Proceedings Paper:
Bird, R. F., Wright, S. A. orcid.org/0000-0001-7133-8533, Beckingsale, D. A. et al. (1 more author) (2013) Performance modelling of magnetohydrodynamics codes. In: Computer Performance Engineering - 9th European Workshop, EPEW 2012, Revised Selected Papers. 9th European Performance Engineering Workshop, EPEW 2012, 30 Jul 2012 Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) . , DEU , pp. 197-209.

https://doi.org/10.1007/978-3-642-36781-6_14

Performance Modelling of Magnetohydrodynamics Codes

R. F. Bird, S. A. Wright, D. A. Beckingsale and S. A. Jarvis

Performance Computing and Visualisation
Department of Computer Science
University of Warwick, UK
bob@dcs.warwick.ac.uk

Abstract. Performance modelling is an important tool utilised by the High Performance Computing industry to accurately predict the run-time of science applications on a variety of different architectures. Performance models aid in procurement decisions and help to highlight areas for possible code optimisations. This paper presents a performance model for a magnetohydrodynamics physics application, Lare. We demonstrate that this model is capable of accurately predicting the run-time of Lare across multiple platforms with an accuracy of 90% (for both strong and weak scaled problems). We then utilise this model to evaluate the performance of future optimisations. The model is generated using SST/macro, the machine level component of the Structural Simulation Toolkit (SST) from Sandia National Laboratories, and is validated on both a commodity cluster located at the University of Warwick and a large scale capability resource located at Lawrence Livermore National Laboratory.

1 Introduction

Increasing compute performance and maximising supercomputer utilisation has long been a major goal within the High Performance Computing (HPC) industry. Users of these supercomputers are building increasingly more complex and computationally intensive applications, furthering research in a wide variety of science and engineering areas.

In order to meet the demands of the industry, HPC centres are starting to move away from traditional architectures and towards new technologies. One such technology is that of many-core, utilising large numbers of processor units, possibly as part of a heterogeneous architecture. The highly parallel SIMD nature of many-core units, such as GPUs and Intel MIC, allows faster processing of large amounts of data, and can offer performance gains for scientific applications [1–3].

With this increase in technical complexity, it is important to ensure these resources are used effectively. By being able to accurately predict the run-time of a code for a given architecture, we are not only able to make more efficient use of the hardware, but we can also rapidly compare code performance on a variety of different architectures. Furthermore, we are also able to extrapolate results past existing core counts, making predictions of code performance at scale.

In order to predict run-time performance, we need to capture the run-time behaviour of the application and the performance characteristics of the target system. We can then infer information from this data. This process is known as performance modelling. In this paper we describe the development of a performance model for the 2-dimensional variant of *Lare*, a representative plasma physics application. Lare is a Lagrangian remap code, used for solving magneto-hydrodynamics (MHD) equations [4], and is being developed at the University of Warwick.

Specifically we make the following contributions:

- We develop a performance model for Lare. This is the first known predictive performance model for Lare and allows for the prediction of run-time on a variety of current and future architectures based on a minimal number of input parameters. It has been developed such that any future changes or optimisations in the code base can be readily incorporated into the model;
- We validate this performance model on two HPC systems: a commodity cluster located at the University of Warwick and a 260 TFLOP/s capability resource located at Lawrence Livermore National Laboratory (LLNL). We demonstrate an accuracy of greater than 90% for both weak and strong scaled problems;
- Finally, we use our model to provide an evaluation of possible optimisations to Lare. Specifically we perform an investigation into the potential improvements that can be gained from a move towards an Arbitrary Lagrangian-Eulerian (ALE) code in which a more expensive remap step can be applied less frequently.

The remainder of the paper is organised as follows: Section 2 provides a summary of related work; Section 3 provides a background to performance modelling and the operation of Lare; Section 4 discusses the approach taken in developing the performance model; Section 5 provides a validation of the accuracy of the model; Section 6 uses the model to detail potential gains from future optimisations. Finally Section 7 concludes this paper.

