Application-level Energy Awareness for OpenMP

Ferdinando Alessi[1], Peter Thoman[1], Giorgis Georgakoudis[2], Thomas Fahringer[1], and Dimitrios S. Nikolopoulos[2]

[1] University of Innsbruck, Austria, {petert,tf}@dps.uibk.ac.at
[2] Queen's University of Belfast, United Kingdom, {g.georgakoudis,d.nikolopoulos}@qub.ac.uk

Abstract. Power, and consequently energy, has recently attained first-class system resource status, on par with conventional metrics such as CPU time. To reduce energy consumption, many hardware- and OS-level solutions have been investigated. However, application-level information - which can provide the system with valuable insights unattainable otherwise - was only considered in a handful of cases. We introduce OpenMPE, an extension to OpenMP designed for power management. OpenMP is the de-facto standard for programming parallel shared memory systems, but does not yet provide any support for power control. Our extension exposes (i) per-region multi-objective optimization hints and (ii) application-level adaptation parameters, in order to create energy-saving opportunities for the whole system stack. We have implemented OpenMPE support in a compiler and runtime system, and empirically evaluated its performance on two architectures, mobile and desktop. Our results demonstrate the effectiveness of OpenMPE with geometric mean energy savings across 9 use cases of 15% while maintaining full quality of service.

1 Introduction

Mobile computing devices such as laptops, tablets and smartphones are becoming more widespread. The performance that these devices offer is increasing at a steady pace, with octa-core processors powering contemporary smartphones. Mobile systems are even being considered as low-cost energy-efficient candidates for HPC [19]. However, mobility comes at a price: energy is a scarce resource on these devices, especially with power-hungry media consumption constituting a major use case. Furthermore, mobile computing is not the sole field where energy is increasingly relevant - the tremendous increase in power consumption by performance-oriented servers has made power budgeting unavoidable in HPC as well.

Several solutions have been proposed to address the energy problem on different levels. On the hardware level, energy-driven circuit design and features such as multiple available frequency and voltage scaling levels (DVFS) are widely

employed. Proposed energy-aware OSes exploit these operating modes according to actual and predicted device load, in some cases with additional knowledge provided by the application itself [22]. On the user level, frameworks have been introduced to implement content adaptation policies based on resource availability [2] and more generally to define a power management strategy steering multi-mode operating devices [14]. While system-level approaches have been explored to a certain extent, we believe there still exists a lot of room for improvement on the application level since no power management interface that can be considered complete, generic and easy to use has been proposed so far.

To fill this gap we propose a dedicated API for application-level energy awareness – and, more generally, multi-objective optimization. Rather than designing a completely new interface, we opt for extending OpenMP API [16], the de-facto standard for parallel shared-memory computing. Our choice is driven by the inherent parallelism of modern architectures, which exposes opportunities to save energy requiring close interaction with the parallel runtime system.

This paper explores application-level energy saving opportunities through the specification, implementation and evaluation of OpenMPE – an OpenMP extension for Energy. OpenMPE adds new directives and clauses to enable per-region customization of multiple optimization objectives and tunable parameters. We provide three major contributions:

- A novel API for application-level energy-aware programming, allowing programmers to expose energy saving opportunities through i) characterizing application behavior by providing a semantic region structure, ii) setting per-code region multi-objective goals and constraints, and iii) exposing application-level tunable parameters.
- A compilation and runtime system supporting OpenMPE. The runtime system exploits opportunities exposed by programmers through several techniques such as dynamic frequency and voltage scaling (DVFS), dynamic concurrency throttling (DCT), and application-level content adaptation.
- An empirical evaluation of the effectiveness of OpenMPE. A video codec reference implementation is enriched with OpenMPE and benchmarked on a desktop and mobile platform.

The rest of this paper is organized as follows: Section 2 motivates the proposed research work. Section 3 introduces OpenMPE, and Section 4 presents a compiler and runtime prototype implementation. To demonstrate the validity of our extension, experimental results are shown in Section 5. Related work and our conclusions are presented in Section 6 and 7, respectively.

