Parallel Performance Modeling using a Genetic Programming-based Error Correction Procedure

Kavitha Raghavachar, North Carolina State University, kraghav@ncsu.edu
Kumar Mahinthakumar, North Carolina State University, gmkumar@ncsu.edu
Patrick Worley, Oak Ridge National Laboratory, worleyph@ornl.gov
Emily Zechman, North Carolina State University, emzechma@ncsu.edu
Ranji Ranjithan, North Carolina State University, ranji@ncsu.edu

***Abstract*— Performance models of high performance computing (HPC) applications are important for several reasons. First, they provide insight to designers of HPC systems on the role of subsystems such as the processor or the network in determining application performance. Second, they allow HPC centers to more accurately target procurements to resource requirements. Third, they can be used to identify application performance bottlenecks and to provide insights about scalability issues. The suitability of a performance model, however, for a particular performance investigation is a function of both the accuracy and the cost of the model.**

A semi-empirical model previously published by the authors for an astrophysics application was shown to be inaccurate when predicting communication cost for large numbers of processors. It is hypothesized that this deficiency is due to the inability of the model to adequately capture communication contention (threshold effects) as well as other unmodeled components such as noise and I/O contention. In this paper we present a new approach to capture these unknown features to improve the predictive capabilities of the model. This approach uses a systematic model error correction procedure that uses evolutionary algorithms to find an error correction term to augment the existing model. Four variations of this procedure were investigated and all were shown to produce improved results than the original model. Successful cross-platform application of this approach showed that it adequately captures machine dependent characteristics. This approach was then successfully demonstrated for a second application, further showing its versatility.

***Index Terms*— Error correction procedure, Genetic programming, Performance modeling.**

I. INTRODUCTION

ACHIEVING high performance on parallel applications can be challenging. Performance of a parallel application depends primarily 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 processor performance, memory performance and network performance, and software metrics, including compiler performance and efficiency of communication and math libraries. Producing applications that work well across a wide range of high performance computing (HPC) platforms is a non-trivial task because the machine characteristics and their interdependence on application characteristics vary from platform to platform. The aforementioned reasons also make performance prediction a difficult task. In recent years, performance prediction for parallel architectures has attracted considerable attention, ranging from kernel benchmarking studies, application performance studies, performance modeling, and detailed comparative analysis of newer architectures (e.g., [1] [3] [15] [16] [17] [18] [19]). Other related work include automatic performance analysis [4] [10] and automatic tuning [9]. Performance prediction across platforms is increasingly important for developers and scientists to determine the performance of their specific applications on a plethora of platforms in deciding the system that best fits their needs in the long run. If scientists were provided with performance estimations for their applications on several large clusters, the process of choosing and gaining accesses to suitable platforms would be considerably simplified.

Standard modeling methodologies may include semi-empirical or empirical modeling [2] [11] [13] and use of modeling assertion tools [1] [17]. Both methodologies have advantages and caveats. In the former approach, application characteristics are captured well but do not adequately model machine dependencies such as noise, IO contention, and/or message contention. In the latter approach, machine parameters are captured well but do not adequately capture application/problem specific dependencies. In this paper, we develop and demonstrate a general approach for modeling performance of parallel applications. This approach starts by developing a semi-empirical model and then improving this further by using a genetic programming-based model error correction procedure (MECP).

The term “semi-empirical” is used to describe a parallel performance model that has both mechanistic and empirical components. The mechanistic component is a functional form that is specific to an application containing known application and machine parameters. The empirical component is represented by the set of parameters that appear in the model that will be fitted using measured data. A semi-empirical parallel performance model can be split into two parts: computation time model and communication time model. Earlier work [13] involving standard semi-

empirical modeling for an astrophysics code EVH1 (Enhanced Virginia Hydrodynamics code) showed that while the computation model prediction was satisfactory, the communication model prediction, particularly for large processor counts, was poor. This deficiency is probably due to the inability of a typical semi-empirical model to capture difficult-to-model components such as message contention and noise. The MECP procedure demonstrated in this paper seeks to address this deficiency by augmenting an existing model with an error correction term (*ect*).

The overall structure of this paper is as follows. First we describe the overall MECP approach, followed by a section that shows the application of this approach to our first application code EVH1. Then we extend this approach to a second application GYRO [6]. Then both performance models are validated across two parallel platforms. Finally, the major conclusions are summarized.

II. MODEL ERROR CORRECTION PROCEDURE

Parallel performance models typically have several components (e.g., communication contention, non-uniform memory access, I/O, etc) that are not adequately represented by terms in the performance model or not adequately captured by the model parameters. One way to address these deficiencies is to augment the original model by constructing another function (hereafter referred to as the model error correction term) that can capture these un-modeled features. In this paper, the Model Error Correction Procedure (MECP) [21] is used to construct the model error correction term while tuning the original model parameters.

Missing or misrepresented processes in the construction of a model will result in a systematic error in modeled predictions (i.e., modeled communication time). The model error correction procedure is designed to improve capabilities of mechanistic models through simultaneous tuning of parameter values and construction of an error correction term (*ect*), which is a function that represents the systematic error and appropriately adjusts the model outputs to match the observed data. The systematic error is dependent on the input conditions and the errors in the simulated outputs. Thus, the error correction term should be constructed accordingly as a function of both the model inputs and outputs to more accurately predict for new input conditions. A simultaneous search for model parameter values and the *ect* forms the basis for MECP to construct the appropriate correction of each source of error. This requires a numeric search method for the parameter estimation and a symbolic search method for *ect* determination. In this paper MECP is implemented using evolutionary computation (EC) procedures, namely genetic algorithms (GAs) and genetic programming (GP), to enable the numeric and symbolic searches, respectively.

