Computer Science - Research and Development manuscript No.
(will be inserted by the editor)

Predictive Analysis of a Hydrodynamics Application on Large-Scale CMP Clusters

J.A. Davis · G.R. Mudalige · S.D. Hammond · J.A. Herdman · I. Miller · S.A. Jarvis

Abstract We present the development of a predictive performance model for the high-performance computing code Hydra, a hydrodynamics benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE). The developed model elucidates the parallel computation of Hydra, with which it is possible to predict its run-time and scaling performance on varying large-scale chip multiprocessor (CMP) clusters. A key feature of the model is its granularity; with the model we are able to separate the contributing costs, including computation, point-to-point communications, collectives, message buffering and message synchronisation. The predictions are validated on two contrasting large-scale HPC systems, an AMD Opteron/ InfiniBand cluster and an IBM BlueGene/P, both of which are located at the Lawrence Livermore National Laboratory (LLNL) in the US. We validate the model on up to 2,048 cores, where it achieves a $> 85\%$ accuracy in weak-scaling studies. We also demonstrate use of the model in exposing the increasing costs of collectives for this application, and also the influence of node density on network accesses, therefore highlighting the impact of machine choice when running this hydrodynamics application at scale.

Keywords Hydrodynamics, Performance Modelling, High Performance Computing, Multi-core

J.A. Davis, S.D. Hammond, S.A. Jarvis
Performance Computing and Visualisation
Department of Computer Science
University of Warwick, Coventry, CV4 7AL, UK
E-mail: jad@dcs.warwick.ac.uk

G.R. Mudalige
Oxford eResearch Centre
University of Oxford, Oxford, OX1 3QG, UK

J.A. Herdman, I. Miller
Atomic Weapons Establishment
Aldermaston, Reading, RG7 4PR, UK

1 Introduction

Hydrodynamics applications represent a significant part of the high-performance computing (HPC) workload at organisations such as AWE in the UK and the national laboratories in the U.S. For this reason, benchmark codes representative of these applications, such as SAGE from Los Alamos National Laboratory (LANL) (Kerbyson et al., 2001) and Hydra from AWE, provide a key tool for evaluating HPC systems during design, procurement, installation and maintenance. The development of such HPC codes, the evaluation of their performance on candidate systems and, sustaining performant execution, is a costly and time consuming exercise. To aid in these activities research has been conducted into developing accurate performance modelling tools and techniques for application analysis (Sundaram-Stukel and Vernon, 1999; Hoisie et al., 2000; Kerbyson et al., 2001; Mathis and Kerbyson, 2004; Mudalige et al., 2008).

In this paper we detail the development of an analytic performance model for Hydra, a hydrodynamics benchmark application developed and maintained by AWE. To this end we elucidate the parallel computational operation of Hydra, validate the model on two contrasting HPC systems and apply the model to assess the impact of application components, and machine architecture, on run-time. Specifically, we make the following contributions:

- An analytic performance model is developed for AWE's hydrodynamics benchmark, reflecting its functional behaviour and operation on large parallel HPC systems. The model allows us to predict the time to solution of the application with only a concise set of application and machine parameters. This is the first predictive performance model for Hydra and represents a significant advance to AWE's ability to assess application/architecture capabilities for this class of application.
- We validate the analytic model on two large-scale HPC system architectures – an IBM BlueGene/P and a com-

modity AMD/InfiniBand system, both located at LLNL. The model predicts execution time on up to 2,048 processor cores with $> 85\%$ accuracy. These systems represent two contrasting HPC architectures, and demonstrate the versatility of the model across CMP platforms.
– The model is utilised to assess the impact of component costs including computation, point-to-point communications, collectives, message buffering and message synchronisation, and their relationship with HPC machine architectures, thus illustrating how the model may be put into use by AWE scientists.
– Using the performance model we also investigate the potential impact on this code of increasing cores per die, for both the BlueGene and the AMD/InfiniBand architectures. Our study investigates 2x, 4x and 8x the current core density, and exposes the potential cost of increased network accesses.

The remainder of the paper is organised as follows: Section 2 provides background on the Hydra application, including a summary of its application, problem sizes of interest, parallel decomposition and previous related work on the performance modelling of this class of application; Section 3 presents the analytic model for Hydra; Sections 4 and 5 provide a validation of the model on two HPC systems of interest and demonstrate how the granularity of the model can be used in scaling studies and in architectural assessment. Section 6 concludes the paper.

2 Background

Predicting the dynamic behaviour of materials as they flow under the influence of high pressure and stress is of considerable importance to understanding weapons. Without recourse to underground testing, access to experimental hydrodynamics facilities and supporting high-performance simulation have an important role in providing data to assess weapon safety and performance. Hydra is a benchmark 3D Eulerian structured mesh hydrocode, with which the explosive compression of materials, shock waves, and the behaviour of materials at the interface between components can be investigated. The Hydra benchmark code simulates a cube of mixed materials under stress by discretising the data onto a 3D grid of cells given by $N_x \times N_y \times N_z$ and utilising message passing for parallelisation. Thus, in a typical SPMD fashion, the 3D cube of data is decomposed onto a number of processing elements (PEs) during execution.

