

Computer Science Department
University of Crete

TPROF: An Energy Profiler for Task-Parallel Programs

Master's Thesis

Ioannis Manousakis

August 2013

Heraklion, Greece

University of Crete
Computer Science Department

TPROF: An Energy Profiler for Task-Parallel Programs

Thesis submitted by
Ioannis Manousakis
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Ioannis Manousakis

Committee approvals: _____
Angelos Bilas, Professor, Thesis Supervisor
Computer Science Department, University of Crete

Dimitrios S. Nikolopoulos, Professor
School of Electronics, Electrical Engineering and Computer Science
Queen's University of Belfast

Evangelos Markatos, Professor
Computer Science Department, University of Crete

Departmental approval: _____
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, August 2013

Abstract

We present TPROF, an energy profiling tool for OpenMP-like task-parallel programs. To compute the energy consumed by each task in a parallel application, TPROF dynamically traces the parallel execution and uses a novel technique to estimate the per-task energy consumption. To achieve this estimation, TPROF apportions the total processor energy among cores and overcomes the limitation of current works which would otherwise make parallel accounting impossible to achieve. We demonstrate the value of TPROF by characterizing a set of task parallel programs, where we find that data locality, memory access patterns and task working sets are responsible for significant variance in energy consumption between seemingly homogeneous tasks. In addition, we identify opportunities for fine-grain energy optimization by applying per-task Dynamic Voltage and Frequency Scaling (DVFS).

Περίληψη

Σε αυτή την εργασία, παρουσιάζουμε το TProf, ένα εργαλείο καταμέτρησης ενέργειας για προγράμματα που έχουν κατασκευαστεί με tasks του στύλ OpenMP. Για να υπολογίσει την ενέργεια που κατανάλωσε κάθε task σε μία παράλληλη εφαρμογή, το TProf καταγράφει όλη την παράλληλη εκτέλεση και χρησιμοποιεί μια νέα τεχνική για την προσέγγιση του, καταμερίζοντας την ενέργεια ανάμεσα στους εκάστοτε πυρήνες εκτέλεσης, μέσα στον επεξεργαστή. Με αυτό τον τρόπο, ξεπερνάμε προβλήματα που θα έκαναν την παράλληλη προσέγγιση ενέργειας για κάθε task πολύ δύσκολη. Για να δείξουμε την αξία του TProf, χαρακτηρίζουμε ένα σέτ παράλληλων εφαρμογών, όπου παρατηρούμε πως το access pattern, το working set και το data locality είναι υπεύθυνα για μεγάλες διακυμάνσεις ανάμεσα στην εκτέλεση φαινομενικά ομογενών tasks. Επιπροσθέτως, εντοπίζουμε ευκαιρίες για λεπτομερή βελτιστοποίηση ενέργειας κατά την εφαρμογή Dynamic Voltage and Frequency Scaling (DVFS) για κάθε task.

Chapter 1

Acknowledgements

I would like to thank my family (Giorgos, Eleni, Ahilleas) for their lifetime support. Also I would like to thank my two advisors Dimitris Nikolopoulos and Angelos Bilas for their guidance and the financial support. Additionally, I want to express my gratitude for *CARV* laboratory and its members, especially Manolis Marazakis, Michael Ligerakis, Polyvios Pratikakis and Manolis Katevenis for their valuable assistance towards the completion of this work. Moreover, I want to thank Foivos Zakkak and Lazaros Koromilas for their valuable contribution. Other people worth to be mentioned for their assistance are: Vassilis Papaefstathiou, Markos Foundoulakis and Michael Alvanos. Lastly, to thank Foundation for Research and Technology Hellas for generously funding this work.

Contents

1	Acknowledgements	iii
2	Introduction	1
2.1	Contributions	2
3	The TPROF energy profiler	5
3.1	Trace Generation	5
3.2	Trace Analysis	7
3.3	Challenges	7
3.4	Energy Measurement Methodology	7
4	Per-Core Energy Accounting	9
4.1	Static Linear Power Model	9
4.2	RAPL Per-Core Apportioning	11
4.3	DVFS-independent Apportioning	13
4.4	Processor Independent Accounting	13
5	Application Characterization and Analysis	17
5.1	Setup and Measurement Methodology	17
5.2	Task Parallel Benchmarks	18
5.3	Application Characterization and Analysis	19
5.4	Dynamic Voltage And Frequency Scaling	24
6	Related Work	29
7	Conclusions	31

List of Figures

3.1	Example execution on a 4-core processor. Each task maintains a private struct.	5
3.2	Energy monitoring of the Intel Sandy Bridge processor. The three separate registers log: (i) the four Cores and the L3 cache, (ii) the on-chip graphics, and (iii) the whole processor die. A non-resistive current sensor measures the power consumption of the memory modules. A dedicated conversion board converts the analog signal into digital values and transfers them back to the main system.	6
4.1	Measured PP0 power for various IPC values.	14
4.2	Measured PP0 power for various FESC values.	15
5.1	Jacobi task-level energy consumption over execution time.	20
5.2	FFT task-level energy consumption over execution time.	21
5.3	Multisort task-level energy consumption over execution time.	22
5.4	Ferret task-level energy consumption over execution time.	23
5.5	Cholesky task-level energy consumption over execution time.	24
5.6	Blackscholes task-level energy consumption over execution time.	24
5.7	DVFS response of energy (dashed) and execution time (continuous) for six task-based benchmarks that we profile with TPROF.	26

List of Tables

4.1	Power model parameters and coefficients.	10
4.2	Power model error rates.	11
4.3	Validation results for the RAPL apportioning.	12
4.4	DVFS independent energy apportioning error.	13
5.1	Benchmark Summary.	18
5.2	Benchmark Average Energy and Linear Dependence (marked as Corr).	25

Chapter 2

Introduction

Energy consumption is a very important characteristic of all computing systems. At the small scale, embedded and mobile systems must perform well under limited energy resources. At the large scale, energy consumption is one of the most important economic vectors for datacenters and supercomputing. Last but not least, the physical characteristics of modern processors are reaching the limits of thermal dissipation, posing additional requirements to how hot a chip can run without failing.

For these reasons, there is an increasing interest in profiling the energy consumption of both hardware architectures and software applications. For instance, energy profilers tend to become a standard part of the development cycle for applications [29, 17, 12] as they identify energy hotspots and their equivalent causes. However, existing state of the art profilers are not suitable for parallel applications, where unique events such as synchronization, communication and interference are responsible for the majority of the energy efficiency degradation. Furthermore, existing profilers either measure the total power for the whole system or they rely on oversimplified models. This provides little information and can occasionally mislead the programmer in finding the best way to optimize the code for energy efficiency.

At the same time, modern multicore and manycore processors include an increasing number of cores, complex memory hierarchies and support for on-chip communication. Under these circumstances, profiling and accounting of parallel applications is impossible when coarsely measuring the total energy consumption of the processor, due to the variety of components that run at the same time, consuming energy at different, unknown rates [36]. Thus, per-core energy accounting is essential for accurate characterization of parallel programs. The latter is currently an open problem, due to the lack of hardware sensors that measure per-core energy consumption. Furthermore, it is not straightforward to apportion the aggregate energy consumption of the shared hardware resources (e.g., the last level cache) to cores and software entities (e.g., threads) that are executed on top of them.

On the other hand, the availability of the aforementioned systems has made parallel programming the norm, even for the average programmer. The difficulty in writing multithreaded parallel programs has led to the development of higher-level abstractions for parallel programming, such as task-parallel languages [7, 22, 14, 3], domain-specific parallel languages, pattern libraries that hide low-level threads from the programmer [34] and task-parallel programming models [8, 32, 2, 43, 43]. Compared to the traditional thread programming, task-parallel programs are easier to write, more portable, and scale better, because the parallelism is not hard-wired into the program, but created at runtime, as needed.

Task parallel languages are also better suited for energy profiling of parallel applications. Tasks to the programmer are fine-grain, self-contained computations; the system abstracts away the scheduling, synchronization and interactions between parallel parts of the program. This way, tasks offer a very convenient abstraction for the programmer to view and understand the energy consumption of the computation due to parallelism, communication and synchronization. Thus, task-based energy profiling gives a better understanding of the application energy efficiency, and also a better way to project and predict the program’s energy efficiency on different processors.

2.1 Contributions

The main contribution of this work is the design and the implementation of TPROF, an energy profiler for task-parallel programs. TPROF can be used to analyze the energy-performance aspect of individual dynamic or static OpenMP-like tasks, correlate unique parallel features such as wait times, work stealing and interference effects with the energy consumption and compare the energy efficiency of different parallel implementations on various architectures. In addition, TPROF is able to predict the best Dynamic Voltage and Frequency Scaling (DVFS) operating point of the processor which is valuable in energy optimization, as it minimizes either the energy or the Energy Delay Product (EDP) of the system. Lastly, TPROF incurs low overheads and produces fine-grain profiling information that is easy to visualize on the program source code, and help the programmer understand and optimize the energy efficiency of the application.

