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

Gihan R. Mudalige, Stephan A. Jarvis, Daniel P. Spooner, Graham R. Nudd
High Performance Systems Group, Department of Computer Science
University of Warwick
Coventry, CV4 7AL, U.K.
{G.R.Mudalige,Stephen.Jarvis}@warwick.ac.uk

Abstract

This paper details the development and application of a model for predictive performance analysis of a pipelined synchronous wavefront application running on commodity processor cluster systems. The performance model builds on existing work [1] by including extensions for modern commodity processor architectures. These extensions, including coarser hardware benchmarking, prove to be essential in countering the effects of modern superscalar processors (e.g. multiple operation pipelines and on-the-fly optimisations), complex memory hierarchies, and the impact of applying modern optimising compilers. The process of application modelling is also extended, combining static source code analysis with run-time profiling results for increased accuracy. The model is validated on several high performance SMP systems and the results show a high predictive accuracy ($\leq$ 10% error). Additionally, the use of the performance model to speculate on the performance and scalability of this application on a hypothetical cluster with two different problem sizes is demonstrated. It is shown that such speculative techniques can be used to support system procurement, run-time verification and system maintenance and upgrading.

1 Introduction

Predictive performance analysis of High Performance Computing (HPC) applications on large-scale parallel platforms provides a valuable insight into the achievable performance of these systems [2, 3, 4]. Such techniques are particularly useful in high performance computing scenarios as these systems often require a large investment not only from the end user but also from the designers, developers and vendors. The advantages of performance modelling also extend beyond this, allowing efficient scheduling by anticipating a workloads behaviour prior to execution [5], which in turn allows efficient utilisation of resources and sustainable levels of Quality of Service (QoS) [6]. It has been recognised that performance analysis techniques can be used throughout the life-cycle of a system [5, 7]. At the design stage they can serve to quantify the advantages and disadvantages of different architectural options. When procuring systems users can use performance predictions to compare alternative vendor systems and at the implementation stage predictions based on an implemented prototype can serve as a forecast for the final system [8]. After installation, predicted results can then be used to validate whether the installation was successful, and whether the configurations were made accurately to obtain optimal system performance. Additionally, during maintenance such approaches can indicate any faults that affect the system performance and also the possible benefits that can be gained by upgrading. Developing methods and tools to aid application performance analysis in a High Performance Computing setting continues to be an important research area. It can be argued that the demand for better and more robust methods and tools will always remain due to the rapid change in hardware technologies, algorithm design and increased application complexity.

This paper details recent developments to the modelling tool PACE [7, 9] demonstrated by an enhanced model of the ASCI parallel pipelined synchronous wavefront application SWEEP3D, a recognised HPC benchmark solving a 1-group time independent discrete ordinates (S_N) three-dimensional Cartesian geometry neutron transport problem. The performance model developed here extends a previous model developed for the same application [1], extending the modelling procedure and benchmarking process so that they are better suited to recent commodity processor cluster systems. These extensions, including coarser hardware benchmarking, prove to be essential in countering the effects of modern superscalar processors (e.g. multiple opera-

tion pipelines and on-the-fly optimisations), complex memory hierarchies, and the impact of applying modern optimising compilers. The process of application modelling is also extended, combining static source code analysis with runtime profiling results for increased accuracy. Additionally this work provides detailed experimentation and validation results for the developed model on several HPC cluster systems, as well as a speculative study that illustrates a typical scenario of using a model such as this to predict the performance of a hypothetical system. In Section 2 a description of the wavefront application SWEEP3D is detailed. This application has been modelled by other researchers using a variety of techniques, attesting to the importance of both the application and also the need to understand its performance characteristics at a detailed level. This related work is discussed in Section 3. The development of a performance model based on the PACE tool set is detailed in Section 4. Model validation on several high performance cluster systems is detailed in Section 5, and Section 6 illustrates the use of the model for scalability predictions of SWEEP3D running on a hypothetical system. Finally, conclusions and future work are presented in Section 7.

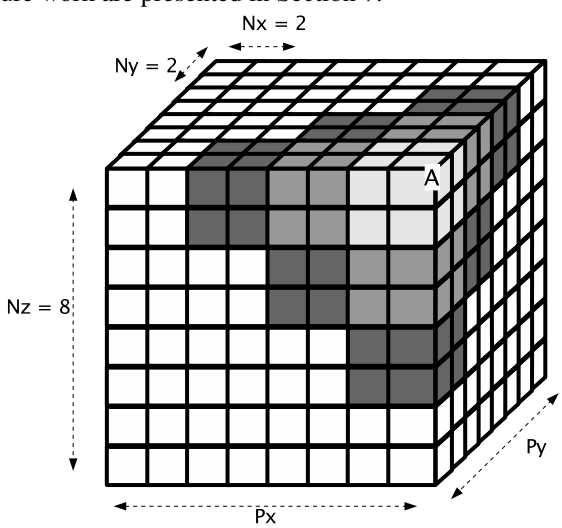

Figure 1. A Sweep originating from vertex A travels across the spatial grid to the opposite unseen vertex. The most recently processed cells are shaded darker. Here the processor count in the x and y direction is $P_x \times P_y = 4 \times 4$ and $N_x \times N_y \times N_z = 2 \times 2 \times 8$.

2 ASCI SWEEP3D

SWEEP3D is a benchmark code representing a real ASCI[1] workload application [10]. A complete discourse of the problem solved by SWEEP3D is given in [11, 12, 13, 14]. A brief description of the problem serves to highlight the main parameters, variables and the operation of the algorithm.

