CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–**??**
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Detailed and simultaneous power and performance analysis

Harald Servat[*], Germán Llort, Judit Giménez and Jesús Labarta

c/Jordi Girona, 29. Barcelona, 08034, Catalunya, Spain.

SUMMARY

In the road to the Exascale computing, it is known that the target is not only to increase the performance, but also to achieve an affordable level of power drained by such kind of systems. The energy issue needs to be tackled at different levels, from the system level to the processor level. There are studies that show that the processor itself is the component of the system that is responsible for most of the energy consumed.
Performance tools will play an important role to make the applications take benefit of the performance of these systems. These tools can be extended to provide power metrics and thus report for each region of code its energy consumed in addition to the performance achieved. We present in this paper a performance tool that takes advantge of recent processor capabilities to measure its own power consumption. The results of the tool are passed to a mechanism called folding that produces detailed metrics and source code references by using coarse grain sampling. We have used the tool with multiple serial benchmarks and also with some MPI applications to demonstrate its usefulness by locating hotspots in terms of performance and power drained.
Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: High Performance Computing; Performance analysis; Power analysis; Instrumentation; Sampling

1. INTRODUCTION

Supercomputers have improved their performance in every new generation, but as supercomputers grow, so does their energy consumption [1, 2]. This tendency will continue in the road to the Exascale computing during the forthcoming years [3, 4]. While in the past the supercomputer community has mainly targeted to increase the performance efficiency to reduce the amount of time needed to execute the applications, now they are focusing also on the power efficiency. In either the performance or power context, it is important to have a mechanism to measure the efficiency of a system before having any possibility of improving it. Some experiments [5] have shown that the CPU is responsible for half of the power needed by the whole system. Thus, as it happened in the performance analysis area, it is important to observe the evolution of the power in order to ultimately reduce it [6].

In the performance context, performance tools are pieces of software to assist in the optimization of applications by giving comprehensive details of their inefficiencies. These tools use different techniques to inject monitors into the application so as to get information as the application runs. The monitors are responsible for gathering performance metrics (including time, number of occurrences, hardware counters) so as to allow a subsequent analysis that correlates the metrics with the application region of code. Tools like Scalasca [7], Vampir [8], HPCToolkit [9], TAU [10] and Paraver [11] have not only proven useful by providing different levels of insights that lead the user

[*]Correspondence to: Harald Servat. E-mail address: harald.servat at bsc.es, Postal address: c/Jordi Girona, 29, Barcelona, 08034, Catalunya, Spain.

to understand and even improve the performance of their application, but also gather information of the application in a simple fashion without excessive overhead.

Opposite to the performance monitoring, power monitoring is rather more complex. There are several methods to evaluate the power consumption: by attaching a powermeter that samples the consumed power to the power supply unit by the whole system or by individual components [12, 13, 14, 15], by simulating and estimating the CPU power consumption by using low-level simulators [16, 17, 18, 19], or by using power models derived from the performance counters [20, 21, 22]. In order to provide detailed performance and power measurements, adding a device to each node of a in production supercomputer is not a feasible approach because most of the supercomputers have the physical access restricted and it would result into a costly solution. Using low-level simulators is not feasible either because they cannot handle whole executions of applications, and when the execution is reduced, they need long execution times to provide the results. Finally, power models derived from the performance counters need several performance counters to be read simultaneously, a capability that cannot be always be fulfilled by the processor, and also the power model may vary between different versions of the same processor. To alleviate such problem, Intel has recently introduced into their SandyBridge processors the Running Average Power Limit (RAPL) infrastructure [23, 24]. RAPL offers a mechanism to limit, control and monitor of the power and energy usage of a single processor socket and PAPI [25] has added a component[†] that interfaces the RAPL monitors thus offering a seamless integration to performance tools that already use PAPI. The inclusion of this component into PAPI, allows the aforementioned performance tools to integrate the power and energy metrics without any modification and gain a correlation between the power consumption with other performance metrics or even the application source code.

In this paper we present a performance tool that uses the RAPL infrastrscture to measure the energy consumption. We demonstrate that power and energy consumption can be studied in a similar way to performance when analyzing full application executions in production environments. Here we show and analyze the energy consumption and efficiency of multiple benchmarks and parallel applications in different execution conditions. While most of the tools typically provide average performance information at application or routine level, our work focus on providing instantaneous metrics of the energy and performance behavior along the application code. To do so, we take the advantage of one existing performance tool, named folding [26], that finely describes the behavior of the performance of the application and we adapt it to use the energy metrics. We will present the instantaneous power consumption of these applications compared to the application performance and activity and will also analyze the results. We will also study whether the Dynamic Voltage and Frequency Scaling (DVFS) mechanism impacts the overall performance and energy behavior of the applications, and also the effects of using the multiple cores available in the socket.

The rest of the paper is structured as follows. In the following section, we briefly review the related work. In section 3 we introduce the RAPL mechanism as described by Intel which is used to gather the power and energy usage. In section 4 we present the mechanism used to extremely detail the evolution of both power and performance in a region of code by using coarse grain sampling and we also present the modifications applied to the mechanism to accommodate the counters offered by RAPL. We continue by doing a thorough study of the power and performance of different benchmarks and parallel applications in Section 5. Finally, we draw some conclusions in Section 6.

2. RELATED WORK

We present in this section different approaches to generate reports of power consumption. As we mentioned, there are several methods to analyze the power consumption of a system, but contrary to our solution, none of them report instantaneous performance and power measurements for full

[†]Available in their GIT repository when this paper was written and it will be publicly available in PAPI version 5.0.

application runs in production conditions. Here we select and summarize some of the work that use each of these methods and we compare them with our work.

PowerMon [12] is a device intended to produce accurate, fine-grained power measurements in computing systems. The device interfaces directly with most computing systems using the standard ATX pin configuration and provides fine measurement by using sampling (with a frequency up to 1024 Hz). It can monitor disks, graphical processing units (GPUs) and other peripherals, as opposite to RAPL. RAPL, however, does not need neither need to plug any device into the computer nor using a special piece of software. The usage of PAPI to access the RAPL infrastructure allows an easy mechanism to gather RAPL instead of requiring physical access as needed by PowerMon. We combine performance and power metrics so as to study their possible correlations in order to mitigate the overall energy consumption. Our solution uses the folding mechanism, which converts the performance and power data gathered using sampling into a continuous function of time, allowing us to provide much more finer details, even at microsecond level.

Wattch [18] is a cycle-level architectural simulator that estimates CPU power consumption of every component of the socket by applying power consumption models. This type of simulation requires full instrumentation of the application, making the power estimation of the full execution unmanageable, not only because the data size to be gathered, but also because the time needed to produce the results. The solution we propose takes advantage of current performance tools, to integrate the power consumption into their available metrics. These performance tools typically show small or negligible overhead, giving the user or the analyst the possibility to analyze the full execution.

Bertran and others presented in [20] a methodology for producing power models based on performance monitoring counters. They focus on the responsiveness of the model, *i.e.* a model that rapidly adapts to power changes. The inputs of the model proposed are component activity ratios, which summed up results in a power estimation of the whole system. The work we propose takes advantage of a component of the CPU that works as a powermeter by estimating the power and energy consumption based on the processor execution. This component saves us using additional performance counters, which would end up in implementing a multiplexing algorithm to gather all the performance counters needed with the consequent accuracy loss, and also avoids the need to generate and, specially, to validate a power model for each processor. In our work we do rely on the monitors provided by the chip manufacturer.

The work described in [15] combines a self designed and implemented power meter that is attached to the computer with the Paraver performance analysis tool. The attached device emits at a frequency that range from 25 to 100 Hz the power consumption of the system. The resulting combination allows the authors to enrich the traces that contain information with power information, giving the analyst the chance to correlate the source code and the parallel runtime calls with power metering reads. Their method of work consists on getting accumulated power measurements at node level by using low sampling rates for previously instrumented regions of code. Their work requires that all the cores have to execute one of the instrumented regions. While the work we propose tackles the same problem, our work is aimed at generating extremely detailed power and performance results by combining instrumentation and sampling information. In our approach, we do only need to instrument the entry and exit points of a region and by smartly combining the sampling and instrumentation information we are able to present the temporal evolution of the power and performance metrics along the instrumented region. Also, the work described here, use a component embedded into the processor, thus removing the need of attaching a device into the system to obtain power measurements.

Other performance tools like Scalasca, Vampir, HPCToolkit and TAU, among others, provide performance information based on the processor hardware counters by using the PAPI interface. These tools provide performance metrics for different sections of the application code. Although these tools would benefit from using the RAPL component from PAPI to supply energy consumption for the application code, to our knowledge our mechanism is the first one to use the integrated mechanism to report the processor power usage.

3. TRACKING POWER CONSUMPTION

The Intel® SandyBridge processors introduce a new infrastructure within the chip named Running Average Power Limit (RAPL). The RAPL infrastructure is responsible for running within user-given (or system-given) power constrains. The functional part that is in charge for such feature is called Package Control Unit (PCU) and is a combination of dedicated hardware state machines and an integrated controller. The PCU is connected to power management agents which are responsible for collecting information such as power consumption and temperature, and also, for controlling transitions between processor performance states (P-states) and processor operating states (C-states). To predict the socket's active power consumption, the PCU collects events from the cores, the I/O and the integrated GPU and weights them with energy factors that depend on the processor itself. The resulting power consumption is scaled accordingly with operation conditions such as voltage and frequency of execution.

The RAPL infrastructure is accessible through the processor model specific registers[‡] (MSR). Although the usage of MSR is restricted to be accessed only from the kernel mode, a regular user can read them in the Linux operating system if the `/dev/cpu/*/msr` files have the appropriate read flags. The RAPL interface exposes several domains of power distributed for every processor socket and each can be monitored and limited in an independent way. The RAPL domains consist of package domain (*i.e.* the whole socket), the basic power plane (*i.e.* the cores of a single processor socket), memory domain (*i.e.* the directly-attached DRAM), and, optionally, an additional power plane (*i.e.* typically assigned to the integrated GPU).

