Fine-grained Energy Consumption Characterization and Modeling

Catherine Mills Olschanowsky, Tajana Rosing, and Allan Snavely
Department of Computer Science and Engineering
University of California at San Diego, CA
cmills@ucsd.edu and {trosing, asnavely}@cs.ucsd.edu

Laura Carrington, Mustafa M. Tikir, and Michael Laurenzano
San Diego Supercomputer Center (SDSC),
San Diego, CA
{lcarring, mtikir, michaell}@sdsc.edu

Abstract

Energy costs comprise a significant fraction of the total cost of ownership of a large supercomputer. As with performance, energy-efficiency is not an attribute of a compute resource alone; it is a function of a resource-workload combination. The operation mix and locality characteristics of the applications in the workload affect the energy consumption of the resource. Our experiments confirm that data locality is the primary source of variation in energy requirements.

The major contributions of this work include a method for performing fine-grained power measurements on high performance computing (HPC) resources, a benchmark infrastructure that exercises specific portions of the node in order to characterize operation energy costs, and a method of combining application information with independent energy measurements in order to estimate the energy requirements for specific application-resource pairings. A verification study using the NAS parallel benchmarks and S3D shows that our model has an average prediction error of 7.4%.

1. Introduction

Energy costs constitute an increasingly significant fraction of high performance computing (HPC) resource costs. This increase makes energy costs a major consideration in resource acquisition decisions. Acquisition committees weigh cost and performance tradeoffs in order to choose the best HPC resource for an anticipated workload[1]. Informing such choices requires evaluating the performance and energy-efficiency of candidate resources.

Predicting the energy consumption of an HPC resource requires understanding how the workload will utilize it. Each application in the workload potentially stresses different components of the resource. Since the components consume energy at different rates, the overall energy consumption changes with workload. For instance, a computationally-intensive application operating exclusively out of L1 cache will not require the activation of any memory units (DIMMs). Deactivating the DIMMs lowers the node energy consumption rate significantly.

In this paper, we present a modeling technique capable of informing resource acquisition decisions. Our model combines resource information with application information, and provides an energy consumption estimate. Resource information is collected using a custom harness that records energy consumption information during specialized benchmark execution. Application information can be collected on any x86/Linux or PowerPC/AIX system.

Performance predictions are already part of the Department of Defense (DoD) HPC Modernization Program Office (HPCMPO) supercomputer acquisition process,[2] and energy consumption predictions are the obvious next step. Our energy consumption predictions provide valuable insight to acquisition committees faced with difficult decisions involving resource tradeoffs. For example, a comparison of up-front hardware cost versus long-term (energy) costs is enabled by our models.

Our approach provides the ability to perform cross-architectural predictions. A cross-architectural prediction is one that does not require the application to be run on the target resource (the resource under evaluation). The implications of this ability with respect to resource evaluation are far-reaching. For example, if you have 10 machines and 10 applications, you would normally be forced to make 100 energy measurements (assuming such measurements were possible); cross-architectural predictions require only 10 measurements plus 10 application characterizations.

978-0-7695-4392-5/10 $26.00 © 2010 IEEE
DOI 10.1109/HPCMP-UGC.2010.54

More importantly, if the target machine does not yet exist at full-scale, benchmarks can be run on a single prototype node, and actual application energy consumption can be predicted. Furthermore, if the machine doesn't exist at all, but the energy consumption can be forecast on our simple benchmarks, our methods allow reasonable estimate of real applications energy consumption.

Our models also allow for the early evaluation of energy-proportionality for specific workloads. System designs that successfully balance application requirements and computing capability in order to maximize energy efficiency are referred to as energy-proportional designs[3]. Each application on a given resource may have a different level of energy-efficiency; this is due to the varying behaviors of HPC applications. Thus, a design that is energy-proportional for one application may be inadequate for another. This implies that energy-proportional designs must be workload-specific or that hardware must dynamically adapt to the workload.

Data movement is a crucial factor in any energy-proportional design. Experiments confirm that variations in per-operation energy costs in memory-bound applications depend mainly on data-movement. Most HPC applications are memory-bound, meaning data-movement limits performance[4,5]. The data-movement determines, to a large extent, which components on the node are active and at what level. The variations in terms of components, active and power-consumed, on today's resources are significant, depending on workload. Future systems are projected to have even more variability as they employ dynamic methods to manage power and performance.

Kogge et al.[6] predict that the cost-per-memory access in an exascale machine will be an order-of-magnitude smaller than current cost due to technological advances. However, the *total* energy costs will continue to increase dramatically, both in absolute terms, and as a fraction of total ownership costs, as datasets become larger and move further away from the processor.

Figure 1 shows the anticipated typical performance and energy costs of accesses to each level of the memory hierarchy in future architectures. The small register file (represented as the top of the triangle) offers the cheapest access to program operands, in terms of both performance and energy. Moving down the triangle, as the dataset grows and operands reside further away from the processing core, the cost of each access increases dramatically, energy cost per operand transfer increases even more quickly than the performance cost (from 40 to 3,200 pJ per operand)[1].

[1] Projected energy per access numbers courtesy of Bill Dally (Stanford University) via personal correspondence.

