

# Performance Evaluation and Modeling of a Parallel Astrophysics Application\*

G. (Kumar) Mahinthakumar\* and Mohamed Sayeed, Dept. of Civil Engineering, North Carolina State University  
John Blondin, Dept. of Physics, North Carolina State University  
Patrick Worley, Computer Science and Mathematics Division, Oak Ridge National Laboratory  
William Raphael Hix and Anthony Mezzacappa, Physics Division, Oak Ridge National Laboratory

**Keywords:** Performance Evaluation, Performance Modeling, Parallel Computing.

**Abstract:** We investigate the performance of a large-scale parallel astrophysics application code on two modern parallel architectures. Machine characteristics are first evaluated by performing micro-benchmarking studies. Application performance is then studied in terms of floating-point performance and speedup. A semi-empirical model consisting of application and machine parameters is then constructed to model the timing of the application. The parameters are first fit using measured times and then the predictive capability of the model is evaluated against additional measured times.

## INTRODUCTION

Achieving maximum possible performance on large-scale scientific applications is more challenging than ever due to the increasing complexity of modern parallel architectures. Performance of a parallel application primarily depends on two characteristics, one pertaining to the application and the other pertaining to the machine. Application characteristics include the primary algorithmic kernels, programming language, parallelization strategy, problem size and other input parameters. Machine characteristics include hardware metrics such as CPU performance, memory performance, and communication performance and software metrics such as compiler performance and efficiency of communication and math libraries. Producing applications that work well on a wide range of hardware platforms is a non-trivial task because the machine characteristics and its interdependence on application characteristics vary from platform to platform. The aforementioned reasons also make performance prediction a difficult task. In recent years, performance

analysis and modeling on parallel architectures have attracted considerable attention. These studies range from kernel benchmarking studies, application performance studies, performance modeling, and detailed comparative analysis of newer architectures [e.g., Dunigan et al. 2003, Petrini et al. 2003, Vetter and Yoo 2002, Parashar and Hariri 2000].

This paper deals with a widely used finite difference astrophysics code containing over 6500 lines of Fortran and MPI (Message Passing Interface, Gropp et al. 1999). The program is currently configured to run simulations of the Sedov-Taylor blast wave solution in 2D and 3D spherical geometries. While the code deals with a specific application area, the algorithmic kernels used are indicative of many numerically intensive finite-difference codes. Therefore, the performance analysis and modeling methodologies used in this paper have more general applicability.

## APPLICATION DESCRIPTION

The astrophysics code, VH1 (Virginia Hydrodynamics), was originally developed by the numerical astrophysics group at University of Virginia. It solves the following equations describing ideal inviscid compressible flow of gas hydrodynamics [Blondin 1999].

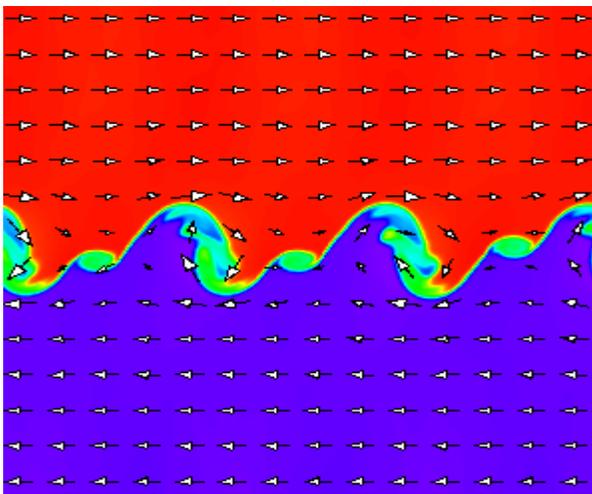
$$\begin{aligned}\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= 0 \\ \frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) + \nabla p &= \mathbf{F} \\ \frac{\partial (\rho \varepsilon)}{\partial t} + \nabla \cdot (\rho \varepsilon \mathbf{u}) + \nabla (p \mathbf{u}) &= G + \rho \mathbf{u} \cdot \mathbf{F}\end{aligned}$$

