Parallel Processing Letters

CHARACTERIZING LARGE-SCALE HPC APPLICATIONS THROUGH TRACE EXTRAPOLATION

LAURA CARRINGTON, MICHAEL LAURENZANO and ANANTA TIWARI

Performance Modeling and Characterization (PMaC) Laboratory
University of California, San Diego
San Diego Supercomputer Center
San Diego, California 92093
Email: {lcarring, michaell, tiwari}@sdsc.edu

Received August 2013
Revised October 2013
Communicated by Guest Editors

ABSTRACT

The analysis and understanding of large-scale application behavior is critical for effectively utilizing existing HPC resources and making design decisions for upcoming systems. In this work we utilize the information about the behavior of an MPI application at a series of smaller core counts to characterize its behavior at a much larger core count. Our methodology first captures the application's behavior via a set of features that are important for both performance and energy (cache hit rates, floating point intensity, ILP, etc.). We then find the best statistical fit from among a set of canonical functions in terms of how these features change across a series of small core counts. The models for a given feature can then be utilized to generate an extrapolated trace of the application at scale. The accuracy of the extrapolated traces is evaluated by calculating the error of the extrapolated trace relative to an actual trace for two large-scale applications, UH3D and SPECFEM3D. The accuracy of the fully extrapolated traces is further evaluated by comparing the results of building performance models using both the extrapolated trace along with an actual trace in order to predict application performance. For these two full-scale HPC applications, performance models built using the extrapolated traces predicted the runtime with absolute relative errors of less than 5%.

Keywords: Performance Modeling, Workload Characterization, Trace Extrapolation

1. Introduction

Application performance models are vital in understanding the complex interactions between applications and High Performance Computing (HPC) systems. These models can aid in determining how future technologies could improve performance and power efficiency. As HPC systems grow in scale, complexity and costs, it is important that design and deployment decisions for the next generation systems rely on solid foundations provided by performance models.

Two of the major goals of performance modeling are to reduce the time-to-solution for strategic applications and to aid in the design of future systems. The for-

mer can be best achieved if the performance sensitivity of applications to attributes of the system are carefully understood and quantified. This enables users to develop and tune their applications for a particular system and scale, often achieving far better performance for their efforts. Given current and projected architectural scale and complexity, time-to-solution of scientific simulations increasingly depends on subtle, complex machine/code interactions. Understanding and modeling this complexity is a prerequisite to designing and tuning applications to achieve substantial fractions of peak hardware performance at large scale. In addition, system architects can benefit from quantitative descriptions of application resource demands to inform the design process. Design tradeoffs can be evaluated in terms of how they are likely to affect a workload, or perhaps customization can be explored in order to improve workload performance.

Historically, performance models have indeed been used to improve system designs, inform procurements, and guide application tuning. Unfortunately, producing performance models at large scale has often been very laborious and required large amounts of time and expertise. These constraints have limited the use of performance models to a small cadre of experienced developers. This is because state-of-the-art application performance prediction and modeling techniques depend heavily on application characterizations. These characterizations, which consist of low-level details of how an application interacts with and exercises the underlying hardware subcomponents, are expensive to construct in terms of both time and space; time because the process of measuring fine-grained application behavior often is possible only at significant runtime expense and space because characterization data typically is gathered per process and thread, generating large amounts of data and thereby creating post-processing and book-keeping challenges. While significant progress has been made to reduce time and space overheads [1], characterizing application execution at scale still poses significant difficulties, which will only be exacerbated as the degree of node-level parallelism and the scale of supercomputers grows.

This paper presents *a methodology to address the challenge of constructing large-scale (or large core count) application characterizations by extrapolating the application behavior characterization data collected at a series of smaller core counts.* We first identify a set of application features that are important for both performance and energy (e.g., cache hit rates, floating point intensity, data dependencies, ILP, etc.). Our extrapolation methodology finds the best statistical fit from among a set of canonical functions in terms of how each of those features changes across a series of small core counts. The statistical models for each of these application features then form a basis for generating a fully extrapolated trace of the application at per instruction level at scale. We explore two different techniques for trace extrapolation and present their results.

By utilizing trace extrapolation to generate the trace file at the higher core count the methodology avoids the large cost of trace collection, enabling analysis and understanding of large-scale behavior using data that can be collected cheaply.

The extrapolated trace can be used in a modeling framework to investigate the application's behavior at the larger core counts to gain insight into the challenges of running at scale.

The extrapolated traces are first validated by comparing them to actual collected traces at the same core count. The sensitivity of the trace file elements is then explored to determine the acceptable level of error. In addition to evaluate the overall accuracy of the approach in generating large-scale performance models, it was incorporated into the PMaC performance modeling framework. The framework was then used to generate performance models for each application using both the extrapolated trace and the actual collected trace at a large core count for two HPC applications: SPECFEM3D_GLOBE [2] and UH3D [3].

This remainder of this paper is organized as follows: Section 3 describes the PMaC performance modeling framework, which is designed to automate modeling the performance and energy of large scale parallel applications. Section 4 details the two techniques used in our trace extrapolation methodology, along with its incorporation into the PMaC framework. Section 5 covers the error sensitivity in the method and Section 6 describes the results of applying the extrapolation methods in order to model the performance of two large scale applications: SPECFEM3D_GLOBE and UH3D. Finally, Section 8 concludes.

