to 4096, but the compute time stays constant. Piz Daint consists of 5272 nodes and so this experiment is over 75% of the whole machine. The figure also shows that parallel efficiency was at around 70% up to 2048 nodes. This is an improvement over Titan, and this is likely due to the better interconnect. Communication times are between 1.3 and 3.2 times *faster* on Piz Daint than Titan over the weak scaling experiment. The computation time of the sweep is also around 14% faster on the GPUs in Piz Daint than those in Titan; this may be due to driver versions and the improved host CPUs in Piz Daint as the GPUs between the two machines are identical.

As before, our performance model can be used as a comparison to the results as is shown in Fig. 6. For the CPU version of the model, we set $\gamma_{\text{CPU}} = 9.0 \times 10^{-9}$ in (3) (half of what we chose for Titan), and for the GPU version we set $\gamma_{\text{GPU}} = 1.4 \times 10^{-4}$ in (4). The GPU scaling results match well to the predicted values,

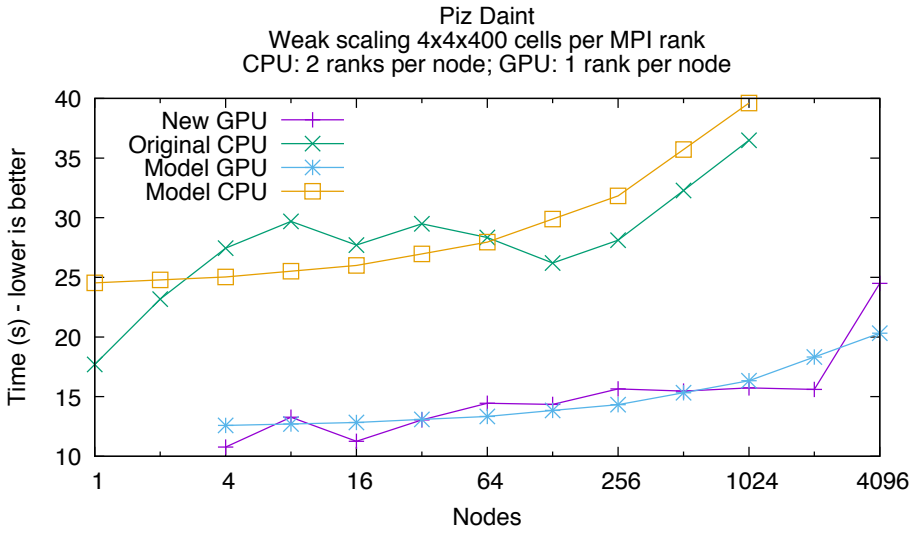


Fig. 6: Weak scaling SNAP on Piz Daint

with an average of a 8.6% difference. The CPU model lies within 11.9% of the achieved values, but does not capture the wave-like trend of the obtained results; importantly the upward curve indicating an increase at node counts beyond 128 is captured by the model showing the scaling performance as a whole with enough accuracy to be of use.

After each inner and outer iteration the convergence of the solution is compared. The maximum change since the previous iteration of the scalar flux per energy group is computed and shared between all MPI nodes. This requires an `MPI_Allreduce` operation. For the outer iteration convergence check, only the maximum change overall is needed and so only a reduction operation over a single value is required. As we weak scale to more nodes, the time to perform these reduction operation increases from less than 1% to approximately 8% of the runtime. The sweep still dominates the solution time requiring around 90% of the total time.

6.3 Choice of Chunk Size

The larger the chunk size, the fewer times we have to exhibit the start-up/tear-down costs (see Fig. 4) and so the total time spent in the computation kernel reduces. Note the total amount of work remains constant for all chunk sizes. However by increasing the chunk size we increase the size of the messages that we need to send and hence copy from the GPU to the CPU, over the network and back onto the neighbouring GPU. Figure 7a shows the effect that chunk size has on performance. The time to copy the data is negligible at small chunk sizes, but the computation time is larger than the minimum achievable with a larger

chunk size. We are also not using much network bandwidth as we are sending many comparatively small messages; approximately 140 kilobytes per message for a chunk size of one.

