A Component Infrastructure for Performance and Power Modeling of Parallel Scientific Applications

Van Bui*
University of Houston
501 Phillip G. Hoffman Hall
Houston, TX 77204, USA
vbui@uh.edu

Boyana Norris
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439, USA
norris@mcs.anl.gov

Kevin Huck
University of Oregon
120 Deschutes Hall
Eugene, OR 97405, USA
khuck@cs.uoregon.edu

Lois Curfman McInnes
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439, USA
curfman@mcs.anl.gov

Li Li
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439, USA
likli@mcs.anl.gov

Oscar Hernandez
University of Houston
501 Phillip G. Hoffman Hall
Houston, TX 77204, USA
oscar@cs.uh.edu

Barbara Chapman
University of Houston
501 Phillip G. Hoffman Hall
Houston, TX 77204, USA
chapman@cs.uh.edu

ABSTRACT

Characterizing the performance of scientific applications is essential for effective code optimization, both by compilers and by high-level adaptive numerical algorithms. While maximizing power efficiency is becoming increasingly important in current high-performance architectures, little or no hardware or software support exists for detailed power measurements. Hardware counter-based power models are a promising method for guiding software-based techniques for reducing power. We present a component-based infrastructure for performance and power modeling of parallel scientific applications. The power model leverages on-chip performance hardware counters and is designed to model power consumption for modern multiprocessor and multicore systems. Our tool infrastructure includes application components as well as performance and power measurement and analysis components. We collect performance data using the TAU performance component and apply the power model in the performance and power analysis of a PETSc-based parallel fluid dynamics application by using the PerfExplorer component.

*Correspondence should be directed to vbui@uh.edu

CBHPC 2008, October 14–17, 2008, Karlsruhe, Germany.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*domain engineering*; J.2 [**Physical Sciences and Engineering**]: Physics

General Terms

Design, Performance, Components, Standardization

Keywords

power modeling, performance modeling, Common Component Architecture, CCA

1. INTRODUCTION

Performance analysis of scientific applications is still more of an art form than a science. While the number of tools and experts in performance analysis is growing, evaluating and improving performance of parallel applications remain difficult, time-consuming, manual processes, often requiring detailed knowledge of different architectures and tools. If in addition to performance one wishes to estimate the power used by an application, the options are even more limited. Cycle-accurate simulation is a popular approach to estimating the power and energy use of computations, but only small benchmarks can be simulated [3, 69]. Furthermore, hardware for direct power measurements is largely nonexistent. Component-based software engineering enables automated code generation for various purposes based on well-defined component interfaces. This has been leveraged to provide performance measurement and analysis capabilities to component-based applications with a much better degree of automation. We make the following contributions towards this effort:

- Present a component-based framework for automated performance and power measurement and analysis.
- Design performance and power models based on performance hardware counter information.
- Implement components that provide commonly needed functionality, such as database access and manipulation.
- Describe how the performance analysis infrastructure based on PerfExplorer and the OpenUH compiler can be extended with power models to estimate power and energy consumption based on hardware counter information.
- Apply our framework and models to measure and analyze the performance and power characteristics of a PETSc-based parallel fluid dynamics application on a distributed-shared memory system.

1.1 The Common Component Architecture

Computational scientists face numerous software development challenges in parallel and distributed high-performance computing (HPC). Rapid advances and increasing diversity in high-performance hardware platforms continue to spur the growing complexity of scientific simulations. The resulting environment presents ever-increasing productivity challenges associated with creating, managing, and applying simulation software to scientific discovery. Component technology (see, e.g., [53]), which is now widely used in mainstream computing but has only recently begun to make inroads in HPC, extends the benefits of object-oriented design by providing coding methodologies and supporting infrastructure to improve software extensibility, maintainability, and reliability. The Common Component Architecture (CCA) Forum [14] is thus developing a component standard [7] for scientific computing, which includes capabilities for language-neutral specification of common component interfaces, interoperability for software written in programming languages important to scientific computing, and dynamic composability, all with minimal runtime overhead.

In addition to aiding software development, the component environment can facilitate the deployment of new computational capabilities to benefit the entire lifecycle of scientific simulation software. As components introduce an additional layer of program abstraction, performance tools are needed that support automated performance measurement and analysis. Component-based performance monitoring is currently supported by the Tuning and Analysis Utilities (TAU) performance system. Few tools support high-level performance measurement and analysis of component-based applications. To the best of our knowledge, there are no component-based systems for performance and *power* measurement and analysis. The work presented in this paper is part of a component initiative on computational quality of service (CQoS) [36,42], which helps application scientists dynamically compose, substitute, and reconfigure component implementations and parameters, taking into account tradeoffs among CQoS factors such as power usage, performance, accuracy, and reliability.

1.2 Motivating Example: The Effect of Compiler Optimizations on Performance and Power

The OpenUH [34] compiler is a branch of the open source Open64 compiler suite for C, C++, and Fortran 95, supporting the IA-64, IA-32e, and Opteron Linux ABI and standards. OpenUH provides complete support for OpenMP 2.5 compilation and its runtime library. OpenUH has been enhanced to support the requirements of TAU [49], KOJAK [39], and PerfSuite [31] by providing an instrumentation API for source code and OpenMP runtime library support. OpenUH provides a complete compile-time instrumentation module that works at different compilation phases and covers a variety of program constructs (e.g., procedures, loops, branches, callsites). We have designed a language-independent compiler instrumentation API that can be used to instrument complete applications written in C, C++, Fortran, OpenMP and MPI [20]. OpenMP constructs are handled via runtime library instrumentation, where the fork and join events, and implicit and explicit barriers are captured [12]. We leveraged the OpenUH compiler's integrated performance measurement environment in exploring the tradeoffs between performance and power as they relate to compiler optimization levels.

