International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
2019, Vol. 33(3) 431–443
© The Author(s) 2018
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1094342018792079
journals.sagepub.com/home/hpc
SAGE

Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms

João Vicente Ferreira Lima[1], Issam Raïs[2], Laurent Lefèvre[2] and Thierry Gautier[2]

Abstract
In this article, we analyze performance and energy consumption of five OpenMP runtime systems over a non-uniform memory access (NUMA) platform. We also selected three CPU-level optimizations or techniques to evaluate their impact on the runtime systems: processors features Turbo Boost and C-States, and CPU Dynamic Voltage and Frequency Scaling through Linux CPUFreq governors. We present an experimental study to characterize OpenMP runtime systems on the three main kernels in dense linear algebra algorithms (Cholesky, LU, and QR) in terms of performance and energy consumption. Our experimental results suggest that OpenMP runtime systems can be considered as a new energy leverage, and Turbo Boost, as well as C-States, impacted significantly performance and energy. CPUFreq governors had more impact with Turbo Boost disabled, since both optimizations reduced performance due to CPU thermal limits. An LU factorization with concurrent-write extension from libKOMP achieved up to 63% of performance gain and 29% of energy decrease over original PLASMA algorithm using GNU C compiler (GCC) libGOMP runtime.

Keywords
OpenMP, task parallelism, linear algebra algorithms, NUMA, energy efficiency

1. Introduction

Energy efficiency is one of the five major challenges that should be overcome in the path to exascale computing (Bergman et al., 2008). Despite improvements in energy efficiency, the total energy consumed by supercomputers is still increasing due to the even quicker increase in computational power. High energy consumption is not only a problem of electricity costs but also impacts greenhouse emissions and dissipating the produced heat can be difficult. As the ability to track power consumption becomes more commonplace, with some job schedulers supporting tracking energy use (Yang et al., 2013), soon users of high performance computing (HPC) systems may have to consider both how many CPU hours they need and how much energy.

Energy budget limitation imposes a high pressure to the HPC community making energy consideration a prominent research field. Most of the gain will come from technology by providing more energy-efficient hardware, memory, and interconnect. Nevertheless, recent processors integrate more and more leverages to reduce energy consumption (e.g. classical Dynamic Voltage and Frequency Scaling (DVFS) and deep sleep states) and low-level runtime algorithms provide orthogonal leverages (e.g. dynamic concurrency throttling (DCT)). However, few of these leverages are integrated and employed in today's local-level software stack such as middleware, operating system, or runtime library. Due to the complexity of this statement, we restricted our investigation to local node energy consumption by HPC OpenMP applications.

OpenMP is an application programming interface (API) standard to express parallel portable programs. Most of the controls are implementation defined and rely on the specific OpenMP programming environment used. The OpenMP standard does not impose any constraint on implementations. Even if there are more precise specifications, for example, mapping of threads to cores, it is very tricky to precisely control performance or energy consumption using what OpenMP specification proposes (Bari et al., 2016).

[1] Universidade Federal de Santa Maria, Santa Maria, Rio Grande do Sul, Brazil
[2] Université de Lyon, INRIA, CNRS, ENS de Lyon, Université Claude-Bernard Lyon 1, LIP, Lyon, France

Corresponding author:
João Vicente Ferreira Lima, Centro de Tecnologia, Universidade Federal de Santa Maria, Prédio 07, Anexo B – Sala 374, Avenida Roraima 1000, Santa Maria, Rio Grande do Sul 97105900, Brazil.
Email: jvlima@inf.ufsm.br

Previous works have dealt with a specific OpenMP runtime (Li et al., 2010; Lively et al., 2011; Marathe et al., 2015; Nandamuri et al., 2014; Porterfield et al., 2013; Su et al., 2012) that may be difficult to generalize to other OpenMP runtime systems without strong development effort. To the knowledge of the authors, there is no related work comparing OpenMP runtime systems in order to analyze performance and energy consumption.

In this article, we analyzed performance and energy consumption of five OpenMP runtime systems over a non-uniform memory access (NUMA) system. We also selected three CPU-level optimizations or techniques to evaluate their impact on the runtime systems: processors features Turbo Boost and C-States, and CPU DVFS through Linux CPUFreq governors. We restrict our experiments on three dense linear algebra algorithms: Cholesky, LU, and QR matrix factorizations. Source codes are based on KASTORS (Virouleau et al., 2014) benchmark suite and the state-of-the-art PLASMA library using its new OpenMP implementations (YarKhan et al., 2016) that rely on OpenMP tasks with data dependencies.