2 Related Work

Performance models are a vital tool used by the HPC industry in order to predict the run-times of an application. These predictions can then be used to aid procurement decisions, identify optimisation opportunities, or to predict the behaviour of an application running on a hypothetical future architecture at scale [5].

Hammond et al. [6] show how performance modelling can be used to provide a comparison between two different systems, and use this comparison to aid procurement decisions. They show that the ability to make predictions at scale can

be more valuable than the information obtained from small scale benchmarks. In [7], Herdman et al. use a performance model of an industry strength hydrodynamics benchmark to provide guidance for the procurement of future systems. The authors use their performance model to generate a range of predicted values for comparison, spanning multiple architectures and compiler configurations.

In addition to allowing us to assess current architectures, performance modelling also plays a vital role in enabling us to look at the performance of applications on future architectures at scale. Pennycook et al. [8] show how performance modelling can be used to provide an insight into how applications will perform on a variety on architectures, highlighting the potential benefits of using many-core architectures. Finally, in [9] it is shown that performance modelling can be applied to emerging distributed memory heterogeneous systems to provide an analysis of the performance characteristics and to accurately predict run-times for an application [10].

In [11] a model of parallel computation, LogGP, based on the LogP [12] model, is introduced. It extends the predictive performance of LogP by including the ability to accurately predict communication performance for small messages. This in turn forms the basis on which plug-and-play models can be built, as shown in [13]. The authors show that a model can be built that is able to accurately predict the run-time of an application on a variety of architectures, whilst taking a minimal set of input parameters. This approach has successfully been used by others, including Davis et al. [14] and Sundaram-Stukel et al. [15], and is the basis of the approach taken in this paper.

The previous methods of developing an analytical performance model have been purely mathematically based, but as levels of concurrency and message passing continue to climb this is becoming increasingly difficult to do accurately. By instead simulating the topologies of machines, and message passing behaviour of applications, we can hope to gain increased accuracy. This simulation of hardware can be done using an abstraction of the machine, using both virtual processors and interconnect. By providing values for the specifications of the processors and the interconnect, we can then reproduce the communications performed by the application. By having this closeness between software and hardware it allows for greater performance optimisation of both, as seen in the co-design approach that is being used to move us towards exascale [16–18].

One such tool that facilitates this machine level simulation is SST/macro, one component of Structural Simulation Toolkit [17] from Sandia National Laboratories. SST/macro allows for simulation style models to represent both the control flow of an application, and the message passing behaviour. In doing this it can fully consider such factors as contention and network topology, areas which had previously introduced inaccuracy into analytical models. In order to make use of these advantages, SST/macro has been used to construct the model used in this paper.

3 Background

3.1 Lare

To solve MHD equations, Lare uses an approach based on control volume averaging using a staggered grid. This approach is extended to include complex components such as magnetic fields and shock forces. Lare is run on a fixed size grid for a set number of iterations, an outline of which is shown in Figure 1. The grid used in Lare is 2-dimensional with its width (N_x) and height (N_y) set at run time. This grid is then decomposed in two dimensions ($P_x \times P_y$) such that each processor receives $nx \times ny$ cells, where $nx = N_x/P_x$ and $ny = N_y/P_y$.

```
1 DO
2       ...
3       CALL lagrangian_step
4       CALL eulerian_remap(i)
5       ...
6 END DO
```

Fig. 1: The main compute loop of Lare, operated over for a fixed number of iterations.

The main area of computation in Lare is represented by two key steps, each executed once per iteration: the *Lagrangian step*; and the *Lagrangian remap*. The *Lagrangian step* contains the majority of the computationally intensive physics, representing a significant proportion of the run-time. During this step the grid on which the calculations are performed gets distorted. The gridding scheme used in Lare cannot tolerate large distortions of the computational domain without frequent remapping operations. And thus, some work must be done to correct the grid before computation can continue. The *Lagrangian remap* reforms the grid to its proper coordinates, and involves a significant amount of computation and a series of near-neighbour exchanges are required, which ensures neighbouring cells hold the appropriate values.