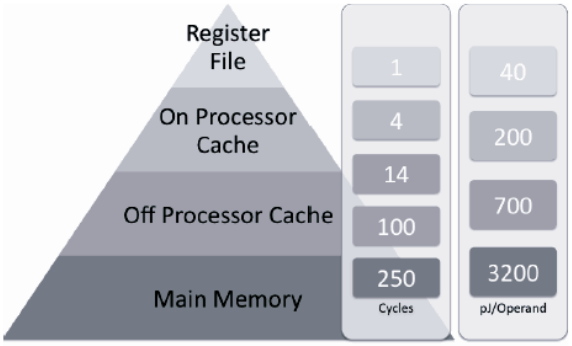


Figure 1. Anticipated operand fetch cost on future architectures as a function of location in memory hierarchy

The major contributions of this work include a method for performing fine-grained power measurements on an HPC resource, a benchmark infrastructure that exercises specific portions of the node in order to characterize operation energy costs, and a method of combining existing application information in order to estimate energy requirements for specific application-resource pairings. The energy consumption predictions in the included verification study have an average prediction error of 7.4%.

This paper is organized as follows: Section 2 presents the power measurement harness and initial findings; Section 3 describes the application signature and modeling approach; Section 4 presents the measurements and a verification study of the model; Section 5 presents related work; and conclusions are in Section 6.

2. Characterizing Energy Consumption

Our energy consumption characterizations, referred to as *machine energy profiles*, focus specifically at on-node power consumption. This includes the CPUs, motherboard, memory and input/output (I/O) controllers, memory, fans, hard disk, and power supply. On-node power consumption is a large percentage of the total system power consumption, and modeling it is a necessary and significant step toward full system modeling.

The CPU and the memory subsystem dominate the energy requirements of the system as a whole. Total system power consumption is composed of three components, reflecting the architecture of the HPC resource: 1) on-node, 2) interconnect, and 3) provisioning and cooling. According to the exascale report[6], the on-node power consumption represents approximately 50% of the overall consumption. The energy demands of the cooling system correspond directly to the variations in on-node energy requirements[7]. The network power consumption represents approximately 27% of the total,

with an estimated 20% of that taking place on-node (which our methodology captures).

Our metric is a measure of energy for each operation i.e., *Watts∗s*/*operation* or in other words *Joules*/*operation*. Not all operations require the same amount of energy; energy consumption is expressed as a function of operation mode and operand location. The operation modes we use are floating-point or integer. Significant and consistent power consumption trends are apparent with respect to operation and working-set size.

The energy consumption profiles presented in this section characterize the energy consumption required to execute operations whose operands are resident in each level of the memory hierarchy. Specialized hardware and software produce the energy consumption characterization. The hardware, referred to as the *power measurement harness*, is physically attached to a single node of the target system and records power measurements, while the software exercises specific components of the system.

A. Power Measurement Harness

Energy consumption rates can be measured by monitoring one of two signals, alternating current (AC) or direct current (DC). A standard 220-volt AC supplies the compute-node with power. The AC signal is converted to a DC signal at a variety of voltage levels by the power supply. Measurements taken on the AC signal provide a high-level, but coarse-grain dataset. Measurements taken on the DC signals provide a more fine-grained, but less complete dataset. Our measurement harness utilizes both types of measurements.

Compute nodes on HPC resources present significant DC measurement challenges. Specifically, the nodes are small and tightly-packed into a rack. Several compute nodes will share a single power-supply, and intercepting the DC signal is not always possible. In order to overcome these challenges, the DC measurements can be taken on a separate system that uses the same key components. The components that need to be consistent are the CPUs and the memory units. The DC measurements taken on the separate machine are verified by comparing them to AC measurements taken on the compute-node.

The energy consumption rates of specific, highly-variable components are measured by monitoring their DC signals. The DC signals are redirected through a set of current-sensing resistors connected to two areas of the compute-node: first, the sockets providing power to the CPUs and second, each memory bank (DIMM). These components were chosen because they are the source of a vast majority of variation in the energy consumption rates.

A current-sensing resistor has a small resistance and is placed in series with an existing circuit. High-side measurements, meaning that the resistor is between the power supply and the load rather than between the load and the ground, are used in order to obtain accurate current measurements. The voltage drop across the current-sensing resistor is measured, and Ohm's law is used to calculate the current. Power consumption is calculated using Equation. 1. R_{ld} is the equivalent resistance of the load, in this case, the CPU or the DIMM.

$$P = I^2 R_{ld} \quad (1)$$

The form of the power measurement harness is shown in Figure 2. The left-side of the figure labels each measurement point numerically. The right-side shows the circuit diagrams for how each of the measurements is taken. Measurement one is the AC measurement, and is taken using an external device, the WattsUp?[8] Power Analyzer/Data Logger. All measurements, except for the AC measurement, are taken and recorded using a Data Acquisition Device (DAQ), the NI-DAQ 6255[9]. The NI-DAQ 6255 is able to measure up to 40 differential channels at a sampling rate of 1.25 Mega-Samples/second with 16-bit accuracy. The input range is between −10 and 10 volts.

