

Experiences with Porting and Modelling Wavefront Algorithms on Many-Core Architectures

S.J. Pennycook, S.D. Hammond, G.R. Mudalige and S.A. Jarvis

High Performance Systems Group,
Department of Computer Science,
University of Warwick,
CV4 7AL

September 29th, 2010

Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU
- 3 Performance of GPU Solution
- 4 Performance Modelling
- 5 Conclusions

Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU
- 3 Performance of GPU Solution
- 4 Performance Modelling
- 5 Conclusions

Predictive Performance Modelling

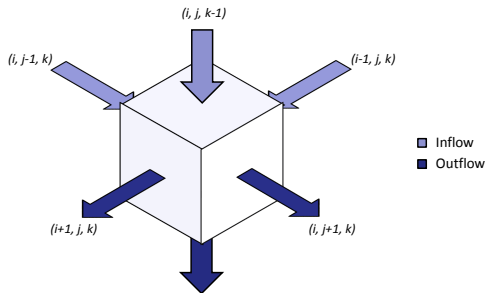
- High-performance computing (HPC) applications can take years to develop and often outlast the **host machines** for which they were **originally developed**.
- Important to ensure **performant execution** of these applications on current and future hardware.
- Performance models provide insight into application behaviour, permitting exploration of design space **without changes to code**.
- Particularly useful for the purpose of **procurement**.

Wavefront Applications (1 / 3)

- Wavefront applications form a large portion of HPC workloads at supercomputing centres (e.g. LANL, AWE).
- The behaviour of wavefront applications on CPUs is already well-understood [Mudalige et al, 2006, 2008].
- Strict **data dependency** presents an additional challenge. Realistic codes are **rarely embarrassingly parallel throughout**.

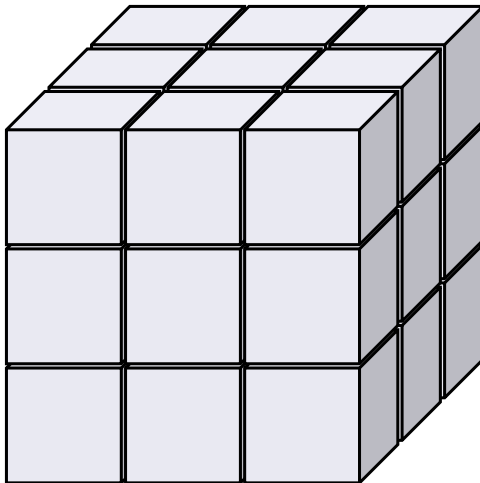
Wavefront Applications (2 / 3)

- 3D wavefronts operate on grids of size $N_x \times N_y \times N_z$.
- Data dependency:

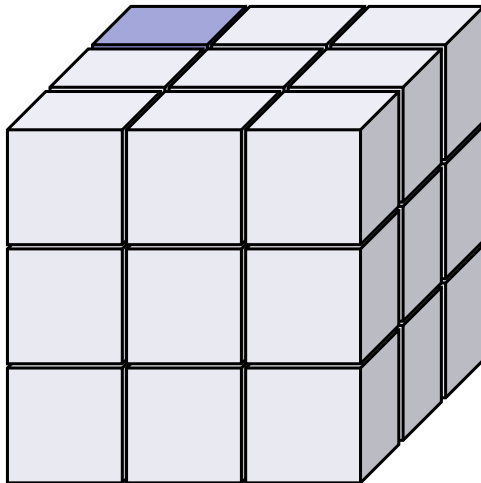


- **Hyperplane Algorithm** [Lampert, 1974]: Compute all grid-points on hyperplane $f = i + j + k$ in parallel.