The decomposition attempts to distribute the problem as evenly as possible between the available PEs. Thus, given P processing elements, the problem will be decomposed on to a processor grid of $P_x \times P_y \times P_z$ such that a block of cells of size $N_x/P_x \times N_y/P_y \times N_z/P_z$ will be held within a single PE. The decomposition is achieved by finding the factors of P where the grid is partitioned successively, favouring decomposition in the dimension with the highest cell count. In the case of a cubic grid, the application favours the order *z*, *y*, *x* with the exception of powers of 2, where the order is adjusted to *y*, *z*, *x*. Table 1 illustrates example decompositions for various cases of P.

Cores	Case	Decomposition
2048	P is a power of 2	$16 \times 8 \times 16$
1000	P has an integer cube root	$10 \times 10 \times 10$
1650	P has three factors	$10 \times 11 \times 15$
817	P has only two factors	$1 \times 19 \times 43$
2003	P is a prime number	$1 \times 1 \times 2003$

Table 1 Sample P_x, P_y and P_z values at scale

Once the data has been decomposed a PE will iteratively solve its allocated sub-grid of cells. A Hydra iteration consists of a number of bulk-synchronous-type steps. These steps belong to one of three operations: (1) a computation by a PE on its local sub-grid, (2) near-neighbour communications in all three dimensions, (3) collective communications between all the PEs. The specifics of these steps are discussed in detail as part of the model development process in Section 3.1. Hydra performs a number of iterations, depending on a predetermined simulation time, with each iteration taking a variable proportion of the simulation time dependent on the simulation parameters. Typical simulation times for the problem sizes documented here are approximately 10 micro-seconds.

2.1 Related Work

Application performance modelling has been refined and adapted over several generations of systems and applications, and regularly features in articles examining the performance of the world's largest computers. The most notable use of application performance modelling to architecture and machine comparison is provided by LANL's Performance Architecture Laboratory (PAL). PAL detail the development of performance models for a range of applications related to LANL's key high-performance computing workloads. The performance models themselves are used in the comparison of (i) the effectiveness of high-performance computing systems (including potential future architectures) (Mathis et al., 2000; Petrini et al., 2003; Hoisie et al., 2006), (ii) the behaviour of systems during installation and upgrade (Kerbyson et al., 2002; Barker et al., 2009) and (iii) different possible hardware optimisations (Johnson et al., 2008; Kerbyson et al., 2008), amongst others.

Kerbyson et al. (2001) introduce the hydrodynamics code SAGE. They also produce a performance model based on communication, computation and memory contention. They validate this model on an IBM SP3, a Compaq AlphaServer ES45 cluster and a SGI Origin 2000. Our work differs from theirs in a number of respects: (1) because of application differences (communication phases, setup patterns and com-

munication patterns) we have had to develop an entirely new model, full details of which we provide here; (2) we compare and validate this model on two recent high performance compute clusters installed at LLNL; (3) the model is validated against 3D domain decomposition, as opposed to the 2D decomposition seen in Kerbyson et al. (2001); (4) our model allows us to expose several attributes of the application including point-to-point communication, collectives, message packing and unpacking and a breakdown of computation into 10 component terms. Therefore, the model is comprehensive, and we include a complete description in this paper.

3 A Predictive Model for Hydra

Hydra proceeds by performing one of three operations – local computation, near-neighbour communication and collective communication – in a bulk-synchronous fashion. An iteration of Hydra employs several parallel functions, each of which consists of a number of the above operations. An aim of this work is to capture the time to solution by modelling the critical-path run-time of the code as demonstrated in previous analytic modelling research (Mudalige et al., 2008). We develop a general analytic model for the first two key operations (local computation, near-neighbour communication) before applying these to specific segments of the Hydra code. To obtain platform agnostic generalised expressions, we assume that the parallel system consists of number of nodes each with C cores sharing a block of memory and a single Network Interface Controller (NIC). The total number of processor cores is denoted by P. The modelling of collective operations is particular to a target platform, due to their platform specific behaviour.

A local computation performed by a processor forms the simplest of the three operations. It consists only of computation over the block of cells, sized $N_x/P_x \times N_y/P_y \times N_z/P_z$ ($= n_x n_y n_z$) held within a processor. We model this by developing a parameter $w_{g,f}$, which denotes the time (work, or *grind time*) to compute a grid cell; we term this w_g. The subscripted f denotes the name of the function (f) during which this grind time is observed. Thus a local computation can be simply modelled as:

$$T_{comp,f} = (w_{g,f}).(n_x n_y n_z) \quad (1)$$

For clarity the model terms are summarised in Table 2.