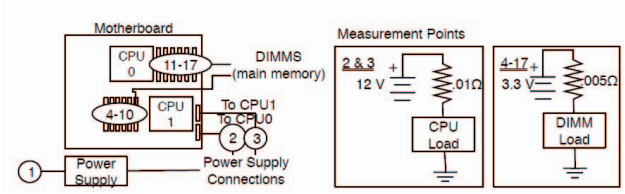


Figure 2. The measurement harness

The DAQ provides a time-series dataset describing the voltage-drop across each resistor. These values are averaged, and the result is the average power consumption rate (Watts) over a period of time. The period of time is set in the software. A software call is made that triggers a voltage to turn on and off. The voltage is controlled via USB, and is used as a hardware trigger to the DAQ indicating that measuring should start or stop. This trigger mechanism allows for accurate power consumption rate sampling to take place during benchmarking. The USB control is implemented using a Phidgets USB interface kit[10].

B. Power Measurement Benchmark

The goal of the power measurement benchmark is to activate each level in the memory hierarchy in a controlled manner, and exercise different functional units at the same time. This enables power measurements to be taken during very specific and known operation modes. The memory benchmark MultiMAPS[11] has been expanded to accommodate these requirements.

MultiMAPS is implemented as a loop that steps through a dataset. The data-set size and the step-size (stride) are varied to explore the performance of the memory subsystem. The memory bandwidth achieved during the execution of the loop is recorded for each dataset size and stride pair. The size and stride pair is measured for approximately two seconds, a long enough period to allow for power measurements to be taken.

For this work, MultiMAPS was expanded in order to explore the energy costs associated with separate functional units. The original benchmark performed one floating point operation per memory access. This work uses two additional variations: first, a variation that does no floating point operations, but focuses only on the integer unit; and second, a version that does four floating point operations per memory access.

3. Energy Consumption Model

The energy consumption model provides a mechanism to estimate the energy requirements of an application-resource pair without ever having executed the application on the resource. Informing acquisition decisions requires the ability to make cross-architectural predictions, since often full-scale implementations of the offered resources have not yet been built and executing full-scale applications is thus not possible.

Conducting an evaluation of hardware choices for a given workload with respect to energy consumption requires an $O(m*n)$ process, if the resources can be accessed. As an illustration, assume that a given workload is composed of 10 applications and that vendors are offering 10 resources. A full evaluation requires that all 100 application-resource pairs be tested, and energy data collected. Our models make it possible to trace the applications once (not even on one of the newly-offered systems, but on any available system), and take an energy profile of each offered system once. That data can be combined using our modeling framework offline, reducing the complexity to $O(m+n)$.

Energy consumption management is becoming such a focal point for HPC systems that future resources, such as the Intel Rock Creek, will have fine-grained energy monitoring and management policies. On systems with these capabilities, creating energy profiles will no longer require external hardware, making the data much easier to collect. Self-aware applications that are given this profile will be able to manage the system power levels in the most energy-efficient manner possible. Our models have the potential to inform decisions of this type.

The energy consumption model follows the PMaC Prediction Framework[11]. The tracing infrastructure is used directly, while a new machine profile and model have been developed.

A. Performance Modeling Framework

The PMaC Framework[11] is a well-established HPC performance modeling toolkit. It is utilized by the Department of Defense (DoD) to evaluate potential resource acquisitions and is consistently accurate, averaging 10% error, for full-scale HPC applications. The framework is specifically designed to perform cross-architectural predictions, and has been shown to be scalable to long-running HPC applications.

The framework separates communication and computation into separate models. The communication model represents the application as a trace of communication events. Between communication events, work is performed on the processor. These bursts of work are referred to as CPU bursts and they are summarized in the computation model.

The communication and computation models are broken down using similar principles. Figure 3 shows the breakdown of each into three main categories: a) *application signatures* which are detailed but compact representations of the fundamental operations inherent to an application collected via trace tools, b) *machine profiles* that represent the capability of systems to perform fundamental operations measured via benchmarks, or estimated via system specifications, and c) *convolutions*[2] that rapidly combine application signatures with machine profiles via simulation to predict performance. In this work we focus on the computational model component (the shaded portion of Figure 3) of the PMaC framework to develop our energy consumption model.

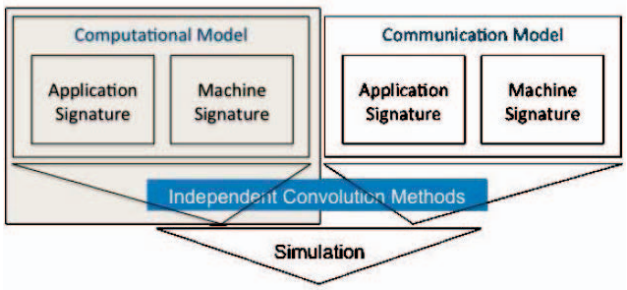


Figure 3. Overview of the PMaC Prediction Framework

The application signature for the computational model is obtained by tracing the application. It consists of the memory address stream, the floating point operation count, and the memory operation count for each basic block in the application. An address stream is the list of memory addresses requested by the application, and a basic block is a unit of code that has a single entry and exit point.

[2] The term convolution is used to refer to the processes used to combine the application signature and machine profiles together to achieve a performance prediction.

The address stream is sent through a cache simulator on-the-fly (during collection). The cache simulator simulates the memory hierarchy of the target resource. Collection is achieved using binary instrumentation[12,13]. The cache hit-rates for each level of cache, as well as the collected operation counts, compose the saved application signature.

