International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 5, No.3 (May-2016)

http://dx.doi.org/10.12785/ijcds/050301

Thread-Level CPU Power Measurement for High Performance Parallel Systems: Impact Analysis of System Control Parameters on HPC Energy Efficiency

David K. Newsom[1], Sardar F. Azari[2], Olivier Serres[1], Abdel-Hameed Badawy[1,3] and Tarek El-Ghazawi[1]

[1]*Department of Electrical and Computer Engineering, the George Washington University, Washington, DC, USA*
[2]*Institute of Electrical and Electronics Engineers (IEEE), NJ, USA*
[3]*Electrical Engineering Department, Arkansas Tech University, Russellville, AR, USA*

Received 2 November 2015, Revised 7 March 2016, Accepted 2 April 2016, Published 1 May 2016

Abstract: Research in high performance computing (HPC) energy optimization is a growing field motivated by cost and environmental drivers. One of the key challenges in reducing program-dependent HPC power consumption is precise measurement of the thread-level CPU (as distinct from other system components) energy use during the initialization and execution phases of a parallel program's execution. In support of our research in Partitioned Global Address Space (PGAS) power optimization we developed a scalable CPU direct measurement framework with its associated reporting and data acquisition components. We utilized this framework to evaluate the execution time and energy consumption impact of our code optimizations. The measurement framework itself and the associated instrumentation is sufficiently scalable to support any program-level energy optimization research in high performance parallel systems.

Keywords: Energy management, High Performance Computing, CPU Power Measurement, MATLAB, Simulink

1. Introduction and Overview

Research in energy efficient high-performance computing (HPC) continues to be driven by both environmental and cost factors. The investigations can be broadly divided into three main focus areas: compiler and runtime optimizations; operating system automations; and power efficient microprocessor design. In order to validate HPC energy efficient compiler/runtime optimizations, a key challenge lies in precisely measuring thread-specific energy consumption and the corresponding impact of code-level optimizations.

To support our own research [1], [2], [3] in CPU power optimization using the Partitioned Global Address Space (PGAS) programming paradigm, we developed a precise and scalable thread-level CPU energy measurement framework. We only measured the energy consumption of the CPU because it has a substantial portion of overall system energy and because the CPU has the greatest energy consumption variance of any system component as a function of program related activity [4].

A prerequisite to experimental validation of our code-level energy optimizations was the construction of an instrumented HPC test-bed cluster system. In this paper, we describe 1) an accurate and scalable CPU power measurement system to assess the energy savings impact of parallel program code modifications, and 2) the key design aspects to consider when building high performance test-bed systems to support power optimization research. The key energy measurement problem we solved was how to tightly synchronize the program's execution phases with the data acquisition system across the network.

This synchronization involved two main aspects. First, the program needed to be controlled by the data acquisition system from a start/stop standpoint. Second, the program also needed to inform the data acquisition system as it transitioned between discrete phases so that the time and energy results that were specific to the program phases could be measured. This paper describes the research and development of the CPU power measurement system that produced the experimental results shown in Section 7.

The paper is organized as follows: Section 2 covers topics that provide a relevant context for the measurement system, energy calculations, and experimental results;

E-mail address: dkn@gwu.edu, Dr.Sardar.Azari@ieee.org, serres@gwu.edu, badawy@gwu.edu, tarek@gwu.edu

Section 3 outlines related work in HPC power measurement and estimation; Section 4 presents the methodology and mathematical basis for the energy calculations of our data acquisition system; Section 5 describes the instrumented hardware platform used for our experiments and the associated instrumentation and data acquisition system; Section 6 explains the software configuration and setup including the OS/compiler setup on the cluster, the MATLAB/Simulink implementation, the Google Cloud job control and user interface design, and the XML data structure; Section 7 presents some sample results the system has produced; and Section 8 discusses our conclusions and future work related to the measurement and instrumentation aspects of our research.

2. Background

This section presents some relevant background information related to the development of the measurement system, the energy calculation methodology, and the evaluation of the experimental results.

A. Metrics

It is useful to describe the metrics used to evaluate the performance and energy efficiency across processor families or between different algorithms running on the same processor under different conditions. Early computer architectures were driven primarily by the quest for improved performance, regardless of the energy consumption. According to Hennesy and Patterson [5] this was evidenced by the legacy metrics used to compare computer performance, such as FLoating-point OPerations per Second (FLOPS) or Million Instructions Per Second (MIPS). All the metrics were in terms of instructions per unit time. As component densities increased with a corresponding rise in energy consumption and heat dissipation, the performance metrics associated with HPC systems are now described in terms of the energy (not time) necessary to complete a given computational task.

The most basic metrics used to measure computational performance and/or energy efficiency are *time*, measured in `seconds` and *energy*, measured in `Joules`. Total execution time is generally considered the most basic metric to compare computational performance between two different computer systems executing identical programs and data sets. At an aggregate level, there is usually a strong correlation between the execution time and the energy consumed over the execution time interval. However, a large number of conditions can affect how much energy is consumed for a given computational problem without significantly impacting execution time. An improvement in computational energy efficiency is defined as a reduction in the energy consumed over the same execution time interval. In the case where an optimization reduces the execution time and a corresponding proportional reduction in the energy consumption is also observed, energy efficiency is not improved even though the overall energy consumption is reduced because the energy reduction can be attributed simply to the reduction in execution time.

The *energy-delay product* (`Joule*seconds`) [6] is defined as the product of the measured energy and time over the time interval. It is good measure of overall improvement in absolute terms and is straightforward to calculate from energy and time measurements. Its main advantage over the energy consumption rate is that it presents, in a single value, both time and energy consumption. This means that it can be used as a dependant variable in a two dimensional graph to show the correlation between both time and energy and some other factor (e.g., temperature). However, as a single value, it is difficult to conclude the degree to which energy or time (or both) were improved. The charts in Section 7 use the energy-delay product as one of the comparison metrics.

B. CMOS Power Dissipation

There are two basic ways that CMOS circuits dissipate (consume) electrical power, *dynamic* power and *static* power. The total power consumption of any CMOS circuit is the sum of the dynamic and static power consumption. The basic works on the calculation of CMOS dynamic (switching) and static (leakage) power are found in [7], [8], [9], [10], [11], [12]. We cover the specifics of our approach to calculating the total CPU and memory power consumption of our instrumented HPC test-bed cluster in Section 4.

Dynamic power is the activity-dependent power consumption and varies according to workload as well as the switching frequency of the circuits. Intel® 's paper [13] on Core™ processor power management explains the techniques used by chip designers to mitigate dynamic power consumption, which include techniques such as *Dynamic Voltage and Frequency Scaling (DVFS)* described in Section 2-C. It is a generally accepted fact that commodity microprocessors consume more dynamic power at higher operating (switching) frequencies.

Static or *leakage* power is consumed by the circuit when it is powered on and tends to be constant regardless of the switching frequency. In other words, static power is *activity-independent* power consumption and is a constant for a given circuit irrespective of the activity level of the processor. Kim et al., [12] provide an excellent overview of the prior works as well as explaining how the effect of Moore's Law is increasing the percentage of static power consumption of the total power consumption in CMOS components due to the increase in chip density. The static power consumption of a CPU represents the lower bound for dynamic power consumption, (i.e.,even when all workload related activity has ceased and the switching frequency is set to the lowest value) the static power consumption continues so long as all the hardware

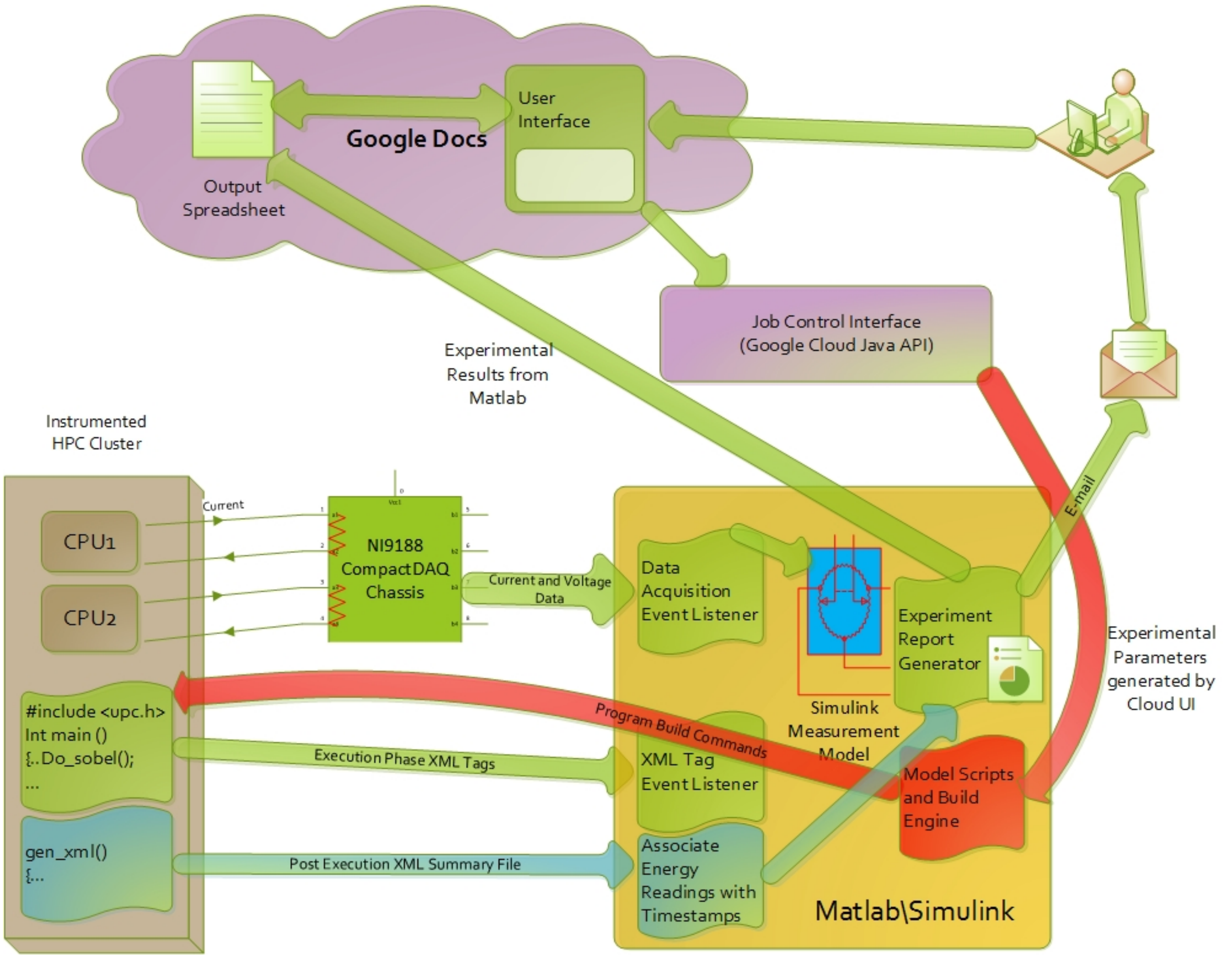

Google Docs
Output Spreadsheet
User Interface
Job Control Interface (Google Cloud Java API)
Experimental Results from Matlab
Instrumented HPC Cluster
CPU1
CPU2
Current
NI9188 CompactDAQ Chassis
Current and Voltage Data
Data Acquisition Event Listener
Simulink Measurement Model
Experiment Report Generator
E-mail
Experimental Parameters generated by Cloud UI
#include <upc.h> Int main () {..Do_sobel(); ...
Program Build Commands
Execution Phase XML Tags
XML Tag Event Listener
Model Scripts and Build Engine
gen_xml() {...
Post Execution XML Summary File
Associate Energy Readings with Timestamps
Matlab\Simulink

Figure 1. Measurement system high-level view

components are powered-up.

C. Dynamic Voltage and Frequency Scaling

DVFS can be used at the program, operating system, or hardware level to reduce CPU dynamic power consumption by lowering the switching frequency (often by reduction of the supply voltage) of the core and/or the entire CPU. DVFS has many applications, but the most common one is to reduce predictable idle periods in a program's execution thereby reducing energy consumption without increasing execution time.

The application of DVFS at the program, operating system, or CPU is one of the most heavily researched areas in computational energy efficiency. All current commodity microprocessors are equipped with DVFS capability. Intel® 's DVFS implementation is called Enhanced Intel® Speedstep™ Technology (EIST) [14] and AMD's DVFS implementation is called PowerNow!™ [15].

D. TurboBoost

TurboBoost [16] is a feature of later generation Intel® microprocessors that allow a subset of microprocessor cores to burst above their maximum rated operating frequency. The TurboBoost feature can be enabled in the system BIOS if DVFS is also enabled. On Intel® 's *Sandy Bridge* [17], *Ivy Bridge* [18], and *Haswell* [19] Core™ processors, TurboBoost steps up a core's frequency in 100MHz increments under the control of the *UnCore* processor. The number of frequency steps, their stride, and the max frequency are a function of how many cores are active (i.e., are not in the C1 or greater idle state) and is dependent on the total number of cores and the maximum thermal envelope of the microprocessor. TurboBoost is an architectural concession to the fact that a large portion of software remains sequential (rather than parallel) in nature and so this feature can boost the performance of application codes that are "lightly-threaded" (i.e., under-utilize the full core capacity of a CPU). While this feature improves performance in cases where computational parallelism does not utilize the entire available cores of a given processor, the energy consumption per unit of computational execution does not scale linearly because as the processor's overall temperature increases it consumes more energy for a given workload [20].