Overall, we make the following contributions:

1. We design and implement TPROF, an energy profiler for the BDDT runtime, which implements OpenMP tasks [43].
2. We propose three methods for *per-core energy accounting*, which is essential for accurate per-task energy estimation: (i) prediction using a linear model, (ii) apportioning of actual energy based on core utilization, and (iii) combining the two, to apportion actual energy measured using a realistic model.
3. We show the value of TPROF by characterizing a set of benchmarks and produce per-task fine grained energy profiles for these applications. The energy profiles give useful insights on application behavior: we discovered that

task granularity, data locality, and data sharing among tasks affect energy consumption even when they do not affect task performance.

4. We use TPROF to explore the effect of various processor DVFS configurations on our benchmarks. We discover that applications with bad locality and limited parallelism can run with 50% less energy, without sacrificing performance. Overall, we observe that applications contain tasks whose energy, performance, or both, vary for different DVFS configurations. We believe this observation enables task schedulers to optimize for energy efficiency in future architectures with per-core DVFS.

The rest of the paper is organized as follows. Section 3 presents the design and implementation of TPROF. Section 4 illustrates the per-core energy accounting methodology. Section 5 presents the evaluation of TPROF by characterizing a set of task-parallel applications, and discusses the energy profiles produced. Section 6 discusses related work and Section 7 concludes our work.

Chapter 3

The TPROF energy profiler

TPROF is an energy profiler for OpenMP-like task-parallel applications. To estimate the per-task energy consumption TPROF measures the total energy consumption of the processor and apportions it into tasks. TPROF is a two-phase profiler. The first phase gathers run-time energy and performance information on each task instantiation and completion to create a trace. The second phase analyzes the trace to create and visualize the per-task energy consumption. This section is dedicated to the description of the above characteristics.

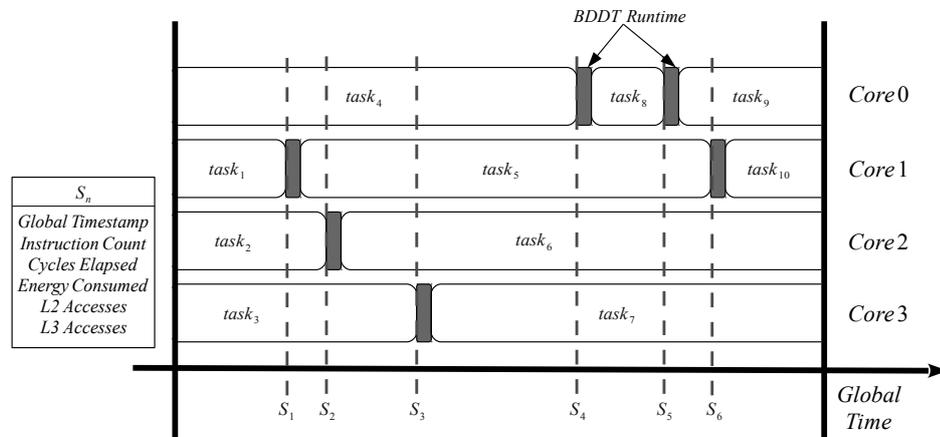


FIGURE 3.1: Example execution on a 4-core processor. Each task maintains a private struct.

3.1 Trace Generation

To profile an application, TPROF creates a trace of the application execution, containing precise information about every task instance. To create the trace, TPROF extends the task instantiation and completion mechanisms to log a sample of the values of energy and performance counters. The traces are stored in separate per-core structures, to avoid thread communication and minimize the synchronization

overhead. Each sample consists of (i) a timestamp, (ii) the instruction count since the last sample, (iii) the number of elapsed cycles, (iv) the number of the last level cache (LLC) misses and (v) the energy consumed by all cores and their caches. To obtain the required information for the samples TPROF uses the High Precision Event Timer (HPET) for the timestamp, the Performance Application Programming Interface (PAPI) [15] for the performance counters and the Running Average Power Limit (RAPL) interface [1] Machine State Registers (MSRs) for the energy measurements.

Figure 3.1 presents an example of TPROF’s trace generation on a 4-core system. The program consists of ten tasks, marked $task_1$ to $task_{10}$, scheduled on 4 cores. Core 0 executes tasks 1, 8 and 9, core 1 executes tasks 2, 5 and 10, core 2 executes tasks 3 and 6, and core 3 executes tasks 4 and 7. After every task and before the next, the runtime library performs the completion of the previous task and the instantiation of the next task, respectively, as depicted by the gray intervals. Each core records a sample at task completion and instantiation. The Figure omits the instantiation samples for readability, showing only completion samples named S_1 to S_6 . Note that each core records the samples independently and the total order of all the samples is achieved using the global timer readings.

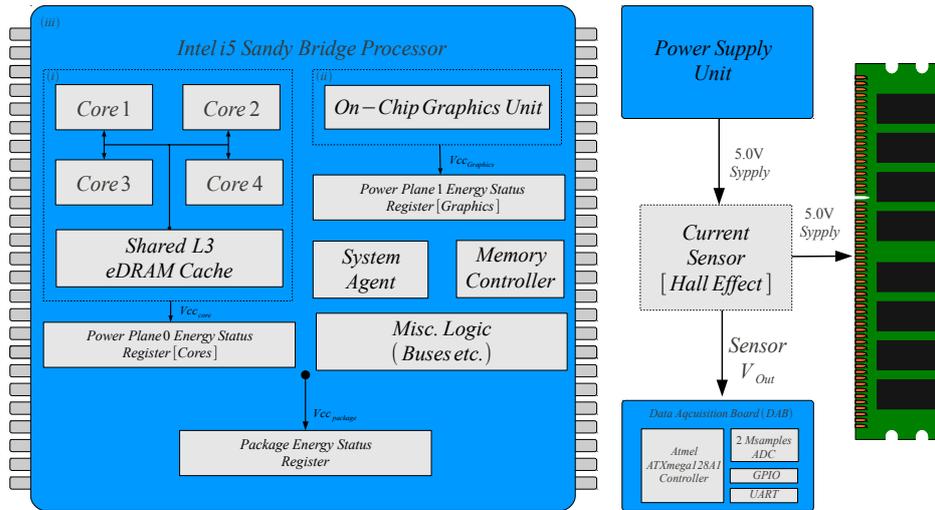


FIGURE 3.2: Energy monitoring of the Intel Sandy Bridge processor. The three separate registers log: (i) the four Cores and the L3 cache, (ii) the on-chip graphics, and (iii) the whole processor die. A non-resistive current sensor measures the power consumption of the memory modules. A dedicated conversion board converts the analog signal into digital values and transfers them back to the main system.

We also evaluate TPROF’s overhead using a microbenchmark which instantiates and executes 2.5 million task instances, resulting in a trace of 5 million samples. The total overhead per sample ranges from $1.9\mu\text{sec}$ to $4.3\mu\text{sec}$ for CPU frequencies of 3.6Ghz and 1.6Ghz respectively. We take one sample at the instantiation of each

task and another at its completion, resulting in two samples per task instance. Since the task execution time for TPROF to report correct results cannot be less than 1ms, TPROF’s overhead is no higher than 0.86% of the minimum affordable task execution time.

3.2 Trace Analysis

After sampling, TPROF performs an offline analysis on the generated traces to infer the per-core and per-task energy breakdowns and obtains the energy profile of the application. First, TPROF combines the per-core traces into a global trace of samples using the global timestamps. Since all samples use the global timer, this is straightforward. TPROF then infers the missing values of per-core performance counters by linear interpolation. Note that each sample is taken by a single core and although the values of the clock and energy register are shared among all cores, the values of the performance counters are local to each core. For each and every one of the missing cores, TPROF takes the previous and the next available sample and interpolates between the previous and next available values of the missing counters to infer the missing values. Based on both the interpolated and measured values of the performance counters, TPROF computes the amount of total energy consumed by each core for each quantum of time between every two samples. Finally TPROF adds the computed energy quanta to calculate the total energy consumption per task instance and groups the task instances of each task to find its energy consumption.

3.3 Challenges

TPROF’s main challenge was to achieve per-task energy accounting, as the tasks utilize shared resources such as the LLC, interconnects and the DRAM on a multicore system. Thus, we had to apportion the energy consumption of these components and map the physical cores to tasks, as there is no other available technique that allows per-core energy accounting. For example, the RAPL energy subsystem, which is the state of the art in fine-grained energy accounting, reports only the total processor energy consumption. The second challenge of TPROF was to achieve DVFS-independent power instrumentation. This characteristic incurs challenges, as there is no available infrastructure to monitor the DVFS changes and TPROF is essentially a userspace application. In Section 4, we present our approach that tries to cope with the above challenges. The third challenge was to achieve energy and performance instrumentation without inter-task synchronization and communication. To achieve this feature, we keep dedicated (local) per-task structures and use global hardware information (such as global timers and energy sensors).

3.4 Energy Measurement Methodology

TPROF uses Intel’s RAPL interface [35] as a primary source for energy measurements. The Sandy Bridge and Ivy Bridge microarchitectures implement on-chip energy counters using a precomputed model instead of analog power meters. Specif-

ically, the power control unit (PCU) collects on the fly a set of architectural events from all the logical cores, the graphics processor and the I/O and combines them using a weighted model to compute energy consumption. The PCU also collects thermal information from on-chip thermal sensors to fit the processor activity on the operating temperature. The total, processor energy reported includes the leakage information of the die, which is hard-coded on the chip in advance.

