Performance Prediction of Weather Forecasting Software on Multicore Systems

Javier Delgado, S. Masoud Sadjadi, Marlon Bright, Malek Adjouadi
College of Engineering and Computing
Florida International University
Miami, USA
{javier.delgado, sadjadi, marlon.bright, adjouadi}@fiu.edu

Hector A. Duran-Limon
Center for Economic Administrative Sciences
University of Guadalajara
Guadalajara, Mexico
hduran@cucea.udg.mx

***Abstract*—Performance prediction is valuable in different areas of computing. The popularity of lease-based access to high performance computing resources particularly benefits from accurate performance prediction. Most contemporary processors are employing multiple computing cores, which complicates the task of performance prediction. In this paper, we describe the methodology employed for predicting the performance of a popular weather forecasting application on systems with between 4 and 256 processors. An average prediction error of less than 10% was achieved after testing on three different multi-node, multicore systems.**

Keywords: performance prediction, regression analysis, application modeling, WRF, multicore

I. INTRODUCTION

The topic of predicting execution times of long-running scientific applications with a reasonable degree of accuracy is useful for satisfying the requirements of time-critical applications, maximizing computing center processor utilization, improving quality-of-service, and providing intelligent scheduling. The latter is applicable for high-performance computing (HPC) applications, which are usually performed on computing resources that are "leased" in HPC datacenters. Performance prediction of applications such as hurricane prediction and medical diagnosis is of increased importance, since results need to be obtained with sufficient time remaining to act accordingly. In this paper, we demonstrate our methodology for predicting the performance of weather simulations performed using the weather research and forecasting (WRF) [1] software.

Several methodologies have been published in the literature to address the performance prediction problem. The methodologies rely on different techniques (e.g. statistics, expert systems, and data mining). Our methodology includes the use of a mathematical model which uses regression analysis to predict execution time. The user hypothesizes what parameters are significant contributors to execution time and applies them to the model. The ability of the parameters to accurately predict execution time is then tested empirically.

WRF has been shown to model weather-related catastrophes with good accuracy. The impact of these catastrophes on human life and on the economy, combined with the complexity of WRF itself, warrants focusing exclusively on it for this work. WRF simulations demand a large amount of computational power if practical (i.e. accurate and high-resolution) simulations are desired. Since the window of time between the detection of a hurricane and its arrival is relatively small, simulations need to execute as rapidly as possible. For example, a 4-km resolution simulation on an area of 300x300km for 24 hours requires approximately 500 billion floating-point operations, depending on different factors (e.g. simulation parameters, state of the atmosphere, and geographical properties of the simulation). Thus, a large number of processors are necessary for the simulation. When setting a job execution file in an HPC data center, the amount of resources and the time they are needed for must be set a-priori. Large requests can result in queue times of several days. If the resources requested are insufficient, the job will not complete. Performance prediction addresses the problem of determining the resources needed to execute the jobs within a given time frame. Due to the time-sensitive nature of this application, we set the constraint of at most 10% error of execution time prediction.

The model was validated by carrying out experiments on three parallel systems. The tests consist of generating data points by executing multiple simulations on the systems and using our prediction model to predict the execution times. Prediction accuracy tests were first performed individually on each of the systems and later cross-cluster prediction tests were performed to test the model's ability to extrapolate.

Our previous work [2] demonstrated our model's ability to accurately predict execution time on two different clusters, 8 and 16 nodes in size. In that work, two parameters were varied, clock speed and parallelism. This paper demonstrates two critical aspects of our prediction methodology in the context of WRF that were not discussed in our previous work - *scalability* to a large number of nodes and *applicability* to multicore architectures. We also show its ability to work on different platforms, including POWER and newer Intel processors. We describe how we have refined our model in order to achieve our accuracy goals in the presence of more nodes, with multiple cores, and/or with different architectures. The rest of the paper is organized as follows. Section II describes our application modeling approach and the challenges faced with the systems we worked on. Section III

describes the software tools used for the experiments. Section IV discusses the experiments performed and the observations made from the experiments. Section V describes the model validation. Section VI provides related work.

II. Application Modeling

A. Overview of Our Approach

The methodology used can be described as a hybrid approach to execution time prediction, in the sense that the prediction model itself has no application- or domain-specific knowledge, but users may add this knowledge after determining the factors that affect performance (i.e. the model itself is oblivious to the application, but human knowledge about the application improves the model's accuracy). Some existing approaches to performance prediction have general and/or specific knowledge about application execution included in the prediction paradigm itself. A possible problem with these approaches is that they can be difficult to deploy; for example, some of these tools require the application used to be compiled with special tracing libraries. The tools themselves may also require intimate knowledge of the target execution platform, and more. Conversely, approaches that are entirely oblivious to the application generally suffer worse prediction accuracy [3]. As we will show, we do not try to tailor our model to any specific application. Instead, we use knowledge of the application and execution platform to improve the model. We use WRF as a proof-of-concept. Our approach has proved successful for different application domains [4], however, this paper focuses only on the WRF.