3.2 Performance Modelling

The general run-time of a parallel application can be described by Equation 1, which states that the total run-time is the combined total of the compute and communication times.

$$T_{total} = T_{compute} + T_{comms} \tag{1}$$

When developing a performance model it is usual to start with the simplistic case of a serial run, as it contains no communications. In doing this you are able to simplify Equation 1, to that shown in Equation 2.

$$T_{total} = T_{compute} \tag{2}$$

This compute term can then be broken down further, to describe the run-time at a function level. This is shown in Equation 3, where w_g is refered to as the 'grind time', and both `nx` and `ny` represent the decomposed grid size in the relevant direction.

$$T_{compute} = \sum w_g \times (nx \times ny) \tag{3}$$

The term grind time is used to describe the per-cell cost of a function. To obtain these values the code can be instrumented with timers. This can either be done using a profiler such as *gprof* or *scalasca*, alternatively the instrumentation can be done manually. Once these grind times have been found they can be put back into Equation 3 to calculate the total compute time.

4 Developing a Performance Model

In order to fully understand the run-time characteristics of Lare, the code was profiled for both serial and parallel runs. This quantifies the time spent in each subroutine, allowing us to focus our efforts when building our simulation.

In order to construct a model using SST/macro, a skeleton of the code has to be constructed that includes the main areas of compute and communication. As the generation of a comprehensive skeleton application can be a non-trivial process, a small tool was written to facilitate this. The tool performs static analysis on the Fortran source code, and transforms this information into a SST/macro skeleton model. The tool parses the Fortran source code line by line, splitting the line into tokens based on whitespace. These tokens are then matched against an in-built list of keywords, identifying key areas such as subroutine declarations and invocations. Once a keyword is matched, the line is processed. Subroutine declarations are parsed and replicated in the skeleton code. These subroutines are then populated by any function calls made within them. One of the key benefits of the tool is that it identifies MPI communications and is able to flag these to the user and input them into the skeleton. The tool is able to auto-complete much of the information about the MPI call, leaving only the size of the communication buffer to be provided by the user.

In addition to the skeleton, SST/macro requires machine specific details to be specified, such as: topology, network bandwidth and on-node and off-node latencies. These values are obtained using a series of micro-benchmarks.

In order to accurately populate the skeleton application, the main contributors of run-time need to be identified. By profiling Lare and combining this with our existing understanding, it is clear that the two most significant contributors are the Lagrangian step and the Lagrangian remap as previously discussed.

By combining these two steps, we can develop an equation that accurately and concisely summarises the total run-time of Lare as shown in Equation 4.

$$T_{total} = \sum_{i=0}^{iterations} (t_{lagrangian\_step} + t_{remap}) \quad (4)$$

In order to make use of this equation, an incremental approach to building a model was taken, starting with the construction of a serial model.

4.1 Serial Model

For a serial run of Lare, there is no inter-process communication – the run-time is singularly representative of the compute, allowing us to apply Equation 2.

This equation can be decomposed further. The term $T_{compute}$ can further be broken up into its subcomponents, as shown in Equation 3. A table of the relevant grind times for Lare can be found in Table 1.

File Name	**Subroutine**	**w$_g$ Term**
diagnostics.f90	energy_account	$w_{energy\_account}$
lagran.f90	lagrangian_step	$w_{lagrangian\_step}$
lagran.f90	predictor_corrector_step	$w_{predictor\_corrector}$
xremap.f90	remap_x	w_{remap_x}
yremap.f90	remap_y	w_{remap_y}
zremap.f90	remap_z	w_{remap_z}
remap.f90	eulerian_remap	$w_{remap\_remainder}$
diagnostics.f90	set_dt	w_{set_dt}

Table 1: A table depicting the grind times used in modeling Lare, along with their relative location in the source code.

