Beyond the Standard Model

-- Towards an integrated modeling methodology for performance and power --

Progress report for the X-Stack Meeting

Collaborative project between
Pacific Northwest National Lab, Lawrence Livermore National Lab, and UC San Diego/SDSC

Adolfy Hoisie (PI), PNNL
Kevin J. Barker (PNNL)
Greg Bronevetsky (LLNL)
Laura Carrington (SDSC)
Marc Casas (LLNL)
Daniel Chavarria (PNNL)
Roberto Gioiosa (PNNL)
Darren J. Kerbyson (PNNL)
Gokcen Kestor (PNNL)
Nathan R. Tallent (PNNL)
Ananta Tiwari (SDSC)

Performance and Architecture Laboratory (PAL)

March 2013

Executive Summary

In this project we set the stage for establishing an integrated approach to modeling of performance and power, which will take us "beyond the standard" modeling and simulation approaches that represent today's state-of-the-art. Co-design *is* modeling, in its broadest definition—so a frontal attack on the way in which we do modeling will enable practical co-design solutions among architectures, applications/algorithms, programming models and runtimes.

In this work we are pursuing four main research directions to which we believe will influence Exascale systems and applications.

Modeling of Performance and Power – we are establishing the modeling of performance and power *in concert* as the ultimate goal, beyond the current state-of-the-art in which (except for limited instances) performance only is the modeling target.

Modeling at different scales – From definition of metrics, to application models, to detailed architectural descriptions, models capture the performance and power characteristics at the various boundaries of the hardware/software stack with the desired accuracy and predictive capability needed to make the decision at hand. The integrated nature of this approach goes beyond the current state-of-the-art.

Dynamic Modeling of Performance, Power and Data Movement – This is at the heart of modeling performance and power together, and aims at going beyond the current practice that regardless of the methodology employed is static (off-line) in nature. We envision models operating in the entire spectrum from static to dynamic, the latter models serving as the engine of intelligent runtime systems, among others.

Techniques for Model Generation – This research direction aims at simplifying static model generation, including through compiler based approaches, and at coming up with methodologies for generating models dynamically based on monitoring of systems and application behavior at runtime.

Modeling methodology, in the context of this project, is the generation of quantitative, accurate mappings of application characteristics onto system resources, whose goal is to explain and predict the power and time consumed by applications running on High Performance Computing (HPC) architectures. Central to this work is the concept that future applications, HPC architectures, and all the system software layers in-between ought to be designed by taking into account the characteristics of each other, and optimized for performance under power/energy constraints.

This report provides an overview of the work in progress in this project since its inception in September 2012. The work represents a collaboration between researchers at the Pacific Northwest National Laboratory, Lawrence Livermore National Laboratory, and San Diego Supercomputing Center.

The progress to date of particular note includes:

1. Power and Performance Modeling: We have developed of accurate power models; these models utilize application characterization to predict system-level power draw. Our current work is extending that methodology to predict performance. The

power and performance models construction methodology will then be brought together within a single framework to automate the process of model development.
2. Modeling at different scales: We have extended the framework to allow precise definitions of what constitutes a computational phase within an application. These definitions and their corresponding characterization signatures will enable interaction with power and performance models at various scales of computation (e.g. loop level, function level, application level etc.).
3. Dynamic Modeling: Developed tool to measure the dependence of application performance on the availability cache storage and bandwidth. Applied tool to measure the performance properties of Lulesh and MCB co-design proxy apps.
4. Model Generation: A preliminary model generation tool based on annotations of application sources code has been developed and applied to NekBone, GTC, and Sweepd3D.

These are further discussed in Sections 3-6 of this report.

1. Introduction & Background

Multiple significant technical challenges punctuate the path to Exascale. Millions of threads of parallelism need be tractable and manageable. New algorithms may be required that can utilize these extreme levels of parallelism and can be efficiently mapped to system resources. Modeling is an ideal vehicle to explore the needs in advance of implementation and system availability as well as for use in dynamic decision-making within intelligent runtime systems.