3.1 A Hydra Iteration

A Hydra iteration (see Figure 1) consists of a number of functional blocks. Through profiling and code analysis we identify the parallel functions of interest: (1) `mdt`, a function that calculates the time step duration, (2) `mlagh`, a Lagrangian hydro phase, (3) `lartvis`, for the calculation of artificial viscous pressure and, (4) `madv`, used for the computation of the advection of materials in three dimensions. These functions account for over 90% of the parallel run-time of the code, and thus form the target areas of interest for performance analysis and optimisation. The remaining 10% of run-time is spent in utility functions and in performing disk I/O, which we ignore in this model.

```
1  BEGIN ITERATION
2      MDT
3          COMPUTE
4          LARTVIS
5              EXCHANGE(LARTVIS)
6              COMPUTE
7          ALLGATHERS
8
9      MLAGH
10         EXCHANGE(MLAGH(1))
11         LOOP(i)
12             COMPUTE
13             EXCHANGE(MLAGH(2))
14             ALLGATHERS
15             IF(i!=0)
16                 LARTVIS
17                     EXCHANGE(LARTVIS)
18                     COMPUTE
19             END IF
20         END LOOP
21
22     MADV
23         COMPUTE
24         EXCHANGE(MADV)
25             MADVX
26                 COMPUTE
27                 MADVMX
28                     EXCHANGE(MADVM)
29                     COMPUTE
30         REPEAT LINES 24-29 FOR MADVY,MADVZ
31         IF(KAPPA)
32             LARTVIS
33                 EXCHANGE(LARTVIS)
34                 COMPUTE
35         END IF
36 END ITERATION
```

Fig. 1 Outline of a single Hydra iteration

`mdt` (line 2, Figure 1) consists of (1) computation over its local grid cells ($T_{comp,mdt}$), (2) the calculation of viscous pressure ($T_{lartvis}$) and (3) 23 MPI allgather collectives (defined as $T_{allgather}$). We can model the time to execute function `mdt` as:

$$T_{mdt} = T_{comp,mdt} + T_{lartvis} + (23.T_{allgather}) \quad (2)$$

$$T_{lartvis} = T_{exch,lartvis} + T_{comp,lartvis} \quad (3)$$

`lartvis` consists of (1) a near-neighbour exchange and (2) a local computation for the calculation of viscous pressure.

`mlagh` can be modelled in a similar manner to that of `mdt`. There are two notable differences: (1) `mlagh` has two near-neighbour exchanges; (2) there are a number of iterations within the function ($iter_{mlagh}$), which varies according to the period of simulation that the iteration represents, thus:

$$\begin{aligned} T_{mlagh} = {} & T_{exch,mlagh(1)} + \\ & (iter_{mlagh}).(T_{comp,mlagh} + T_{exch,mlagh(2)} + \\ & (2.T_{allgather})) + (iter_{mlagh} - 1).(T_{lartvis}) \end{aligned} \quad (4)$$

Term	Definition	Term	Definition
Compute		Comms.	
P	Total number of processing elements (cores)	$T_{exch,f}$	Time for near-neighbour exchange for function f
$n_{\{x\|y\|z\}}$	Number of local grid cells in *x*,*y*,*z*	$T_{allgather}$	Time to perform an allgather
$T_{comp,f}$	Compute time for function f	C	Cores per node
$iter_f$	Number of iterations within function f	$Nodes_{\{x\|y\|z\}}$	Nodes in *x*,*y*,*z* dimension
$T_{alloc-dealloc}$	Time for array setup	$inter_{\{x\|y\|z\}\_link}$	Number of inter-node comms in an *x*, *y*, or *z* row
T_{mdt}	Wall time of `mdt`	$intra_{\{x\|y\|z\}\_link}$	Number of intra-node comms in an *x*, *y*, or *z* row
T_{mlagh}	Wall time of `mlagh`	$T_{inter\{x\|y\|z\}}(msg)$	Time for inter-node comms (size *msg*) in $x\|y\|z$
$T_{lartvis}$	Wall time of `lartvis`	$T_{intra\{x\|y\|z\}}(msg)$	Time for intra-node comms (size *msg*) in $x\|y\|z$
T_{madv}	Wall time of `madv`	$T_{comm,inter,n}(msg)$	Time to comm *n* messages (size *msg*) on ext. link
$T_{madv\{x\|y\|z\}}$	Wall time of `madv{x,y,z}` subroutine	$T_{comm,intra,n}(msg)$	Time to comm *n* messages (size *msg*) on int. link
$T_{madvm\{x\|y\|z\}}$	Wall time of `madvm{x,y,z}`	$T_{pack,msg}$	Time to pack an MPI message of size *msg*
κ	Control parameter, 1 or 0	$T_{unpack,msg}$	Time to unpack an MPI message of size *msg*

Table 2 Terms for compute and communication model

$mlagh(1)$ and $mlagh(2)$ distinguish the two different near-neighbour message exchanges (see lines 10, 13 of Figure 1).