As noted earlier, RAPL allows gathering information about energy consumption of every RAPL domain through its power metering interface. According to the Intel manual, the energy consumption information is updated at a frequency rate of 1 KHz, and by default, the processor socket reports the energy measurements in multiples of 15.2 nJoules. Also, it is important to note that, opposite to the performance counters which count events for a particular process, the RAPL component sums up all the energy used by the entire socket. Although the frequency rate seems high, we will discuss in the following section how to convert the energy measurements into a almost continuous function so as to provide finer information for a region of code.

4. FOLDING

There are two mechanisms a performance tool can use to gather metrics from an application: sampling and instrumentation. Whereas the former is meant to gather metrics by issuing probes periodical and independently from the application source code, the latter refers to inject probes at specific application points. To give more insights of the application behavior, the performance tool may increase the sampling frequency or instrument other application points. But, no matter what mechanism a tool uses, the more detailed results requested, the more overhead the application suffers during execution.

To alleviate the problem, the folding mechanism [26] combines performance information (including callstack references and performance counters) gathered from both instrumentation and sampling points to describe the performance evolution in a synthetic region. The folding mechanism takes advantage of the repetitive patterns found in many applications to provide very detailed progression of the application performance even using coarse grain sampling. It has been demonstrated in [27] that, the longer the application run, the lesser difference between the results of fine grain-sampling and those obtained by the combination of coarse grain sampling and folding (up to a difference of 5%). Thus the usage of coarse grain sampling and the folding mechanism brings the analyst detailed application performance data without incurring a negligible overhead during the application run.

[‡]For further reference, see *Intel® 64 and IA-32 Architectures Software Developers' Manual: Volume 3 (3A, 3B & 3C): System Programming Guide.*

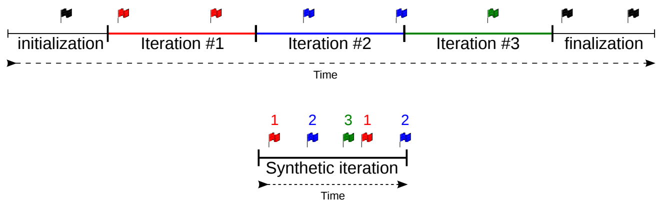
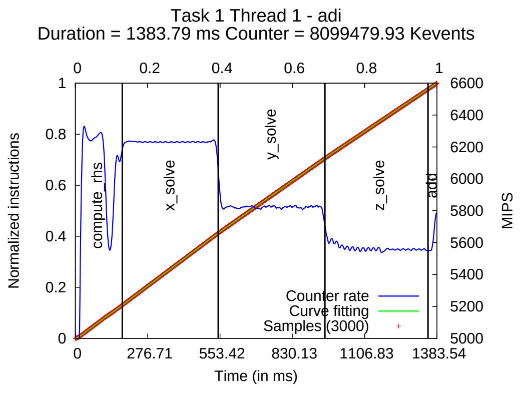
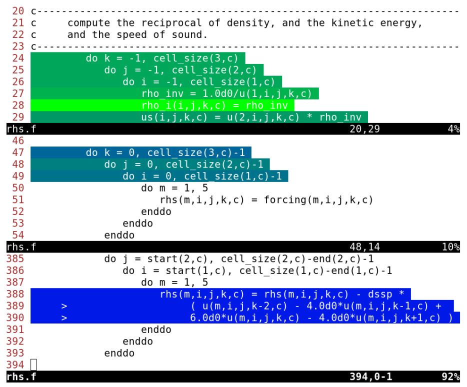


Figure 1. The top figure shows the actual execution of the application with the data gathered at sampling (through flags) and instrumentation points. The bottom figure represents the folded results and where the flags are colored and indexed according to the original iteration they come from.



(a) Evolution of the instruction counter and the MIPS rate in NAS BT



```
c-----------------------------------------------------------------------
c     compute the reciprocal of density, and the kinetic energy,
c     and the speed of sound.
c-----------------------------------------------------------------------
         do k = -1, cell_size(3,c)
            do j = -1, cell_size(2,c)
               do i = -1, cell_size(1,c)
                  rho_inv = 1.0d0/u(1,i,j,k,c)
                  rho_i(i,j,k,c) = rho_inv
                  us(i,j,k,c) = u(2,i,j,k,c) * rho_inv
rhs.f                                                  20,29          4%

         do k = 0, cell_size(3,c)-1
            do j = 0, cell_size(2,c)-1
               do i = 0, cell_size(1,c)-1
                  do m = 1, 5
                     rhs(m,i,j,k,c) = forcing(m,i,j,k,c)
                  enddo
               enddo
            enddo
rhs.f                                                  48,14         10%
            do j = start(2,c), cell_size(2,c)-end(2,c)-1
               do i = start(1,c), cell_size(1,c)-end(1,c)-1
                  do m = 1, 5
                     rhs(m,i,j,k,c) = rhs(m,i,j,k,c) - dssp *
     >                    ( u(m,i,j,k-2,c) - 4.0d0*u(m,i,j,k-1,c) +
     >                    6.0d0*u(m,i,j,k,c) - 4.0d0*u(m,i,j,k+1,c) )
                  enddo
               enddo
            enddo

rhs.f                                                  394,0-1       92%
```

(b) Folding results of the instruction counter annotated into the source code using GVim

Figure 2. Folding results for the main time-step function (adi) in the NAS BT benchmark.

Folding works by projecting the performance data associated to each sample into synthetic representative instances of a computation region that has been delimited by instrumentation. The folding uses the instrumented and the sampled information for two different objectives. While the instrumented data delimits the duration and the aggregate performance counter, the sampled information tracks the progression of the performance counter and the call-site references within the region. More precisely, the folding maps every sample found within a computation region into their respective synthetic region by preserving their relative time. As a result of doing this for every sample, the synthetic regions get populated with performance metrics that depict the internal evolution of the computing region as depicted in Figure 1.

In Figure 2(a) we illustrate the results of the folding mechanism applied to the executed instructions counter when applied to the NAS bt.B [28] time-step function (adi). The set of red crosses are shown on the left Y-axis and describes the cumulative of the instructions executed since the beginning of adi. That is, a cross at position (X, Y) represents a sample that occurred at time X within adi and that has executed Y instructions since the beginning of the routine (at $(0, 0)$). Once the samples are mapped into the synthetic region we apply a contouring algorithm which will report a continuous evolution of the aggregated metric within the region. By using this continuous evolution we calculate the instantaneous counter rate by calculating the derivative of the contoured results. We show the instantaneous rate by using a blue line that traverses the region using the right Y-axis. We do notice that, although the progression of the samples does not exhibit a clear variance across time, the derivative is capable of showing differences. In fact, the instantaneous metric (Millions of Instructions Per Second or MIPS, in this example) shows that x_solve, y_solve and z_solve runs uniformly at 6300, 5800 and 5600 MIPS, respectively. And also shows that compute_rhs does not have uniform behavior, and the metric ranges from 5600 to 6300 MIPS.

The folding is also capable of attributing the performance to the source code to allow the user to correlate the performance and the source code when looking for hot and cold spots. In Figure 2(b)

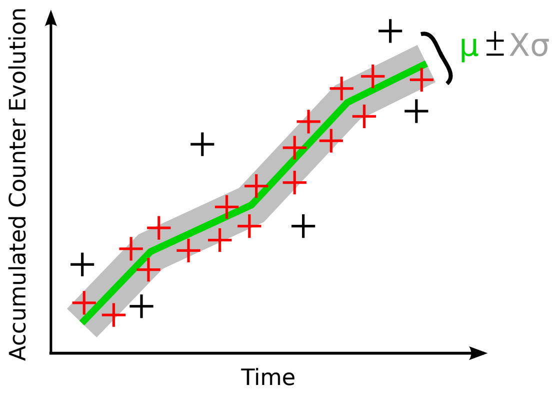


Figure 3. Filtering applied in the folding mechanism. The green line refers to the curve fitting result when using with the samples (μ) and the grey color shows the area (ranging between $\mu \pm X\sigma$). Those samples that have to be taken into account are shown in red whereas the samples that can be ignored are colored in black.

we show a capture of the GVim editor showing three different parts of the `rhs.f` file from the bt.B benchmark and the performance achieved by each by using a color gradient that ranges from green (meaning low) to blue (meaning high). In this Figure, the first section (between lines 24 and 29) refers to the lowest performing code of `rhs.f` which is at the very beginning of 2(b). The latter section of the code (which involves the loop ranging from line 385 to line 393) is related to the rightmost part of the `compute_rhs` function depicted in 2(b) and is executed by the processor at approximately 6000 MIPS.

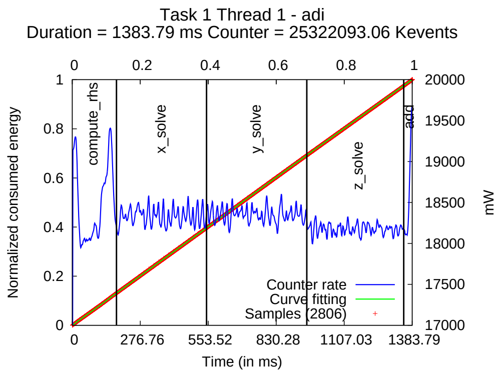
4.1. *Folding improvements to accommodate power measurements*

As we have discussed before, the folding mechanism mainly consists on building a curve fitting for all the samples gathered during the execution. To generate such curve fitting, the folding mechanism has a pre-filter mechanism that removes the instances that are too different in terms of duration in respect of the mean duration. This pre-filter step allows ignoring those instances that have suffered from any sort of perturbation (for instance, the application performs a checkpoint, there network is congested or the node is overloaded). By applying the pre-filter the resulting folding signal becomes less noisier and allows applying a more much strict curve fitting parameters.