Fig. 1 provides a depiction of our modeling approach. The cycle is iterative and incremental and starts in Stage *A (Application/Code/Platform inspection)*. In Stage *A,* specific details about the application and/or execution platform are studied. The purpose of this step is to determine what parameters affect the execution time of the application. An example of a question that this step can answer is how the CPU clock speed of the execution platform affects the execution time. The depth of knowledge required for this step depends on the application. For example, some applications are I/O-bound, and increasing the CPU clock speed will not provide any performance improvement.

We begin the parameter evaluation stage with the most obvious parameters, such as CPU, number-of-nodes, and cores-per-node. WRF is computationally intensive due to the large amounts of calculations involved in weather simulations, so we researched effective ways of modeling the performance of the CPU. WRF is also communication-intensive, with several halo exchanges and collective communications required throughout the simulation; as a result, parameterization and modeling of the communication systems of the execution platform was performed as well. We started this work by using the *parallelism* parameter, which provided accurate predictions in our previous work when combined with *clock speed*. We then refined the model as necessary for accurately modeling multicore systems and for extrapolating between different CPU architectures/platforms (e.g. Intel Xeon to POWER 970MP).

In Stage *B*, the application is modeled based on what is learned about the application in Stage *A*. The mathematical model is described in detail in [2]. Basically, it assumes the value of a parameter (e.g. execution time) can be modeled as a product of the parameters that affect it (e.g. CPU and parallelism). The parameters themselves may be polynomial equations of arbitrary length, which results in Equation (1), in which m is the number of parameters, m_i is the maximum polynomial degree of the current parameter, a_{ij} is the coefficient contribution of the ith parameter, and z_i^j is ith parameter. Until now, we have had success with a simplified model in which the maximum polynomial degree of all parameters is equal to one (i.e. first-order polynomials).

$$T_{exec} = \prod_{i=0}^{m-1} \sum_{j=0}^{m_i} a_{ij} z_i^j \quad (1)$$

Based on this assumption, the model attempts to determine the contributions of each of these parameters (i.e. the a_{ij} values). The resource properties are all combined to form a sum-of-products, plus an error term to account for model inaccuracies and absent parameters. Regression analysis is used to determine the values of the coefficients.

In Stage *C*, we perform executions under different conditions and/or with different runtime configurations. We define a *runtime configuration* as the number of nodes and processes per node used for any single execution on a particular system. When we refer to a *data set* or *data series*, we are referring to a collection of execution statistics for a single instance of all possible runtime configurations (e.g. the execution times for all runs performed at a particular time with 8, 16, 32, and 64 nodes). When all executions are finished, the acquired data is fed into the prediction model, which estimates the contribution of each parameter (Stage *D*). Based on these individual estimates, the total execution time is estimated for a target execution platform, and compared to the actual execution time. This process is continued (another iteration of the *A-B-C-D* cycle) until all execution time predictions are within 10% error. The time it takes to iterate through the cycle depends on the data being collected, but the tools were designed to provide fast results and use regular text files so that data can be easily added or removed using common text-processing tools.

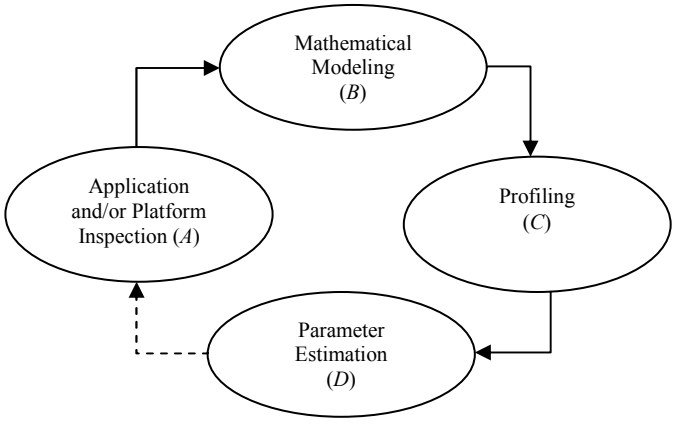


Figure 1. Overview of the performance prediction methodology.

B. Challenges with Large-scale Systems

Our previous work showed the efficacy of our approach on relatively small systems with at most 16 compute nodes. Also, the user base of the systems was limited, so the test systems were well-controlled. Large clusters (or *supercomputers*) usually have hundreds of users and their large size introduces platform-related issues for performance prediction that application modeling is not able to overcome.

One challenge for performance prediction is that execution times may vary significantly from execution to execution, despite using the same type of nodes, application, and input data. This could be caused by any "jitter" in the computing infrastructure, differences in system utilization, resource allocation, etc. Also, different nodes may have different performance characteristics, depending on the strictness of the manufacturing tolerance of their parts, their current physical condition, their usage, etc. This can have a significant effect on execution time. For parallel jobs, the interconnection of the assigned nodes may be different for each job. For parallel applications that perform a lot of intra-node communication, different interconnection configurations have a significant effect on execution time.