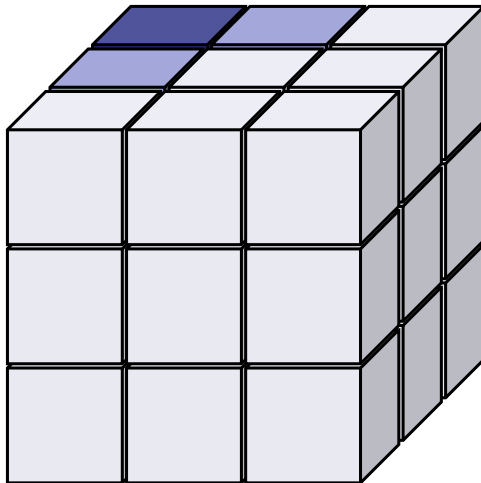
Wavefront Applications (3 / 3)



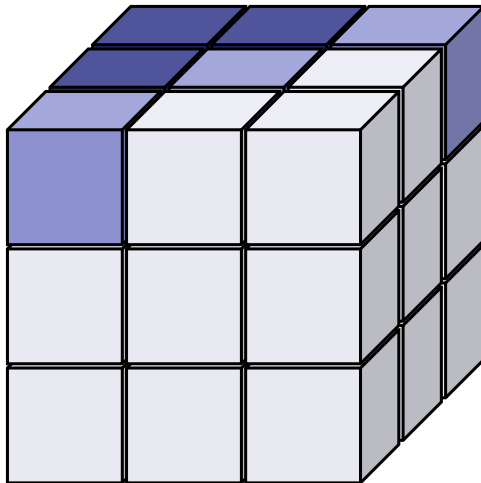
Wavefront Applications (3 / 3)



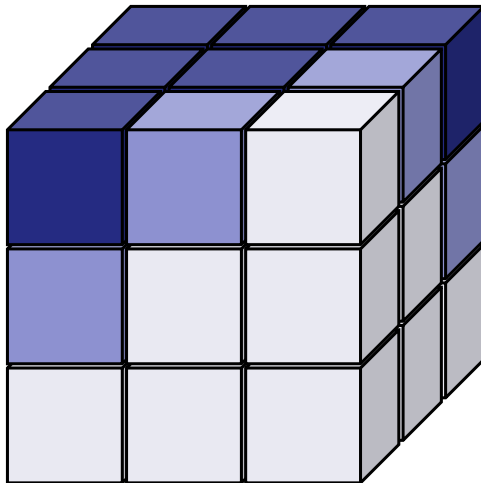
Wavefront Applications (3 / 3)



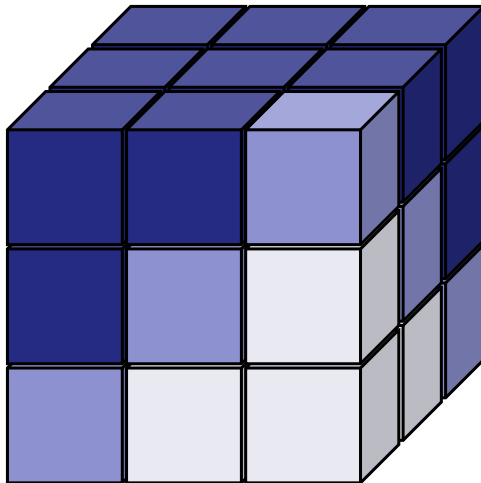
Wavefront Applications (3 / 3)



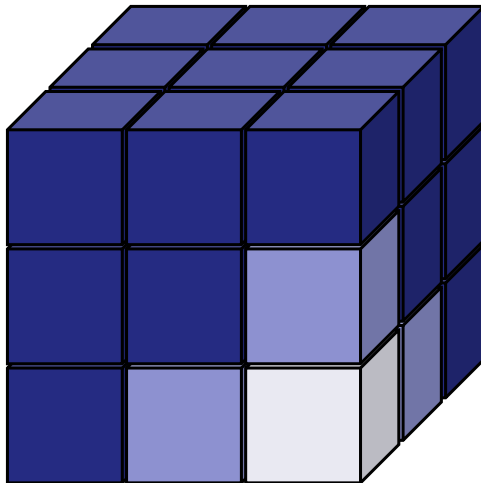
Wavefront Applications (3 / 3)