The article is an extended version of the article presented at the 8th Workshop on Applications for Multi-Core Architectures (Lima et al., 2017). The contributions of this article are as follows:

- We present early experiments of performance and energy consumption over OpenMP runtime systems.
- We report the impact of three CPU-level optimizations in order to present the respective gains with different combinations.
- Small algorithmic and runtime improvements may allow significant performance gain and energy reduction on dense linear algebra algorithms. An LU factorization with concurrent-write (CW) access mode achieved up to 63% in performance gain and 29% in energy reduction over original LU algorithm using GNU C compiler (GCC) libGOMP runtime.
- In addition, our findings suggest that Turbo Boost and C-States had significant impact on performance and energy. CPUFreq governors had more impact with Turbo Boost disabled since Turbo Boost with the *performance* governor reduced performance due to CPU thermal limits.

The remainder of the article is organized as follows. Section 2 presents the related work. Section 3 gives some details of the OpenMP task programming model and an overview about five runtime implementations. Section 4 details the experimental hardware and methodology used. Our experimental results are presented in Section 5. Finally, Sections 6 and 7, respectively, present the discussion and conclude the article.

2. Related work

Multiple techniques or *leverages* dealing with the energy–performance trade-off are exposed and used in the literature.

Most research in energy efficiency is devoted to DVFS (Zhuravlev et al., 2013) which is a well-known technique to reduce energy consumption by slowing down the speed of processors. Benoit et al. (2017) demonstrate the challenge of using the shutdown and wakeup leverages for large-scale HPC infrastructure without altering the throughput of needed computation. In the work of Ribic and Liu (2014), DVFS is used to lower the speed of threads that are not in the critical path with a warranty on performance. Etinski et al. (2010) bind DVFS with EASY backfilling job scheduling to answer possible system load variation in an energy-efficient way. Considering communications between nodes, Rountree et al. (2009) reduce the frequency of tasks which would block for message passing interface (MPI) communication. Laros et al. (2012) analyze the effects of both DVFS and network bandwidth scaling.

Another commonly used technique is DCT that changes the number of computing threads at runtime. Other works use the simplicity proposed by OpenMP to vary the number of threads, for energy efficiency. Curtis Maury et al. (2006) and Porterfield et al. (2013) defend DCT and underline the fact that using OpenMP to control the number of threads could be energy efficient, depending on the algorithm or the chosen hardware. De Matteis and Mencagli (2017) use DVFS and DCT in the context data stream processing applications.

In addition to DVFS, Intel processors support duty cycle modulation (Schöne et al., 2016) that "squashes" CPU clock without changing the real frequency for each individual core. Zhang et al. (2009) use duty cycling to efficiently manage on-chip shared resources. Bhalachandra et al. (2015) employed this technique in MPI applications, while Cicotti et al. (2014) designed a library to save energy with duty cycling and DVFS. Wang et al. (2015) used DCT and duty cycling on OpenMP parallel loops.

Many of the related works made experiments based on NASA advanced computing (NAS) benchmarks (Curtis Maury et al., 2006; Freeh et al., 2007; Li et al., 2010; Marathe et al., 2015; Sundriyal et al., 2014) that are widely accepted as representative of scientific applications. Lagravière et al. (2015) investigate the MFlops/Watt metric between MPI, unified parallel C (UPC), and OpenMP. Schöne and Molka (2014) propose a framework to dynamically change configuration of hardware and software. They applied their method to reduce the energy consumption of parallel regions where stalled operation may occur.

Previous works show that various energy behaviors of computing nodes are possible through various leverages (DVFS, DCT, etc.). But none of the previous work focus on OpenMP runtime systems as a leverage. None of the previous work dealt with the energy–performance trade-off and thus underlined possible variability concerning energy and performance for existing runtime systems. Thus, to the knowledge of the authors, no related work was trying to compare several OpenMP runtime libraries together for various representative workloads, as presented in our article.

```
for (k=0; k<NB; k++) {
#pragma omp task untied shared(M) \
    depend(inout: M[k*NB+k])
  lu0(M[k*NB+k]);
  for (j=k+1; j<NB; j++)
#pragma omp task untied shared(M) \
 depend(in: M[k*NB+k]) depend(inout: M[k*NB+j])
    fwd(M[k*NB+k], M[k*NB+j]);

  for (i=k+1; i<NB; i++)
#pragma omp task untied shared(M)\
 depend(in: M[k*NB+k]) depend(inout: M[i*NB+k])
    bdiv(M[k*NB+k], M[i*NB+k]);

  for (i=k+1; i<NB; i++)
    for (j=k+1; j<NB; j++)
#pragma omp task untied shared(M)\
 depend(in:M[i*NB+k], M[k*NB+j]) depend(inout:M
      [i*NB+j])
    bmod(M[i*NB+k],M[k*NB+j],M[i*NB+j]);
}
```

Figure 1. LU factorization with OpenMP-dependent task.