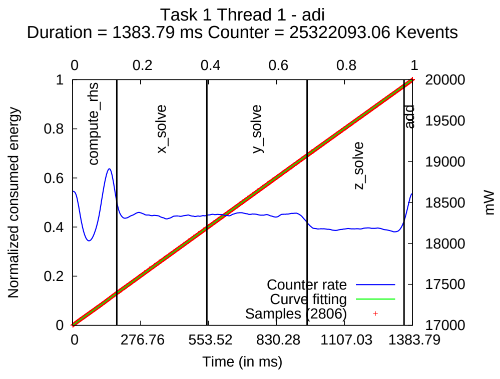
By using such pre-filter step, the folding mechanism has proven useful to reflect the evolution of the performance hardware counters. However, the performance counters and the energy counters differ in their update frequency and also in their granularity. While the performance counters are updated at every processor cycle and report the events occurred for a particular thread of the application, the energy counters are updated at every millisecond by the PCU in factors of 15.2 nJoules and report the whole socket energy consumption. As a result, the curve fitting results follows the energy counters and shows staggered results.

To address this issue, we have improved the folding mechanism in two different ways. First, we have added an additional filter step based on the distance to the curve fitting result. To apply this filter, the folding interpolates all the samples and constructs a curve fitting, as it occurred normally. The folding does not stop at this point, but calculates the distance from the samples to the curve fitting and keeps only those samples that lie within $[\mu - X * \sigma, \mu + X * \sigma]$, as shown in Figure 3, being μ the interpolation point and σ the standard deviation of the distance to the curve fitting. The value of X defaults to 2.0, which means that the range includes about 95% of the samples, but it can be modified by the user.

The second modification relies on adapting the fitting parameter of the interpolation algorithm to the counter being used. The fitting parameter determines how strict is the resulting interpolation. Thus the fitting parameter acts as a low-pass filter in the sense that the smaller the parameter, the more fitted and noisier the results. When applying the curve fitting to the folded results of the processor performance counters we set the fitting parameter to 10^{-6}. However, if we apply the same



(a) Fitting parameter = 10^{-6}



(b) Fitting parameter = 10^{-4}

Figure 4. Folding results using different fitting parameters for the consumed energy of the main time-step function (`adi`) in the NAS BT benchmark.

value to the fitting parameter when doing the curve fitting of the energy counters the results are much more noisier, as depicted in Figure 4(a). The plot in Figure 4(b) shows a smoother result by using the same data as in the previous example, but using a more relaxed fitting parameter (10^{-4}). We consider that, although producing smoother results, the relaxed parameter gives a good understanding of the power consumption evolution across time and thus will be the default value for the power and energy counters.

5. EXPERIMENTS

We have designed a twofold experiment to demonstrate the usefulness of adopting this new mechanism to capture the power consumption from the processor itself. The first experiment is aimed at studying the performance and power consumption in a single socket by using widely-known serial benchmarks. To address this experiment we have used a set of serial benchmarks that use iterative methods from different benchmarks suites, including SPEC CPU 2006 [29], NAS Parallel Benchmarks, and also the Stream [30] and Lulesh [31] benchmarks. We also evaluate the impact of changing the processor frequency in terms of performance and power. The second experiment focuses on analyzing the performance and power consumption in a parallel environment by using three MPI applications.

We have executed all the experiments in *Altamira*. This supercomputer consists of 160 nodes, each containing two Intel® Xeon® CPU E5-2670 (SandyBridge-EP) 8-core processors running at a nominal frequency of 2.60 GHz with maximum thermal design power (TDP) of 115 Watts. The system runs Linux kernel 2.6.32 and allows changing the processor frequency between 1.2 and 2.6 GHz in 0.2 GHz steps. The Intel® Turbo Boost, which can accelerate the Xeon E5-2670 processor up to 3.3 GHz incurring in additional power consumption, has been disabled from BIOS to perform all the tests at a uniform frequency. We have used the GNU compiler suite version 4.4.6 with `-O3 -g` as compile flags and OpenMPI version 1.6 for the parallel applications. We have used the Extrae [32] instrumentation package to gather performance and power metrics by using its sampling and instrumentation capabilities. In reference to the sampling resolution, we have set the sampling frequency to 50 Hz, which is the same sampling frequency used by default by the gprof profiler [33]. So as to use the RAPL power counters, we compiled Extrae against a PAPI library with the RAPL component enabled. Finally, in every execution we have pinned the processes to a particular core so as to disallow process migration, which would affect the application performance and its consumption. For the serial benchmarks, we have executed them sequentially and we have pinned them to the first core of the socket. For the parallel applications, the pinning depends upon

Benchmark Suite	Name	Time-step code region
SPEC CPU 2006	434.zeusmp	`src/zeusmp.F` 675-709
	435.gromacs	`src/md.c` 413-820
	436.cactusADM	`src/PUGH/Evolve.c` 96-147
	437.leslie3d	`src/tml.f` 330-435
	444.namd	`src/spec_namd.C` 184-225
	465.tonto	`src/mol.F90` 13617-13629
	470.lbm	`src/main.c` 44-58
	481.wrf	`src/module_integrate.F90` 274-288
NPB 3.3	bt.B	`BT/adi.f` 8-20
	ft.B	`FT/appft.f` 62-72
	is.C	`IS/is.c` 446-641
	lu.B	`LU/ssor.f` 102-232
	mg.B	`MG/mg.f` 255-264
Stream	stream	`stream.c` 220-262
Lulesh	lulesh	`full/lulesh.cc` 2893-2904

Table I. Benchmarks used for the experiments and the location of the begin and end points for the iterative part of the application.

the execution configuration applied in therms of processes per socket but we granted that each core executes a single process.

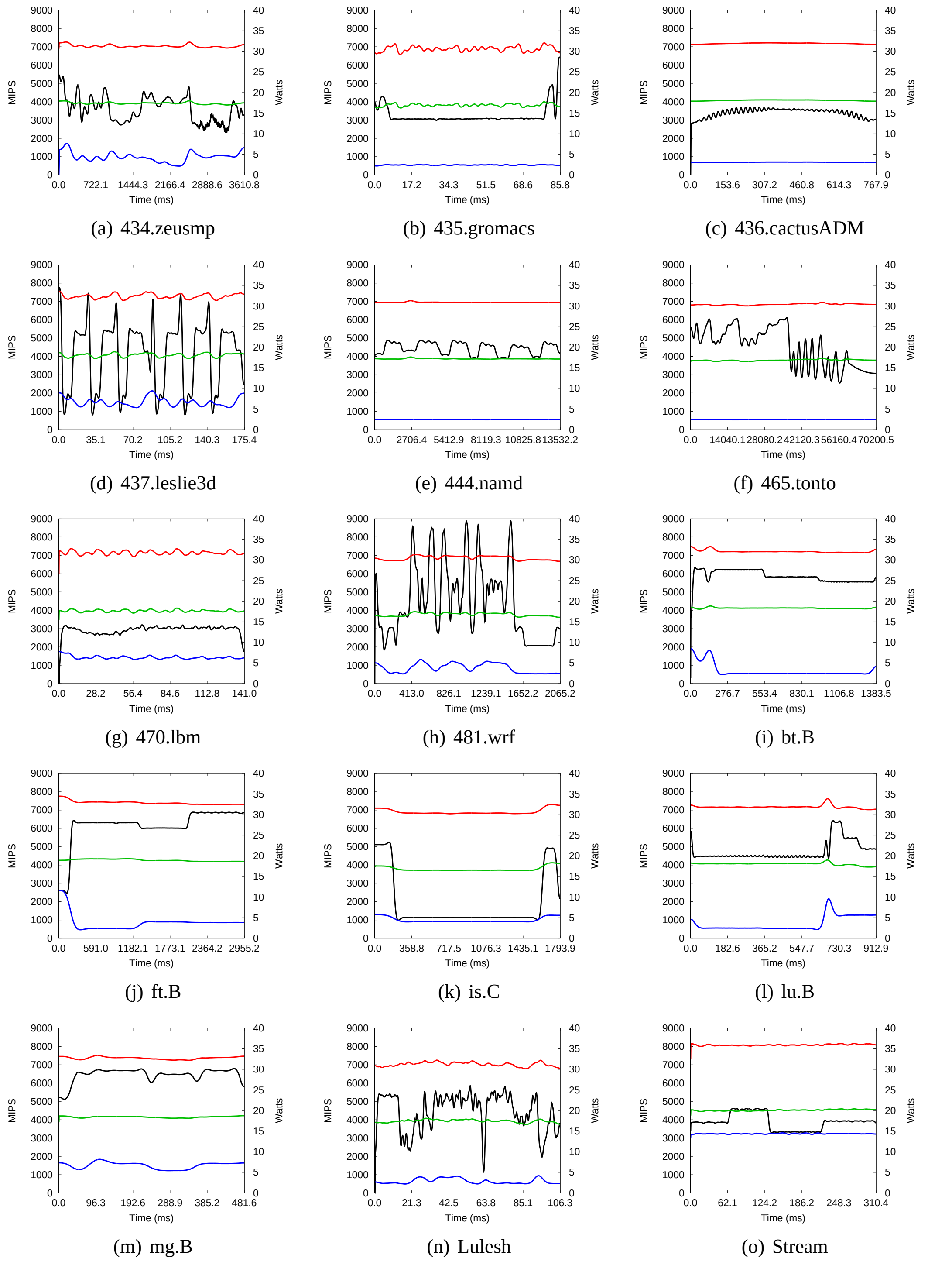
5.1. *Analysis of serial benchmarks*

The selected benchmarks are listed in Table I. The Table illustrates the subset of benchmarks and also shows the location within the source code where a time-step begins and ends. The placement of the begin and end points of the time-stepper region of code determines the instrumentation points.