In this project new approaches to predict, assess and model performance and power both statically and dynamically are being developed. This includes methodologies, tools, and new ways to extract insight. Required is a rational system design: co-design of systems (hardware, software, run-time systems, execution models) and applications through an accurate and quantitative methodology. We envision models will be part of the lifecycle of systems and applications from their design phases, to optimization, to production. Overall, models in all facets, area of applicability and use, and methodology base will become ubiquitous.

Our high level view of *dynamic modeling* is presented in Figure 1. The idea is that models will sit in the middle of and help all the other layers of the Exascale stack make intelligent decisions in real time. For example, the goal of minimizing data movement requires models to effectively guide runtime decisions made by the runtime software to determine when and where data should be moved/copied/recomputed. Only then can the data motion challenges be conquered and the door to Exascale be opened.

Research in Beyond the Standard Model (BSM) is structured along four areas of investigation and progress in each is detailed in this report. In section 2 we discuss the extension of application-centric modeling to include power in addition to performance. Section 3 describes how we are “bridging of gaps”, that is a general framework that allows modeling to be applicable not only statically (off-line) but dynamically (on-line) as well. The central challenge of performance-power optimization, minimization of data movement,

specifically the role of modeling in negotiating the tradeoffs among various "knobs" is discussed in section 4. Section 5 concerns itself with the easing the task of model generation.

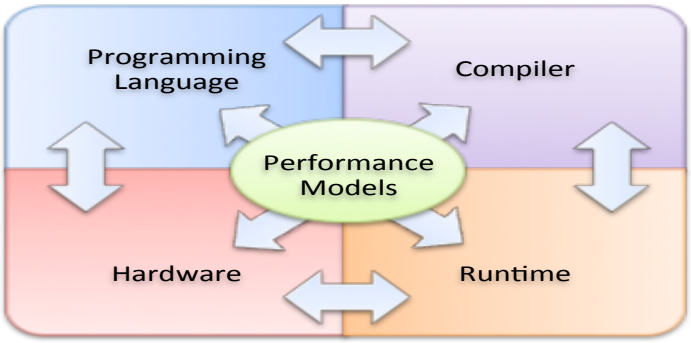


Figure 1: Performance models enable real-time decision-making between system levels.

2. Modeling Performance and Power

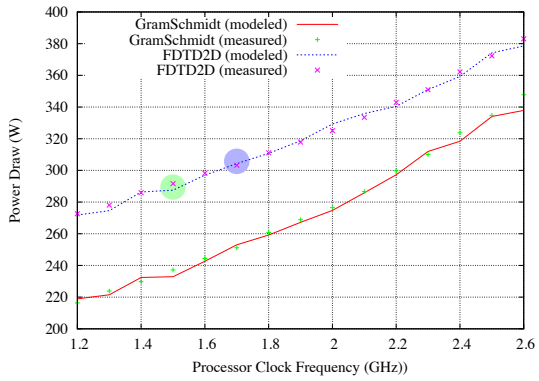
Application-specific performance modeling techniques have demonstrated their value over years of use in the areas of system comparison, application performance prediction on new and future systems, and application/system design space explorations. As the designs of systems have grown more complex and have incorporated more and more degrees of freedom (e.g., multi-core processor architectures, novel network technologies and topologies, and hybrid processor architectures), performance models have provided a capability that allows researchers and architects to quantify the impacts of design considerations well in advance of implementation. Performance models allow for a rapid exploration of a potentially large design space, allowing effort to be focused on areas that will be the most profitable. In a similar way, in this project we aim to develop methods and techniques to model the power consumption of systems under particular workloads. In particular many optimization design decisions tacitly assume that "faster is better" and that "faster" by shrinking one component of the power delay product, will usually save power too.

The energy cost of running HPC systems is growing to a point where it can easily eclipse the cost of the original hardware purchase a few years into operation. In addition, emerging systems are growing to scales that can overwhelm the power limitations of their centers. In most current systems the main way to adjust power is via Dynamic Voltage-Frequency Scaling (DVFS) and thus recently research has focused on utilizing DVFS to reduce energy usage [5]. PMaC's Green Queue framework takes a similar approach, however, what separates Green Queue from existing body of work is its methodology to create customized application-aware DVFS settings to reduce the energy required to run large scale scientific applications. Green Queue automates the capture of detailed analysis of a given large-scale HPC application to determine the energy-optimal DVFS settings for each of its computational phases. This optimization could easily be modified to optimize computations within power limits, as is anticipated for exascale systems.