Evolutionary algorithms (EA) are search methods that operate on a population of solutions and mimic the mechanisms of natural selection to find optimal solutions for a defined problem. In the EA-based MECP used in this paper, GP algorithmic steps are used for generating a function to represent the error correction term (*ect*) and GA algorithmic steps are used for determining the unknown or uncertain model parameter values (e.g., message bandwidth, communication latency, etc.). GA performs a numeric search by representing these parameters as a string of real, binary or integer values. GP conducts a symbolic search over a set of mathematical operators to construct a function that minimizes the error between a set of data and the output from that function. For the performance modeling application considered in this paper, the GP procedure is used to identify a function to represent *ect* using mathematical operators, including +, −, ×, /, exp, and log, that operate on a set of known input variables, such as number of processors, problem size, and number of time steps.

Figure 1 shows a generic example of how GP works on a set of data to form a function that fits the data. GP works with two sets of unknowns. One set is called the terminal set, which contains the independent variables {X}, and model constants {C}. The other set contains the mathematical operators used to construct the function. A symbolic search is performed using these two sets. GP explores a large number of possible functional relationships to identify the function that best fits the data.

A tree data structure (see example in Figure 2) is used to represent each potential solution for *ect*, which is composed of a combination of elements of the terminal set consisting of the variables and constants, and the set of operators. These potential solutions in a population undergo crossover, mutation, and selection as in any EA-based search.

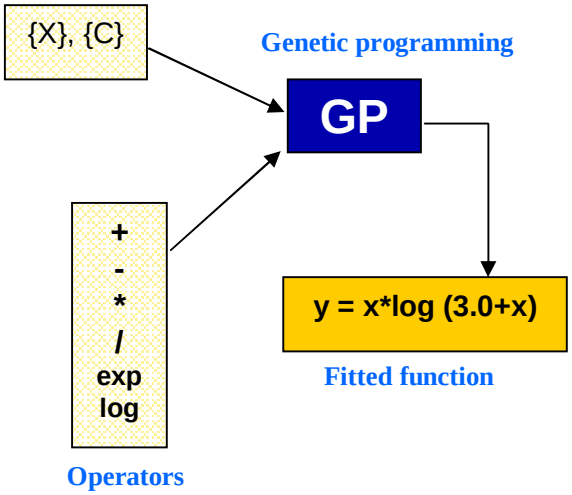

Figure 1 Example of GP procedure

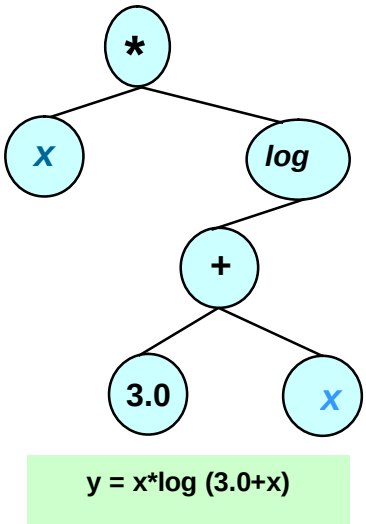

Figure 2 A sample tree representation of the function shown in Figure 1

III. General Approach and Case Studies

The general approach consists of developing a semi-empirical model and then correcting its deficiencies (if needed) by including an empirical error correction term using MECP. Figure 4 shows the flowchart of the general approach. The first step is to develop the semi-empirical model. If the semi-empirical model predictions are satisfactory, we use these models to validate the rest of the data. If the semi-empirical model predictions are not satisfactory, it is corrected using MECP to improve its predictive capability.

a. Semi-empirical modeling

The semi-empirical model has a mechanistic functional form with empirical parameters that are fitted using available data for computation and communication times. The mechanistic functional form is based on some preliminary knowledge of the application: i.e., how computation and communication time approximately scale with various application and machine parameters such as grid size, number of time steps, number of processors, etc. A set of data for developing the semi-empirical model is collected by recording the total run time and communication times for several executions of the application code using different application parameters (problem size, simulation time) and machine parameters (different processor speeds, number of processors). The model parameter values are determined using an optimization search procedure to best fit the data by minimizing the error between the observed values and the model predicted values. In this paper Levenberg-Marquadt (LM) search procedure is used for semi-empirical modeling. LM is adequate as the model function is smooth and the number of parameters to be determined is small. Lower and upper bounds were specified to constrain the parameter values within acceptable ranges. A subset of the data is used for fitting the model parameters and the remainder is used for testing the predictive effectiveness of the model. The procedure for modeling using semi-empirical approach is shown in Figure 3.

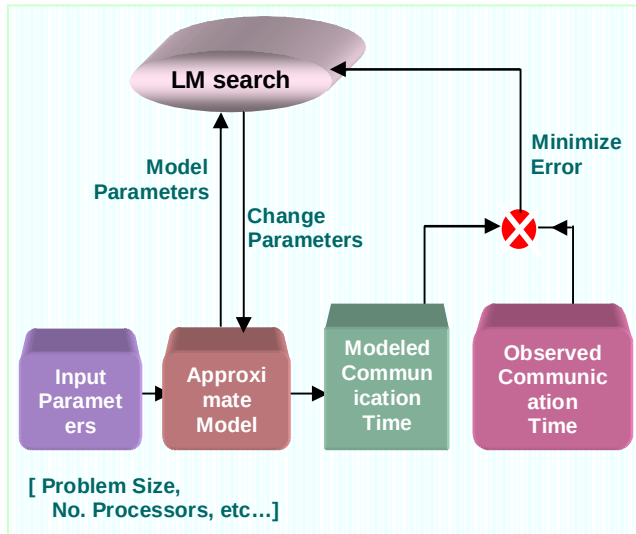

Figure 3 Semi-empirical modeling

b. Application of MECP

MECP can be applied in different ways depending on how effective the semi-empirical model is in capturing the timing behavior of an application. This in turn depends on the user's knowledge of the application and the architecture. In this paper we have investigated four variations of MECP-based models. These variations depend on whether *ect* is embedded in the model or simply appear as an additive term and whether the original parameter values of the semi-empirical model are preserved or adjusted. These cases are summarized in Table 1 and are described below. Generally, we can start with any of these four variations for a particular model. In the first two cases, the parameter values of the semi-empirical model are preserved (as determined by the LM procedure) and *ect* is generated either as a simple additive term (Case 2) or as an inclusive term (Case 1). In these cases only a symbolic search is performed (using GP) and the numeric search is excluded. For Case 1, *ect* is integrated in the model and its output replaces the modeled output. In Case 2, *ect* is determined as a term that is added to the modeled output. In the next two cases (Cases 3 and 4), the parameter values dictating the semi-empirical model are adjusted using a numeric search (using GA) and a symbolic search (using GP) to determine *ect*, which is either inclusive (Case 3) or additive (Case 4). To preserve the mechanistic characteristics of the original semi-empirical model to the extent possible, the numeric search is constrained by allowing the parameters to vary within a small range around the values determined by the LM procedure.