\* This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

\* Corresponding author. [gmkumar@ncsu.edu](mailto:gmkumar@ncsu.edu), 919-515-7696. Campus Box 7908, Raleigh, NC 27695.

where  $\rho$  is the mass density,  $\mathbf{u}$  is the velocity vector,  $p$  is the pressure,  $\epsilon$  is the total specific energy, and  $\mathbf{F}$  and  $\mathbf{G}$  are momentum and energy source terms (e.g., gravity, radiative cooling etc.). The unknowns are  $\mathbf{u}$ ,  $\rho$ , and  $p$ . Note that the second equation is a vector equation. For example, for a two dimensional problem we have 4 unknowns ( $u_x$ ,  $u_y$ ,  $\rho$ , and  $p$ ) and 4 equations. i.e., 2 scalar equations (first and third equations) and 1 vector equation (2<sup>nd</sup> equation) that can be expressed as two scalar equations.

VH1 uses finite-differences with a piecewise parabolic approximation [Colella and Woodward 1984] and a Riemann solver to solve the aforementioned equations. Time stepping is explicit. VH1 uses 2nd order operator splitting and 1D Lagrangian hydrodynamics in each coordinate direction (explicit PPM method). The code uses a Lagrangian coordinate system with a remap routine that transforms the results back to a fixed Eulerian grid. Cartesian (1D or 2D), Spherical (2D), and symmetrical cylindrical (1D) geometries are supported. The original VH1 code was written in Fortran 77. EVH1 is an enhanced parallel version of VH1 developed jointly by NCSU and ORNL that includes extension to 3D, parallelization, and Fortran 90 constructs. Operator splitting is implemented with explicit `MPI_Alltoall` calls, to restructure domain decomposition.



**Figure 1.** EVH1 simulation for Kelvin-Helmholtz instability at a shear surface

Figure 1 shows output from a typical EVH1 simulation in a 2D Cartesian box for the evolution of the Kelvin-Helmholtz instability at a shear surface. The top and bottom of the computational domain have opposite  $X$  velocity producing a shear layer in the middle. This contact is wiggled (a sine wave) in order to excite the KH

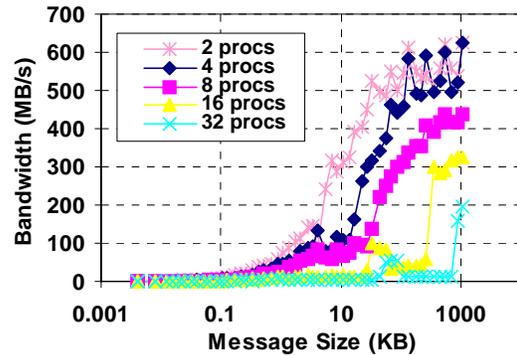
instability. Density of the two fluids is slightly different to make the instability visible.

## ARCHITECTURE DESCRIPTIONS

Two popular parallel architectures are used in this investigation: (i) IBM p690 cluster at ORNL (Cheetah), and (ii) an Intel Itanium 2 Linux cluster at NCSA (TG cluster).

The IBM cluster at ORNL has 27 32-processor IBM p690 nodes connected by the IBM SP Switch2 (Colony) and Corsair PCI network adapters. Each one of these 864 processors is a 1.3 GHz superscalar processor capable of up to 5.2 Gflops. The theoretical peak performance of the entire machine is about 4.5 Tflops.

The teragrid cluster at NCSA consists of 256 2-processor Itanium2 nodes connected by Myrinet. The Itanium2 processor clocks at 1.3 Ghz and is also capable of 5.2 Gflops. Even though both processors have the same theoretical peak, TG cluster performance is generally superior because all three of its cache levels are integrated into the Itanium2 chip. The combined theoretical peak of this system is about 2 Tflops.