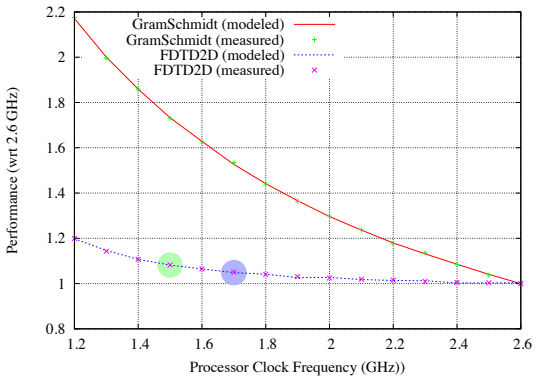
2.1 Power and Performance Modeling

In our work, we want to develop a methodology that automates the training of performance and power models. Our initial focus has been on leveraging application specific characteristics to identify energy optimal clock frequency settings for computational phases within an application. To accomplish this goal, we need to relate important computation and communication properties of HPC applications with the total system power draw and application performance degradation. This relationship, which is captured by constructing power and performance models, helps us identify and exploit the trade-offs that exist between power and performance.

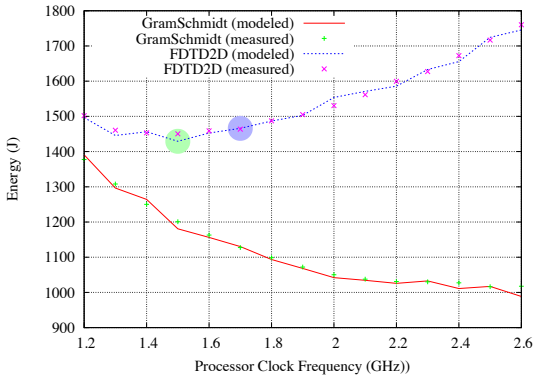
Development of power and performance models is based on benchmarking approach – utilize micro- benchmarks to "probe" a given platform and measure its power and performance response. This methodology helps to identify and isolate the important software properties that affect power draw and performance. We use a benchmarking framework called `pcubed` (PMaC's Performance and Power benchmark) [1]. The measured power and performance characterization data from the micro-benchmarks of `pcubed` serve as the training set to develop power and performance models for a system. The models can be used to explore the trade-offs between power and performance and ultimately make runtime decisions to optimize for energy or power.



(a) Power Modeling



(b) Performance Modeling



(c) Energy (Power x Time) Modeling

Figure 2: Power, performance and energy modeling for a memory bound (FDTD) and a compute bound kernel (Gram-Schmidt) kernel. Green and blue circles in the graphs provide a flavor of performance-energy tradeoff decisions that can be made with models.

2.2 Example Workflow

The process of utilizing the power and performance models of an application to identify opportunities to reduce energy consumption during execution starts by first collecting characterization data for the application using a suite of tools developed at PMaC [2]. This data is used to identify all distinct phases in an application and also to derive phase characterization data. To illustrate the idea of choosing energy optimal hardware settings through the use of power and performance models we use two highly prevalent HPC kernels – two-dimensional Finite-Difference Time-Domain (FDTD) kernel from the stencil computation domain and Gram-Schmidt kernel from the dense linear algebra domain. FDTD is a memory bound computation, i.e., number of floating point calculations done per byte of data moved from the memory is low.