The plots shown in Figure 5 describe time-step routine of the benchmarks listed in Table I by using performance and power metrics. In each plot, the instantaneous MIPS (Million of Instructions Per Second) is shown in black and it is referred to the left Y-axis. In respect to the power metrics, they are referred to the right Y-axis and are colored in blue, green and red for the DRAM, the cores and the total of the package, respectively. From these plots we observe that despite the performance achieved by the application, the cores consume essentially the same (between 16 and 18 Watts). 437.leslie3d (5(d)), 481.wrf (5(h)) and ft.B (5(j)) show the largest difference in MIPS within the time-stepper region (from 1000 to 7500, 3000 to 9000, and from 2500 to 7000, respectively) with small variation of the core power consumption. Not only this, but we note at the end of lu.B (5(l)) and ft.B (5(j)), that the more MIPS achieved the lesser power drained. The reader may also observe that the power consumption of the DRAM is mostly uncorrelated with the performance in all executions but in 437.leslie3d (5(d)), 481.wrf(5(h)) and is.C (5(k)) where the high peak performance results in higher consumption of the DRAM. Regarding the total power consumption, we note that the DRAM and core do not sum up for the total consumption of the package. Our guess is that, although the processors of the system we have used do not have any integrated GPU on their power plane, there would be some power drain, possibly in the I/O, that sums up for the total. Finally, and although the package wattage follows the core wattage shape, the results shown in lu.B (5(l)), ft.B (5(j)), 434.zeusmp (5(f)) and 437.leslie3d (5(d)) reflect that the power consumed by the DRAM is underweighted in terms of energy factors by the PCU when summing up for the total consumption of the package.

5.2. *Application of the DVFS techniques to the serial benchmarks*

The power dissipated by a processor using the current CMOS technology is divided in two parts, the static and the dynamic power, being the latter the main source of power consumption. Dynamic



(a) 434.zeusmp (b) 435.gromacs (c) 436.cactusADM
(d) 437.leslie3d (e) 444.namd (f) 465.tonto
(g) 470.lbm (h) 481.wrf (i) bt.B
(j) ft.B (k) is.C (l) lu.B
(m) mg.B (n) Lulesh (o) Stream

Figure 5. Comparison of the performance and power consumption of the main iteration of several benchmarks when the processor runs at 2.6 GHz. The MIPS performance (in black) is referred to the left Y-axis, whereas the power consumption (red for package, green for chip and blue for DRAM) are referred to the right Y-axis.

power is the power needed to commute the transistors of the circuit and it is equal to

$$P_{dynamic} = \alpha C V^2 f \tag{1}$$

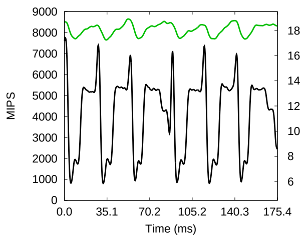
where α is the proportion of activity in a clock cycle (which is related to the application executed), C refers to the capacitance of the chip, V is the voltage and f is the frequency. Current operating systems increase or reduce the frequency of the processor (f) depending on the load of the system using a technique called Dynamic Voltage and Frequency Scaling (DVFS). There are two possibilities to reduce the energy consumption, by reducing the processor voltage or by reducing the processor frequency. While the former cannot be usually tackled by the user in a production environment, the latter can still be applied. In order to reduce the energy consumption, the operating system lowers the processor frequency when the system becomes idle and reestablishes the processor frequency when the system gets loaded. Others, like [34] and [35] have used such technique to reduce the energy consumption on parallel applications that present load unbalance by decreasing the processor frequency on the processors with less work. Here we will detail the effects of applying the DVFS to the previous serial benchmarks.

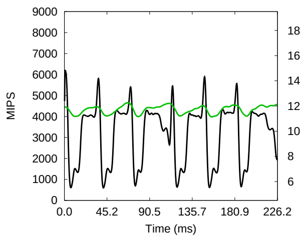
In Figure 6 we show the impact of different execution frequencies on the power and performance metrics in a subset of benchmarks. In the plots shown, we first notice that neither the power nor the performance shape of the benchmarks change when modifying the core frequency, but the amplitude of the signal. For instance, the highest and lowest peak in the is.C benchmark which are depicted in Figures 6(m), 6(n) and 6(o) range from 1100-5000, 900-4000 and 600-2400 MIPS, and, 16.4-18.3, 10.6-11.9 and 4.9-5.8 Watts, respectively in terms of performance and power usage. These plots illustrate that the highest the core frequency, the highest amplitude exists in both performance and power metrics, and hence the more room for energy and performance improvements can be reached in highest frequencies, as it would be expected by Equation 1. And as we noted earlier, although the power consumption may be directly related to the processor performance, as it occurs in 437.leslie3d, 481.wrf and is.C, sometimes the power consumption runs independent from the performance achieved, as it can be seen in ft.B and lu.B.

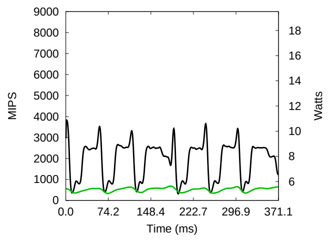
The plots also show that the reduction factor observed in the power usage is higher than the reduction factor of the clock rate. For instance, while lu.B at 2.6 GHz consumes about 18 Watts, when decreasing the clock rate to 1.2 GHz we would expect according to 1 a consumption about 9 Watts. However, we observe that the power consumption does not reach 6 Watts. This means that at constant capacitance (C) and activity rate (α), the processor also lowers its voltage (V) when lowering its frequency (f).

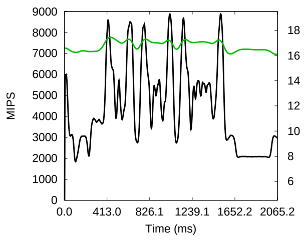
To summarize all the results of the executions, we tabulate them in Table V and Table VI at the end of the document. Table V shows the average duration, the core consumption and the whole-socket consumption of the main iteration of benchmark when run at one of the selected processor frequencies (2.6, 2.0, 1.6 and 1.2 GHz). In this Table, we observe that the energy needed by the cores to execute the time-stepper function decreases as the frequency decreases but at lesser scale. Although reducing the processor frequency from 2.6 to 1.2 GHz makes the application run more than two times slower, the energy drained by the cores does not reduce accordingly. We can also note that the total energy consumed by the whole package increases as the frequency of the processor decreases. This may occur because reducing the operative frequency increases the time needed to finish the task and thus the energy dissipated. These results agrees with the results of a group of experiments done by Le Sueur and Heise in [36].

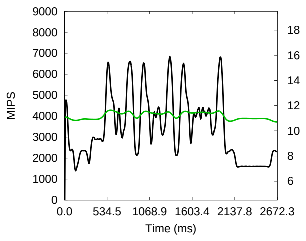
In Table VI we provide the average number of instructions, and the average number of L1D, L2 and Last Level Cache (LLC) cache misses of the main iteration of the benchmark. Also in this Table, we show additional performance and power metrics derived from the aforementioned data. We present the average MIPS and the average MIPJ per core ($MIPJ_C$) and per package ($MIPJ_P$) (analogously to MIPS, MIPJ stands for Millions of Instructions per Joule) achieved by each benchmark. The results in the latter Table show that if we consider $MIPJ_C$ as the power efficiency metric, the lowest frequency provides the best results in terms of instructions per Joule achieved. However, if we consider $MIPJ_P$ as the power efficiency metric, then the most fruitful frequency ranges between 2.6 and 2.0 GHz. In this case, using the highest frequencies allows the application to finish, and thus stop consuming energy, earlier. Four of the slowest benchmarks (434.zeusmp, 470.lbm, is.C and Stream) achieve the best $MIPJ_P$ (114.51, 92.45, 61.26 and 108.69) running at 2.0 GHz, while the rest of the benchmarks show better $MIPJ_P$ when running at 2.6 GHz.

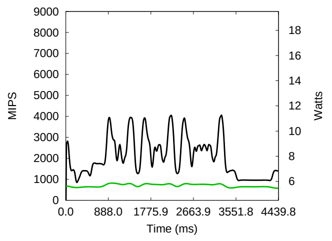

(a) 437.leslie3d at 2.6 GHz

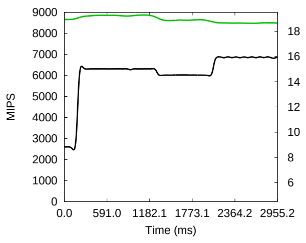

(b) 437.leslie3d at 2.0 GHz

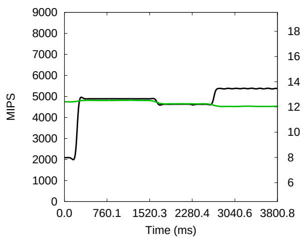

(c) 437.leslie3d at 1.2 GHz

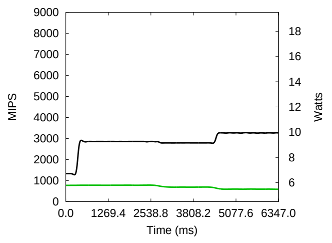

(d) 481.wrf at 2.6 GHz

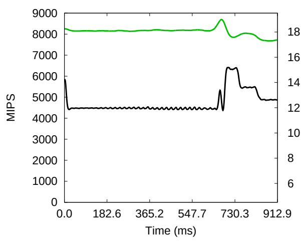

(e) 481.wrf at 2.0 GHz

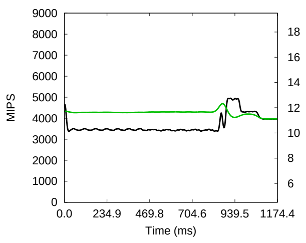

(f) 481.wrf at 1.2 GHz

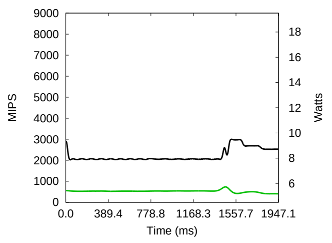

(g) ft.B at 2.6 GHz

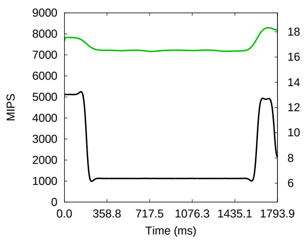

(h) ft.B at 2.0 GHz

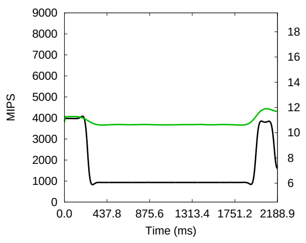

(i) ft.B at 1.2 GHz


(j) lu.B at 2.6 GHz


(k) lu.B at 2.0 GHz


(l) lu.B at 1.2 GHz


(m) is.C at 2.6 GHz


(n) is.C at 2.0 GHz

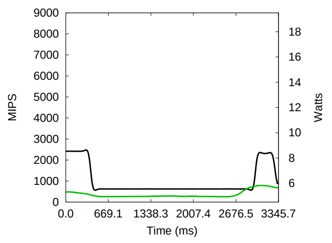

(o) is.C at 1.2 GHz

Figure 6. Performance and power progression of five benchmarks when run at three different core frequencies (2.6, 2.0 and 1.2 GHz). The black line is referred to the instantaneous MIPS on the left Y-axis and the green line is referred to the instantaneous power on the right Y-axis.