**Figure 2.** MPI\_Alltoall performance on the IBM P690

Theoretical vendor estimates of machine characteristics such as Mflop performance, communication bandwidth, and latencies are most often indicative of upper bounds in an ideal situation. Most full applications do not see performance based on these numbers due to factors such as memory issues, synchronization issues, message contention, and environmental noise. Therefore we conducted several microbenchmarking studies to identify realistic computation and communication characteristics for each architecture. The [llcbench](#) (low-level characterization benchmarks) from the University of Tennessee was used. Single processor matrix-matrix multiply DGEMM

performances indicated that the TG processors outperform the IBM processors by a factor of 1.3 to 1.5 (3-4 Gflops vs. 4-5 Gflops). Several MPI benchmarks, including those that give MPI\_Send latencies, bisection, point-to-point, MPI\_Allreduce and MPI\_Alltoall bandwidths, were run on both architectures. Since our application code uses MPI\_Alltoall as its primary communication call we present only these results in this paper. Figures 2 and 3 show the MPI\_Alltoall performance on the IBM P690 and the TG cluster. The 2-processor bandwidth varies from 100 MB/s to 600 Mb/s for the IBM P690 and from 20 MB/s to 180 MB/s for the TG cluster for messages sizes ranging from 1 KB to 32 MB. The bandwidths for other processor counts are up to a factor of 3 lower on both architectures.

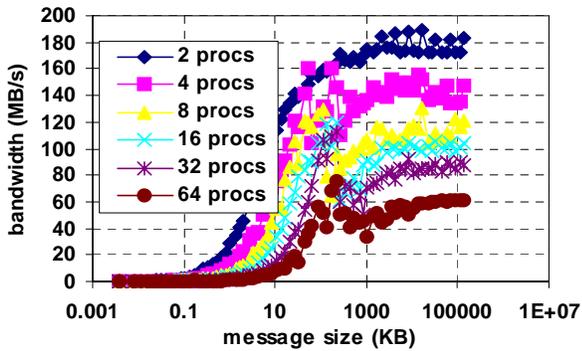


Figure 3. MPI\_Alltoall performance on the TG cluster

Machine Name	Intranode			Internode		
	MAx	Avg	Min	Max	Avg	Min
TG Cluster	14	4	3	98	18	16
IBM P690	22	7	6	118	10	7

Table 1. Measured MPI latencies in microseconds

Measured MPI latencies for p690 cluster and the TG cluster are shown in Table 1. These latencies indicate that the IBM P690 latencies are much higher than the TG cluster latencies for both intranode and internode communication. This is expected for internode communication since the IBM p690 uses the Colony interconnect and the TG cluster uses Myrinet. The measured intranode latencies may be higher on the IBM p690 because its nodes are much larger (32 processors compared to just 2 processors on the TG cluster) and the tests were conducted in a non-dedicated mode.

### PERFORMANCE RESULTS

Performance of the EVH1 code was analyzed extensively using a number of tools including hpm, gprof, mpiP, svpablo, PAPI, and

MPI\_wtime. Many of the aforementioned tools give highly detailed routine level performance. While these were useful in identifying performance bottlenecks and resulted in subsequent improvement of the code, much of the analysis is too specific to the application and not of interest here. Therefore we focus only on the overall performance characteristics of the code and a comparison among the architectures. Performance is analyzed in terms of speedup and Gflop performance. In Figure 4 we compare the Gflop performance of the code for a fixed size problem when increasing number of processors. The numbers indicate that the TG cluster outperforms the IBM p690 by less than 25% up to 32 processors. At 64 processors, TG cluster performance is almost double that of the IBM p690. This is obviously due to the slow MPI performance on the IBM when going across the nodes.