Another challenge with large systems is modeling speedup. Applications that scale well with up to 16 nodes may not scale well with 64 or more nodes. It is difficult to predict what the execution pattern of an application will be like without knowing about its design. Even with some knowledge about its design, it is not trivial to know how it will scale when run on a large number of nodes, particularly for communication-intensive programs. Also, while it may scale well on one system, it may not on another. This could be caused by the system's network speed, for example. Since contemporary clusters and supercomputers consist of more than 16 nodes, it was necessary to ensure our model works with these larger systems.

C. Challenges with Multicore Architectures

Multicore architectures have become commonplace for all types of computing systems. Single-processor systems have become rare even on ordinary workstations, due to physical limitations of how much higher a processor's clock speed can be. Having multiple cores further complicates performance prediction. However, given their popularity for current and future systems, it is necessary to be able to accurately predict execution time on multicore systems.

To determine the optimal parameters to introduce to the model in order to model execution on multicore systems, a closer look into multicore architectures is necessary (Stage *A*). For parallel jobs in which only one core of each node is used and the system specifications are kept constant, speedup is affected by interconnection network performance and the application's parallelization ability. The latter is a combination of computational redundancy, synchronization requirements, etc. When multiple cores are used, several complications arise. For one, intra-node communication may take place. Since the bandwidth and latency for messages passed inside a processor/bus between processing cores is different from that of different nodes communicating through the network, multiple communication factors need to be modeled. Furthermore, the cores need to share system components, such as cache, main memory, network cards, etc. This introduces the possibility of contention occurring when accessing different hardware components, leaving less effective capacity for each core. For example, one of the systems used consisted of nodes with two dual-core IBM 970MP [5] processors. Each core has its own dedicated L2 cache of 1MB, but the arbitration logic is shared. Measuring contention is not possible, which makes modeling difficult. The performance counters of CPUs can be utilized to measure statistics, such as cache misses, which can reveal the effects of sharing components, but the information they provide about what is happening is still limited. Also, it is necessary to use special tools to record the statistics generated by these counters.

Knowledge about the application is helpful to determine what parameters to model. For example, it has been shown that WRF is memory-bandwidth and latency bound [6][7], so the model needs to account for memory bandwidth in order to provide accurate predictions. Memory bandwidth was introduced to the model as one of the parallelism parameters. The recorded value is modified depending on the number of processes used for the execution, since it is shared by the processors.

The behavior of a multicore node itself is generally consistent as long as whatever it is executing is constant, so an approach that relies on previous execution data is able to cope with the fact that sharing components amongst cores leads to non-trivial execution patterns. However, when combining several multicore nodes, prediction is more difficult due to the fact that some processors are in the same node while others are connected externally through the network. As a result, it is necessary to give the model separate parameters for number-of-nodes and cores-per-node. For even greater accuracy, memory bus and interconnection network bandwidth should also be included in the model.

D. Challenges Predicting Among Different Architectures

In our previous work, our model turned out to be accurate enough when performing cross-cluster predictions, despite a slight difference in CPU architecture between the test systems. In those experiments, a Pentium 4 based cluster and a 64-bit Xeon based cluster were used as test beds. Despite this difference, clock speed alone was able to accurately model performance, probably due to the fact that the processors in the test systems were designed at similar points in time, so their performance characteristics were similar. Also, both systems were using a 32-bit version of WRF, so the 64-bit system was not being fully exploited.

However, advances in CPU technology and power requirements will cause this relation between clock speed and execution time to change, so clock speed is not enough to model CPU performance. To understand why a given processor is faster than another requires in-depth knowledge about its design. There are several factors that contribute to the speed at which a processor executes a given set of instructions, besides its clock speed, including pipelining, instructions/cycle, efficiency of internal components, etc. For example,

manufacturers have realized that processors need to be more energy efficient, so fewer cycles are being used for the same tasks on newer processors. An approach that models the CPU would provide accurate results. Such an approach was done on much older architectures such as the TI ASC and CRAY-1 [8], but this approach may not work on modern processors. Due to the complexity of this task, which lies outside the scope of our work, it was necessary to use a simpler approach that still gives acceptable accuracy[1].

An alternative to low-level modeling of the CPU is to find metrics that correlate well with execution time. The metrics will change depending on the application. For example, in [7] the authors ranked several metrics and found that WRF execution time correlated best with strided access to main memory, while for most other applications random access to L1 cache had better correlation. In [6], the authors found node bandwidth and latency to be the most significant parameters for the scalability of WRF. To properly evaluate the metrics with the highest contribution, it is necessary to measure several of them. Special tools may be required for some of these measurements, so we attempt to only use metrics that are easy to acquire.

Benchmarking is another alternative that can give a good indication of CPU performance for different applications, without requiring special tools to extract any metrics. The caveat with benchmarking is that, for best results, the benchmark needs to be representative of the application being modeled, which requires some knowledge of the application. The best benchmark for a particular application is the application itself. However, benchmarking every application that is to run on a given system may not be possible. A good tradeoff is to run a few benchmarks, each one representative of the behavior of several applications. Each application that is to be run on the system can be mapped to a particular benchmark. For example, WRF calculations include differential equations and finite difference approximations, so a benchmark application that consists of these kinds of calculations should provide acceptable accuracy.