2. Related Work

Performance models have been used to improve system designs, inform procurements, and guide application tuning [4, 5, 6, 7]. A variety of different approaches to developing performance models have been explored in the literature. Kerbyson et al. [8, 9] utilize detailed application-specific knowledge to construct performance models. These models are highly accurate, however, the mostly manual modeling exercise has to be largely repeated when the structure of the code or the algorithmic implementation changes. Vetter et al. [10] combine analytical and empirical modeling approaches to incrementally construct realistic and accurate performance models. Code modification must be made in the form of adding annotations or "modeling assertions" around key application constructs, which limits the scalability of this approach for large scale HPC applications that have many thousands of lines of code. Various other researchers [11, 12, 13, 14] have also used application-specific approaches to generate performance models, however in general they are difficult to automate and generalize because they require extensive guidance from domain experts.

An alternate approach to application-specific model construction is the trace-driven approach [15, 16, 17, 18]. The basic guiding principle of this approach consists of probing the target architecture with a set of carefully constructed microbenchmarks to learn fine-grained details pertaining to the performance of important system components like the CPU, memory and network subsystems' and then mapping that knowledge to different computation and communication phases within an ap-

plication. Unlike the application-specific approach, trace-driven modeling is easy to automate and generalize. However, one of the drawbacks of trace-driven modeling is the increased time, space and effort associated with collecting and storing application trace data. Reducing the trace collection overhead by extrapolating large traces from smaller traces is the focus of this paper.

There has also been some work done on predicting the scalability of HPC applications based on execution observations made from smaller core count runs. Barnes et al. [19] use regression-based approaches on training data consisting of execution observations with different input sets on a small subset of the processors and use the models to predict performance on a larger number of processors. Others [20, 21] have used machine learning and piecewise polynomial regression to model input parameter sensitivity of HPC applications. The aforementioned modeling techniques are application-specific and the training configurations for regression and machine learning are drawn from the input parameter space. Our method extrapolates low-level measurable features of applications' computation phases at smaller core counts to predict execution time at larger core counts and therefore is more generalizable.

Finally, we note work done in communication trace extrapolation. Wu et al. [22] synthetically generate the application communication trace for large numbers of nodes by extrapolating traces from a set of smaller traces. The work presented in this paper is for scaling an application's computation behavior, which can be complemented by communication trace extrapolation.

3. The PMaC Prediction Framework

The PMaC prediction framework is designed to accurately model parallel applications on HPC systems. In order to model a parallel application, the framework is composed of two models – a computation model and a communication model. The computation model focuses on the work done on the processor in between communication events, while the communication model deals with modeling communication events. Below a brief description is provided but for a detailed description of the framework please see previous work [17][16][23][24] on the subject. The PMaC prediction framework is comprised of three primary components: an application signature, a machine profile, and a convolution method. The machine profile is a description of the rates at which a machine can perform certain fundamental operations through simple benchmarks or projections. These simple benchmarks *probe* the target machine to measure the behavior of different kinds of memory access patterns, arithmetic operations, and communications events, at various working set sizes and message sizes.

The application signature, which is collected via PMaC's application tracing tools [25][26], includes detailed information about the operations required by an application, its data locality properties and its message sizes. The machine profile and application signature are combined using a convolution method – a mapping of the operations required by the application (the application signature) to their expected

behavior on the target machine (the machine profile). This mapping takes place in the PSiNS simulator that replays the entire execution of the HPC application on the target/predicted system in order to calculate a predicted runtime of the application on the target system. The models generated by the framework have shown good accuracy (usually less than 15% absolute relative error) when predicting execution time for full-scale applications running production datasets on existing HPC systems [27][28].

In this paper we extend the computation model within the PMaC framework. Recall that the computation model focuses on the work done on the processor or core in between communication events, details of which are captured as part of the application signature. Extrapolating the application signature collected at a series of small core counts to the application signature for a large core count and using the extrapolated signature to predict application execution time at the large core count are the key contributions made by this work.

3.1. *Application Signatures and Machine Profiles*

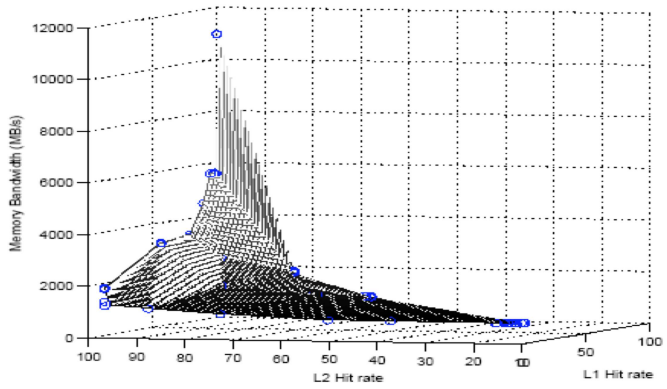

Fig. 1. Measured bandwidth as function of cache hit rates for Opteron

For the computation model there are two main classes of operations that normally comprise a majority of the run time: arithmetic operations and memory operations. Arithmetic operations are floating-point and other math operations; memory operations are load and store memory references. Memory time modeling typically, but not always, dominates the computation model's run time. Because references from different locations and access patterns can perform orders of magnitude better or worse than others, to accurately model memory time we need to determine not only the number of bytes that need to be loaded or stored but also the locations and access patterns of those references. For example, a stride-one load access pattern from L1 cache can perform significantly faster than a random-stride load from main memory. Figure 1 is an example of a surface created using the MultiMAPS

benchmark [15] for a two cache level Opteron processor. MultiMAPS probes a given system to generate a series of memory bandwidth measurements across a variety of stride and working set sizes, which in Figure 1 is reflected by varying cache hit rates on the x and y axes. The MultiMAPS surface for a processor is used as a component of the machine profile within the PMaC modeling framework, and allows the computation behavior to be modeled as a function of differing memory behaviors of different sections of an application.

