Predictive Analysis and Optimisation of Pipelined Wavefront Computations*

G.R. Mudalige, S.D. Hammond, J.A. Smith, S.A. Jarvis
High Performance Systems Group, University of Warwick
Coventry, CV4 7AL, UK
{g.r.mudalige, sdh, jas, saj}@dcs.warwick.ac.uk

Abstract

Pipelined wavefront computations are a ubiquitous class of parallel algorithm used for the solution of a number of scientific and engineering applications. This paper investigates three optimisations to the generic pipelined wavefront algorithm, which are investigated through the use of predictive analytic models. The modelling of potential optimisations is supported by a recently developed reusable LogGP-based analytic performance model, which allows the speculative evaluation of each optimisation within the context of an industry-strength pipelined wavefront benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE). The paper details the quantitative and qualitative benefits of: (1) parallelising computation blocks of the wavefront algorithm using OpenMP; (2) a novel restructuring/shifting of computation within the wavefront code and, (3) performing simultaneous multiple sweeps through the data grid.

1 Introduction

Pipelined Wavefront Computations, originally described as "hyperplane methods" by Lamport [1], form a large portion of the high performance scientific computing workload at supercomputing sites such as the Los Alamos National Laboratory (LANL) in the US and the Atomic Weapons Establishment (AWE) in the UK. Since these algorithms heavily consume parallel computing resources, there is naturally a great deal of interest in the development of faster or optimised methods for running or solving wavefront problems. In recent work [2] a reusable *plug-and-play* analytic model for the predictive analysis of pipleined wavefront computations was presented. The model is based on LogGP and requires only a few input parameters to obtain a predictive model for a range of wavefront applications on parallel architectures. Moreover, [2] demonstrated the significant utility of analytic models in providing insights into performance issues for this class of application running on modern HPC systems.

In this paper we utilise this reusable analytic model to speculatively assess the benefits of three algorithmic optimisations. We then evaluate each optimisation within the context of an industry-strength benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment. The aim of this study is therefore to assess whether code optimisations explored through analytic performance models can be evaluated before subsequent code changes are commissioned. If so, this has significant implications with regard to the future development/maintenance costs of such codes – the investment required for code optimisations need only be realised if the perceived benefits, demonstrated through the models, are deemed significant enough to warrent the additional effort and cost. We investigate the following possible code optimisations: (1) Parallelising the main computation block of a wavefront code using OpenMP pragmas; (2) Restructuring/shifting computation within the wavefront algorithm; (3) Performing multiple simultaneous sweeps.

The remainder of the paper is organised as follows: Section 2 provides a brief introduction to the general operation of wavefront codes and summarises the components of the reusable analytic performance model found in [2]; Section 3 describes the main contributions of this research and provides speculative analysis of the quantitative and qualitative benefits of each of the proposed optimisations; the conclusions are to be found in Section 4.

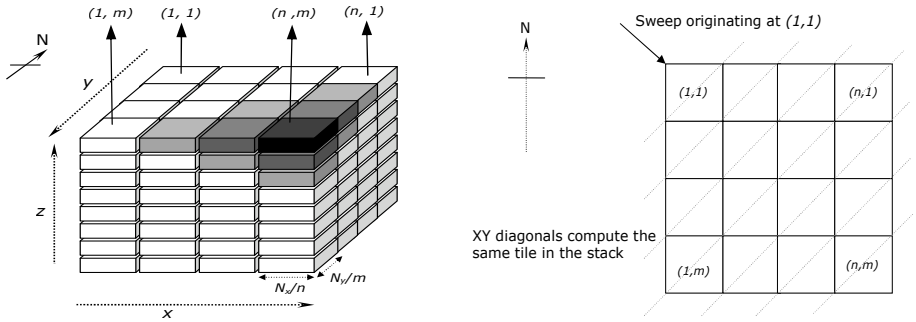
2 Background

2.1 Pipelined Wavefront Computations

Typical implementations of wavefront algorithms operate over a three-dimensional data grid (although the algorithm operates equally well for 1 or 2-dimensional problems) of size $N_x \times N_y \times N_z$. A two dimensional processor array of size $n \times m$ is employed with data

*This work was sponsored in part by grants CDK0660 and CDK0724 from the UK Atomic Weapons Establishment.