E. Dependency Domains and Packages

Dependency domains (also referred to as a *voltage island* or *package* when the number of cores equals the total number of cores on a physical die) are groups of physical or logical processors, that must be controlled together for a particular feature to take effect. Some microprocessors require that all cores on a CPU must have their frequencies scaled together (i.e., all cores must be set to the same frequency and governor for any change to occur). This would make the dependency domain for P-States (DVFS) at the CPU level. For example, up until the *Haswell* [19] series of microprocessors, the Intel® Core™ (including Xeon™) family of microprocessors could only frequency scale all package cores together, while the Intel® Phi™ [16], [21] co-processor can frequency scale sub-groups of cores. *Haswell* processors can voltage and frequency scale individual cores [22] as each core is fed by its own power supply. It should be noted that while AMD's Opteron™ [23] microprocessor (and later processors) could frequency scale at the core level, the cores had to do this inside a fixed supply voltage envelope as all the cores were fed by a single voltage source which limited the usefulness of the feature.

F. Processor Asymmetry

We experimented with simulating an *Asymmetric Processor* (AMP) system on our AMD cluster [2] by forcing one of the two physical CPUs to the lowest frequency step and the other CPU to the highest frequency step. At program load time, some number of worker threads were launched on the "slow" processor and would block on local operating system mutexes. When remote communications needed to be performed by one of the threads running on the "fast" processor, the thread would use a `MUTEX` (mutually exclusive synchronization construct) to release a worker-thread to handle the remote communications at the slower frequency. This setup was designed to avoid the DVFS transition time penalty because it avoided the CPU-local frequency down/up-shift during the remote communications phase. It also avoided the much larger overhead of rescheduling a thread between physical processors (using a thread-specific CPU affinity mask and calling `sleep(0)` system call) which would invalidate the `L1`, `L2`, and `L3` caches.

The approach suffered, however, from the longer memory access times when a thread is accessing a region of memory which does not have hardware affinity with the processor on which it is executing. When the thread first allocates the thread-local memory into which it will write the remote data, the memory is pinned to the memory bank that has affinity with the processor on which the thread is running (the "fast" CPU in this case). When the worker-thread running on the "slow" CPU tries to access the memory, the access must transit the inter-CPU connection bus and the remote memory controller. We disabled the *Non-Uniform Memory Access* (NUMA) optimization (in the system BIOS) which had the effect of creating a low-order memory bank interleave between the two processors' respective memory banks. This configuration created even more delay for a particular thread because the memory access time for the thread became the lowest common denominator access time between the two processors because when the NUMA optimization is turned off, the memory interleave creates a consistent memory access time for any thread on the system, irrespective of the relative proximity of CPU and memory bank. (With NUMA optimization enabled, the lowest memory access time would be to the memory bank that is associated with a particular CPU.)

3. Related Work

Two principal approaches have been followed with regard to high performance computing CPU power measurement: direct measurement and estimation.

A. Direct Measurement Approaches

Cui et al., [4] describe a complete system for measuring the energy consumption of all the system components in a commodity computer system. While they touch briefly on the issue of synchronizing the energy measurements with program phases, they do not develop the idea into a practical implementation for systematic exploration of the permutation space.

The most recent developments in direct measurement of CPU power is described by Laros et al., [24] the authors describe an non-intrusive embedded measurement system, *PowerInsight*, that intercepts CPU power via the power supply harness (similar in principle to the approach we describe in 5-B). The measurement and data acquisition systems are combined on a small footprint embedded system that can be installed in each node of a cluster and then polled by a master system for overall data assembly and analysis. The primary focus of *PowerInsight* was the hardware aspects of the power measurement and data acquisition; they did not address the problem of program-level measurement synchronization.

In [25] the authors had access to a built-in, albeit still non-intrusive measurement capability built into a Cray™ blade chassis – a so called Computer Remote Management and Serviceability (CRMS) system that allowed them to pull accumulating power statistics from each of the blades at a sample rate of not more than 1/sec. They

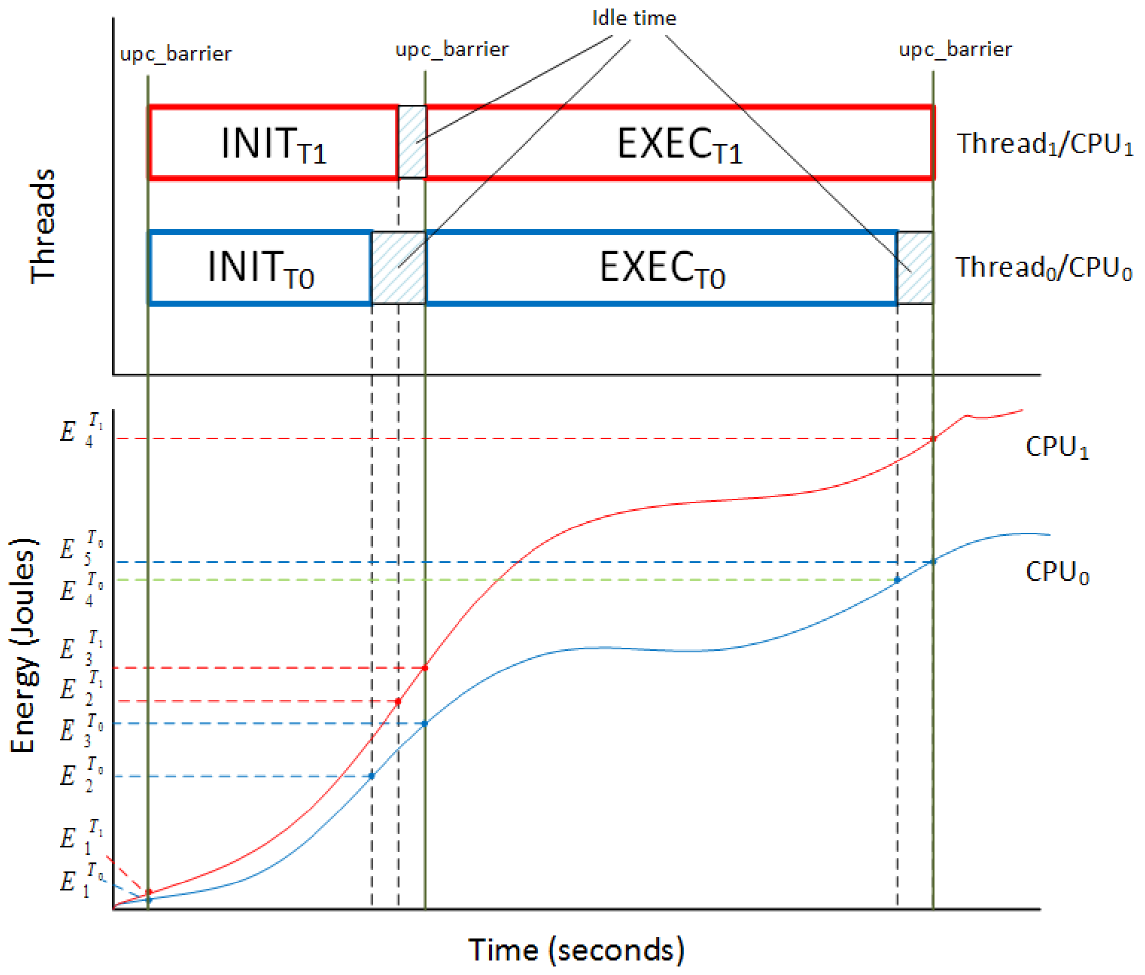


Figure 2. Energy measurement by program phase at thread level

also make a claim regarding the ability to tune the OS to take advantage of sleep cycles on the processor as a power savings technique, yet all modern commodity processors force all their cores into a C-state during idle periods and this feature is not usually under the control of the OS.

The overhead associated with direct measurement approaches is one of the main reasons why the direct measurement approach is a minority in the HPC energy research community; researchers tend to favor indirect methods such as *estimation* [26] using the built-in performance counters of most CPUs as a proxy for direct measurement; or aggregation methods such as just inserting a current meter between the main system power supply and the utility source (wall socket) [27]. While this approach has a certain “brute force” simplicity about it, this measurement approach introduces a lot of measurement noise into the data as many other system components are also getting measured along with the CPU/DRAM.

B. *Estimation Approaches*

Beyond direct measurement methods is the approach of estimating system power consumption based on CPU performance counters. This topic is covered in detail by Bircher and John [26], but they conclude that there can be up to a 9% error per subsystem using this method. Based on our own work, this level of error may obviate the value of certain optimizations that may provide energy savings within this margin.

In addition to performance counter estimation, Pakin and Lang [28] explore the use of time, power, and energy models for supercomputers running large-scale scientific application suites. Their work provides an evaluation of the impact of DVFS techniques in terms of the energy-delay product (see Section 2-A). While their research is useful in describing the potential impact of frequency scaling for certain common workloads, the work does not address optimization techniques beyond frequency scaling (such as the effect of idle-power management of recent processor generations).

4. Energy Calculation Methodology

This section outlines our energy calculation methodology. The basis of our power optimization research was to evaluate a particular parallel algorithm and predict the code regions where the CPUs may be in an idling state such as waiting on remote communications or a synchronization construct (e.g., `lock, barrier`, etc.).

For purposes of experimentation, we divided an algorithm into two distinct operating phases: initialization (`INIT`), in which the optimized program is performing the bulk of its predictable remote communications; and execution (`EXEC`), in which the program is highly CPU/local memory bound. Each phase of operation is delineated by a `barrier` construct that forces all threads to synchronize before entering the next phase. Depending on the algorithm being tested, we employed a number of techniques to improve energy consumption in comparison to the original form of the algorithm.

Our initial measurement design [1] relied on the program under investigation to output *eXtensible Markup Language* XML [29] tags in real-time to which a system-level "listener" on the data acquisition computer would monitor cluster activity. We compensated for the inherent system latencies by using an energy/time ratio correlated with the program's self-reported execution times by phase.

Each CPU being measured represents a single channel on our data acquisition system. The data acquisition hardware (described in Section 5) is constantly measuring current flow to the active processors, and our energy calculations are based on the delta between the start and stop timestamps that the program is generating for each thread as it executes each program phase. Depending on the type of experiment being run, we measure different time intervals on the CPU measurement channels to accurately capture the energy usage of a program during its `INIT` and `EXEC` phases. Our system keeps track of a specific thread's energy consumption, even if it transitions between CPUs (see Section 2-F for a use case), through careful placement of the calls to the instrumentation runtime. For ease of explanation, Figure 2 shows the measurement model in the case where we have two threads (one per CPU) running on each cluster node and DVFS is in use. As presented in Section 7, we tested many different thread combinations.

With the exception of our work to simulate processor asymmetry described in Section 2-F, we used a *CPU affinity mask* [30] to "pin" a thread to a core or set of cores on the same physical processor when the thread is initialized. We do this to prevent a thread from migrating between physical processors when executing system calls (e.g., changing CPU frequency). A system call transfers context to the Linux scheduler to execute the requested (privileged) operation and the calling thread is suspended during this interval. When the thread context is restored, it may have been rescheduled to a core on a different physical processor which also means that the thread moved from one current measurement channel to another. Since we do not necessarily have the timestamp data to correlate the thread's movement, this change of processor would invalidate the measurement continuity for that thread. The equations that relate the thread activity to the energy calculations are shown below.

The power (P) in `Watts` that a CPU draws is expressed by Equation 4 that describes the power draw of a CMOS circuit [12]:

$$P = CfV^2 + P_{Static} \tag{1}$$

where C is the capacitance of a CMOS Integrated Circuit, f is the operating frequency, V is the supply voltage, and P_{static} is the *leakage* power consumption of the circuit. Since C and f in Equation 4 are not directly measurable values, we apply Equation 2 where we measure I and U over a time interval t and then numerically integrate the product using the trapezoidal rule [31]. The integral formula to calculate the total energy E over the measurement interval t_1 to t_2 is shown in Equation 2:

$$E = \int_{t_1}^{t_2} I(t) \cdot U(t)\, dt \tag{2}$$

where E is the electrical energy measured in Watt-hours, $I(t)$ is the instantaneous current measured in Amperes and $U(t)$ is the instantaneous voltage measured in Volts. The curves on the bottom of Figure 2 are derived by the process shown (for a single node) in Figure 3 and are the energy readings calculated using the formula in Equation 2. The clocks on the cluster nodes are synchronized with the clock on the MATLAB system at the start of each program run and then the timestamps are correlated with the energy readings after the program terminates using the approach described in 6-B. Equation 3 shows how the `INIT` phase energy readings are calculated for a single node when both CPUs are in use. By way of illustration, $E_3^{T_0}$ is the timestamp marker for Thread `0` at the beginning of its `EXEC` phase. If only one thread been running on the node, the second term in Equation 3 would not be present. Equation 4 shows a similar calculation for the `EXEC` phase.

$$E_{INIT} = \left(E_3^{T_0} - E_1^{T_0}\right) + \left(E_3^{T_1} - E_1^{T_1}\right) \tag{3}$$

$$E_{EXEC} = \left(E_4^{T_0} - E_3^{T_0}\right) + \left(E_5^{T_1} - E_3^{T_1}\right) \tag{4}$$