SWEEP3D solves a 1-group time independent discrete ordinates (S_N) three-dimensional Cartesian geometry neutron transport problem. The discrete ordinates method is a common approach to calculating solutions to neutron transport problems [11]. This involves an iterative procedure whereby the 1st order form of the transport equation (1) [3] discretizes the energy variable E, angular directions Ω and the spatial domain R [3, 15].

$$\begin{gathered}\nabla.\Omega\Psi(r,E,\Omega) + \iint \sigma(r,E)\Psi(r,E,\Omega) = \\ \iint dE'd'(r,E' \rightarrow E,\Omega.\Omega')\Psi(r,E',\Omega') + (1/4\pi) \\ \iint dE'd\Omega'\chi(r,E' \rightarrow E)\nu\sigma(r,E')\Psi(r,E',\Omega') \\ + Q(r,E,\Omega) \qquad (1)\end{gathered}$$

The unknown quantity is Ψ which represents the flux of particles at the spatial point $r \in R$ with energy $g \in E$ travelling in direction $d \in \Omega$.

The SWEEP3D code solves for only one energy group and the angular direction Ω is discretized into a set of quadrature points. This gives rise to the notation S_N where N represents the number of angular ordinates used. The solution involves a transport sweep through the entire grid space R and angle space Ω in the direction of particle travel. For a given discrete angle, each grid cell (i.e. spatial point $r \in R$) has seven unknowns. These are particle fluxes on the six cell faces and the flux within the cell centre. Boundary conditions (vacuum or reflective) allow the sweep to be initiated at the object's exterior. Thereafter, for a given cell, three incoming fluxes from three cell faces for a given discrete angle provide the solution of the cell centre flux and the solution to the three outgoing cell faces through which the particle fluxes leave the cell. A cell cannot compute its outflows before its inflows become available. Therefore a computation travels across the spatial domain as a recursion in all the three dimensions.

The only inherent parallelism is over the discrete angles where each angle is independent. However, due to the reflective boundary conditions, this is restricted to a single octant of angles. The transport sweeps are implemented as a series of pipelined sweeps through a three dimensional cube of cells or grid points. This grid of cells is mapped on to a two dimensional array of processors of size $P_x \times P_y$ where these dimensions represent the number of processors in the i and j direction respectively. Each processor is mapped to perform calculations for $N_x \times N_y \times N_z$ number of cells as in Figure 1.

To improve the parallel efficiency, blocks of work are pipelined through the processor array. SWEEP3D is coded with blockings for angles and in one direction (k-in the current implementation). The k-plane blocking parameter *MK* and angle blocking parameter *MMI* specify the size of the

[1] ASCI is a collaboration among the national labs in the U.S. for advance defence oriented computer modelling and simulation.

number of k-dimension cells and number of angles that are solved before boundary data is forwarded to the next processor in the pipeline. Thus, a sweep can be conceptualised as originating from one vertex in the 3D cube to its opposite vertex (Figure 1). There are eight octants of angles; each octant corresponding to one of the eight corners (vertices) of the spatial domain. Therefore eight distinct sweep directions can be seen in three-dimensions.

SWEEP3D uses message passing to communicate the boundary fluxes from one processor to the next downstream processors. In addition, through a specific ordering of octants an upper and lower octant pair also gets pipelined. Finally the sweeps of the next octant pair starts before the previous wavefront is completed but this is limited to two octant pairs at a time due to reflective boundary conditions [10, 15]. One iteration of the code consists of sweeps in all 8 octants. Twelve such iterations are performed in the implemented SWEEP3D code and the time to complete this process is recorded. The current SWEEP3D code has implementations for both MPI and PVM message passing interfaces.

3 Related Work

The ASCI SWEEP3D application has been modelled using a range of HPC application prediction methodologies. This includes use of the LogGP methods, documented in [16], and also by researchers at the Los Alamos National Laboratory, see [2, 3]. Additional modelling of this application has been done using the POEMS system [17]. Both the LogGP model and the Los Alamos models can be loosely described as non-automated analytical models, due to the largely manual model development procedure. POEMS however provides tools to automate model development, and thus can be described as a semi-automated approach. The PACE model described in this paper is also generated semi-automatically. Automated approaches have numerous advantages, they are time-saving and also require less knowledge of the application. However they normally require manual tuning if they are to achieve comparable results to the hand-crafted approaches.

Both the LogGP model and the Los Alamos model for SWEEP3D represent analytical methods. In fact the Los Alamos model uses a specialised approach based on the same LogGP parameters. Both models require an application expert to manually extract the operation of the application and describe these operations using a set of parameters to form a mathematical formulae capturing the run-time of the application. The parameters in LogGP abstractly define the computing bandwidth, communication bandwidth, communication delay and the overlap of communication and computation. The focus of the LogGP model [16] is to capture the communication pattern of the application by both a detailed parallel operation model as well as a system specific communication resource usage model. For instance the model developed in [16] specifically characterises the MPI communication implementation on an IBM SP/2.

The Los Alamos model [2, 3] uses a similar set of parameters to describe the execution time of the application. However, in this case it is expressed in terms of the total computation time, total communication time and the overlap of computation and communication times (2).

$$T_{total} = T_{computation} + T_{communication} - T_{overlap} \quad (2)$$

The total time is modelled for each term independently. This is somewhat different from the LogGP model where the latter extracts the model by modelling and interleaving the computation time and the communication time at each step of the application.

By contrast the semi-automated Performance Oriented End-to-end Modelling System (POEMS) [17] is an environment for the performance modelling of parallel and distributed systems in which all levels of the system are considered, including application software, run-time and operating system software and underlying hardware systems and architectures. In addition to this complete coverage of *multi-domains*, POEMS also consists of *multi-paradigm* modelling which allows the use of multiple evaluation paradigms – analysis (by incorporating analytical models such as LogP and LoPC), simulation (by means of supporting simulators) or by the direct measurement of software or hardware systems – in a single system model. The POEMS environment also consists of a repository of performance data gathered during previous evaluation, modelling and measurement processes. These are used as historical data to estimate the performance of widely used algorithms as functions of system/architectural characteristics and configurations. The SWEEP3D model developed using POEMS [17] is analysed using these multi-paradigms. This includes static and dynamic task graphs, the use of the LogGP model developed in [16] and simulation using MPI-Sim [18] and SimpleScalar [19]. Each of these models are then integrated to generate a total system model. The main benefit of this multi-paradigm approach is the ability to validate each model against each other for high levels of predictive accuracy.