978-1-4244-3750-4/09/$25.00 ©2009 Crown



(a) 3D data grid mapping on to a 2D processor array (b) wavefront operation on a 2D processor array

Figure 1: Pipelined wavefront operation

decomposed into per-processor columns sized $N_x/n \times N_y/m \times N_z$. Where the dimensions do not equally split, load balancing is employed to improve parallel efficiency. The data column held by each processor is commonly thought of as a 'stack' of N_z tiles, each $N_x/n \times N_y/m \times 1$ in size (see Figure 1(a)).

Listing 1: General pipeleined wavefront algorithm

```
FOR EACH TILE DO
        PRE–COMPUTATION
        RECEIVE FROM WEST PROCESSOR
        RECEIVE FROM NORTH PROCESSOR
                MAIN COMPUTATION ON TILE
        SEND TO EAST PROCESSOR
        SEND TO SOUTH PROCESSOR
END FOR
```

The wavefront algorithm (see Listing 1) proceeds from a corner of the processor array. The first processor will solve the first tile in its stack, communicating the edge data to its immediate neighbours on completion. The neighbours then solve the first tile in their stack whilst the originating processor solves its second. This creates a diagonal sweep through the processor array as depicted in Figure 1(b), giving rise to the terms 'sweep' and 'wavefront'. The diagonals of the 2D processor array act as stages of a pipeline, where processors on each diagonal compute one tile at a time. A sweep is complete when all processors have solved all of the tiles in their stack. The shaded tiles in Figure 1(a) depict the tiles that are processed during the final three wavefronts (or final three sweep steps) - light gray, then medium gray, then dark gray - belonging to a sweep that originated at the bottom-most tile on processor $(1, 1)$, ending at the topmost tile on (n, m). Wavefront applications may perform multiple sweeps from multiple corners of the 3D data grid. The pre-computation block is optionally present in some applications, but as it does not require any off-processor boundary values to compute, the blocking of computation only occurs until receives are satisfied from processors to the west and to the north.

Table 1: Plug-and-Play Reusable Model Parameters

Parameter	Chimaera
N_x, N_y, N_z	$Inputsize$
W_g	$measured$
$W_{g,pre}$	0
$H_{tile}(cells)$	1
n_{sweeps}	8
n_{full}	4
n_{diag}	2
$T_{nonwavefront}$	$T_{allreduce} + \delta_h$
$MessageSize_{EW}$ (Bytes)	$8H_{tile} \times \#angles \times N_y/m$
$MessageSize_{NS}$ (Bytes)	$8H_{tile} \times \#angles \times N_x/m$

2.2 The Plug-and-Play Analytic Model

There are many variations to this general wavefront operation [2]. Applications such as Sweep3D from LANL [3], Chimaera from AWE and LU [4] from the NAS benchmark suite employ wavefronts to solve their underlying mathematical and scientific problems in a range of different configurations and setups. The reusable analytic model presented in [2] was developed specifically to capture and evaluate the effects of many, if not all, of these variations. The model comprises of a set of parameters that easily enables a user to define different structures to the wavefront order. Table 1 lists these parameters and their values for the AWE Chimaera benchmark code. We use this industry-based code as the base application upon which we evaluate the benefits of the optimisations in this paper. W_g and $W_{g,pre}$ account for the computation time per cell, that is they represent the main computation and the pre-computation respectively. $W_{g,pre}$ is set to zero in Chimaera since it has no computational sections in the algorithm, prior to the MPI receives. These compu-

Table 2: Equations of the Reusable Analytical Wavefront Model

$W_{pre} = W_{g,pre} \times H_{tile} \times N_x/n \times N_y/m$	(1)
$W = W_g \times H_{tile} \times N_x/n \times N_y/m$	(2)
$StartP_{1,1} = W_{pre}$	(3)
$StartP_{i,j} = max(StartP_{i-1,j} + W_{i-1,j} + Total\_comm_E + Receive_N, StartP_{i,j-1} + W_{i,j-1} + Send_E + Total\_Comm_S)$	(4)
$T_{diagfill} = StartP_{1,m}$	(5)
$T_{fullfill} = StartP_{n,m}$	(6)
$T_{stack} = (Receive_W + Receive_N + W + Send_E + Send_S + W_{pre})N_z/H_{tile} - W_{pre}$	(7)
$Time\ per\ iteration = n_{diag}T_{diagfill} + n_{full}T_{fullfill} + n_{sweeps}T_{stack} + T_{nonwavefront}$	(8)

tation costs, and the communication costs for an MPI send, receive and end-to-end message transfer are the only machine dependent parameters in the model.