2 Motivation

In this section we illustrate the three main considerations that motivate our proposed OpenMPE extensions.

Firstly, single-objective optimization is no longer sufficient. While execution time was the main and only concern in the past, with the advent of mobile platforms and the currently unsustainable energy consumption of supercomputers,

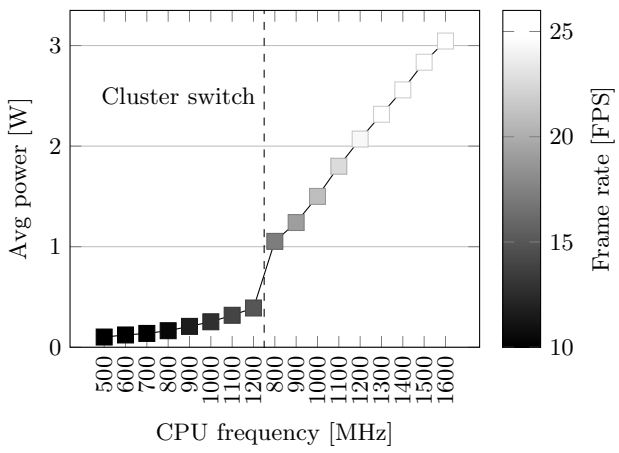


Fig. 1: Average power consumption of a video decoder at various DVFS levels

new objectives must be considered as well. With OpenMPE, energy and power are also raised to first-class resources, forming a total of four, potentially conflicting, goals with the inclusion of quality of service. Furthermore, our generic design is extendable for any additional objectives which might become relevant in the future.

A second observation concerns phase detection, which is still generally hard without application-level support. Different program regions have different requirements and applying the same optimization strategies without distinction can deteriorate optimization goals such as execution time and energy use. Consequently, correctly identifying program phases is a main concern for dynamically adaptable systems. Many solutions have been proposed so far to detect and predict program behavior automatically, both online and offline, yet current techniques still suffer from overhead and misclassification [4]. Therefore, we believe that allowing developers to easily expose program phases directly at the application level is the most practical approach.

Thirdly and most crucially, *application semantics are not derivable by underlying layers*. Functional constraints and goals are only known by the application programmer. For example, in soft real-time use cases, the time constraints are unknown to underlying layers unless explicitly communicated. As such, developers have access to important information that is unattainable by runtime and operating systems: in this paper we demonstrate how beneficial it can be to forward such application-level knowledge to underlying layers.

As a concrete example of this issue, Figure 1 illustrates the average power consumption of a video decoder executed at multiple CPU frequencies. This data was collected on a mobile development board (detailed in Section 5) where two heterogeneous CPU clusters coexist. As the frequency is lowered, the average power decreases but the same occurs for the frame rate achieved by the application: only the performance-oriented cluster can maintain an optimal frame rate,

Table 1: OpenMPE constructs and clauses

Constructs
// Definition of an explicit region
#pragma omp region [**objective**(...)] [**param**(...)]
structured-block
Clauses
// Specification of a multi−objective optimization goal
objective(*weights* : *constraints*)
// Specification of a tunable parameter
param(*var*, [**range**(*value-range*:*quality-range*) | **enum**(*values, size*:*quality-range*)])

down to a minimum of 1100 MHz. Clearly, this frame rate threshold should be taken into account by DVFS algorithms, but the desired framerate is semantic information only known at the application level.

As a side rationale, ease of use cannot be disregarded while designing an application-level interface. Our choice for a minimal directive-based API extension is intended to address this point.

3 OpenMPE

OpenMPE is based on OpenMP 4 and extends it for multi-objective optimization. Preserving the execution and memory models, directives and API functions defined in the base OpenMP language, OpenMPE adds one new construct and two clauses as listed in Table 1. Both clauses may annotate the `parallel`, `for`, `task` and `region` constructs.

Explicit regions Since OpenMP addresses parallel computing, it allows marking code regions for parallelization, worksharing and synchronization. While adding clauses to existing constructs would be sufficient to achieve our goals for such regions, for completeness we also offer the possibility to delineate code regions independently of parallelization purposes, by introducing the `region` construct. The `region` construct defines a region encompassing the subsequent single-entry-single-exit language block. This construct supports both OpenMPE clauses, and its syntax is specified in Table 1.