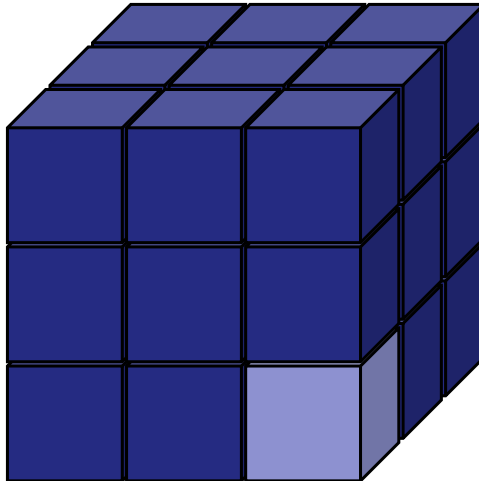
Wavefront Applications (3 / 3)



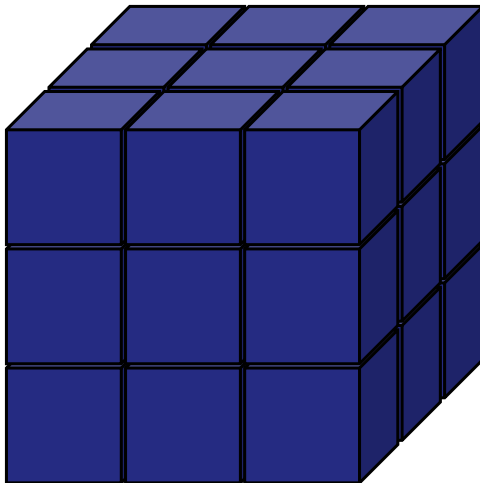
Wavefront Applications (3 / 3)



Wavefront Applications (3 / 3)



Wavefront Applications (3 / 3)



Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU**
- 3 Performance of GPU Solution
- 4 Performance Modelling
- 5 Conclusions

LU benchmark from the NAS Parallel Benchmark (NPB) suite selected as a case study, to represent a real-world application:

- Methods called over hundreds of iterations.
- Methods exhibit different parallel behaviours.
- Wavefront dependency accounts for **majority of execution time**.

Naïve Wavefront Kernel

- Global synchronisation achieved via the host, through **separate kernels**.
- Kernels launch $O(N^2)$ threads, with each thread mapping to a **single (i, j) pair**.
- Threads with no grid-points to process in the current wavefront step **exit immediately**.

Kernel Optimisations

- Re-order threads using mapping shown in the Figure.
- Rearrange memory to ensure **coalescence**.
- No utilisation of shared memory (yet).

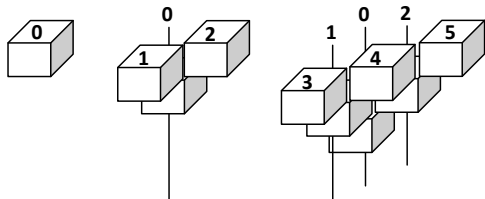


Figure: An efficient mapping of threads to the data grid.

Application Optimisations

- Important to minimise the overheads of PCI Express (PCIe) transfers by:
 - Porting all methods to the GPU.
 - Moving MPI packing/unpacking step onto the GPU.
- Simple **memory rearrangement kernel** ensure coalescence in all compute kernels.

Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU
- 3 Performance of GPU Solution**
- 4 Performance Modelling
- 5 Conclusions

Single Workstation (1 / 2)

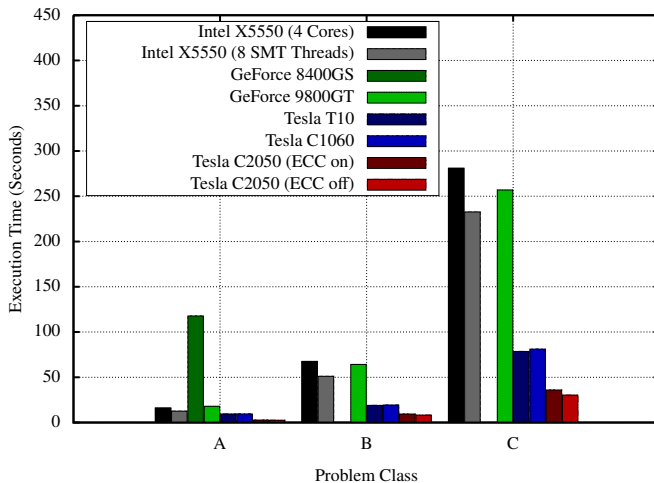


Figure: Execution times for a single workstation in single precision.