Even if using the application itself as a benchmark, it may be necessary to make tradeoffs for practicality. One obvious tradeoff is the execution time of the benchmark. For example, a WRF benchmark consisting of a simulation over the entire USA at high resolution is very time consuming. On the other hand, a simulation with a small input should be feasible while still capturing the application's characteristics well.

We develop a formula that is a function of clock speed and a constant, the *platform contribution*, to model the CPU parameter. The *platform contribution* is determined by benchmarking. It only needs to be calculated when performing predictions among systems with very different CPU architectures. Our previous results showed that clock speed is a good indicator of the performance of systems with the same or similar architectures, so it is not necessary to measure separate *platform contribution* parameters for systems with similar CPUs but different clock speeds.

[1] If it becomes apparent during the validation stage that we need to consider CPU modeling to obtain accurate predictions, it will be done.

The underlying software environment needs to be considered as well. The operating system and installed libraries affect the execution of a program; the compiler and compiler optimizations used to build the application also contribute to the application's performance. For the purpose of this work, we assume that the compilers used for benchmarking are the same as those used for the execution of the program whose performance is being predicted.

E. The Refined Prediction Model

Accounting for all the challenges above, different parameters were measured and added to the prediction model. Unknown parameters were measured using third-party tools. Memory bandwidth was measured using a tool[2] that performs sequential reads and writes of different amounts of data. The read bandwidth for 16MB of sequential data is used as the memory-read-bandwidth (MBW_{RD}). The write bandwidth for the 16MB of sequential data is used as the memory-write-bandwidth (MBW_{WR}). For network bandwidth, the theoretical bandwidth of the underlying technology is used.

Multiple steps of refinement were required to obtain acceptable accuracy with multicore experiments. The combination of parameters that best modeled the application was MBW_{RD} , MBW_{WR} , number-of-nodes, total processors, network bandwidth, and cores-per-node. Using the "total processors" parameter lead to consistently better results, although it depends on the model's ability to suppress less-representative parameters (in this case, separate number-of-nodes and cores-per-node parameters). For validation across different execution domains (e.g. *Mind* and *Abe*), a *platform contribution* parameter was measured using a benchmark. Inserting these parameters into Equation (1) results in Equation (2),

$$T_{exec} = \Gamma_{MBW_{RD}} \times \Gamma_{MBW_{WR}} \times \Gamma_{nn} \times \Gamma_{nc} \times \Gamma_{p} \times \Gamma_{NBW} \times \Gamma_{pc} \qquad (2)$$

where Γ_x refers to the contribution of parameter x to the overall execution time. The values of the contribution parameters are as follows:

$$\Gamma_{MBW_{RD}} = A_0 + \frac{A_1 \times MBW_{RD}}{N_c}$$

$$\Gamma_{MBW_{WR}} = B_0 + \frac{B_1 \times MBW_{WR}}{N_c}$$

$$\Gamma_{nn} = C_0 + \frac{C_1}{N_n}$$

$$\Gamma_{nc} = D_0 + \frac{D_1}{N_c}$$

$$\Gamma_{p} = E_0 + \frac{E_1}{N_c \times N_n}$$

$$\Gamma_{net} = F_0 + \frac{F_1 \times NBW}{N_c}$$

$$\Gamma_{pc} = G_0 + \frac{G_1}{pc}$$

[2] http://home.comcast.net/~fbui/bandwidth.html

N_c is the number of cores-per-node, N_n is the number-of-nodes, *NBW* is the network bandwidth, and *pc* is the platform contribution. Note that for each parameter, there is a constant contribution, i.e. the X_0 factor, and a contribution due to the magnitude of the resource contribution, i.e. the X_1 factor. The memory and network bandwidth parameters are divided by the amount of cores used per node since each processing core in the node needs to share the memory bus and network card. Parameters that have an inverse relationship with execution time (e.g. number-of-nodes) are inversed in the formula.

III. SOFTWARE TOOLS

The tools used were described in detail in [2]. A brief summary of how they were deployed is described here.

The tool that performs the prediction is *aprof*. *Aprof* is executed on a server node and receives reports from the monitoring programs distributed over the compute nodes. The gathered data is then used as statistical data to be used for prediction. As described in [4], *amon* adds negligible overhead to application execution and *aprof* is able to compute prediction in under one second.

The tool that monitors the parameters during executions is *amon*. It runs as a background process, tracking new processes on the system and recording statistics related to their resource utilization. *Amon* is executed on every worker node. Upon completion of a process, it reports the execution time and the values of resource consumption parameters that it monitored.

One of the design goals of *aprof* and *amon* is ease-of-use. This can be a detrimental factor on leased resources where limited control is available. Deploying *amon* in our previous work on a Rocks[3] cluster, where jobs are run directly, was trivial since all nodes were directly accessible. This is not the case on large systems at supercomputing centers, where the worker nodes are an abstraction to the user and no direct interface is provided to them until they have been allocated for a job. In order to execute applications concurrently with *amon*, *amon* must be included in the job execution script. This required adding less than 10 lines of code had to the job scripts.

IV. EXPERIMENTS AND OBSERVATIONS

Experiments were carried out in the test systems described in Table I. Four data sets were obtained for each configuration and the average execution time of each run was measured. In cases where an outlier was detected, it was discarded.