Multi-objective optimization goals By means of this clause, programmers can instruct the system about multi-objective optimization goals in terms of execution time, power, energy and quality of service for a specific code region: the compiler and runtime system are guided by the `objective` clause in their transformation and resource allocation policies.

Objectives can be expressed through a set of weights or constraints with the following syntax:

$$weights = f_1 * P_1 + f_2 * P_2 + \cdots + f_N * P_N$$
$$constraints = \{P_i < c_i; constraints\} \mid \emptyset$$

where $c_i \in \mathbb{R}$, $f_i \in \mathbb{R}$ and $\sum_{i=1}^{i<=N} f_i = 1.0$. P_i can be any of the non-functional parameters T (execution time), P (power consumption), E (task energy) or Q (quality of service) and the unit for c_i is, respectively, seconds, watts, joules, or a pure integer value. While estimating performance in terms of execution time, power and energy consumption is straight-forward, quality of service requires a specific definition. We define the quality of service delivered by a code region indirectly by its degradation, as an integer value between 0 and ∞ where 0 is the best achievable.

Some usage examples of the `objective` clause are:

```
1  #pragma omp ... objective(E)
2  #pragma omp ... objective(0.8*E+0.2*T)
3  #pragma omp ... objective(T : P<p)
```

On line 1 the programmer instructs the OpenMPE system to attempt minimizing the task energy required for executing the binding region. Instructed with line 2, the system performs a weighted optimization between energy and time, with 0.8 as the weighting factor for energy and 0.2 for time. Processing line 3, the system tries to achieve the minimum run time possible while staying below a given power consumption p (specified in watts in double precision floating point – p can be a dynamically evaluated expression).

Tunable application parameters The `param` clause specifies tunable parameters affecting the behavior of the program which are not introduced by the compiler or hardware platform but are inherent in user code. Its syntax specifies a base language variable storing the tunable parameter and either i) `range` specified as a lower bound expression, upper bound expression and step, all three expressions of the same type as the variable, ii) `enum` specified as a base language array storing elements of the same type as the variable and the size of the array, or iii) if the variable type is boolean, no `range` or `enum` expressions need to be specified. Each of these specifications can optionally be enriched with an expression defining the mapping between possible values for the variable and quality-of-service ratings, evaluated by the system to achieve Q objectives.

Some usage examples of the `param` clause are:

```
1  #pragma omp ... param(rate, range(24, 74, 10))
2  #pragma omp ... param(rate, range(24, 74, 10: 5, 0, 1))
3  #pragma omp ... param(name, enum(names_array, names_len)
```

The example on line 1 specifies that the base language variable `rate` can assume for the binding code region any value val which satisfies $val = 24 + i * 10; val <= 74$ where $i \in N$. Line 2 enriches the semantics of line 1 with a mapping to quality weights: a `rate` of 24 has an associated quality metric of 5, 34 maps to quality 4 and so on up to value 74 with quality 0 (the best possible). With line 3, the value of the variable `name` for the binding code region is picked by the OpenMPE runtime system among the first `names_len` elements of the array `names_array`.

