Inferring Large-scale Computation Behavior via Trace Extrapolation

Laura Carrington Michael A. Laurenzano Ananta Tiwari
Performance Modeling and Characterization (PMaC) Laboratory
University of California, San Diego
San Diego Supercomputer Center
San Diego, California 92093
{lcarring, michaell, tiwari}@sdsc.edu

***Abstract*—Understanding large-scale application behavior is critical for effectively utilizing existing HPC resources and making design decisions for upcoming systems. In this work we present a methodology for characterizing an MPI application's large-scale computation behavior and system requirements using information about the behavior of that application at a series of smaller core counts. The methodology finds the best statistical fit from among a set of canonical functions in terms of how a set of features that are important for both performance and energy (cache hit rates, floating point intensity, ILP, etc.) change across a series of small core counts. The statistical models for each of these application features can then be utilized to generate an extrapolated trace of the application at scale. The fidelity of the fully extrapolated traces is 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 at using each. For two full-scale HPC applications, SPECFEM3D and UH3D, the extrapolated traces had absolute relative errors of less than 5%.**

I. Introduction

Each new generation of High Performance Computing (HPC) systems is designed based on experience with existing systems and projections about how future technologies and applications will interact with each other. As HPC systems grow in scale, complexity and costs, it is important that these design and deployment decisions for the next generation systems rely on solid foundations. Application performance models provide such foundations.

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 former 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 characterization. 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 characterization 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 developing large-scale application characterization and modeling by extrapolating it from application behavior characterization data collected at a series of smaller core counts. We first identify a set of basic block 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 full extrapolated trace of the application at per basic block level at scale. This methodology avoids collection of the most costly data (that at larger core counts), instead inferring large-scale behavior using data that can be collected cheaply. This new larger core count generated data can be used to model application behavior at the larger core counts to gain insight into the

challenges that may be faced as the application scales. To evaluate this approach, it been incorporated into the PMaC performance modeling framework. The framework was then used to generate performance models for each application using both the extrapolated trace at some large core count and the actual collected trace from that core count for two large scale applications: SPECFEM3D_GLOBE [2] and UH3D [3].

The rest of this paper is organized as follows: Section III describes the PMaC performance modeling framework, which is designed to automate modeling the performance and energy of large scale parallel applications. Section IV details the trace extrapolation methodology, along with its incorporation into the PMaC framework. Section V 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 VII concludes.

II. 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 application. 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.

III. 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 then using the extrapolated signature to predict application execution time at the large core count is the key contribution made by this work.

A. Application Signatures and Machine Profiles

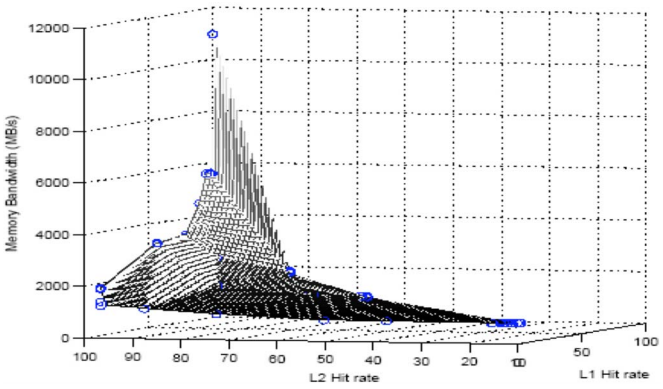

Fig. 1. Measured bandwidth as function of cache hit rates for Opteron

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.

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.

B. 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

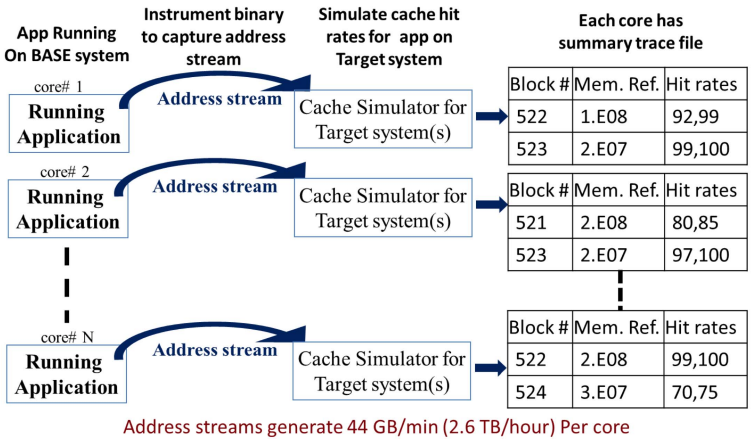

Fig. 2. Application signature collection, built on-the-fly using the memory address stream of the running application

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, (2) number of memory operations, (3) size of memory operations, (4) cache hit rates in all levels of the target system and (5) working set size.