In this example, we studied the performance and power characteristics of a comprehensive and powerful real world simulation code, GenIDLEST [54]. GenIDLEST (Generalized Incompressible Direct and Large-Eddy Simulations of Turbulence) solves the incompressible Navier-Stokes and energy equations and implements support for multiple levels of parallelism with "virtual cache blocks" at the lowest level. "Embedded" or "hybrid" parallelism [55] is exploited by using OpenMP in each MPI process. We created PerfExplorer scripts to compute power dissipation and energy consumption estimates and analysis. Different levels of standard optimizations for the OpenUH compiler were applied ranging from O0 (all optimizations are disabled) to O3 (applies the most aggressive optimizations including loop nest optimizations). The application was run in parallel with MPI on 16 processors on an Altix 300.

The results from the study show that power dissipation generally increases with higher optimization levels, while energy consumption decreases as more aggressive compiler optimizations are applied (see Table 1). These results are consistent with previous studies that examine the effects of compiler optimizations on power dissipation and energy consumption [47, 62]. Also consistent with a previous research study [62], we found that the instruction count was directly proportional to energy consumption and that a similar relationship exists between instructions per cycle (IPC) and power dissipation. A higher instruction count translates to more work for the CPU, and so energy consumption increases. Optimizations such as common subexpression elimination and copy propagation that decrease the number of instructions are generally beneficial when compiling for energy efficiency. When compiling for power efficiency, however, optimizations that increase the overlap in instruction execution while keeping the instruction count fairly constant (and therefore increasing IPC) result in higher power consumption. Examples of such optimizations include software pipelining, instruction scheduling, and vectorization.

In the following sections, we discuss existing tool support for performance and power monitoring and analysis of

Table 1: GenIDLEST relative differences for different optimization settings, using 16 MPI processes on a 90riblet problem. Optimization level O0 is the baseline.

Metric	-O0	-O1	-O2	-O3
Time	1.0	0.338	0.071	0.049
Instructions Completed	1.0	0.471	0.059	0.056
Instructions Issued	1.0	0.472	0.063	0.061
Instructions Completed Per Cycle	1.0	1.397	0.857	1.209
Instructions Issued Per Cycle	1.0	1.400	0.909	1.316
Watts	1.0	1.025	1.001	1.029
Joules	1.0	0.346	0.071	0.050
FLOP/Joule	1.0	2.867	13.684	19.305

HPC applications. We then detail our measurement system, which includes performance and power models and related software components. We present results from our experiments applying our measurement infrastructure to explore the performance and power characteristics of an application based on the scientific numerical library PETSc. Finally, we close with a discussion of plans for future work.

2. RELATED WORK

Several tools support performance and power measurements and analyses. However, most toolsets are targeted for either performance or power, not both at the same time. There is a lack of support for integrated performance and power measurement and analysis tools, especially at the abstraction level of components. In this section, we detail the state-of-the-art tools that currently support performance and power measurments and analyses.

2.1 Performance Monitoring and Analysis Tools

Performance measurement determines what performance event is captured and in what manner. Two common forms of measurement are tracing and profiling. Numerous trace-based performance tools are available in both research and industry. Examples of tracing tools include TAU [49], SCALEA [61], KOJAK [67], Vampirtrace [43], and the Sun Analyzer [27]. Tracing can potentially incur large overheads and generate very large trace files. Profiling addresses the overhead problems that arise from tracing. A profiling technique known as *statistical sampling* mitigates data collection overheads by capturing a subset of events. Statistical sampling involves periodically sampling system counters. Performance tools such as PerfSuite [44], VTune [68], HPC-Toolkit [38], and JRockit [52] collect performance data using statistical sampling. A drawback of statistical sampling is that the performance data can be less precise and complete compared to tracing. If the sampling threshold is set to a high value, the tool will not be able to measure the events that execute faster than the threshold value. If the sampling threshold is set to a low value, large levels of overhead can result from the high frequency of sampling.

The TAU performance system [48] is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java, and Python. The primary way TAU collects measurement data from applications is by auto-instrumenting the code at the compile stage and linking in the TAU measurement runtime. TAU can measure at a granularity that ranges from the entire application down to single lines of code but commonly occurs at function and loop boundaries. Instrumentation and measurement tools such as TAU can collect detailed performance data from parallel applications.

Performance analysis tools process the performance data and transform it into a format that can be more easily accessible to the performance analyst. Since processing the data can incur a large runtime overhead, some tools process the performance data offline [24, 40, 45]. Performance analysis can also be accomplished online [2, 19, 57]. Online analysis usually supports runtime optimizations. In order for online analysis to be worth the runtime overheads, the improvements from optimizations should outweigh the overhead costs. In the case of performance analysis using low level hardware performance counters, analysis can be performed either online or offline.

PerfExplorer [24], a framework for parallel performance data mining, and knowledge discovery, was developed for analyzing on large collections of performance experiment data. The framework architecture enables the development and integration of data mining operations that can be applied to parallel performance profiles. PerfExplorer is built on a performance data management framework called PerfDMF [23], which provides a library to access the parallel profiles and save analysis results in a relational database. PerfDMF includes support for nearly a dozen performance profile formats, including TAU profiles. PerfExplorer is integrated with existing analysis toolkits (e.g., R [58] and Weka [66]) and provides for extensions using those toolkits. Both PerfDMF and PerfExplorer are free, open-source tools included in the TAU distribution.