The function `madv` consists of some local computation followed by calls to three subroutines: `madvx`, `madvy` and `madvz`. A near-neighbour exchange precedes each of these subroutines; at the end of `madv`, viscous pressure may be recalculated (κ is an input parameter set to true (1) or false (0) in the input deck).

$$T_{madv} = T_{comp,madv} + 3.T_{exch,madv} + T_{madvx} + T_{madvy} + T_{madvz} + (\kappa.T_{lartvis}) \quad (5)$$

We define *madvx*, *madvy* and *madvz* as:

$$T_{madv\{x|y|z\}} = T_{comp,madv\{x|y|z\}} + T_{madvm\{x|y|z\}} \quad (6)$$

$$T_{madvm\{x|y|z\}} = T_{exch,madvm\{x|y|z\}} + T_{comp,madvm\{x|y|z\}} \quad (7)$$

The specific message sizes sent during each near-neighbour communication operation are detailed in Table 3.

3.2 Communication Models

Hydra possesses multiple communication phases as highlighted in 3.1, consisting of either point-to-point near-neighbour communication or collective operations.

While these phases communicate different collections of data, the patterns of communication are very similar between the phases and thus we are able to represent a single phase using an abstract communication model. This is then applied across all phases in combination with knowledge of the amount of data to be sent. Likewise, the collective communications are primarily the same type, `MPI_AllGather`, and thus we can substitute a single model for this collective wherever it appears within the Hydra model as a whole.

Since the communication performance will vary between differing machines, we make use of a benchmark application such as SkaMPI (Reussner et al., 1998). This allows us to record important characteristics of the network and understand the impact of setup costs and bandwidth.

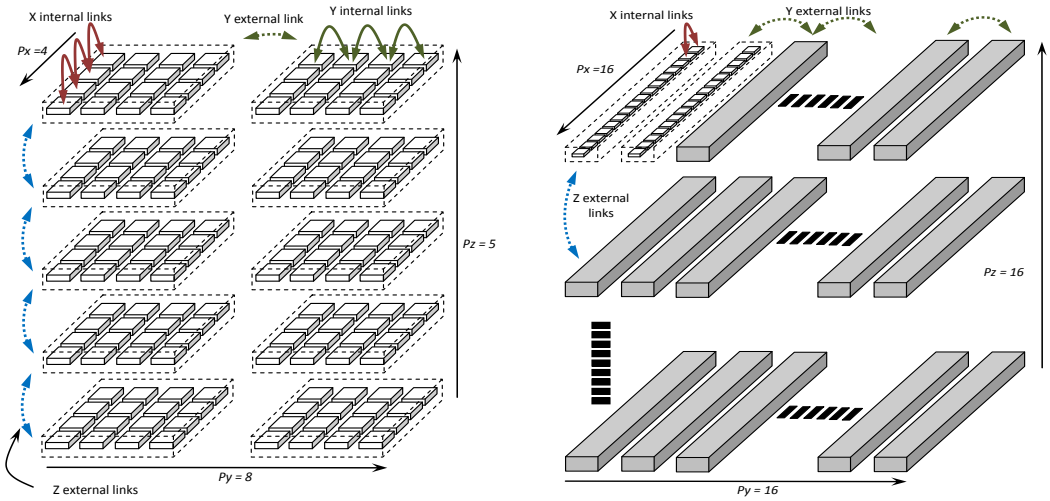
```
1   LOOP DIMENSION={x,y,z}
2     PACKING
3     MPI_ISEND/IRECV both faces in DIMENSION
4     MPI_WAITALL(Near-Neighbour)
5     UNPACKING
6   END LOOP
```

Fig. 2 Typical point-to-point phase

3.2.1 Near Neighbour Point-to-Point

On an individual PE, neighbouring cells will either be stored locally or on a near neighbour PE, as per the 3D decomposition. These are termed *internal* boundaries. Cells that exist on the very edge of the data grid have no neighbouring cell in one or more directions and these are termed *external* boundaries. A near-neighbour communication operation involves a processing element exchanging each of its six faces to the local grid boundaries with its respective neighbours, and consists of non-blocking MPI send/receives in each direction. The six message exchanges are done in each of the three dimensions, first in the *x*-dimension and then followed by the *y* and *z* dimensions respectively. Due to the use of non-blocking MPI communication primitives, an `MPI_Waitall` is called to synchronise the end of the communication operations with all near neighbours along a dimension axis before proceeding to the next dimension.

Within a single Hydra cycle there exist eleven different point-to-point communication instances, yet due to the similarity between some of these calls, we define these as five distinct exchanges: $T_{exch,lartvis}$ (lines 4, 16, 32), $T_{exch,mlagh(1)}$ (line 10), $T_{exch,mlagh(2)}$ (line 13), $T_{exch,madv}$ (line 24, and twice for line 30), and $T_{exch,madvm\{x|y|z\}}$ (line 28, and twice for line 30). The pattern of communication for an exchange remains the same. The key difference is the data being exchanged; that is, each exchange represents different sets of variables to be communicated per cell. This results in different message sizes, and in the case where variables of more than one datatype exist, the number of messages to be sent. These message sizes (defined as *msgsize(f,d)*, where f is the function and d is the dimension) are summarised in Table 3.