Parameter H_{tile} represents the number of tiles computed per sweep step. In the general wavefront algorithm in Listing 1, the number of tiles solved per sweep step is one. Nevertheless, the reusable model accounts for the case when multiple tiles are computed per sweep step. The parameter n_{sweeps} captures the number of sweeps that propagate through the 3D grid of data. For Chimaera this is set to 8, as a distinct sweep originates at each of the 8 corners of the 3D data grid. n_{full} and n_{diag} allow the model to capture the critical path time of the multiple sweeps, and include the possibility of the sweeps overlapping to improve overall efficiency. For instance, in Chimaera sweeps 1 and 2 start at the top-right processor, while sweeps 3 and 5 start at the bottom-right processor (see Figure 2). However, as soon as the final wavefront of sweep 2 is finished on the bottom-right processor, that processor begins processing the first tile for sweep 3. Thus there is an overlap of the final stages of sweep 2 and the beginning of sweep 3 (until sweep 2 ends at the bottom-left processor). The values n_{full} and n_{diag} are therefore dependent on the ordering of the sweeps. $T_{nonwavefront}$ accounts for any non-wavefront blocks of code that occur during an iteration of the algorithm. In the case of Chimaera an MPI allreduce operation occurs per iteration, as well as several other negligible operations which we have denoted by δ_h. Finally, the message size will depend on the application being modelled. The model parameters in Table 1 are then used to form the reusable analytic model given in Table 2.

The terms *Send*, *Receive* and $Total\_Comm$ account for the MPI send, receive and end-to-end costs for a given HPC machine respectively. Equations (1) and (2) account for the computation time per sweep step, while $StartP_{i,j}$ (used in (3), (4), (5) and (6)) models the elapsed time before the computation of the first tile for sweeps beginning on a given processor (i, j). This time is often termed the "pipeline fill", and describes the time during which processor (i, j) remains idle before the first wavefront of a sweep arrives. Equation (7) models the time spent by a processor to complete all of its tiles for a given sweep. Finally, (8) acts as a top level expression that captures the time per iteration of the wavefront algorithm, collating the parameter values and the associated submodels.

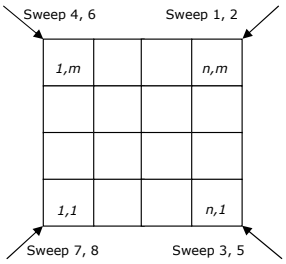


Figure 2: Chimaera pipelined wavefront operation on the 2D processor array

3 Optimisations

The utility of the reusable LogGP-based application performance model for speculative assessments (as well as its high predictive accuracy) has been well documented [2]. Here we present three candidate optimisations and evaluate their quantitative and qualitative performance behaviours. In contrast to previous work our aim is to consider algorithmic-level variations for this class of application. The suitability of the optimisations for a particular application area will additionally be governed by the underlying numeric and scientific conditions under which the wavefront algorithm operates. Nevertheless, an investigation such as this has significant benefits for application developers in assessing the most effective wavefront design given their domain science requirements. Moreover, an evaluation through predictive models is far less costly than exploring these algorithmic changes through code rewrites; our experience is that this approach also allows code developers to quickly assess the best *combination* of optimisations, particularly in the light of future scientific requirements, and that this is often extremely difficult and time consuming to quantify through code re-writes alone. The optimisations identified are applicable to wavefont algorithms in general. However

we verify their applicability by conducting our performance analysis and evaluations using AWE's Chimaera particle transport benchmark. The target architecture for this study is a 12TFLOP/s Intel Xeon/InfiniBand cluster based at the Warwick Centre for Scientific Computing (described throughout as CSC-Francesca).

3.1 Parallelising Computation with OpenMP

The first optimisation that we consider is the introduction of OpenMP [5] pragmas to automate the parallelation of the main compute block. Previous studies that have investigated the performance impact of MPI, OpenMP and hybrid applications [6, 7], have found that the performance of certain applications, particularly those with fine-grained computation, can be improved for small SMP-clusters. In this optimisation the idea is to use thread-level parallelism inside nodes and message passing between nodes. The hypothesis is that using intra-node messaging via threads and inter-node messaging via MPI, will result in fewer but larger MPI messages and will therefore make better use of the available bandwidth. Additionally this allows equivalent computation times to that of having one MPI process for each thread. As the core-density of modern processors continues to grow it is our expectation that the intra-node contention to gain network access will result in significant mesage delays. The use of OpenMP may therefore provide a mechanism for reducing this contention without requiring significant code changes.

$$W = T_{startup} + W_{original}/N_{omp} + T_{sync} \qquad (9)$$

(9) presents a proposed adjustment to the equation for W. This allows us to model the introduction of an OpenMP parallelisation strategy in which the original W, which we term '$W_{original}$' in the adapted model to avoid confusion, can be parallelised evenly over a set of threads N_{omp}. The extra terms $T_{startup}$ and T_{sync} are added to allow more accurate performance models to be produced where the cost associated with the startup and tear-down of OpenMP threads is known.