2.2 Power Monitoring and Analysis Tools

For large-scale server systems, timely predictions of power consumption are critical for preventing thermal emergencies, reducing cooling costs, and maximizing the lifetime and reliability of the system. Power can be measured by using live measurement techniques; however, tools and necessary equipment are not readily available or easy to use. An alternative is to measure power by using thermal sensors, but this method is highly inaccurate because of the effects of thermal inertia and slow thermal sensor readings. Research has shown that power can be accurately modeled by using on-chip performance hardware counters [8, 9, 25]. Most modern processors are equipped with complete hardware-based support for performance monitoring in the form of performance counters. Thus, modeling power by using performance counters includes the benefits of low overhead and high-accuracy measurements.

Several strategies exist for reducing total power dissipated and energy consumed by a microprocessor. Power- and energy-saving techniques can be applied at the level of circuits, architectures, system software, and application layer [63]. Power analysis and optimizations at the system software and application layers have not been adequately explored, but some progress has been made in recent times. Seng and Tullsen [47] studied the effects of power and energy savings for both standard compiler optimizations and individual optimizations on the Pentium 4. Their experiments suggest that compiling for the best performance is equated with high energy savings. Valluri and John [62] performed a similar but more in-depth study on the Alpha 21264 pro-

cessor. They found that optimizations that improve performance by reducing the instruction count are optimized for low energy. They also found that optimizations that improve performance by increasing the amount of overlap in execution of instructions increase average power dissipation in the processor. LUNA [17] is a high-level power analysis framework for multicore networks-on-chips (NoCs). The SUIF2 compiler exploits LUNA for power estimation and optimizations. The COPPER project [4] applies dynamic compilation strategies for dynamic power management. Project members have introduced techniques for compiler-controlled dynamic register file reconfiguration and profile-driven dynamic clock frequency and voltage scaling. At the application layer, PowerPack [13] provides library routines that allow users to apply the Advanced Configuration and Power Interface (ACPI) in applications and reduce the CPU's processing speed via dynamic voltage scaling (DVS). Few tools provide an automated framework enabling the nonexpert to successfully apply these optimization techniques to achieve low energy consumption and power dissipation rates in their applications.

3. COMPONENT-BASED PERFORMANCE AND POWER MODELING ENVIRONMENT

In this section we introduce our hardware counter-based performance and power models that are ultimately aimed at guiding compiler optimizations and automated algorithm adaptation. Most modern processors are equipped with complete hardware-based support for performance monitoring. Major hardware vendors including Intel, AMD, IBM, Compaq, and Cray provide this base level of support for performance monitoring in their processors. In the subsequent discussion, we focus on the Itanium 2 processor (Madison 9M), but the same general approach can be applied to different architectures (e.g., we are defining and validating models for the SiCortex platform).

3.1 Performance Model

The performance model is designed to enable an application tuning cycle consisting of iterative application runs of data collection that identify locations and explanations for performance inefficiencies. The performance model presented here is based on a methodology applied to bottleneck analysis as discussed in [28] and [22]. Jarp [28] applies a methodology for bottleneck analysis using hardware performance counters. The user initially counts the most general events and drills down to more fine-grained events. Event counters are structured to enable the decomposition of more general events. Our performance model emphasizes more floating-point inefficiencies and so extends the model shown in [28] in this respect.

To identify regions of code where there is a higher then average level of stall cycles in the processor back-end, one can apply the following formula.

$$GlobalStalls = Stall\_Cycles/Total\_Cycles \quad (1)$$

For applications that are floating-point based, we derive several floating point inefficiency metrics. These metrics can be easily extended for integer-based programs.

The following metric computes the percentage of stall cycles from floating-point interregister dependencies, latencies of load instructions, and stalls at the floating-point unit.

$$\%FLPStalls = FLP\_Stalls/Total\_Cycles \quad (2)$$

The following metric identifies the regions of code that have a higher than average percentage of floating-point stalls and a high count of floating-point operations. Note that this metric is dependent on problem size and so should be used in tuning applications where the problem size does not vary.

$$FLPIneffD = FLP\_OPS* \\ (FLP\_Stalls/Total\_Cycles) \quad (3)$$

The following metric computes a problem-size-independent floating-point inefficiency value.

$$FLPIneffI = (FP\_OPS/Retired\_Inst)* \\ (Stall\_Cycles/Total\_Cycles) \quad (4)$$

These metrics are calculated by using PerfExplorer for each region being measured. The regions with the highest inefficiency are those that the programmer and compiler should focus on optimizing.

We use hardware counters to perform the memory bottleneck analysis based on the following formula.

$$Memory\_stalls = \\ (L2\_data\_ref - L2\_misses) * L2\_mem\_latency + \\ (L2\_miss - L3\_missed) * L3\_mem\_latency + \\ (L3\_miss - Remote\_mem\_access) * Local\_mem\_latency + \\ (Remote\_mem\_access)* Remote\_mem\_latency + \\ TLB\_misses * TLB\_miss\_penalty$$

$$Remote\_mem\_access\_ratio = Remote\_mem\_access/L3\_miss$$

The coefficients in this formula are the different latencies (in cycles) for the different levels of memory for the Itanium 2 processor (Madison), and the interconnection latencies of the SGI NumaLINK 4 for local and remote memory accesses. The value for remote memory latency accesses is an estimation of the worst-case scenario for a pair of nodes with the maximum number of hops and is system dependent.