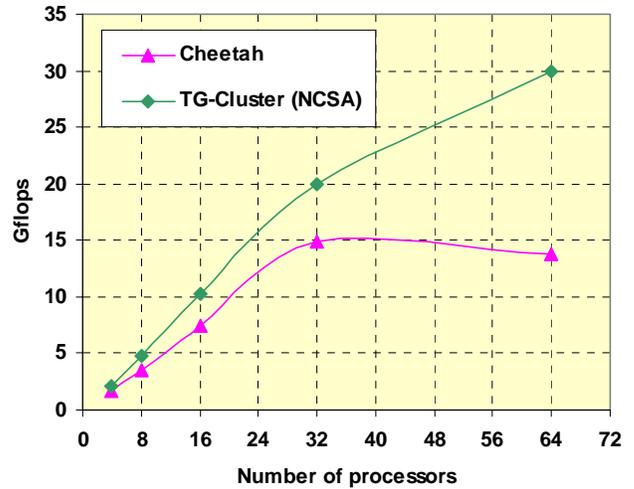


Figure 4. Performance comparison of EVH1 for a 64x64x64 problem

In Figure 5, we show percentage of time spent in

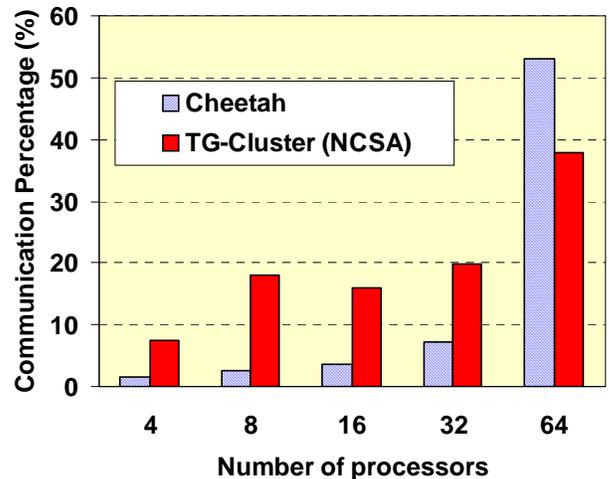


Figure 5. Percentage of time spent in communication for a 64x64x64 problem

communication for both architectures for the same problem. This percentage is calculated by (communication time/total time)\*100. These results confirm our previous assertion that the slower inter-node communication on Cheetah when going outside the 32-processor nodes causes the big jump in communication time when going from 32 to 64 processors (from less than 10% to over 50%).

## PERFORMANCE MODELING

EVH1 was run several times with different application parameters (problem size, simulation time) and machine parameters (different machines, number of processors) and the total run time and the communication time were recorded for each run. Our goal here is to first fit these times using an appropriate model and a subset of the measured data and then test the effectiveness of this model by predicting the times for remainder of the data.

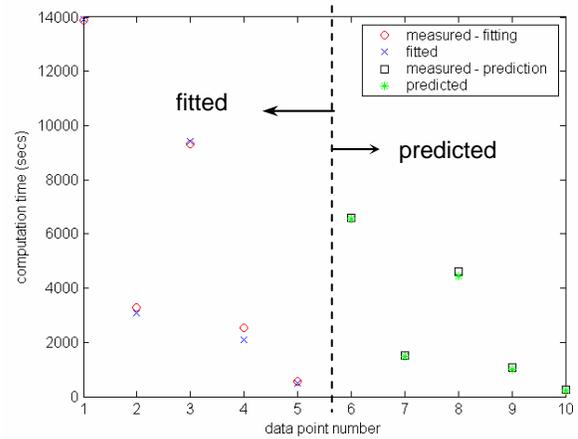
In our first attempt we developed a completely empirical model that used an arbitrary power law expression with constants fit from measured data using a nonlinear regression technique. While this model fit the measured data well, its prediction capabilities were poor even on the same architecture. This warranted a replacement of our original power law expression with a semi-empirical model that conforms to computations and communications that are actually being performed in the code. This required slightly different computation models for 2D and 3D problems due to the manner in which time stepping is performed in EVH1. Further, lower and upper bounds were placed on the fitted parameters so that they are within acceptable ranges. The MATLAB function `lsqcurvefit` was used to perform the nonlinear regression. For problems with bound constraints, `lsqcurvefit` uses the conjugate gradient trust-region method for non-linear regression.