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 for the basic blocks then 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.

IV. Trace Extrapolation

The goal of this work is to develop a methodology that is capable of generating the application signature for a large core count (e.g. 8192 cores) of application run given the application signatures of a series of smaller core counts (e.g. 1024, 2048, and 4096 cores) in order to eliminate the cost of collecting an application signature at the high core count. An application signature consists of a series of trace files, one file for each MPI task. For this work we focus on extrapolating the trace data from the MPI task that consumed the most computational time, this is because this task tends to have the most impact on overall execution time. This task is identified using a lightweight MPI profiling library based on the PSiNSTracer package [26].

Given that each application is viewed only in terms of its longest MPI task, the application signature of each smaller core count is comprised as a single trace file which represents the work done on the most computationally demanding MPI task. As described in Section III-B, this trace file is a collection of information about each basic block executed by the task. For this work the trace file includes more detailed information for extrapolation and therefore contains data for each instruction of all basic blocks executed by the task. Recall that this data is represented by a feature vector. When an application is strongly scaled, each property within each basic block of the application might exhibit different scaling behavior. Some properties 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 of the instruction for the 3 core counts is used to extrapolate those values at the larger core count, illustrating the principle behind our trace extrapolation methodology: functions of various canonical form are fitted to each individual element of each feature vector and the best of

Core count	Mem. Ref.	FP op	Hit rates	Data size	Size ref.
1024	1.E8	5E7	92,99	1E8	8 B
2048	5.E7	1E7	92,100	5E7	8 B
4096	1.E7	5E6	96,100	1E7	8 B
	⇩	⇩	⇩	⇩	⇩
8192	5E6	1E6	100,100	5E7	8 B

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

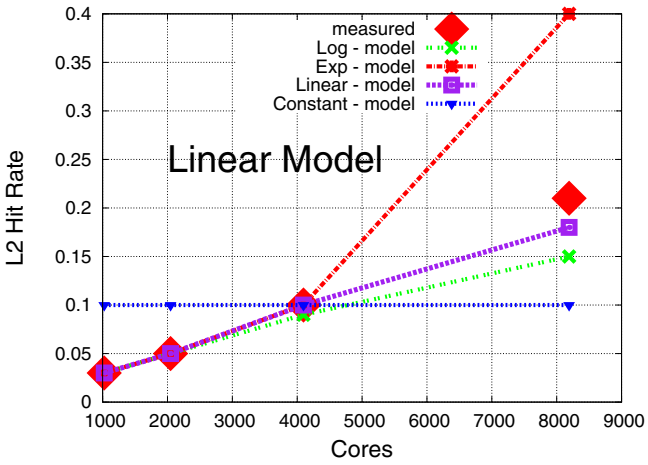

Fig. 4. Linear Model captures the scaling behavior of the L2 Hit Rate.

those fits is used.

We use four canonical forms in this work: constant, linear, exponential and logarithmic. The trace files 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 vs. 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 happens to best be described by the linear form. Each element in the feature vector of a given instruction can have different behavior 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 instructions feature vector for a single instruction and find the model that best fits its behavior and use this to generate the vector at the higher core count. This process is used for all the elements of an instruction's feature vector for all the instructions of an MPI task to generate synthetic application signature at the higher core count to be used to predict performance of the full application.