The die of a desktop Sandy Bridge processor uses two separate power planes, as shown in Figure 3.2. Power Plane 0 (PP0) supplies the processor cores, the ring interconnect and the LLC. Power Plane 1 (PP1) supplies the graphics processor, that can use a different voltage and frequency. The processor exposes one MSR for each of the power planes and one for the whole processor package. The package energy register includes the System Agent (SA), the Memory Controller (MC), and the rest of the processor logic energy consumption. These registers are updated almost every millisecond ($957 \mu s$). This refresh rate places a limit on the task granularity of the profiled application, requiring tasks to run for more than 1ms. This granularity is equal to existing state of the art energy instrumentation schemes such as PowerPack [16].

Server Sandy Bridge and Ivy Bridge processors expose a MSR for DRAM energy consumption instead of PP1. This MSR is absent in the majority of the available processors, thus TPROF uses a direct physical instrumentation scheme for measuring the power consumption of the DRAM. Our scheme, shown in Figure 3.2, consists of a Hall-Effect analog current sensor that intercepts the 5.0 Volt power supply line of the power supply unit. A dedicated board performs analog to digital conversion to the sensor signal and transfers them back to the system via a serial interface. The effective sampling rate of the board is up to 40Khz which is also responsible for implementing a small running average with a window of 10 samples resulting in an available sampling rate of 1Khz. We present further information about our hardware instrumentation scheme in [25].

Chapter 4

Per-Core Energy Accounting

Previous work in modeling the power consumption of computer systems has mainly focused in coarse-grained whole system or processor-wide chip estimation [4, 20, 26]. However, those approaches cannot predict the energy consumption of each core due to the hardware resource sharing and the dynamic behavior of those cores.

In this section we describe three methods for achieving per-core energy breakdown and accounting of our target processor. The first uses a static linear model that describes the power consumption of the processor. The second enhances the precision of the static model approach by *apportioning* the RAPL energy counter. We also show that this approach is *independent of the operating voltage and frequency* of our processor. As a third approach, we propose a *processor independent* method for calculating the per-core energy consumption. All our methods can be generalized in a wide range of profiling applications that require energy breakdown of individual cores that share energy estimation resources and are available to use with TPROF.

4.1 Static Linear Power Model

Prior work dedicated to modeling and forecasting the energy and the power consumption of various processors [23, 13, 19] suggested that a Processor's activity has linear relation with its power consumption and can effectively be modeled by a linear model. The number of parameters varies among different models. The larger the parameter count, the more accurate the model is. Despite that, processors often cannot provide an unlimited number of parameters (common case is four). Respecting this limitation, we build a linear model of three parameters.

Processor power consumption exhibits a linear relationship to activity rates of individual processor components, such as the instruction fetch logic, ALUs, branch predictors and cache memories [5, 41, 19]. These activity rates are available to the power modeler via counters included in hardware performance monitoring (HPM) units. While several linear power models with input from hardware performance counters have been proposed in the literature, these models tend to either use many counters, making online power modeling intrusive, or rely on counters which are

not available across processor architectures.

Equation 4.1 illustrates our power model for Intel’s Sandy Bridge and Ivy Bridge processors. It includes IPC, Level 2 cache and LLC accesses (both in GB/s) along with the parameter weights and the processor leakage and background power. We do not include DRAM accesses in our model since the power consumption of the memory is directly measured and accounted to each core depending on its equivalent traffic. In Table 4.1 we include a detailed description of every symbol in our model along with its corresponding value.

$$P = \sum_{i=1}^n (IPC_i \times a + L2_i \times b + LLC_i \times c + C_1) + C_2 \quad (4.1)$$

Symbol	Value	Description
IPC_i	float	Instructions Per Cycle for core i
$L2_i$	float	L2 cache accesses caused by core i in GB/s
LLC_i	float	LLC accesses caused by core i in GB/s
a	1.10	Coefficient of IPC parameter
b	0.08	Coefficient of $L2$ parameter
c	0.19	Coefficient of LLC parameter
C_1	8.00	Core Static and background Power
C_2	5.20	Processor Static and background Power

TABLE 4.1: Power model parameters and coefficients.

To obtain the model parameters we run a large variety of application kernels and perform linear regression. We use both micro-benchmarks that stress each component of the model, as well as real kernels that stress all components. To maximize the training quality, our kernels range from CPU intensive to memory intensive.

We evaluate the accuracy of our model and verify whether it is sufficient for multicore prediction by running a series of computational kernels. Those kernels are a representative sample of the Polybench [33] benchmark suite including linear algebra, stencil and data mining computational kernels. During the evaluation procedure, we cross-compare the estimated power and the power reported by RAPL. To assess whether the model is suitable for multicore power prediction, we compare two versions of our model, one constructed with input from serial execution ($n = 1$) and one constructed with input from parallel (quad-core) execution ($n = 4$). For the quad-core version, we run each computation kernel as an independent Linux

process. No communication or synchronization is performed by the workloads during execution. Finally, in order to avoid unpredicted system interference, we run each kernel 10 times and average the reported power.

Table 4.2 shows the error rates of fifteen computational kernels. For the single-core version of our model, we observe a maximum error of 11.6% while for the quad-core version we record a maximum error as high as 14.6%. The results indicate that our model is able to predict the power consumption of compute and cache intensive kernels accurately. However, our model is unable to produce acceptable results for the memory intensive kernels and more specifically for those where the data access pattern is not continuous in memory (e.g. jacobi). Furthermore, we observe that the error on the quad-core version is larger on average. We regard the small number of model parameters as the primary source of this error factor. Specifically on the Sandy Bridge and Ivy Bridge microarchitectures, inclusion of DRAM CAS, RAS and pre-charge commands in the model would improve significantly its accuracy.

Kernel	Single Core Error %	Quad Core Error %
gemm	3.18	10.80
jacobi1d	2.49	12.65
jacobi2d	1.97	6.71
fdtd-2d	2.80	10.31
seidel-2d	6.03	0.38
2mm	8.44	4.15
3mm	11.6	4.16
adi	3.14	7.92
correlation	2.11	14.42
covariance	3.66	14.62
symm	8.67	3.46
gemver	6.67	3.52
mvt	6.07	3.31
syrk	1.47	4.20
trmm	1.80	6.28
Mean	4.67	7.12

TABLE 4.2: Power model error rates.

4.2 RAPL Per-Core Apportioning

We have established that our power model is sensitive to parallelism, yielding lower precision with more active cores. To improve the accuracy of multicore power estimation, we apportion the RAPL energy counter. To estimate the portion of each core’s power (and therefore energy consumption) we partition execution time in quanta of length T and adopt the following procedure:

1. Collect performance counters per time quantum T and core C_i ($Perf_i$).
2. Collect processor-wide energy consumption from RAPL per time quantum T ($Rapl_i$).
3. Fit the performance counter values ($Perf_i$) in the model and estimate the power consumption per core C_i ($Power_i$).
4. Based on the ratio of all the available estimated power values ($Power_i$), calculate the **percentage** of $Rapl_i$ that each core C_i is accountable for during quantum T .

To prove that our approach can accurately predict the per-core power consumption, we perform a series of experiments. In those experiments we run a variety single-threaded kernels from Table 4.2 in parallel. We collect the RAPL values and we use our model to estimate the power consumption. In addition, we use our methodology to calculate the estimated portions of the power that each core is responsible for. Then, we run each kernel in isolation and record the actual power consumption that RAPL reports. Finally, we compare the real RAPL values with the predicted model values and the estimated RAPL portions for each core. To ensure that the single-core workloads result into the same performance and power consumption when executed as part of a parallel workload, we schedule the parallel runs so the included programs avoid contention for shared resources.

In Table 4.3, we present the obtained results for the validation procedure of the RAPL apportioning. We group every parallel run in a single row. In column *Real* we report the real single-core power consumption of each kernel that we compare against the static model and the RAPL apportioning (noted as *Model* and *Slice* respectively). Additionally, we present the equivalent error rates of both techniques.

Kernel	Real	Slice %	Slice	Model	Err. Slice %	Err. Model %
atax	9.44	47.2	8.69	9.55	+7.8	+1.1
2mm	8.95	52.7	9.74	10.66	+7.8	+19.0
atax	9.44	44.0	9.50	9.61	+0.6	+1.8
jacobi1d	12.20	56.0	12.09	11.89	-0.9	-2.5
atax	9.31	51.9	9.22	9.59	+0.96	+3.0
adi	8.29	48.1	8.54	8.89	+3.01	+7.23
bicg	9.09	45.7	8.56	9.03	-5.8	+0.6
2mm	9.15	54.2	10.15	10.71	+10.9	+17.0
corr	9.65	49.9	9.83	8.47	+1.8	-12.2
cov	9.59	50.1	9.86	8.49	+3.3	- 11.4

TABLE 4.3: Validation results for the RAPL apportioning.

We justify energy fluctuations between two different runs of the same kernel as a result of the variable operating temperature of the processor, OS scheduling as

well as caching effects. The equivalent model fluctuations are a result of the small performance variation across different runs and the variable LLC cache interference with their parallel workloads.