3.2 Power Model

In our study of modeling processor power and energy consumption, we use PerfExplorer to compute a power metric based on [26]. Isci and Martonosi [26] construct a model of power consumption based on performance hardware counters and on several on-die components of an Intel Pentium 4 processor.We apply a modified version of this power model for a more complex processor (i.e., Madison 9M Intel Itanium 2 processor) and on a multiprocessor system at the National Center for Supercomputing Applications (NCSA). We extend the power model to use transistor counts as initial values for the architectural scaling factor instead of component area ratios [26]. The architectural scaling factor in the model is applied to the maximum power value for each component (see Eq. 5). Given that transistor activity and leakage current are important factors for both dynamic and static power consumption, respectively, we base the architectural scaling factor on transistor counts for each component. Furthermore, we simplify our power model and do not

initially take into account the power-saving effects of clock gating. Our power model is based on on-die components and initially considers the caches and core logic of the processor. Our power model metric is as follows.

$$Power(C_i) = AccessRate(C_i) * \\ ArchitecturalScaling(C_i) * MaxPower \quad (5)$$

$$TotalPower = \sum_{i=0}^{n} Power(C_i) + IdlePower \quad (6)$$

In Eq. 5, power is computed for each component (C_i) of the processor. The maximum power value is the published thermal design power for the processor. The power for each component is weighted based on the access rates for the different cache levels and core logic. Table 2 lists the access rate metrics for each component. Equation 6 computes the total power consumed by the processor and is based on the sum of the power consumed by n components and the idle power. For multiprocessor or multicore systems, the total power across all processing elements can be modeled by summing the total power computed in Eq. 6 for each processor or core.

3.3 Components

Next we describe the components involved in the performance instrumentation, data gathering, database management, and performance and power analysis. Our ultimate goal is to provide components that automate all steps in the process of collecting and analyzing performance information, enabling the definition of custom analyses, such as identification of the sources of inefficiency or estimating power and energy use.

3.3.1 TAU Performance Component

The TAU performance analysis system includes a component for use in component applications [35]. This component addresses evaluation and optimization issues in high-performance component environments, through the use of wrappers and proxies. The wrapper approach builds on TAU's dynamic performance mapping features. At the time port bindings are made by the CCA framework, it is possible to inform the port performance wrappers of the connection (i.e., user) identity. This step can occur either at the user or provider interface, or both. With this information, a performance mapping can be created to associate measured performance data with the specific service invocation. The proxy approach creates a proxy component for each component the user wants to analyze. The proxy component shares the same interface as the actual component. When the application is composed and executed, the proxy is placed directly "in front" of the actual component. Since the proxy implements the same interface as the component, the proxy intercepts all of the method invocations for the component and measures its performance.

3.3.2 Performance Database Components

We have defined interfaces for storing and querying performance data and associated metadata (see Fig. 1). The *DB* interface provides methods for establishing a connection to a database and for storing and accessing data. The data is described through the *Property* and *PropertySet* interfaces. The comparator interfaces compare and/or match properties of two problems under user-specified conditions.

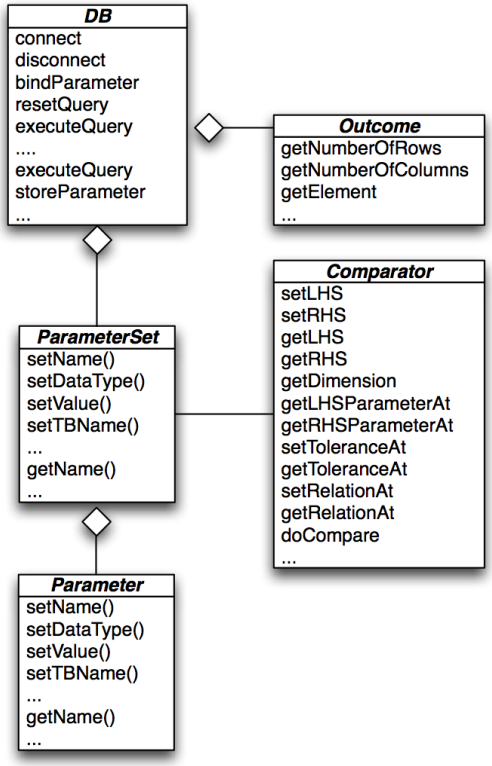

Figure 1: Database interfaces excerpt.

A set of components implements these interfaces. In our current implementation, we are using the database component mainly to store performance data into the database at the end of an application run for later (offline) analysis with the PerfExplorer component described in Section 3.3.3. In the future, we plan to provide utility components based on the basic database component (whose interfaces are close to generic SQL) to support domain-specific database operations.

3.3.3 PerfExplorer Analysis Component

We have created a simple component interface to the PerfExplorer engine. At present, the component accepts a Python analysis script as input, which contains the actual analysis implementation. The Ccaffeine GUI snapshot in Fig. 2 illustrates the current implementation. The next step is to design interfaces that allow the implementation of different components for analysis without having to rely on a separate Python script. This would enable greater flexibility, not only in terms of language choice, but in exposing more of PerfExplorer's functionality through a multilanguage API that is not restricted by Jython's limitations (e.g., having to write code compatible with an older version of Python, a smaller `os` module, and the inability to use C extension libraries).

3.3.4 The Application

In our experiments we treated the application as a single black-box component. We instrumented the entire PETSc

Table 2: Component access rate metrics.