In choosing the number of nodes requested for simulations and the sizes of the WRF input domains chosen, compromises had to be made so as to avoid prohibitive times in the queue. The data sets consisted of multiple WRF simulations with domains with resolutions between 10 and 30km and areas between 2.25×10^6 and 4×10^6 km^2. For the large systems (*Abe* and *Marenostrum*), 8, 16, 32, and 64 node executions were performed. On *Mind*, the data sets consisted of 4, 8, 12, and 16 nodes. For all systems, 1, 2, and 4 processes per node were used. These jobs usually required less than one day waiting in the queue before executing.

[3] http://www.rocksclusters.org

TABLE I. SYSTEMS USED TO TEST OUR METHODOLOGY

Host Name	CPU	Cores	Max Nodes	Interconnect
Mind	Xeon Nocona 3.6GHz	2	16	1GigE
Abe	Xeon Clovertown 2.33GHz	8	64	10GigE
Mare-nostrum	Power 970MP 2.3GHz	4	128	Myrinet
Kitana	Core 2 Duo Yorkfield 2.66GHz	4	1	N/A

In Section II.B, the problem of execution time variation was mentioned. The severity of this on *Marenostrum* is demonstrated in Fig. 2a. It shows that the execution times for different jobs, with the same runtime configuration, vary by over 15% in some cases, which can make obtaining accuracy within our target of 10% impossible. On *Abe,* similar variation to *Marenostrum* was observed. On *Mind*, which does not have the network-related inconsistency problem, the execution times were within 1% of each other.

Since addressing the issue of runtime variation was out of the scope of this work, we worked around the problem. On *Abe*, back-to-back executions of each simulation were performed on a single job allocation. This ensured that each execution was performed by the same set of nodes. Doing this resulted in longer queue times, since job queues favor faster jobs, but resulted in less than 2% variation in execution time. *Marenostrum*'s resource manager provides more sophisticated node allocations. We performed experiments to see if consistent results could be achieved without necessarily using the same nodes for all the jobs. The interconnection of *MareNostrum* is configured as a k-ary-n tree [9], with k=16 and n=3. There are 41 servers distributed into four *Clos Networks* [10], which are a kind of switching network with multiple stages. It is possible to request a dedicated *Clos* for performance reasons. Taking advantage of this feature resulted in a minor queue time penalty, while giving execution times within 2%, due to the more consistent network bandwidth and latency, as can be seen in Fig. 2b.

Due to the problem mentioned in Section II.C, looking at a cluster of multicore systems as a simple "collection of processors" will not properly model the execution. There is a non-linear relationship between execution time and the number of processors. The speedup from adding additional core(s) is less than from adding the equivalent amount of additional node(s), in all cases. For example, an eight-node, 2 process-per-node execution took 579 seconds, on average; a four-node, 4 process-per-node execution took 786 seconds.

Our model attempts to form linear relationships amongst the parameters of execution. To achieve linearity, it is necessary to distinguish between processes running on separate nodes and processes running on separate processors/cores within a node. Fig. 3 shows that when the number of cores-per-node is kept constant, the execution pattern is linear or semi-linear and predictable. A similar relation holds when keeping number of nodes constant while varying the number of cores. In the figures, we use the inverse of number-of-nodes, since the execution time is inversely proportional to the number of nodes (i.e. more nodes should result in lower execution time).

In Section II.D, the different contributors to execution time that vary across platforms were discussed. The solution proposed for this problem is to use benchmarks to determine a *platform contribution* coefficient, which indicates the impact factor on application execution time by virtue of the platform the application is running on.

Well-known synthetic benchmarks were executed on *Mind*, *Abe,* and *Kitana*. Since only the single-process contribution was of interest, only serial benchmarks were performed. The first set of benchmarks were the SP and BT from the NASA Advanced Supercomputing (NAS) multizone suite [11]. The results from this test show that clock speed alone is a poor indicator of processor performance. The execution times were highest for the system with the highest clock speed (*Mind*) and lowest for the system with the lowest clock speed (*Abe*). The same relation held for the operations-per-second value reported by NAS.

The same relation between execution time and clock speed was observed after performing a WRF benchmark, which was a popular 12-hour simulation recorded during January of 2000.

Ultimately, the BT-MZ, class A benchmark was used for the *platform contribution* for the cross-platform predictions. The inputs for these experiments were the single-process-per-node executions on *Mind* and *Abe* with 8, 10, 14, and 16 nodes (*Mind*) and 8, 16, 32, and 64 nodes (*Abe*), using three different WRF domains.