4.3 DVFS-independent Apportioning

Different DVFS states heavily affect the power consumption of our processor as the power consumption is a function of operating voltage and frequency. Although DVFS-aware power modeling and prediction is possible [41, 44], fast transitions can effectively destroy our effort for fine grained energy accounting. The primary reason is that our task runtime should be aware of *each* DVFS state change during the complete execution of the workload. This approach is problematic since the system can alter its state asynchronously of TPROF via the hardware power control unit, the O/S or an user-space control governor.

We conduct another experiment to show that this apportioning approach is independent of the processor’s DVFS state. In this experiment, we use our model which we train at the 3.3Ghz DVFS state, in order to perform parallel energy accounting of our processor when it operates at 1.6Ghz. Table 4.4 presents the values obtained by the experiment. The acceptable error rate is a result of the constant energy cost ratio between our model parameters during every possible DVFS state. For example, if the energy ratio between the pipeline activity and the LLC cache activity is 1 : 2 when the processor operates at 3.0Ghz, the ratio will remain the same during a potential scale down at 1.6Ghz.

Kernel	Real	Slice %	Slice	Err. Slice %
atax	3.12	48.1	3.17	+1.6
2mm	3.24	51.9	3.42	+5.5
atax	3.12	45.7	3.63	+16.1
jacobi1d	4.54	54.2	4.30	-5.3

TABLE 4.4: DVFS independent energy apportioning error.

4.4 Processor Independent Accounting

An alternative method to perform per-core energy apportioning is to use a single metric instead of a pre-calibrated static model. Reducing the effective metric to one, can be useful in cases where the processor’s architecture is unknown, a model cannot be trained, or the available performance counters count is restrictive. To conform with such cases, we use the processor’s pipeline activity as our single metric. We try two different pipeline activity metrics and present the relation between them and power consumption. Specifically, we conduct experiments with the *Instructions Per Cycle* (IPC) and the *Front-End Stalled Cycles* (FESC). IPC is traditionally used in the literature as a good predictor of the energy consumption [23, 13, 19]. FESC on the other hand, is the number of stalled cycles over the total clock cycles, where the pipeline’s front-end could be stalled due to a data

hazard (e.g., Read After Write (RAW)), a structural hazard (lack of resources) or a control hazard (e.g., branch). In the following experiments we compute the metric's ratio for each core and divide the total energy among cores proportionally to this ratio. We show that both IPC and FESC are linearly correlated with total core power consumption.

Figure 4.1 presents the relation between power consumption reported for PP0 and the IPC. To sufficiently cover the IPC range, we designed custom workloads, tuned to achieve different levels of IPC (0.4–2.8), using varying levels of cache and memory activity. The relatively good linear fit of the data supports our decision to use IPC ratios to infer per-core energy consumption.

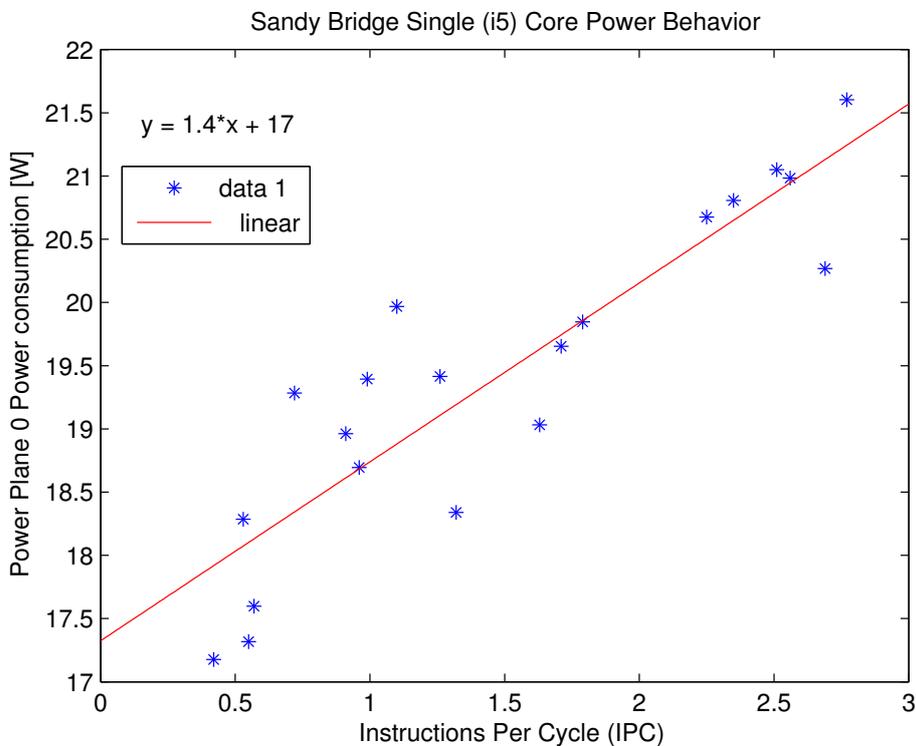


FIGURE 4.1: Measured PP0 power for various IPC values.

Figure 4.2 presents the results, namely the power consumption in PP0 for different FESC ratios, produced using the same custom workloads as above. We observe that the FESC metric has a better linear fit than IPC and can reliably be used in cases where our model cannot be trained.

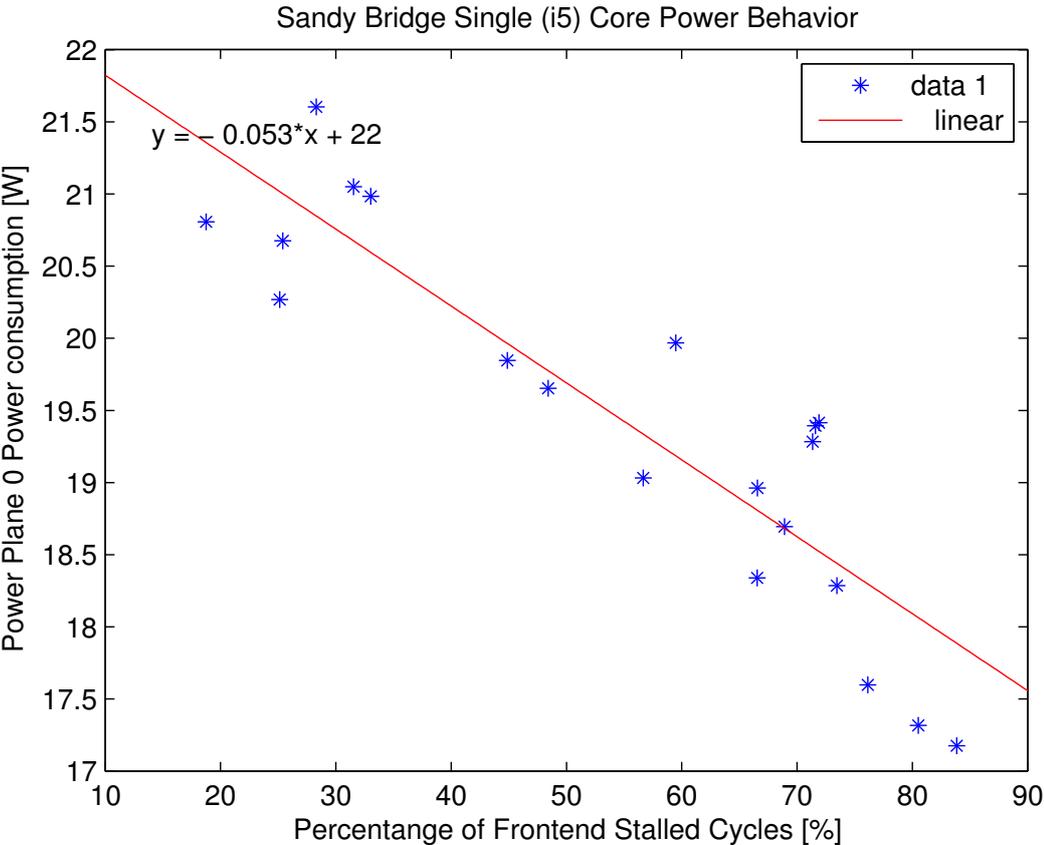


FIGURE 4.2: Measured PP0 power for various FESC values.

Chapter 5

Application Characterization and Analysis

In this section we characterize six task-parallel TPROF programs using TPROF. As a baseline task runtime we use BDDT which provides the mechanisms for task issuing, instantiation and completion [43]. BDDT uses an OMP-like task syntax and it was preferred over other alternatives such as openMP or ompSS due to the authors' familiarity with it. Such alternatives are also compatible with TPROF. Furthermore, we present our test system along with a detailed description about the benchmarks such as number of static tasks and input size.

During our characterization we discuss various observations about the energy distribution of each static task and how it correlates with various performance factors such as locality, access pattern and task footprint. We conclude that those factors result into higher energy consumption as they result into a massive amount of lower level cache misses. Additionally, we present the energy and performance impact of Dynamic Voltage and Frequency Scaling (DVFS) on each task. DVFS is a technique that lowers the voltage and the frequency of the processor at runtime to achieve lower workload energy consumption. Usually, DVFS introduces a trade-off between energy and execution time, as lowering the frequency results into lower performance. Using TPROF, we find that the performance of some tasks is *independent* of the operating DVFS state, due to locality and access pattern issues.