The POEMS system is a complementary approach to the PACE tool set. However, in reality the multi-paradigm approach of POEMS means that performance experts who are proficient in a multitude of analysis techniques are required to make best use of it. For instance, the developer will need to be familiar with the LogGP analytical methods, as well as having experience in using the simulator tools. PACE on the other hand can be seen as more intuitive for model development by non-performance experts, which at the same

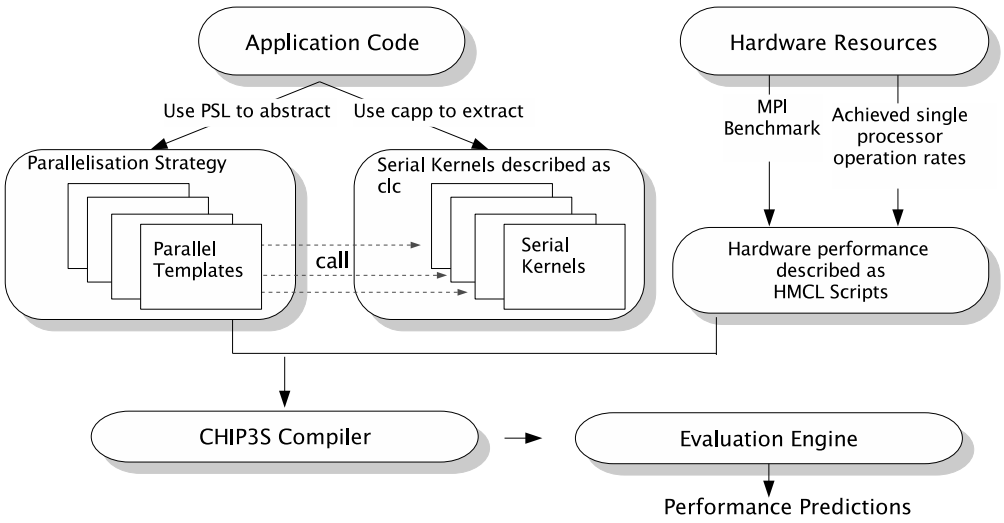

Figure 2. Outline of the PACE tool set used for modelling and performance prediction. Application code characterisation is supported by the PSL to form parallel templates and *capp* is used to extract C language characterisations. The application model can then be combined with the desired hardware resource model to obtain performance predictions.

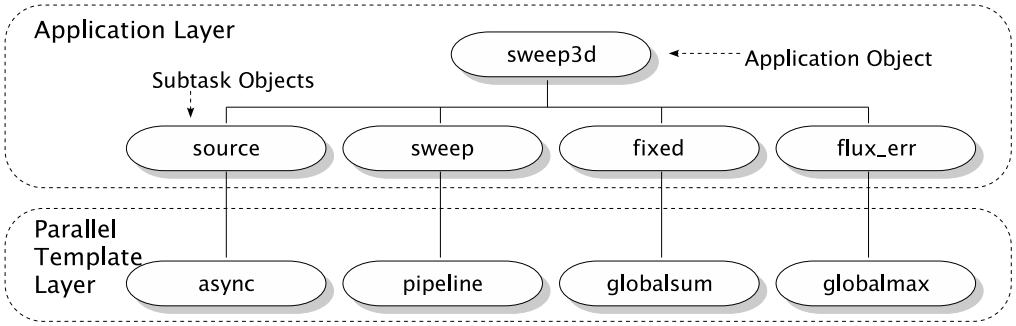

Figure 3. SWEEP3D PACE model Object Hierarchy.

time enables the rapid generation of results with acceptable levels of predictive accuracy.

4 PACE Performance Model

PACE (Performance Analysis and Characterisation Environment) [7] is a layered performance characterisation method encompassing all aspects of a system including a software execution graph, the parallelisation strategy and the system's resources and architecture. PACE is largely based on independent application and resource modelling, an overview of which can be found in Figure 2. PACE consists of a static source code analyser called '*capp*', which extracts the control flow of the application and the frequency of performance-critical operations (opcodes). *Capp* is used to extract the operation of a serial kernel in terms of C language micro-characterisations (*clc*s). The core of the PACE system is a Performance Specification Language (PSL) named CHIP^{3}S (Characterisation Instrumentation for Performance Prediction of Parallel Systems) and a related compiler. The PSL provides a description of the application and its parallelisation in an intuitive language syntax. The resource modelling is supported by a Hardware Modelling and Configuration Language (HMCL), which provides a description of the computation and communication resource performance of a system. HMCL scripts consist of hardware resource performance values obtained by processor and MPI benchmarks. Once both the application and resource models are created, they can be combined as inputs to the PACE evaluation engine to obtain predictions of execution time within seconds. An important aspect of this process is the ability to reuse the models with different resource or application models.

The PACE model of SWEEP3D developed here builds on an original model detailed in [1]. The extended model accounts for a number of architectural modifications found in modern SMP systems. It was identified that static source code analysis combined with the original PACE hardware benchmarks under estimated the effect of several important optimisations performed on modern hardware and in modern optimising compilers. These optimisations include the

myriad array of superscalar features (such as multiple operation pipelines, on-the-fly optimisations), compiler optimisations (such as instruction scheduling, branch prediction) and memory hierarchy affects. The work in [1] relies on a set of opcode benchmarks, which when combined with the tally of opcodes produced by the *capp* source code analyser, allow summative results to be calculated; these results then form the basis for performance predictions. This method was acceptable for processors available at the time. Producing accurate predictions based on this very fine-grained benchmarking relied on the processor executing the code exactly as it was (or with very little modification) when the *capp* tool did the static source code analysis. This assumption does not hold for modern processor systems and compilers where it under estimates run-time hardware/compiler performance optimisations when the application is actually executed. Predictions based on this approach in some cases (such as on the AMD Opteron 2-way SMP cluster) gave a prediction error as large as 50%. An alternative approach, and the one that is adopted in this research, is to use profiling to obtain a coarser level measurement of the achieved performance of the the serial source code of the application. The version of the SWEEP3D code used for model development, and later for validation, is a C language version of the application from the Los Alamos National Laboratory. The object hierarchy diagram for the SWEEP3D model is shown in Figure 3. The starting point of model development is the application object *sweep3d* which is modularised into four subtask objects. The application object calls each subtask object in turn for 12 iterations. Of the four subtask objects *sweep* implements the core sweeping functionality and is responsible for 97% of the computation. The other three subtasks are less significant, but contribute to the control flow of the execution. Each subtask object (of the performance model) is evaluated using a separate parallelisation strategy in the parallel template layer. The most important parallel template object is *pipeline* which describes the computation/communication structure of the sweep function.