We are able to derive values of the relevant w_g times by running a version of Lare instrumented with timers. Using these values we are able to develop a model that can predict serial run-time to an exceptionally high level of accuracy, using Equation 4.

4.2 Parallel Model

Once a serial model was developed, a parallel model could then be considered in the form shown in Equation 1.

The communication in Lare is dominated by two MPI functions, send-receives and all reduces. The send-receive functions are used to swap neighbour cells, whilst the all reduces collate data. By summing the times taken by these operations, we can represent the communications time as:

$$T_{comms} = \sum t_{Sendrecv} + \sum t_{Allreduce} \quad (5)$$

During the point-to-point communications, the amount of data sent is dependent on the grid size set at compile time. The grid undergoes a coarse decomposition in two dimensions, and is distributed among the processors. This method of decomposition is performed with the aim of minimising the surface-area-to-volume ratio, which in turn increases the ratio of computation to communication. This decomposition strategy is replicated in the model, with SST/macro simulating an exact copy of the communications. Once all the required terms have been identified, they can be incorporated into the model. In order for SST/macro to accurately simulate communications, it requires values for the latency and bandwidth of the target system. These values can be found experimentally with a set of micro-benchmarks that are distributed with SST/macro.

Figure 2 shows elements of both the model and original Lare source code for two methods, `dm_x_bcs` and `remap_x`. It compares the original source to the equivalent representation in the model. In (a) we see the `dm_x_bcs` subroutine that features an *MPI_Sendrecv*. In (b) we can see this has been translated to the equivalent SST/macro MPI call, to be dealt with by the simulated network. Similarly (c) shows an area of compute performed by the original source, this is then replaced by a w_g based calculation in (d).

5 Validation

In order to validate our model, we compare application run times with simulation times for a variety of grid sizes and processor counts on 2 different machines.

5.1 Machines

The two machines used in the validation of the model were the resident supercomputer at the University of Warwick, Minerva, and a large scale capability resource, Sierra, located at LLNL. The specification of the two machines used in this study are summarised in Table 2.

	Sierra	**Minerva**
Processor	Intel Xeon 5660	Intel Xeon 5650
Processor Speed	2.8 GHz	2.66 GHz
Cores/Node	12	12
Nodes	1849	258
Memory/Node	24 GB	24 GB
Interconnect	QLogic TrueScale 4X QDR InfiniBand	
Compilers	Intel 12.0	Intel 12.0
MPI	MVAPICH2 1.7	OpenMPI 1.4.3

Table 2: Details of the experimental machines used.

(a) Original Fortran `dm_x_bcs` Subroutine

```
1  SUBROUTINE dm_x_bcs
2      ...
3     CALL MPI_SENDRECV(dm(nx-1, 0:ny+1), ny+2, mpireal, &
4           proc_x_max, tag, dm(-1, 0:ny+1), ny+2, mpireal, &
5           proc_x_min, tag, comm, status, errcode)
6      ...
7  END SUBROUTINE dm_x_bcs
```

(b) Model `dm_x_bcs` Subroutine

```
1  void dm_x_bcs(int rank) {
2      ...
3      mpi->sendrecv(ny + 2, sstmac::sw::mpitype::mpi_real, \
4      proc_x_max, tag, ny + 2, sstmac::sw::mpitype::mpi_real, \
5      proc_x_min, tag, world(), stat);
6      ...
7  }
```

(c) Original Fortran `remap_x` Subroutine

```
1  SUBROUTINE remap_x ! remap onto original Eulerian grid
2      ...
3      DO iy = -1, ny+2
4           iym = iy - 1
5           DO ix = -1, nx+2
6                ixm = ix - 1
7                ...
8           END DO
9
10     END DO
11     ...
12 END SUBROUTINE remap_x
```

(d) Model `remap_x` Subroutine

```
1  void remap_x(int rank) {
2      ...
3      sstmac::timestamp t(remap_x_w * nx * ny);
4      compute(t);
5      ...
6  }
```

Fig. 2: Code snippet comparing original source code with its representation in the model, including a w_g based compute call and a SST/macro MPI call.