Single Workstation (2 / 2)

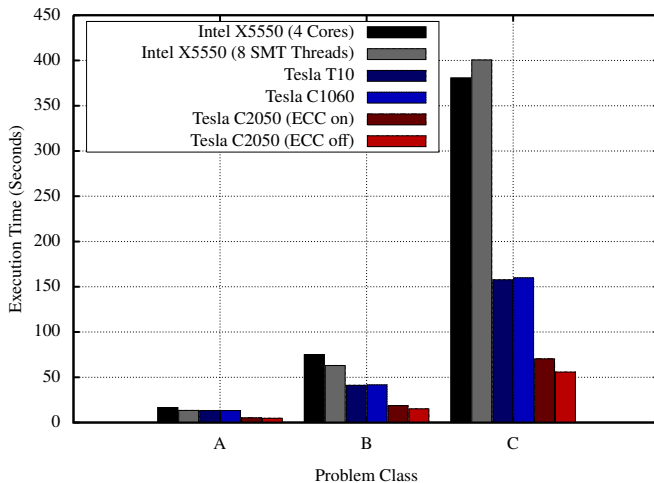


Figure: Execution times for a single workstation in double precision.

Cluster Scale (1 / 2)

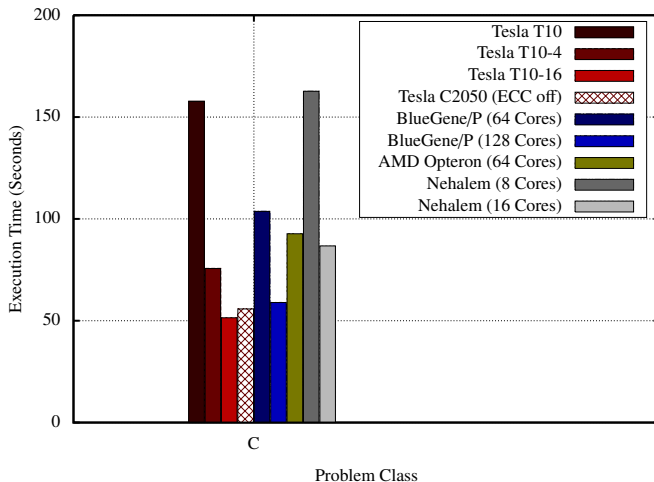


Figure: Execution times for clusters running Class C in double precision.

Cluster Scale (2 / 2)

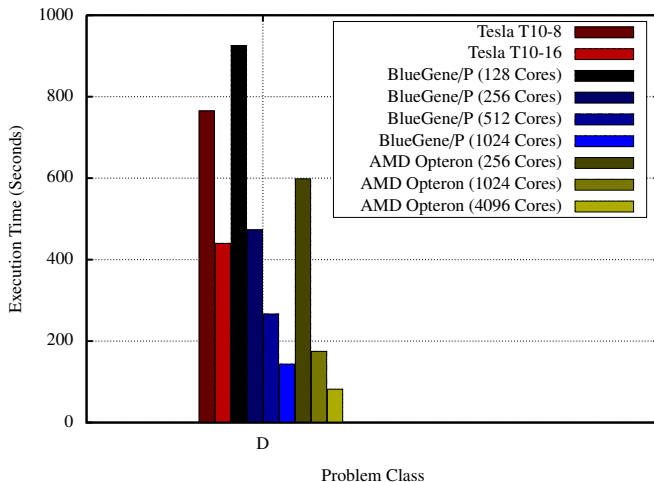


Figure: Execution times for clusters running Class D in double precision.

Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU
- 3 Performance of GPU Solution
- 4 Performance Modelling**
- 5 Conclusions

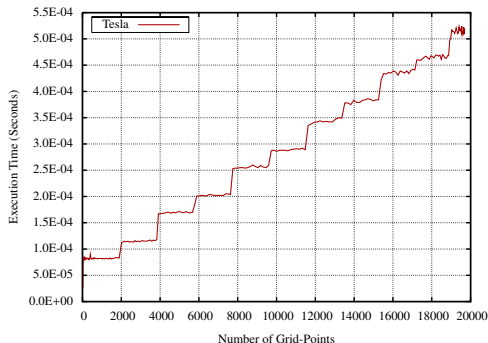
Distributed Memory CPU Model (1 / 2)

- Existing distributed-memory model captures critical path time, covering two aspects of wavefront execution:
 - ① Serial **compute time** per processor.
 - ② MPI **communication time** between processors.
- Compute time typically acquired via **code instrumentation** or a CPU submodel.
- Communication time typically calculated through **network benchmarking**.