Figure 2 shows a series of plots for these two kernels on a 16-core Intel Sandybridge node with 15 available clock frequencies. Figure 2a illustrates the accuracy of our power models for the two kernels. As seen in the graph, the model tracks measured power draw fairly accurately for all clock frequencies. Figure 2b shows the interaction between performance and CPU clock frequency. For Gram-Schmidt, as expected, the impact that clock frequency has on performance is fairly large. Again the modeled performance impact tracks measured impact accurately. Finally, Figure 2c combines the power and performance models to predict energy usage. The models also allow making energy and performance tradeoff decisions. For example, in the case of FDTD, one could choose to run the kernel at 1.7GHz operating frequency to only allow a 5% performance penalty while still saving 16% in energy usage – tradeoff decision graphically depicted in Figure 2c with green and blue circles.

3. Modeling and Analysis at Multiple Scales

In BSM we are exploring techniques that will bridge multiple scales from being able to rapidly observe and monitor system effects, up to exploring the impact of novel programming paradigms. Our initial work in BSM has been to develop a prototype monitoring system that has low intrusion overhead, and will form a run time environment in which other aspects of BSM, including the dynamic modeling aspects, will use. In addition we have started to analyze and model the impact of novel programming paradigms that are centered on task-based models.

3.1 High-accuracy, Low-overhead System Monitoring

In order to achieve the desired performance in future dynamic systems with the limited power budget and with a reasonable level of resilience, exascale system will be required to dynamically adapt to the changing execution environment and/or applications' execution. However, in such a large and complex context, it is prohibitive for a programmer or a system administrator to manually handle each changing event while guaranteeing resilience and contained power consumption. Exascale systems, thus, will require the development of a self-aware, self-adaptive system software stack that monitors the evolution of an application and automatically and autonomously reacts to changes in the execution environment to guarantee the correct termination of the application while respecting the power constraints.

A self-aware/self-adaptive system software works as a closed-loop system (see Figure 3) where a Monitor continuously inspects the evolution of a system through observation of Sensors. The Monitor provides feedback information that is compared to reference values: If there is a significant difference between the observed and the reference values, the Monitor notifies a Controller (self-awareness), which will apply corrective actions to stabilize the system (self-awareness).

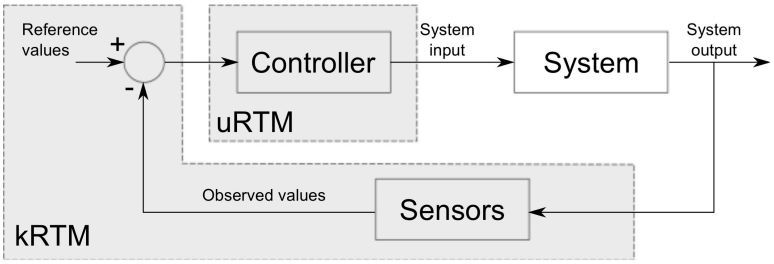


Figure 3: Overview of our system monitor containing a user-level (uRTM) and kernel-level (kRTM) runtime components.

We have developed the prototype of a novel high-accuracy, low-overhead system monitor solution that serves as the base for a model-driven self-aware/self-adaptive exascale system software. We analyzed the run-time overhead of kRTM on parallel applications when varying the monitoring frequency f from 1Hz to 1000Hz. The kRTM runtime overhead is "virtually zero" for all tested frequencies across all applications, in the sense that is impossible to measure the kRTM runtime overhead and distinguish it from the normal execution time variation (2-3%) that an application running on a real system experiences from run to run.

3.2 Model-driven task scheduling

Most task scheduling algorithms implemented in the task-based programming models' runtime focus on either data-locality or processor utilization. Although these targets seem reasonable, they may produce sub-optimal results: for example, a pure locality-driven system may produce load imbalance (low processor utilization) while a utilization-driven approach may suffer from data movement overhead. Moreover, future emerging applications are likely to have various phases during the execution and will need to adapt to changing execution environment caused by power or resilience issues, hence, task schedulers need to make a decision based on the current statue of system.

In this project, we are designing and implementing an intra-node task-scheduling algorithm based on a model-driven approach. Our novel model-driven task-scheduling algorithm will take into consideration both the computing component of each task and the communication. Rather than directly implementing one particular model-driven scheduler for a specific programming model runtime, we will first perform a study of the trade-offs of different scheduling algorithms and modeling techniques. Thus, we are designing and implementing an emulation framework that will allow us to study different scheduling algorithms. In particular, we will use a discrete event-driven simulation engine that takes as input the directed acyclic graph (DAG) of the application and the characteristics of each

task in the DAG. Both the DAG and the tasks' characteristics can be artificial/synthetic or derived from real applications. We are using different scheduling algorithms (short-job first, long-job first, data-locality aware, CPU utilization driven, work stealing) and will compare to model-driven schedulers.