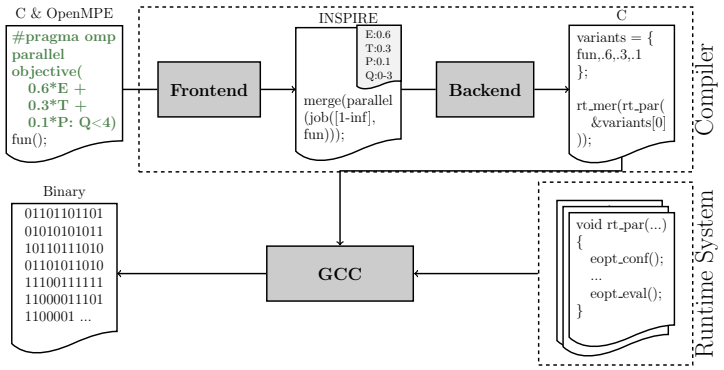


Fig. 2: OpenMPE compilation workflow

4 Compilation and Runtime System

This section outlines a reference implementation of OpenMPE, built upon the *Insieme* project [12]. It comprises two central components, the *Insieme Compiler* and the *Insieme Runtime System* (Insieme RS). To implement OpenMPE, these were modified by extending (1) the compiler frontend to process OpenMPE clauses and directives, (2) both the Insieme internal representation and encoding of meta-information statically collected by the compiler to reflect OpenMPE semantics, (3) the compiler backend to forward OpenMPE meta-information to the runtime system, (4) the runtime system framework to acquire OpenMPE information and achieve defined multi-objective goals through an optimization algorithm, and (5) the runtime system instrumentation facilities according to the requirements of the optimization algorithm.

Source-to-source compilation The Insieme compiler is a source-to-source compiler for C/C++ which supports OpenMP. Source code is translated into an internal representation (IR), optimizations are performed, and it is converted back to C. Figure 2 depicts the compilation process.

To integrate OpenMPE directives into Insieme, we directly translate `region` and `param` directives into IR constructs, while the `objective` clause is handled as meta-information annotating IR nodes. The compiler backend was extended to convert this representation into suitable Insieme RS calls and meta-information into appropriate C data structures.

Runtime system Insieme RS is an execution framework complementing the Insieme Compiler. Its application model is based on low-overhead user-level task processing and scheduling enriched by the availability of a large set of meta-information. It features a powerful instrumentation infrastructure capable of collecting per-region performance data from a variety of architectures and APIs, including PAPI, Intel RAPL – taking care of potential overflow issues – and an interface specifically designed for the mobile system described in Section 5.

`iteration`	iteration counter
`done`	true if a definitive configuration has been identified
`best_conf`	best configuration found so far
`best_perf_data`	performance data for the best configuration
`curr_conf`	current configuration
`THRESHOLD`	iterations before hill climbing

```
1:  if !done then
2:      if iteration++ < THRESHOLD then
3:          curr_conf = random_selection()
4:      else
5:          (curr_conf, done) = hill_climbing()
6:      end if
7:  else
8:      curr_conf = best_conf
9:  end if
10: annotated_task()
11: if !done and get_curr_perf_data() > best_perf_data
            and constraints_are_satisfied(get_curr_perf_data()) then
12:     best_perf_data = get_curr_perf_data()
13:     best_conf = curr_conf
14: end if
```

Fig. 3: The *e-optimizer* algorithm

Within the scope of OpenMPE, the runtime system is also responsible for achieving per-region multi-objective goals specified by programmers. An additional module, the *e-optimizer*, was introduced for this purpose. Its task is the (re)configuration of available optimization knobs based on collected performance data to fulfill defined per-region objectives. In this implementation, we consider as optimization knobs system-level features such as DVFS and DCT as well as OpenMPE tunable parameters exposed via the `param` clause.

The *e-optimizer*, outlined in Figure 3, is an online optimizer, exploiting the fact that `objective` enriched code regions are typically executed several times. For one, the `objective` clause augments constructs such as `for` which semantically require multiple executions. Moreover, many applications iterate over those constructs, as is the case for the benchmark described in Section 5. This structure provides an opportunity to evaluate different configurations for the same task in a single program execution. To this end, *e-optimizer* configuration and evaluation function calls are inserted at the beginning and end of each code region representing a task annotated with multi-objective goals. During the configuration phase, a *configuration* of optimization knobs is selected, including a specific CPU frequency, number of threads to employ and a set of values for the possible `param` variables. During the evaluation phase, performance data is collected and stored for the previous configuration and region. After a given task has been executed a threshold number of times, the *e-optimizer* picks the best configuration fulfilling the defined goals among the ones evaluated. This configuration is subsequently refined by a multi-dimensional hill climbing over all optimization knobs until a definitive solution is found. Configurations are initially selected randomly for two main reasons: (a) it is not trivial to determine how different values for a specific knob will affect energy consumption (reducing core frequency will reduce power consumption but execution time will increase) and (b) to mitigate the issue of local optima which might occur in a pure hill-climbing approach.

```
1  #pragma omp parallel for schedule(dynamic)
2  for (int y=0; y<rows; y+=2)
3      for (int x=0; x<cols_2; x++) { ... }
```