(a) 128 Processor cores :$P_x = 4, P_y = 8, P_z = 4$ (b) 4096 Processor cores :$P_x = 16, P_y = 16, P_z = 16$

Fig. 3 Number of external and internal links on a 16 core CMP/SMP-node system

Function (f,d)	Message size (msg)
`lartvis`	$1_{double} \times face$
`mlagh(1)`	$(7_{double} + 1_{int}) \times face$
`mlagh(2)`	$(3_{double} + 1_{int}) \times face$
`madv`	$(5_{double} + 1_{int}) \times (face \times 2)$
`madvmx` `madvmy` `madvmz`	$13_{double} \times (face \times 3)$

where

$$face = \begin{cases} (n_y) \times (n_z) & \text{if } d = x \\ (n_x) \times (n_z) & \text{if } d = y \\ (n_x) \times (n_y) & \text{if } d = z \end{cases}$$

Table 3 Message sizes ($msgsize$) for message exchange

As the data to be exchanged are not stored in contiguous memory, a packing operation needs to be performed before it is sent via MPI; similarly an unpacking operation needs to be performed after message reception. The pseudocode for near-neighbour communication is detailed in Figure 2.

As our assumed parallel system consists of nodes each made up of C cores sharing a NIC, the MPI task allocation becomes crucial when determining which communications are performed between cores on the same processor (intra-node) and between cores on different processors (inter-node). Assuming a Node-Fill Rank assignment, the MPI ranks are assigned along the logical x-dimension of Hydra's decomposition before repeating in the y-dimension (until there are P_y rows), before finally repeating across each xy-plane in the z-dimension. As a result the number of inter/intra connections within a given dimension is a function of P_x,P_y,P_z and C, the number of cores per node. This operation is summarised in equations (8)-(12):

$$Nodes_x = min(\lceil (P_x/C) \rceil, P_x) \quad (8)$$

$$Nodes_y = min(\lceil ((P_x P_y)/C) \rceil, P_y) \quad (9)$$

$$Nodes_z = min(\lceil ((P_x P_y P_z)/C) \rceil, P_z) \quad (10)$$

$$inter_{\{x|y|z\}\_link} = Nodes_{\{x|y|z\}} - 1 \quad (11)$$

$$intra_{\{x|y|z\}\_link} = (P_{\{x|y|z\}} - inter_{\{x|y|z\}\_link} - 1)/Nodes_{\{x|y|z\}} \quad (12)$$

Where $inter_{\{x|y|z\}}$ is the number of inter-node communications in an x, y or z row respectively and $intra_{\{x|y|z\}}$ is the number of intra-node communications in an x, y or z row per node.

We illustrate the external and internal links for a system with 128 processor cores, configured as 8 nodes with 16 cores in each, in Figure 3. Here there are 3 internal x-links communicating within a node. In the y-dimension there are also 3 internal links as well as a single external link. As there is only one NIC per node, we can expect some contention on the NIC for communicating in the y-dimension. That is, there are 4 cores attempting to exchange messages in the y-dimension with their off-node neighbours. In the z-dimension all communications are off-node. Thus there are 16 cores per node attempting to access the NIC simultaneously during message exchanges in the z-dimension.

Given that there are message exchanges contending for the NIC, we adopt the following notation to denote the time to exchange n simultaneous messages of size msg over an external link: $T_{comm,inter,n}(msg)$.

From the number of inter- and intra-node connections (equations 11 and 12), we can derive a model for near-neighbour communications in a similar fashion to that found in Mudalige et al. (2008). This makes the assumption that the communication network is full-duplex and that the time for two nodes to perform a non-blocking send and receive is equivalent to the time for a single blocking send and receive because of Hydra's use of $MPI\_Waitall$.

	DawnDev (BlueGene/P)	Hera
Processor	PowerPC 450(d)	AMD 2.3GHz Opteron
Compute Nodes	1,024	847
Cores/Node	4	16
Total Cores	4,096	13,552
Memory/Node (GB)	4	32
Interconnect	BlueGene torus and tree	4xDDR-InfiniBand switch
Theoretical peak (TFLOP/s)	13.9	127.2
O/S	IBM CNK	CHAOS 4.3
Compilers	IBM XL 11.0 Fortran, 9.0 C	PGI 8.0
MPI	IBM BlueGene MPI	OpenMPI 1.3.2