5.2 Weak Scaled Problem

For a weak scaled problem, the grid size is increased with the processor count with the aim of keeping the compute per processor fixed. This is the approach taken for solving increasingly difficult problems in a fixed amount of time. As the processor count increases, more communication between grid cells is required, leading to a general increase in communication time. As the compute per processor remains the same throughout, we expect that our w_g will not change, allowing us to be confident of the predictions for compute time. Table 3 presents a comparison of the experimental run-times against predicated run-times for a weak scaled problem with 3,000,000 cells per core, running for 100 iterations.

(a) Minerva

Nodes	Grid Size	Time (s)	Prediction (s)	Error (%)
1	6000	543.10	527.03	-3.05
4	12000	554.90	528.57	-4.98
9	18000	560.63	541.55	-3.52
16	24000	569.41	549.06	-3.71
21	30000	570.08	551.14	-3.44
36	36000	578.24	558.15	-3.60

(b) Sierra

Nodes	Grid Size	Time (s)	Prediction (s)	Error (%)
1	6000	480.70	465.46	-3.29
4	12000	485.26	466.17	-4.10
9	18000	493.59	466.83	-5.73
16	24000	498.32	476.30	-4.62
21	30000	499.07	478.43	-4.31
36	36000	499.01	480.49	-3.85
49	42000	499.47	481.98	-3.63
64	48000	499.15	483.68	-3.20
81	54000	499.31	487.22	-2.48
100	60000	499.58	488.59	-2.25
121	66000	500.00	490.12	-2.02
144	72000	500.57	491.54	-1.84
169	78000	500.29	492.91	-1.50
196	84000	500.27	495.44	-0.98
225	90000	500.85	496.88	-0.80
256	96000	500.29	499.44	-0.17

Table 3: A table comparing the run-times to simulation times of Lare for Minerva and Sierra.

From the table we can see that the model was able to accurately predict the run-time to an accuracy of greater than 90%. The predicated runtime being consistently slightly lower than the experimental time can be attributed to a

small percentage of the run time behaviour not being incorporated in the prediction, such as the set up costs, which are not captured by the model.

5.3 Strong Scaled Problem

Strong scaling describes the process of solving a fixed problem size with an increasing number of processors. As the processor count increases the aim is to decrease the run-time. A comparison between experimental run-time and predicted run-time is shown in Table 4 for a 16,800 × 16,800 strong scaled problem, running for 100 iterations. This problem size was chosen to give a sufficiently long run time, but still fit in the available memory.

(a) Minerva

Nodes	**Time (s)**	**Prediction (s)**	**Error (%)**
8	518.01	532.85	2.78
12	348.16	364.61	4.51
16	262.74	277.77	5.41
24	172.01	189.51	9.24
32	128.67	133.48	3.61

(b) Sierra

Nodes	**Time (s)**	**Prediction (s)**	**Error (%)**
16	251.06	236.00	-6.38
32	119.60	121.78	1.79
64	61.02	64.16	4.90
128	33.38	35.55	6.12

Table 4: A table comparing the run-times to simulation times of Lare for Minerva and Sierra fora strong scaled problem.

The performance model was able predict the run-time to an accuracy of greater than 90% for a range of core counts.

6 Evaluation of Future Optimisations

An Arbitrary Lagrangian Eulerian (ALE) generalisation of Lare is under development. This would mean the requirement to remap each iteration will no longer hold, and instead a move to ALE would allow the remap step to only be done once the grid becomes sufficiently deformed. By performing an investigation into the expected performance of an hypothetical ALE variant of Lare we can gain valuable insight into the potential performance gains.