V. VALIDATION

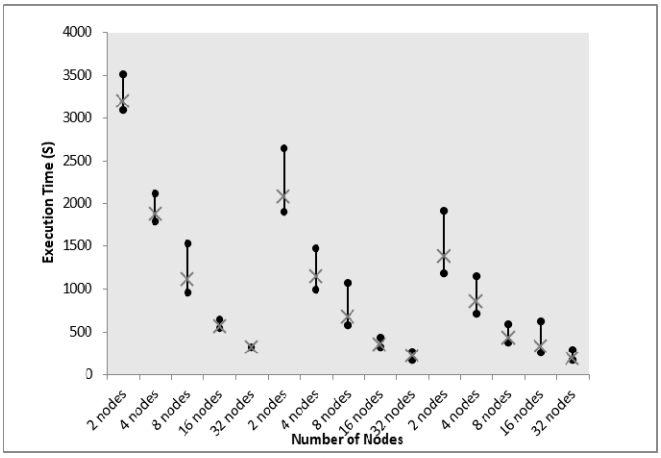
The error percentage was calculated using Equation (3). The rest of this section shows the results obtained for each of the experiments performed.

$$error = 100 \times \frac{|t_{acual} - t_{predicted}|}{t_{actual}} \quad (3)$$

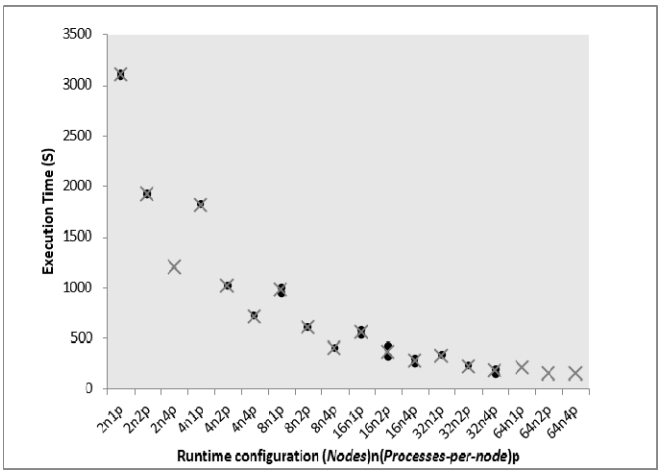
A. Executing on up to 128 nodes

Data points for 8, 16, 32, 64, and 128 nodes were recorded for the large-scale experiments on *Marenostrum*. First, the model was tested by using all data points as input and output, to ensure accuracy when performing interpolation. In order to test how well it could predict scalability from 32 nodes to 128 nodes (i.e. to test extrapolation accuracy), another test was run, with input data for only 8, 16, and 32 nodes executions used as input, and up to 128 nodes as output.

The model's predictions in the first test had an average error of 3.54%, a maximum error of 7.82%, and a minimum error of 0.83%. Similarly, in predictions where data for only 8, 16, and 32 nodes were fed to the model, the mean error obtained was 2.67%, the maximum error was 6.55%, and the minimum error was 1.27%. In both cases, the error with 64 nodes was greater than with the rest of the executions. The greatest change in slope for the actual execution times occurs when 64 nodes are used. Since no profiler was available, we could not determine the exact cause of this inconsistency. This is likely the point at which the communication overhead becomes the bottleneck for this particular data set on this platform. The low overall error rate for this particular data set



(a)



(b)

Figure 2. Range of execution times with (a) and without (b) using a dedicated *Clos* for the executions. Average execution times are marked with "X."

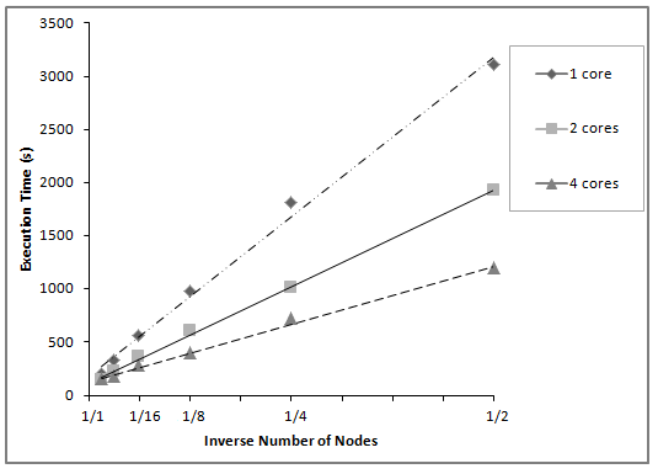


Figure 3. Relationship between execution time and inverse of number-of-nodes, when number-of-cores is kept constant.

shows that the prediction model is robust against these minor inconsistencies in execution behavior.

B. Multicore

Multicore data sets consisting of executions with between 8 and 64 nodes on *Marenostrum* and *Abe*, and 8-16 nodes on *Mind,* were generated. Executions with 1, 2, and 4 processes per node were generated. The observed error on *Marenostrum* is shown in Table II. The actual versus predicted execution times are shown in Fig. 4.

C. Different Architectures

The fact that the model worked well on *Marenostrum* and *Abe* shows that it can be used on different architectures without modification. What is left to show is how well it works when the input consists of data points from multiple systems, with different processors, to predict for one or more of these systems. For this study, we used input data from *Abe* and *Mind* to predict first for *Abe* and then for *Mind*. Out of all the experiments performed, the maximum error observed was 10.12%; the maximum of the mean error values was 6.74%. Fig. 5 shows the actual versus predicted execution times. For the *platform contribution*, the operations-per-second value of the NAS BT-MZ, Class A benchmark was used. Since there was no combination of systems containing the same kinds of processors but different clock speeds, the clock speed was not considered for the *platform contribution*.