4.1 Application-Layer Model

The application-layer model consists of one application object and the four subtask objects. Part of the *sweep3d* application object's PSL description is detailed in Figure 4. The initial declarations consist of *include* statements, *var* external variable declarations, *link* statements and *options* statements. A complete description of the meaning and application of these statements are given in [7, 9]. The *include* statements declare other objects that are referenced by this object, the *var* declares externally (by user at evaluation time) modifiable variables, the *link* statements enable variables in other referenced objects to be modified by this object and the *options* statements define a set of default values. The *proc exec* statements declare a subroutine or a function. The evaluation of the model starts from the procedure *init* which calls the evaluation of the four subtask objects one after the other for 12 iterations (lines 196 - 200) (depending on the *epsi* convergence variable defined in the input file that details the problem size). It can be seen that procedures directly implement the control flow of the application. Thus, evaluation of the model means that these statements are directly executed (in a similar fashion to a set of C code statements). By coding the control flow of the

```
1   application sweep3d {
2
3   include hardware;
4   include source;
5   include sweep;
6   include fixed;
7   include flux_err;
8
9   var numeric: npe_i = 1,
    ...............
50  link {
51  hardware: Nproc = npe_i * npe_j;
52  sweep: it = it,
    ...............
86  }
87  option {hrduse = "IntelP31266"; }
88  proc exec min
89          var x, y;
90          {       if (x > y) return y;
92                  else return x;
93          }
            ...............
112 proc exec init {
113         var numeric: i, tmp;
114         if (isct == 0) nm=1;
115         else if (isct == 1) nm=4;
116
117         it = it_g / npe_i ;
118         jt = jt_g / npe_j + 1 ;
119         if( mk > kt ) mk = kt;
            ..................
189         for (i=1;i<=tmp;i=i+1){
190            for(mi=1;mi<=mmi;mi=mi+1){
191                ndiag = ndiag+max
                   (min(i,min(jt,
                   min(nk, jt+nk-i))),0);
192            }
193         }
194         (*get average of ndiag*)
195         ndiag = ndiag/tmp;
196         for( i=1;i<=-epsi;i=i+1){
197                call source;
198                call sweep;
199                call fixed;
200                call flux_err;
201         }
202     }
203 }
```

Figure 4. Application Object - *sweep3d*: The entry to the performance model. This initialises the default values of the model and then in turn calls each subtask object.

```
1   subtask sweep {
2   include hardware;
3   include pipeline;
4
5   var numeric:
            ...............
30  link {
31  pipeline:
32     Tx_sweep_init = sweep_init(),
33     Tx_octant = octant(),
       ...............
40     Tx_work = work(),
       ...............
52  }
54  proc exec init {
       ...............
67  }
    ...............
168 proc cflow work {
169 loop(<is clc,LFOR>,jt+nk-1+mmi-1){
172   loop (<is clc, LFOR>, ndiag){
174     compute <is clc,2*MFDG>;
176       loop (<is clc,LFOR>,nm-1){
178         loop (<is clc,LFOR>,it){
179           compute <is clc,2*MFDG>;
180         }
182       }
183       case (<is clc,IFBR>){
184         (-ifixups)/(-epsi):
185         loop(<is clc,LFOR>,it{
186            compute <is clc,19*MFDG>;
187         }
188         1-((-ifixups)/(-epsi)):
190         loop (<is clc,LFOR>,it){
191            compute <is clc,19*MFDG>;
193         }
             ...............
             ...............
205        case (<is clc,IFBR>){
206        do_dsa:
           loop (<is clc,LFOR>,it){
208            compute <is clc,6*MFDG>;
209                    }
210                }
212            }
213        }
214    } (* End of work *)
    ...............
314 }
```

Figure 5. Subtask Object - *sweep*: This characterises the core serial operations of the sweep function. The *pipeline* parallel template, noted by the *link* statement, calls these serial functions during evaluation.

application, run-time values that decide loop iterations that cannot be determined before run-time are automatically calculated. Such variables include the *ndiag* value (calculated dynamically in lines 189 - 193), which directly determines the per cell work modelled in the *sweep* subtask object. In order to establish the complex relationship that determine the value of *ndiag*, we have used the average value resulting from the actual C code implementation of the application.

The structure of the models subtask objects are similar to the application objects, but can additionally contain C language characterisation (*clc*) descriptions. Part of the *sweep* subtask object can be found in Figure 5. The *clc* descriptions represent the core computation units of the application. The source code analysis parser *capp* provides an automated procedure to obtain these descriptions. For reasons described in subsection 3.3 the *clc* descriptions are described solely in floating-point operations. The abbreviations LFOR and IFBR represent the time cost for a loop start-up and conditional branch check respectively. As explained in subsection 4.3, these costs are considered negligible when calculating the total time of the application. The mnemonic MFDG represents a floating point operation. Unlike control flow statements, the *clc* instructions are not executed, but are accumulated depending on the number of loop counts and branch probabilities to give a time for each serial computation described by the *clc*. In the *sweep* function, work within a cell contains some non-structural *goto* statements. In this case a reasonable estimate of the average work related to these statements is manually coded into the *clc*. A subtask object includes the parallel template used when it is evaluated. In the case of *sweep* in Figure 5 the related parallel template is *pipeline* (line 3). The branches are assigned a probability score and loops are given an average iteration count that can be calculated from profiles of the execution of the application and data analysis. The procedure *work* outlined in Figure 5 represents the bulk of the computation. The point at which it is evaluated from within the *pipeline* parallel template object can be seen at line 260 in Figure 6.