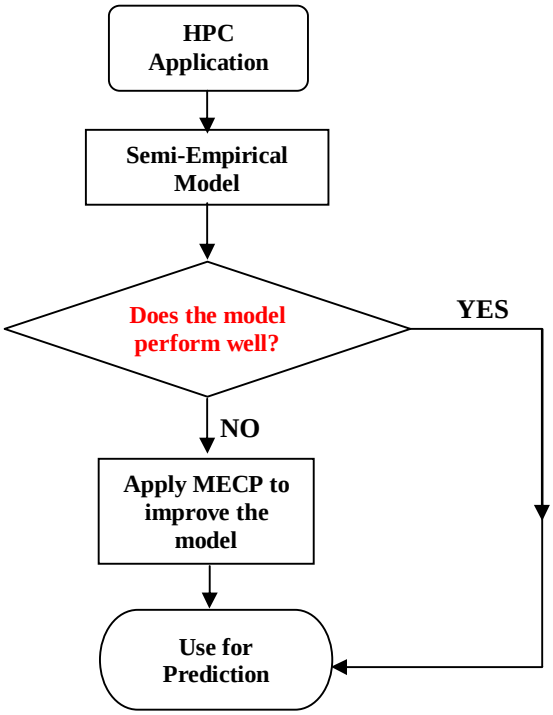

Figure 4 Flow chart of the overall approach

Cases	Improved communicate-on time	Parameter Values
Case -1	$t_{calculated} = f(t_{comm}, ect)$	Set
Case -2	$t_{calculated} = t_{comm} + ect$	Set
Case -3	$t_{calculated} = f(t_{comm}, ect)$	Search
Case -4	$t_{calculated} = t_{comm} + ect$	Search

Table 1 Summary of different cases: *ect* = error correction term, t_{comm} = original communication model, $t_{calculated} = f(t_{comm}, ect)$

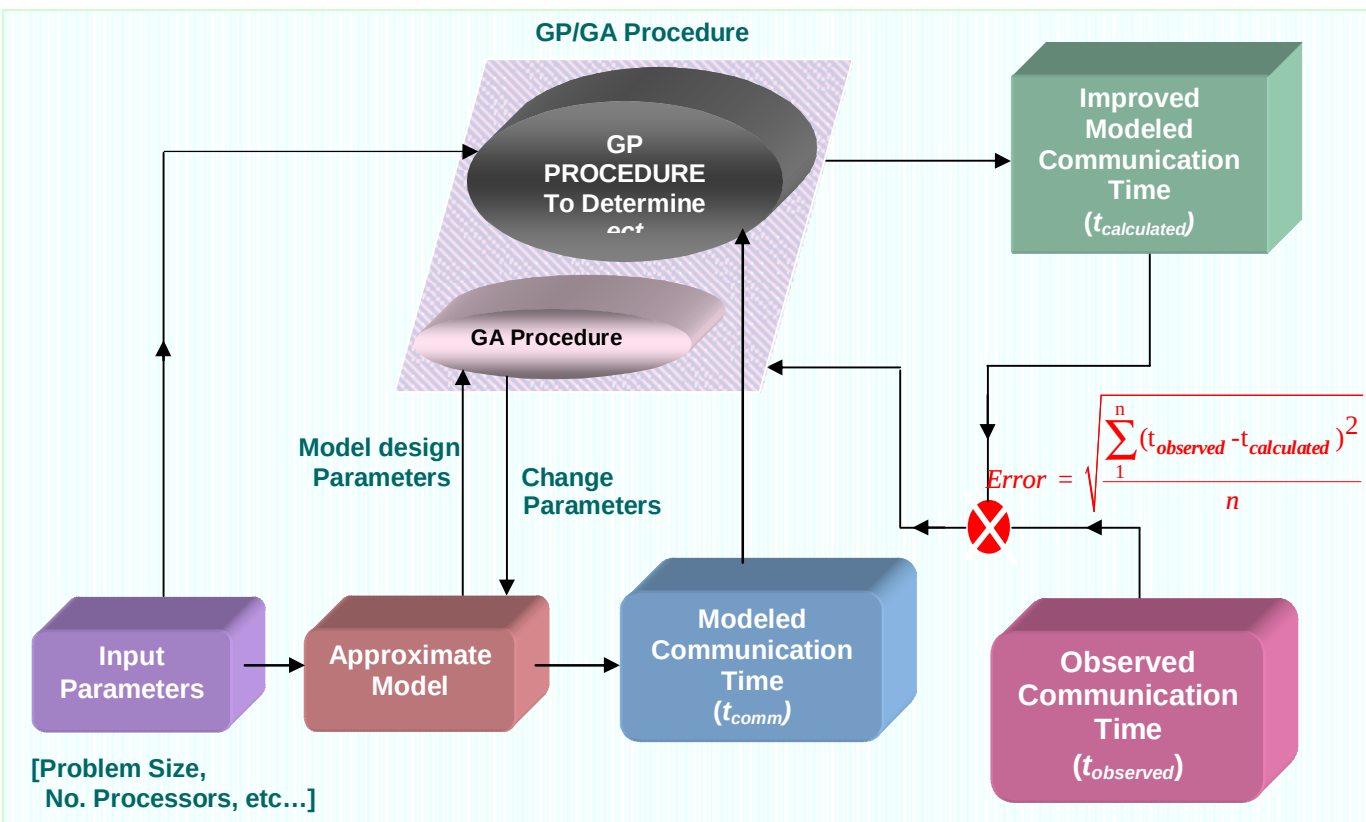

Figure 5 Model Error Correction Procedure

The error correction term *ect* is a function of the input parameters *problem size*, *number of processors*, and *number of time steps*. The *ect* function form is constructed using the set of mathematical operators +, −, ×, /, exp and log. The objective function to be minimized in the GA/GP procedure is a measure of the error between the calculated and measured communication times, which is given as:

$$Error = \frac{\sqrt{\sum (t_{observed} - t_{calculated})^2}}{n} \qquad (1)$$

In equation (1), n = number of data points, $t_{observed}$ = observed communication time, and $t_{calculated}$ = calculated communication time.

III. MECP APPLIED TO EVH1 COMMUNICATION MODEL

EVH1 (Enhanced version of Virginia Hydrodynamics) is a finite-difference astrophysics code containing over 6500 lines of Fortran and MPI (Message Passing Interface) [8]. Even though EVH1 deals with a specific application area, the algorithmic kernels used are representative of many numerically intensive finite-difference codes. Therefore modeling methodologies used in this paper have more general applicability. In earlier work [13] EVH1 runtime was modeled for 2-Dimensional (2D) and 3-Dimensional (3D) problems on two different architectures. Semi-empirical models were developed in this study involving a set of machine and application parameters for predicting computation and communication times. These model parameters were then determined using the LM procedure based on a subset of the measured data. Prediction tests performed in this paper indicated that although the semi-empirical model predicted computation times well, it was inadequate in predicting the communication times. Thus, the communication model would benefit most from the application of MECP. In this paper we restrict our attention to modeling communication time for 2D problems on a single architecture.

The original EVH1 2D semi-empirical communication model [13], which is dominated by MPIALLTOALL calls involved in a matrix transpose operation, is given as:

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

where, g, h, i and j are the constants to be fitted and f is fixed, T = stop time, nx = horizontal resolution, and np = number of processors. It is noted that the communication time includes the memory copy time (the term with the g parameter) involved in the matrix transpose operation.

Application parameters such as nx and T are known and are input to the model that is described by equation (2). All machine dependent parameters that are to be fitted are unknown. Here f represents the number of cycles proportional to the simulation time, g represents the per word memory-to-memory copy time, i represents the message latency, j represents the message bandwidth, and h represents non-linearity due to message contention. The value of f, which is obtained from the computation model, was fixed equal to 4965 [13]. In this model, the fitted machine parameters were bounded based on acceptable values for the IBM P690 architecture (see [13] for details) as shown in Table 3.

The observed communication times ($t_{observed}$) used in this process are called the "training data." The combined GA/GP procedure is depicted in Figure 5. The parameters nx,, np, T and the measured communication time are the inputs to the current model for estimating t_{comm}. GA searches for the new values for the model parameters g, h, i and j in the original model t_{comm}. GP searches over symbols and constants to construct the error correction term. These two searches in concert to provide an improved communication time model that minimizes the error.

a. EVH1 Model Results

A total of 32 data points reported in [13] are used for demonstrating the application of MECP. Half the data was used for fitting and the remaining half was used for testing the prediction accuracy.

Table 2 summarizes the GA/GP parameters and settings used in MECP. As GA and GP are based on probabilistic operators, the robustness of the procedure was first tested for 30 random trials for each of the four cases (Cases 1, 2, 3 and 4 (see Table 1)). The procedure did not converge in four out of the 120 trials, and they were excluded from these calculations. These results are based on a validation data set consisting of 16 data points that were used for testing the predictive capability. For Cases 1 and 2 the fitted parameter values reported in Table 3 were obtained by the LM procedure. For Cases 3 and 4 the model parameter values fitted by MECP falls well within the range of the bounded values.

A comparison of the prediction errors for the original model and the best models obtained (out of the random trials) for the four cases are shown in Figure 6. This figure shows that the best solutions for all four cases outperform the original model. Also, Case 1 yields the best model with an error less than half of that of the original model. The frequency of each of the cases performing better than the original model in the random trials is

compared in Figure 7. This gives an estimate of the robustness of each approach. This shows that Cases 1, 2, 3 and 4 outperform the original model in 77%, 73%, 50% and 90%, respectively, of the trials, indicating that Case 4 is the most robust.

The predictive capability of the best model found for each case is shown in Figure 8. This figure compares for the 16 validation data points the predicted values from the original model and the new models. The observed values are also shown in this figure. Note that the timings (y-axis) are in log scale. The data points on the x-axis represent different problem instances (i.e., different sizes or different number of time steps) and different processor counts. Generally, the larger data point numbers (>4) correspond to larger problem sizes or larger processor counts, hence representing larger communication times. Figure 8 shows that while the original model prediction is better for smaller data point numbers (corresponding to smaller communication times), models obtained using MECP perform better for larger data point numbers (i.e., for larger communication times). From a modeling point of view, a model that performs well for larger communication times is more valuable as these times contribute more to the overall time. Therefore we conclude that the models obtained using MECP are more useful.

We also tested a fifth variation where we exclude the original model and develop a fully empirical model using a symbolic search. Even though this search produced some good results, multiple trials (recall that our symbolic search uses GP, a probabilistic method) using different starting random seeds indicated that this approach is not robust. For instance, out of 50 random trials, this variation produced only 3 functions that yielded better predictions than the original semi-empirical model. Hence we conclude that MECP is best applied when an approximate performance model already exists or can be developed.