These four benchmarks show high L1D, L2 and LLC miss rates, which would indicate that the applications are memory bound. According to these results, a naive approach to select the most effective frequency in terms of $MIPJ_P$ would depend on the ratio between the executed instructions and the LLC misses.

5.3. *Analysis of parallel applications*

We have used three MPI applications to analyze the performance and power consumption in a parallel environment. The first application is HydroC, which is a a proxy benchmark of the RAMSES [37] application. This application solves a large scale structure and galaxy formation problem using a rectangular 2D space domain split in blocks. The second application is Mr. Genesis [38], which employs a finite volume approach in order to evolve the Relativistic Euler equations combined with a Constrained Transport scheme to account for the divergence free evolution of the dynamically included magnetic field. The last application is SIESTA [39], which implements a self consistent density functional method using standard norm-conserving pseudopotentials and a flexible, numerical linear combination of atomic orbitals basis set.

We have executed Mr. Genesis and HydroC using 8 MPI processes to evaluate their performance and power consumption when using different socket occupancy rates. For these experiments we have used combinations of MPI processes per socket (MPIpps) that sum up 8 MPI processes (*i.e.* 1, 2, 4 and 8 MPIpps using 8, 4, 2 and 1 sockets, respectively). We also evaluate these executions changing the processor frequencies at four different speeds (1.2, 1.6, 2.0 and 2.6 GHz). Regarding SIESTA, we have studied its scalability by executing it from 16 to 256 MPI processes, thus using multiple nodes using all the 16 cores available on the node.

We show different performance and energy metrics for these executions. More precisely, to evaluate the performance, we show the time required to execute the application. To evaluate the energy and power usage we provide results of the folding process detailing the instantaneous power consumption, the overall energy consumed, and also the Energy Delay Product (EDP). The EDP is a metric that represents a compromise between the application performance, typically measured by the application execution time, and its dissipated energy. From the folding results we also derive the energy footprint of the application. Such footprint shows the percentage of time in a particular power consumption rate and the duration of the most power consuming regions of code.

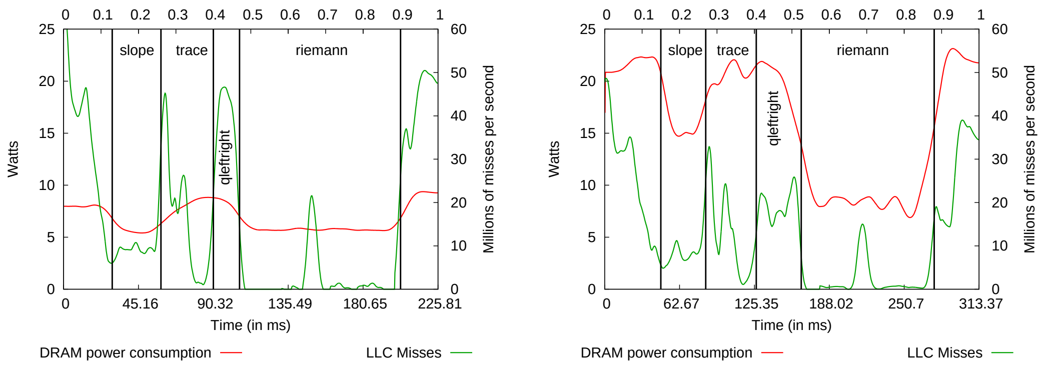
5.3.1. HydroC We show in Table II the timing, the energy consumption and the EDP results for the HydroC application. Note that time of the full execution of the application not only increases when decreasing the processor frequency, but also when increasing the occupancy of the processor because of the sharing of the resources. Also, the more processes executing in a socket, the lesser energy consumed at a given frequency. From the results shown in the table we can extract some guidance depending on the metric to minimize. To reduce the overall execution time of HydroC, we should use a single processor per socket at maximum frequency. However, to reduce the overall energy drained by HydroC,we should use all the processors available in a socket at 1.6 GHz. The best trade-off between the performance and energy consumption (EDP), is achieved by using half of the processors of the socket at full speed.

Figure 7 shows the temporal evolution of the power consumption of the DRAM at socket level and the LLC cache misses per core within the time-stepper routine when using one MPIpps and eight processes per socket running at 2.6 GHz. We observe a tight correlation between the rate of LLC misses and the power consumed by the DRAM. This occurs because the LLC misses involve accessing into the memory, thus increasing its energy consumption. This is expected, as high miss rates involve big amounts of data movement, which is one of the important components of system power. This effect is most noticeable when using all the eight cores because the power consumption counter reflects the accumulated energy consumption by the whole socket and the signal presents a wider amplitude.

By using the source code referencing capabilities, we have delimited in the plots the routines that were executed across time and we have labelled the regions by adding the routine names. Comparing the two subfigures (7(a) and 7(b)), we first note the increase on the DRAM power consumption which is below 10 Watts in routines `trace` and `qleftright` when using one MPI process per socket, and increases to more than 20 Watts when using eight processes per socket.

	2.6 GHz			2.0 GHz			1.6 GHz			1.2 GHz		
	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP
1MPIpps	**49**	144	7101	61	147	8984	75	158	11855	103	186	19184
2MPIpps	50	98	4923	62	89	5557	76	96	7359	100	104	10484
4MPIpps	52	64	**3369**	65	59	3868	78	59	4676	101	65	6622
8MPIpps	72	65	4720	83	53	4457	96	**49**	4795	122	50	6137

Table II. Duration (in seconds), energy consumption (in KJoules) and the Energy Delay Product for the HydroC application when using combinations of MPI processes per socket and processor frequencies. Optimal values for each metric is highlighted in bold font.



(a) 1 MPIpps (b) 8 MPIpps

Figure 7. Temporal evolution of the time-stepper routine of HydroC by using a different numbers of MPIpps.

These results mean that even multiplying by eight the number of cores accessing to memory, the power consumption by the DRAM gets only multiplied by a factor between 2 and 3. The `riemann` routine shows better power scalability when moving from one process per socket to eight processes per socket. In this routine, while the DRAM consumes about 5 Watts in 1 MPIpps, it turns only into 8 Watts when running in 8 MPIpps.

In terms of performance, when the socket is fully utilized the execution takes longer to compute (313.37 ms) in comparison to when the processor socket is under utilized (225.81 ms). We can observe that the rate of LLC misses decrease when using all the eight cores of the socket, although the shape for the LLC miss rate is rather similar in both cases. We observe that the proportion of the `qleftright` increases in the case when using eight cores per socket and its relative duration increases from 8% to 12%, which represents an increase of from 18 to 37.5 milliseconds. The `riemann` routine also increases its duration, from 95 to 113 milliseconds, although its weight decreases, from 42% to 36%. According to the performance data gathered, the total number of LLC misses per core increases with respect to the execution with a single MPIpps socket by 2%, 6.2% and 12.5% when using two, four and eight processes per socket, respectively. This effect is reasonable because the LLC is a shared resource among cores within the socket and cannot sustain its performance per core with the increased number of LLC misses experienced by the application.

We also illustrate the socket energy consumption by using the aforementioned combination of processes per socket and executing frequencies in Figure 8. We observe in the Figure that the shape for the different plots are likely the same but in the amplitude, which varies according to the processor frequency. When the system is fully occupied and its clock rate operates at 2.6 GHz, the power dissipated because the execution of HydroC presents two modes, 80, 70 Watts, which means that the processor consumes about 70% of its maximum TDP (115 Watts). In the same Figure we observe that the power consumption does not increase linearly with respect of the number of executing cores. This is explained because the remaining cores in the executions with 1, 2 and 4 MPIpps are still consuming energy because they are not completely halted, but idle.