4.2 Parallelisation Strategy Model

The pipelined wavefront structure of SWEEP3D is modelled in the parallel template layer. The core template implementing this is the *pipeline* parallel template object. Part of the *pipeline* parallel template object is illustrated in Figure 6. The structure of this template has been derived directly from the *sweep* function found in the application; it should be noted therefore, that a level of understanding of the application is required to extract the parallel decomposition. Nevertheless, due to the intuitive syntax of the PSL scripts, this process is straightforward for an engineer with some understanding of the application. The communication resource usage and the communication pattern is also described in this layer. The *init* procedure describes the start of the *per octant, per angle block, per k-plane block* loop in lines 194 to 211. For each iteration of this loop, MPI receives are posted (line 222) followed by the per processor work (line 260). Next the MPI sends are executed by sending outbound cell face values (line 269). In addition to the *init* procedure, *pipeline* defines and makes use of sev-

```
1     #include <mpidefs.h>
2     partmp pipeline {
      ..............
168   proc exec init {
169   var numeric: phase,
      ..............
      ..............
194   for( phase = 1; phase <= 8; phase = phase + 1)
195   {      ..............
203           for( i = 1; i <= mmo; i = i + 1 )
204           {     ..............
210                   for( j = 1; j <= kb; j = j + 1 )
211                   {    ..............
216                           for( x = 1; x <= npe_i; x = x + 1 )
217                           for( y = 1; y <= npe_j; y = y + 1 )
218                           {   ..............
220                               ew_rcv = Get_ew_rcv( phase, x, y );
221                               if( ew_rcv != 0 )
222                               { step mpirecv { confdev ew_rcv, myid, nib*8; } }
223                               else { step cpu on myid { confdev Tx_else_ew_rcv; } }
224                           }
                              ..............
238                           for( x = 1; x <= npe_i; x = x + 1 )
239                           for( y = 1; y <= npe_j; y = y + 1 )
240                           {   ..............
243                               ns_rcv = Get_ns_rcv( phase, x, y );
                                  ..............
257                           }
258
259                           step cpu {
260                               confdev Tx_work;
261                           }
262
263                           for( x = 1; x <= npe_i; x = x + 1 )
264                           for( y = 1; y <= npe_j; y = y + 1 )
265                           {   ..............
267                               ew_snd = Get_ew_snd( phase, x, y );
268                               if( ew_snd != 0 )
269                               { step mpisend { confdev myid, ew_snd, nib*8; } }
270                               else { step cpu on myid { confdev Tx_else_ew_snd; } }
271                           }
272
273                           for( x = 1; x <= npe_i; x = x + 1 )
274                           for( y = 1; y <= npe_j; y = y + 1 )
275                           {   ..............
277                               ns_snd = Get_ns_snd( phase, x, y );
                                  ..............
300                           }
301                   }
                      ..............
308           }
309       }
310   }
311}
```

Figure 6. Parallel Template Object - *pipeline*: Characterises the parallel synchronous sweeper operation.

eral procedures such as *Get_ew_rcv* (line 220). The serial kernel code characterised in the related subtask object are called from the parallel template (e.g. *Tx_work* at line 260, *Tx_else_ew_rcv* at line 223). The remaining parallel template objects describe different parallel communication patterns. *globalsum* and *globalmax* implement collective communications for reduction operations. The *async* object implements a sequential template and the subtask object that uses this has no communications.

```
config  IntelP31266
{
  hardware {
      Tclk = 1 / 1266,
      Desc = "Intel P3 1266MHz,2GB",
      Desc = "PC, Intel P3 1266Mhz,
              2GB RAM,
              Linux 2.4.21-37.0.1.ELsmp",
      Source = "SCS IBM cluster";
  }
  clc {
      (* 50x50x50 problem
      1 / achieved flop rate*)
      (*Floating-point addition*)
      AFDG =  0.009090,
      (*Floating-point multiplication*)
      MFDG =  0.009090;
  }
  mpi {
      DD_COMM_A = 1024,
      DD_COMM_B = 10.7866,
      DD_COMM_C = 0.0158239,
      DD_COMM_D = 41.7131,
      DD_COMM_E = 0.00616761,
      DD_TSEND_A = 1024,
      DD_TSEND_B = 0.665026,
      DD_TSEND_C = 0.000726049,
      DD_TSEND_D = -49.4555,
      DD_TSEND_E = 0.0087964,
      DD_TRECV_A = 1024,
      DD_TRECV_B = 3.00234,
      DD_TRECV_C = 0.0014768,
      DD_TRECV_D = -43.1711,
      DD_TRECV_E = 0.0088473;
  }
}
```

Figure 7. Hardware Resource model for Pentium 3 2-Way SMP cluster with Myrinet Interconnect.

4.3 Hardware Layer - Serial Kernel Benchmarking

SWEEP3D is a compute-intensive application and as such the main contributing operation type to the computation time comes from double precision floating-point operations. The processor resource usage was therefore characterised by describing each subtask layer object in terms of a flow description of floating-point operations. As described in section 4, the source code analyser *capp* was used to obtain these flow descriptions. The benchmarking process then entailed profiling the application to obtain the achieved floating-point operation rate for a particular problem size on a small number of processors (single processor -1x1 decomposition and 2 processors - 1x2 decomposition). The single processor achieved floating-point operation rate is noted for a particular number of cells per processor workload. This coarse benchmarking was sufficient to obtain the required accuracy for the computation portion of the model. Additionally it was seen that any performance improvement achieved by compiler optimisation could also be captured with this method. Complex investigations into how the original source is modified by the optimising compiler was not required to get a good level of predictive accuracy. Effectively, this approach simplifies model development and benchmarking and at the same time gives a high level of accuracy in the predictions, as can be seen from the results in Section 4. The profiling tool used in this case was PAPI [20], which using hardware counters directly allows the measuring and observation of the actual execution rates and operation counts on the processors. The profiling also allows the results from the source code analysis to be verified, where any unforeseen operation counts can be included into the floating-point operation flow manually if their significance becomes apparent.