The machine profile is gathered using benchmarks and/or machine specifications depending on availability of the machine hardware. The benchmark, MultiMAPS[4], is run on the target machine and collects information on the achievable memory bandwidths for each level of the memory hierarchy. The benchmarks used to collect the machine profile of the target system are small and simple. They only need 1–2 nodes of the target system to be available.

The convolution method combines the application signature with the machine profile in order to produce an estimated execution time for each basic block of the execution. The process of determining the achieved memory bandwidth and execution time is described in detail in Reference 4.

The performance model provides insight to the data movement rates that are achieved by the application, and indicates where in the memory subsystem the working set for each basic block resides. This information is exported to the energy consumption model for the work done in this paper.

B. Energy Model

The energy model combines an application signature from the PMaC framework, with a machine energy profile in order to predict the energy (joules) required for execution. The model is a mathematical combination of the two sets of data.

The model is created at a basic block level. The total energy cost of each basic block is predicted based on the average energy cost-per-operation and the visit count (number of times the basic block was executed) information. The energy cost of each basic block is summed in order to get a full execution energy cost estimate.

The obvious first attempt at such a model is a simple linear combination of cache hit-rates and operation costs described by Equation. 2. E_{bb} is the energy cost-per-operation in basic block, H_{Ln} is the hit-rate for the n^{th}-level of cache and E_{Ln} is the energy required to access the n^{th}-level cache:

$$E_{bb} = H_{L1} * E_{L1} + H_{L2} * E_{L2} + H_{L3} * E_{L3} + H_{MM} * \mathrm{E}_{MM} \quad (2)$$

The cache hit-rates are available for each basic block in the application signature. This simplistic approach does not capture the realities of energy consumption in a node and, therefore, produces predictions with errors as high as 85%. The majority of the power consumption variation comes from the powering-up and down of main memory. This model ignores the fact that DIMMs are not powered-up and down on a small time-scale.

As an example, consider an access pattern that requires a cache-line be loaded from main memory. Once in main memory, the line is accessed three more times. This yields an L1 cache hit-rate of 75% with the remaining 25% of accesses going out to main memory. It is unlikely that main memory will be powered-down during the three L1 cache hits and, therefore, during those accesses the higher main memory power rate is used, but for a shorter amount of time than a main memory access. This indicates the need to change the energy consumption rate when main memory accesses are present.

The lasting effect of DIMM activations are modeled by separating the operation rate from the energy consumption rate. The energy consumption rate is assumed to remain constant for some time after a main memory access. The energy consumption rate during a main memory access is used in conjunction with the data rate achieved by L1 cache to calculate the cost of accessing L1 after a recent main memory access ($E_{L1/MM}$). This pattern is followed to calculate similar values for all levels of the memory hierarchy expressed as $E_{Ln/Lm}$. *Ln* is the level of cache being accessed and *Lm* is the level of cache that the energy consumption rate is associated with.

The energy consumption rate for each term in the equation is found by determining which level of the memory hierarchy dominates energy consumption-wise. To achieve this, we dilate each cache hit-rate by multiplying it by a weight. The weight is proportional to the memory bandwidth achieved by L1 cache. For instance, if L1 is twice as fast as L2, the weight for L2 is 2. The weight for L1 is always 1.

Table 1 shows an example of the weighting scheme applied to a few basic blocks. The first basic block has an L1 hit-rate of 50%, and the remaining accesses take place in main memory. After the weighting scheme, it is obvious that the main memory energy consumption rate is dominant in this case.

Table 1. An example of the weighting scheme used to choose the dominant level of the memory hierarchy

Simulated				Weighted			
L1	L2	L3	MM	L1	L2	L3	MM
50.00	0	0	50.00	50.00	0	0	885.0
66.66	0	0	33.33	66.66	0	0	590.9
98.62	0	1.37	0.002	98.62	0	6.16	0.04

The weighted values are used to choose the term P_{LD} the power level of the dominant cache level. In this equation, *LD* refers to the dominant memory hierarchy level. If the power consumption rate of the current level

of cache is higher than that of the dominant level, the current level is used, e.g., max (P_{L2}, P_{LD}).

The energy consumed accessing each level of cache is summed to produce the energy consumption for the entire basic block. The hit rate for each level is multiplied by a new energy term. The energy term is determined by multiplying the power consumption rate of the dominant level of cache by the rate of access achievable in that level, as seen in Equation 3:

$$E_{L1,LD} = P_{LD} * (1/BW_{L1}) * bytes \quad (3)$$

$$E_{bb} = H_{L1} * E_{L1,LD} + H_{L2} * E_{L2,max(L2,LD)} + H_{L3} * E_{L1,max(L3,LD)} + H_{MM} * E_{MM,MM} \quad (4)$$

The energy requirements for the entire application are calculated by summing the energy requirements of each of the basic blocks in the execution. The basic block frequency information is retrieved from the PMaC application signature.

The goal of the energy model is to achieve accuracy without becoming overly complicated. Our verification study shows that this model has an average error of 7.4%.

4. Experimental Results

The goal of this work is to accurately model the energy consumption of HPC resources. A necessary step to achieving this goal is identifying the major causes of variation in energy consumption rates. This section begins by presenting experimental data showing that we have captured the variation, and then moves on to present a high-level verification study using HPC benchmarks.