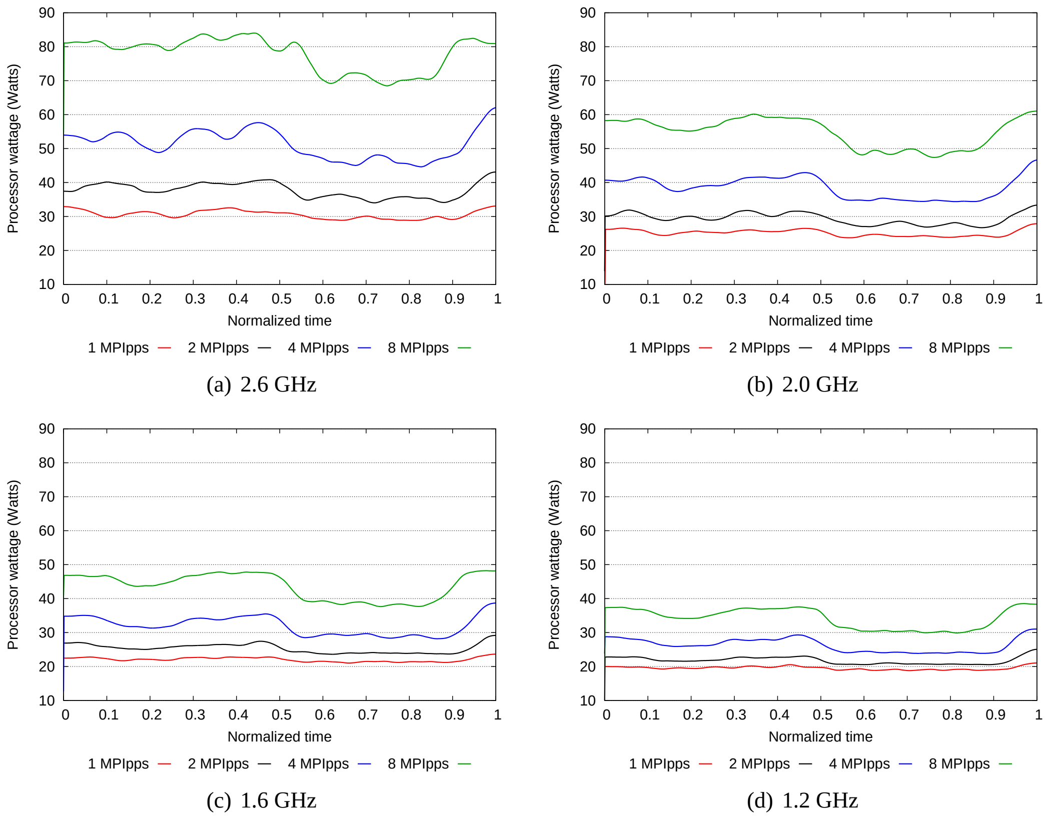


Figure 8. Progress of the main time-stepper routine power consumption of HydroC by using different combinations of MPIpps and processor frequencies on a single socket.

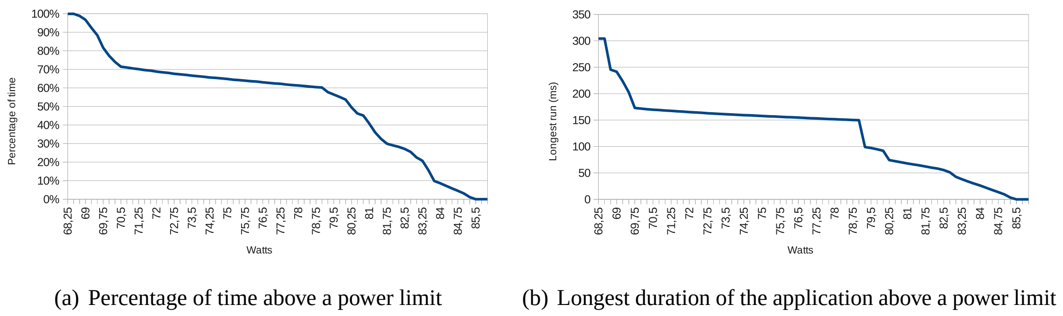


Figure 9. Energy footprint for the execution of HydroC when using 8 MPIpps running at 2.6 GHz.

Finally, we present in Figure 9 the energy footprint of the application for the whole socket. The Figure 9(a) shows the amount of time that the application has been consuming a certain amount of power. For instance, we observe that the application used up to 80 Watts about half of the execution and that only 10% of the whole execution needed more than 84 Watts. We illustrate on the Figure 9(b) the longest execution time with a given sustained consumption. From this Figure we observe that those regions of the application consuming more than 84 Watts take less than 25 milliseconds to execute.

5.3.2. Mr. Genesis We show in Table II the timing and energy consumption results for Mr. Genesis when it is executed with 8 tasks and using a combination of 1, 2, 4 and 8 MPI processes per

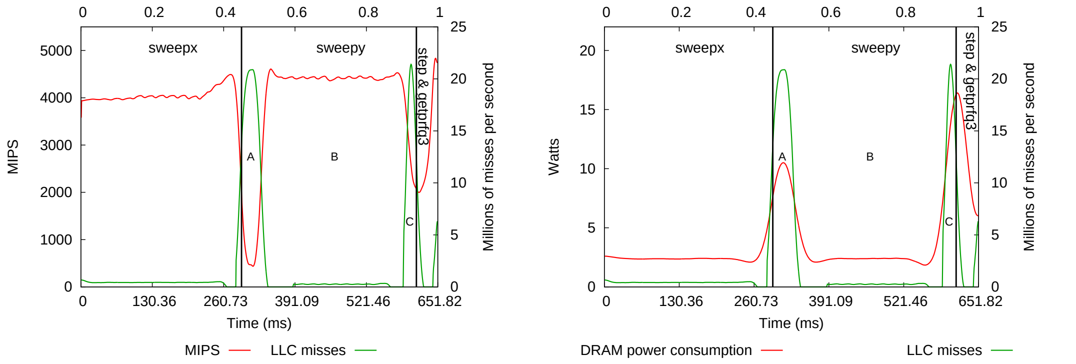
	2.6 GHz			2.0 GHz			1.6 GHz			1.2 GHz		
	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP
1MPIpps	**190**	608	115562	249	636	158484	309	691	213634	412	813	335049
2MPIpps	192	374	71901	249	373	92969	310	395	122603	418	456	190919
4MPIpps	193	256	49548	249	261	65016	314	246	77368	416	330	137542
8MPIpps	197	204	**40299**	255	186	47552	317	**179**	56899	422	183	77571

Table III. Duration (in seconds), energy consumption (in KJoules) and Energy Delay Product of the Mr. Genesis application when using combinations of MPI processes per socket and processor frequencies. Optimal values for each metric is highlighted in bold font.

socket and at four different processor frequencies (1.2, 1.6, 2.0 and 2.6 GHz). In contrast with the previous observations on HydroC, the performance of Mr. Genesis when using all the cores shows an overhead less than 4% in the worst case. Again, to obtain the best performance and reduce the time needed to execute the application, we should execute the application placing every MPI process into a different socket. In therms of energy consumption, we observe, as in Hydro, that the best combination involves using all the cores of the socket and running at 1.6 GHz. Finally, to minimize the EDP metric we would use all the resources available in the socket at full speed.

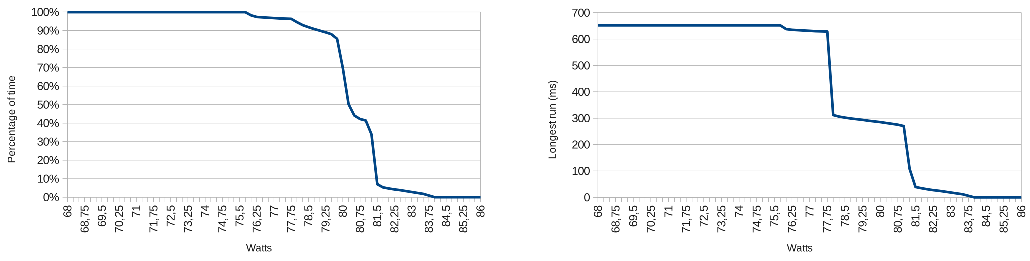
In Figure 10 we show the performance evolution of two metrics (MIPS and LLC miss rate) for the main time-step section of code in the best case in energy terms (*i.e.* 8 MPIpps at 2.6 GHz). Regarding the performance, the application has two important routines in terms of timing, `sweepx` and `sweepy`. On the one hand, the routine `sweepx` has a uniform performance in terms of MIPS (4000) and LLC miss rate (less than 1 million per second), but at the end that shows a slight increase of performance. On the other hand, `sweepy` presents different phases in terms of performance which are related to three parts of the code enumerated with A, B, and C that achieve 500, 4500 and 1800 MIPS, respectively. Focusing on the sections that achieve lowest performance, we observe that phase A refers to the loop in lines 419-435 of file sweep `sweep.f`. This loop performs performs multiple operations similar to matrix transpositions in the loop body except that the data is read from two 3D matrices and stored into multiple 2D matrices. Each cell of these 3D matrices points to five-field structures which translates into a sequential access to each field of the structure thus exploiting spatial locality. This phase reaches up to 20 millions of LLC misses per second per core, and at such moment the DRAM consumption per socket reaches the 12 Watts. Phase C, involves the execution of the loop in lines 556-564 of the file `sweep.f` which partially undoes the work done in Phase A by applying another matrix transposition. In this phase, the LLC miss rate increases again to 20 millions of LLC misses per second, but at this phase the power consumed increases to 17 Watts. Finally, at the very end of the time-step we also found that the performance starts at a low MIPS rate (1500) and increases to 4500, and also the LLC miss rate and the DRAM power consumption decreases. We have identified two loops for such region: a four-nested loop in `step.f` ranging from lines 151 to 163 that does multiple mathematical operations including floating point divisions, and a call to the routine `getprfq3`.

We show in Figure 11 the energy footprint of the application. The reader can see that the 60% of the total execution time of Mr. Genesis, the socket is draining up to 80 Watts. Then the consumption rapidly goes down, and less than 10% of time the processor drained more than 81.5 Watts. It is interesting also to note that the longest duration sustained in 81.5 Watts by the whole socket lasts less than 40 milliseconds. Such values could be useful for the acceleration mechanism that speeds up the processor frequency in two directions. First, the processor could increase the frequency in the regions with lesser power consumption in a safe manner without surpassing the TDP. These results also means that limiting the power consumption to 81.5 Watts would only affect less than 10% of the application. Finally, if such power limit is enabled but the processor supports acceleration mechanisms that allow it to drain more than 81.5 Watts, the required time for such acceleration would be less than 40 milliseconds.