Conversely for large chunk sizes the data transfer time over PCIe becomes noticeable and the network is required to send fewer, larger messages; approximately 28 megabytes per message for a chunk size of 200. Note that in this case we are also holding the sweep up and the MPI rank starting the sweep will sit idle for much of the sweep execution time.

The minimum height bar in Fig. 7a will give the best trade-off between computation time per node, network bandwidth *and* latency, and keeping all nodes busy throughout the sweep by providing enough chunks along the pencil length for spatial over-decomposition. Due to the data dependency of the wavefront sweep MPI communications cannot be overlapped with computation as each rank must always have a blocking receive. As can be seen from Fig. 7a, for Titan the best chunk size we found was eight; on Piz Daint we found the best was 20.

Our performance model from Sec. 5 (using $\gamma_{\text{GPU}} = 8.0 \times 10^{-6}$ as before) also shows this increase in running time with larger chunk size, as pictured in Fig. 7b. The model predicts the lowest running time could be achieved with chunk size of 2, however most small chunk sizes are very similar in runtime. The predicted values are most sensitive to the bandwidth parameter β.

6.4 Strong Scaling

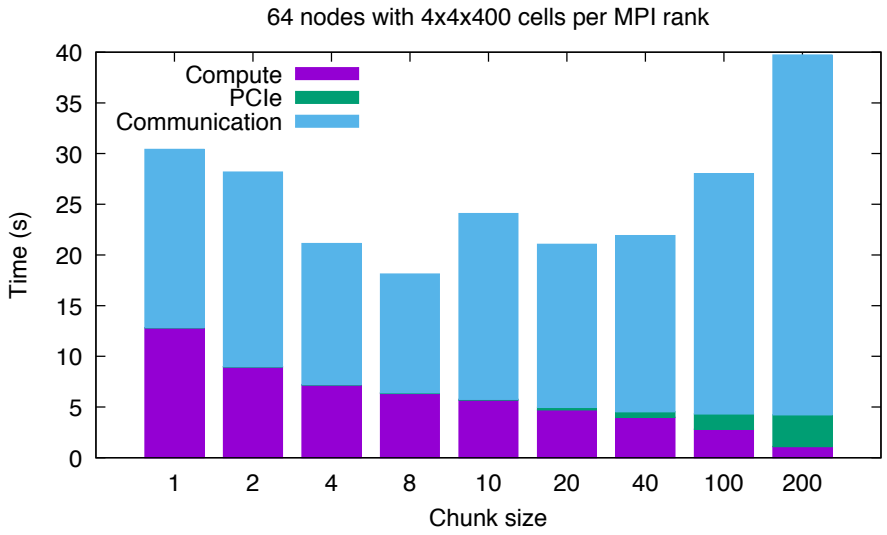
The strong scaling of deterministic transport has always presented a challenge in the real world. The limit imposed by memory capacity means a realistic starting point for a strong scaling study (at a few nodes) is difficult.

To begin an investigation into strong scaling, a relatively large grid was chosen with common angular and energy group fidelity. The physical spatial and time dimensions were chosen only so that the problem converged within a short amount of time. We therefore use the following parameters which has a memory footprint of approx. 690 GBytes:

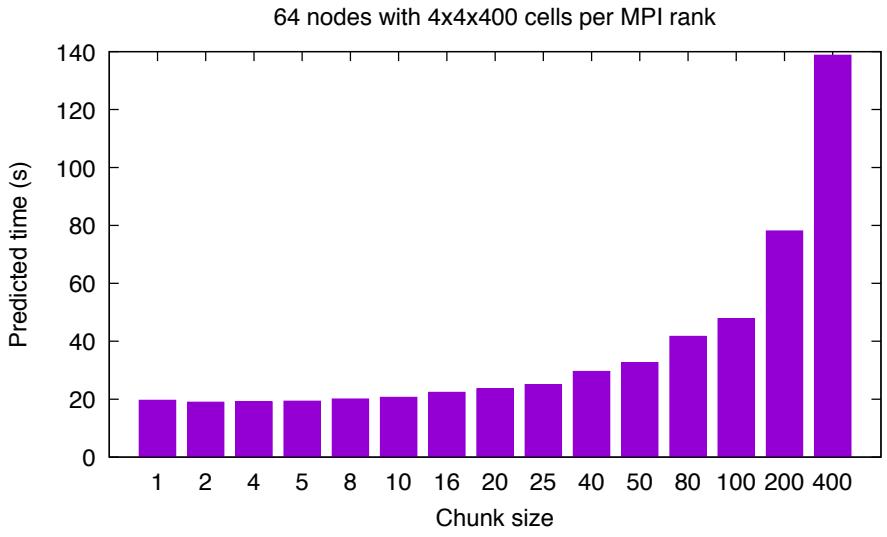
- $256 \times 256 \times 256$ cells with mesh size of 2.56cm $\times$ 2.56cm $\times$ 2.56cm
- S_8: 10 angles per octant and 32 energy groups
- 1 time-step of 0.01 seconds with inner convergence criteria of 1.0E-5
- 2 orders of scattering expansion
- Computation of 4 XY planes before communication