All of the experiments described in this section were conducted on Dash at San Diego Supercomputer Center (SDSC). Dash has two main types of nodes: 64 compute nodes and 4 dedicated IO-nodes. Both node-types contain two quad-core 2.4GHz Intel Nehalem Processors and 48GB of DRAM (12 4GB DDR3 DIMMs).

A. Identifying Energy Consumption Rate Variations

The variations in energy consumption rates are captured by the DC power measurements. We show that the CPUs and DIMMs are the main source of this variation. By measuring this subset of node components the energy consumption of the entire node can be characterized.

All of the power consumption data is presented using a format based on a MultiMAPS curve. Figure 4 is the MultiMAPS curve for a Dash-node. The Y-axis is the achieved memory bandwidth and the X-axis represents a series of working-set sizes increasing to the right. The curves shown represent accesses with strides 2, 4, and 8.

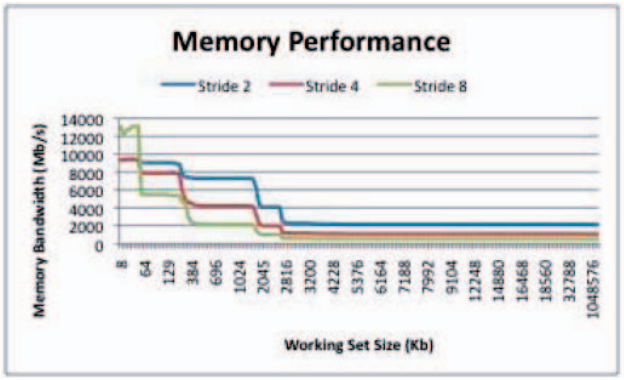


Figure 4. The memory bandwidth achieved by MultiMAPS

Each plateau in the curves represents a different level in the memory hierarchy. The Nehalem processor has L1, L2, and L3 caches, and the first three plateaus correspond to those. The rest of the curve represents main memory. There are four performance levels in main memory (only two are easily visible at this scale).

Recall that the machine energy profile is created by running three versions of the MultiMAPS benchmark on Dash. The original has one floating-point operation corresponding to each 8-byte load from memory (labeled floating point). Additionally, a version with an extra floating point add-multiply operation (labeled floating-point+), and a version using only integer operations (labeled integer) were run. The performance and energy consumption rates are combined to create the profiles.

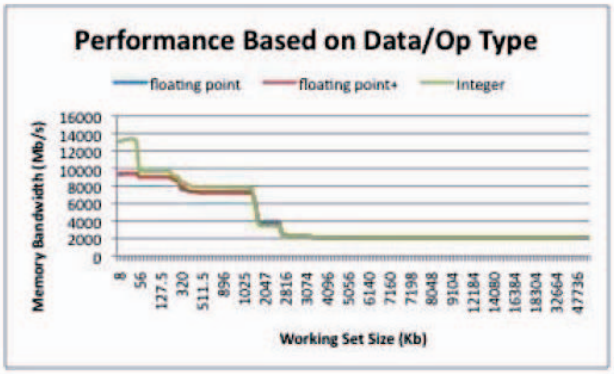


Figure 5. Performance difference between integer- and floating point operations

Figure 5 shows the performance for the stride 4 case of each. The floating point and floating point + cases match closely. The integer case is faster, especially when in L1. Similar results are observed for the stride 1, 2, and 8 cases as well.

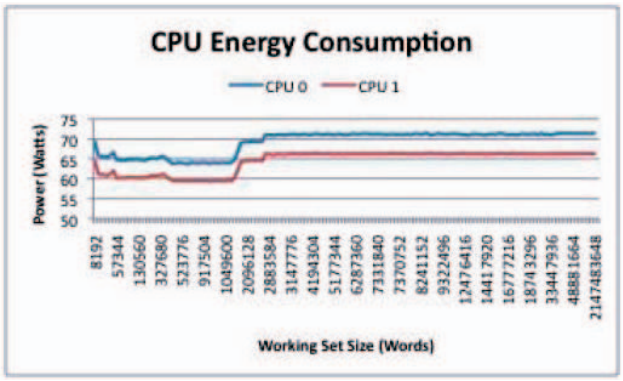


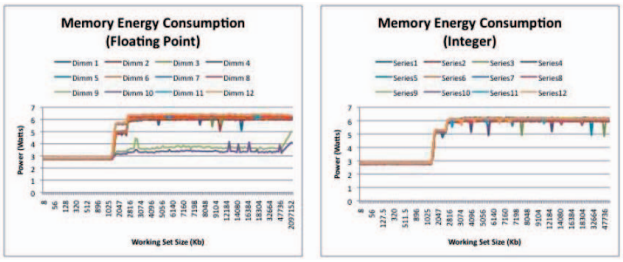
Figure 6. Performance difference between integer and floating point operations

The variations in CPU energy consumption depend both on the instructions being executed and the operand location. As with performance, the energy consumption rates for the floating point and floating point + cases matched very closely. The energy consumption rate of the CPU during the integer benchmark execution varied only slightly. In the L1 case, the energy consumption rate increased by 2 Watts during integer execution.

Figure 6 shows the CPU energy consumption rates for MultiMAPS (floating point). The consumption patterns for both CPUs are the same, but at different levels. One of the CPUs consistently consumes 5 Watts more. We believe that this difference is due to the QuickPath Interconnect.