Parameters	Settings
Function set	+, - , x, /, ^ , log, exp
Terminal set	*R, nx, np, T, tcomm*
Population Size	3000
Max. No. Generations	100
Crossover	90 %
Mutation	10 %
Selection Strategy	Graduated Elitist
Max. Initial depth	7

Table 2 GA/GP parameter settings for MECP

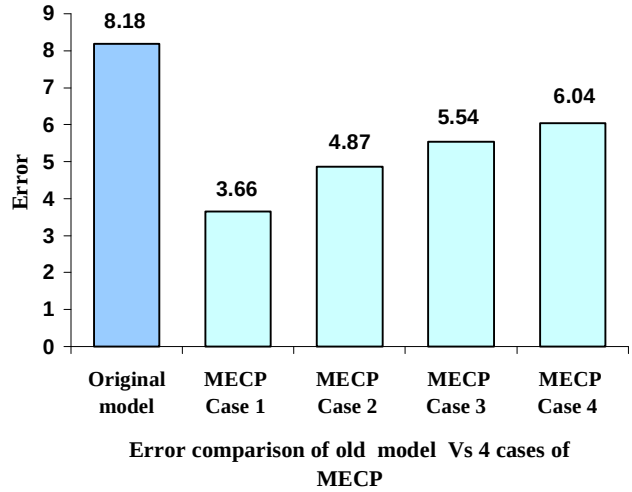

Figure 6 Prediction errors for the original model and the four cases of MECP

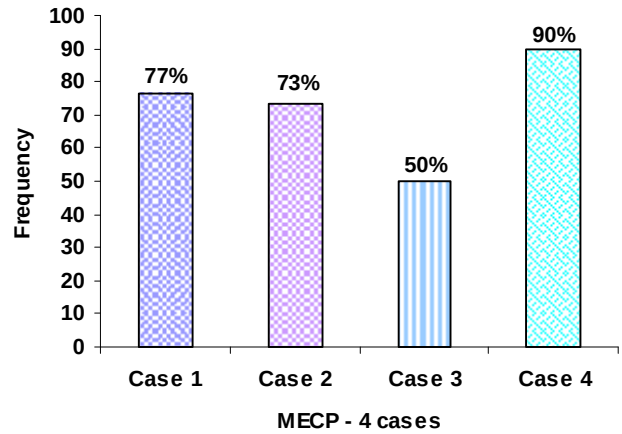

Figure 7 Frequency of predictions showing improvement over the original model in the random trials

Fitted Parameters	Lower bound	Upper bound	Fitted Value
g	1.00E-09	1.00E-03	2.45E-08
h	1	1.5	1.13216
i	1.00E-09	1.00E-03	1.69E-05
j	1.00E-09	1.00E-03	1.62E-07

Table 3 Summary of fitted values for model parameters for EVH1

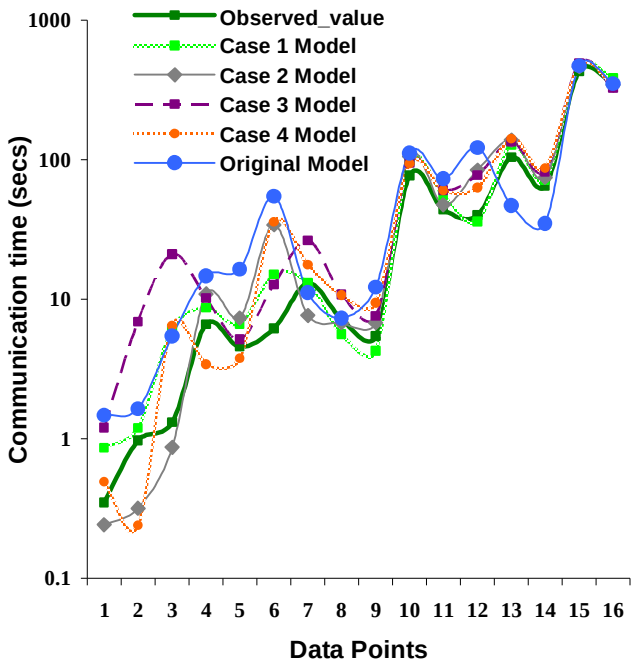

Figure 8 Comparison of prediction efficiency of EVH1 communication timing models

IV. EXTENSION OF MECP TO SECOND APPLICATION GYRO

GYRO [6], an Eulerian gyrokinetic-Maxwell solver, computes numerical simulation of tokamak micro-turbulence by solving time-dependent, nonlinear equations. GYRO comes with three standard test problems B1-std, B2-cy, and B3-gtc. A total of 24 data points were generated for these three problems using various processor counts. Of these 24 data points, points 1-8 correspond to B1-std with processor count increasing from 16 to 512, points 9-16 correspond to B2-cy with processor count increasing from 16 to 512, and data points 17-24 correspond to B3-gtc with processor count increasing from 64 to 512. The communication time was obtained using the mpiP profiling tool [14] that gives the MPI communication time as a percentage of total runtime.

a. Computation Timing Model

The semi-empirical model for computation time for GYRO application is developed by considering the input parameters that contribute significantly to the computation time (grid dimensions, time steps, etc.). The model for computation time is expressed as:

$$t_{comp} = \frac{1}{np} \cdot ts \cdot \left(Tg^{a} \cdot rg^{b} \cdot tc \right) \cdot k \qquad (3)$$

In the above equation, a, b *and* tc are constants to be fitted, k is a known constant, np = number of processors, ts = time step, Tg = toroidal grid, rg = radial grid, k = 1 for the B1-std and B2-cy problems, and k = 2 for the B3-gtc problem. The three parameters a, b, and tc were estimated using a single measured computation time from each problem (B1-std, B2-cy, and B3-gtc). Thus, there were three unknowns, three equations, and three data points. The three unknown parameters were readily solved using the standard nonlinear algebraic solver, *fsolve*, in MATLAB. The model was then validated for the remaining data points. The model results shown in Figure 9 indicate that the semi-empirical model described by equation (3) has excellent predictive capabilities for computation time and MECP is not needed (see flowchart in Figure 3).