5.1 Setup and Measurement Methodology

All experiments were conducted on a computer node with 4GB memory and an Intel Core i5-2500. The Intel Core i5-2500 processor features a three-level cache hierarchy. The level 1 data cache as well as the instruction cache is of 32KB size per core. The level 2 cache is of 256KB size per core and the LLC is of 6MB for the whole package. All executables were compiled using GCC 4.6.3 and the -O3 flag on. Our experiments measure the performance of the parallel section of the code, excluding any initialization and I/O at the start and end of each benchmark.

5.2 Task Parallel Benchmarks

Table 5.1 gives a summary of the benchmarks. The first column shows the name of the benchmark. The second column shows the size of the largest benchmark’s task footprint. The task footprint is calculated by its arguments only, meaning that its actual memory footprint can be larger. The third column shows the number of the task invocations in the code and the fourth column shows the number of the actually instantiated tasks at runtime. Because TPROF samples every task creation and completion, the number of task instances reflects the number of samples taken by TPROF, while the number of tasks shows the workload’s polymorphism.

Benchmark	Task footprint	Tasks	Task instances
BlackScholes	1.2 MB	1	363
Ferret	N/A ¹	1	1000
Cholesky	2.4 MB	4	5984
Jacobi	32.8 MB	1	800
FFT	0.6 MB	6	1363
Multisort	12.3 MB	2	256

TABLE 5.1: Benchmark Summary.

Black-Scholes is a parallel implementation of a mathematical model for price variations in financial markets with derivative investment instruments, taken from the PARSEC [6] benchmark suite. It decomposes and processes the data in rows. All Black-Scholes tests use an input data file of 16 million elements with task work size equal to 44000 elements.

Ferret is a content-based similarity search engine toolkit for feature rich data types (video, audio, images, 3D shapes, etc). We used an image similarity search engine configuration from the PARSEC benchmark suite. We extract parallelism by issuing multiple queries to the search engine. A query consists of an image, thus the number of images determines the amount of parallelism. We use 1,000 images to issue queries to a database which contains 59,695 images.

Cholesky is a factorization kernel used to solve normal equations in linear least squares problems taken from the benchmarks distributed with SMPSS [38] benchmarks. It consists of four parallel phases that perform tiled operations, corresponding to a task per tile per phase. We used a 10240×10240 double precision input matrix in tiles of 320×320 elements.

Jacobi also taken from SMPSS benchmarks, is a parallel implementation of the Jacobian method for solving systems of linear equations. It uses a 2-dimensional

¹The Ferret implementation in BDDT runs one task instance per image. Each task instance loads independently the data it needs thus we don’t know the actual task footprint

array with tiled layout. This Jacobi implementation is part of the SMPSs distribution. Each parallel task in Jacobi processes a tile of the array with a kernel implementing a 5-point stencil computation. We test Jacobi on a 5120×5120 input matrix and a tile size of 1024×1024 elements.

FFT is an implementation of a 2-dimensional Fast Fourier Transform algorithm, taken from the SMPSs benchmarks. It consists of five parallel loops that alternate in transposing the input array and performing 1-dimensional FFT on each row. Each task created in the FFT calculation loop operates on an entire row of the array, while transposition phases break the array into tiles and create a task to transpose a group of tiles. In FFT tests, the input matrix contains 5656×5656 elements, the block size is 200×200 elements, and the transpose tile size is 200×200 elements.

Multisort is a parallel implementation of Mergesort. Multisort is an alternative implementation of the Cilk sort test from Cilk [7]. It has two phases: during the first phase, it divides the data into chunks and sorts each chunk. During the second phase, it merges those chunks. We use Multisort on 256MB of input data with the tasks operating on about 4MB for the sort task and about 12MB for the merge task.

5.3 Application Characterization and Analysis

We present the task-based application characterization with a single scatter plot for each benchmark (Figures 5.1 - 5.6). On the x-axis lies the execution time of each task instance while on the y-axis lies its equivalent energy consumption. Additionally, each static task stands with a different marker. To investigate whether any imbalances between task running on different cores, we include a separate plot for each of our four cores. Our scatter plots merge the following four energy-performance dimensions:

1. the estimated energy per task over the application execution time
2. the task execution time over the application execution time
3. the energy cumulative distribution of all tasks
4. the execution time cumulative distribution of all tasks

Although TPROF produces various performance metrics such as complete application timeseries that show inter-task effects such as delays from dependencies and worker starvation among with cache hierarchy misses we do not present them for readability reasons. Also, despite carefully configured, some tasks' execution time happens to be less than the processor's energy counters update interval as described in chapter 3, resulting in near-zero energy consumption measurements. Lastly, because our memory has very low dynamic power (in the range of 2 Watts), we measure only the processor's energy consumption, excluding the energy from the memory.

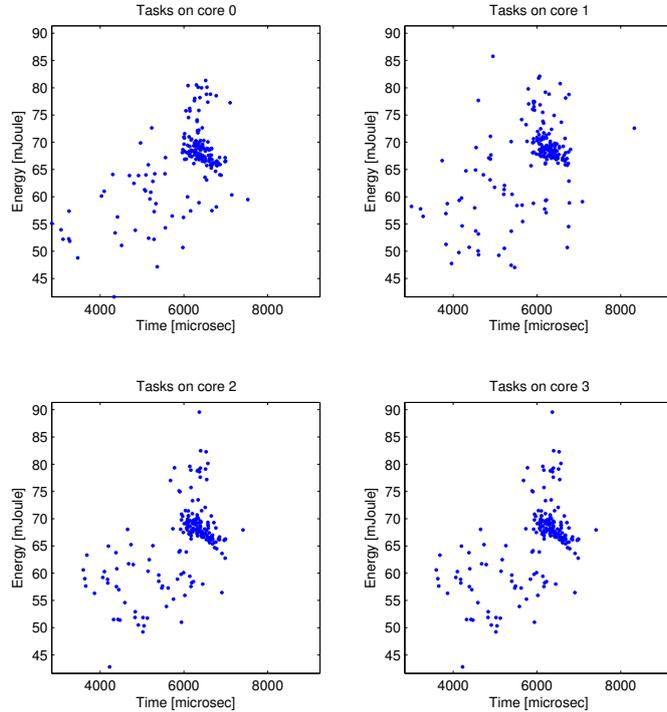


FIGURE 5.1: Jacobi task-level energy consumption over execution time.

In Figure 5.1 we present the energy-performance characterization of Jacobi. Jacobi has only one task that we note with a single dot. For this single task, we observe a mean processor energy consumption of 67.3 mJoules. The variation that we observe in energy consumption as well as the execution time is a result of its large task footprint and the tiled access pattern used by the kernel. With a task footprint of almost 33MB, Jacobi tasks create a lot of last level cache misses. More cache misses mean more halted CPU cycles while consuming energy to fetch the data from the RAM. While there are 4 tasks running simultaneously and the application is memory intensive the energy consumption depends highly on caching and prefetching. At the same time the tiled access pattern reduces efficiency of the prefetcher. All that considered it is expected to have large variation in the power consumption among the task instances. Another interesting point of these sub-figures fact is the task's energy consumption on core 3. Note that on cores 0, 1 and 2 tasks are being executed from the very beginning of time, while core 3 is executing the runtime. When core 3 also starts executing tasks the memory traffic is increased resulting in more LLC misses and larger delays.

Figure 5.2 provide the equivalent results for the Fast Fourier Transformation (FFT). Our implementation of FFT has six different tasks. Each of these tasks appear with a different marking. Two tasks are the two FFT rounds while the other four are transpose phases. The transpose phases, due to their simplicity have very

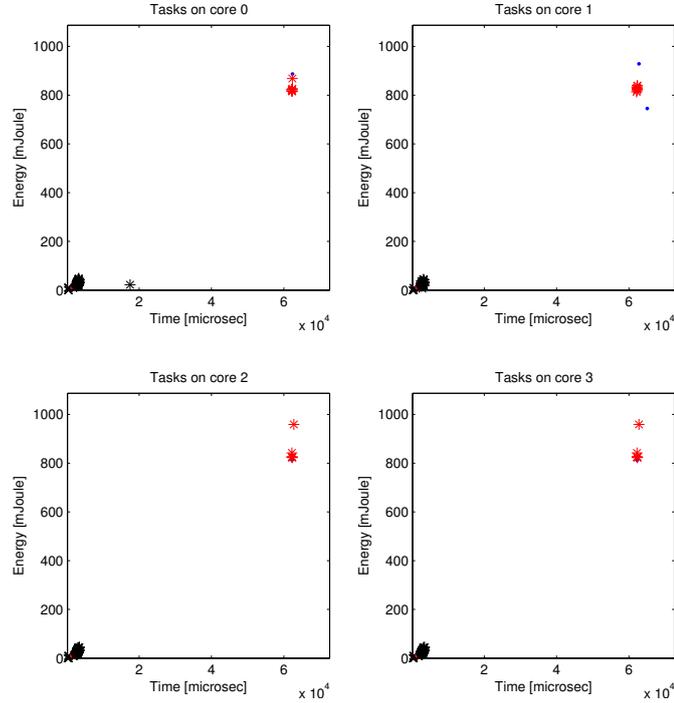


FIGURE 5.2: FFT task-level energy consumption over execution time.

short execution time, resulting in near-zero energy consumption measurements. This is also the reason why the mean energy and execution values are so low for those four tasks. On the contrary the two FFT rounds dominate almost 100% of the total benchmark energy, that equals to 850 mJoules. both FFT rounds have steady execution time and only the first and the last task instances of both vary from the rest regarding the energy consumption. We believe that this is caused by cache prefetching activity due to a higher miss rate caused by the continuous execution of many different tasks per core in the initial transpose phases. Regarding the reduced energy consumption for some FFT round task instances, we correlate it to the fact that the last FFT phases operate on zero valued elements due to the array dimensions not being a multiple of the block size.