We expect the model-driven scheduler to provide higher performance and be more flexible than the ones proposed in the literature. Based on the previous knowledge, we plan to implement a model-driven scheduler in one of the inter-node task-based programming models' runtime such as Global Futures, Chapel, Charm++, and compare it to the standard heuristics-based runtime schedulers.

4. Dynamic Modeling of Performance, Power and Data Movement

Dynamic modeling represents a major advancement over current modeling capability. Runtime models require the extension of our models from static off-line tools to those that can be used within a runtime software framework to dynamically guide decision making on the use of resources from both a performance and power perspective. Unlike with static predictive models, on-line evaluation is no longer concerned with the overall impact of particular analytics on a particular system; rather, they are concerned with making decisions for use in the short-term (e.g., application phase or iteration). Of particular concern is how application information can be used in the optimization of data movement, since this is expected to be a major consumer of energy.

In BSM our focus to date has been on exploring the dynamic modeling techniques, and on developing techniques to assess application memory resource use.

4.1 Developing Dynamic Modeling Methodologies

The modeling in will expand the current state-of-the-art in modeling in two primary areas:

1. Developing tools and techniques for characterizing dynamic application behavior in order to provide application-specific models with relevant inputs, and
2. Developing dynamic modeling techniques to allow models to be created and utilized during application runtime.

We envision that by deploying low-overhead data gathering tools, we will be able to dynamically inform models of on-going application behavior, allowing for intelligent runtime software to make decisions regarding performance, power/energy, and reliability optimization.

Another aspect of this work will be to develop a runtime framework capable of modeling application performance using as input behavioral information gleaned from a low-overhead instrumentation infrastructure. This modeling software will be integrated into an intelligent software system capable of modifying application behavior in response to a user-defined metric of concern, such as overall performance or power consumption. This will leverage BSM's runtime monitoring framework described in Section 2.

4.2. Active Measurement of Memory Resource Consumption.

We are developing a new performance analysis technique that captures the application's effective use of the storage capacity of different levels of the memory hierarchy as well as

the bandwidth between adjacent levels. Our approach models various memory components as resources and measures how much of each resource the application uses from the application's own perspective. To the application a given amount of a resource is “used” if not having this amount will degrade the application's performance. This is in contrast to the hardware-centric perspective that considers “use” as any hardware action that utilizes the resource, even if it has no effect on performance.

4.2.1 Measurement Methodology

We measure the application’s use of memory resources via a proactive methodology. While the application runs it uses a given fraction of each resource at a given level of a memory hierarchy (denoted ``level X cache''), such as 2MB out of a 12MB L3 cache or 5GB/s of memory bandwidth out of an available 30GB/s. If this level X cache is shared among multiple cores it is possible to measure this use by running on another core an interference thread that utilizes a known amount of cache capacity or bandwidth, where the use of a separate core limits the thread's effects to just the chosen resource.

The algorithm increases the amount of resource used until the main application's performance is observed to degrade. The difference between the total amount of the resource and the amount used by the interference thread at that point is the amount actively used by the application. If the application's performance is not sensitive to the interference then it either primarily uses a higher level of the memory hierarchy (little use of level X) or doesn't fit in level X and is thus not sensitive to reductions in its capabilities. The two cases can be differentiated by the application's miss rates for level X cache.

4.2.2 Preliminary Application Analysis

Our initial analysis has focused on Lulesh [3] and MCB [4] to characterize their sensitivity to being provided less of either of memory bandwidth or cache capacity. They are performed by running the benchmarks on one or more nodes of a Xeon20MB allocating some cores to the application and executing the bandwidth or cache capacity interference on some or all of the remaining cores. In bandwidth experiments we run 1 or 2 bandwidth threads with a buffer size of 420KB each to interfere with up to 32% of the total memory bandwidth. In storage experiments we run between 1 and 5 capacity threads with a buffer size of 4MB. In our experiments, MCB is run with 24 MPI processes and Lulesh with 64 MPI processes. An analysis of MCB is shown in Figure 4.