(a) Performance evolution of MIPS and LLC miss rate (b) Evolution of the LLC miss rate and the DRAM power consumption

Figure 10. Performance and energy footprint for Mr. Genesis when running with 8 MPIpps at 2.6 GHz.



(a) Percentage of time above a power limit (b) Longest duration of the application above a power limit

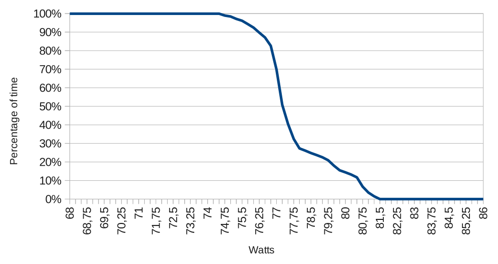
Figure 11. Performance and energy footprint for SIESTA when running with 128 MPI processes at 2.6 GHz.

number of tasks	Energy	Duration	Speedup	Parallel Efficiency	EDP
16	8205	51824	1	-	425220
32	8296	27375	1.89	0.96	227123
64	9885	16252	3.19	0.80	160660
128	13373	10883	4.76	0.60	145546
256	25610	11001	4.71	0.29	281766

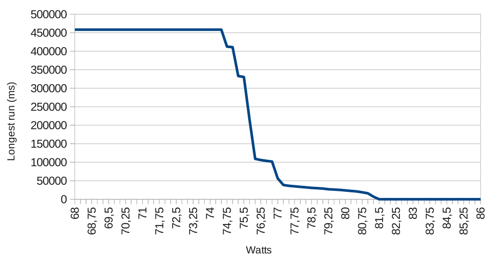
Table IV. Scalability of SIESTA. Energy is shown in KJoules, Duration is shown in seconds and EDP is shown in $MJoules * seconds$

5.3.3. SIESTA We have studied the scalability of SIESTA in terms of performance and energy efficiency, through the time, the energy consumed and the EDP metrics with executions ranging from 16 to 256 processes at 2.6 GHz and using the two sockets of each node. Table IV shows the behavior of the application when using different number of MPI processes. We observe that the application does not even scale linearly, but the application takes longer to execute when using 256 processes showing a speedup of 4.71 compared to the execution of 16 tasks, resulting in a parallel efficiency about 0.29. In terms of performance, the execution that used 128 processes shows the best performance, although its parallel efficiency could be considered low (0.60). For these experiment, we conclude that lowering the number of processes results in better resource utilization and also in less energy consumption. However, the execution that uses 128 processes results in the best execution in terms of EDP.

Regarding to the energy footprint, if we consider the best case of SIESTA in terms of EDP, we obtain the plots shown in Figure 12. As it occurred in the previous studies within this section, the



(a) Percentage of time above a power limit



(b) Longest duration of the application above a power limit

Figure 12. Performance and energy footprint for Mr. Genesis when running with 8 MPIpps at 2.6 GHz.

processor does not typically consume more than 80 Watts (about 70% of the TDP) when running the application. For this particular case, we also observe that those regions of code that drain more than 80 Watts do not last more than 25 seconds. This shows, again, that the processor typically drains an amount of power, and only for some small periods of time it requires consuming more power. More precisely, if we consider the same 81.5 Watts limit as in the previous example, we observe that SIESTA would require surpassing this limit for 7 seconds approximately.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have shown the usefulness of using the novel Intel RAPL infrastructure to obtain power and energy metrics combined with performance counters and the usage of the folding mechanism. By combining them, we have been able to study, with high level of details, the evolution of both the performance and the power in a wide variety of benchmarks and also in parallel applications executed in many different scenarios of a production scenario. We believe that having tools that easily correlate performance and power measurements, and source code are useful to understand the behavior of any application and also allows the continuous development of supercomputing systems and improve their usage. In fact, we have seen with a high level of details that both performance and energy consumption are related and also influenced by similar factors (like the application executed, the processor frequency, the occupancy of the socket). Depending on the metric to optimize, the suggestions to the analyst can be completely different.

In respect to the power and performance analysis, although we have shown many results with small variance in power consumption, there is space for improvement on the power consumption. This is particularly true for the DRAM power consumption because it is heavily related to the memory accesses and a better memory access patterns would result in less power consumption. In this sense, the overall effort to reduce the execution time of an application by improving the source code will also result in a reduction of the energy consumption. As a result, the usage of performance tools to improve the system utilization will result in benefits on the performance efficiency of the application, but also on its energy efficiency.

Not only we are confident that having the Intel RAPL infrastructure will easily accommodate the power measurement into the existing performance tools, but we think that the power measurements could also help the RAPL and the Intel® TurboBoost acceleration mechanism to adjust itself to the load and power conditions. In particular, we have seen that most of the executions does not reach the maximum TDP of the processor. The energy footprints we have shown can be used either to enable the acceleration in phases of low-consumption and also limit the acceleration for a certain duration of the execution. We think that applying the RAPL infrastructure to limit the power usage and comparing it to the DVFS techniques would be also interesting to be studied.

ACKNOWLEDGEMENTS

This work is partially funded by Intel under the Intel-BSC Exascale Center Agreement. We would like to thank Charles H. Finan and Philippe Thierry for their insightful comments on the Intel® SandyBridge processor. The authors also thankfully acknowledge the computer resources, technical expertise and assistance provided by the Red Española de Supercomputación, and specially to Luis Cabellos for his valuable comments and management tasks in the Altamira cluster.

REFERENCES

1. June 2012 Highlights of the TOP500 list 2012. http://top500.org/lists/2012/06/highlights - Last accessed July, 2012.
2. Mudge T. Power: A First-Class Architectural Design Constraint. *Computer* Apr 2001; **34**(4):52–58, doi:10.1109/2.917539. URL http://dx.doi.org/10.1109/2.917539.
3. Duranton M, Black-Shaffer D, Yehia S, Bosschere KD. The HIPEAC vision 2011/2012 2012. http://www.hipeac.net/system/files/hipeac-roadmap2011.pdf - Last accessed July, 2012.
4. Dongarra J, Beckman P, Moore T, Aerts P, Aloisio G, Andre JC, Barkai D, Berthou JY, Boku T, Braunschweig B, *et al.*. The International Exascale Software Project roadmap. *International Journal of High Performance Computing Applications* 2011; **25**(1):3–60, doi:10.1177/1094342010391989. URL http://hpc.sagepub.com/content/25/1/3.abstract.
5. Kappiah N, Freeh VW, Lowenthal DK. Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs. *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, IEEE Computer Society: Washington, DC, USA, 2005; 33–, doi:10.1109/SC.2005.39. URL http://dx.doi.org/10.1109/SC.2005.39.
6. Sharma S, Hsu CH, Feng Wc. Making a case for a green500 list. *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, IEEE Computer Society: Washington, DC, USA, 2006; 299–299. URL http://dl.acm.org/citation.cfm?id=1898699.1898827.
7. Wolf F, Wylie BJN, Ábrahám E, Becker D, Frings W, Fürlinger K, Geimer M, Hermanns MA, Mohr B, Moore S, *et al.*. Usage of the SCALASCA for scalable performance analysis of large-scale parallel applications. *Tools for High Performance Computing*, Springer Berlin Heidelberg, 2008; 157–167.
8. Nagel WE, Arnold A, Weber M, Hoppe HC, Solchenbach K. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer* 1996; **12**(1):69–80.
9. Tallent N, Mellor-Crummey J, Adhianto L, Fagan M, Krentel M. HPCToolkit: performance tools for scientific computing. *Journal of Physics: Conference Series* 2008; **125**(1):012 088.
10. Shende SS, Malony AD. The TAU parallel performance system. *Int. J. High Perform. Comput. Appl.* 2006; **20**(2):287–311, doi:http://dx.doi.org/10.1177/1094342006064482.
11. Pillet V, Labarta J, Cortés T, Girona S. Paraver: A tool to visualize and analyze parallel code. *Transputer and occam Developments* April 1995; :17–32
http://www.bsc.es/paraver - Last accessed July, 2012.
12. Bedard D, Lim MY, Fowler R, Porterfield A. PowerMon: Fine-grained and Integrated Power Monitoring for Commodity Computer Systems. *Proceedings of the IEEE SoutheastCon 2010*, 2012; 479–484.
13. Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems* 2009; **99**(RapidPosts):658–671, doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.76.
14. Watts up? .net.
https://www.wattsupmeters.com/secure/products.php?pn=0 - Last accessed July, 2012.
15. Alonso P, Badía RM, Labarta J, Barreda M, Dolz MF, Mayo R, Quintana-Ortí ES, Reyes R. Power-Energy Modelling and Analysis of Parallel Scientific Applications. *ICPP'12: Proceedings of the 2012 International Conference on Parallel Processing*, 2012. To appear.
16. Wang S, Chen H, Shi W. Span: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems* 2011; **1**(1):23 – 34, doi:10.1016/j.suscom.2010.10.002. URL http://www.sciencedirect.com/science/article/pii/S221053791000003X.
17. Chen H, Li Y, Shi W. Fine-grained power management using process-level profiling. *Sustainable Computing: Informatics and Systems* 2012; **2**(1):33 – 42, doi:10.1016/j.suscom.2012.01.002. URL http://www.sciencedirect.com/science/article/pii/S2210537912000030.
18. Brooks D, Tiwari V, Martonosi M. Wattch: a framework for architectural-level power analysis and optimizations. *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, ACM: New York, NY, USA, 2000; 83–94, doi:10.1145/339647.339657. URL http://doi.acm.org/10.1145/339647.339657.
19. Ye W, Vijaykrishnan N, Kandemir M, Irwin MJ. The design and use of SimplePower: a cycle-accurate energy estimation tool. *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, ACM: New York, NY, USA, 2000; 340–345, doi:10.1145/337292.337436. URL http://doi.acm.org/10.1145/337292.337436.
20. Bertran R, González M, Martorell X, Navarro N, Ayguadé E. Decomposable and responsive power models for multicore processors using performance counters. *ICS*, 2010; 147–158.
21. Singh K, Bhadauria M, McKee SA. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News* Jul 2009; **37**(2):46–55, doi:10.1145/1577129.1577137. URL http://doi.acm.org/10.1145/1577129.1577137.