Listing 1.1: *tmndec* main loop parallelized using a dynamic schedule

```
1  #pragma omp parallel for schedule(dynamic)
       objective(E : T<1/f_rate; Q<3) param(scaling, range(1:8:1))
2  for (int y=0; y<rows; y+=2*scaling)
3      for (int x=0; x<cols_2; x+=scaling) {
4          ...
5          if(scaling > 1) { ... }
6      }
```

Listing 1.2: *tmndec* with multi-objective goals and tunable parameter

5 Evaluation

To demonstrate the effectiveness of OpenMPE, we have annotated an existing real-world application with our proposed API. We subsequently employed our reference OpenMPE compiler and runtime system, and analyzed the energy consumption of the resulting program on two distinct hardware architectures. The results obtained with our implementation are compared to the same application parallelized with plain OpenMP and compiled by GCC.

Hardware setup For our experiments we use systems representative of two device classes, mobile and desktop. The mobile system is an ODROID XU+E developer board based on a Samsung Exynos 5 Octa (5420) SoC, implementing the ARM big.LITTLE architecture comprising a Cortex-A15 quad-core and a Cortex-A7 quad-core. Either one of the clusters can be active at a time and both offer DVFS, with 9 frequencies available for the Cortex-A15 and 8 for the Cortex-A7. The board is equipped with current and voltage sensors to individually measure power consumptions of both core clusters, memory and gpu.

The desktop system is an Intel i7-3770k Ivy Bridge quad-core offering 16 frequency settings. For this system, energy estimations are collected from the Intel RAPL interface. In terms of software infrastructure, the Exynos board runs Linux kernel version 3.4.75, while the Ivy Bridge system uses 3.11.0. GCC 4.8.3 was employed as the backend compiler and for comparison purposes on both systems.

Benchmark application To evaluate our proposal we choose an application from the benchmark suite MediaBench II [8]. Among the available options we selected *tmndec*, a video decoder based on the ITU H.263 standard. Although more recent codecs are available in the suite, their far greater complexity would add major engineering and parallelization effort to our study while not providing significant new insight. We optionally enable vertical and horizontal deposterization filters in order test our proposal on additional load scenarios.

Since *tmndec* is a purely sequential implementation, as a first step we parallelize it via OpenMP. The application features a central two-level nested loop

which accounts for video frame decoding: thus, it was a prime target for optimization (Listing 1.1).

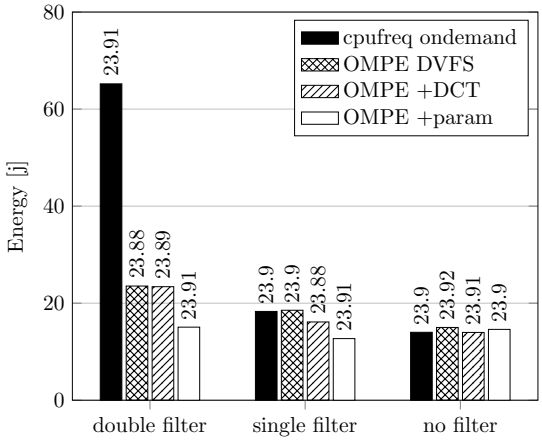
Semantically, this code region needs to be executed sufficiently fast such that the application can still achieve its target frame rate. A constraint of this type is easily expressible through OpenMPE as shown in Listing 1.2. The *weights* expression of the `objective` hints at a minimization of energy consumption without regard for power or time, while the *constraints* expression guarantees that a specific frame rate *f_rate* is maintained.