Table 4 Experimental machines

$$T_{intra\{x|y|z\}}(msg) = \begin{cases} 0 & \text{if } intra_{(x|y|z)\_link} = 0 \\ 1.(\alpha+\beta) & \text{if } intra_{(x|y|z)\_link} = 1 \\ 1.(\alpha+\beta) & \text{if } intra_{(x|y|z)\_link} > 0 \\ & \& \; inter_{(x|y|z)\_link} > 0 \\ 2.(\alpha+\beta) & \text{if } intra_{(x|y|z)\_link} > 1 \\ & \& \; inter_{(x|y|z)\_link} = 0 \end{cases} \quad (13)$$

$$T_{inter\{x|y|z\}}(msg) = \begin{cases} 0 & \text{if } inter_{\{x|y|z\}\_link} = 0 \\ 1.(\alpha+\beta) & \text{if } inter_{(x|y|z)\_link} > 0 \\ & \& \; intra_{(x|y|z)\_link} > 0 \\ 1.(\alpha+\beta) & \text{if } inter_{(x|y|z)\_link} = 1 \\ & \& \; intra_{(x|y|z)\_link} = 0 \\ 2.(\alpha+\beta) & \text{if } inter_{(x|y|z)\_link} > 1 \\ & \& \; intra_{(x|y|z)\_link} = 0 \end{cases}$$

where $\alpha = T_{comm,intra,n}(msg)$ or $T_{comm,inter,n}(msg)$ as appropriate (these values are derived from SKaMPI benchmarks (Reussner et al., 1998)) and $\beta = T_{pack,msg} + T_{unpack,msg}$.

Based on these definitions we can complete the definition for a near-neighbour exchange $T_{exch,f}$.

$$\begin{aligned} T_{exch,f} = {} & T_{intra_x}(msgsize(f,x)) + T_{intra_y}(msgsize(f,y)) + \\ & T_{intra_z}(msgsize(f,z)) + T_{inter_x}(msgsize(f,x)) + \\ & T_{inter_y}(msgsize(f,y)) + T_{inter_z}(msgsize(f,z)) + \end{aligned} \quad (14)$$

3.2.2 MPI Collectives

`MPI_AllGather` is the primary MPI collective used within Hydra, frequently within `mdt` and with more limited use within `mlagh`.

In many respects the `MPI_AllGather` is similar to that of an `MPI_AllReduce`. However, for `MPI_AllGather`, as the number of communication steps scale with P, so the amount of data does also. We assume a pair-wise exchange, where the ranks are split into pairs and exchange data. New pairs are formed on a tree-like basis until all ranks have received from all other ranks, directly or indirectly as described in Benson et al. (2003). This results in a log_2 arrangement, where the amount of data sent doubles per step:

$$\begin{aligned} T_{allgather}(dts) = {} & (\sum_{i=0}^{(log_2 C)-1} T_{comm,intra,n}(2^i * dts)) + \\ & (\sum_{i=log_2 C}^{(log_2 P)-1} T_{comm,inter,n}(2^i * dts)) \end{aligned} \quad (15)$$

Where dts is the size of the datatype in bytes.

3.3 Hydra Model Summary

The Hydra model therefore consists of several microbenchmarks combined through equations 1 through 15. The cost of execution is calculated as the sum of computation, near-neighbour point-to-point communication (including packing and unpacking costs for message exchange) and the global collectives:

$$\begin{aligned} T_{walltime,Hydra} = {} & T_{comp,alloc-dealloc} + T_{mdt} + \\ & T_{mlagh} + T_{madv} \end{aligned} \quad (16)$$

Where $T_{comp,alloc-dealloc}$ is a microbenchmark for memory allocation/deallocation.

4 Model Validation and Projections

4.1 Machines

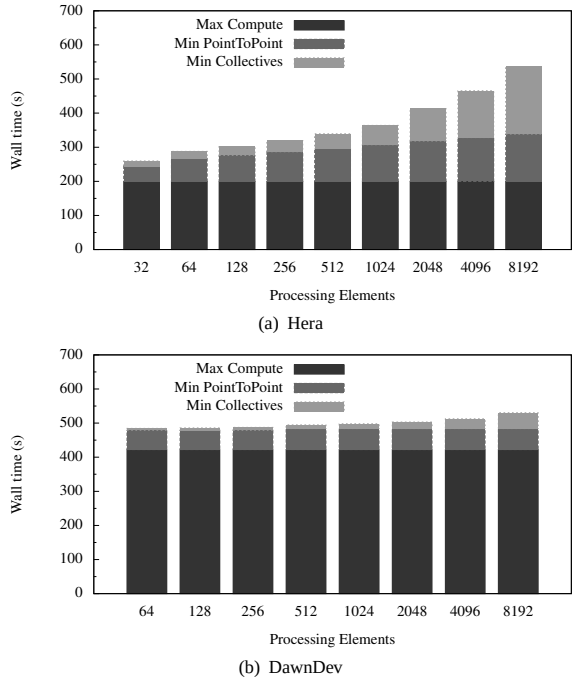
Table 4 summarises the key architectural features of the experimental systems used in this study – an IBM BlueGene/P (DawnDev) and an AMD/InfiniBand system (Hera), both located at LLNL. DawnDev is the development system for Dawn, which itself is the initial delivery system for Sequoia, a 20 PFLOP/s system to be delivered at LLNL starting in 2011 for deployment in 2012. DawnDev is an IBM BlueGene/P system and thus follows the tradition of IBM BlueGene architectures: a large number of lower performance processors (quad-core Power PC 450d running at 850MHz), a small amount of memory per node (4GB per node, 1GB per core), and a proprietary BlueGene torus high-speed interconnect. The BlueGene architecture is recognised as being highly scalable, with a relatively low power footprint. Hera on the other hand utilises densely packed nodes that consist of high-performance multi-core CPUs – four-way 2.3GHz AMD Opteron quad-core CPUs (16-cores per node) with 32 GB memory per node (2 GB per core). Hera uses the InfiniBand DDR high-speed interconnect and exemplifies a typical large capacity resource (127 TFLOP/s peak).