Figure 7 details the hardware model for a Pentium 3, 2-way SMP cluster. It includes the SWEEP3D achieved floating-point operation rates as a time for one floating point operation in micro-seconds (in the *clc* section). The parameters MFDG and AFDG are variables used to store global floating-point multiplications and additions respectively. The time for conditional branch opcodes (IFBR) and loop opcodes (LFOR) are taken to be negligible. This was due to the assumption that the achieved floating-point operation rate is an overall estimate of the processor hardware (from processor through the memory hierarchy) and thus it would capture the time cost of all operations including branches and loop start-up costs. As can be seen from the validation of the model, this assumption is acceptable for a compute-intensive application such as SWEEP3D. The naming convention for these opcode variables follows the older PACE benchmark. In this case the achieved flop-rate is for the 50^3 cells per processor problem size. This rate changes according to the problem size per processor and requires updating according to the problem size that will be modelled to obtain accurate results for each case.

This hardware layer model is very good at characterising compute intensive applications such as SWEEP3D where single node/processor efficiency determines the runtime. But it is in general applicable to characterising serial kernels of any application. For applications whose performance is not significantly determined by single node performance, such as applications bound by communication performance, the model itself should take into account the significance of the communication performance. For instance in an application dominated by sparse matrix vector products, serial kernel characterisation will be as simple as taking into account the time for a number of multiplications and additions. This time will be overshadowed by the collective communications performance. Thus the model development will be mainly concerned with characterising

these communication patterns.

4.4 Hardware Layer - Message passing Communication Benchmarking

In Figure 7 the *mpi* section denotes the parameters representing the message passing performance of the system's interconnect. The parameters A to E describe an equation of the form:

$$Transfer\ time\ of\ x\ bytes = \begin{cases} B + Cx, & \text{for } x \leq A \\ D + Ex, & \text{for } x \geq A \end{cases} \quad (3)$$

where x is the size of a message in bytes. This is simply a curve fit for a set of data points. There are three sets of A to E parameters as in Figure 7, representing the gradient and intercept for the above equation for MPI send times, MPI receive times and ping-pong times respectively. Parameter A represents a message size where communication characteristics of the interconnect display different gradients. The data points for this regression are obtained using an MPI benchmark program that carries out timed MPI sends, receives and ping-pongs for increasing message sizes. This simple communication resource model has proved to be sufficient for the communication behaviour exhibited by an application such as SWEEP3D. This could be attributed to the one way blocking sends and receives that dominate the application. If, on the other hand, a large number of collective communications are to be modelled, then a more detailed communication resource model and benchmark procedure will be required – this is on going research.

5 Model Validation

In this section results from the SWEEP3D model written using PACE are presented for three SMP systems. The model results are compared with actual run-time results from these systems. The results are for one energy group with 12 iterations, which is the normal setup for SWEEP3D. The three clusters used here are chosen so as to validate the model for a variety of representative architectures. This includes an Intel Pentium 3 cluster validation of the SWEEP3D model (Table 1) running on a cluster of commodity processors comprising of a traditional x86 Intel architecture. The AMD Opteron cluster validation (Table 2) investigates the performance on the x86_64 AMD architecture. Both of these systems are SMP clusters consisting of 2 processors per SMP node. Finally the SGI Altix system allows the exploration of performance on a genuinely shared memory system with up to 56 processors comprising of Intel Itanium2 (IA-64) processors (Table 3). For each case the problem size consists of 50^3 cells per processor with weak scalability. The k-blocking factor (*mk*) is kept constant at a value of 10. The linear increase in runtime (as well as the corresponding prediction) is due to the increase in the number of pipeline stages which in turn is due to the increase in the number of processors in the 2-D processor array. The variance in predictions are attributed largely to background processes, network load and minor fluctuations in the actual run time of the application.

6 Applications of the Performance Model

An additional advantage of a performance model is its ability to be used for speculative studies in situations such as procurement, installation, maintenance and upgrading. In this section, the developed model is used to speculate on the scaling behaviour of a hypothetical system based on the 2-Way Opteron SMP cluster architecture. As the interconnect of this system (Gigabit Ethernet) is not one that is generally found in HPC systems that are specifically used for extreme scaling, in order to make a realistic speculation, the communication model for the Myrinet 2000 interconnect is used instead of the Gigabit Ethernet communication model. Such model re-usability is a typical advantage of performance modelling and in this case demonstrates the ease of reusing models in the PACE layered approach.

Two problem sizes of interest to the ASCI targets [3] are investigated. These are the 20 million (20x10^6) and 1 billion (10^9) cell problems. Realistic applications of S_N particle transport multi-group problems would expect to include around 30 groups (as opposed to the one group that SWEEP3D implements) and a number of dependent time steps (around 1000 for the ASCI target) [10]. The authors of SWEEP3D suggest scaling results such as these to understand the resource usage and behaviour of particle transport problems in realistic settings. Modern large-scale HPC systems consist of thousands of processors. We use a fixed per processor size of 5x5x100 and 25x25x200 for the 20 million and 1 billion problem sizes respectively, this in turn requires 8000 processors for both the 20 million cell problem and the 1 billion cell problem.

Using an achieved floating-point operation rate of 340 MFLOPS for both the 5x5x100 and 25x25x200 cells per processor problems, combined with the communication model for the Myrinet 2000 interconnect, scalability predictions for up to 8000 processors are shown in Figure 8 and Figure 9. In both cases the model predicts good scaling behaviour. These results concur with those gained through other related analytical models such as [2, 3] and [16]. It can also be seen that this problem configuration when scaled up to 30 energy groups and 10000 time steps will grossly overrun ASCI execution time goals [3] for this type of application. In addition to the speculations based on the achieved floating-point operation rate, Figure 8 and Figure 9 details speculations for this configuration when the achieved floating-point operation rate is increased by 25% and 50%.