Finally, we introduce content-adaptation by the OpenMPE `param` clause and a constraint on the quality of service. With the addition of the subsampling factor *scaling* it is possible to adjust the resolution – and thus quality – of the decoded video. As shown in Listing 1.2, the `param` clause indicates that the variable *scaling* can assume any integral value in range [1,8] at runtime. Subsampling is enabled accordingly on line 5. The addition of the quality constraint (Q<3) prevents the optimizer from choosing scaling factors which significantly degrade quality – a user-adjustible variable could be employed in real-world scenarios.

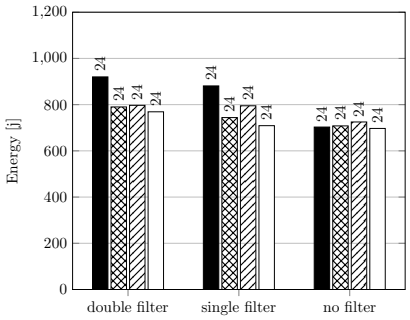
Experimental results We performed experiments using two resolutions, 704x576 (4CIF) and 1408x1152 (16CIF), and for each of them we used three load variants: i) full (horizontal and vertical) deposterization, ii) a single vertical deposterization pass, and iii) no filter at all. On the *mobile* system only the 4CIF resolution was tested, while both were explored on the *desktop* system for a total of 6 configurations. For comparison purposes, the **ondemand** *cpufreq* governor [17], default for most Linux systems, was also evaluated for each configuration.

The *e-optimizer* search phase is generally very short (around 20 frames, less than a second) and will not significantly impact performance in production scenarios. However, for our testing, the insufficient temporal resolution of the energy measurement hardware provided by our *mobile* system required us to evaluate the otherwise single execution of each OpenMPE code region in groups, with a resulting expansion of the *e-optimizer* search time. A longer, realistic video playback scenario would still mitigate this initial loss of performance but would also dramatically increase the experiment duration. For this reason, the data collected from our experiments and shown in Figure 4 is limited to the final 3000 frames of a 12000 frames video.

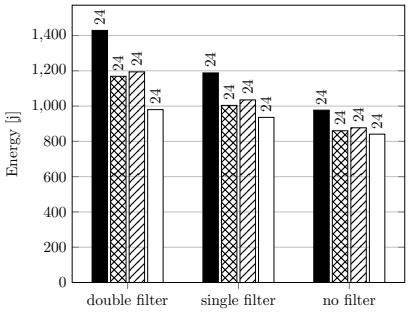
When discussing the results presented in Figure 4, we would first like to note that despite the energy savings it achieves, our system maintains a quality of service (FPS, as indicated by the numbers on each bar in the chart) on par with the reference governor in all scenarios. Some observations can be made across both platforms and are related to the features of this particular benchmark application: (1) allowing the *e-optimizer* to use DCT capabilities (`+DCT`) does not add to energy gains for this testing scenario. This is due to the fact that both systems provide an ample variety of DFVS frequencies, and that the algorithm features nearly linear parallel scaling. (2) the addition of content-adaptation (`+param`) is not effective when no filters are applied, as in such cases the overall computational load is very small. (3) the geometric mean of the energy savings



(a) *Mobile* platform, 704x576 resolution



(b) *Desktop* platform, 704x576



(c) *Desktop* platform, 1408x1152

Fig. 4: Energy consumption of *tmndec* with different optimization knobs and filtering applied; frame rate achieved noted on top of each bar

achieved by our system using all of its capabilities compared to the baseline, across all 9 scenarios, is 15%.

On the *mobile* system we observe a large range of findings from an energy point of view, with our system performing significantly better than the **ondemand** governor for the full filter configuration, saving up to 77% energy, while managing about 20% energy savings in the single filter case and performing on par with no filtering. This range of effectiveness is related to the relative CPU load incurred in the various scenarios: in the very low-load no-filtering scenario, the default governor is able to determine that the lowest CPU frequency is sufficient because of the long sleep periods of the application, bringing its performance up to par with our approach.