Equation 5 sums the energy for the two program

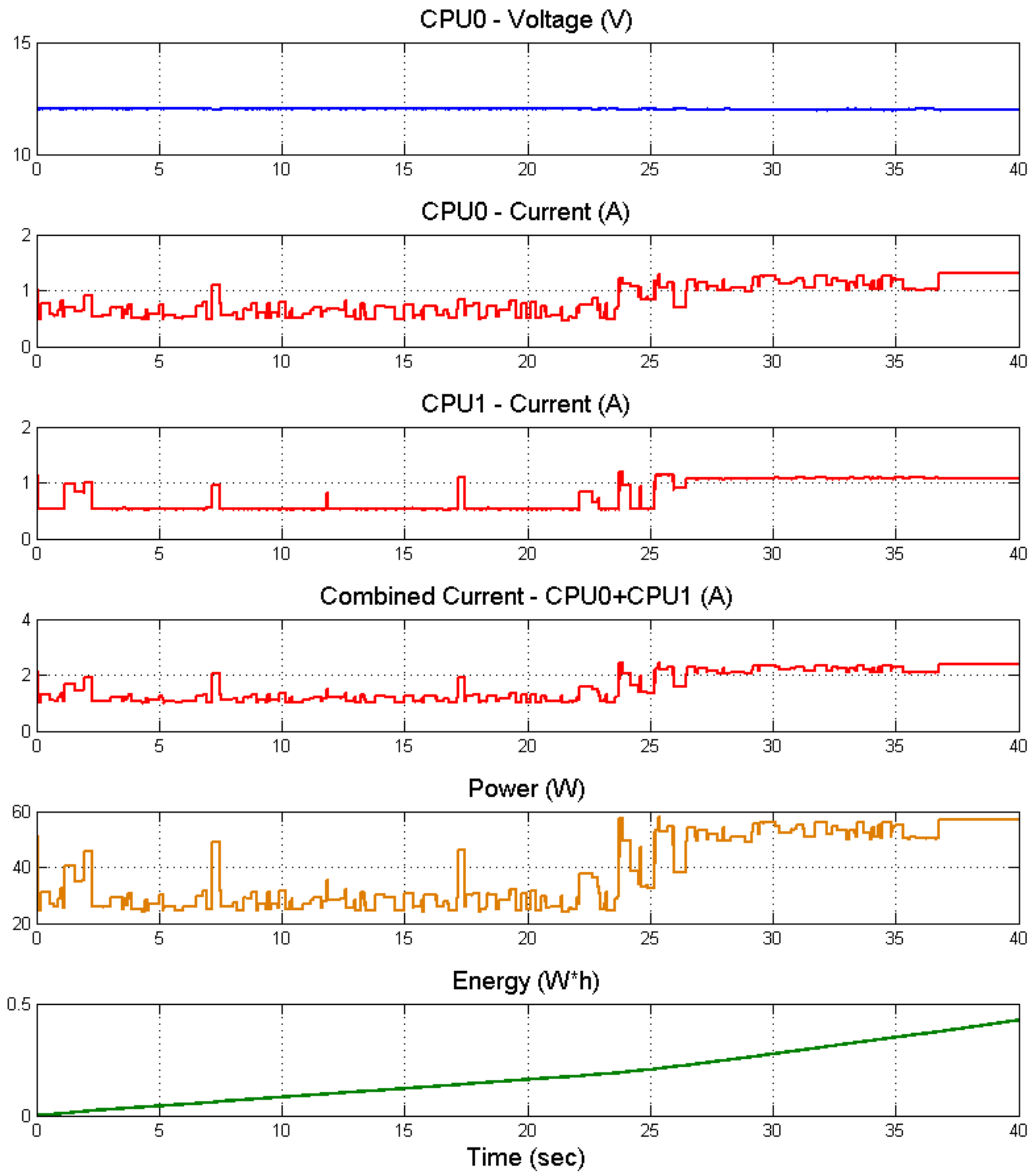


Figure 3. Energy measurement by data acquisition channel

phases to calculate the node specific energy and Equation 6 shows how the total energy across the cluster is calculated.

$$E_{Node} = E_{INIT} + E_{EXEC} \qquad (5)$$

$$E_{Cluster} = \sum_{i=1}^{n} E_{Node_i} \qquad (6)$$

The energy of threads waiting on synchronization barriers (BW = Barrier Waste) is calculated from the idle periods directly preceding a threads arrival timestamp on a barrier construct. Similar to Equation 6, the total *Barrier Waste* energy can be summed for the cluster.

$$E_{BW} = \left(E_5^{T_0} - E_4^{T_0}\right) + \left(E_3^{T_1} - E_2^{T_1}\right) \qquad (7)$$

5. Hardware Configuration

This section describes the hardware configuration of the test bed, its associated power measurement instrumentation, and the current and voltage data acquisition system at the hardware level.

TABLE I. Evolution of test-bed clusters

Label	CPU Type	Frequency	cpu#	core#	node#	RAM	HT?	TB?	CS?
A	Intel® E6550	2.00-2.33GHz	1	2	6	4GB	N	N	N
B	Opteron™ 2354	1.1-2.2GHz	2	4	16	16GB	N	N	N
C	Xeon™ E5-2640v2	1.1-2.0GHz	2	8	4	32GB	Y	Y	Y

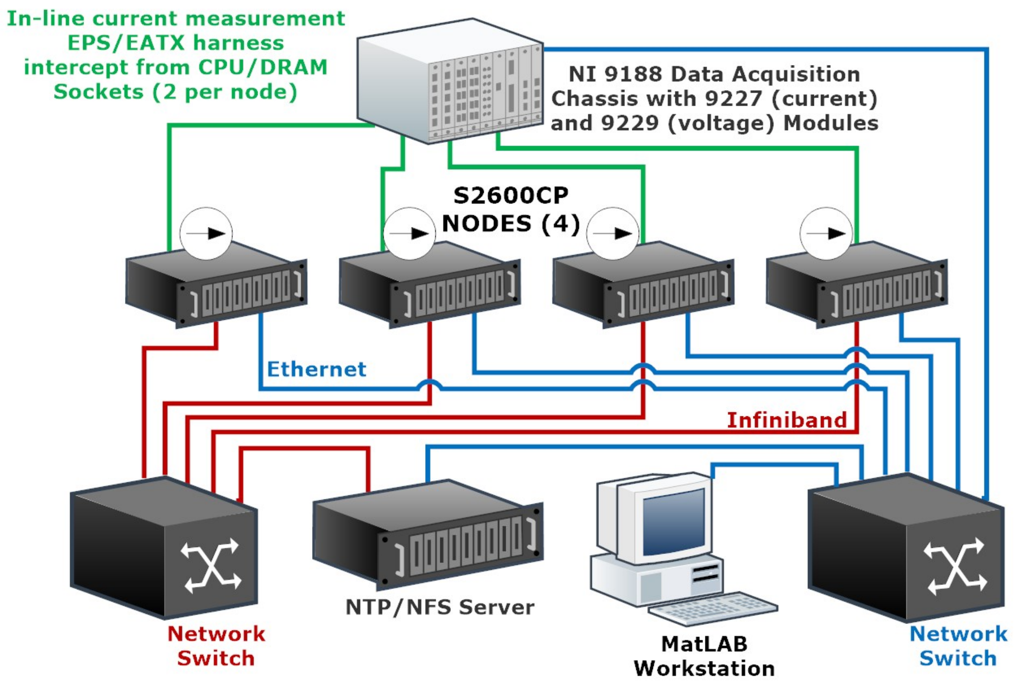


Figure 4. Instrumentation diagram four node Intel® Ivy Bridge cluster

A. Instrumented HPC Test Cluster

Over the period covered by this research we built three distinct generations of instrumented test-bed clusters as shown in Table I. The first generation (A) system [1] used Hall Effect meters on a single Intel® desktop CPU six node cluster and hinted at the possibilities inherent in a scalable in-line measurement of CPU current. The second generation (B) system [2] and was based on a dual processor AMD Opteron™ [23] and had 16 nodes interconnected by both Ethernet and Infiniband networks. Our third generation (C) system that was used for the experiments described in Section 7 is documented below.

The test-bed cluster used for the experiments consists of 4 dual-socket Intel® S2600CP™ systems [32], instrumented as shown in Figure 4 and Reference C in Table I. Each socket contains an 8-core Intel® 's *Ivy Bridge* Xeon™ E5-2640 v2 @ 2.00GHz [13], with 16GB of RAM per CPU, for a total of 32GB memory per node. Each processor core is capable of running two execution threads when the hyper-threading [33] feature is enabled in the BIOS. Each node of the cluster holds two CPUs, so there are 8 CPUs x 8 cores/CPU for a total physical core capacity of 64 cores (the system will report 128 cores with hyper-threading enabled). The controllable frequency domain of the cluster was at the socket level, thus there were 8 independent frequency domains (one per socket) which had to be separately controlled when the cluster's frequency needed to be changed. Figure 6 shows a single node of the cluster and Figure 5 shows the entire cluster in operation including the data acquisition hardware described in Section 5-B.

In addition to the measurement instrumentation, the cluster nodes were connected by two different interconnection networks, 40Gb/s QDR Infiniband and Gigabit Ethernet so that we could explore the energy consumption impact of different network latency on various types of HPC codes. These networks could be used singly or in tandem and their usage was controllable at runtime by selection of the UPC (see Section 6-A) *GASNet* [34] conduit.

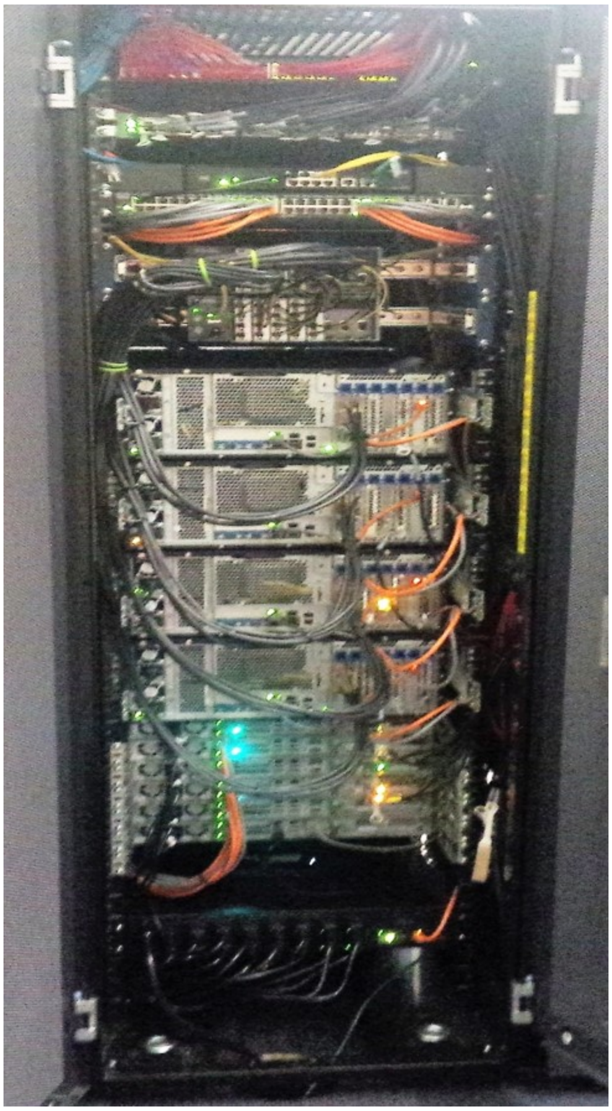


Figure 5. Four node Ivy Bridge cluster with instrumentation

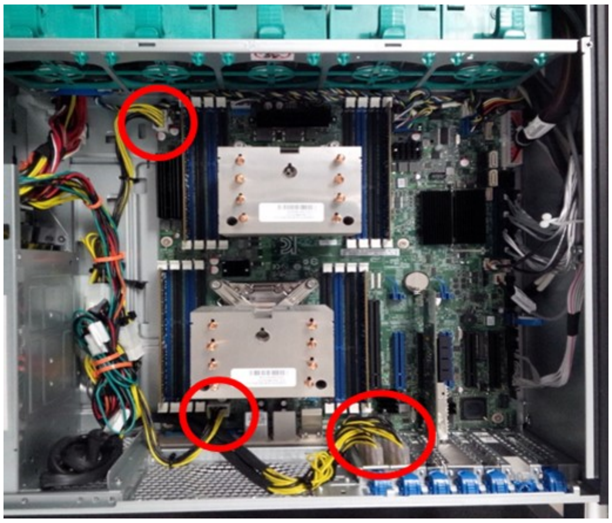


Figure 6. Instrumented dual Ivy Bridge Xeon™ chassis

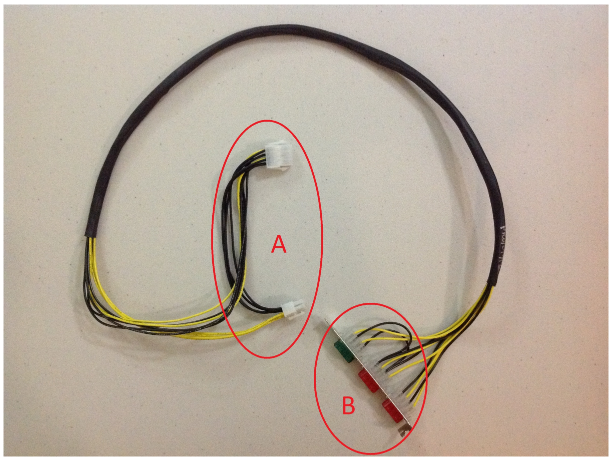


Figure 7. Custom wiring harness to route EPS/EATX connector

B. Hardware Instrumentation

This section describes the way we performed our energy measurements. The system measures the instantaneous static and dynamic *power* (see Section 2-B) consumption and then integrates the power over the execution time interval to calculate the actual energy consumption of the full program (or job set) execution. The energy is calculated as described in Section 4.

The principle design goal of the measurement system was to measure CPU/DRAM energy consumption without any impact to the CPU performance (which might also affect the overall energy consumption). We measured the current usage over time (in combination with a constant voltage) running over the wires between the switching power supply and the CPU sockets. The measurement system uses two National Instruments (NI) Compaq-DAQ™ model 9188 ethernet data acquisition chassis, each with 4x4-channel model 9227 current [35] module and 1x4 channel model 9229 voltage acquisition module [36]. This setup provided a total measurement capacity of 32 current data acquisition channels, 16 of which capture CPU current and 16 of which capture DRAM current. There are a total of 8 processors so each processor consumed two channels for the CPU and two channels for the memory. Figure 8 shows the decomposition of the wiring harness (see Figure 7) that allows the non-intrusive interception of the current and voltage feeding the CPU/DRAM module. (A) shows the connectors that connect to the motherboard/backplane power connector, and (B) shows the external ports that connect to the NI data acquisition current (2 red connectors) and voltage (1 green connector) modules.