By moving to an ALE code, we can vary the frequency of the remap, a metric will be developed to formally determine the value of this frequency (F_r),

but initial indications show that remapping will be required, on average, once every tenth iteration ($F_r = 0.1$) over the course of the simulation. By varying the frequency of the remap, the code will be affected in two main ways. Firstly, it will significantly reduce the general cost per iteration in terms of compute, as the remap step will no longer be present. Secondly, reducing the frequency of the remap step reduces the frequency of inter-process communication. In changing the code in this way, the total cost is no longer as described in Equation 4, but instead includes a term to denote the new remap, as in Equation 6.

$$T_{total} = T_{lagrangian\_step} + T_{remap\_new} \tag{6}$$

This equation can then be reduced further, as shown in Equation 7.

$$T_{total} = \sum_{i=0}^{iterations} (t_{lagrangian\_step}) + \sum_{j=0}^{iterations/F_r} t_{remap\_new} \tag{7}$$

In order to express the new total cost, relative to the old, we can extend Equation 4 to include terms for the relative costs. This is shown in Equation 8.

$$\begin{aligned} T_{total\_new} = &(T_{lagrangian\_step} \times C_{lagrangian\_step}) + \\ &(T_{lagrangian\_remap} \times C_{remap\_new} \times F_r) \end{aligned} \tag{8}$$

If we assume no change to the cost of the Lagrangian step ($C_{lagrangian\_step} = 1$), we can perform an investigation into how the frequency of remap and the cost of remap affect the overall performance. Table 5 shows the percentage decrease in run-time obtained for different values of F_r and C_{remap_new} for a 8,192 square problem on 36 processors performing 100 iterations, in which the remap step contributes just under 65% of the run-time.

		F_r					
		1	**0.5**	**0.25**	**0.2**	**0.1**	**0.001**
C_{remap_new}	**1**	0.00	32.15	48.22	51.44	57.87	64.24
	2	-64.30	0.00	32.15	38.58	51.44	64.17
	4	-192.90	-64.30	0.00	12.86	38.58	64.04
	5	-257.20	-96.45	-16.07	0.00	32.15	63.98
	10	-578.69	-257.20	-96.45	-64.30	0.00	63.66

Table 5: A table showing the percent decrease in run-time for different values of F_r and C_{remap_new} for a 8,192 square problem on 36 processors performing 100 iterations.

From Table 5 we can clearly see that reducing the remap frequency offers large performance gains as the remap frequency decreases for reasonable values of C_{remap_new}. Optimistic projections for this optimised code hope that it will have a similar cost for the lagrangian step ($C_{lagrangian\_step} = 1$), a remap cost that

is around twice as large ($C_{remap\_new} = 2$) and allow the remap to be performed on average every ten steps ($F_r = 0.1$). From Table 5 we can see this may offer a speed-up greater than 50%.

7 Conclusion

In this paper we have presented a predictive performance model for Lare, a MHD code developed by, and maintained at, the University of Warwick. This model allows us to predict the run-time of Lare accurately on a variety of platforms. We have validated the accuracy of the model to 90% on two clusters, a commodity cluster located at the University of Warwick and a 360 TFLOP/s capability resource located at LLNL.

The model was shown to perform well for both weak and strong scaling over a wide range of core counts. We have also used our model to provide a forward look at possible optimisations in the Lare code base, with an evaluation of the gains that may be expected. We also plan to extend the model to the 3-dimensional version of Lare and develop predictive performance models for similar physics codes with the aim of drawing comparisons between these and Lare.

Acknowledgements

We would like to thank Todd Gamblin and Scott Futral for their invaluable help in utilising Sierra and the other Open Compute Facility resources at Lawrence Livermore National Laboratory. This work is supported by the EPSRC grant: A Radiation-Hydrodynamic ALE Code for Laser Fusion Energy (EP/I029117/1).