Data Size	Num. of PEs	2D Proc. Array	Measurement(sec)	Prediction(sec)	Error(%)
100x100x50	4	2x2	26.54	28.59	-7.72
100x150x50	6	2x3	30.25	30.03	0.74
150x200x50	12	3x4	31.18	32.12	-3.01
200x200x50	16	4x4	32.28	32.78	-1.55
150x300x50	18	3x6	33.72	34.77	-3.11
200x250x50	20	4x5	32.72	34.11	-4.25
200x300x50	24	4x6	33.94	35.44	-4.42
250x300x50	30	5x6	34.73	36.1	-3.94
200x400x50	32	4x8	35.89	38.09	-6.13
200x450x50	36	4x9	37.33	39.42	-5.6
250x400x50	40	5x8	36.8	38.75	-5.3
300x400x50	48	6x8	37.53	39.42	-5.04
250x500x50	50	5x10	39.35	41.41	-5.24
300x500x50	60	6x10	40.24	42.08	-4.57
400x400x50	64	8x8	40.03	40.75	-1.8
300x550x50	66	6x11	41.67	43.4	-4.15
350x500x50	70	7x10	41.19	42.74	-3.76
400x450x50	72	8x9	41.22	42.08	-2.09
400x500x50	80	8x10	43.09	43.4	-0.73
400x550x50	88	8x11	44.22	44.75	-1.2
450x500x50	90	9x10	43.7	44.07	-0.85
500x500x50	100	10x10	44.37	44.73	-0.81
500x550x50	110	10x11	45.09	46.06	-2.16
400x700x50	112	8x14	46.32	48.71	-5.16

Table 1. SWEEP3D performance prediction and validation results on an Intel Pentium-3 2-way SMP cluster. The system consists of 64 nodes interconnected with a Myrinet 2000 interconnect. Each processor is a 1.4GHz Intel Pentium 3 processor with 2GB memory per node. The compiler used is the GNU C compiler version 2.96 on Red Hat Linux 7.2. Compiler flags include -O1 for optimisation. The x87 floating-point instruction set is used by default. The calculated achieved Flop-rate was 110 MFLOPS per processor. The maximum prediction error is less than 10% while the average error is 3.41%; the variance is 4.33%

Data Size	Num. of PEs	2D Proc. Array	Measurement(sec)	Prediction(sec)	Error(%)
100x100x50	4	2x2	8.98	9.69	-7.9
100x150x50	6	2x3	9.59	10.25	-6.83
150x150x50	9	3x3	9.94	10.54	-6
150x200x50	12	3x4	10.57	11.07	-4.7
200x200x50	16	4x4	10.77	11.33	-5.22
200x250x50	20	4x5	11.18	11.85	-5.97
200x300x50	24	4x6	11.95	12.38	-3.59
250x250x50	25	5x5	11.73	12.11	-3.24
250x300x50	30	5x6	12.07	12.64	-4.68

Table 2. SWEEP3D performance prediction and validation results on an AMD Opteron 2-way SMP cluster. The system consists of 16 nodes interconnected with a Gigabit Ethernet. Each processor is a 2GHz AMD x86_64 processor with 2GB memory per node. The compiler used is the GNU C compiler version 3.4.4 on Red Hat Linux (Kernel 2.6). Compiler flags include -O1 for optimisation and -mfpmath=387 for the x87 floating-point instruction set. PAPI profiling and benchmarking demonstrates an achieved Flop-rate of 350 MFLOPS per processor. The maximum prediction error is less than 10%; the average error is 5.35%; the variance is 2.24%.

7 Conclusions and Further Work

A predictive analytical model for the pipelined wavefront application SWEEP3D has been developed, targeting commodity processor cluster systems. The modelling methodology is based on the PACE layered characterisa-

Data Size	Num. of PEs	2D Proc. Array	Measurement(sec)	Prediction(sec)	Error(%)
100x100x50	4	2x2	14.66	13.95	4.81
100x150x50	6	2x3	15.38	14.6	5.07
150x200x50	12	3x4	16.46	15.58	5.35
200x200x50	16	4x4	17.31	15.91	8.09
150x300x50	18	3x6	18.08	16.87	6.69
200x250x50	20	4x5	17.57	16.55	5.82
200x300x50	24	4x6	18.29	17.2	5.98
250x300x50	30	5x6	18.71	17.52	6.33
200x400x50	32	4x8	19.83	18.48	6.79
200x450x50	36	4x9	20.22	19.13	5.39
250x400x50	40	5x8	20.02	18.81	6.04
300x400x50	48	6x8	20.54	19.19	6.57
350x350x50	49	7x7	19.95	18.81	5.71
250x500x50	50	5x10	21.56	20.1	6.76
450x300x50	54	9x6	21.21	19.78	6.74
350x400x50	56	7x8	21.04	19.46	7.51

Table 3. SWEEP3D performance prediction and validation results on an SGI Altix Intel Itanium-2 56-way SMP. The system consists of a single node 56-way SMP (56 processors in total). This is a shared memory system interconnected with a SGI NUMA link4 interconnect. Each processor is a 1.6GHz Intel Itanium 2 processor sharing 112GB memory in total. The compiler used is the Intel C compiler version 8.1 on Red Hat Enterprise Linux AS 3.0. Compiler flags include -O1 for optimisation. The x87 floating-point instruction set is used by default. The calculated achieved Flop-rate was 225 MFLOPS per processor. The prediction error is less than 10%. The average error is 6.23%, while the variance is 0.78%.

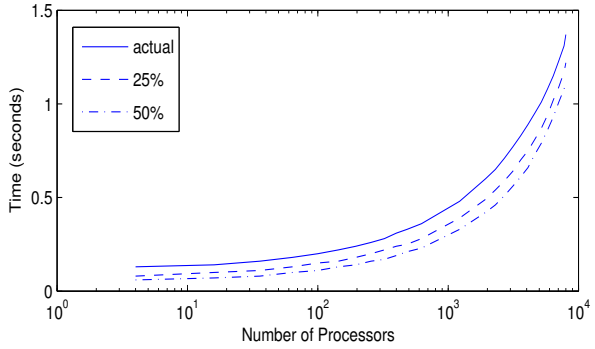

Figure 8. Speculated SWEEP3D Execution Time (with actual achieved floating-point operation rate and with 25% and 50% increase to the achieved rate) - Twenty Million Cell Problem (mk=10, mmi=3, 5x5x100 cells per processor).