The CPU energy consumption data is impacted by the power management policy used on the node. During machine profiling, the frequency throttling mechanism was enabled. Several governors (policies) are available for this processor; the on-demand governor is the default and is used for all measurements. The on-demand governor does not throttle frequency during application execution; dynamic voltage frequency throttling can be used to increase energy-efficiency during execution[14].

The energy consumption rate of the DIMMs changes with the memory bandwidth as well. Figures 7a and 7b show the changes for two cases. The first is MultiMAPS (floating point) run on a large data-set. The DIMMs remain in a standby-state consuming less than 3 Watts until the working-set no longer fits in cache. The number of DIMMs activated and the level that they are activated to depends on where the data was allocated. The data-set allocated for the second figure is smaller than the first, but it was laid out over all of the DIMMs, and therefore requires that they all be active during execution.



Figures 7a and 7b. Energy consumption rates for the stride 4 multiMAPS case on Dash

Figure 8 confirms that the targeted components of the compute-node are, in fact, producing the variations in power consumption. The solid sections at the bottom of the graph represent the sum of all of the measured DC power components. The DC measurements were taken on an IO-node of Dash. The top line is the AC power measured on a single compute-node of Dash. Before powering on the node, the constant power demands of the sub-rack were measured at 174 Watts. The sub-rack contains three power supplies, each of which has some operational overhead, as well as networking infrastructure. This shows that we are capturing all but a constant 65 Watts. This energy is being consumed by the motherboard and other components such as disk and fans. The majority of the variation is being captured within the DC measurements.

Most importantly, Figure 8 shows that DC measurements taken on matching configurations, even if they are running on a different motherboard, can be used to characterize the compute-node power consumption. All of the DC measurements are taken on the IO-node, which has the same CPUs and DIMMs as the compute-nodes, but in a more accessible configuration. The AC measurements are taken outside of the sub-rack that contains the power supply and the compute node. For the measurements, only a single compute node was active.

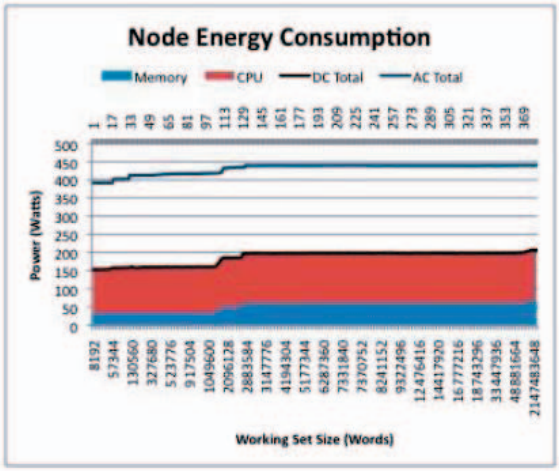


Figure 8. The CPU and node energy consumption rates for Dash

Table 2 shows the energy consumption values used for the machine energy profile. The energy required per integer operation is significantly lower than those required for floating point operations. The largest variation is found during the step from L3 cache to main memory during floating point execution.

Table 2. The power and energy components used by the energy model

	Floating-Point				Integer			
	CPU Watts	Memory Watts	Total Watts	nJ/op	CPU Watts	Memory Watts	Total Watts	nJ/op
L1	120	33	153	126	126	33	159	92
L2	125	34	159	225	127	33	160	127
L3	124	34	158	576	130	33	163	158
MM	137	69	206	2,965	142	73	215	788

B. Model Verification

The energy model should be capable of accurately predicting the energy consumption of an application run on a specific machine, without actually running it there. This implies the application signature should be generated on a machine other than the machine that the energy profile was created on. The main advantage to our approach is that the machine characterization only has to be done once per machine and the application characterization only has to be run once per application; one can quickly generate the cross-product of the two sets of characterizations.

We verified the accuracy of our energy consumption model by predicting the energy consumption of a set of HPC benchmarks on Dash. Dash has dual quad-core Nehalem processors and therefore, each benchmark was executed on 8 cores. The set of benchmarks and applications chosen for verification includes a subset of the NAS parallel benchmarks (NPBs)[15] and S3D. The NPBs are each run using two data-sets, A and B. A is the smallest NPB data-set available and B is the next-largest. This enables us to see the difference in how the model handles larger working-set sizes. S3D solves the full-compressible Navier-Stokes equations that describe the conservation of mass, momentum, and energy, and laws of gas behavior, while simultaneously tracking the evolution of reactive species on a rectangular mesh. It uses the Message Passing Interface (MPI) to efficiently distribute the calculation among parallel processors, and is built on a hierarchical, modular structure.

The machine energy profile collected from Dash is shown in Table 2. The model used only the energy consumption values collected from the floating point version of MultiMAPS. The DC measurements were taken using one of the dedicated IO-nodes. The benchmark measurements are also taken using DC measurements.

The predictions are cross-architectural predictions. The application signatures were not collected on the target resource. The S3D signature was collected on a Power 4 resource using pmacInst[12], and the NPB signatures were collected on an SGI ICE system using pebil[13]. The overhead for collecting application signatures is less than a 10X-slowdown.

The energy consumption of each of the benchmarks was measured during execution on the Dash IO-node. The same DC measurement harness was used during the benchmark measurements. The average energy consumption rate was measured during execution between the MPI initialization and finalization. The execution time (between MPI initialization and finalization) was multiplied by this rate to produce an overall energy consumption value.