22. Merkel A, Bellosa F. Balancing power consumption in multiprocessor systems. *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, ACM: New York, NY, USA, 2006; 403–414, doi:10.1145/1217935.1217974. URL http://doi.acm.org/10.1145/1217935.1217974.
23. David H, Gorbatov E, Hanebutte UR, Khanna R, Le C. RAPL: memory power estimation and capping. *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '10, ACM: New York, NY, USA, 2010; 189–194, doi:10.1145/1840845.1840883. URL http://doi.acm.org/10.1145/1840845.1840883.
24. Rotem E, Naveh A, Ananthakrishnan A, Rajwan D, Weissmann E. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 2012; **32**:20–27, doi:http://doi.ieeecomputersociety.org/10.1109/MM.2012.12.
25. Browne S, Dongarra J, Garner N, Ho G, Mucci P. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.* 2000; **14**(3):189–204, doi:http://dx.doi.org/10.1177/109434200001400303.
http://icl.cs.utk.edu/papi - Last accessed July, 2011.
26. Servat H, Llort G, Giménez J, Labarta J. Detailed performance analysis using coarse grain sampling. *Euro-Par Workshops (Workshop on Productivity and Performance, PROPER)*, 2009; 185–198.
27. Servat H, Llort G, Giménez J, Huck K, Labarta J. Unveiling internal evolution of parallel application computation phases. *ICPP'11: International Conference on Parallel Processing*, 2011.
28. Bailey DH, Barszcz E, Barton JT, Browning DS, Carter RL, Fatoohi RA, Frederickson PO, Lasinski TA, Simon HD, Venkatakrishnan V, *et al.*. The NAS parallel benchmarks 1991; URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.3829.
29. Henning JL. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News* Sep 2006; **34**(4):1–17, doi:10.1145/1186736.1186737. URL http://doi.acm.org/10.1145/1186736.1186737.
30. McCalpin JD. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* Dec 1995; :19–25.
31. Hydrodynamics challenge problem. *Technical Report*, Lawerence Livermore National Laboratory 2011.
https://computation.llnl.gov/casc/ShockHydro/LULESH-files/spec.pdf.
32. Extrae instrumentation package.
http://www.bsc.es/paraver - Last accessed July, 2012.
33. Graham SL, Kessler PB, Mckusick MK. Gprof: A call graph execution profiler. *SIGPLAN '82: Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, ACM: New York, NY, USA, 1982; 120–126, doi: \\\url{http://doi.acm.org/10.1145/800230.806987}.
34. Etinski M, Corbalan J, Labarta J, Valero M, Veidenbaum A. Power-aware load balancing of large scale MPI applications. *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, IEEE Computer Society: Washington, DC, USA, 2009; 1–8, doi:10.1109/IPDPS.2009.5160973. URL http://dx.doi.org/10.1109/IPDPS.2009.5160973.
35. Rountree B, Lownenthal DK, de Supinski BR, Schulz M, Freeh VW, Bletsch T. Adagio: making DVS practical for complex HPC applications. *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, ACM: New York, NY, USA, 2009; 460–469, doi:10.1145/1542275.1542340. URL http://doi.acm.org/10.1145/1542275.1542340.
36. Le Sueur E, Heiser G. Dynamic voltage and frequency scaling: the laws of diminishing returns. *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, USENIX Association: Berkeley, CA, USA, 2010; 1–8. URL http://dl.acm.org/citation.cfm?id=1924920.1924921.
37. RAMSES. http://web.me.com/romain.teyssier/Site/RAMSES.html - Last accessed May, 2012.
38. Mimica, P, Giannios, D, Aloy, M A. Deceleration of arbitrarily magnetized grb ejecta: the complete evolution. *A&A* 2009; **494**(3):879–890, doi:10.1051/0004-6361:200810756. URL http://dx.doi.org/10.1051/0004-6361:200810756.
39. Soler JM, Artacho E, Gale JD, García A, Junquera J, Ordejón P, Sánchez-Portal D. The SIESTA method for ab initio order- N materials simulation. *Journal of Physics: Condensed Matter* 2002; **14**(11):2745. URL http://stacks.iop.org/0953-8984/14/i=11/a=302.

Benchmark	2.6 GHz			2.0 GHz			1.6 GHz			1.2 GHz		
	Time[†]	Core	Package	Time	Core	Package	Time	Core	Package	Time	Core	Package
434.zeusmp	3647	63405	113983	4571	52371	113458	5570	46535	121589	7253	37537	135152
435.gromacs	85	1455	2626	111	1221	2714	139	1133	2989	186	932	3417
436.cactusADM	766	13857	24426	977	11482	24670	1177	10221	26008	1586	8542	29810
437.leslie3d	175	3168	5671	226	2646	5782	279	2402	6235	370	1985	7043
444.namd	13528	232393	417330	17600	194785	430706	22043	180532	474418	29414	148442	541220
465.tonto	70151	1182319	2130323	90937	1033436	2252233	114428	964800	2489647	152462	793420	2828812
470.lbm	140	2484	4468	171	2070	4457	205	1838	4669	263	1609	5218
481.wrf	2039	34147	62090	2638	29518	65262	3285	27192	71354	4383	24845	83751
bt.B	1383	25322	44361	1789	21440	45560	2215	19566	49366	2953	16458	56119
ft.B	2954	55938	97097	3798	46666	98251	4777	43094	107667	6338	35884	121350
is.C	1817	30417	55683	2215	23935	53971	2646	21433	57157	3379	17270	62810
lu.B	912	16443	29073	1174	13600	29451	1457	12577	32176	1946	10453	36606
mg.B	481	8866	15742	618	7328	15891	766	6803	17325	1015	5597	19455
Stream	310	6226	11136	382	5224	11056	461	4675	11560	596	3861	12563
Lulesh	106	1853	3315	137	1657	3526	169	1403	3686	225	1350	4371

Table V. Time and energy consumption for the selected benchmarks executing at different frequencies.

Benchmark	Instructions[‡]	L1D	L2	LLC	2.6 GHz			2.0 GHz			1.6 GHz			1.2 GHz		
					$MIPS$	$MIPJ_C$	$MIPJ_P$	$MIPS$	$MIPJ_C$	$MIPJ_P$	$MIPS$	$MIPJ_C$	$MIPJ_P$	$MIPS$	$MIPJ_C$	$MIPJ_P$
434.zeusmp	12988177	2.2%	0.4%	0.2%	3561.09	204.84	113.95	2842.25	252.91	114.51	2333.02	279.30	106.89	1792.89	346.45	96.22
435.gromacs	278947	1.2%	<0.1%	<0.01%	3250.90	191.63	106.19	2507.00	228.52	102.80	2005.34	246.31	93.41	1502.00	299.57	81.77
436.cactusADM	2603516	0.9%	0.4%	0.1%	3398.03	187.87	106.58	2665.62	226.83	105.57	2213.30	254.90	100.18	1643.96	305.25	87.47
437.leslie3d	716501	3.0%	0.8%	<0.01%	4087.98	226.16	126.33	3171.41	270.82	123.95	2567.19	298.44	114.98	1934.65	361.39	101.86
444.namd	60102000	1.1%	<0.1%	<0.01%	4442.78	258.62	144.02	3415.88	306.65	139.59	2728.36	333.14	126.77	2045.81	126.77	111.19
465.tonto	52627913	1.3%	0.4%	<0.01%	4501.21	267.07	148.23	3473.47	305.65	140.25	2761.39	327.51	126.92	2073.73	398.48	111.77
470.lbm	411983	5.2%	0.9%	0.2%	2926.22	165.82	92.19	2401.60	199.05	92.45	2003.26	224.28	88.29	1564.68	256.34	79.05
481.wrf	9132508	1.0%	0.2%	<0.01%	4478.79	267.44	147.08	3462.06	309.48	139.98	2781.50	336.08	128.07	2085.93	368.04	109.18
bt.B	8099479	1.9%	0.2%	<0.01%	5853.10	319.86	182.58	4528.40	377.84	177.81	3656.86	414.14	164.14	2745.20	492.54	144.46
ft.B	18206181	3.6%	0.9%	<0.01%	6161.94	325.47	187.50	4794.32	390.22	185.34	3812.30	422.67	169.18	2875.04	507.80	150.16
is.C	3304812	3.3%	0.5%	0.4%	1818.81	108.46	59.35	1492.38	138.04	61.26	1250.23	154.36	57.88	980.00	191.76	52.73
lu.B	4329609	2.1%	0.4%	0.1%	4742.97	263.30	148.92	3687.77	318.43	147.05	2971.76	344.45	134.64	2226.43	414.62	118.41
mg.B	3106833	1.7%	0.4%	0.1%	6454.72	350.39	197.36	5022.88	424.01	195.55	4052.77	456.82	179.40	3061.46	555.45	159.82
Stream	1201408	5.2%	4.2%	0.9%	3871.06	192.94	107.88	3141.75	230.00	108.69	2601.96	257.10	103.98	2016.33	311.49	95.73
Lulesh	472003	1.4%	0.3%	0.2%	4443.30	254.67	142.34	3426.91	284.07	133.50	2790.35	336.61	128.12	2095.19	349.98	108.10

Table VI. Performance and energy metrics for the selected benchmarks using different processor frequencies.

[†]Time refers to the average duration of the time-stepper function in milliseconds. Core and Chip refers to the energy used in Joules by all the cores and the whole socket, respectively.
[‡]Instructions is shown in millions. L1D, L2 and LLC refer to the ratio of L1, L2 and LLC with respect to the instructions executed. $MIPS$, $MIPJ_C$ and $MIPJ_P$ stand for, Millions of Instructions per Second, Millions of Instructions per Joule (using the core or the package energy counter, respectively).