The results obtained from this model, for the AWE Chimaera benchmark executing on a hypothetically enlarged version of the CSC-Francesca machine, are shown in Figure 3. This shows a base-line improvement over the assignment of one MPI process per processor core. Against this we compare 4-way, 16-way and 64-way thread-level parallelism, which one might use if there are 4, 16 or 64 cores per node. Note that these results use a startup and synchronisation cost of 0 to evaluate the maximum possible speed improvement from the use of OpenMP-based parallelisation. Using this graph we can compare 1024 processor cores communicating through MPI (the base line) with three alternatives: 256 processes communicating through MPI, each with 4-way OpenMP threading (diamonds); 64 processes communicating through MPI, each with 16-way OpenMP threading (circles); 16 processes communicating through MPI, each with 64-way OpenMP threading (triangles). The maximum performance gain is 15-20%, however, this does not take into account thread overhead and synchronisation costs. The graph also shows that when the data grid assigned to a node is split unevenly across the threads, the load imbalance will stall execution and significantly impact the execution time. In this case the time to compute a tile is determined by the slowest thread. We see this most acutely at 4096 processor cores, where the 4-way OpenMP deployment takes a considerable performance hit.

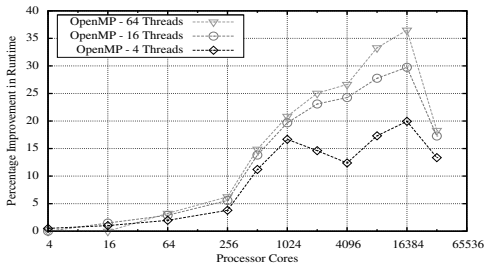


Figure 3: Performance improvement of using OpenMP threading on the main compute block

In (10) we present an adapted version of the OpenMP model which relates specifically to the mathematical problem solved by particle transport codes.

$$W = T_{startup} + W_{original}/\Omega * \lceil \Omega/N_{omp} \rceil + T_{sync} \qquad (10)$$

In these codes the Boltzmann transport equation is solved with respect to a number of angles, Ω [8, 9]. If the mathematics of the problem can be relaxed such that these angles can be solved independently (as it is the case with Chimaera) then the use of OpenMP might be best applied by dividing the work, W, into $\lceil \Omega/N_{omp} \rceil$ blocks of computation. This effectively breaks the work into chunks of processing which relate to a single angle, if there are more angles than threads, or the number of threads does not divide equally, then some processors will remain idle.

Figure 4 presents the application of this model to the AWE Chimaera benchmark assuming a calculation over ten angles. Again we see that a balanced workload (where the thread-to-angle ratio is even - e.g. OpenMP 5 and 10 threads) results in small performance gains; if the division of threads-to-angles results in an uneven split in work, then the performance is worse than the original run time.

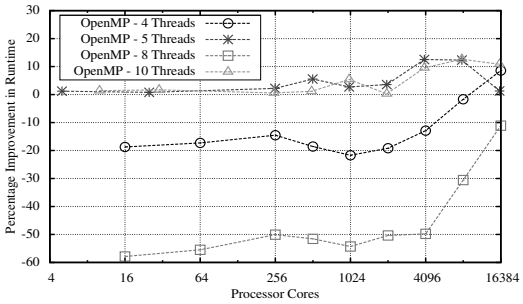


Figure 4: Performance improvement of using OpenMP threading of the main compute block using the angle division method (negative values indicate worse performance)

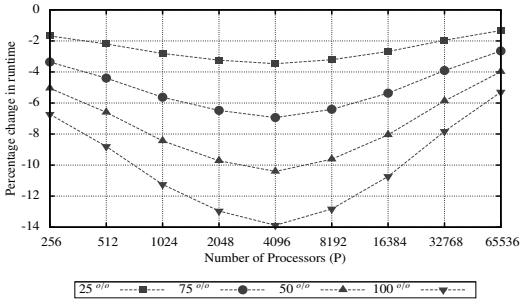


Figure 5: Shifting computations to pre-compute - strong scaling (Chimaera 240x240x240 total problem size, 1 time step, 16 energy groups, 419 iterations)