To capture the properties of memory behavior for an application, we utilize a tool implemented using the PEBIL binary instrumentation platform [25] to instrument every memory access in the application for trace collection. PEBIL works directly on the compiled and linked executable so that the automation of this complex instrumentation step is very straightforward and easy, even on large-scale HPC applications. After PEBIL instruments the executable/binary to capture the memory address from each memory reference, this instrumented executable is run on a base system and the address stream of the application is processed on-the-fly through a cache simulator which mimics the structure of the system being predicted (the target system). Figure 2 gives an example of this. It shows that each process (or MPI task) in a parallel application has its own memory addresses instrumented to generate a memory address stream. The address stream of a single process can generate over 2 TB of data per hour so to alleviate this space requirement the address stream is processed *while the application is running* through the cache simulator in order to produce a summary trace file for each MPI task.

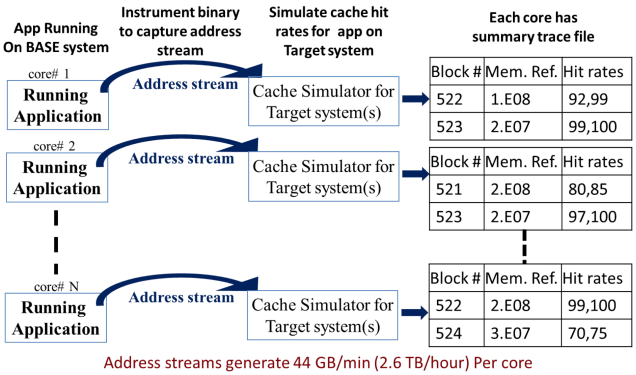

Fig. 2. Application signature collection, built on-the-fly using the memory address stream of the running application

This trace file contains, for each basic block of the application, information about: 1) the location of the block in the source code and executable, 2) number of floating-point operations and their type, 3) number of memory references and whether they are loads or stores, 4) size of its memory references in bytes, and 5) the expected cache hit rates for those references on the target system. It is these hit

rates that provide key information about the runtime behavior of the application's memory references and enable accurate predictions of their performance via their corresponding data on the MultiMAPS surface (part of the machine profile).

The set of trace files from all MPI ranks constitutes the application signature on the target system at that particular core count. It is important to note that the application signature is collected on a base system which need not be the same as the target system. The base system is some system that the application can run using the same parallelization mode (e.g., MPI or hybrid MPI/OpenMP) that will be used on the target system. The framework enables the application to be traced on the base system, supplying the memory address stream of the running application to a cache simulator for the target system, enabling the framework to predict the performance of the application on the target system. This means that an application signature for a target system can be generated without having access to the target system. Therefore, a model for the application running on the target system can be generated without ever having ported the application to the system, or without the existence of a target system. Performance predictions generated in this fashion are known as cross-architectural predictions. For this work the application characterizations were gathered on a CRAY XT5.

3.2. *Modeling Computational Behavior*

In the computation model the majority of the time usually comes from the memory time – the time required to move data through the memory hierarchy. Arithmetic time is also modeled but memory time tends to dominate in the cases we studied. A detailed description of the memory time calculation can be found at Tikir et al. [27]. The form of the memory time equation is:

$$memory\_time = \sum_{i}^{allBBs} \frac{(memory\_ref_{i,j} \times size\_of\_ref)}{memory\_BW_j} \quad (1)$$

In Equation 1, the summation denotes that we take a sum of the predicted memory time for all the basic blocks in the application. The denominator $memory\_BW_j$ is the memory bandwidth if the j^{th} type of memory reference on a target system. $size\_of\_ref$ is the size in bytes of the reference and $memory\_ref_{i,j}$ is the number of memory references for basic block i of the j^{th} type. Where a block falls on the MultiMAPS curve – its working set and access pattern as expressed through its cache hit rate – is encompassed in its type.

Equation 1 shows that the memory time of an application is calculated as the sum of the memory time for all basic blocks within the application. Floating point time is modeled in a similar way with some overlap of memory and floating-point work. The majority of the computation time is in moving data throughout the memory hierarchy and thus the focus of this presentation moving forward will be in accurately capturing the information required for the calculation of memory time.

In Equation 1 the parameter $memory\_ref_{i,j}$ refers to memory references for basic block i of the j^{th} type, which also contains information relating to its cache hit rates. In the application signature this information is represented for a single basic block by the simulated cache hit rates for the target system along with its data footprint size. This memory-related information, the floating-point operation data, and the remaining parameters in Equation 1 make up a feature vector for a given basic block. Each basic block for a given MPI task or core is represented by a feature vector which contains (1) amount and composition of floating point work (FPops), (2) number of memory operations (Mops), (3) size of memory operations (Msize), (4) cache hit rates in all levels of the target system (L1hr, L2hr, L3hr) and (5) working set size (WSsize). An example representation of the feature vector for a basic-block is as follows:

```
<FPops, Mops, Msize, L1hr, L2hr, L3hr, WSsize>
```

A collection of feature vectors for all the basic blocks that were executed by each MPI task make up the final signature for the given application. The PSiNS simulator consumes these feature vectors and for each basic block, utilizes Equation 1 to compute the corresponding memory and floating-point time. Memory bandwidth for each block is found at the appropriate location on the MultiMaps curve for the target system. The result of this exercise is the predicted computation time for the application on the target system.