Table 3 shows the results of the model. The DC power consumption of the node varied between 154 (EP.A) and 227 (S3D) Watts. This variation implies that a large range of conditions was covered. A consumption rate of 154 Watts is only achieved during operation exclusively out of L1 cache, and a rate of 227 Watts implies that all of the DIMMs were active at their highest level. This was not recorded in the execution of the floating point benchmark, which explains why the prediction of S3D was low.

Table 3. Measured execution time, energy consumption rate, and energy consumption compared to predicted energy consumption

	Measured			Predicted	Relative Error
Benchmark	Time sec	Power watts	Energy joules	Energy joules	Energy
CG.A	0.84	212	180	162	10%
CG.B	22.5	211	4,762	4,550	4.5%
EP.A	2.1	154	324	337	-3.9%
EP.B	8.0	156	1,251	1,342	-7.3%
FT.A	1.8	200	361	361	-.01%
FT.B	19.5	199	3,886	4,085	-5%
IS.B	2.3	192	445	369	-17%
LU.A	14.6	188	2,744	2,977	-8.5%
LU.B	59.2	191	11,316	12,359	-9.2%
MG.A	1.04	210	218	200	8.7%
MG.B	3.4	209	706	727	-3%
S3D	776	227	176,421	155,840	11.7%

The energy model predicts a wide range of execution times and energy consumption rates with impressive accuracy. The average error (using absolute values) is 7.4%. This is very accurate for such a simple model. The highest error occurred for IS.B. This benchmark is difficult to model for two reasons. First, it is the only one of the NPBs that does primarily integer-operations, and second, the majority of the access patterns are random accesses done using indexed-arrays. The random accesses complicate the modeling process, because the current machine signatures do not estimate the performance achieved during random accesses. This implies that the rate for stride-8 accesses is used to estimate random accesses. Expanding the benchmarks and model to cover these cases is part of future work.

5. Related Work

Early work in energy consumption models is primarily focused on embedded-systems and used to improve the hardware design and software optimizations that will allow for longer battery life. Our work focuses on large HPC centers, and specifically their variable workloads. The other main difference is that we are performing cross-architectural predictions, while previous work has not included a cross-architectural component.

Instruction-level power analysis was introduced by Tiwari et al.[16,17] and later expanded to include component interactions using linear regression by Givargis[18]. Whole-system power estimates, including on-board interconnect memory and power conversion systems, were presented in Reference 19. Instruction-level power analysis estimates the overall energy consumption of an application by measuring the energy consumption of required instructions. Complex cache structures, which can cause large variations in energy consumption, complicate the use of this method.

Projects that model the energy consumption on a processor by component; processing core, caches, and main memory[20–22], are specifically designed for *systems-on-a-chip* (SOC). In these systems-specific software will be run, and it is known *a priori* allowing for a higher-level of optimization of both performance and energy consumption. In these models, the memory energy consumption rate is estimated using a cost-per-access model, which does not model the different costs of different levels of memory.

A more recent approach, presented in Reference 23, combines multiple power estimators into one simulation engine, thus enabling detailed simulation of some components, while using high-level models for others. This approach is able to account for interaction between memory, cache and processor at run-time, but at the cost of potentially long run-times. Longer run-times are caused by different abstraction levels of various simulators and by the overhead in communication between different components. The techniques that enable significant simulation speed-up are presented, but at the cost of the loss-of-detail in software design and in the input data-trace.

Cycle accurate register transfer level energy estimation is presented in Reference 24. Any kind of cycle-accurate simulation is not practical at large-scale data centers and HPC centers. The long running applications, which run across hundreds of processors, would incur a tremendous slowdown.

An alternative approach for energy estimation using measurements as a basis for estimation is presented in the PowerScope tool[25]. PowerScope requires two computers to collect the measurement statistics, some changes to the operating system source code and a digital multimeter.

Although this system enables accurate code-profiling of an existing system, it would not be appropriate for use in cross-architectural estimates.

JouleTrack[26] is a tool for software energy profiling that is able to achieve an impressive 97% accuracy in power usage estimates. It is; however, designed for embedded-systems and requires cycle accurate simulation. Cycle accurate simulation is not reasonable at the level-of-data of HPC Centers.

SoftWatt[27] targets complete-system power profiles of high-end systems, that can be used for design and evaluation of power optimization techniques. This tool is similarly aimed at high-end systems, but depends on cycle-accurate simulation.

A system-wide energy consumption model is created by Reference 7. The model depends on the correlation of system bus traffic with task activities, memory-access metrics and board-level power measurements. The result is a run-time energy consumption estimate for an application. This method shows promise for use in a scheduled environment, given an intelligent scheduler. Our method offers a different insight to energy consumption by providing cross-architectural predictions, bypassing the need to execute the application on the target architecture.

6. Conclusion

Modeling energy consumption provides valuable information that can inform system acquisition and design decisions. As energy costs grow to comprise a larger portion of the cost of operating large computer resources, more attention is being paid to the energy efficiency of next-generation resources. Energy efficiency cannot be evaluated independent of workload, and the models presented here are a tool for better understanding the interactions that affect energy consumption.