When running this problem on Titan it was not possible to run the original CPU code with hybrid MPI+OpenMP as the original code would hang, and so the results were collected in this section with flat MPI with 16 ranks per node. The original code required 347 inners to converge.

Figure 8a shows the strong scaling results obtained on Titan with the original CPU implementation and in Figure 8b our new version running on the GPUs. Predicted times for both implementations are added, using predicted grind times of $\gamma_{\text{CPU}} = 1.8 \times 10^{-8}$ and $\gamma_{\text{GPU}} = 3.0 \times 10^{-5}$.

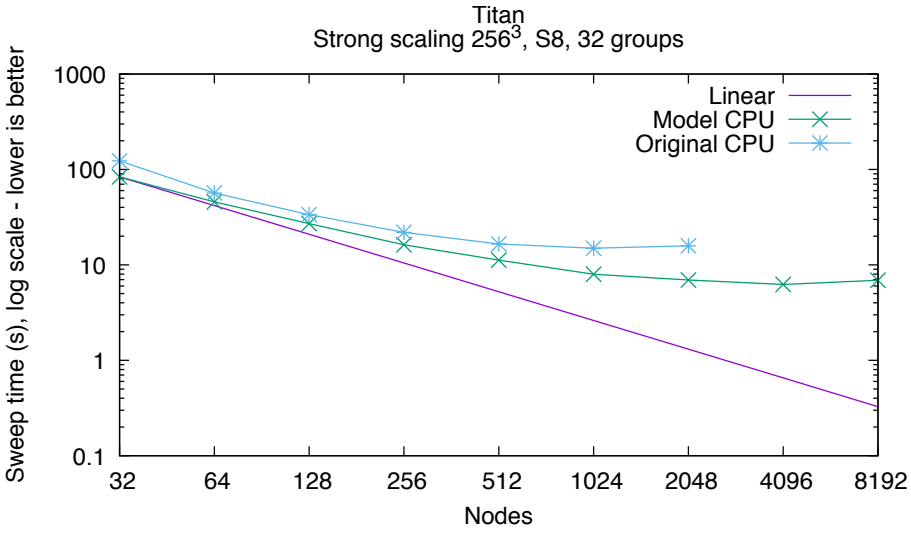


(a) Experimental

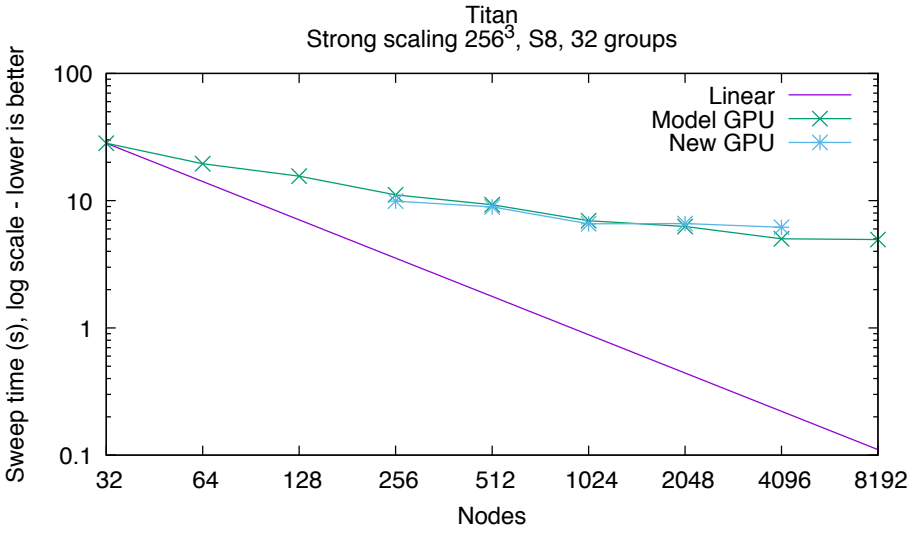


(b) Predicted

Fig. 7: Sweep timings for various chunk sizes on 64 nodes of Titan



(a) CPU



(b) GPU

Fig. 8: Sweep timings for strong scaling

A linear line is also plotted to demonstrate that the model (and obtained times) are not expected to scale linearly with node count; this shows that strong scaling is a real challenge for transport code. Note that the CPU version scaled nearly linearly to begin with; this implies that in order to improve the scalability of the many-core version we must reduce the node count by increasing the potential memory capacity of each GPU node. However, our solution strong scales at least as well as the state of the art for deterministic transport.

7 Conclusion

Many-core devices such as GPUs deliver memory bandwidth advantages over traditional multi-core CPU architectures along with improved hiding of memory latency and so are attractive for increasing performance of memory bound codes, of which deterministic transport is one such. However, deterministic transport is also network bound and is sensitive to careful balance of intra-node work levels in order to obtain good performance on a large system.