4. Trace Extrapolation

The goal of this work is to develop a methodology that can generate an application signature for a large core count (e.g., 8192 cores) given the application signatures of a series of smaller core count executions (e.g. 1024, 2048, and 4096 cores) of the application. The methodology will eliminate the cost of collecting application signatures at high core counts. An application signature consists of a series of trace files, one file for each MPI task. In this work we explore two methods of extrapolating the trace files. The end goal of both the methods is the same – for each of the smaller core counts, derive a single trace data file that represents the work done by application's computational units (e.g., basic blocks and instructions). The first method involves extrapolating the trace data from the MPI task that consumed the most computational time, i.e., the task that has the most impact on the overall execution. The slowest running task is identified using a lightweight MPI profiling library based on the PSiNSTracer package [26]. Given that each MPI application is viewed only in terms of its longest running MPI task, the application signature of each smaller core count is comprised as a single trace file that represents the work done on the most computationally demanding MPI task. The second method involves extrapolating a *centroid* trace file from each of the smaller core counts; e.g., for a given basic block and a given core count, the *centroid* feature vector is generated by simply calculating the average of each element in the feature vector across all MPI tasks. The *centroid* trace file attempts to represent an average computational

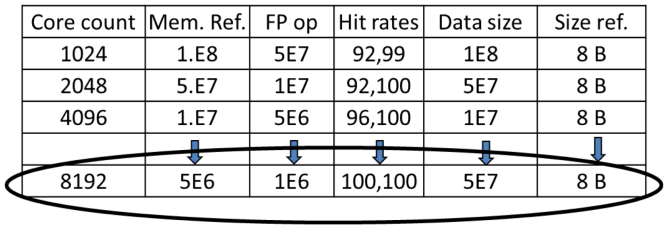

Fig. 3. Extrapolating individual elements within a basic block's prediction vector

behavior of all the tasks rather than the task contributing to the most computation. For applications where the work is evenly distributed the centroid trace file will not differ greatly from any of the individual task trace files.

Our discussions thus far on application signatures and how they are collected and processed have mostly focused on per basic block level characterizations. For this work, however, the signatures include more detailed information to enable extrapolation studies and therefore contain data at per instruction level, i.e., the trace files contain data for each instruction of all basic blocks executed by each of the MPI tasks.

Once the trace files are processed to generate single trace file for each of the smaller core count executions (either via the *centroid* trace file or the most computationally intensive task's trace file), the next step is to utilize these representative trace files to generate an extrapolated trace file for a larger core count. When an application is strongly scaled, each element in the instruction feature vector could exhibit different scaling behavior. Some elements might remain constant as the number of cores increase while others might scale according to some function (e.g., logarithmic or linear) with the number of cores. To identify how each behavior scales as core count increases, each element of the feature vector needs to be treated separately in order to extrapolate the behavior of a particular property of a particular instruction. An example of this is shown in Figure 3 for a single instruction with four elements of its feature vector, where each element is extrapolated on its own.

Figure 3 shows how each element in an instruction feature vector for the 3 core counts is used to extrapolate those values at the larger core count, illustrating the principle behind our trace extrapolation methodology. Four functions of various canonical form are fitted to each individual element of each feature vector and the best of those fits is used to extrapolate that element for the vector. We use the following four canonical forms in this work:

`b+a*ln(x)`	Logarithmic Model
`bexp(x)`	Exponiental Model
`constant`	Constant Model
`ax + b`	Linear Model

(where `a`, `b` are fitted parameters and `x` is number of cores).

Once the value of each element of a vector has been extrapolated for the higher

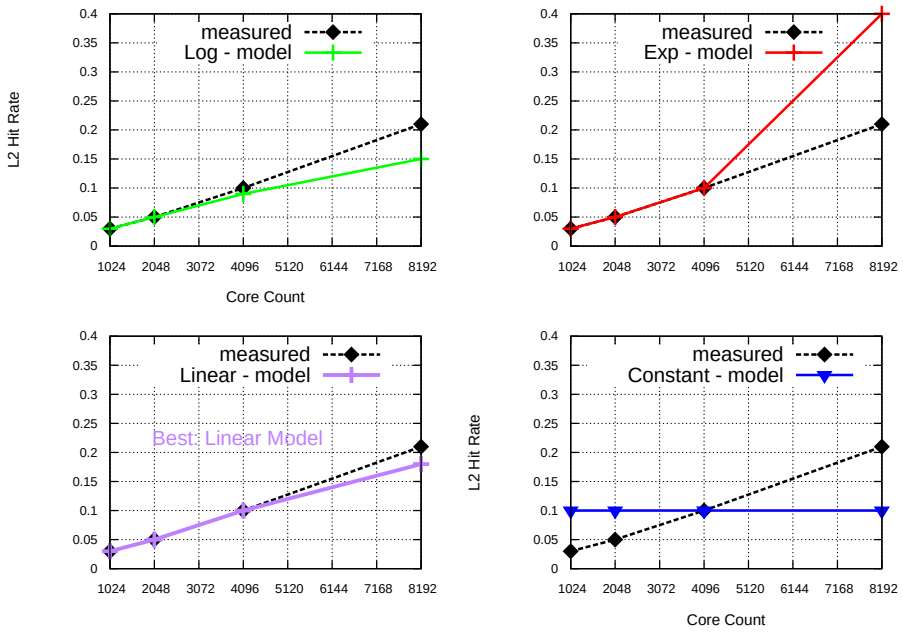

Fig. 4. Linear Model captures the scaling behavior of the L2 Hit Rate.

target core count then the vector is recorded in a new synthetic trace file. Thus, the single trace file for three core counts are used as input, which is used to generate a fourth trace file for a core count of a specified size. Note that using more than three core counts could improve the quality of the fit but it became evident during testing that three generally provided adequate accuracy.