### Computation Timing Model

For 2D problems the number of time steps (or cycles) is linearly proportional to the stop time ( $T$ ) and the horizontal resolution ( $nx$ ). Note that the number of time steps is also a function of horizontal grid resolution because of the limitation imposed by the Courant condition (higher grid resolution translates to smaller grid spacing which in turn requires smaller time steps). Therefore we could use the following expression for approximating the computation time:

$$t_{comp} = a \cdot T \cdot nx \cdot \left[ b + c \left( \frac{nz \cdot nx^2}{np} \right)^d \right] + e$$

where  $T$  is the simulation stop time,  $nz \cdot nx^2$  is the overall problem size, and  $np$  is the number of processors.  $a$ ,  $b$ ,  $c$ ,  $d$ , and  $e$  are the constants to be fitted. Of these,  $a$  represents the number of cycles per time step,  $b$  is the startup overhead within each cycle,  $c$  is computation time per grid point, exponent  $d$  accounts for non-linearity due to cache effects, and  $e$  accounts for startup overhead. The imposed bound constraints are the following:  $a$ : 1000 – 10000,  $b$ : 0 – 1e-5,  $c$ : 1e-8 – 1e-3,  $d$ : 1 – 2,  $e$ : 1 – 20. Even though a portion of the startup overhead  $e$  could be



**Figure 6.** Computation fit for 3D problems on Cheetah

a function of problem size and the number of processors, we have opted to keep this as a constant parameter to keep the number of fitted parameters to a minimum. The fitted computation model parameter values for both architectures are summarized in Table 2.

For 3D problems, time stepping is adaptive; i.e., a smaller time step size is used in the beginning of the simulation and then it is gradually increased with increasing number of time steps. Therefore, the number of time steps (or cycles) for 3D problems is non-linearly proportional to the stop time ( $T$ ) and the horizontal resolution ( $nx$ ). Therefore one additional parameter ' $k$ ' that accounts for this non-linearity is required. With this parameter  $k$ , the computation timing model for 3D problems can be written as:

$$t_{comp} = a \cdot (T \cdot nx)^k \cdot \left[ b + c \left( \frac{nz \cdot nx^2}{np} \right)^d \right] + e$$

Due to the limited amount of 3D data available for fitting,  $b$  is assumed to be zero (note that it is very close to zero for the 2D cases). Only  $a$ ,  $c$ ,  $d$ ,  $e$ , and  $k$  are fitted. Again, the fitted values are shown in Table 2. Note that the values of  $a$  and  $k$  are nearly equal for both architectures. This is expected since both these parameters are application dependent and not machine dependent. Also, the value of  $c$  (computation time per grid point) is slightly higher for Cheetah than the TG cluster since it is a slower architecture. To illustrate the effectiveness of our computation time model, computation fit and prediction for 3D problems is shown in Figure 6 for Cheetah. The fitting and prediction are both very good for this case. The computation fit for the other cases are not shown due to length considerations but these were observed to be equally good.

Parameter	Fitted Values			
	Cheetah		TG Cluster	
	2D	3D	2D	3D
$a$	4965	5034	4979	5031
$b$	1.1e-12	0	4.5e-14	0
$c$	7.78e-6	1.9e-4	3.48e-6	6.2e-5
$d$	1.102	1.06	1.003	1.13
$e$	6.64	20	7.36	20
$f$	4965	5034	4979	5031
$g$	1.24e-8	1.02e-5	1.33e-7	3.72e-4
$h$	1.01	1.49	0.998	1.50
$i$	1e-5	2.9e-4	2.78e-5	1.26e-4
$j$	4.6e-7	9.18e-5	1.93e-7	3.57e-4
$k$	-	0.81	-	0.82

Table 2. Fitted parameter values

### Communication Timing Model

The primary communication operation in the EVH1 code is the MPI\_Alltoall call for a matrix transpose like operation. The decomposition is similar to a 1-D striped partitioning of an  $nx \times nx$  matrix. In this operation, each processor performs an all-to-all broadcast of blocks of size  $nx^2/np^2$ . Every processor needs to exchange  $(np-1)$  blocks of size  $nx^2/p^2$  with every other processor. i.e., an all-to-all broadcast. Diagonal blocks undergo only local (intraprocessor) transposition. Once the appropriate blocks are exchanged, every processor should perform a local transposition on  $np$  blocks. Figure 7 shows this operation schematically.