We used the KBA algorithm for spatial decomposition and saturated the GPU devices with work by solving all angles and energy groups for all cells in the local wavefront. We have shown that good scaling is still possible when using GPUs to accelerate the computation on the node. The GPUs themselves are also being utilised well as we have shown they are obtaining performance increases relative to their memory bandwidth.

By accelerating the intra-node solve, we do however make the final performance of the transport code more dependent on the network. Over half of the running time is spent in communication with our accelerated version. Placing more GPU devices in a single node to avoid network communication and providing increased bandwidth and lower latency network interfaces are two possible solutions for this at a hardware level. Additionally if the memory capacity of GPU devices was increased it would be possible to run on fewer MPI ranks.

Our accelerated version of the SNAP mini-app shows that time-dependant solves of the transport equation scale well enough using the regular KBA algorithm. This approach is therefore valuable in the preparation of transport solvers for the upcoming many-core Department of Energy CORAL procurement systems, Sierra and Summit.

7.1 Future work

We intend to investigate schemes to improve the strong scalability by increasing the size of the spatial sub-domain per MPI rank. We can stream the data to the device so we are not limited by the memory capacity of the device, instead using the larger memory capacity of the host and thus reducing the number of MPI ranks required. Additionally we will implement our scheme in OpenMP 4.5 to compare a single source implementation across GPUs and Knights Landing Xeon Phis.

Acknowledgements

This work has been financially supported by A.W.E. The authors would like to thank the University of Bristol High Performance Computing Group and Intel Parallel Computing Center; and Maria Grazia Giuffreda of CSCS at the Swiss National Supercomputing Centre for access to Piz Daint. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