Figure 4 and Figure 5 show that for each element in the feature vector of an instruction a different model captures the behavior as the core count is increased. In Figure 4 the measured L2 hit rate for a single instruction is plotted versus the core count along with the model fit for each of the four canonical forms. As the core count increases the hit rate increases also, which is best described by the linear form. Each element in the feature vector of a given instruction can have different behavior and require and best be described by a different form. Figure 5 plots the behavior of the memory operations for a single instruction as the core count increases. For this operation the log model clearly has the best fit.

The framework is designed to take each element of an instruction's feature vector and choose a model that best fits its behavior and use the model to generate the vector at the higher core count. This process is used for all the instructions of an MPI task to generate a synthetic application signature at the higher core count. The synthetic signature (or trace) is used to predict performance of the full application.

For most of the extrapolated elements this method of model fitting showed good accuracy (absolute relative error $<20\%$). Most of the elements that had higher er-

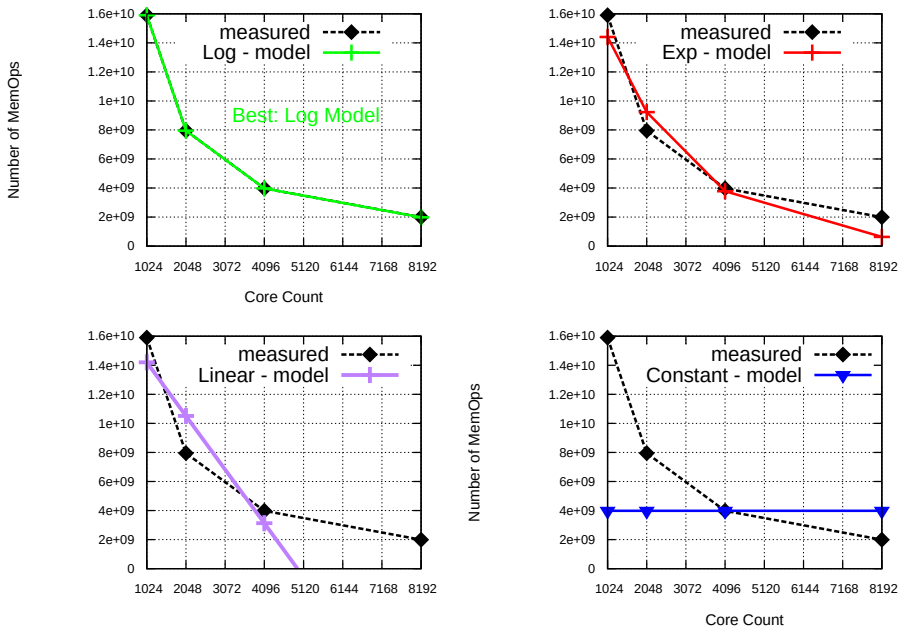

Fig. 5. Logarithmic Model captures the scaling behavior of the number of memory operations.

ror in the fit were from instructions that didn't have a significant influence on the overall runtime. This influence was determined by the ratio of memory operations the instruction had to the total number of memory instructions and for those instructions without memory operations, floating-point operations were used. The percentage deemed to have influence was anything over 0.1%. For the applications used within this work, every extrapolated element within all of the influential instructions had an absolute relative error of less than 20%.

We further explored the use of the number of canonical forms by analyzing which form was chosen the most and if using less forms would result in more error. Of the four forms, logarithmic was chosen to fit the best the most number of times, followed by linear, constant, and exponential.

We also explored which form, if used for all elements of the vector, resulted in the best fit across all the elements in the feature vector. In this case, the form with the best fit was the logarithmic, followed by linear, constant, and exponential. The size of error and the number of elements with larger errors grows considerably when we utilize just a single form. The largest error seemed to come from fits of the L1 hit rate data. This could be attributed to the fact that for some instructions the L1 hit rate values changed dramatically as the application was strong scaled and data moved all the way from main memory into L1 cache. These explorations with using fewer forms were meant to rationalize our extrapolation framework's use of different fitting forms for different elements in the feature vector. We do suspect

that increasing the number of forms used within this methodology has a strong chance of driving down the error further, though this is a matter for future work.

5. Feature Vector Error Sensitivity

To reduce the error in extrapolation and of the performance models developed using the synthetic traces, we need to further understand which elements of the feature vector are most sensitive to error. Given those elements, we can begin to make progress towards reducing the extrapolation error by, for example, adding more canonical forms to model highly sensitive elements.