Distributed Memory CPU Model (2 / 2)

- “Plug-and-play” nature of model means it is **easily extensible**.
- Calculate compute time via **GPU submodel**.
- Add additional **PCIe overheads** as a setup cost in communication model.

GPU Model (1 / 4)



- Execution time increases in steps.
- Steps correspond to the number of blocks scheduled to each stream multiprocessor (SM).

Figure: Execution times for Tesla C1060.

Model the execution time for a given number of grid-points, g , thus:

$$B(g) = \lceil g / (S \times T) \rceil$$

$$t(g) = h + (C \times (B(g) - 1) \times h)$$

where:

B = number of blocks per SM

S = number of SMs

T = number of threads per block

h = time to process a single grid-point

C = “concurrency factor”

GPU Model (3 / 4)

- Tesla C1060 has 30 SMs which can run 2 blocks of 64 threads concurrently.
- Small increase in execution time every $30 \times 64 = 1920$ threads.
- Large increase in execution time every $30 \times 64 \times 2 = 3840$ threads.

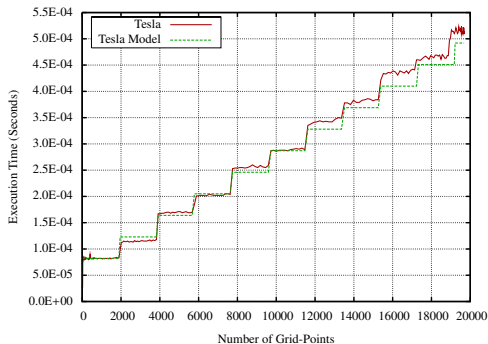


Figure: Execution times for Tesla C1060 and model predictions.

GPU Model (4 / 4)

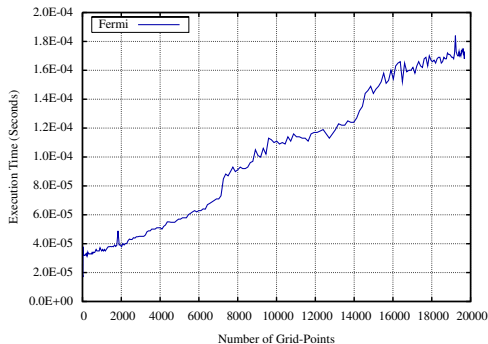


Figure: Execution times for Tesla C2050.

- Only **two** notable steps, once every $14 \times 64 \times 8 = 7168$ threads.
- Remainder of execution time appears to be **linear**.
- Execution time grows more inconsistent as the number of grid-points increases.

Outline

- 1 Background
- 2 Implementing Wavefronts on the GPU
- 3 Performance of GPU Solution
- 4 Performance Modelling
- 5 Conclusions**

Conclusions and Future Work

- Performance modelling offers useful insights into [suitable optimisations](#) for the GPU.
- Also permit [performance projections](#) for future architectures and at large scale.
- For Tesla, it appears we may be able to produce models at a [high level of abstraction](#).
- Future work:
 - Project execution times for GPUs at scale, for [fair comparison](#) with “pure” MPI solutions.
 - Construct similar models for [other application kernels](#).
 - Extend the model to [Fermi and future architectures](#).

References



L. Lamport.

The Parallel Execution of DO Loops.

Commun. ACM, 17(2):83–93, 1974.



G. R. Mudalige, S. A. Jarvis, D. P. Spooner, and G. R. Nudd.

Predictive Performance Analysis of a Parallel Pipelined Synchronous Wavefront Application for Commodity Processor Cluster Systems.

In Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2006). IEEE Computer Society, September, 2006.



G. R. Mudalige, M. K. Vernon, and S. A. Jarvis.

A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures.

In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008). IEEE Computer Society, April, 2008.