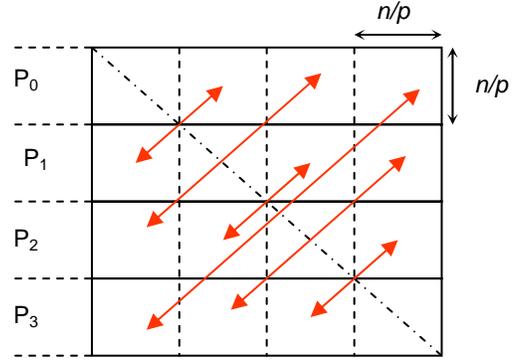


Figure 7. Communication in a matrix transpose operation using 1-D striped partitioning.

The communication operation is the same for both 2D and 3D problems. However, the total number of cycles will be different in the 2D and 3D problems as described for the computation model. In each time step two matrix transpose type operations (each involving an MPI\_Alltoall) are performed. The following expression is used for the communication timing for 2D problems:

$$t_{comm} = 2f \cdot T \cdot nx \cdot \left\{ \left( \frac{nx^2}{np} - nx \right) g + (np-1)^h \left( i + j \frac{nx^2}{np^2} \right) \right\}$$

Where  $g$ ,  $h$ ,  $i$ , and  $j$  are the constants to be fitted. With the exception of  $f$ , all parameters are machine dependent. Here  $f$  represents the number of cycles proportional to the simulation time (set equal to  $a$ ),  $g$  represents the per word

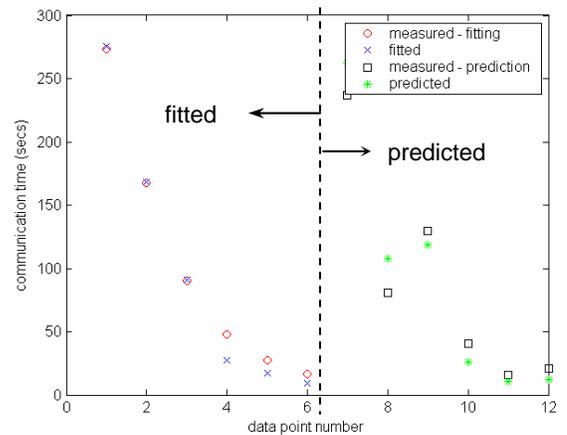
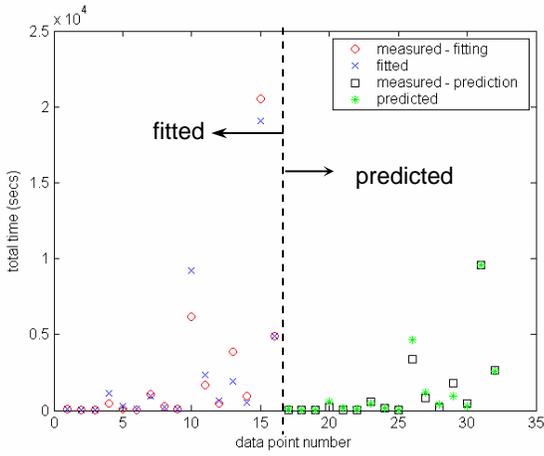