Components	Metrics
Core Logic	Total Instructions Retired/ΔCycles
L1 Cache	L1 Total Cache Access/ΔCycles
L2 Cache	L2 Total Cache Access/ΔCycles
L3 Cache	L3 Total Cache Access/ΔCycles

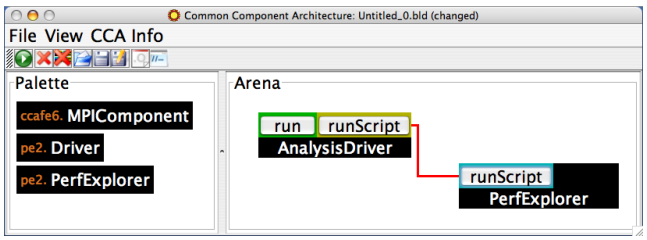

Figure 2: Example of an analysis component using the PerfExplorer component.

library with TAU and the Program Database Toolkit (PDT). Work is in progress to extend the automated performance instrumentation to the Terascale Optimal PDE Simulations (TOPS) interfaces and components [16, 50] and collect the performance data automatically through the performance proxy mechanism described in Section 3.3.1. In general, we cannot expect to have fully instrumented numerical libraries and must employ the performance proxy approach to automatically collect performance information for arbitrary computations.

4. EXPERIMENTAL RESULTS

Our case studies were conducted on the Altix 3600. The Altix 3600 is a distributed-shared memory system consisting of 256 nodes with two Itanium 2 processors per node, for a total of 512 processors. A single address space is seen by all the processors or nodes, and its global memory is based on a cache-coherent Non-Uniform Memory Access (ccNUMA) system implemented via the NUMAlink. Each node has a local memory; two nodes are connected via a memory hub to form a computational brick (C-brick). The C-bricks are connected via memory routers in a hierarchical topology. The Itanium 2 (Madison 9M) processor has 16 KB of Level 1 instruction cache and 16 KB of Level 1 data cache. The Level 2 cache is unified (both instruction and data) and is 256 KB. The Level 3 cache is also unified. The different characteristics of the main components of the Itanium 2 processor can be measured via the hardware counters.

4.1 Driven Cavity Flow Simulation

The Portable, Extensible Toolkit for Scientific Computing (PETSc) [5, 6] is a suite of numerical libraries for the parallel solution of scientific applications modeled by partial differential equations (PDEs). Our initial experiments in power issues for PETSc focus on the solution of large-scale linear systems, which often dominates overall runtime for PDE-based applications.

We profile a 2-D simulation of driven cavity flow [15], where the resulting system of nonlinear PDEs has the form

$$f(u) = 0, \tag{7}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. We have selected this model problem because it has properties representative of many large-scale, nonlinear PDE-based applications in domains such as computational aerodynamics [1], astrophysics [18], and fusion [56]. We use fully implicit Newton-Krylov methods (see, e.g., [41]) to solve Eq. 7 through the two-step sequence of (approximately) solving the Newton correction equation

$$(f'(u^{k-1}))\ \delta u^k = -f(u^{k-1}),$$

in the sense that the linear residual norm $||f'(u^{k-1}))\delta u^k + f(u^{k-1})||$ is sufficiently small, and then updating the iterate via $u^k = u^{k-1} + \delta u^k$.

As mentioned earlier, the most time-consuming portion of the simulation is the solution of large, sparse linear systems of equations. Table 3 confirms this fact – the primary sources of inefficiency as determined by Eqs. 2 and 4 are methods used in the solution (and preconditioning) of the linear systems. The table shows events for which both efficiency metrics exceed the average by more than 50% and whose inclusive time accounts for more than 50% of the total wall clock time. $I1$ designates the derived metric value for the event computed by Eq. 2, while $I1_{avg}$ is the average value for that metric over all events. Similarly, $I2$ is the derived metric value computed by Eq. 4, and $I2_{avg}$ is the corresponding average value over all events. These results are for the weak-scaling case, in which the problem size increases with the number of processors.

Table 3: Highest inefficiency regions in the driven cavity application; P designates the number of processors, and T is the total wall clock time.

P	Function	$I1/I1_{avg}$	$I2/I2_{avg}$	% of T
1	KSPSolve	2.21	8.89	56.72%
2	KSPSolve	1.73	4.22	68.52%
4	KSPSolve	1.76	3.38	79.07%
8	KSPSolve	1.52	2.27	89.53%
	MatLUFactorNumeric	1.73	2.74	55.74%
16	KSPSolve	1.26	1.52	95.63%
	MatLUFactorNumeric	1.53	1.52	71.63%
32	KSPSolve	1.26	1.52	95.63%
	MatLUFactorNumeric	1.53	1.52	71.63%

The same methods are identified as having lower than average local to remote memory reference ratios by using the memory stalls model described in Section 3.1; these are inherently memory-bound operations. The multigrid solution method performs a redundant sequential LU factorization at the coarse-grid level, so the size of the matrix being factorized on each processor increases as the number of processors

grows, resulting in the `MatLUFactorNumeric` event being one of the top sources of inefficiency for tests involving more processors. To avoid this scalability issue, one could use a parallel LU method for solving the coarse problem or could increase the number of levels in the multigrid solver as more processors are used to keep the coarse mesh size constant. Subsequently, the fine-grid linear solution method becomes the main source of inefficiency. In the remainder of this section we focus on the effect of using different linear solution methods for the fine-grid solution.

We profiled three linear system solution methods, using the performance and power models described in Section 3 to evaluate the effect of each method on the performance and power use. Weak scaling is applied in the experiments where the problem size on each processor is 16 x 16. Figure 3 shows the total execution time for the application when using each of three Krylov subspace linear solvers in conjunction with a multigrid preconditioner: FGMRES(30), GMRES(30), and BiCGS. The experiments were performed on the Cobalt Itanium cluster at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign.