3.2 Restructuring/Shifting Computation

In the second proposed optimisation we investigate a candidate design change aimed at restructuring the computation blocks in the general wavefront algorithm. The basic wavefront algorithm in Listing 1 illustrates two computation blocks - a main computation and a pre-computation. The analytic model accounts for these by modelling them in W_{pre} and W respectively. Here we investigate the quantitative and qualitative effects of completing part of the main computation during the pre-computation block. More specifically, in an application such as AWE Chimaera where there is no pre-computation, we speculate as to the impact of completing parts of the main computation in an artificial pre-compute block. Of course, the amount that can be shifted to the pre-compute block depends on the underlying mathematics, but in this optimisation we are interested in the speculative case where there are no such limitations.

The pre-computation occurs before receives are posted and therefore does not require any boundary values from near neighbour processors. This in turn means that all processors can compute the pre-computation simultaneously. From the model we see that there should be opportunity to complete the

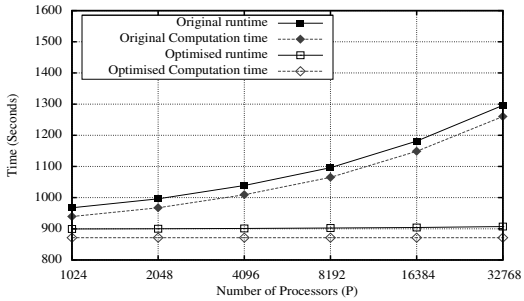


Figure 6: 100% pre-compute shift - weak scaling (Chimaera 32x32x500 cells/processor, 1 time step, 16 energy groups, 419 iterations)

pipeline fill times, given by (3), faster if more computation is done before the MPI receives. Thus we investigate how much saving (if any) can be gained by re-structuring the computation in the pipelined wavefront code. Predictions for this optimisation can be easily obtained without any extensions to the reusable LogGP analytic model. If we consider the computation per sweep step (W) in a Chimaera-type wavefront application, then we simply multiply the W in (4) and (7) by a factor of α, where $0 \leq \alpha \leq 1$, while replacing W_{pre} in (3) and (7) by $(1-\alpha)W$ to obtain the predictions. For example, if $\alpha = 0.25, 0.75$ and 1, then this represents a 25, 75 and 100 percent shift of computation on to the pre-computation block.

Figure 5 and Figure 6 present the model predictions when this optimisation is applied to a Chimaera-type wavefront code. In Figure 6 a strong-scaling study is presented for the cases where a 25, 50, 75 and 100 percent of the computation is shifted to the pre-computation block. It should be noted that the case of shifting 100% of the computation simply serves as boundary case; it will not be achievable in reality. The total number of cells solved is 240^3. In *strong scaling* the number of cells computed by a processor decreases as the number of processors increases. This has the effect of decreasing the benefits of the optimisation as predicted by (4). However, as the length of the pipeline (given by the number of $x-y$ diagonals) increases, the time spent in the pipeline fill increases with the number of processors. In the 100%-shift case, the impact of these opposing constraints results in a maximum run time reduction of approximately 14% on 4K processors. Figure 6 details a *weak scaling* study that highlights the gains in run time obtained through reducing the computation time during a pipeline fill. Thus the contribution of this optimisation clearly increases as the number of processors increases. In this case the length of the pipeline increases with the number of processors, while the computation time per sweep step remains constant.

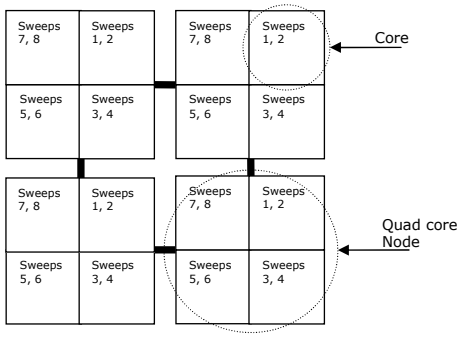


Figure 7: Sweep assignment for executing multiple simultaneous sweeps on quad-core nodes