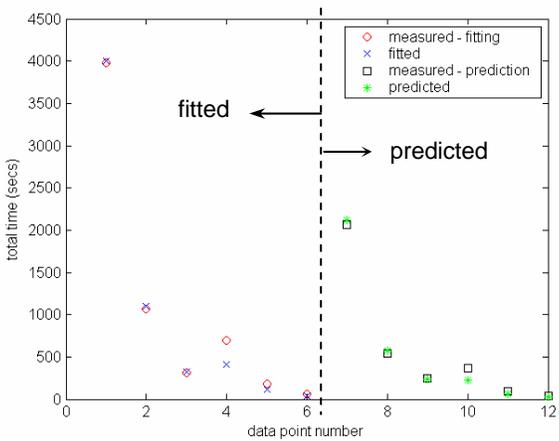
Figure 8. Communication fit for 2D problems on the TG cluster

memory-to-memory copy time ( $t_m$ ),  $i$  represents the message latency,  $j$  represents the message bandwidth, and  $h$  represents non-linearity due to message contention. All parameters except  $f$  are fitted; for  $f$ , the fitted  $a$  value from the computation time is used. As in the fitting of computation timing, the fitted parameters are bounded based on realistic machine parameters. The following bound constraints are used:  $g: 1e-9 - 1e-3$ ,  $h = 1 - 1.5$ ,  $i = 1e-9 - 1e-3$ ,  $j = 1e-9 - 1e-3$ . The fitted values are summarized in Table 2.

The fitted values of  $i$  and  $j$  are acceptable for each architecture as described in the next paragraph. To illustrate the effectiveness of our communication time model, communication fit for 2D problems on the TG cluster is shown in Figure 8.



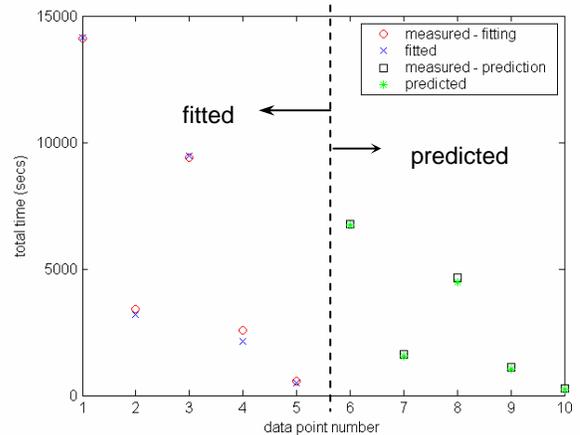
**Figure 9.** Model fitting and prediction for 2D problems on Cheetah



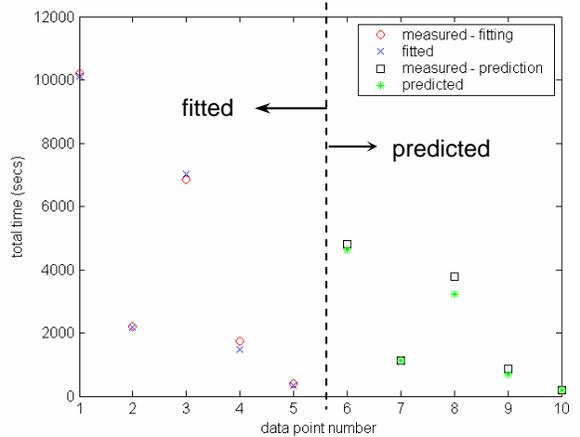
**Figure 10.** Model fitting and prediction for 2D problems on TG cluster

While the communication prediction is not as good as the computation prediction, it still captures most points within 80% accuracy. Communication fit for other cases are not shown but are equally good in most cases.

The measured MPI latencies for the TG cluster vary between 3 - 14 microseconds for intranode communication and 16 - 98 microseconds for internode communication (Table 1). The fitted value,  $i = 27$  microseconds, is well within these values, indicating that our model fit captures realistic values. The measured MPI bandwidth for MPI\_Alltoall varies between 20 Mb/s to 180 Mb/s depending on the number of processors and message size. This translates to about 0.04 to 0.4 microseconds per word transfer. The fitted value of  $j = 0.2$  microseconds falls well within this range.