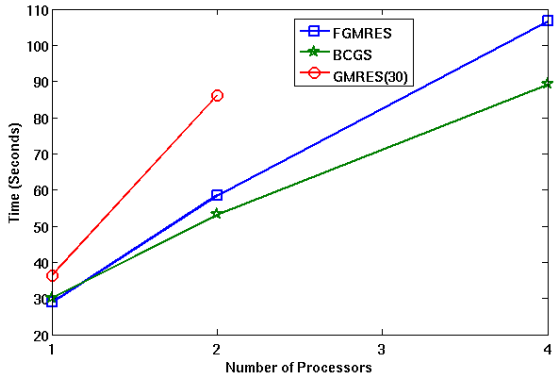

Figure 3: Total execution time of the driven cavity application. The problem size on each processor is 16×16 (weak scaling).

Figure 4 shows a metric derived from the power models described in Section 3.2. For one processor, the linear solution method with the best performance in terms of execution time (FGMRES) was not the best one in terms of power efficiency (rather, for a very small degradation of performance, BiCGS provides better efficiency in terms of FLOP/Joule). For larger numbers of processors, the best performing method among the three we tried is also the one with the best power efficiency. We note that, by default, the FGMRES method is used. The ability to easily try multiple methods and automatically evaluate their performance using any metric of interest can lead to better performance without having to dedicate large amounts of developer time to analyzing the performance of the application.

5. FUTURE DIRECTIONS: COMPILER PERFORMANCE AND POWER COST MODELING

We will continue evolving the component interfaces and corresponding implementations, both for database manage-

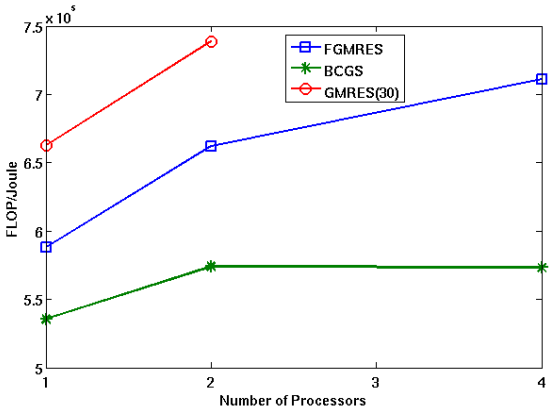

Figure 4: FLOP/Joule for the driven cavity application.

ment and for analysis. Based on our experiences with the current script-based PerfExplorer component, we will design the Scientific Interface Description Language (SIDL) interfaces for performance modeling that can eventually be used for interacting not just with PerfExplorer but with other available performance analysis tools.

Our future work will include the application of our performance and power measurement infrastructure for compiler feedback analysis in order to support more automated optimizations. Optimizing compilers often rely on analytical models of both applications and platforms to guide program transformations. The static estimate of execution cost provides support for finding optimal parameters for one optimization phase [37] and/or the best ordering of optimization phases [65]. Compiler-internal analytical models have many features in common with stand-alone performance analysis and prediction tools but are unique in that they must combine precision with high efficiency. Because of the increasing complexity of applications and hardware and the limited information available, versatile, accurate and fast compile-time cost modeling has always been a grand challenge for compiler developers.

Energy and power models with various accuracies have been proposed for processors [10, 46, 60], memory subsystems [21,30,51], buses [71], and operating systems [33]. The main method [32,59] is to quantify power consumption per instruction and interinstruction effects. Current power models work well for simple in-order, embedded processors, and an energy-aware compilation (EAC) framework [29] based on them was recently proposed. However, out-of-order superscalar processors pose significant challenges. In most cases, realistic power consumption evaluation relies on cycle-accurate simulators [3,69] extended with power models [11, 64].

The major difficulty in building effective cost models is the required deep knowledge of the applications, the platforms, and their interactions. Modeling parallel programs is significantly harder, given the nondeterministic behavior of concurrent executing threads and their interactions. Multicore and multithreaded platforms add another level of difficulty to accurate modeling, even without energy awareness. Our biggest future challenge, however, is to integrate the traditional model with a cost model for power. In practice,

model-based optimizations are widely regarded to be less successful than empirical tuning [70] except when handling small kernels like the well-studied matrix-matrix multiplication. Thus, they may be used to complement dynamic compilation and empirical optimizations.

6. CONCLUSIONS

We have described the initial implementation of component-based infrastructure for automated performance monitoring, data management, and analysis. We presented performance and power analysis results for a 2-D driven cavity simulation involving Newton-Krylov methods. Based on our early experiences with designing and implementing components for handling different tasks involved in performance and power analysis, we believe that combining the extensive functionality of tools such as TAU and PerfExplorer with component-based software engineering can potentially greatly influence the efficiency and effectiveness of performance analysis and tuning of scientific applications. The ability to easily define derived performance metrics and analyses, as well as the automation of the difficult and time-consuming tasks involved in performance data gathering and analysis, will further allow compilers to gain access to information that can lead to more effective optimizations, targeting the metrics that reflect the application developers' specific performance goals. The success of this effort will depend largely on the quality of the component interfaces for database access and performance analysis, as they must be able to eventually support a wide range of applications and tools.

Acknowledgments

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357 and in part by a CISE-BPC supplement to NSF 0444345.University of Oregon research is sponsored by contracts DE-FG02-07ER25826 and DE-FG02-05ER25680 from the MICS program of the U.S. DOE, Office of Science and NSF grant #CCF0444475. University of Houston research is sponsored by the NSF grants #CCF-0444468 and #CCF-0702775. The authors thank the following people for their discussions and implementation work: Dan Jackson, Lawrence Stewart, Canturk Isci, Sunita Chandrasekaran, and Danesh Tafti.

7. REFERENCES

[1] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keys, and B. F. Smith. Achieving high sustained performance in an unstructured mesh CFD application. In *SC99*, 1999.