Our error sensitivity experiments utilized a full-scale HPC application –UH3D, which is a global code to model the Earth's magnetosphere developed at UCSD that treats the ions as particles and the electrons as a fluid. We first ran the collected trace for UH3D through PMaC performance prediction framework to determine the predicted time, which served as the reference time. We then introduced error into the elements of some or all of the feature vectors of the trace file. Recall from Section 3 that the feature vector consists of the following 7 elements:

`<FPops,Mops,Msize,L1hr,L2hr,L3hr,WSsize>`

The error sensitivity was explored one element at a time in a series of experiments. The predicted runtime of the modified (or error-injected) trace was compared to the reference predicted runtime to determine the effect the introduced error has on the overall prediction. The error was introduced by randomly changing the value by + or − 25%. In addition, the frequency of the error, or the percentage of the vectors subjected to the error was also investigated. The frequencies investigated were 5%, 25%, 50%, and 100%. Since the error value was non-systematic (i.e. + or − 25%) and could thus result in error cancellation, we further looked at applying a systematic error where the value was always reduced by 25%.

Table 1 shows the prediction error when introducing the 25% error into the five main elements of the vector. These are the elements that are part of calculating computational time in the prediction framework. The table shows that error in the floating-point element does not have a huge affect on the prediction error. This is to be expected – time spent in data motion tends to dominate the execution time and floating point calculations often overlap with data movement. The significance of the memory time can be seen in the prediction error number when the number of memory operations are subjected to a systematic error injection; non-systematic errors cancel each other out. For the systematic error of −25%, the runtime is reduced by that exact amount.

Of the three cache hit rates, L1 hit rate is most sensitive to error. The reason for this is in the way that the $memory\_BW_j$ variable is calculated in the prediction framework. This variable is composed of three sub-components where each sub-component represents a level of cache and is calculated as a function of the number of missed references (and hit rate) from the previous level and the hits at current level of cache. The effect this has on error sensitivity can be seen in Table 1. The

effect is also a function of where the data for the application is in the memory sub-system. The table illustrates that for UH3D application the data is in the upper levels of cache, therefore when error is introduced into the L3 hit rate it has very little affect. This is because the amount of data moved from L3 and main-memory is determined by the number of elements missed in L2. Since the hit rate for L2 is 100% for most of the vectors in the trace file, any error introduced into the L3 hit rate element has little to no effect.

Table 1. Prediction errors for UH3D when 25% error introduced.

Element	5%	25%	50%	100%	100%†
Memory Ops	-0.1%	-0.1%	-0.1%	0.1%	-25.0%
FP ops	0.0%	0.2%	0.2%	0.0%	-0.2%
L1 hit rate	-0.7%	-5.0%	-7.6%	-19.3%	-44.8%
L2 hit rate	0.1%	-3.0%	-6.0%	-12.4%	-34.6%
L3 hit rate	0.0%	-1.0%	-1.5%	-2.4%	-10.2%

†*Systematic error of -25%*

The exploration of the error sensitivity revealed that the hit rate elements in the vector result in the largest prediction error. Table 1 illustrates that the location of the data in the memory sub-system also have an effect on the error sensitivity. In addition, even the most sensitive elements (e.g., L1 hit rate) with a fairly large (e.g., 25%) error have to have a systematic error in every vector in order to introduce more than 20% error in the prediction. This bodes well for the extrapolation methodology described in the previous section which introduced less than 20% error for only a few of the vectors and elements.

6. Results

After exploring the sensitivity of error for the methodology we test the accuracy for extrapolating the traces by performing the extrapolation utilizing both the single trace file and the centroid trace file methods. We then use those extrapolated traces within the PMaC prediction framework to predict the performance of two full-scale HPC applications: SPECFEM3D [2] and UH3D [3]. SPECFEM3D is a spectral-element application enabling the simulation of global seismic wave propagation in 3D anelastic, anisotropic, rotating and self-gravitating Earth models at unprecedented resolution.

Each application was scaled using strong scaling, where the data set size is held constant as the core count is increased. The effect of this on the computational work is that, as the core count increases, the work and data footprint per core begins to decrease for most computational phases in the application. We collected application traces at a large count (6144 for SPECFEM3D and 8192 for UH3D) and in addition built an extrapolated trace using three smaller core counts for each application. The accuracy of the extrapolated traces were then tested by comparing the results of using the extrapolated trace to predict performance versus using a traditional trace to predict application performance for a target system. Table 2 shows the results

of the predictions for the extrapolated traces using the single trace (Extrap-S) and centroid (Extrap-C) trace as well as the traditional collected trace (Coll). The extrapolated traces using the single trace file method produce runtime predictions with absolute relative errors from the correct runtime of less than 5%, while the centroid trace file method had higher error rates <11%. The error in the centroid trace method could be attributed to the averaging of computational work of all the MPI tasks. For both applications the spread of computational work per task varied by about 7-8%, indicating that both applications have balanced work distribution among tasks.

Table 2. Prediction errors for SPECFEM3D and UH3D using extrapolated and collected application traces.

Application	Core Count	Trace Type	Predicted Runtime (s)	% Error
SPECFEM3D	6144	Extrap-S	139	1%
SPECFEM3D	6144	Extrap-C	127	11%
SPECFEM3D	6144	Coll.	139	1%
UH3D	8192	Extrap-S	537	5%
UH3D	8192	Extrap-C	523	8%
UH3D	8192	Coll.	536	5%

For SPECFEM3D, the three core counts used to generate the extrapolated traces were 96, 384, and 1536. All traces were collected on Kraken, a Cray XT5 system at the National Institute for Computational Sciences (NICS). Traces from these small core count runs were used to generate an extrapolated trace for SPECFEM3D at 6144 cores using both the single trace file (slowest runner) and *centroid* trace file methods. Extrapolated trace for 6144 cores was then used within the PMaC modeling framework to predict the performance of the application on the Phase I BlueWaters system. In order to test the accuracy of the extrapolated traces in predicting performance, a set of actual trace files were collected at 6144 cores and were similarly used to predict the performance. In Table 2, we compare the performance predictions made using synthetic traces and actual traces. The prediction made using the traces generated using the single trace file method are comparable to that made using an actual trace. The accuracy in predicting the real measured time is also very close – predicted run time of 139 seconds using single file synthetic traces compared to the real measured runtime is 143 seconds.