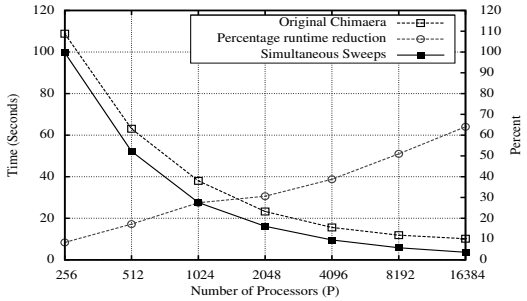


Figure 8: Simultaneous sweeps on separate cores (speculative Chimaera-type application, 240x240x240 Cells, 1 time step, 16 energy groups, 419 iterations)

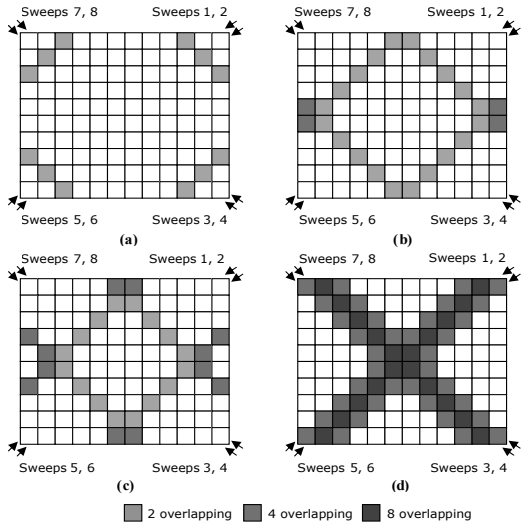


Figure 9: Simultaneous multiple wavefronts overlapping steps

3.3 Multiple Simultaneous Sweeps

The reusable LogGP-based analytic performance model specified the ordering of sweeps using the parameters n_{full} and n_{diag}. AWE Chimaera, for example, performs eight sweeps with two sweeps overlapping up to a maximum of half of the pipeline fill time. In this third optimisation we investigate the case where the multiple sweeps executed are performed simultaneously, that is, the case when the wavefront sweeps are fully overlapping. This requires beginning all 8 sweeps at the same time at their respective corners of the 3D data grid, and each sweeping across the grid to the opposite corner. When executed simultaneously, the eight sweeps are expected to only have a single pipeline fill time. We investigate this optimisation in two forms for a typical modern HPC system consisting of CMP nodes. First we consider the case where each sweep is computed using separate processor cores. Second we explore the case when all cores compute all the sweeps.

3.3.1 Simultaneous Sweeps on Separate Cores

Modern HPC systems consist of nodes with multiple processing units. Nodes with multi-core processors are indicative of such systems. Thus the availability of high processor counts may enable us to assign separate sweeps to separate processing elements on a node. Thus the purpose of this evaluation is to speculate on the performance behaviour of such an assignment.

Consider the case where a total problem size of 240^3 is solved for a Chimaera-type application with the eight sweeps computing simultaneously, beginning from the four corners of the 2D processor array. Assuming that a node consists of a single quad-core processor, we assign two sweeps to be computed by a single core on each node; Figure 7 depicts such a distribution of work. In this case the time to solution can be predicted simply by considering the time to compute a single sweep from one corner of the processor array to the opposite corner. However, as two sweeps are computed simultaneously per core, we model the computation time per sweep step to be double the computation time (W) for a single sweep step. Similarly the message size will also double to account for larger messages being sent per sweep step. We compare the predictions of this optimisation with the original Chimaera runtime predictions in Figure 8.

The results show that the benefits of performing multiple wavefronts increases for large processor counts. This is directly attributable to the saving from the pipeline fill time. It should be noted that any contention due to multiple wavefronts crossing node boundaries has been ignored in the model. In a system such as the Cray XT4 used in [2], we explored communication contention on the node due to the sharing of communication resources such as the single bus to RAM and the NIC. A concrete application that uses multiple simultaneous sweeps will be needed to explore such issues; nevertheless, we believe that the results presented here demonstrate the maximum possible benefits of this optimisation.