The top graphs of Figure 4 show in detail the performance degradations we have measured for several different mappings of the processes of MCB to compute nodes when the input set size is 20,000 particles. The x-axis corresponds to different numbers of interference threads running on the available cores. The y-axis shows MCB's execution time in this mapping and interference level. The data shows that when MCB simulates 20,000 to 260,000 particles, there is little performance degradation with one, two or three compute threads and significant degradation of 20-25% with four or five threads. The bottom-right graph of Figure 4 shows MCB's performance degradation when one or two bandwidth threads are executed to reduce the available bandwidth.

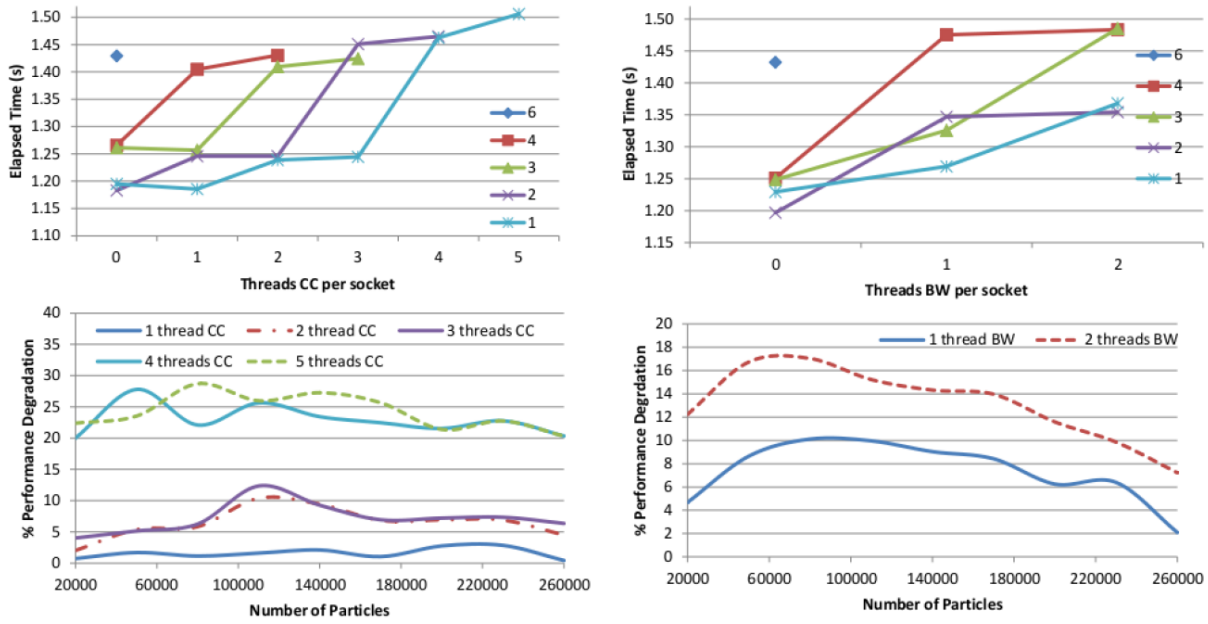


Figure 4: Performance Degradations of MCB on 24 MPI tasks

5. Model Generation

Creating analytic models of full applications is both intellectually challenging and labor intensive. The urgency of identifying methods to semi-automatically generate application models is further underscored given that reaching Exascale will require new modeling work. That new modeling work will include power models and performance models at a higher resolution than existing models.

To ease the process of creating application models, we are researching techniques and methods for tackling the problem of model generation. We define the scope of model generation widely so as to include models of performance, power, and data movement. Our goal is to see these techniques realized in tools that are intuitive enough that sophisticated domain scientists can use them with only minimal input from computer scientists.