On the *desktop* system achieving an optimal frame rate is not an issue and our implementation generally performs better than the *cpufreq* governor with

energy savings up to 31%. Once again, the only exception is the 704x576 resolution scenario with no filtering: due to the comparatively light computational load of this configuration, both approaches detect that the lowest possible frequency setting is sufficient. It is interesting to note that even though the range of available frequencies is larger on the *desktop* system, they offer a smaller gain in terms of energy savings compared to the *mobile* system. This is evident from Figure 4c: the *desktop* system is capable of maintaining an optimal frame rate with each of the available frequencies, but the energy savings with DVFS, while significant, are comparatively minor remaining between 12% and 18%.

6 Related Work

Proposed solutions for energy and power management range from the lowest to the upper levels of the system stack involved: hardware-, system software- and application-based approaches have been investigated over the last decade.

At the hardware level, energy savings can be achieved through low-power circuit design [1] or providing different operational modes for a particular component [9].

At the system software level, studies range from energy-oriented operating systems [5, 15, 18], over compilers [11, 21] to runtime systems [10, 20]. Interactions between OpenMP and energy consumption have also been investigated. A pure OpenMP runtime that applies DVFS and DCT according to predicted performance of application phases has been proposed [3], and this concept was also generalized to hybrid MPI/OpenMP programming [13]. However, none of these works consider application-level knowledge for power management.

At the application level, Odyssey [6, 7] is one of the first projects to demonstrate the benefits of content adaptation for a reduced energy profile. In this work, the user provides a goal for battery duration to the operating system that, assessing the system status, informs the application about a target quality for the output. A multimedia oriented operating system is proposed with Grace OS [22]. With starting time and duration of tasks provided by applications, Grace OS determines a system-wide CPU voltage and frequency. A more generic approach is evaluated within *Anole* [2]. This proposed framework updates the application about the current energy status of the device and the application can then adapt its behavior in an arbitrary manner. Notifications about energy availability are forwarded to the operating system as well: hardware and service adaptation can then be offered through ad-hoc modules. A different triggering strategy characterizes the *Chameleon* interface [14]. A compliant application does not react to energy events, rather it can monitor processor load and set a desired speed.

Even though some of these studies propose power management interfaces with application-level involvement, they differ substantially from our work. Previous work focuses on dynamic objectives and overall system load, while we opt for highlighting per-application-region static energy saving opportunities to the underlying levels of the system. The OpenMPE API, compared to prior work, is:

– less intrusive and easier to integrate, since we provide a minimal directive-based interface,
– more expressive, since it is possible to directly specify energy constraints (power budgeting) and arbitrary tunable parameters (content adaptation),
– more generic, as user-defined multi-objective goal functions and constraints across an existing, easily extensible set of four metrics are supported,
– more flexible, since the OpenMPE specification does not prescribe or restrict the optimization techniques applied by the runtime system, and
– up-to-date, since we inherently deal with parallel codes in the parallel architecture era by basing our approach on OpenMP.

7 Conclusion and Future Work

This paper describes an extension to OpenMP, OpenMPE, which provides two novel features: *multi-objective goals and constraints* and *application adaptation.* These features allow our interface to address the issue of energy consumption and power budgeting, fundamental on modern mobile and HPC systems. Application programmers know the non-functional requirements and adaptation opportunities of each code region, and with OpenMPE they can conveniently provide this knowledge to all underlying layers. We have developed a compiler and associated runtime system for OpenMPE which are able to perform system- and program-level adjustments to achieve specified multi-objective goals while respecting given constraints. Experimental results demonstrate energy savings up to 77% are feasible with this prototype implementation.

In the future, more extensive evaluation of the OpenMPE API would be desirable, targeting several applications. Furthermore, analysis and refinement of the system's interaction with external load is an important goal.

Acknowledgments

This research has been partially funded by the FWF Austrian Science Fund under contract I01079 (GEMSCLAIM).

References

[1] AP. Chandrakasan et al. "Low-power CMOS digital design". *Solid-State Circuits, IEEE Journal of* 27.4 (1992), pp. 473–484.

[2] Hui Chen et al. "Anole: A Case for Energy-Aware Mobile Application Design". *Parallel Processing Workshops (ICPPW), 2012 41st Int. Conf. on.* 2012, pp. 232–238.