**Figure 11.** Model fitting and prediction for 3D problems on Cheetah



**Figure 12.** Model fitting and prediction for 3D problems on the TG cluster

As in computation timing modeling the following modification is adapted for communication timing of 3D problems with the additional parameter  $k$ :

$$t_{comm} = 2f \cdot (T \cdot nx)^k \cdot \left\{ \begin{array}{l} \left( \frac{nx^2}{np} - nx \right) g \\ + (np - 1)^h \left( i + j \frac{nx^2}{np^2} \right) \end{array} \right\}$$

where  $g$ ,  $h$ ,  $i$ , and  $j$  are the constants to be fit. A computation fitted value of  $k$  ( $= 0.82$ ) is used. Again,  $f$  is set to the computation fitted value of  $a$ . All parameters are defined as before. As in the fit of computation timing, the fitted parameters are bounded based on realistic machine parameters. Same bounds as in the 2D case are assumed for the 3D problems. The fitted values are again shown in Table 2.

The overall timing is obtained by simply adding the communication time to the computation time. The measured, fitted and predicted overall timing data points for all cases are shown in Figures 9 to 12. Of the 63 data points about half (32) were used for fitting and the remainder were used for testing the prediction. Of the 31 data points predicted, 26 showed an accuracy of 85% or higher when compared to the measured values. Four of the 5 points that resulted in a poor prediction were for 2D problems on Cheetah. We attribute this to the fact that the overall time for 2D problems is more sensitive to communication timing and our communication prediction on Cheetah was not as good as on the TG cluster.

## SUMMARY AND CONCLUSIONS

In this paper we have analyzed and compared the performance of an extensively used astrophysics code on two modern parallel architectures. We have developed a semi-empirical model to predict the timings of this application for various problem configurations on the two architectures. The model includes both application and machine dependent parameters. The fitted machine dependent parameters are well within realistic values for both architectures. Prediction accuracy is over 80% for most data points.

While the model contains certain specificities to the target application, several aspects of the model could be easily extended to other finite-element or finite-difference codes running on parallel architectures. For example, in the computation time model, the following

parameters will occur in most codes: (i) a parameter represents the number of cycles or time steps, (ii) a parameter that accounts for the computation time per grid point, (iii) parameters that account for startup overhead and I/O time, and (iv) a parameter that accounts for cache effects. Similarly, in the communication time model, the following parameters will be generic to most applications: (i) a parameter that accounts for message bandwidth, (ii) a parameter that accounts for latency, and (iii) a parameter that accounts for message contention. Therefore we conclude that the approach used in the development of this model shows promise and could be used in other applications.

## References

- Blondin, JM, (1999). [VH-1 User's Guide](#), North Carolina State University.
- Collela, P., and Woodward, P.R., (1984). The Piecewise Parabolic Method (ppm) for Gas-Dynamical Simulations, *Journal Of Computational Physics*, 54 (1): 174-201, 1984.
- Dunigan, T.H., M.R. Fahey, J. B. White, and P. H. Worley, (2003). Early evaluation of the Cray X1, Proceedings of SC 2003, Phoenix, AZ.
- Gropp, W., Lusk W., and Skjellum A., (1999). Using MPI: Portable Parallel Programming with the Message-Passing Interface, 2<sup>nd</sup> edition, The MIT Press, Cambridge, MA.
- Petrini, F., D. J. Kerbyson, and S. Pakin, (2003). The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q, Proceedings of SC 2003, Phoenix, AZ.
- Parashar, M., and Hariri, S., (2000). Interpretive Performance Prediction for Parallel Application Development, *Journal of Parallel and Distributed Computing*, Vol. 60, No. 1, pp. 17 – 47, January 2000.
- Vetter, J.S., and A. Yoo, (2002). An empirical performance evaluation of scalable scientific applications, Proceedings of SC 2002, Dallas, TX.

## Acknowledgements

This work was supported by the Department of Energy's SciDAC program (Scientific Discovery through Advanced Computing). The authors gratefully acknowledge the supercomputer resources provided by the National Center for Supercomputing Applications and the Oak Ridge National Laboratory that was necessary for this work.