The core of our research vision is to develop techniques for a general modeling tool that will combine top-down (human-provided) semantic insight with bottom-up static and dynamic analysis. Neither a top-down nor bottom-up approach is completely adequate by itself. Models based on high-level information often overly simplify key interactions by substituting simple for complex expressions. Tools that provide low-level information have difficulty abstracting and parameterizing that information. Our research, therefore, focuses on developing effective ways to use the two approaches in a complementary way.

Figure 7 shows the basic workflow of the model-generation tool we have been developing. The input to the tool chain is source code annotated with special modeling annotations. Given annotated source code, the modeling tool chain generates a model based on the static and dynamic mapping of annotations to program behavior. The model, which is itself a program, is a hierarchical synthesis of annotation expressions, runtime program values, and runtime measurements. When given appropriate parameters, the model generates a

prediction, and optionally, diagnostics. The modeler may then choose to refine the modeling annotations and generate another model. Once the model is done, one can perform "what if" scenarios by changing parameters or redefining, in the generated model, the expressions associated with different blocks of code.

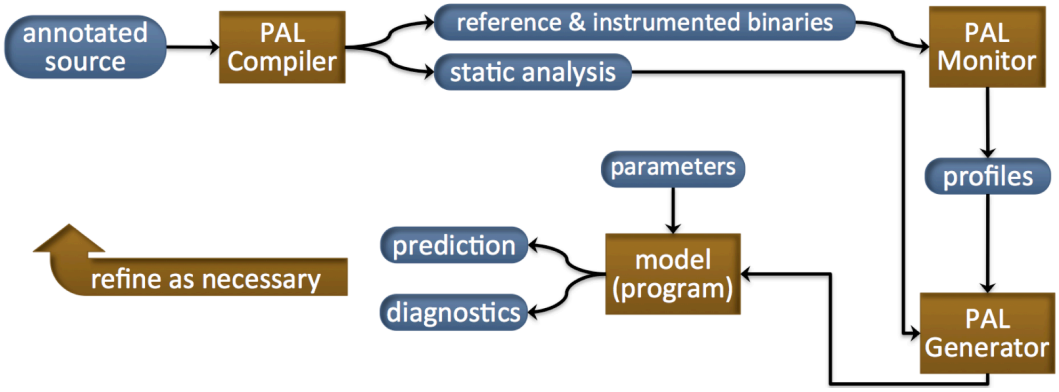


Figure 7. Workflow of PALM, PAL's modeling tools.

External Project Presentations and Publications to Date.

Presentations

"Active measurement of computational resource consumption", M. Casas, G. Bronovetsky,

"Dynamic Modeling for Exascale", A. Hoisie,

"Power and Performance Modeling for Exascale Systems and Applications", A. Hoisie

Papers

"Active measurement of memory resource consumption", M. Casas, G. Bronovetsky, in preparation.

"Designing High-Accuracy, Low-Overhead Self-Aware System Software for Exascale Systems", R. Gioiosa, K.J. Barker, D.J. Kerbyson, A. Vishnu, A. Hoisie, in preparation.

References

[1] 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, Euro-Par'11, pages 79–90, 2011.

[2] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely. Pebil: Efficient static binary instrumentation for linux. In International Symposium on Performance Analysis of Systems Software (ISPASS), march 2010.

[3] Livermore unstructured lagrangian explicit shock hydrodynamics (lulesh). https://computation.llnl.gov/casc/ShockHydro/.

[4] Monte carlo benchmark. http://www.osti.gov/estsc/details.jsp?rcdid=4793.

[5] D.J. Kerbyson, A. Vishnu, K.J. Barker: Energy Templates: Exploiting Application Information to Save Energy. In Proc. IEEE Int. Conf. on Cluster Computing, Austin, Tx, 2011.

[6] D.J. Kerbyson, K.J. Barker, D.S. Gallo, D. Chen, J.R. Brunheroto, K.D. Ryu, G.L. Chiu, A. Hoisie: Tracking the Performance Evolution of Blue Gene Systems. To appear in Int. Supercomputing Conference (ISC), Leipzig, Germany, June 2013.