We had 4 nodes to instrument and we designed and built a custom wiring harness to intercept the EPS/EATX power lines between the power supply and the moth-

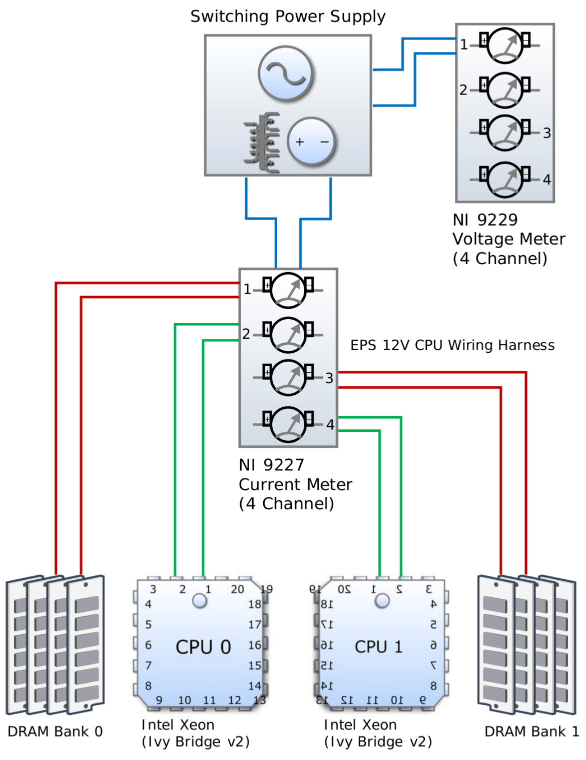


Figure 8. Node level measurement diagram

erboard CPU socket power interface as shown in Figures 7 and 6. The negative side of the circuit (black) connected the power supply directly to the motherboard CPU power interface, and the positive side (yellow) was routed through the back-plane connector and then through the in-line current meters of the data acquisition system. One harness was used for each CPU/memory unit and thus two harnesses per node were installed to capture the current draw from each unit. Figure 4 shows the schematic for the in-line current and reference voltage measurement.

To measure power consumption, we needed to measure the current usage over time (in combination with a constant voltage) running over the wires between the switching power supply and the CPU sockets. Most current generation of commodity microprocessor motherboards use a dedicated EATX [37] power connector running at 12VDC to feed power to the CPU sockets. As the standard wire gauge used in computer power supply harnesses is 18 American Wire Gauge (AWG) [38], it is necessary to use multiple single wires (as opposed to increasing the wire gauge) to safely carry the current load that a CPU can draw. On a dual CPU system like the *Ivy Bridge*, there are eight pairs of 18 AWG wires feeding the CPU sockets and associated DRAM banks. Each CPU and each DRAM bank are fed by two pairs of wires. Thus each CPU/Memory unit provides four separate pairs of wires, and each pair of wires was routed through separate in-line current measurement modules.

While the NI 9227 in-line current measurement module is limited to 5A/channel and the maximum current draw of an *Ivy Bridge* CPU is over 7A, this current load was divided (by the EATX harness itself) into two separate pairs of wires. During the experimental data post-processing step, the current draw of both channels is summed together to calculate the total current draw for each processor. For the Intel® *Ivy Bridge* Xeon™ cluster (see Figure 5, we had the capability to report on the CPU power consumption, the memory power consumption, or the combined sum of the two for each CPU/memory unit.

We used identical computer systems with identical power supplies, so we only measured the voltage of four sample nodes to detect the delta between the actual and ideal supply voltage. An "ideal" switching power supply would not show any fluctuation in supply voltage during changes in current demand from the CPU. However, our "real world" power supplies did show an approximate 0.2V drop during load, which correlated with the increased current draw. The final energy measurement result takes into account the systematic nature of this error, although it became quickly apparent during our initial experiments that any voltage instability in the 12V rail feeding the CPU/DRAM sockets was easy to account for and well within the margin of measurement error, even at the comparatively high sample rate of the NI current meters.

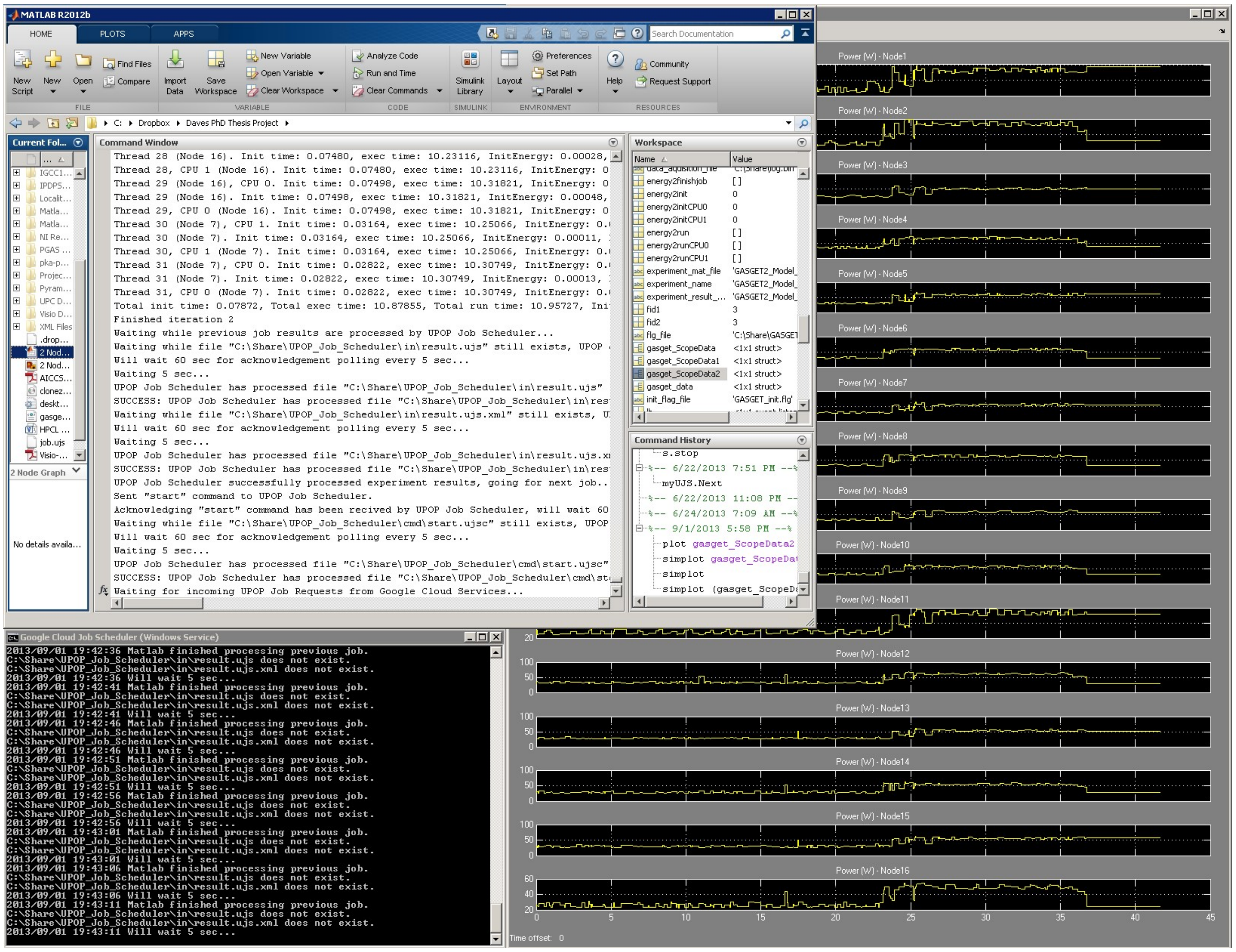
6. Software Configuration

A. Operating System and Compiler

The cluster test bed was installed with 64-bit CentOS [39] version 6.2. An extra node was configured as the mount point for an *Network File System* (NFS) share of the home directory. *Secure Shell* (SSH) keys were pre-shared among all the nodes to allow inter-node program execution without requiring interactive logon authentication. The compiler used for building the test applications was Berkeley *Unified Parallel C* (UPC) [40] and its network transport runtime, GASNet [34] configured with Infiniband (ibv) and Gigabit Ethernet (udp) conduits for the two interconnection networks.

B. System Orchestration

This section describes the orchestration of the measurements in conjunction with program execution. The system is flexible enough to run single applications that consume the entire cluster's resources, or to schedule sets of jobs that run concurrently across the cluster in many different placement configurations. In either case, the experiments can be queued to run for any desired number of iterations, from which can be extracted data with a high statistical confidence. This aspect is particularly important in the context of the minor variations (noise) in the measurement data for identical experiments. Our initial



```
Thread 28 (Node 16). Init time: 0.07480, exec time: 10.23116, InitEnergy: 0.00028,
Thread 28, CPU 1 (Node 16). Init time: 0.07480, exec time: 10.23116, InitEnergy: 0
Thread 29 (Node 16), CPU 0. Init time: 0.07498, exec time: 10.31821, InitEnergy: 0
Thread 29 (Node 16). Init time: 0.07498, exec time: 10.31821, InitEnergy: 0.00048,
Thread 29, CPU 0 (Node 16). Init time: 0.07498, exec time: 10.31821, InitEnergy: 0
Thread 30 (Node 7), CPU 1. Init time: 0.03164, exec time: 10.25066, InitEnergy: 0.
Thread 30 (Node 7). Init time: 0.03164, exec time: 10.25066, InitEnergy: 0.00011,
Thread 30, CPU 1 (Node 7). Init time: 0.03164, exec time: 10.25066, InitEnergy: 0.
Thread 31 (Node 7), CPU 0. Init time: 0.02822, exec time: 10.30749, InitEnergy: 0.
Thread 31 (Node 7). Init time: 0.02822, exec time: 10.30749, InitEnergy: 0.00013,
Thread 31, CPU 0 (Node 7). Init time: 0.02822, exec time: 10.30749, InitEnergy: 0.
Total init time: 0.07872, Total exec time: 10.87855, Total run time: 10.95727, Ini
Finished iteration 2
Waiting while previous job results are processed by UPOP Job Scheduler...
Waiting while file "C:\Share\UPOP_Job_Scheduler\in\result.ujs" still exists, UPOP
Will wait 60 sec for acknowledgement polling every 5 sec...
Waiting 5 sec...
UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\in\result.ujs"
SUCCESS: UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\in\res
Waiting while file "C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml" still exists, U
Will wait 60 sec for acknowledgement polling every 5 sec...
Waiting 5 sec...
UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\in\result.ujs.x
SUCCESS: UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\in\res
UPOP Job Scheduler successfully processed experiment results, going for next job..
Sent "start" command to UPOP Job Scheduler.
Acknowledging "start" command has been recived by UPOP Job Scheduler, will wait 60
Waiting while file "C:\Share\UPOP_Job_Scheduler\cmd\start.ujsc" still exists, UPOP
Will wait 60 sec for acknowledgement polling every 5 sec...
Waiting 5 sec...
UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\cmd\start.ujsc"
SUCCESS: UPOP Job Scheduler has processed file "C:\Share\UPOP_Job_Scheduler\cmd\st
fx Waiting for incoming UPOP Job Requests from Google Cloud Services...
```

Name	Value
data_aquisition_file	'C:\Share\log.bin'
energy2finishjob	[]
energy2init	0
energy2initCPU0	0
energy2initCPU1	0
energy2run	[]
energy2runCPU0	[]
energy2runCPU1	[]
experiment_mat_file	'GASGET2_Model_
experiment_name	'GASGET2_Model_
experiment_result_...	'GASGET2_Model_
fid1	3
fid2	3
flg_file	'C:\Share\GASGET
gasget_ScopeData	<1x1 struct>
gasget_ScopeData1	<1x1 struct>
gasget_ScopeData2	<1x1 struct>
gasget_data	<1x1 struct>
init_flag_file	'GASGET_init.flg'

```
   s.stop
%-- 6/22/2013 7:51 PM --%
   myUJS.Next
%-- 6/22/2013 11:08 PM --
%-- 6/24/2013 7:09 AM --%
%-- 9/1/2013 5:58 PM --%
   plot gasget_ScopeData2
   simplot gasget_ScopeDat
   simplot
   simplot (gasget_ScopeD
```

```
2013/09/01 19:42:36 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:42:36 Will wait 5 sec...
2013/09/01 19:42:41 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:42:41 Will wait 5 sec...
2013/09/01 19:42:46 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:42:46 Will wait 5 sec...
2013/09/01 19:42:51 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:42:51 Will wait 5 sec...
2013/09/01 19:42:56 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:42:56 Will wait 5 sec...
2013/09/01 19:43:01 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:43:01 Will wait 5 sec...
2013/09/01 19:43:06 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:43:06 Will wait 5 sec...
2013/09/01 19:43:11 Matlab finished processing previous job.
C:\Share\UPOP_Job_Scheduler\in\result.ujs does not exist.
C:\Share\UPOP_Job_Scheduler\in\result.ujs.xml does not exist.
2013/09/01 19:43:11 Will wait 5 sec...
```

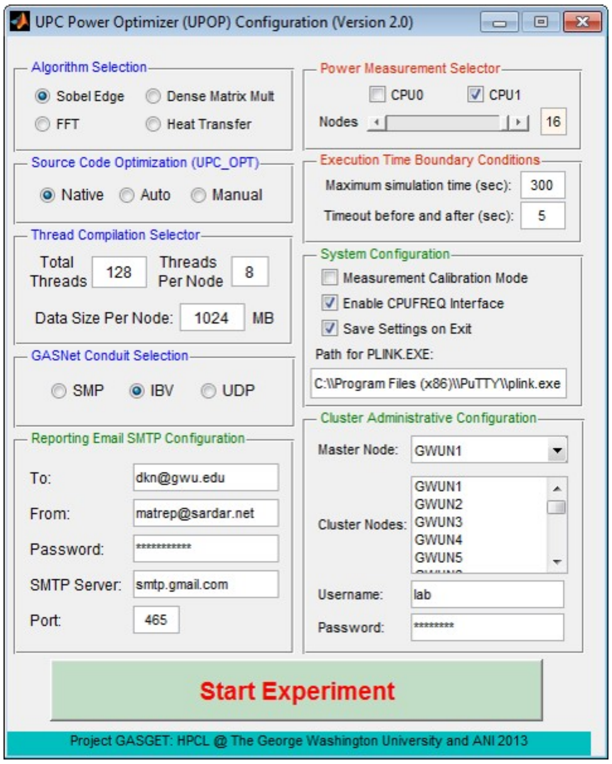
Figure 9. Console screen showing composite view of measurement control interface

design [1] relied on the program under investigation to output XML tags in real-time to which a system-level "listener" on the data acquisition computer would react for program and measurement duration control. We tried to compensate for the inherent system latencies by using an energy/time ratio correlated with the program's self-reported times for each phase of execution. Following the initial presentation of this research [1] we redesigned the data acquisition process to compensate for these latencies, though not without an increase in the overall complexity of the measurement system. At the same time, we rebuilt the user interface and experiment job control sub-systems to improve ease-of-use and permit iterative experimental runs and post-experiment data analysis.