b. Communication Timing Model

The semi-empirical communication timing model is developed by considering both application characteristics (the number and size of the messages) and machine characteristics (bandwidth, latency, number of processors). It is based on the following simple expression:

$$tcomm = no.cycles \times \left[\left(\frac{Message_size}{Bandwidth} \right) + Latency \right] \times no.processors^{e} \qquad (4)$$

where no. of cycles represents the total number of messages and e is an exponent that accounts for message contention. Equation (4) could be used in an expanded form as follows:

$$t_{comm} = k \cdot ob \cdot np^{e} \cdot nt \cdot \left\{ \left[\begin{pmatrix} rg \cdot ((tg + pg) \cdot eg) \cdot Tg \\ \cdot \left(\frac{1}{np} \right) \cdot \left(\frac{1}{bw} \right) \end{pmatrix} + tl \right] \right\} \qquad (5)$$

where e is a constant to be fitted, tl = latency, bw = bandwidth, np = number of processors, ob = orbit grid, ts = time step, tg = trap grid, Tg = toroidal grid, pg = pass grid, eg = energy grid, rg = radial grid, and k = a known constant that could be different for each of the three problems. In equation (5), e, bw, and tl are fitted and the other parameters are known inputs. Table 4 shows the bounds and the fitted parameter values for this model. Again, the model parameter values were obtained by using the Levenberg-Marquadt procedure.

c. Model Results

Out of the 24 data points, the odd 12 were used for fitting and the even 12 were used for prediction. Figure 10 shows the model performance for fitting and Figure 11 shows its performance for prediction. The semi-empirical model, which is also called as the original model in the graph, fitted and predicted well for B1-std and B2-cy problems. For the B3-gtc problem, however, the model failed in both fitting and prediction. Further investigation using mpiP profiling indicated that the semi-empirical model does not correctly capture message size for the B3-gtc problem. It was assumed that the message size is approximately a scalar multiple of the product of the grid sizes divided by the number of processors. While this is true for the B1-std and B2-cy problems, it is not correct for the B3-gtc problem. Message sizes for the B3-gtc problem were significantly larger as it uses the FFT method for solution as opposed to the default direct method. B3-gtc problem has nearly three-times larger message size than that assumed by the semi-empirical model.

Since we were interested in a single model that predicts well for all three problems of GYRO, we applied MECP to improve the prediction performance of the semi-empirical model. As in EVH1, we tested all 4 cases (see Table 1) of MECP for the GYRO communication model. All four cases performed nearly equally well (Figures 10 and 11). Thirty random trials were conducted for each case. Out of these 30 trails, in terms of frequency, Case 1 showed improved results than the original model 93% of the time, Case 2 83%, Case 3 90% and Case 4 80%. The best model out of 30 trials for each case and the frequency of improved prediction for each case is shown in Figure 12. In terms of error reduction, when compared to the semi-empirical model, Case 1 best model showed 81% reduction in error, Case 2 80%, Case 3 78% and Case 4 75% error reduction. This indicates that for GYRO, Case 1 is superior even though other cases are not far behind.

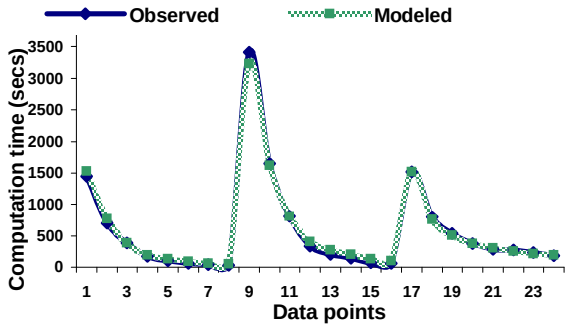

Figure 9 Gyro computation time model for B1-std, B2-cy and B3-gtc problems

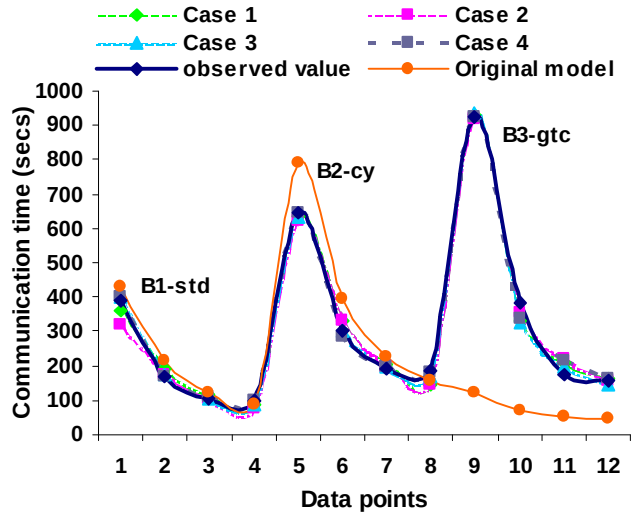

Figure 10 Gyro communication time model for B1-std, B2-cy and B3-gtc problems (fitting)

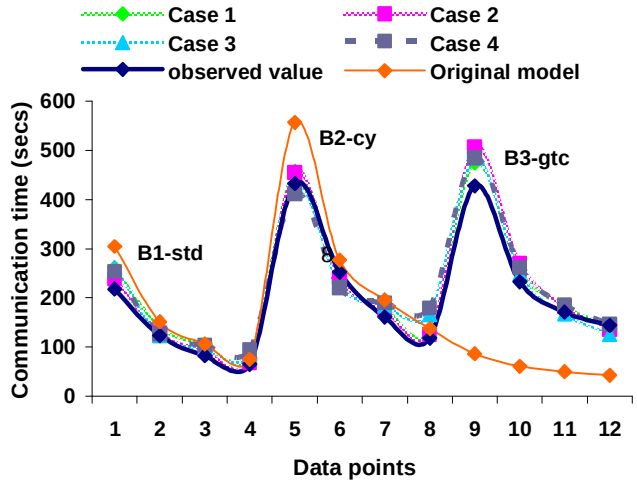

Figure 11 Gyro communication time model for B1-std, B2-cy and B3-gtc problems (prediction)