For UH3D, the three core counts used to generate the synthetic extrapolated traces were: 1024, 2048, and 4096. These traces were also run through the same procedure as described above for SPECFEM3D to generate extrapolated trace files at 8192 cores using the two methods. Actual trace files were also collected at 8192 to determine the prediction accuracy of using collected trace files versus synthetic trace files. The results are again shown in Table 2: the extrapolated trace using the single trace file method produces a runtime prediction of 537 seconds, which is close to the prediction produced using the actual collected trace (536 seconds). The

centroid trace file method exhibits a slightly larger prediction error of 8% with a run time prediction of 523 seconds, where the measured runtime is 566 seconds.

For both applications the method of extrapolation provides an accurate application signature. This allows such signatures to confidently be used in the exploration of how the computation and data layout is affected when scaling to larger core counts on a particular target system. An example of this can be seen in Table 3, which shows the cache hit rates of a given basic block on the target system for a variety of core counts. The table shows that as the core count increases the data slowly moves into the L3 and L2 cache indicated by the increase in the hitrate for those cache levels.

Table 3. Changes in cache hitrates of target system as core count increases

Core Count	L1 HR	L2 HR	L3 HR
1024	87.4	87.5	87.5
2048	87.4	87.5	90.7
4096	87.4	88.4	91.6
8192	87.4	89.0	95.0

The same investigative process can be approached another way to investigate how the application would behave on two target systems whose memory hierarchies differ in some way. Table 4 illustrates this by showing the L1 cache hit rate for two systems which have identical L2 and L3 caches but which differ in their L1 cache size (12KB vs. 56KB). The table illustrates the capabilities of using this to explore the optimal cache structure for a given application and core count, all without the system even existing because the trace data is collected on a base system that is not the target system. The table shows a particular basic-block that doesn't change its behavior (i.e. L1 hitrate) when the core count is increased, thus the data for this particular computation is not affected by the strong scaling. But if the size of L1 is increased from 12KB to 56KB then the data for the computation moves into L1 cache.

Table 4. Application trace data (L1 hitrate)
for single basic-block of SPECFEM3D for two target systems

System	96 cores	384 cores	1536 cores	6144 cores
A (12 KB L1)	85.6	85.6	85.8	85.8
B (56 KB L1)	99.6	99.6	99.6	99.6

7. Future Work

Future research will add more canonical forms (e.g., polynomial) in the extrapolation of the instruction feature vector elements to improve the accuracy of the extrapolation. This will be guided by the error sensitivity analysis of the feature vector elements. Applying this methodology to weak-scaled problems is also of interest, and may pose additional challenges to our methodology.

An application signature consists of a series of trace files – for a run at 1024

cores the prediction framework uses 1024 trace files, one for each MPI task. In generating synthetic trace files from 1024, 2048, and 4096 core trace files we need to generate 8192 trace files. The challenge in extrapolating the individual trace files is determining how the work distribution per core changes as the application strong scales (a fixed problem size across multiple core counts). One approach would be to identify groups of MPI tasks that do similar work and adapt the grouping as one scales to a larger number of cores. The current work explored two methods – using the slowest running task's prediction vector as a base to scale the data in the trace files and creating a centroid trace to represent all task's work for a given core count. While each method proved to be fairly accurate we believe that we can improve the accuracy of the synthetic traces by using clustering algorithms. These algorithms could be used to first cluster MPI-tasks with similar properties and then use the "centroid" file from each cluster as a base to extrapolate data in the centroid trace files.

Furthermore, the methodology proposed in this work lays a strong foundation for determining how application input parameters affect application behavior. For example, one could attempt to determine how working set size of a computational phase is affected by the size or composition of an input file. To start to address this, a plausible approach is to employ the same scaling and extrapolating strategies used in this work to capture and model how changes in input set parameters changes the feature vectors of the application.

8. Conclusion

In this work we proposed a methodology for extrapolating the computational behavior of large-scale HPC applications by capturing the details of computational behavior at a series of smaller core counts. We then used one of the four canonical functions (linear, logarithmic, exponential or constant) to best describe how each element in an instruction level feature vector (e.g., cache hit rates) changes as the application scales. We explored two techniques to represent the application's behavior at each core count – single trace file for the slowest running MPI task and the centroid trace . Both techniques were shown to produce accurate extrapolated application traces for two full-scale HPC applications, SPECFEM3D and UH3D, by comparing the results of using the extrapolated traces against using actual collected traces in order to predict the performance of those applications at scale using the PMaC performance prediction framework. The sensitivity of the elements in the trace file were investigated to reveal that the cache hit rates were most sensitive. Extrapolating application traces is critical not only for understanding how an application scales on a particular system, but also can be useful for detecting the impact of incremental or major changes in the hardware being used to run the application.