C. MATLAB

Captured data was recorded with and processed with the 64-bit version of MATLAB/Simulink [41] running in a 64-bit Microsoft Windows environment. Figure 9 shows the Google Cloud java interface in the lower left, the MATLAB console interface in the upper left), and Cluster composite current measurement trace for 32 measurement channels on the right. MATLAB was used not only to process the raw data being acquired from the measurement sources, it was also used to orchestrate the overall measurement system process control. Figure 1 shows the high-level process flow as a an experimental run is initiated.

Once the experimental program terminates, MATLAB performs an orderly shutdown of the measurement capture streams, and then performs post-execution data processing. Part of the post-processing step is to assemble these time-stamps (which are thread-specific) into an consolidated XML file. MATLAB picks up this XML file, associates the time-stamp data with the power measurement streams, integrates the data into energy consumption values. The system can be loaded to run for days or weeks of continuous execution to exhaustively explore a particular set of input parameter permutations (e.g., thread count, network interconnect, CPU frequencies, TurboBoost states, etc.).


UPC Power Optimizer (UPOP) Configuration (Version 2.0)

Algorithm Selection: [✓ Sobel Edge] Dense Matrix Mult FFT Heat Transfer

Power Measurement Selector: CPU0 [✓ CPU1]; Nodes: 16

Source Code Optimization (UPC_OPT): [✓ Native] Auto Manual

Execution Time Boundary Conditions: Maximum simulation time (sec): 300; Timeout before and after (sec): 5

Thread Compilation Selector: Total Threads: 128; Threads Per Node: 8; Data Size Per Node: 1024 MB

System Configuration: Measurement Calibration Mode; [✓ Enable CPUFREQ Interface]; [✓ Save Settings on Exit]; Path for PLINK.EXE: C:\\Program Files (x86)\\PuTTY\\plink.exe

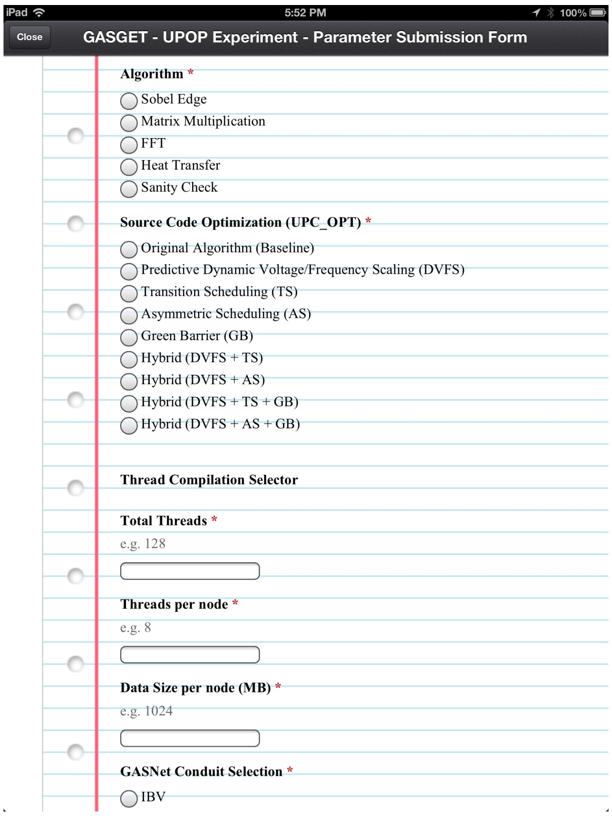
GASNet Conduit Selection: SMP [✓ IBV] UDP

Cluster Administrative Configuration: Master Node: GWUN1; Cluster Nodes: GWUN1, GWUN2, GWUN3, GWUN4, GWUN5; Username: lab; Password: ********

Reporting Email SMTP Configuration: To: dkn@gwu.edu; From: matrep@sardar.net; Password: ************; SMTP Server: smtp.gmail.com; Port: 465

Start Experiment
Project GASGET: HPCL @ The George Washington University and ANI 2013

Figure 10. Matlab UI for experiment parameter entry


GASGET - UPOP Experiment - Parameter Submission Form

Algorithm *
- Sobel Edge
- Matrix Multiplication
- FFT
- Heat Transfer
- Sanity Check

Source Code Optimization (UPC_OPT) *
- Original Algorithm (Baseline)
- Predictive Dynamic Voltage/Frequency Scaling (DVFS)
- Transition Scheduling (TS)
- Asymmetric Scheduling (AS)
- Green Barrier (GB)
- Hybrid (DVFS + TS)
- Hybrid (DVFS + AS)
- Hybrid (DVFS + TS + GB)
- Hybrid (DVFS + AS + GB)

Thread Compilation Selector

Total Threads * e.g. 128

Threads per node * e.g. 8

Data Size per node (MB) * e.g. 1024

GASNet Conduit Selection *
- IBV

Figure 11. Google UI (iOS) for experiment parameter entry

D. Graphical User Interface for Experimental Parameters

We designed a custom graphical user interface (GUI) to simplify the complex and often repetitive entry of the large number of experimental parameters. The first implementation of the measurement system [2] used a MATLAB generated GUI as shown in Figure 10 that was tightly integrated with the *Simulink* model. While the first GUI was effective for initiating experiments, it had two significant drawbacks. First, it suffered from a lack of an automated iteration capability. This was particularly important in light of the variation we were seeing in terms of energy utilization between successive experiments with the same parameter sets. To achieve a reliable and repeatable result usually involved running a particular experiment a minimum of five iterations and then selecting the *median value* of the set. Second, because the Simulink model itself did not provide an automated results logging mechanism thus experimental results needed to be manually entered in a spreadsheet before data analysis could be conducted. In order to address these issues, we designed a second generation GUI and results reporting mechanism using the Google Cloud Java API as described in Section 6-E and as shown in Figure 9 below.

E. Google Cloud User Interface and Job Control

This section describes how the measurement system's user interface (UI) and job control (JC) mechanisms work. The approach described here is a second generation technique that uses XML and Google Spreadsheets API [42] to extend the capability of MATLAB/Simulink's native scripting and user interface features beyond the design described in Section 6-D. As shown in Figure 1, an experimenter launches the UI by connecting to a Google Cloud URL. Once authenticated, the experimenter configures the parameters of the experiment he/she wishes to run using the UI shown in Figure 11. Upon submission of the form, a job queue spreadsheet (also hosted in the Google Cloud) is populated with the parameter configuration of the experiment.

We developed a custom polling agent using Google Cloud Java API libraries which runs as a Windows service. This agent, running on the MATLAB computer, picks up the job from the Google spreadsheet, deletes the active row, and triggers a MATLAB script to execute the experiment. When control passes to the Simulink model, a script creates a custom build string containing all the various parameters of the compilation and execution (that were set from the UI) of the UPC program running on the test bed cluster. The compilation and execution are monitored through an SSH console session. Upon the successful termination of the test program, another script retrieves the XML file that the program created, parses it, and then adds to each time field an associated energy measurement for the respective data acquisition channel.

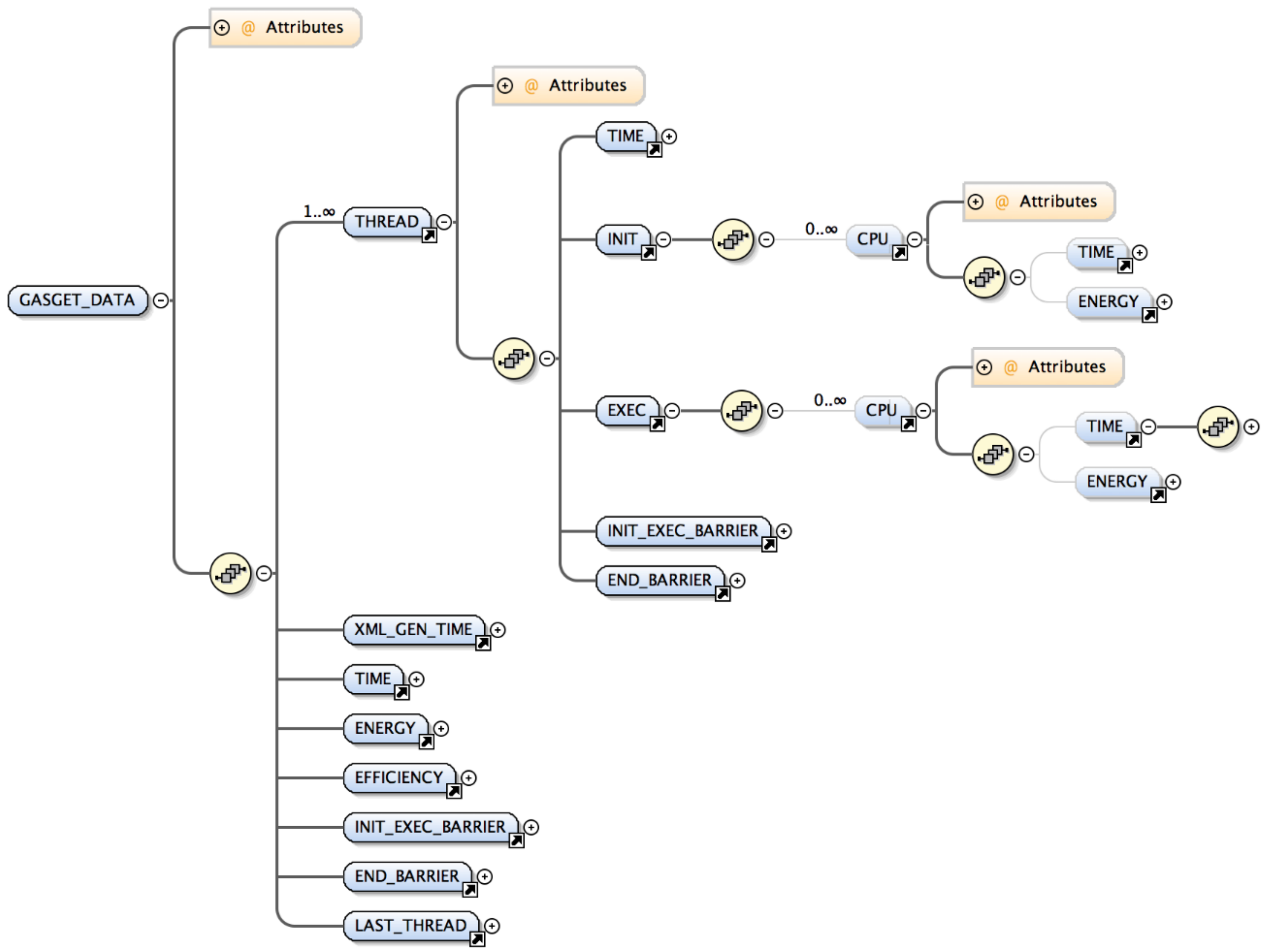


Figure 12. XML structure for CPU measurement data

The mathematical calculations performed by the *Simulink* model to compute the energy values from this data are described in Section 4.