[2] M. Arnold, S. Fink, D. Grove, M. Hind, and P. F. Sweeney. Adaptive optimization in the jalapeno jvm. In *OOPSLA '00: Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 47–65, New York, NY, USA, 2000. ACM Press.

[3] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[4] A. Azevedo, R. Cornea, I. Issenin, R. Gupta, N. Dutt, A. Nicolau, and A. Veidenbaum. Architectural and compiler strategies for dynamic power management in the COPPER project. In *IWIA '01: Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'01)*, page 25, Washington, DC, USA, 2001. IEEE Computer Society.

[5] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.3, Argonne National Laboratory, 2007. `http://www.mcs.anl.gov/petsc`.

[6] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[7] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou. A component architecture for high-performance scientific computing. *Int. J. High-Perf. Computing Appl., ACTS Collection special issue*, 20:163–202, 2006.

[8] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. In *ISPASS 2007: IEEE International Symposium on Performance Analysis of Systems & Software*, pages 158 – 168, April 25-27 2007.

[9] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED '05: Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pages 275–280, New York, 2005. ACM.

[10] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. An instruction-level functionally-based energy estimation model for 32-bits microprocessors. In *Design Automation Conference*, pages 346–351, 2000.

[11] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, New York, 2000. ACM Press.

[12] V. Bui, O. Hernandez, B. Chapman, R. Kufrin, D. Tafti, and P. Gopalkrishnan. Towards an implementation of the OpenMP collector API. In *PARCO*, 2007.

[13] K. W. Cameron, R. Ge, and X. Feng. High-performance, power-aware distributed computing for scientific applications. *Computer*, 38(11):40–47, 2005.

[14] Common Component Architecture (CCA) Forum. `http://www.cca-forum.org/`.

[15] T. S. Coffey, C. T. Kelley, and D. E. Keyes. Pseudo-transient continuation and differential-algebraic equations. *SIAM J. Sci. Comput.*, 25(2):553–569, 2003.

[16] D. Keyes (PI). Terascale Optimal PDE Simulations (TOPS) Center. http://tops-scidac.org/, 2006.

[17] N. Eisley, V. Soteriou, and L.-S. Peh. High-level power analysis for multi-core chips. In *CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 389–400, New York, 2006. ACM.

[18] B. Fryxell, K. Olson, P. Ricker, and et al. FLASH: An adaptive-mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophys. J. Suppl.*, pages 273–334, 2000.

[19] D. Griswold. The Java HotSpot virtual machine architecture, 1998.

[20] O. Hernandez, F. Song, B. Chapman, J. Dongarra, B. Mohr, S. Moore, and F. Wolf. Instrumentation and compiler optimizations for MPI/OpenMP applications. In *International Workshop on OpenMP (IWOMP 2006)*, 2006.

[21] P. Hicks, M. Walnock, and R. M. Owens. Analysis of power consumption in memory hierarchies. In *ISLPED '97: Proceedings of the 1997 international symposium on Low power electronics and design*, pages 239–242, New York, 1997. ACM Press.

[22] K. Huck, O. Hernandez, V. Bui, S. Chandrasekaran, B. Chapman, A. D. Malony, L. McInnes, and B. Norris. Capturing performance knowledge for automated analysis. In *Supercomputing*, 2008.

[23] K. Huck, A. Malony, R. Bell, and A. Morris. Design and implementation of a parallel performance data management framework. In *Proceedings of the International Conference on Parallel Computing, 2005 (ICPP2005)*, pages 473–482, 2005.

[24] K. A. Huck and A. D. Malony. PerfExplorer: A performance data mining framework for large-scale parallel computing. In *Conference on High Performance Networking and Computing (SC'05)*, Washington, DC, USA, 2005. IEEE Computer Society.

[25] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, December 03-05 2003.

[26] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. In *36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, 2003.

[27] M. Itzkowitz. The sun studio performance tools. Technical report, Sun Microsystems Inc., November 2005.

[28] S. Jarp. A methodology for using the itanium-2 performance counters for bottleneck analysis. Technical report, HP Labs, August 2002.

[29] I. Kadayif, M. Kandemir, G. Chen, N. Vijaykrishnan, M. J. Irwin, and A. Sivasubramaniam. Compiler-directed high-level energy estimation and optimization. *Trans. on Embedded Computing Sys.*, 4(4):819–850, 2005.

[30] M. B. Kamble and K. Ghose. Analytical energy dissipation models for low-power caches. In *ISLPED '97: Proceedings of the 1997 international symposium on Low power electronics and design*, pages 143–148, New York, 1997. ACM Press.

[31] R. Kufrin. PerfSuite: An accessible, open source, performance analysis environment for Linux. In *6th International Conference on Linux Clusters (LCI-2005)*, Chapel Hill, NC, April 2005.

[32] S. Lee, A. Ermedahl, and S. L. Min. An accurate instruction-level energy consumption model for embedded risc processors. In *LCTES '01: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, pages 1–10, New York, 2001. ACM Press.

[33] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 160–171, New York, 2003. ACM Press.

[34] C. Liao, O. Hernandez, B. Chapman, W. Chen, and W. Zheng. OpenUH: An optimizing, portable OpenMP compiler. In *Proceedings of the 12th Workshop on Compilers for Parallel Computers*, 2006.

[35] A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile. Performance technology for parallel and distributed component software. *Concurrency and Computation: Practice and Experience*, 17:117–141, Feb - Apr 2005.