Figure 5.3 we present the obtained results for the Multisort benchmark. The *Sort* task of Multisort, noted with a dot, dissipates on average 173.4 mJoules of processor energy while the *Merge* task, presented as a star, features an average energy consumption of 51.3 Jmoules. This behavior is a result of the *Sort* task performing noticeably more work than the *Merge* task. Both *Merge* and *Sort* tasks have noticeably energy variation among its different instances. This variation is caused by the different data locality of the arguments and the data exchanging during the final merge step but also due to the large task footprint - 12MB - which is well higher than our LLC cache. The latter is the reason why both tasks have high variation in execution and energy but also high dispersion.

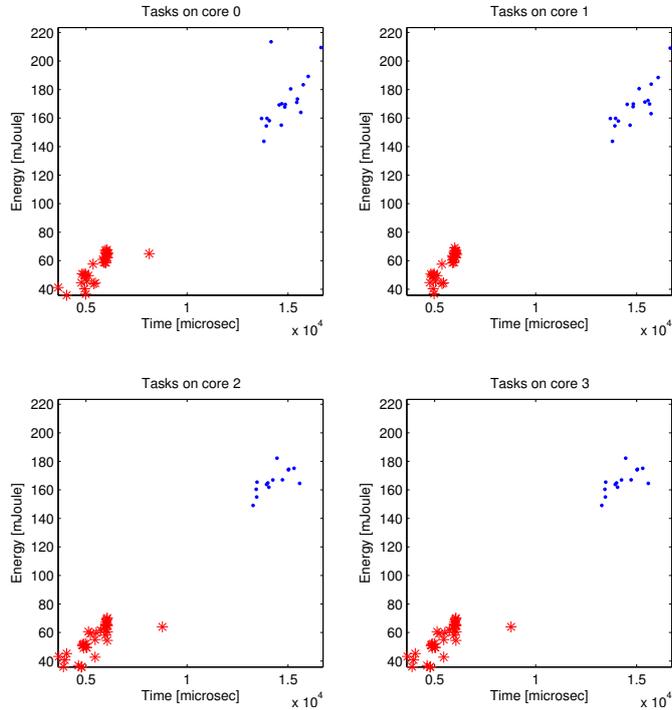


FIGURE 5.3: Multisort task-level energy consumption over execution time.

In Figures 5.4 we present the energy-performance characterization of Ferret. Ferret has only one task that we note with a single dot. This task has a mean processor energy consumption of 300 mJoules per instance. We also calculate the standard deviation to range from 140 - 180 mJoules depending on the core. As we previously noted, Ferret performs a high number of I/O operations to load the images. Reading and writing from/to the filesystem along with the random-like access pattern, that generate cache misses and potentially disc accesses, are the cause of the observed variation. Thus, the execution time ranges from a few to 60 milliseconds. Furthermore, for the region with lower execution time, the energy is linearly correlated. On the contrary, for higher execution time its equivalent energy has high deviation from the linear trend. Ferret shows how factors like data locality and I/O can affect energy consumption. To improve ferret's energy efficiency and behavior one could increase the I/O operation size, perform image prefetching and redesign the process to reduce context switches.

In Figure 5.5 we explore the energy-performance behavior of Cholesky. Out of the four total tasks, the three most energy demanding tasks are grouped into two distinct zones. Their execution time is stable but their equivalent energy consumption has notable variation. The most demanding task (trsm), noted with x-mark, dissipates from 85 to 120 mJoules of energy. The gemm (noted with a star) and syrkm tasks (noted with a dot) follow the same behavior - although with lower per-task energy consumption. We account this energy variation to the memory

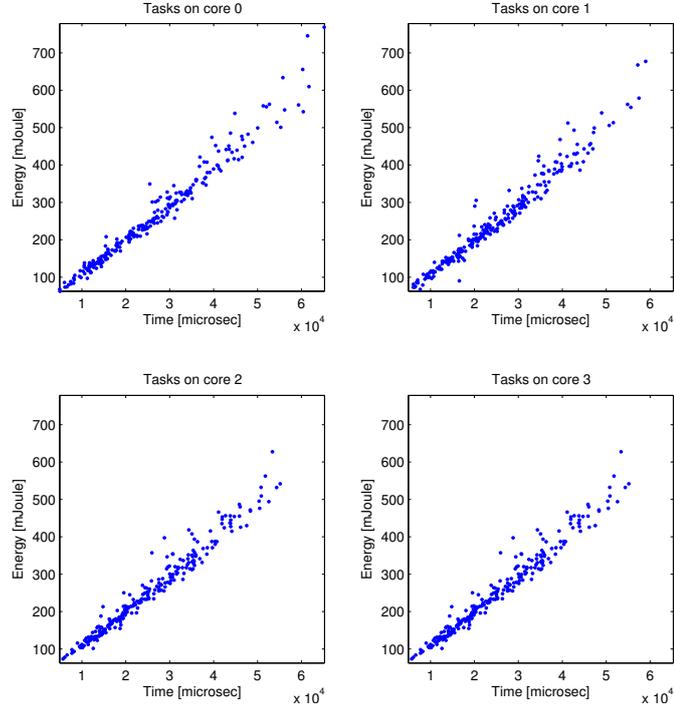


FIGURE 5.4: Ferret task-level energy consumption over execution time.

and inter-cache hardware prefetcher which generates speculative traffic but does not alter the task execution time. We consider the latter side-effect as the primary reason behind the lack of linear correlation between the execution time and the energy consumption.

In Figure 5.6 we present the energy-performance characterization of Black-Scholes. Black-Scholes has only one task that we note with a single dot. This task has a mean processor energy consumption of 58.5 mJoules per instance and the standard deviation is about 2.5 mJoules. The task instances are separated into two discrete groups when it comes to the energy consumption. One group consumes about 62 mJoules while the other consumes about 52 mJoules, 16% less. We believe that this variation in the energy consumption is the result of two main code paths in the task. To confirm this assumption we examined the source code and found that there are several if statements, in the task's scope, resulting to different call graphs. The energy consumption appears to be stable among all the task instances following the same code path. Note that the task footprint is 1.2MB. With such small task footprints and the normal access pattern of Black-Scholes, LLC misses are kept to the minimum. As a result the reported numbers for both energy consumption groups are stable.

In Table 5.2 we provide the mean energy and the (linear) dependence of each static task. We use dependence as metric of linear correlation between the energy consumption and the execution time, to quantify the possibility of optimization

in cases where the execution time variation is high. To generate it, we use a standard correlation algorithm that outputs a value from 0 (low) to 1. During our characterization, we find that memory intensive tasks and tasks with bad locality tend to have high diversity that is under 0.8. For example jacobi, merge and sort tasks have strong time variance but their energy trend is not linear. In such cases there is a strong evidence of locality issues and potential optimization. On the contrary, processor intensive tasks tend to have very low diversity that approaches 1.0 (e.g ferret). The tasks of Cholesky and Blackscholes produce misleading linear dependece values due to the the minimal variance in their execution time. This observation implies increased parallel activity which is accounted to the hardware prefetcher.

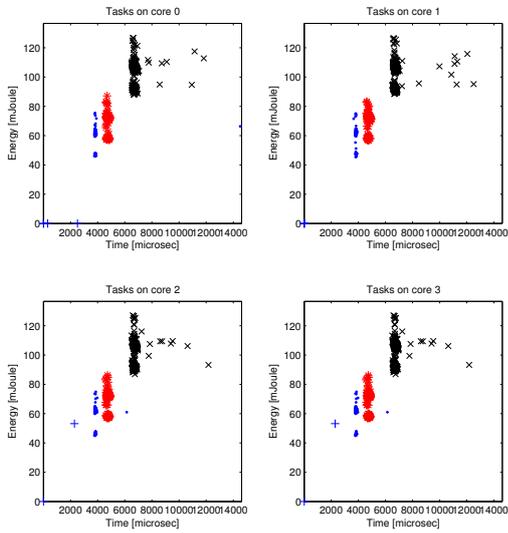


FIGURE 5.5: Cholesky task-level energy consumption over execution time.