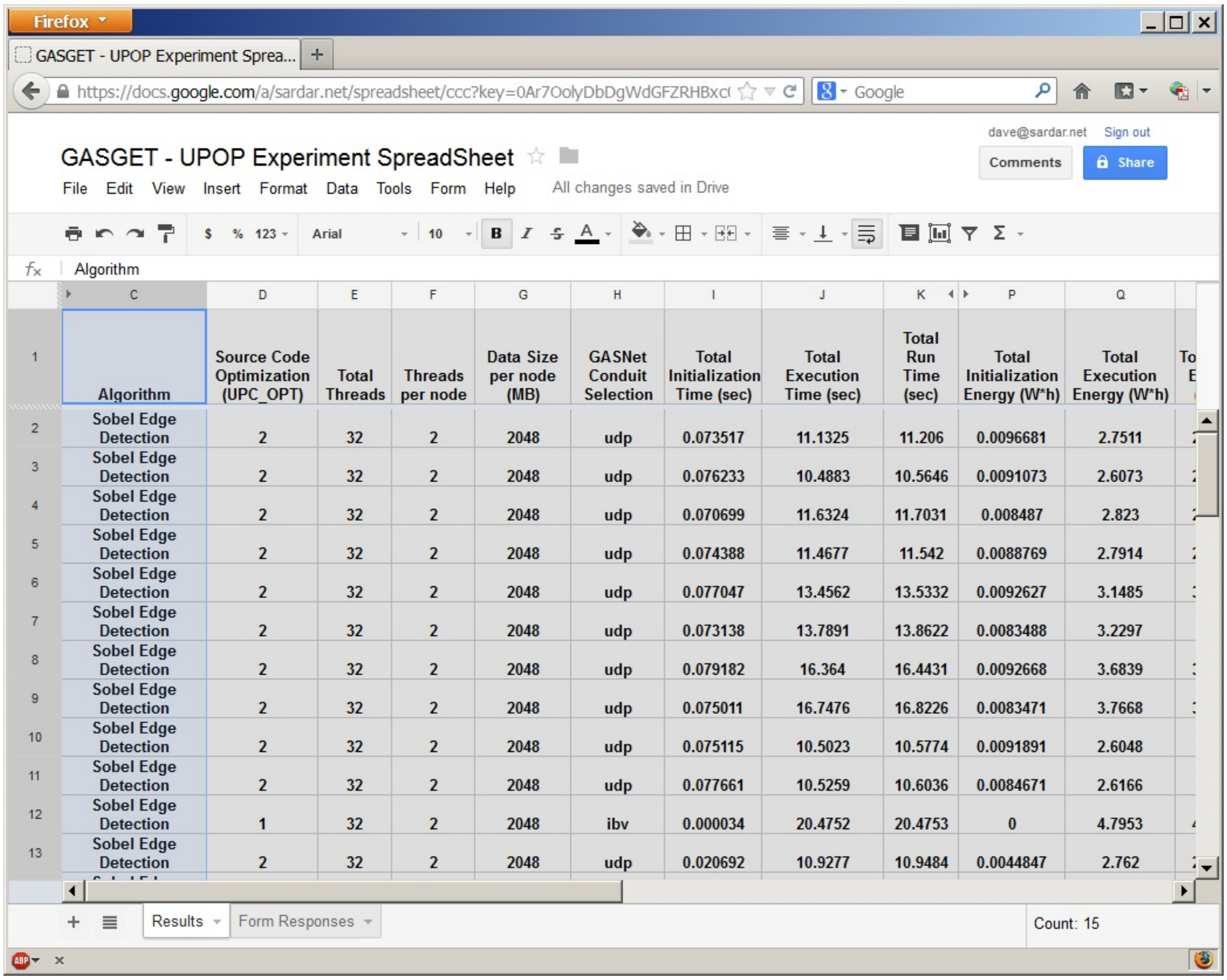
F. XML Data Structure

Our initial data capture approach assumed that threads executing on identical hardware nodes had execution times that differed within our margin of measurement error. During a set of detailed calibration exercises, we discovered that this was not true and that inherent system latencies and synchronization constructs could cause thread execution times to vary significantly. Since the precision of our energy measurement was largely dependent on the accuracy of our time measurements, this presented a serious issue. To address it, we subsequently developed an XML schema that would consolidate all the information needed for precise energy measurement on a multi-core SMP compute cluster.

The high level schema of our XML file is presented in Figure 12. Every thread generates its own start and stop times for each program phase. The node, CPU, core IDs, core frequency and `cpufreq` [43] power governors active at the time the timestamps are recorded are also captured. When a thread generates this section, only time information is populated since the thread does not have information about its energy usage. Energy information is inserted after the experiment by the MATLAB portion of measurement framework working backward through its captured power data and matching the measurement channel results against the corresponding thread activity based on the recorded timestamps.

G. Google Spreadsheets API

We developed an aggregate reporting mechanism to create a row in the Google spreadsheet for each iteration of an experiment. After sending the experiment report email, the new row is populated with the experiment specific data as shown in Figure 13. Once in the spreadsheet, it can be exported and manipulated with ease for the creation of charts, graphs, and deeper analytical reports.


Firefox
GASGET - UPOP Experiment Sprea...
https://docs.google.com/a/sardar.net/spreadsheet/ccc?key=0Ar7OolyDbDgWdGFZRHBxc
Google
dave@sardar.net Sign out
Comments
Share
File Edit View Insert Format Data Tools Form Help
All changes saved in Drive
Arial
10
Results
Form Responses
Count: 15

GASGET - UPOP Experiment SpreadSheet

fx Algorithm

	C	D	E	F	G	H	I	J	K	P	Q
1	Algorithm	Source Code Optimization (UPC_OPT)	Total Threads	Threads per node	Data Size per node (MB)	GASNet Conduit Selection	Total Initialization Time (sec)	Total Execution Time (sec)	Total Run Time (sec)	Total Initialization Energy (W*h)	Total Execution Energy (W*h)
2	Sobel Edge Detection	2	32	2	2048	udp	0.073517	11.1325	11.206	0.0096681	2.7511
3	Sobel Edge Detection	2	32	2	2048	udp	0.076233	10.4883	10.5646	0.0091073	2.6073
4	Sobel Edge Detection	2	32	2	2048	udp	0.070699	11.6324	11.7031	0.008487	2.823
5	Sobel Edge Detection	2	32	2	2048	udp	0.074388	11.4677	11.542	0.0088769	2.7914
6	Sobel Edge Detection	2	32	2	2048	udp	0.077047	13.4562	13.5332	0.0092627	3.1485
7	Sobel Edge Detection	2	32	2	2048	udp	0.073138	13.7891	13.8622	0.0083488	3.2297
8	Sobel Edge Detection	2	32	2	2048	udp	0.079182	16.364	16.4431	0.0092668	3.6839
9	Sobel Edge Detection	2	32	2	2048	udp	0.075011	16.7476	16.8226	0.0083471	3.7668
10	Sobel Edge Detection	2	32	2	2048	udp	0.075115	10.5023	10.5774	0.0091891	2.6048
11	Sobel Edge Detection	2	32	2	2048	udp	0.077661	10.5259	10.6036	0.0084671	2.6166
12	Sobel Edge Detection	1	32	2	2048	ibv	0.000034	20.4752	20.4753	0	4.7953
13	Sobel Edge Detection	2	32	2	2048	udp	0.020692	10.9277	10.9484	0.0044847	2.762

Figure 13. Google spreadsheet for output data

H. *Program Level Time-Energy Synchronization*

The program executing the test algorithm is augmented with a set of function calls to utility routines that record precise time stamps for each phase of its operation. We use the Linux API `gettimeofday` which provides microsecond level accuracy. These values are stored in a *thread local* data structure which also captures the *thread ID* and the associated CPU and processor core for the respective phase of the program. This data is then processed into an XML file at the end of the program run which is used to associate the measurement channel energy consumption with the time stamps as described in Section 6-F. Upon completion of each simulation (experiment run) the *Simulink* model's `StopFcn` routine uses a java class to send an email report to the email address specified in the GUI.

7. Experimental Results

In this section, we provide examples that show the utility of our measurement system described above. We investigate two benchmarks both written in UPC: `Fast Fourier Transform` (FFT) and `Heat Transfer` (HT). Both benchmarks are non-trivial applications, representative of common HPC codes, for which several hand-optimizations (three for FFT, two for HT) were coded. We successively ran each benchmark version using 8, 16, 32, and 64 total threads, evenly distributed across the cluster. For the experiments shown, all nodes participate in every experiment, with at least one active core per processor.

Since the cluster nodes are actively interconnected by two different network fabrics (described in Section 5-A) we sequentially execute each benchmark on each network type. The network is selected at compile time by selecting the UPC *GASNet* [34] conduit: `ibv` (Infiniband) or `udp` (Ethernet). Though not configurable at compile/run time (because it requires a system re-boot), the machines