References

1. Pang, B., li Pen, U., Perrone, M.: Magnetohydrodynamics on Heterogeneous architectures: a performance comparison. CoRR **abs/1004.1680** (2010)
2. Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.m.W.: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08, New York, NY, USA, ACM (2008) 73–82
3. Griebel, M., Zaspel, P.: A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. Computer Science - Research and Development **25** (2010) 65–73 10.1007/s00450-010-0111-7.
4. Arber, T., Longbottom., A., Gerrard, C., Milne, A.: A Staggered Grid, Lagrangian-Eulerian Remap Code for 3-D MHD Simulations. Journal of Computational Physics **171**(1) (2001)
5. Kerbyson, D., Hoisie, A., Wasserman, H.: Modelling the performance of large-scale systems. IEE Proceedings – Software **150**(4) (2003) 214
6. Hammond, S.D., Mudalige, G.R., Smith, J.A., Davis, J.A., Jarvis, S.A., Holt, J., Miller, I., Herdman, J.A., Vadgama, A.: To upgrade or not to upgrade? Catamount vs. Cray Linux Environment. In: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. (2010) 1 –8

7. Herdman, J.A., Gaudin, W.P., Turland, D., Hammond, S.D.: Benchmarking and Modelling of POWER-7, Westmere, BG/P, and GPUs: An Industry Case Study. ACM SIGMETRICS Performance Evaluation Review **38**(4) (2011)
8. Pennycook, S.J., Hammond, S.D., Mudalige, G.R., Wright, S.A., Jarvis, S.A.: On the Acceleration of Wavefront Applications using Distributed Many-Core Architectures . The Computer Journal **55**(2) (2011) 138–153
9. Mudalige, G.R., Giles, M.B., Bertolli, C., Kelly, P.H.: Predictive modeling and analysis of OP2 on distributed memory GPU clusters. In: Proceedings of the second international workshop on Performance modeling, benchmarking and simulation of high performance computing systems. PMBS '11, New York, NY, USA, ACM (2011) 3–4
10. Giles, M.B., Mudalige, G.R., Sharif, Z., Markall, G., Kelly, P.H.: Performance analysis of the OP2 framework on many-core architectures. SIGMETRICS Perform. Eval. Rev. **38**(4) (2011) 9–15
11. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: LogGP: incorporating long messages into the LogP model - One step closer towards a realistic model for parallel computation. In: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. SPAA '95, New York, NY, USA, ACM (1995) 95–105
12. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: towards a realistic model of parallel computation. In: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming. PPOPP '93, New York, NY, USA, ACM (1993) 1–12
13. Mudalige, G.R., Vernon, M.K., Jarvis, S.A.: A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures. In: 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS08). (2008)
14. Davis, J.A., Mudalige, G.R., Hammond, S.D., Herdman, J., Miller, I., Jarvis, S.A.: Predictive Analysis of a Hydrodynamics Application on Large-Scale CMP Clusters. In: International Supercomputing Conference (ISC11). Volume 26 of Lecture Notes in Computer Science (R&D). Springer, Hamburg, Germany (2011) 175–185
15. Sundaram-Stukel, D., Vernon, M.K.: Predictive analysis of a wavefront application using LogGP. SIGPLAN Not. **34**(8) (1999) 141–150
16. Hammond, S.D., Mudalige, G.R., Smith, J.A., Jarvis, S.A., Herdman, J.A., Vadgama, A.: WARPP: A Toolkit for Simulating High Performance Parallel Scientific Codes. In: 2nd International Conference on Simulation Tools and Techniques (SIMUTools09). (2009)
17. Janssen, C.L., Adalsteinsson, H., Kenny, J.P.: Using simulation to design extremescale applications and architectures: programming model exploration. SIGMETRICS Perform. Eval. Rev. **38**(4) (2011) 4–8
18. Rodrigues, A.F., Hemmert, K.S., Barrett, B.W., Kersey, C., Oldfield, R., Weston, M., Risen, R., Cook, J., Rosenfeld, P., CooperBalls, E., Jacob, B.: The structural simulation toolkit. SIGMETRICS Perform. Eval. Rev. **38**(4) (2011) 37–42