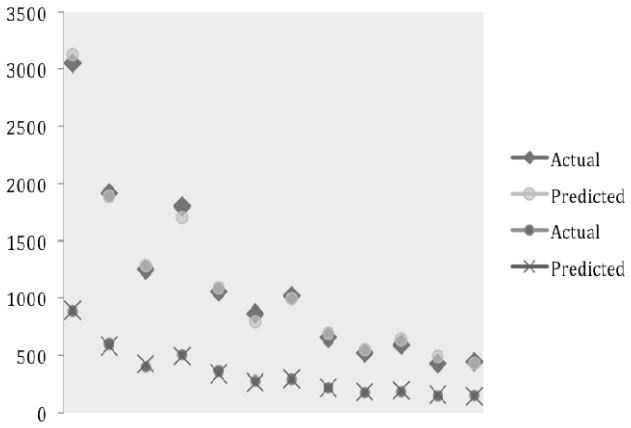


Figure 4. Actual versus predicted execution time for two different WRF simulations. All data is from executing on *Marenostrum* using 8-64 nodes and 1-4 cores per node.

TABLE II. PERCENT ERROR OF PREDICTED EXECUTION TIME

Data Set	*Execution Platform*	Mean	Min	Max
300x300x4	*Marenostrum*	4.37	0.18	8.68
1500x1500 x15	*Marenostrum*	5.2	1.50	15.08
300x300x4	*Abe*	5.19	1.64	8.44

VI. RELATED WORK

Performance prediction has a long history, but new uses have emerged, particularly for HPC applications. Early work was done for the purpose of architecture design [8]. Current performance prediction uses include resource allocation and data center design. Due to the large history of this field of study, we only provide related work on performance prediction techniques specifically related to HPC applications.

Some of the wide range of approaches include those based on using well known analysis and optimization tools such as *Dimemas* and *Vampir* [12], tasks scheduling [13], and statistics [2][7][14]. Other works focus on system specific approaches [15]. In [16], the authors describe the use of the *Dimemas* tool to perform prediction of the execution behavior of message passing applications. Their results are good, but the tools used require intimate knowledge about the execution domain and special trace files need to be generated for each application. It is possible to dynamically link the trace-file-generation library with most applications, but in some cases a complete recompilation of the application being profiled is necessary. Results of using *Dimemas* with more modern supercomputing infrastructure were not available for comparison.

Smith, Foster and Taylor [14] use linear regression for their prediction model, but they base their prediction model on application "templates." Their approach relies on user input, we attempt to create a solution that predicts without user input about the application, unless its necessary to accurately model it.

Gruber et al. [15] present a similar approach in trying to model the execution time of an application on a particular set of resources for use in meta-scheduling decisions in a grid environment as part of the *Ianos* project. Their model includes several application-specific parameters and characteristics, which can be done with our model, but is not necessarily a prerequisite for good results. A large number of parameters may need to be input by the user in regards to the application and target architecture, which can result in long times for deployment.

The PACE [17] system includes a method for predicting execution time in parallel and distributed environments. Their method is based on source code analysis, quite opposed to the philosophy of our work where no such analysis is required.

The effort in [18] allows for cross-platform performance prediction of parallel applications. The prediction is achieved by combining the application's performance in a reference system and the relative performance between the two systems derived from a partial execution on the target platform. The source code of an application is analyzed to identify the major time step loops and the source code is then modified to include the API for the partial execution measurements. A key drawback of this effort is that since run-time profiling is not incorporated, dynamic changes in the system environment unavoidably lead to inaccurate predictions.

Reference [19] provides an approach similar to ours. The authors predict application scalability on up to 1024 processors. They estimate communication and computation times separately. Our approach models execution as a function of

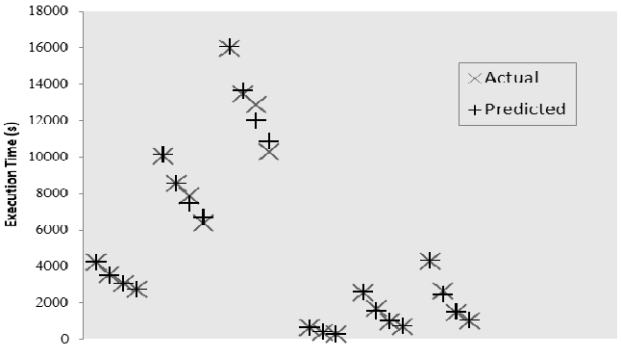


Figure 5. Actual versus predicted execution times when using 2 execution platforms and 3 WRF domains.

communication time and computation time. Another fundamental difference with their work is that they use PMPI, which works at the source code level, to instrument the communication. Our approach is to profile externally using operating system facilities. Also, our approach uses more platform-related parameters to fine-tune the prediction model, whereas theirs uses inputs to the program's execution. Also, it seems that the processes-per-node parameter remains constant in all of their experiments. We test the module with different combinations of processes-per-node. Finally, they do not show their model's cross-platform prediction accuracy, since they test on a single system with 1000 2-core processors.