Acknowledgements

This work was supported in part by NCSA's Blue Waters project (NSF OCI 07-25070 and the State of Illinois) and by the DOE Office of Science, Advanced Scientific Computing Research, under award number 62855 "Beyond the Standard Model – Towards an Integrated Modeling Methodology for the Performance and Power"; PNNL lead institution; Program Manager Sonia Sachs. This research was supported by an allocation of advanced computing resources provided by the National Science Foundation. The computations were performed on Kraken at the National Institute for Computational Sciences `http://www.nics.tennessee.edu/`.

References

[1] M. A. Laurenzano, J. Peraza, A. Tiwari, L. Carrington, W. Ward, and R. Campbell, "A static binary instrumentation threading model for fast memory trace collection," *International Workshop on Data-Intensive Scalable Computing Systems*, 2012.

[2] "SPECFEM 3D Globe," http://www.geodynamics.org/cig/software/specfem3d-globe.

[3] H. Karimabadi, H. X. Vu, B. Loring, Y. Omelchenko, M. Tatineni, A. Majumdar, U. Ayachit, and B. Geveci, "Petascale global kinetic simulations of the magnetosphere and visualization strategies for analysis of very large multi-variate data sets," *5th international conference of numerical modeling of space plasma flows*, 2010.

[4] D. Bailey and A. Snavely, "Performance modeling: Understanding the present and predicting the future," 2005.

[5] A. Hoisie, D. J. Kerbyson, C. L. Mendes, D. A. Reed, and A. Snavely, "Special section: Large-scale system performance modeling and analysis," *Future Generation Comp. Syst.*, vol. 22, no. 3, pp. 291–292, 2006.

[6] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. Sancho, "Using performance modeling to design large-scale systems," *Computer*, vol. 42, no. 11, pp. 42 –49, nov. 2009.

[7] D. Kerbyson, A. Vishnu, K. Barker, and A. Hoisie, "Codesign challenges for exascale systems: Performance, power, and reliability," *Computer*, vol. 44, no. 11, pp. 37 –43, nov. 2011.

[8] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, ser. Supercomputing '01. New York, NY, USA: ACM, 2001, pp. 37–37. [Online]. Available: http://doi.acm.org/10.1145/582034.582071

[9] D. J. Kerbyson and P. W. Jones, "A performance model of the parallel ocean program," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 3, pp. 261–276, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1177/1094342005056114

[10] S. Alam and J. Vetter, "A framework to develop symbolic performance models of parallel applications," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, april 2006, p. 8 pp.

[11] J. Brehm, P. H. Worley, and M. Madhukar, "Performance modeling for spmd message-passing programs," *Concurrency - Practice and Experience*, vol. 10, no. 5, pp. 333–357, 1998.

[12] H. Shan, E. Strohmaier, J. Qiang, D. H. Bailey, and K. Yelick, "Performance modeling and optimization of a high energy colliding beam simulation code," in *Proceedings of*

the 2006 ACM/IEEE conference on Supercomputing, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188557

[13] K. Barker, K. Davis, and D. Kerbyson, "Performance modeling in action: Performance prediction of a cray xt4 system during upgrade," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –8.

[14] T. Hoefler, "Bridging performance analysis tools and analytic performance modeling for HPC," in *Proceedings of the 2010 conference on Parallel processing*, ser. Euro-Par 2010. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 483–491.

[15] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, ser. Supercomputing '02. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=762761.762785

[16] L. Carrington, A. Snavely, X. Gao, and N. Wolter, "A performance prediction framework for scientific applications," in *ICCS Workshop on Performance Modeling and Analysis (PMA03*, 2003, pp. 926–935.

[17] L. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis, "How well can simple metrics represent the performance of hpc applications?" in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 48–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.33

[18] S. Sharkawi, D. DeSota, R. Panda, S. Stevens, V. Taylor, and X. Wu, "Swapp: A framework for performance projections of hpc applications using benchmarks," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1722–1731. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2012.214

[19] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22nd annual international conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 368–377. [Online]. Available: http://doi.acm.org/10.1145/1375527.1375580

[20] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPoPP '07. New York, NY, USA: ACM, 2007, pp. 249–258. [Online]. Available: http://doi.acm.org/10.1145/1229428.1229479

[21] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, ser. Euro-Par'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 196–205. [Online]. Available: http://dx.doi.org/10.1007/11549468_24

[22] X. Wu and F. Mueller, "Scalaextrap: trace-based communication extrapolation for spmd programs," in *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, ser. PPoPP '11. New York, NY, USA: ACM, 2011, pp. 113–122. [Online]. Available: http://doi.acm.org/10.1145/1941553.1941569

[23] M. A. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. M. Tikir, and S. Poole, "Reducing energy usage with memory and computation-aware dynamic frequency scaling," in *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp.

79–90.

[24] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely, "Modeling power and energy usage of hpc kernels," in *Proceedings of the Eighth Workshop on High-Performance, Power-Aware Computing 2012*, ser. HPPAC '12, 2012.

[25] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely, "Pebil: Efficient static binary instrumentation for linux," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, march 2010, pp. 175 –183.

[26] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely, "Psins: An open source event tracer and execution simulator for mpi applications," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 135–148. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03869-3_16

[27] M. M. Tikir, L. Carrington, E. Strohmaier, and A. Snavely, "A genetic algorithms approach to modeling the performance of memory-bound computations," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 47:1–47:12. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362686

[28] L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michea, N. Le Goff, A. Snavely, and J. Tromp, "High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, nov. 2008.