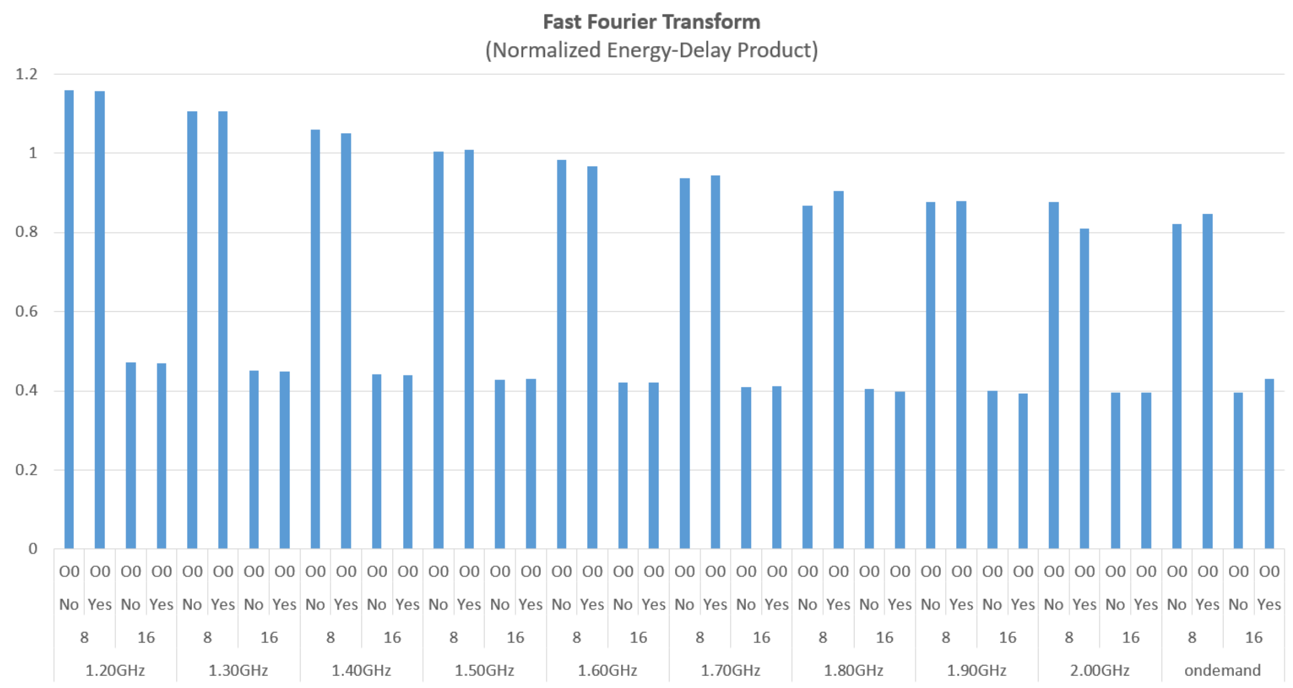


Figure 14. Normalized Fast Fourier Transform as function of frequency steps

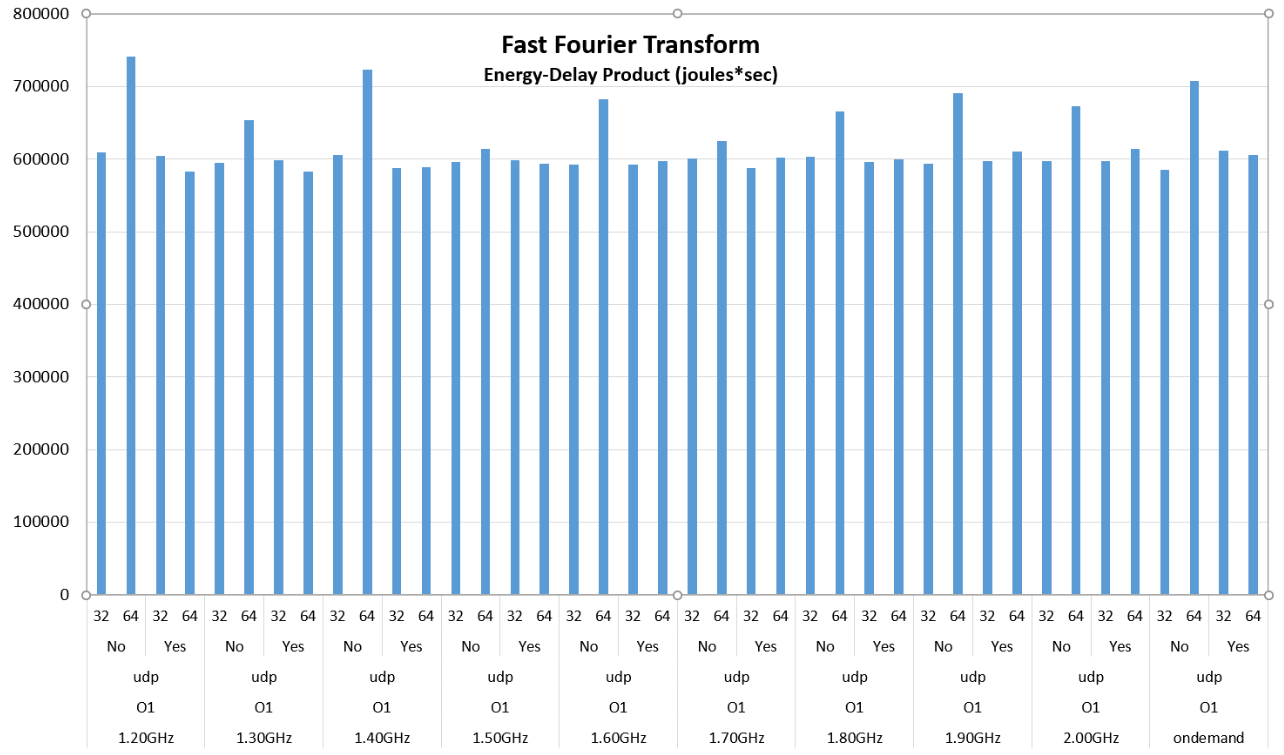


Figure 15. FFT energy consumption (joules) vs. frequency

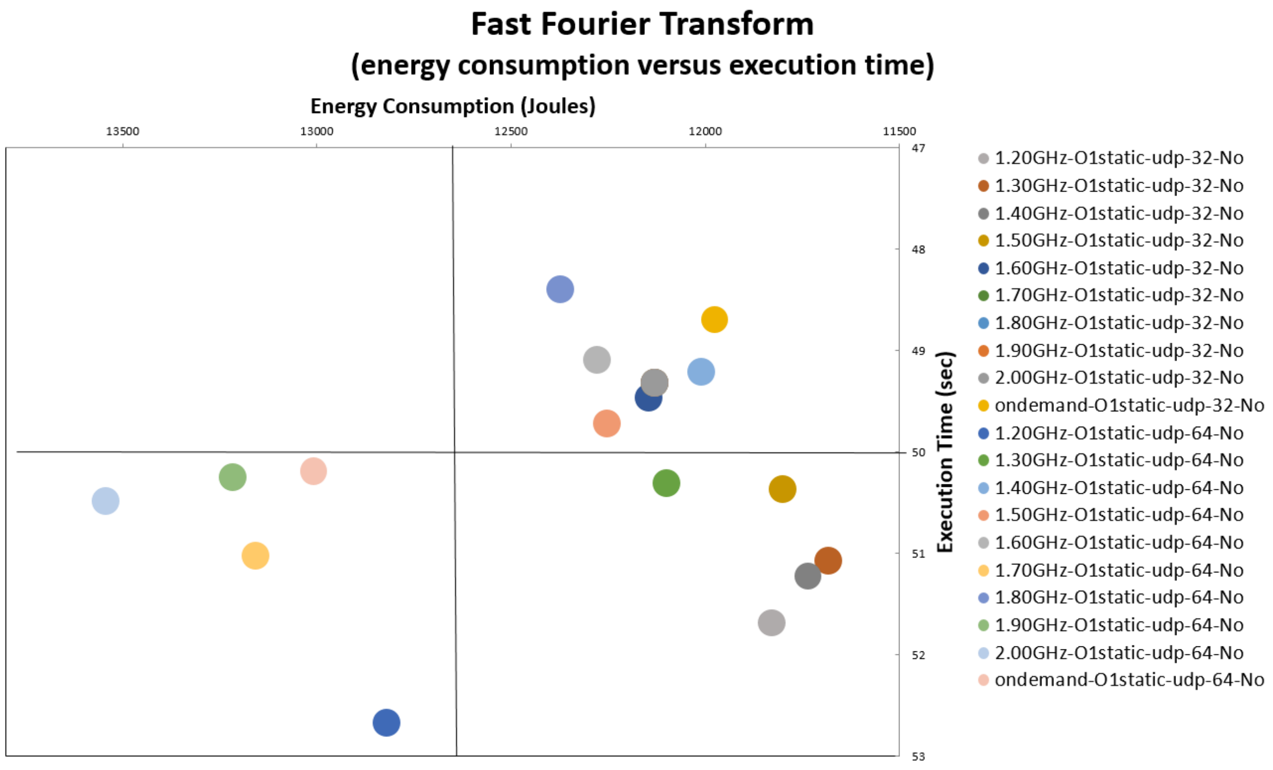


Figure 16. FFT Magic Quadrant for execution time (`seconds`) vs. energy consumption (`Joules`) for FFT

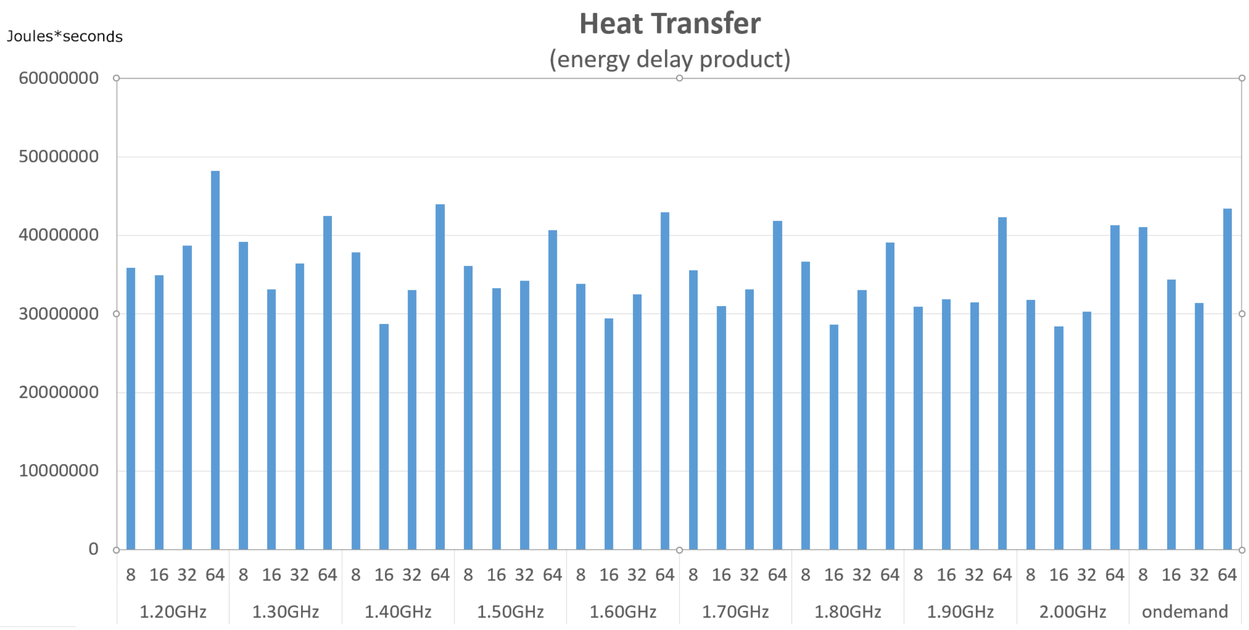


Figure 17. Heat Transfer energy-delay product (`Joule*seconds`) as a function of thread count and frequency steps

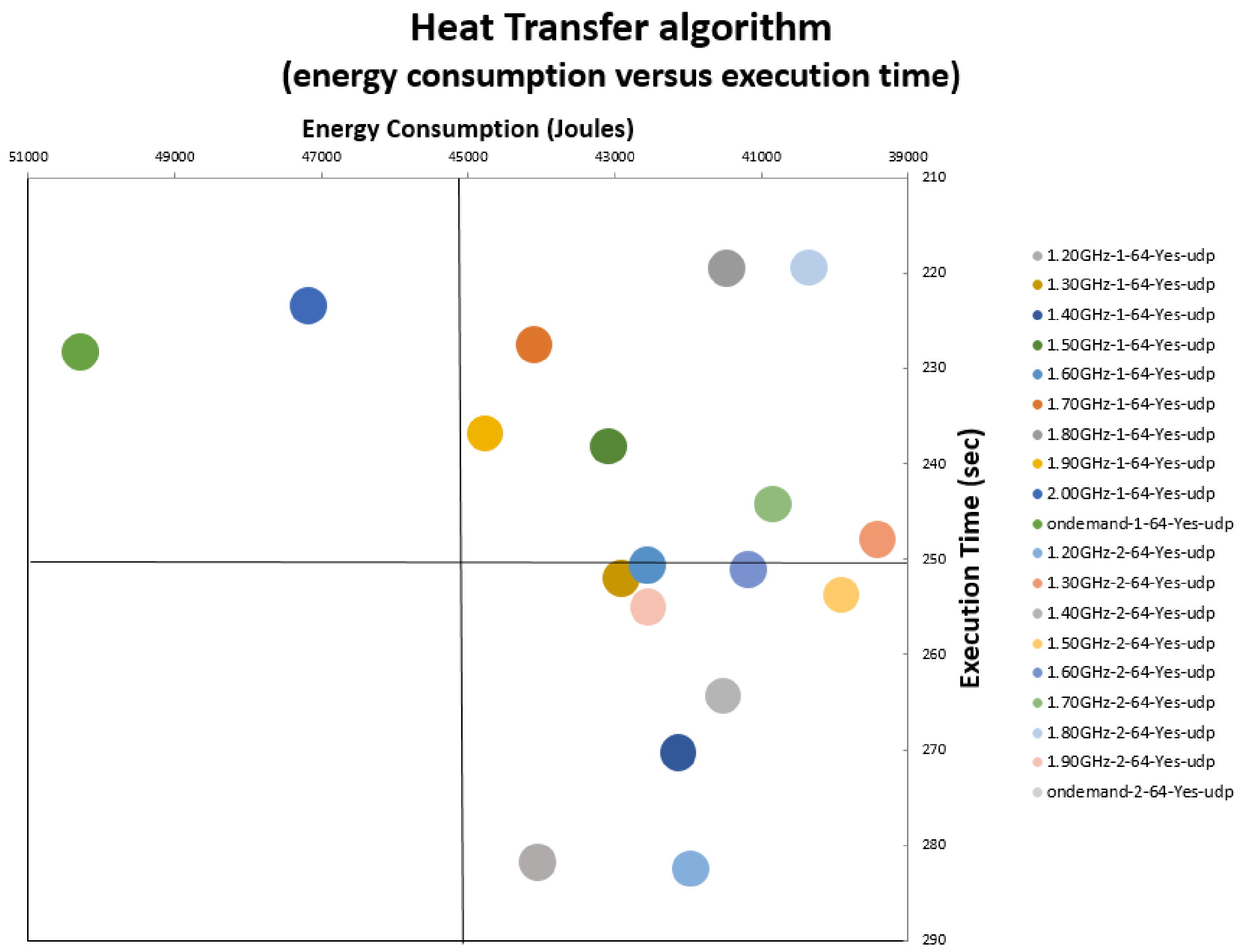


Figure 18. Magic Quadrant for execution time (seconds) vs. energy (Joules) consumption for the Heat Transfer benchmark

can be BIOS configured to have Intel® 's CPU Turbo-Boost [16] feature ON or OFF. We denote ON and OFF as Yes and No, respectively in the Figures displaying the results.

The Intel® *Ivy Bridge* Xeon™ processors have a total of 9 fixed frequency steps, and a workload-adaptive variable frequency mode (referred to as the ondemand CPU governor [43]). The results that are shown were for experiments designed to systematically discover the combination of all the control factors that produced the minimum energy consumption. The Magic Quadrant charts show the execution energy and time in a combined (two axis) manner. These graphs show the energy/time correlation without performing the multiplication that would show a typical energy/delay product [44]. The advantage of the two-axis charts is that the values of the energy consumption and execution time are not lost in the calculation (and subsequent display) of the energy-delay product, yet the two values can be shown in a correlated fashion as a single bubble in the chart. The upper right-hand quadrant is where the minimum energy and time values are reported for the specific combination of frequency, thread count, network interface, and TurboBoost state.

Figure 14 shows the normalized energy-delay product of Fast Fourier Transform as a function of frequency steps running with udp with 8 and 16 threads and with the O0 code variant (hand optimization). The data is normalized using the 8 threads at 1.2GHz as the base case. The TurboBoost state is shown as YES or NO. We normalized the data to the 8 threads at 1.2GHz base case. Executing 16 threads (2 threads per CPU) consistently outperforms 8 threads, all other factors held constant. The best performance for 8 threads takes place at 2GHz with TurboBoost ON, while for the 16 thread execution, the best performance is with TurboBoost OFF and the ondemand frequency governor enabled.

Figure 15 shows FFT energy consumption in joules vs frequency under 32 and 64 threads with TurboBoost ON and OFF, with udp and using the O1 code variant. With TurboBoost ON and 64 threads, the minimum energy consumption occurs at 1.2GHz. With TurboBoost OFF, 32 threads has the lowest energy usage with the ondemand governor enabled.

Figure 16 shows the Magic Quadrant for execution time vs. energy consumption for FFT running with the O1Static variant, 32 and 64 threads with TurboBoost OFF. The best (lowest) energy/time values are shown in the top right corner. The ondemand frequency governor with 32 threads achieves the lowest energy consumption but not best performance (shortest time). Second, the 1.8GHz with 64 threads achieves the best performance, but at the cost of slightly higher energy consumption than the other option. These kinds of results can only be determined through a direct measurement system as predictive models cannot provide granular performance differentiation across a range of frequency steps.

Figure 17 shows the energy-delay product (see Section 2-A) for the Heat Transfer Application in Joule*seconds as a function of thread count (8 to 64) for all frequency steps. The lowest energy-delay product occurs at 1.6GHz with 16 threads. Execution with 16 threads does better than any other thread count in terms of energy-delay product.

Figure 18 shows the Magic Quadrant for execution time in seconds against energy consumption in joules for the Heat Transfer application. The figure compares two hand-coded optimizations and 64 threads with TurboBoost ON. The lowest energy consumption/shortest execution time is shown in the top right corner. Both optimizations do best at 1.8GHz.

8. Conclusions and Future Work

In this paper we have presented an extensible CPU power measurement framework that supports our own research but is also generally applicable to the computer engineering community in general for accurate computational power consumption measurements. Our results illustrate the need for a precise direct energy measurement capability since it is very hard to predict the operating parameters (e.g., frequency, thread count, TurboBoost, and network interface) that will achieve the lowest energy consumption.

In our future work we are planning the design for a more scalable direct measurement method by moving the data acquisition system directly into the node chassis. This system could potentially capture real-time energy measurements from cluster systems with large node/CPU counts. In addition, we are investigating how to calculate the actual power consumption of individual CPU cores within the restriction of measuring the power consumption of the aggregate CPU. Finally, we are exploring how to automatically profile a particular CPU or system for the power optimization feature set it exposes to operating system or program level control so that the total space of control combinations can be systematically examined.

Acknowledgment

This paper is a significant extension and update of a paper that appeared in the proceedings of the Workshop on Power Measurement and Profiling of in conjunction with the 4th International Green Computing Conference in June 2013 [2]. All trademarks ™ and registered trademarks ® mentioned, cited, or referenced in this document remain the property of their respective owners.

References

[1] Newsom, Azari, Anbar, and El-Ghazawi, "Predictive Energy Management Techniques for PGAS Programming," in *Proceedings of the 2013 ACS International Conference on Computer Systems and Applications*, ser. AICCSA '13. Ifrane, Morroco: IEEE, May 2013, pp. 1–8.