[36] L. C. McInnes, J. Ray, R. Armstrong, T. L. Dahlgren, A. Malony, B. Norris, S. Shende, J. P. Kenny, and J. Steensland. Computational quality of service for scientific CCA applications: Composition, substitution, and reconfiguration. Technical Report ANL/MCS-P1326-0206, Argonne National Laboratory, Feb 2006.

[37] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.*, 18(4):424–453, 1996.

[38] J. Mellor-Crummey, R. Fowler, G. Marin, and N. Tallent. Hpcview: A tool for top-down analysis of node performance. *The Journal of Supercomputing*, 23:81–101, 2002. Special Issue with selected papers from the 2001 Los Alamos Computer Science Institute Symposium.

[39] B. Mohr and F. Wolf. KOJAK - a tool set for automatic performance analysis of parallel applications. In *Proc. of the European Conference on Parallel Computing (EuroPar)*, pages 1301–1304, 2003.

[40] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *SUPERCOMPUTER*, 12(1):69–80, January 1996.

[41] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, 1999.

[42] B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. E. Bernholdt, W. R. Elwasif, A. D. Malony, and S. Shende. Computational quality of service for scientific components. In *Proc. Int. Symp. on Component-Based Software Engineering*, Edinburgh, Scotland, 2004.

[43] Pallas GmbH. *Vampirtrace 2.0 Installation and User's Guide*, November 1999.

[44] Perfsuite. http://perfsuite.ncsa.uiuc.edu/.

[45] V. Pillet, J. Labarta, T. Cortes, and S. Girona. PARAVER: A Tool to Visualize and Analyze Parallel

Code. In P. Nixon, editor, *Proceedings of WoTUG-18: Transputer and occam Developments*, pages 17–31, March 1995.

[46] G. Qu, N. Kawabe, K. Usarni, and M. Potkonjak. Function-level power estimation methodology for microprocessors. In *Proceedings of the 37th Design Automation Conference*, 2000.

[47] J. S. Seng and D. M. Tullsen. The effect of compiler optimizations on Pentium 4 power consumption. In *INTERACT '03: Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures*, page 51, Washington, DC, USA, 2003. IEEE Computer Society.

[48] S. Shende, A. Malony, A. Morris, S. Parker, and J. de St. Germain. Performance evaluation of adaptive scientific applications using TAU. In *Parallel Computational Fluid Dynamics - Theory and Applications*, pages 421–428. Elsevier B.V., 2006.

[49] S. Shende and A. D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006.

[50] B. Smith et al. TOPS Solver Components. `http://www-unix.mcs.anl.gov/scidac-tops/solver-components/tops.html`, 2005.

[51] C.-L. Su and A. M. Despain. Cache design trade-offs for power and performance optimization: A case study. In *ISLPED '95: Proceedings of the 1995 International Symposium on Low Power Design*, pages 63–68, New York, 1995. ACM Press.

[52] T. Suganuma, T. Yasue, M. Kawahito, H. Komatsu, and T. Nakatani. Design and evaluation of dynamic optimizations for a Java just-in-time compiler. *ACM Trans. Program. Lang. Syst.*, 27(4):732–785, 2005.

[53] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press, New York, 1999.

[54] D. K. Tafti. Genidlest - a scalable parallel computational tool for simulating complex turbulent flows. In *Proceedings of the ASME Fluids Engineering Division*, November 2001.

[55] D. K. Tafti and G. Wang. Application of embedded parallelism to large scale computations of complex industrial flows. In *Proceedings of the ASME Fluids Engineering Division*, pages 123–130, Anaheim, CA., November 1998. ASME-IMECE.

[56] X. Z. Tang, G. Y. Fu, S. C. Jardin, L. L. Lowe, W. Park, and H. R. Strauss. Resistive magnetohydrodynamics simulation of fusion plasmas. Technical Report PPPL-3532, Princeton Plasma Physics Laboratory, 2001.

[57] C. Tapus, I.-H. Chung, and J. K. Hollingsworth. Active harmony: towards automated performance tuning. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[58] The R Foundation for Statistical Computing. R Project for Statistical Computing. `http://www.r-project.org`, 2007.

[59] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.*, 2(4):437–445, 1994.

[60] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. *J. VLSI Signal Process. Syst.*, 13(2-3):223–238, 1996.

[61] H. Truong and T. Fahringer. SCALEA: A performance analysis tool for parallel programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.

[62] M. Valluri and L. John. Is compiling for performance == compiling for power, 2001.

[63] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, 2005.

[64] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *ISCA '00: Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106, New York, 2000. ACM Press.

[65] D. L. Whitfield and M. L. Soffa. An approach for exploring code improving transformations. *ACM Trans. Program. Lang. Syst.*, 19(6):1053–1084, 1997.

[66] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005. `http://www.cs.waikato.ac.nz/~ml/weka/`.

[67] F. Wolf and B. Mohr. Automatic performance analysis of hybrid MPI/OpenMP applications. *Journal of Systems Architecture: The EUROMICRO Journal*, 49(10-11):421–439, 2003.

[68] A. Wolfe. Toolkit: Intel's heavy-duty dev tools. *Queue*, 2(2):12–17, 2004.

[69] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. *Design Automation Conference, 2000. Proceedings 2000. 37th*, pages 340–345, 2000.

[70] K. Yotov, X. Li, G. Ren, M. Cibulskis, G. DeJong, M. Garzaran, D. Padua, K. Pingali, P. Stodghill, and P. Wu. A comparison of empirical and model-driven optimization. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, pages 63–76, New York, 2003. ACM Press.

[71] Y. Zhang, R. Chen, W. Ye, and M. Irwin. System level interconnect power modeling. In *Eleventh Annual IEEE International ASIC Conference*, 1998.