1. Adams, M.P., Adams, M.L., Hawkins, W.D., Smith, T., Rauchwerger, L., Amato, N.M., Bailey, T.S., Falgout, R.D.: Provably optimal parallel transport sweeps on regular grids. International Conference on Mathematics, Computational Methods & Reactor Physics pp. 2535–2553 (2013)
2. Adams, M.P., Adams, M.L., Mcgraw, C.N., Till, A.T., Bailey, T.S.: Provably Optimal Parallel Transport Sweeps with Non-contiguous Partitions. In: Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method. pp. 1–19. No. ANS MC2015, American Nuclear Society, Nashville, Tennessee (2015)
3. Bailey, T.S., Falgout, R.D.: Analysis of Massively Parallel Discrete-Ordinates Transport Sweep Algorithms with Collisions. In: International Conference on Mathematics, Computational Methods, and Reactor Physics. pp. 1–15. American Nuclear Society, New York, New York, USA (2009)
4. Baker, C., Davidson, G., Evans, T.M., Hamilton, S., Jarrell, J., Joubert, W.: High performance radiation transport simulations: Preparing for TITAN. 2012 International Conference for High Performance Computing, Networking, Storage and Analysis pp. 1–10 (2012)
5. Baker, R., Koch, K.: An Sn Algorithm for the Massively Parallel CM-200 Computer. Nuclear Science and Engineering pp. 312–320 (Mar 1998)
6. Baker, R.S.: An Sn Algorithm for Modern Architectures. In: Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method. No. ANS MC2015, American Nuclear Society, Nashville, TN (2015)
7. Baker, R., McGhee, J., Koch, K., Morel, J.: Two Sn Algorithms for the Massively Prallel CM-200 Computer. Submitted to Nuclear Science and Engineering (1996)
8. Deakin, T., McIntosh-Smith, S.: GPU-STREAM: Benchmarking the achievable memory bandwidth of Graphics Processing Units (poster). In: Supercomputing. Austin, Texas (2015)
9. Deakin, T., McIntosh-Smith, S., Gaudin, W.: Expressing Parallelism on Many-Core for Deterministic Discrete Ordinates Transport. In: Workshop on Representative Applications at IEEE Cluster. Chicago (2015)
10. Deakin, T., McIntosh-Smith, S., Martineau, M., Gaudin, W.: An Improved Parallelism Scheme for Deterministic Discrete Ordinates Transport. International Journal of High Performance Computing Applications (special issue) (2015), In press
11. Evans, T.M., Joubert, W., Hamilton, S.P., Johnson, S.R., Turner, J.A., Davidson, G.G., Pandya, T.M.: Three-Dimensional Discrete Ordinates Reactor Assembly Calculations on GPUs. In: Joint International Conference on Mathematics

and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method. No. ANS MC2015, American Nuclear Society, Nashville, Tennessee (2015)
12. Evans, T.M., Stafford, A.S., Slaybaugh, R.N., Clarno, K.T.: Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE. Nuclear Technology 171, 171–200 (2010)
13. Freed, J., Gupta, S., Tiwari, D.: An Analysis of Network Congestion in the Titan Supercomputers Interconnect (poster). In: Supercomuting. pp. 1–2 (2015)
14. Hoisie, A., Lubeck, O., Wasserman, H.: Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications (2000)
15. Koch, K., Baker, R., Alcouffe, R.: Solution of the first-order form of three-dimensional discrete ordinates equations on a massively parallel machine. Transactions of the American Nuclear Society 65, 198–199 (1992)
16. Lewis, E., Miller, W.J.: Computational methods of neutron transport. American Nuclear Society (1993)
17. McCalpin, J.D.: Memory Bandwidth and Machine Balance in Current High Performance Computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter pp. 19–25 (Dec 1995)
18. Munshi, A.: The OpenCL Specification, Version 1.1 (2011)
19. Pedretti, K., Vaughan, C., Barrett, R., Devine, K., Hemmert, K.S.: Using the Cray Gemini Performance Counters
20. Pennycook, S.J., Hammond, S.D., Mudalige, G.R., Wright, S.A., Jarvis, S.A.: On the acceleration of wavefront applications using distributed many-core architectures. Computer Journal 55(2), 138–153 (Jul 2012)
21. Rabenseifner, R., Schulz, G.: B_eff v3.6. https://fs.hlrs.de/projects/par/mpi/b_eff/
22. Strohmaier, E., Simon, H., Dongarra, J., Meuer, M.: Top 500 - November 2015. http://www.top500.org (2015), www.top500.org
23. Villa, O., Johnson, D.R., OConnor, M., Bolotin, E., Nellans, D., Luitjens, J., Sakharnykh, N., Wang, P., Micikevicius, P., Scudiero, A., Keckler, S.W., Dally, W.J.: Scaling the power wall: a path to exascale. In: Supercomputing (2014)
24. Zerr, R.J., Baker, R.S.: SNAP: SN (Discrete Ordinates) Application Proxy - Proxy Description. Tech. rep., LA-UR-13-21070, Los Alamos National Labratory (2013)