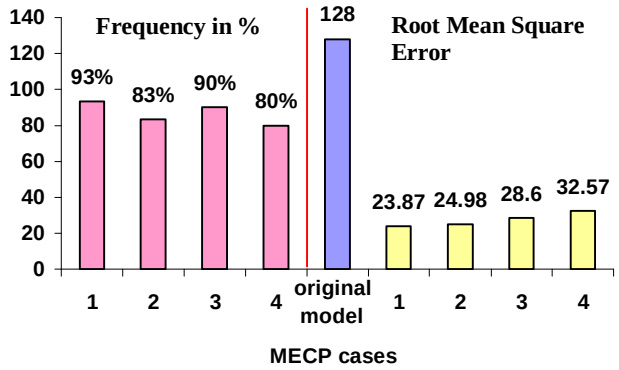

Figure 12 Comparison of Semi-empirical model with four cases of MECP model for GYRO: Frequency of trials performing better than the original model and best prediction error

Fitted Parameters	Lower bound	Upper bound	Fitted Value
ex	1.00E-02	5.00E-01	4.96E-01
tl	1.00E-08	1.00E-02	1.12E-08
bw	1.00E+03	1.00E+06	9.85E+05

Table 4 Summary of fitted values for model parameters for GYRO

V. CROSS PLATFORM PREDICTION

In the preceding sections the MECP models were developed and validated using data from a single architecture (IBM P690). The question remains whether these same models could be applied to a different architecture simply by fitting or adjusting only the machine parameters (i.e., latency, bandwidth, etc.) A test of this hypothesis would indicate how well our previously developed models delineated the application specific parameters from machine specific ones. The Teragrid Linux cluster at NCSA was chosen for this test as the timing data for both applications were already available for this architecture [20][13]. This cluster consists of 1024 Itanium II processors connected by a Myrinet network. We could predict the performance of our models either by fitting the machine parameters or by using the published MPI latency and bandwidth values for the Myrinet network. In this paper we demonstrate the use of both approaches. For EVH1, we used the IBM P690 model and only the machine parameters were fitted (using the Teragrid data), instead of using published values for the Teragrid architecture. As shown in Figure 13, Case 2 and Case 4 models for EVH1 predict well for the Teragrid architecture, while Case 1 and Case 3 predictions are not as satisfactory. This implies that keeping the base model separate from *ect* (as in Cases 2 and 4) yields a better model for cross platform prediction. This is expected as it would be difficult to delineate machine characteristics from application characteristics in a combined model (Cases 1 and 3). It is also noted that all cases under-predict data point 1 that corresponds to $np = 1$, which is dominated by the memory-to-memory copy operation. This means that the fitted parameter value g is too small for the Teragrid architecture. This is expected since some data points have a strong dependence on this parameter (the first two or three points).

The above tests show that MECP approach could be effectively used to predict performance of an application on a completely different architecture by simply changing the appropriate machine values. The main caveat for using this approach is that unless existing data is available for fitting, accurate values for machine parameters, such as latency and bandwidth, memory-to-memory copy time, etc., can be difficult to obtain.

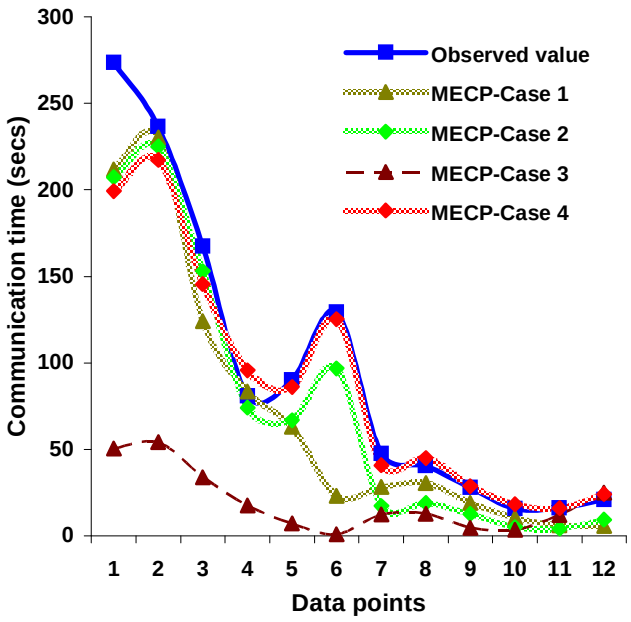

Figure 13 Cross platform prediction for Teragrid using IBM P690 MECP-based models for EVH1 application

For the GYRO application, the MECP-based model was developed in the previous section using the data from the Teragrid architecture. The IBM P690 architecture was chosen for testing the cross platform prediction. Unlike EVH1, here we applied the average MPI published *tl* and *bw* values for IBM P690 to the MECP-based GYRO models. As shown in Figure 14, all four models predict well for the IBM P60 architecture, indicating the versatility of these models for cross-platform application.