Table 3: Modelling multiple simultaneous sweeps - Extentions to the reusable analytical wavefront model

$$StartP_{i,j} = max(StartP_{i-1,j} + \eta(W_{i-1,j} + \frac{1}{2}(Total\_comm_E + Receive_N)),$$
$$StartP_{i,j-1} + \eta(W_{i,j-1} + \frac{1}{2}(Send_E + Total\_Comm_S))) \quad (11)$$

$$T_{stack} = \eta(\frac{1}{2}(Receive_W + Receive_N) + W + \frac{1}{2}(Send_E + Send_S) + W_{pre})\,(N_z/H_{tile} - (m+n-1)) - W_{pre} \quad (12)$$

$$Time\ per\ iteration = 2n_{full}T_{fullfill} + n_{sweeps}T_{stack} + T_{nonwavefront} \quad (13)$$

3.3.2 Simultaneous Sweeps on All Cores

In the second case all 8 sweeps are processed simultaneously by all the processor cores, beginning at each corner of the 2D processor grid. During this operation a processor may compute a minimum of 2 sweeps up to a maximum of 8 sweeps. This is illustrated in Figure 9, where based on the progression of the first wavefront of each sweep there are (a) 2 sweeps, (b) and (c) 4 sweeps, or (d) 8 sweeps being computed by some processor (i, j).

If we assume that the time to compute a block of cells of height H_{tile} for a single sweep on a single processor is W, and the time to compute η simultaneous sweeps on a single processor is ηW, then the critical path time to complete all 8 sweeps depends on the step during which each sweep starts to overlap with any other sweep. We assume that the communication primitives used are non-blocking MPI sends. Therefore, the extensions required to model multiple sweeps will at least require finding the steps during which η varies from 2 to 4 to 8. These extentions are detailed in Table 3.

At the beginning of a sweep each processor will be computing 2 sweeps (one originating from the top of the 3D cube and one from the bottom). Assuming $n > m$, after $m/2$ steps we find that sweeps 1, 2 and 3, 4 as well as sweeps 5, 6 and 7, 8 overlap. If $m > n$, then sweeps 1, 2 and 5, 6 as well as 3, 4 and 7, 8 overlap. In both cases this will amount to a maximum of four sweep overlaps. Finally, as seen in in Figure 9 (c), after $(n+m)/2$ steps have occurred, all sweeps overlap. As the overlapping occurs during the initial pipeline fill stages of the wavefront operation, we increase the computation time per sweep step from $2W$ to $4W$ and to $8W$ after $m/2$ and $(n+m)/2$ steps respectively, as in (11). Thus the value of η during the operation of the wavefronts changes as follows:

$$\eta = \begin{cases} 2, & \text{default} \\ 4, & \text{if } m/2 < i+j \text{ or } n/2 < i+j \\ 8, & \text{if } i+j \geq (m+n)/2 \end{cases}$$

As there are 2 sweeps originating from one corner of the 2D processor array, we again assume that the communication operations pack boundary values for both these sweeps into one MPI message with double the message length. Thus the total number of distinct sweeps originating from the corners of the 2D processor array is reduced to 4; however, since they operate simultaneously, the critical path time includes only the time to process one stack of tiles (i.e. $n_{sweeps} = 1$). After the maximum overlapping is achieved, wavefronts complete each step with $\eta = 8$ in a steady state operation until the pipeline empty stages. During the pipeline empty stages the reverse of a pipeline fill occurs, where the overlapping number of sweeps per processor reduces from 8 to 4 and finally to 2.

The time for a pipeline empty is equivalent to the time for a pipeline fill. We use the multiplier 2 in the first term of (13) to account for this. The number of steps for a pipeline empty $(m + n - 1)$ is subtracted in (12). The communication terms in the above equations are also multiplied by η, to account for the worst case (sequential) communication when multiple wavefronts are processed. Although we assume that MPI non-blocking primitives are used, we believe that contention on the NICs may have the effect of sequentialising messages. If the number of tiles in the z dimension is less than the number of pipeline fill stages given by $N_z/H_{tile}-(m+n-1)$, the multiplier reduces to 0. This is due to the fact that in such a setup the maximum overlapping of all sweeps only occurs briefly during the pipeline fill stages; this cost is included in (11). It is also important to note that such a configuration is not efficient as many processors will remain idle during the pipeline fill and empty stages.

Previous related work on models for multiple wavefront sweeps have been published in [10] and [11]. The former uses the maximum number of multiple wavefronts (i.e. 8) crossing the boundaries of a processor and neglects the dynamic overlapping during pipeline fill and empty. The latter mentions the use of multiple simultaneous sweeps in the solution of an unstructured grid particle transport application without specifics of the model extensions. Thus we believe the model detailed in this section provides a precise and yet intuitive model for the prediction of multiple simultaneous sweeps. Moreover, the extension also shows the re-usability of the LogGP-based application per-