4.2 Model Validation

In order to validate the model we conduct an extensive series of experiments on DawnDev and Hera. The Hydra application is linked with our own supporting Performance Modelling and Timing Module (PMTM) instrumentation library, which allows us to benchmark critical sections of the code

	50^3 problem size						75^3 problem size					
	DawnDev			Hera			DawnDev			Hera		
Cores	Execution	Prediction	Error	Execution	Prediction	Error	Execution	Prediction	Error	Execution	Prediction	Error
	(sec)	(sec)	(%)	(sec)	(sec)	(%)	(sec)	(sec)	(%)	(sec)	(sec)	(%)
32	-	-	-	253.3	259.55	2.44	-	-	-	806.94	783.36	-2.92
64	505.23	484.37	-4.13	291.58	287.76	-1.33	1665.8	1561.46	-6.26	901.91	832.89	-7.65
128	525.15	485.77	-7.15	295.74	302.07	2.1	1702.63	1562.87	-8.21	905.04	859.41	-5.04
256	534.24	487.44	-8.76	310.06	319.77	3.1	1718.68	1564.53	-8.97	974.75	888.32	-8.87
512	528.76	494.53	-6.47	325.15	339.31	4.33	1707.78	1577.7	-7.62	1002.52	918.38	-8.39
1024	544.43	497.75	-8.57	337.54	363.9	7.78	1729.02	1580.92	-8.57	1039.3	953.54	-8.25
2048	584.97	503.04	-14.01	398.1	413.01	3.71	1779.78	1586.21	-10.88	1172.02	1013.88	-13.49

Table 5 Model validations, weak-scaled, 50^3 and 75^3 meshes per core



(a) Hera

(b) DawnDev

Fig. 4 Breakdown of weak-scaled 50^3 per core problem sizes

during execution; we also use the SKaMPI benchmark to measure point-to-point and collective communication costs for both contended and non-contended communication. The benchmarks are used to prime the performance model (details of this process have been previously documented; see Mudalige et al. (2008); Hammond et al. (2009)).

We conduct weak- and strong-scaling studies on a range of problem sizes; for the sake of brevity, we provide a sample of results that demonstrate the capability and accuracy of the performance model. In Table 5 we provide the results of weak-scaling studies on 50^3 and 75^3 data sets on DawnDev and Hera. As this is a weak-scaling study, the total execution time gradually increases with the number of cores, up to a maximum of 1779.78 seconds for a 2,048-core run (DawnDev, 75^3). The model predictions are also shown, demonstrating a model accuracy exceeding 85%. In Figure 4 we demonstrate the ability of the model to expose component costs that contribute to the wall-clock time and project out to 8,192 cores on both the Hera and DawnDev machines.

The increasing cost of collectives is clear to see, and with this information we are able to assess future code development directions as well as the impact of choosing one machine type over another. The impact of collectives is less marked on DawnDev as the IBM BlueGene/P has a dedicated collective network capable of delivering low latency and high bandwidth for fan-in/fan-out collectives.

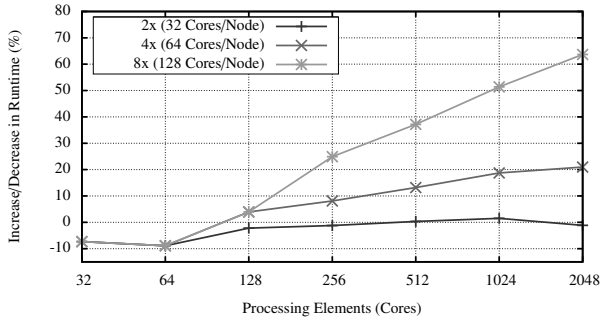
5 Analysis of Future Architectures

A trend of recent processor developments has been growth in the number of cores per socket. Recent processor releases may contain as many as 16-cores per single processor die and predictions are for future designs to have higher core densities still. Whilst providing significant increases in theoretical performance and the potential for greater parallelism at lower cost, the utilisation of more cores per individual die is not without its problems, including higher contention for memory bandwidth and, importantly for a 3D-hydrocode such as Hydra, increased contention for network accesses.