For most of the extrapolated elements this method of

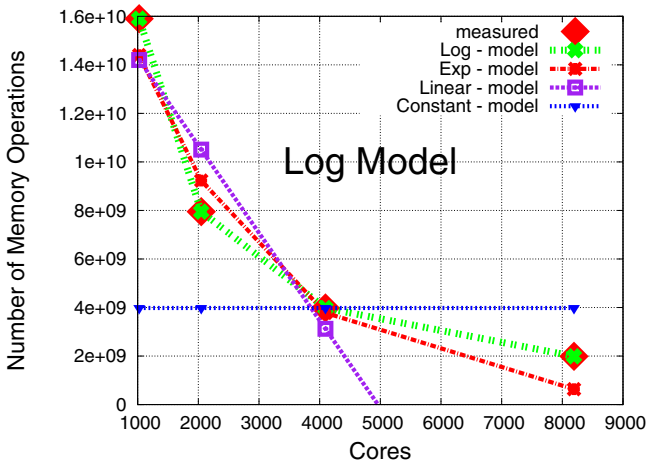

Fig. 5. Logarithmic Model captures the scaling behavior of the number of memory operations.

model fitting showed good accuracy. Most of the elements that had higher error 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 suspect that increasing the number of forms used within this methodology has a strong chance of driving down this error further, though this is a matter for future research.

V. Results

We test the accuracy of our methodology for extrapolating traces by performing the extrapolation and then using 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. UH3D 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.

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 collect an application trace at some large count (6144 for SPECFEM3D and 8192 for UH3D), then build an extrapolated trace using three smaller core counts for each application. The accuracy of the extrapolated traces are then tested by comparing the results of using the extrapolated trace and a traditional trace to produce application performance predictions for a target system. As shown in Table I, the

extrapolated traces produce runtime predictions with absolute relative errors from the correct runtime of less than 5%.

Appliation	Core Count	Trace Type	Predicted Runtime (s)	% Error
SPECFEM3D	6144	Extrap.	139	1%
SPECFEM3D	6144	Coll.	139	1%
UH3D	8192	Extrap.	537	5%
UH3D	8192	Coll.	536	5%

TABLE I
PREDICTION ERRORS FOR SPECFEM3D AND UH3D USING EXTRAPOLATED AND COLLECTED APPLICATION TRACES.

For SPECFEM3D the three core counts used to generate the extrapolated trace were 96, 384, and 1536. All traces were collected on Kraken, a Cray XT5 system at the National Institute for Computational Sciences (NICS). Traces at these small core counts were collected on Kraken and then used to generate an extrapolated trace for SPECFEM3D at 6144 cores. The trace for 6144 cores was then supplied to the PMaC modeling framework to predict the performance of the Phase I BlueWaters system. In order to test the accuracy of the extrapolated trace, another set of trace files were collected at 6144 cores and were similarly used to predict the performance of the Phase I NCSA BlueWaters system. The performance predictions for the two are shown in Table I; their differences are insignificant and both methods produce the same predicted run time of 139 seconds where the real measured runtime is 143 seconds.

For UH3D the three core counts used to generate the synthetic extrapolated trace were: 1024, 2048, and 4096. These traces were also run through the same procedure as described above for SPECFEM3D to generate an extrapolated trace file at 8192 cores. Trace files were also collected at 8192 to determine the prediction accuracy of using collected trace files verses generated trace files. The results are again shown in Table I: the extrapolated trace produces a runtime prediction of 537 seconds, which is very close to the prediction produced using the collected trace (536 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 II, 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.

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

TABLE II
CHANGES IN CACHE HITRATES OF TARGET SYSTEM AS CORE COUNT INCREASES

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 III 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.

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

TABLE III
APPLICATION TRACE DATA (L1 HITRATE) FOR SINGLE BASIC-BLOCK OF SPECFEM3D FOR TWO TARGET SYSTEMS.

VI. FUTURE WORK

Future research will add more canonical forms (e.g., polynomial) in the extrapolation of the basic block elements to improve the accuracy of the extrapolation. Applying this methodology to weak-scaled problems is also of interest, and may pose additional challenges to our methodology. We also applied the methodology only to the trace file for the longest-running MPI task within an application run, which may not be sufficient to capture how application trace data scales.

An application signature is made 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). Meaning is there groups of tasks that do similar work and as you scale the number of cores the size of the group (e.g. the number of MPI tasks in the group) also scales. The current work just uses the slow running task's prediction vector as a base to scale the data in the trace files. However, 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 has laid 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.

VII. 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 then using one of a small set of canonical functions (linear, logarithmic, exponential or constant) to describe how each of a set of important features changes as the application scales. This methodology was 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 collected traces in order to predict the performance of those applications at scale using the PMaC performance prediction framework. Extrapolating application traces in this fashion 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 by NCSA's Blue Waters project (NSF OCI 07-25070 and the State of Illinois). 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.