VII. CONCLUSION

This paper has shown how accurately the performance of WRF on multi-node, multi-core systems can be predicted using our performance prediction methodology. We found that refinement of our prediction model was necessary for both parallelism and CPU contribution when predicting for multicore systems with very different processors. After all of the refinements, the accuracy achieved is comparable to related work in the field. In the future, we plan to add more systems to our test-bed and test with different programs.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation (grants OISE-0730065, OCI-0636031, HRD-0833093, IIS-0552555, HRD-0317692, CNS-0426125, CNS-0520811, CNS-0540592, and IIS-0308155) and in part by IBM. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF and IBM. The authors would like to thank Javier Figueroa for his assistance with the benchmarking process, Dr. Rosa Badia and the *Marenostrum* support group for helping us run our benchmarks on *Marenostrum*, and the CEPBA development team for their help using the *Dimemas* tool. This research was supported in part by the National Science Foundation through TeraGrid [20] resources provided by NCSA.

REFERENCES

[1] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, "The weather reseach and forecast model: software architecture and performance," i*n Proc. 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, 2004.

[2] S. Masoud Sadjadi, Shu Shimizu, Javier Figueroa, Raju Rangaswami, Javier Delgado, Hector Duran, and Xabriel Collazo, "A modeling approach for estimating execution time of long-running scientific applications." *In Proc. 22nd IEEE International Parallel & Distributed Processing Symposium, the Fifth High-Performance Grid Computing Workshop,* 2008.

[3] Coregird Network of Excellence Deliverable Number D.RMS.06, "Review of performance prediction models and solutions," 2006.

[4] Shuichi Shimizu, Raju Rangaswami, Héctor A. Durán-Limón, and Manuel Corona-Perez. "Platform-independent modeling and prediction of application resource usage characteristics". *The Journal of Systems & Software*, vol. 82, pp. 2117-2127, Dec. 2009.

[5] IBM. PowerPC 970MP Datasheet [online]. Available: https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/B9C08F2F7CF57095872 56F8C006727F1 [Accessed: October 10, 2009].

[6] John Michalakes, Josh Hacker, et. al., "WRF nature run," *Proceedings of the 20th ACM/IEEE conference on Supercomputing*, 2007, pp. 1-6.

[7] T. Chen, M. Gunn, B. Simon, L. Carrington, and A. Snavely, "Metrics for ranking the performance of supercomputers," *Cyberinfrastructure Technology Watch Journal: Special Issue on High Productivity Computer Systems,* vol. 2. Feb. 2007.

[8] D. E. Lang , T. K. Agerwala , K. M. Chandy, "A modeling approach and design tool for pipelined central processors," *In Proceedings of the 6th annual symposium on Computer architecture*, 1979, p.122-129.

[9] M.M. Hafizur Rahman, and S. Horiguchi, "HTN: a new hierarchical interconnection network for massively parallel computers," *IEICE Trans. Inf. Syst*. E86-D (9), pp. 1479-1486, 2003.

[10] C. Clos, "A study of non-blocking switching networks," *Bell Syst. Tech. J.*, vol. 32, pp. 406-424, 1953.

[11] R. van der Wijngaart and H. Jin, "NAS Parallel Benchmarks, Multi-Zone Versions," NASA Ames Research Center, Moffett Field, CA. Tech. Rep. NAS-03-010, 2003.

[12] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, and Luis Gregoris, "DiP: A parallel program development environment," *In Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*, 1996, p. 665-674.

[13] D. Katramatos and S.J. Chapin, "A Cost/Benefit Estimating Service for Mapping Parallel Applications on Heterogeneous Clusters*," In proc. 7th IEEE International Conference on Cluster Computing*, 2005, pp. 1-12.

[14] Warren Smith , Ian T. Foster, and Valerie E. Taylor, "Predicting application run times using historical information," *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp.122-142.

[15] Hassan Rasheed, Ralf Gruber, and Vincent Keller, "IANOS: An intelligent application oriented scheduling middleware for a HPC grid". CoreGRID. Tech. Rep. TR-0110, 2007.

[16] Rosa M. Badia, Francesc Escalé, et. al., "Performance prediction in a grid environment," *In proceedings 1st European Across Grids Conference*, 2004, pp. 257-264.

[17] S. A. Jarvis, D. P. Spooner, et. al. "Performance prediction and its use in parallel and distributed computing systems*," Future Gener. Comput. Syst.* 22 (7) (2006), pp.745–754.

[18] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution". *In proceedings of 18th Supercomputing Conference*, 2005, pp. 40–49.

[19] Bradley J. Barnes, Barry Rountree, David K. Lowenthal, Jaxk Reeves, Bronis de Supinski, and Martin Schulz. "A regression-based approach to scalability prediction." *In Proceedings of the 22nd annual international conference on Supercomputing*, 2008, pp. 368–377.

[20] C. Catlett, et al. "TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications". *HPC and Grids in Action, Ed. Lucio Grandinetti, IOS Press 'Advances in Parallel Computing' series*, Amsterdam, 2007.