Future resources, while operating at lower power levels per CPU, will suffer the same consequences of data movement that today's resources do. This is because the trend of large cost-increases associated with accessing larger areas of memory is likely to continue in future systems. As systems begin to make energy counters and system management mechanisms, such as dynamic frequency-throttling, available to applications it becomes very important to understand how energy efficiency can be achieved by the application. The models presented here provide an avenue for providing the necessary information.

Our models provide a means to greatly reduce the amount of work required to evaluate resource acquisition tradeoffs. The amount of work is reduced significantly, and more importantly, if the machine doesn't exist yet but the energy consumption can be forecast on simple benchmarks, our methods allow reasonable estimates of the actual energy consumption of real applications.

Acknowledgments

This work was supported in part by the DoD High Performance Computing Modernization Program and by Defense Advanced Research Projects Agency (DARPA) award and by the DARPA Multi-Scale Systems Center (MuSyC). The authors would like to thank Eva Hocks and Jeffrey Bennett for their support during the use of Dash.

References

1. Kramer, W.T.C., "Best practice in hpc procurements." *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. New York, NY, USA: ACM, p. 230, 2006.

2. Post, D.E., "Guest editor's introduction: Computational science and engineering for the US Department of Defense." *Computing in Science and Engineering*, vol. 9, pp. 10–11, 2007.

3. Barroso, L.A. and U. Hölzle, "The case for energy-proportional computing." *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

4. Tikir, M., L. Carrington, E. Strohmaier, and A. Snavely, "A genetic algorithms approach to modeling the performance of memory-bound computations." *Proceedings of SC07*, Reno, NV, November 2007.

5. Carrington, L., M. Laurenzano, A. Snavely, R. Campbell, and L. Davis, "How well can simple metrics represent the performance of hpc applications?" *Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing*, 2005.

6. P.K., et al., "Exascale computing study: Technology challenges in exascale computing study: Technology challenges in achieving exascale systems." *DARPA Tech. Rep*., September 2008.

7. Lewis, A., S. Ghosh, and N.F. Tzeng, "Run-time energy consumption estimation based on workload in server systems." *HotPower '08: Workshop on Power Aware Computing and Systems*, December 2008.

8. Available, https://www.wattsupmeters.com/secure/index.php.

9. Available, http://sine.ni.com/nips/cds/view/p/lang/en/nid/203822.

10. Available, http://www.phidgets.com/index.php.

11. Snavely, A., L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for application performance modeling and prediction." *ACM/IEEE Conference on High Performance Networking and Computing*, 2002.

12. Tikir, M., M. Laurenzano, L. Carrington, and A. Snavely, "The pmac binary instrumentation library for powerpc." *Workshop on Binary Instrumentation and Applications*, 2006.

13. Laurenzano, M.A., M.M. Tikir, L. Carrington, and A. Snavely, "Pebil: Efficient static binary instrumentation for

linux." *International Symposium on the Performance Analysis of Systems and Software*, Mar. 2010.

14. Dhiman, G. and T.V. Rosing, "System-level power management using online learning." *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 5, pp. 676–689, 2009.

15. Available, http://www.nas.nasa.gov/Resources/Software/npb.html, 2008.

16. Tiwari, V., S. Malik, A. Wolfe, and M.-C. Lee, "Instruction level power analysis and optimization of software." *VLSI Design, Proceedings of the Ninth International Conference*, pp. 326–328, Jan. 1996.

17. Tiwari, V., S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, vol. 2, no. 4, pp. 437–445, Dec. 1994.

18. Givargis, T., F. Vahid, and J. Henkel, "Fast cache and bus power estimation for parameterized system-on-a-chip design." *Design, Automation and Test in Europe Conference and Exhibition Proceedings*, pp. 333–338, 2000.

19. Simunic, T., L. Benini, and G.D. Micheli, "Energy-efficient design of battery-powered embedded systems." *Special Issue of IEEE Transactions on VLSI*, May 2001.

20. Li, Y. and J. Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems." *Design Automation Conference Proceedings*, pp. 188–193, June 1998.

21. Kapoor, B., "Low power memory architectures for video applications." *VLSI: Proceedings of the 8th Great Lakes Symposium*, pp. 2–7, Feb. 1998.

22. Landman, P. and J. Rabaey, "Activity-sensitive architectural power analysis." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions, vol. 15, no. 6, pp. 571–587, June 1996.

23. Lajolo, M., A. Raghunathan, S. Dey, and L. Lavagno, "Efficient power co-estimation techniques for system-on-chip design." *Design, Automation and Test in Europe Conference and Exhibition Proceedings*, pp. 27–34, 2000.

24. Vijaykrishnan, N., M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using simple power." *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 95–106, 2000.

25. Flinn, J. and M. Satyanarayanan, "Powerscope: a tool for profiling the energy usage of mobile applications." *Mobile Computing Systems and Applications, Proceedings WMCSA '99*, pp. 2–10, Feb. 1999.

26. Sinha, A. and A.P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling." *DAC '01: Proceedings of the 38th Annual Design Automation Conference*, New York, NY, pp. 220–225, 2001.

27. Gurumurthi, S., A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using complete machine simulation for software power estimation: the softwatt approach." *High-Performance Computer Architecture, Proceedings, Eighth International Symposium*, pp. 141–150, Feb. 2002.