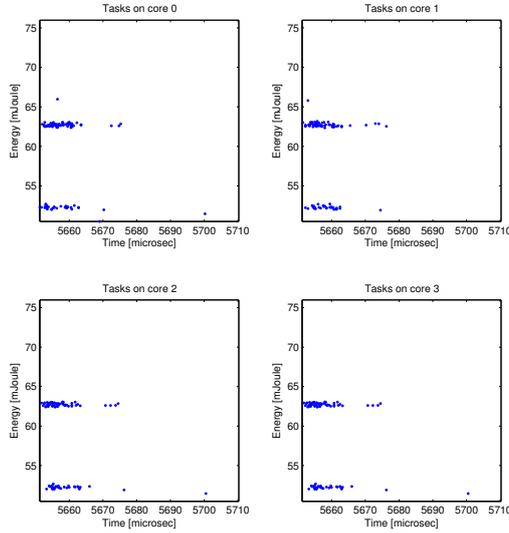


FIGURE 5.6: Blackscholes task-level energy consumption over execution time.

5.4 Dynamic Voltage And Frequency Scaling

Dynamic Voltage And Frequency Scaling (DVFS) is a processor feature that allows lowering the operating frequency and voltage in order to save energy. The DVFS implementation in the Sandy Bridge architecture is known under the name *P-States*. P-States do not allow voltage (V) and frequency (F) to be set individually. Instead, hardware-predefined pairs of V and F exist. Additionally, P-States cannot be applied in a per-core basis, they can only be applied on the whole package. Despite that, per-core frequency tuning is possible with voltage constraints. The DVFS transition, from one P-State to an other, costs about 10 μ sec.

Previous work show the energy gains, the performance degradation and the power impact of the DVFS during various workloads. Spiliopoulos et al. [40] propose *The Green Governor*, an adaptive Linux governor that uncovers possibilities

Energy and Linear Dependence			
Benchmark	Task	Energy (J)	Corr
Jacobi	jacobi	67.3	0.483
FFT	fft ₁	849.8	0.904
	fft ₂	837.9	0.409
	transp_blk	3.6	-0.084
	transp_swp	6.0	-0.014
	tw_transp_blk	15.1	0.291
Multisort	tw_transp_swp	29.0	-0.022
	merge	173.4	0.579
Ferret	sort	51.3	0.794
	ferret	262.8	0.983
Blackscholes	blackscholes	58.5	0.062
Cholesky	gemm	59.8	0.091
	syrc	69.4	-0.081
	potrf	102.7	0.053
	trsm	1.3	NaN

TABLE 5.2: Benchmark Average Energy and Linear Dependence (marked as **Corr**).

in processes to find the optimal energy operating point. On the other hand, Isci et al. on [18], splits the workload into *phases* that share performance characteristics and decide on the fly which DVFS state to apply.

During our characterization in Section 5 we concluded that task-based applications perform natural categorization on their work. For example Multisort has two distinct parts, i) divide and sort individual sub-arrays and ii) merge the sorted arrays into a global sorted array. Those two tasks have different execution characteristics such as their execution intensity, their data access pattern and their data exchange rate. In a sense, tasks are natural phases of an application. As we show in our characterization, even the memory intensive task instances have uniform behavior as they follow a pretty well defined pattern and they are in general predictable. Furthermore, tasks are more fine grained than processes and threads, allowing greater control and logical breakdown.

In this sub-section, we use TPROF to characterize the behavior of each individual task of our benchmarks, while we alter the P-State of our processor. During every run, we keep the same P-State in the whole benchmark execution and we record the energy consumption of our processor. As we note before, we exclude the memory modules due to their low energy variation. The P-States that we explore are all the 19 available frequencies of the processor (1.6 - 3.61 GHz). Their equivalent processor voltages are fixed and can be determined by datasheet inspection. Figure 5.7 presents the obtained results. We present one sub-figure for each benchmark. In each of those figures we feature the energy (dashed line) and the performance (continuous line) of each task across the whole P-State range. We

note multiple tasks with small identification numbers. On each P-State, we calculate the equivalent energy and execution time of each task by taking the mean values of their task instances. To maximize our precision, we conduct the same experiment ten times. Before the exploration of the obtained results, the reader might want to read again the cache hierarchy characteristics of our processor and the task footprints of our benchmarks.

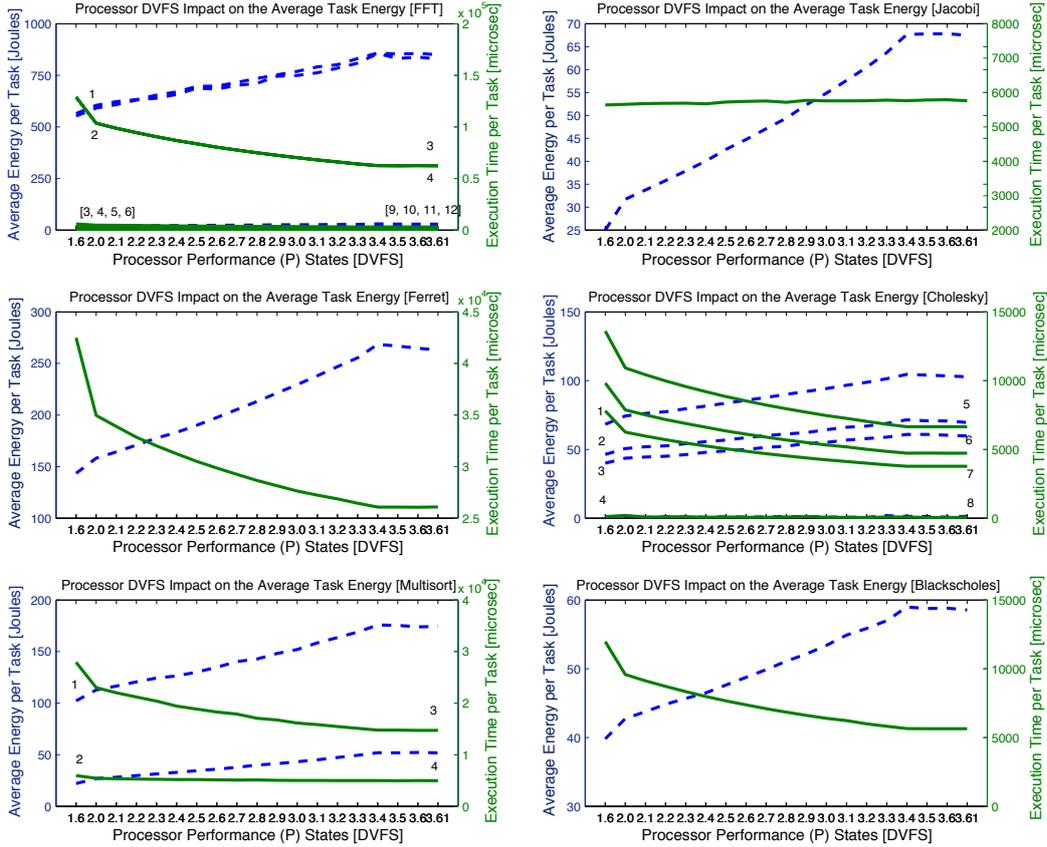


FIGURE 5.7: DVFS response of energy (dashed) and execution time (continuous) for six task-based benchmarks that we profile with TPROF.

The first sub-figure presents the DVFS response of FFT. Dashed lines 1 and 2 among with continuous lines 7 and 8 represent the energy and the performance of the two similar and dominant FFT tasks respectively. As we decrease the P-State frequency for those tasks, from maximum to minimum, we observe a 16% increase in execution time while the energy consumption is reduced by 34% percent.

The second sub-figure presents the DVFS response of Jacobi. The energy consumption as we lower the P-State decreases by up to 63%. Even more interesting is the fact that the execution time remains at the same levels regardless the processor P-State frequency. We can explain this behavior in three architectural ways. The first is a memory access pattern that creates a high amount of off-chip memory

accesses. The second is an access pattern that stresses the last level cache which has also a great access time compared to the L2 or the L1 cache. The third is the task footprint which is 32.8 MB per task, a size that far exceeds the capacity of the L1, L2 and the L3 (shared) cache. All those three scenarios create a high number of pipeline stalled cycles. This is the reason why the decreased core frequency has no impact on execution time, as the maximum benchmark execution rate cannot exceed the slower memory bandwidth.

The third sub-figure presents the DVFS response of Ferret. The behavior of this benchmark features 45% decrease of the task energy consumption along with 63% increase of execution time.

Sub-figure four presents the DVFS behavior of Cholesky. Cholesky's behavior is similar to FFT's. The three non zero energy consuming tasks yield 30%, 29% and 27% less energy consumption and 104%, 101% and 97% more execution time respectively comparing the highest with the lowest frequency.

Sub-figure five presents one of the most interesting benchmarks, Multisort. Multisort has 2 tasks (Sort and Merge). Sort, noted with numbers 1 and 3 (energy consumption and execution time) is computation intensive. Sort's energy consumption decreases by 43%, while its execution time increases by 89% comparing the highest with the lowest frequency. On the other hand, the Merge task is memory intensive with a footprint of 12.3MB. This results in a behavior similar to Jacobi's, where there is no increase in execution time when the P-state frequency lowers. In this case, we observe 60% reduction in the energy consumption with only 18% increase in the execution time. This benchmark is one of the main indications that per-core DVFS is necessary. Heterogeneous task behaviors found in workloads such as Multisort will benefit even more by a hypothetical system such as the one described in [21]. This can become a motivation for a per-core DVFS task scheduler that will co-operate with TPROF task profiler to intelligently choose the right DVFS state for each task according to each characteristics.