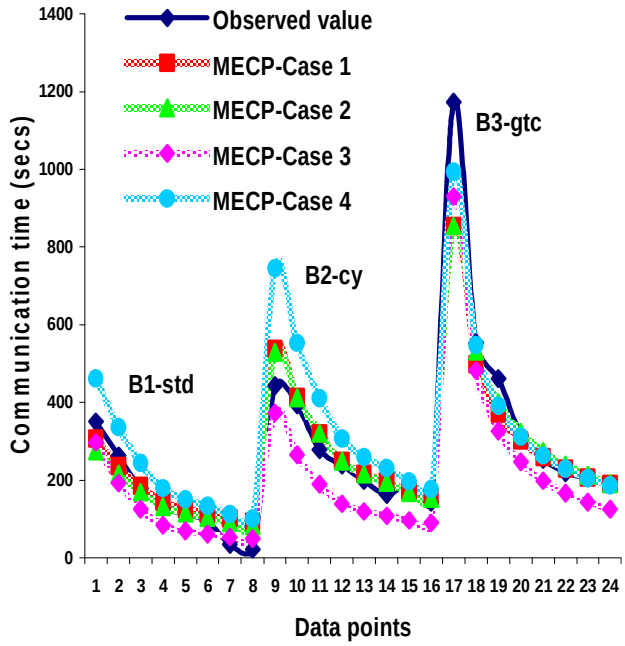

Figure 14 Cross platform prediction for IBM P690 using Teragrid MECP models for the GYRO application

VI. CONCLUSIONS

The following points summarize the results and conclusions from this paper:

- § We have developed a new 'GA/GP Model Error Correction Procedure' approach for modeling performance of parallel applications and demonstrated if through two HPC applications across two platforms.
- § Four variations of this approach were investigated and all variations showed improved results than a standard semi-empirical approach.
- § The MECP approach captures difficult to model features such as noise, message contention, or I/O contention thus providing improved results than a standard semi-empirical approach.
- § This approach augments an existing semi-empirical model either by adding an error correction term (Cases 2 and 4) or by including the error correction term in the function (Cases 1 and 3).
- § Overall, Case 1 performed reasonably well for both applications across both platforms even though in several specific instances other models performed better. This suggests that an integrated model that uses parameter values from the original model will work well most of the time.
- § While the semi-empirical component of the model contains certain specificities to the target application, the approach used to obtain the error correction term of the model is generic and could be easily extended to model performance of other parallel applications.

Therefore we conclude that the MECP approach shows promise for wider applicability in parallel performance modeling.

VII. REFERENCES

[1] Alam, S. R., and Vetter, J. S. A framework to develop symbolic performance models of parallel applications, *Proc. 5th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 2006).*

[2] Almasi, G.S., Cascaval, C. et al., . Demonstrating the scalability of a molecular dynamics application on a petaflop computer, Proc. Of International conference on supercomputing, 2001, pp. 393-406.

[3] Bailey, D. H., and Snavely, A. S. Performance Modeling: Understanding the present and predicting the future, *Proceedings of SIAM PP04, 2005.*

[4] Calzorossa, M., Massari,, L. and Tessera, D. . A methodology towards automatic performance analysis of parallel applications, *Parallel Computing*, 30, 211-223.

[5] Dunigan, T. H., Fahey, M. R., White, J. B. and Worley, P. H., Early evaluation of the Cray X1, Proceedings of SC 2003, Phoenix, AZ.

[6] Fahey, M. and Candy, J. GYRO: A 5-d gyrokinetic-maxwell solver. In *Proceedings of Supercomputing*, 2004.

[7] Genetic Programming for Subjective Fitness Function Identification. *Springer Link-Volume 3003/2004. Genetic Programming:*

[8] Gropp, W., Lusk W., and Skjellum. A., (1999). Using MPI: Portable Parallel Programming with the Message-Passing Interface, *2nd 7th European Conference, EuroGP 2004, Coimbra, Portugal, April 5-7, 2004. Proceedings.edition, The MIT Press, Cambridge, MA.*

[9] Hunold, S., and Rauber, T. Automatic tuning of PDGEMM towards optimal performance, *Euro-Par (2005). 22055, LNCS 3648, pp. 837-846.*

[10] Jorba, J., Margalef, T. and Luque, E. Automatic performance analysis of message passing applications using the KappaPI 2 tool, *EuroPVM/MPI 2005, LNCS 3666, pp. 293-300.*

[11] Kerbyson, D. J., Henry, A. J., Hoisie, A. Petrini, F., Harvey, W. J., Gittings, M. Predictive performac nce and scalability modeling of a large-scale application. *Proceedings of Int'l Conf. Supercomputing, 2001.*

[12] Koza, J. R.: Genetic Programming II. *The MIT Press, Cambridge, Massachusetts (1994)*

[13] Mahinthakumar, G., Sayeed, M., Blondin, J., Worley, P., Mezzacappa, A. and Hix, R. Performance Evaluation and Modeling of a Parallel Astrophysics Application, *Proceedings of the High Performance Computing Symposium 2004*, p. 27-33, *Ed: Joerg Meyer, The Society for Modeling and Simulation International, ISBN 1-56555-278-4, Arlington, VA*

[14] mpiP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/.

[15] Parashar, M., and Hariri, S.. Interpretive Performance Prediction for Parallel Application Development, *Journal of Parallel and Distributed Computing, Vol. 60, No. 1, pp. 17 – 47, January 2000.*

[16] Petrini, F., Kerbyson,, D. J. and Pakin, S. . The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q, *Proceedings of SC 2003, Phoenix, AZ.*

[17] Snavely, A, Carrington, L., Wolter, N., Labarta, J., Badia, R. and Purkayastha, A., A framework for

application performance modeling and prediction, *SC2002, Baltimore, MD.*

[18] Yang, T., Ma, X. and Mueller, F. Predicting Parallel Applications' Performance Across Platforms Using Partial Execution, *Supercomputing 2005*.

[19] Vetter, J.S., and Yoo, A., An empirical performance evaluation of scalable scientific applications, *Proceedings of SC 2002, Dallas, TX*.

[20] Worley, P. Candy, J. Carrington, L., Huck, K., Kaiser, T., Mahinthakumar G., Maloney, A., Moore, S., Reed, D., Roth, P., Shan, H., Shende, S., Snavely, A., Sreepathi, S., Wolf, F., and Zhang, . . Performance Analysis of GYRO: A Tool Evaluation, in the *Proceedings of the 2005 SciDAC Conference, San Francisco, CA, June 26-30, 2005*

[21] Zechman, E. M. (2005). Improving Predictability of Simulation Models using Evolutionary Computation-Based Methods for Model Error Correction, *Ph.D. dissertation, North Carolina State University, Raleigh, NC.*

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.