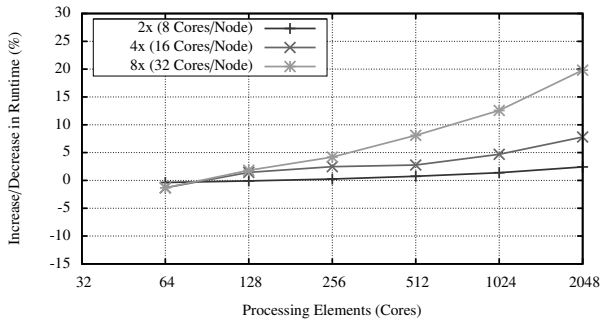
The model developed in this paper is parameterised to accomodate for contention on compute nodes which arises from the use of multiple processing-elements (cores) per node. The number of cores parameter, C, permits flexible evaluation of potential future platforms which may contain significantly higher numbers of cores per node. In Figure 5 we present projections for potential future systems in which the per-core performance is identical to that of the existing benchmarked machines but in which the number of cores per node has been increased by 2x, 4x and 8x.

In these configurations the utilisation of higher core densities results in degraded runtime; this may be advantageous if the higher core counts enable more parallelism to be used in the machine (through increased throughput or, alternatively, strong-scaling operations). The projections in Figure 5 demonstrate that performance may degrade from 25 to 70% on DawnDev and Hera respectively. The lower figure for DawnDev is an indication of the higher interconnect performance versus compute in the BlueGene/P design.

The indications of our projections are that future workloads are likely to experience considerably degraded runtime if the current strategy of increasing core density per die without increasing the number of network interfaces per node continues. Insights such as this give AWE, and similar computing sites, the ability to quantitatively assess the impact of machine architecture choices as well as the oppor-



(a) Hera



(b) DawnDev

Fig. 5 Predicted increase/decrease in run-time for increased core densities per node (weak-scaled, 50^3 per core)

tunity to engineer their codes to minimise the impact of such machine designs before they are deployed.

6 Conclusions

We present the development of a predictive performance model for the high-performance computing code Hydra, a hydrodynamics benchmark developed and maintained by the United Kingdom Atomic Weapons Establishment (AWE). The model allows us to predict the time to solution of the application with only a concise set of application and machine benchmarks. This is the first predictive performance model for Hydra and represents a significant advance to AWE's ability to assess application/architecture capabilities for this class of application. We validate the analytic model on two large-scale HPC system architectures – an IBM BlueGene/P and a commodity AMD/InfiniBand based system. The model predicts execution time on up to 2,048 processor cores with $> 85\%$ accuracy. These systems represent two contrasting HPC architectures, and demonstrate the versatility of the model across CMP platforms. The model is utilised to assess the impact of component costs including computation, point-to-point communications and collectives and their relationship with HPC machine architectures, thus illustrating how the model may be put into use by AWE scientists.

Acknowledgements Access to the AWE Hydra benchmark was provided under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme) from the United Kingdom Atomic Weapons Establishment. We are grateful to Scott Futral, Jan Nunes and the Livermore Computing Team for access to, and help in using, the DawnDev and Hera machines located at LLNL. We also acknowledge support of the Centre for Scientific Computing at the University of Warwick and the UK Science Research Investment Fund.

References

K.J. Barker, K. Davis, and D.J. Kerbyson. Performance Modeling in Action: Performance Prediction of a Cray XT4 System During Upgrade. In *IEEE International on Parallel and Distributed Processing Symposium (IPDPS 2009)*, May 2009.

G. Benson, C.W. Chu, Q. Huang, and S. Caglar. A comparison of MPICH allgather algorithms on switched networks. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 335–343, 2003.

S.D. Hammond, G.R. Mudalige, J.A. Smith, S.A. Jarvis, J.A. Herdman, and A. Vadgama. WARPP - A Toolkit for Simulating High-Performance Parallel Scientific Codes. In *Proc. 2nd International Conference on Simulation Tools and Techniques*, March 2009.

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

A. Hoisie, G. Johnson, D.J. Kerbyson, M. Lang, and S. Pakin. A Performance Comparison Through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm, and Purple. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC 2006)*, pages 74–84, 2006.

G. Johnson, D. J. Kerbyson, and M. Lang. Optimization of Infiniband for Scientific Applications. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008)*, April 2008.

D.J. Kerbyson, H.J. Alme, A. Hoisie, F. Petrini, H.J. Wasserman, and M. Gittings. Predictive Performance and Scalability Modelling of a Large-Scale Application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing(SC 2001)*, 2001.

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*, 2002.

D.J. Kerbyson, M. Lang, and G. Johnson. Infiniband Routing Table Optimizations for Scientific Applications. *Parallel Processing Letters*, 18(4):589–608, 2008.

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 2004)*, Santa Fe, NM, April 2004.

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.

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 2008)*, April 2008.

F. Petrini, D.J. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8, 192 Processors of ASCI Q. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC 2003)*, pages 55–62, 2003.

R. Reussner, P. Sanders, and M. Muller. SKaMPI: A Detailed, Accurate MPI Benchmark. *Recent advances in Parallel Virtual Machine and Message Passing Interface*, pages 52–59, 1998.

D. Sundaram-Stukel and M.K. Vernon. Predictive Analysis of a Wavefront Application Using LogGP. In *Proceedings of Principles and Practice of Parallel Programming*, pages 141–150, 1999.