[3] Matthew Curtis-Maury et al. "Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores". *Proc. of the 17th Int. Conf. on Parallel Architectures and Compilation Techniques.* PACT '08. Toronto, Ontario, Canada: ACM, 2008, pp. 250–259.

[4] Ashutosh S. Dhodapkar and James E. Smith. "Comparing Program Phase Detection Techniques". *Proc. of the 36th IEEE/ACM Int. Symp. on Microarchitecture.* MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003.

[5] Krisztián Flautner et al. "Automatic Performance Setting for Dynamic Voltage Scaling". *Wirel. Netw.* 8.5 (09/2002), pp. 507–520.

[6] Jason Flinn and M. Satyanarayanan. "Energy-aware Adaptation for Mobile Applications". *Proc. of the Seventeenth ACM Symp. on Operating Systems Principles*. SOSP '99. Charleston, South Carolina, USA: ACM, 1999, pp. 48–63.

[7] Jason Flinn and M. Satyanarayanan. "Managing Battery Lifetime with Energy-aware Adaptation". *ACM Trans. Comput. Syst.* 22.2 (05/2004), pp. 137–179.

[8] Jason E. Fritts et al. "MediaBench II Video: Expediting the Next Generation of Video Systems Research". *Microprocess. Microsyst.* 33.4 (06/2009), pp. 301–318.

[9] Hewlett-Packard Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., Toshiba Corp. *Advanced Configuration and Power Interface Specification (ACPI)*. Specification Revision 5.0. 2013.

[10] Chung-hsing Hsu and Wu-chun Feng. "A Power-Aware Run-Time System for High-Performance Computing". *Proc. of the 2005 ACM/IEEE Conf. on Supercomputing*. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1–.

[11] Chung-Hsing Hsu and Ulrich Kremer. "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction". *Proc. of the ACM SIGPLAN 2003 Conf. on Programming Language Design and Implementation*. PLDI '03. San Diego, California, USA: ACM, 2003, pp. 38–48.

[12] Herbert Jordan et al. "A Multi-objective Auto-tuning Framework for Parallel Codes". *Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012, 10:1–10:12.

[13] Dong Li et al. "Hybrid MPI/OpenMP power-aware computing". *Parallel Distributed Processing (IPDPS), 2010 IEEE Int. Symp. on*. 2010, pp. 1–12.

[14] Xiaotao Liu et al. "Chameleon: Application-Level Power Management". *Mobile Computing, IEEE Transactions on* 7.8 (2008), pp. 995–1010.

[15] Jacob R. Lorch and Alan Jay Smith. "Operating System Modifications for Task-Based Speed and Voltage". *Proc. of the 1st Int. Conf. on Mobile Systems, Applications and Services*. MobiSys '03. San Francisco, California: ACM, 2003, pp. 215–229.

[16] OpenMP Architecture Review Board. *OpenMP Application Program Interface*. Specification Version 4.0. 2013.

[17] Venkatesh Pallipadi and Alexey Starikovskiy. "The ondemand governor". *Proc. of the Linux Symp.* Vol. 2. sn. 2006, pp. 215–230.

[18] N. Pettis et al. "Automatic Run-Time Selection of Power Policies for Operating Systems". *Design, Automation and Test in Europe, 2006. DATE '06. Proc.* Vol. 1. 2006, pp. 1–6.

[19] Nikola Rajovic et al. "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" *Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 2013, 40:1–40:12.

[20] Barry Rountree et al. "Adagio: Making DVS Practical for Complex HPC Applications". *Proc. of the 23rd Int. Conf. on Supercomputing*. ICS '09. Yorktown Heights, NY, USA: ACM, 2009, pp. 460–469.

[21] Qiang Wu et al. "Dynamic-Compiler-Driven Control for Microprocessor Energy and Performance". *IEEE Micro* 26.1 (01/2006), pp. 119–129.

[22] Wanghong Yuan and Klara Nahrstedt. "Practical Voltage Scaling for Mobile Multimedia Devices". *Proc. of the 12th ACM Int. Conf. on Multimedia*. MULTIMEDIA '04. New York, NY, USA: ACM, 2004, pp. 924–931.