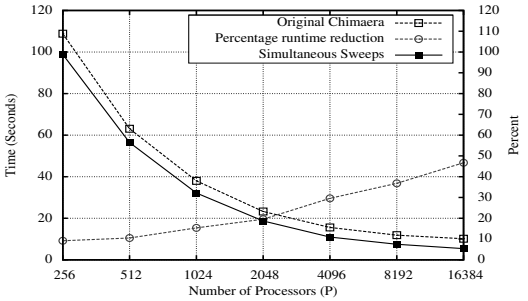


Figure 10: Simultaneous sweeps on all cores (Speculative Chimaera-type application, 240x240x240 Cells, 1 time step, 16 energy groups, 419 iterations)

formance model. Figure 10 details the run time predictions gained from this model. As can be seen, the model speculates an increasing performance improvement (up to 50% at 16K processors) due to the saving of multiple pipeline fill times.

Particle transport codes such as AWE Chimaera use a convergence criterion to halt the execution of the application. The convergence of the application depends very much on the order in which the sweeps operate in the 3D grid of data. Thus we anticipate that the number of iterations for convergence may be more when simultaneous multiple sweeps are executed, and this should be taken into account when assessing the benefits of this optimisation. A further consideration of multiple simultaneous sweeps would be the increased requirement for memory and memory bandwidth between processor cores and main memory. More memory will be required for holding intermediate steps of the computation performed by all 8 sweeps; note that this will be less of a problem for smaller tile sizes. High memory bandwidth is needed to continually feed the cores with data to perform the computations for all 8 sweeps per sweep step. Thus selecting a node architecture with high memory bandwidth (such as the Cray XT series) will be advantageous.

4 Conclusions

In this paper we have evaluated three potential optimisations to the class of parallel algorithms known as pipelined wavefront computations. We use a recently developed reusable LogGP-based analytic predictive performance model to evaluate: (1) computation block parallelisation using OpenMP threads; (2) computation block restructuring to reduce the cost of the pipeline fill time and, (3) a redesign of the algorithm so that multiple sweeps are performed simultaneously. Results from our analysis shows that there is considerable improvement to performance when OpenMP is used to parallelise computation blocks given careful load balancing. We also quantify the benefits that shifting computation to a pre-computation block will achieve. Finally, we demonstrate that operating simultaneous multiple sweeps provides considerable benefits if the underlying mathematical solution permits.

Each of our proposed optimisations is explored within the context of an industry-strength benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment. Through this study we aim to assess whether code optimisations explored using analytic performance models can be reasonably translated to a real industry code of this type. We demonstrate that application performance modelling can focus and lead future code development, allowing organisations to identify and code optimisations which are deemed significant enough to warrent the additional effort and cost.

References

[1] L. Lamport. The parallel execution of do loops. *Commun. ACM*, 17(2):83–93, 1974.

[2] G.R.Mudalige, M.K.Vernon, and S.A. Jarvis. A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, April, 2008.

[3] Sweep3d. The ASCI Sweep3d Benchmark. `http://www.llnl.gov/asci\_benchmarks/asci/limited/sweep3d/asci\_sweep3d.html`.

[4] Nas parallel benchmark. `http://www.nas.nasa.gov/Resources/Software/npb.html`.

[5] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.0. 2008.

[6] M. Sato, S. Satoh, K. Kusano, and Y. Tanaka. Design of OpenMP Compiler for an SMP cluster. In *EWOMP99*, 1999.

[7] L. Smith and Mark Bull. Development of Mixed Mode MPI / OpenMP Applications. In *Workshop on OpenMP Applications and Tools (WOMPAT 2000)*, 2000.

[8] K. R. Koch, R. S. Baker, and R. E. Alcouffe. Solution of the First-Order form of the 3D Discrete Ordinates Equation on a Massively Parallel Processor. *Transactions of the American Nuclear Society*, 65:198–199, 1992. Annual Meeting, Boston, MA.

[9] M.R. Dorr and C.H. Still. Concurrent Source Iteration in the Solution of Three-Dimensional Multigroup Discrete Ordinates Nutron Transport Equations. Technical Report UCRL-JC-116694 Rev 1, Lawrence Livermore National Laboatory, Livermore, CA, May 1995.

[10] M.M. Mathis, N.M. Amato, and M.L. Adams. A General Performance Model for Parallel Sweeps on Orthogonal Grids for Particle Transport Calculations. Technical report, Texas A&M University, 2000.

[11] M.M.Mathis and D.J.Kerbyson. Performance Modeling of Unstructured Mesh Particle Transport Computations. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, April 2004.