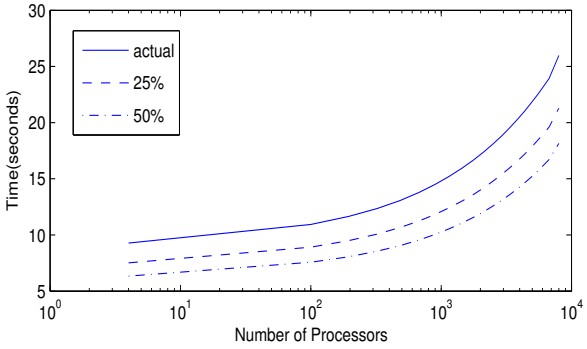

Figure 9. Speculated SWEEP3D Execution Time (with actual achieved floating-point operation rate and with 25% and 50% increase to the achieved rate) - One Billion Cell Problem (mk=10, mmi=3, 25x25x200 cells per processor).

tion approach. A coarser benchmarking and serial kernel characterisation has been introduced to counter discrepancies that resulted from the older PACE opcode benchmarks. The new modelling approach is applicable to modern superscalar features of processors, compiler optimisations and memory hierarchy affects. Using this approach a predictive accuracy is achieved with an error of less than 10%.

The model has been used to speculate on the performance and scaling behaviour of the SWEEP3D application running on a hypothetical system. Two problem sizes of interest to the ASCI targets were modelled in this speculative study. The results were seen to be in good agreement with other related analytical models, while at the same time providing a methodology with reduced model development time and effort.

The SWEEP3D application has a very structured communication pattern. Future work will investigate methods for providing tools to support the modelling of applications with other types of complex communication patterns such as large volumes of collective communications, overlapped computation and communication and communica-

tion in dynamic mesh structures. These types of communication require additional characterisation methods to model their scaling behaviour, network resource usage, and network and message processing contention.

Acknowledgments

The computing facilities: Pentium-3 SMP and SGI Altix systems were provided by the Centre for Scientific Computing at the University of Warwick with support from Joint Research Equipment Initiative grant JR00WASTEQ and Science Research Investment Fund grant.

References

[1] J. Cao, D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd. Performance Modeling of Parallel and Distributed Computing Using PACE. In *Proc. 19th IEEE Int. Performance, Computing and Communications Conf.*, pages 485–492, Phoenix, AZ, USA.

[2] D.J. Kerbyson, A. Hoisie, and H.J.Wasserman. A Comparison Between the Earth simulator and Alphaserver Systems using Predictive Application Performance Models. *Computer Architecture News (ACM)*, December 2002.

[3] A. Hoisie, H. Lubeck, and H.J. Wasserman. Performance and Scalability Analysis of Teraflop-Scale Parellel Architectures using Multidimensional Wavefront Applications. *Int. J of High Performance Computing Applications*, 14(4):330–346, Winter,2000.

[4] A. Hoisie, O. Lubeck, H.J Wasserman, F. Petrini, and H. Alme. A General Predictive Performance Model for Wavefront Algorithms on Clusters of SMPs. In *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, page 219. IEEE Computer Society, 2000.

[5] D.J. Kerbyson, A. Hoisie, and H. J. Wasserman. Modeling the performance of large-scale systems. *IEE Proceedings: Software*, 150(4), July 2003.

[6] G. R. Nudd and S. A. Jarvis. Performance-based middleware for grid computing. *Concurrency and Computation: Practice and Experience*, 17:215–235, 2005.

[7] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, and D.V. Wilcox. PACE: A Toolset for the Performance Prediction of Parallel and Distributed Systems. *Int. Journal of High Performance Computing Applications*, 14(3):228–251, Fall 2000.

[8] D.J. Kerbyson, A. Hoisie, and H.J. Wasserman. Use of Predictive Performance Modeling During Large-Scale Systems Installation. In *1st Int. Workshop on Hardware/Software Support for Parallel and Distributed Scientific and Engineering Computing(SPDEC-02)*, Charlottesville, September 2002.

[9] E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd, T.J. Atherton, and J.S. Harper. An Introduction to the Layered Characterisation for High Performance Systems, December 5, 1997. Research Report CS-RR-335, University of Warwick, Dept. of Computer Science.

[10] Sweep3d. The ASCI Sweep3d Benchmark. `http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/`.

[11] 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.

[12] 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.

[13] E.E. Lewis and W.F.Miller. *Computational Methods of Nutron Transport*. American Nuclear Society, Inc., LaGrange Park, IL, 1993.

[14] R.E. Alcouffe, R.Baker, F. W. Brinkley, D. Marr, R.D. O'Dell, and W. Walters. DANTSYS: A Diffusion Accelerated Neutral Particle Transport Code. Technical Report LA-12969-M, Los Alamos National Laboratory, Los Alamos, NM, 1995.

[15] 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.

[16] D. Sundaram-Stukel and M.K. Vernon. Predictive Analysis of a Wavefront Application Using LogGP. In *PPoPP '99: Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 141–150. ACM Press, 1999.

[17] V.S. Adve, R. Bagrodia, J.C. Browne, E. Deelman, A. Dube, E.N. Houstis, J.R. Rice, R. Sakellariou, D.J. Sundaram-Stukel, P.J. Teller, and M.K. Vernon. POEMS: End-to-End Performance Design of Large Parallel Adaptive Computational Systems. *IEEE Trans. Softw. Eng.*, 26(11):1027–1048, 2000.

[18] R. Bagrodia, E. Deelman, , and T. Phan. Parallel Simulation of Large Scale Parallel Applications. *International Journal of High-Performance Computing Applications*, 15, Spring 2001.

[19] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical report, University of Wisconsin Madison Computer Sciences Department, 1997.

[20] Performance Application Programming Interface. `http://icl.cs.utk.edu/papi/`.