[2] Newsom, Azari, Anbar, and El-Ghazawi, "Granular CPU Power Measurement for SMP Clusters," in *Workshop Proceedings of the 2013 International Green Computing Conference*, ser. IGCC '13. Arlington, VA: IEEE, Jun. 2013, pp. 1–6.

[3] Newsom, Azari, Anbar, and El-Ghazawi, "Locality-Aware Power Optimization and Measurement Methodology for PGAS Workloads on SMP Clusters," in *Proceedings of the 2013 International Green Computing Conference*, ser. IGCC '13. Arlington, VA: IEEE, Jun. 2013, pp. 1–10.

[4] Cui, Zhu, Bao, and Chen, "A Fine-Grained Component-Level Power Measurement Method," in *Proceedings of the 2011 International Green Computing Conference and Workshops*, ser. IGCC '11. Orlando, FL: IEEE, Jul. 2011, pp. 1–6.

[5] Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., ser. The Morgan Kaufmann Series in Computer Architecture and Design. Waltham, MA: Elesvier, 2012.

[6] Gonzalez and Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277 – 1284, Sep. 1996. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=535411

[7] Eisenmann and Kohl, "Power Calculation for CMOS Gate Arrays," in *Proceedings of the Fourth Annual IEEE International ASIC Conference and Exhibit*, ser. ASIC '91. Rochester, NY, USA: IEEE, Sep. 1991, p. 5.

[8] Davari, Dennard, and Shahidi, "CMOS Technology for Low Voltage/Low Power Applications," in *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference*, ser. CICC '94. San Diego, CA, USA: IEEE, May 1994, pp. 3–10.

[9] Horowitz, Indermaur, and Gonzalez, "Low-Power Digital Design," in *Digest of Technical Papers of the 1994 IEEE Low Power Electronics Symposium*, ser. IEEE Low Power Electronics. San Diego, CA, USA: IEEE, Oct. 1994, pp. 8–11.

[10] Frank, Solomon, Reynolds, and Shin, "Supply and Threshold Voltage Optimization for Low Power Design," in *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, ser. ISPLED '97. Monterey, CA, USA: ACM, Aug. 1997, pp. 317–322.

[11] Oklobdzija, "Design for Low Power," in *High-Performance System Design: Circuits and Logic*. Wiley-IEEE Press, 1999, pp. 169–259. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5265218

[12] Kim, Austin, Baauw, Mudge, Flautner, Hu, Irwin, Kandemir, and Narayanan, "Leakage Current: Moore's Law Meets Static Power," *Computer*, vol. 36, no. 12, pp. 68–75, Dec. 2003, computer, the flagship publication of the IEEE Computer Society, publishes peer-reviewed articles written for and by professionals representing the full spectrum of computing technology from hardware to software and from current research to new applications. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1250885

[13] Intel Corporation, "Intel® Xeon® Processor E5-2640 v2 Data Sheet," 2013. [Online]. Available: http://ark.intel.com/products/75267/Intel-Xeon-Processor-E5-2640-v2-20M-Cache-2_00-GHz

[14] Intel Corporation, "Using Enhanced Intel SpeedStep® features in HPC Clusters | Intel® Developer Zone," Aug. 2009. [Online]. Available: https://software.intel.com/en-us/articles/using-enhanced-intel-speedstep-features-in-hpc-clusters

[15] Advanced Micro Devices, Inc., "AMD PowerNow!™ Technology," Nov. 2000, publication #: 24404 Rev: A. [Online]. Available: http://support.amd.com/TechDocs/24404a.pdf

[16] Intel Corporation, "Intel® Turbo Boost Technology 2.0," 2015. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[17] Rotem, Naveh, Rajwan, Ananthakrishnan, and Weissmann, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6148200

[18] Jahagirdar, George, Sodhi, and Wells, "Power Management of the Third Generation Intel Core Micro Architecture Formerly Codenamed Ivy Bridge," Cupertino, CA, USA, Aug. 2012. [Online]. Available: http://hotchips.org/wp-content/uploads/hc_archives/hc24/HC24-1-Microprocessor/HC24.28.117-HotChips_IvyBridge_Power_04.pdf

[19] Hammarlund, Kumar, Osborne, Rajwar, Singhal, D'Sa, Chappell, Kaushik, Chennupaty, Jourdan, and others, "Haswell: The Fourth-Generation Intel Core Processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, Mar. 2014. [Online]. Available: http://www.computer.org/csdl/mags/mi/2014/02/mmi2014020006-abs.html

[20] Charles, Jassi, Ananth, Sadat, and Fedorova, "Evaluation of the Intel® Core™ i7 Turbo Boost feature," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization*, ser. IISWC 2009. Austin, TX: IEEE, Oct. 2009, pp. 188–197.

[21] Kidd, "Power Management States: P-States, C-States, and Package C-States," Apr. 2014. [Online]. Available: https://software.intel.com/en-us/articles/power-management-states-p-states-c-states-and-package-c-states

[22] Hackenberg, Schone, Ilsche, Molka, Schuchart, and Geyer, "An Energy Efficiency Feature Survey of the Intel® Haswell Processor," in *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, ser. IPDPSW '15. Hyderabad, India: IEEE, May 2015, pp. 896–904, 00009.

[23] Advanced Micro Devices, Inc., "Opteron™ 2354 Processor Data Sheet," Jun. 2010. [Online]. Available: http://support.amd.com/TechDocs/43374.pdf

[24] James H Laros, Phil Pokorny, and David DeBonis, "PowerInsight - a Commodity Power Measurement Capability," in *Proceedings of the 2013 International Green Computing Conference*, ser. IGCC '13. Arlington, VA: IEEE, Jun. 2013, pp. 1–6.

[25] James H. Laros III, Kevin Pedretti, Suzanne M. Kelly, Wei Shu, Kurt Ferreira, John Van Dyke, and Courtenay Vaughan, *Energy-Efficient High Performance Computing*, 1st ed., ser. SpringerBriefs in Computer Science. Berlin Heidelberg: Springer International Publishing AG, 2013, vol. 1.

[26] Bircher and John, "Complete System Power Estimation Using Processor Performance Events," *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 563–577, Apr. 2012. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5714687&isnumber=6156497

[27] Huck, "Measuring Processor Power TDP vs. ACP," Apr. 2011, revision: 1.1. [Online]. Available: http://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf

[28] Pakin and Lang, "Energy Modeling of Supercomputers and Large-Scale Scientific Applications," in *Proceedings of the 2013 International Green Computing Conference*, ser. IGCC '13. Arlington, VA: IEEE, Jun. 2013, pp. 1–6.

[29] W3C Information and Knowledge Domain, "Extensible Markup Language (XML)," May 2015. [Online]. Available: http://www.w3.org/XML/

[30] Kerrisk, *The Linux Programming Interface*. San Francisco, CA: No Starch Press, Oct. 2010.

[31] Johnson, "Notes on the Convergence of Trapezoidal-Rule Quadrature," 2010. [Online]. Available: http://mitocw.fermielearning.it/courses/mathematics/18-335j-introduction-to-numerical-methods-fall-2010/lecture-notes/MIT18_335JF10_Lec31a_hand.pdf

[32] Intel Corporation, "Intel Performance Tuning Utility 4.0 Update 5 - Release Notes," Apr. 2011. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-tuning-utility-product-overview

[33] Upton, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal Q*, vol. 1, no. Q1, p. 12, 2002, 00003.

[34] Bonachea and Jeong, "GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages," Nov. 2006, version 1.8. [Online]. Available: https://gasnet.lbl.gov/dist/docs/gasnet.pdf

[35] National Instruments, Inc, "Operating Instructions and Specifications-NI 9229 Voltage Module," Jul. 2014. [Online]. Available: http://www.ni.com/pdf/manuals/374184l.pdf

[36] National Instruments, Inc, "Operating Instructions and Specifications-NI 9227 In-Line Current Module," Jul. 2014. [Online]. Available: http://www.ni.com/pdf/manuals/375101e.pdf

[37] Intel Corporation, “ATX Motherboard Specification 2.2,” 2004, version 2.2. [Online]. Available: http://www.formfactors.org/developer%5Cspecs%5Catx2_2.pdf

[38] ASTM International, “Standard Specification for Standard Nominal Diameters and Cross-Sectional Areas of AWG Sizes of Solid Round Wires Used as Electrical Conductors,” Apr. 2014. [Online]. Available: http://www.astm.org/Standards/B258.htm

[39] CentOS Community, “CentOS Project,” 2015, community ENTerprise Operating System. [Online]. Available: https://www.centos.org/about/

[40] California, “Berkeley Unified Parallel C (UPC) Project,” Apr. 2015. [Online]. Available: http://upc.lbl.gov/

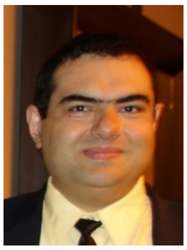
[41] MathWorks, “MATLAB,” Sep. 2015. [Online]. Available: http://www.mathworks.com/products/matlab/

[42] Google, Inc., “Google Spreadsheets API Version 3.0,” Aug. 2015. [Online]. Available: https://developers.google.com/google-apps/spreadsheets/

[43] Pallipadi and Starikovskiy, “The Ondemand Governor,” in *Proceedings of the 2006 Linux Symposium*, ser. OLS ’06, vol. 2. Ottawa, Ontario, Canada: Linux Symposium, Jul. 2006, pp. 215–230.

[44] Freeh, Lowenthal, Pan, Kappiah, Springer, Rountree, and Femal, “Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, Jun. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4180349

David K. Newsom David K. Newsom earned his Ph.D. (2015) in Computer Engineering at The George Washington University’s School of Engineering and Applied Science where he has been conducting research in green high performance and parallel computing. He holds a M.Sc. degree in Electrical Engineering from Johns Hopkins University (1992), a B.Sc. degree in Electrical Engineering from the University of Maryland (1990), and a B.A. degree in International Relations from San Francisco State University (1983). Dr. Newsom has been a staff member at the World Bank Group since 1996, where he currently holds the position of Senior Manager, Enterprise Platforms in the central IT department. Prior to the World Bank Group, he was part of the NASA Antarctic Field Team that deployed the first geostationary satellite communications system to support telephony and data communications from the Geographic South Pole in 1994. Dr. Newsom has been a member of the IEEE and ACM since 1988.



Sardar F. Azari Sardar F. Azari is an independent researcher, scientist, and hobbyist who has continued to conduct academic research and development in conjunction with his 28 year career as an IT professional. He is currently a senior staff member and Lead for the Enterprise Search and Data Analytics Team of the World Bank Group which he joined in 1999. He was awarded his Ph.D. (2003) and M.Sc (1993) in Electrical and Computing Engineering from Azerbaijan State Oil Academy. Dr. Azari has published many research publications in his area of academic interest since 1990, which include green computing, electrical power measurement and instrumentation, smart sensors and soft computing (fuzzy logic and ANN’s). He has been an IEEE Member since 2000.

Olivier Serres Olivier Serres earned his Ph.D. (2015) in Computer Engineering at The George Washington University’s School of Engineering and Applied Science. He is a postdoctoral research scientist at GWU’s High Performance Computing Laboratory. He obtained a research Master’s degree (2008) in Mechatronics and a B.S. (2007) in Computer Science from the University of Technology of Belfort-Montbeliard, France. His research interests include high-performance computing and reconfigurable computing. Dr. Serres’s primary research focus is programming models for high-performance many-core based and heterogeneous systems.

Abdel-Hameed A. Badawy Abdel-Hameed A. Badawy is a lead research scientist at GWU High Performance Computing Laboratory (HPCL) and is a tenure-track assistant professor in the Department of Electrical Engineering at Arkansas Tech University. He received his Ph.D. and M.Sc. both from the University of Maryland, College Park, in Computer Engineering. His research interests include locality optimizations, interactions of computer architectures and compilers, machine intelligence techniques and their applications to computer architecture, and Green Computing. He has published in ACM/IEEE conferences and Journals. He is a Senior Member of the IEEE, IEEE Computer Society, and a Professional member of the ACM. He serves as the vice chair of the Arkansas River Valley IEEE section.

Tarek El-Ghazawi Tarek El-Ghazawi is a Professor in the Department of Electrical and Computer Engineering at The George Washington University, where he leads the university-wide Strategic Academic Program in High-Performance Computing. He is the founding director of The GW Institute for Massively Parallel Applications and Computing Technologies (IMPACT) and a founding Co-Director of the NSF Industry/University Center for High-Performance Reconfigurable Computing (CHREC). El-Ghazawi's research interests include high-performance computing, computer architectures, reconfigurable, embedded computing and computer vision. He is one of the principal co-authors of the UPC parallel programming language and the first author of the UPC book from John Wiley and Sons. He received his Ph.D. degree in Electrical and Computer Engineering from New Mexico State University in 1988. Dr. El-Ghazawi has published more than 200 refereed research publications in this area. Dr. El-Ghazawi has served in many editorial roles and is currently an Associate Editor for the IEEE Transactions on Computers. He has chaired and co-chaired many international conferences and symposia including the 2009 Conference on Partitioned Global Address Space (PGAS) Pro-gramming Models and Languages (PGAS2009), The 10th IEEE International Conference on Scalable Computing and Commu-nications (ScalCom-10), 2010, and the 9th ACS/IEEE Confer-ence on Computer Systems and Applications, AICCSA2011. Dr. El-Ghazawi's research has been frequently supported by Federal agencies and industry including DARPA/DoD, NSF, DoE/LBNL, NASA, IBM, HP, Intel, AMD, SGI, Microsoft, and Mellanox. Professor El-Ghazawi is a Fellow of the IEEE and a Research Faculty Fellow of the IBM Center for Advanced Studies, Toronto. He is a member of the Phi Kappa Phi national honor society and an elected member of the IFIP WG10.3.