In the last sub-figure, we present the DVFS response of Black-Scholes. This benchmark is also purely compute intensive with very small task footprint. For Black-Scholes the energy consumption decreases by 32% and the execution time increases by 112% when lowering the P-state frequency from maximum to minimum.

We examine several cases and conclude that:

1. For compute intensive tasks (or tasks with a good locality and small task footprint that fits the caches), as we decrease the processor P-State frequency, we have an evident trade-off to decide for. The energy requirements will drop but the execution time will increase with a rate greater than the rate of the energy consumption reduction.
2. For memory intensive tasks where the core frequency suffers a mismatch with the memory speed, lowering the frequency has no significant impact on the task performance and the performance loss is lower than the energy gains.

3. The fact that an application can become more energy efficient if run on a lower P-State frequency does not necessarily result in more energy efficient system. Taking in account the whole system (memory and I/O where no energy reduction techniques are available), there are cases where the increase in the execution time leads to greater system energy consumption due to leakage. Thus, the operating frequency should be carefully chosen.

Chapter 6

Related Work

PowerPack [16] is a suite of hardware and software tools for component-level power instrumentation of HPC clusters. PowerPack uses external power sensors and post-mortem off-line analysis of traces to derive a breakdown of power consumption between hardware components. TPROF, similarly to PowerPack, uses multiple sensors, to derive component-level breakdowns of energy between processors and memory. However, TPROF performs energy accounting using a combination of on-chip energy sensors and power models. Combined, these techniques achieve lower measurement overhead and finer granularity in energy accounting than the external sensors and voltage meters used by PowerPack. Fine-grain instrumentation is of essence for emerging parallel applications that expose a large number of short-lived parallel tasks to utilize systems with many cores. Furthermore, TPROF's energy instrumentation methodology avoids intrusive hardware modifications.

PowerPack, similarly to TPROF attributes energy to software events. However, while PowerPack requires the user to identify the events via manual instrumentation, TPROF performs automatic instrumentation with compiler and runtime support. Energy accounting in TPROF is thus integrated with the parallel programming language to minimize overhead and allow for dynamic optimization. Prior work by Springer et al. [42] has used a similar transparent library instrumentation approach for measuring energy between MPI call sites. TPROF provides a more comprehensive approach, suitable for energy accounting on multi-core processors running task parallel applications.

The regression-based power modeling approach used in TPROF to account energy between concurrently executing tasks follows a long line of earlier work that derives power models using input from hardware performance counters [5, 9, 10, 11, 24, 37]. Prior work establishes power models composed of linear or piecewise polynomial functions of sampled hardware event rates, using events that correlate strongly with power consumption. Such models derive power predictions within 5% or less of the actual power of multi-core processors with high design complexity [10].

Prior work on energy accounting between software components uses models to distribute the energy budgets between coarse-grain system-level abstractions, such

as virtual machines [28], kernel threads [27], or whole programs [39]. TPROF follows the paradigm of EProf [30], which accounts energy at the finer granularity of system calls and application-level library calls [31]. TPROF differs from EProf in that it accounts energy between parallel tasks exposed by the programming language and in that it considers the concurrent execution of tasks on the same processor during the accounting process. EProf differs from TPROF in the power modeling methodology as well. EProf uses a finite-state machine (FSM) abstraction of the power states for each software component identified by a call site, while TPROF attributes directly hardware energy measurements between software tasks using sensors and performance counters. TPROF thus strives for lower overhead and more accuracy in energy instrumentation.

Chapter 7

Conclusions

This paper presents TPROF, an energy profiler for OpenMP-like task-parallel programs. TPROF can accurately compute and visualize detailed breakdowns of the energy consumed by each task and can help the programmer understand where the energy is spent inside his parallel application. TPROF uses either hardware instrumentation or a processor model to create per-core energy profiles of a parallel application. The profile process is distributed as TPROF polls the required information locally for each task thus avoiding synchronization and data sharing. Furthermore TPROF is able to uncover the energy impact of unique task-parallel characteristics such as partitioned data locality, synchronization and resource interference, that otherwise would be impossible to discover with conventional tools.

In the future, we plan to develop more realistic models that fit better the energy measurements, and produce more precise traces. We plan to create a friendly user interface showing per-task profiling information to the programmer, making TPROF a useful part of the development cycle. Moreover TPROF support for runtime monitoring of consumed energy at a low overhead enables energy constraints to be programmed in the runtime. We plan to take advantage of the energy information to extend the BDDT scheduler, and optimize execution depending on energy consumption. Finally, the availability of energy information at runtime enables the extension of the programming language with constraints that will enable the programmer to customize the application, guide the runtime system by declaring energy constraints, or optimize for various energy requirements.

Bibliography

- [1] Intel, *Intel[®] and IA-32 Architectures Software Developer's Manuals Volume 3B*.
- [2] E. Ayguadé, N. Coptý, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang. The design of openmp tasks. *IEEE Trans. Parallel Distrib. Syst.*, 20(3):404–418, Mar. 2009.
- [3] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. Legion: Expressing locality and independence with logical regions. In *Supercomputing (SC)*, 2012.
- [4] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing, ICS '10*, pages 147–158, New York, NY, USA, 2010. ACM.
- [5] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing, ICS '10*, pages 147–158, New York, NY, USA, 2010. ACM.
- [6] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques, PACT '08*, pages 72–81, New York, NY, USA, 2008. ACM.
- [7] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: an efficient multithreaded runtime system. In *Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, PPOPP '95*, pages 207–216, 1995.
- [8] R. L. Bocchino, Jr., V. S. Adve, D. Dig, S. V. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian. A type and effect system for deterministic parallel java. In *Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications, OOPSLA '09*, pages 97–116, New York, NY, USA, 2009. ACM.

- [9] V. Bui, B. Norris, K. Huck, L. C. McInnes, L. Li, O. Hernandez, and B. Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance*, CBHPC '08, pages 6:1–6:11, New York, NY, USA, 2008. ACM.
- [10] G. Contreras and M. Martonosi. Power prediction for intel xscale reg; processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, pages 221 – 226, aug. 2005.
- [11] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM.
- [12] T. Do, S. Rawshdeh, and W. Shi. ptop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, 2009.
- [13] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *Parallel Architectures and Compilation Techniques, 2003. PACT 2003. Proceedings. 12th International Conference on*, pages 220–231. IEEE, 2003.
- [14] K. Fatahalian, D. R. Horn, T. J. Knight, L. Leem, M. Houston, J. Y. Park, M. Erez, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: programming the memory hierarchy. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, 2006.
- [15] B. D. Garner, S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14:189–204, 2000.
- [16] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, may 2010.
- [17] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, 2010.
- [18] C. Isci, A. Buyuktosunoglu, and M. Martonosi. Long-term workload phases: duration predictions and applications to dvfs. *Micro, IEEE*, 25(5):39 – 51, sept.-oct. 2005.

- [19] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.
- [20] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA'06: Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, 2006.
- [21] W. Kim, M. Gupta, G. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134. Ieee, 2008.
- [22] C. E. Leiserson. The cilk++ concurrency platform. *J. Supercomput.*, 51(3):244–257, Mar. 2010.
- [23] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, June 2003.
- [24] M. Y. Lim, A. Porterfield, and R. Fowler. Softpower: fine-grain power estimations using performance counters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 308–311, New York, NY, USA, 2010. ACM.
- [25] I. Manousakis and D. S. Nikolopoulos. Epc: a power instrumentation controller for embedded applications. *SIGBED Rev.*, 9(2):28–32, June 2012.
- [26] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf*, 2011.
- [27] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 403–414, New York, NY, USA, 2006. ACM.
- [28] R. Nathuji and K. Schwan. Vpm tokens: virtual machine-aware power budgeting in datacenters. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 119–128, New York, NY, USA, 2008. ACM.
- [29] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 29–42. ACM, 2012.

- [30] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.
- [31] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 153–168, New York, NY, USA, 2011. ACM.
- [32] A. Pop and A. Cohen. A stream-computing extension to openmp. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 5–14, New York, NY, USA, 2011. ACM.
- [33] L. Pouchet. Polybench: The polyhedral benchmark suite.
- [34] J. Reinders. *Intel threading building blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.
- [35] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27, Mar. 2012.
- [36] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power containers: An os facility for fine-grained power and energy management on multicore servers. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, pages 65–76. ACM, 2013.
- [37] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009.
- [38] *SMP Superscalar (SMPSs) v2.3 User's Manual*, 2010.
- [39] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: a platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM.
- [40] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, july 2011.
- [41] V. Spiliopoulos, A. Sembrant, and S. Kaxiras. Power-sleuth: A tool for investigating your program's power behavior. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 241–250. IEEE, 2012.

- [42] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing execution time in mpi programs on an energy-constrained, power-scalable cluster. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '06, pages 230–238, New York, NY, USA, 2006. ACM.
- [43] G. Tzenakis, A. Papatriantafyllou, J. Kesapides, P. Pratikakis, H. Vandierendonck, and D. S. Nikolopoulos. Bddt:: block-level dynamic dependence analysis for deterministic task-based parallelism. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPOPP '12, pages 301–302, New York, NY, USA, 2012. ACM.
- [44] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX Annual Technical Conference*, 2012.