We use state-of-the-art PLASMA library (YarKhan et al., 2016), on three main kernels in dense linear algebra (Cholesky, LU, and QR factorizations), that implements dependent tasks model. This model is new and never addressed in related works. Nandamuri et al. (2014) and Porterfield et al. (2013) based their experiments using the BOTS (Duran et al., 2009) benchmarks that require only the independent tasks.

3. OpenMP task programming model and implementations

In 2013, the OpenMP Architecture Review Board introduced, in the OpenMP revision 4.0, a new way of expressing task parallelism using OpenMP, through the task dependencies. This section introduces the task dependency programming model targeted by the selected benchmark suites. We also present how the model is implemented in various runtime libraries.

3.1. Dependent task model

OpenMP-dependent task model allows to define dependencies between tasks using declaration of accesses to memory with *in*, *out*, or *inout*. Two tasks are independent (or concurrent) if and only if they do not violate the data dependencies of a reference sequential execution order.[1]

Figure 1 illustrates an LU factorization based on PLASMA. The programmer declares tasks and the accesses `in, inout` they made to a memory region (here only *lvalue* or memory reference, i.e. pointer).

The OpenMP library computes tasks and dependencies at runtime and schedules concurrent tasks on the available processors. The strategy for task dependencies and task scheduling depends on the runtime implementation. Nevertheless, their implementations impact the performance and the energy consumption. Moreover, the absence of precise OpenMP specification about the task scheduling algorithm is the key point to allow research to improve performance and energy efficiency with implementation concerns.

Table 1. Characteristics of OpenMP runtime systems.

Name	Dependencies	Task scheduling	Remarks
libGOMP	Hash table	Centralized list	Task throttling
libOMP	Hash table	Work stealing	Bounded dequeue
OmpSs	Hash table	Socket-aware Work stealing	
Xkaapi	Hash table	Non-blocking Work stealing	Task affinity
libKOMP	Resizable Hash table	Non-blocking Work stealing	Task affinity Concurrent write

3.2. Runtime system implementations

Table 1 summarizes the properties of five OpenMP runtime systems.

libGOMP is the OpenMP runtime that comes with the GCC compiler. Dependencies between tasks are computed through a hash table that maps data (pointer) to the last task writing the data. Ready tasks are pushed into several scheduling dequeues. The main dequeue stores all the tasks generated by the threads of a parallel region. Tasks seem to be inserted after the position of their parent tasks in order to keep an order close to the sequential execution order. Because threads share the main dequeue, serialization of operations is guaranteed by a pthread mutex which is a bottleneck for scalability. To avoid overhead in task creation, libGOMP implements a task throttling algorithm that serializes task creation when the number of pending tasks is greater than a threshold proportional to the number of threads.

libOMP was initially the proprietary OpenMP runtime of Intel for its C, C++, and Fortran compilers. Now it is also the target runtime for the LLVM/Clang compiler and sources are open to community. libOMP manages dependencies in the same way that libGOMP by using a hash table. Memory allocation during task creation relies on a fast thread memory allocator. libOMP task scheduling is based on Cilk almost non-blocking work stealing algorithm (Frigo et al., 1998), but dequeue operations are serialized using locks. Nevertheless, it implies distributed dequeues management with high throughput of dequeue operations. libOMP also implements a task throttling algorithm by using bounded size dequeue.

OmpSs (Bueno Hedo et al., 2012) is a runtime system developed at the Barcelona Supercomputing Center (Barcelona, Catalonia, Spain), compatible with the OpenMP specification. It has a specific compiler, called Mercurium, that transforms OpenMP directives to calls to Nanos++ runtime entrypoints. As libGOMP and libOMP, OmpSs computes task dependencies at task creation using hash map. In our experiments, we select the breadth-first scheduler. In our experiments, we selected the Socket-aware scheduler (*socket*) which is a work-stealing–based task scheduler with distributed dequeues implementations.

XKaapi (Gautier et al., 2013) is a task library for multi-CPU and multi-GPU architectures which provides binary compatible library with libGOMP (Broquedis et al., 2012). Task scheduling is based on the almost non-blocking work stealing algorithm from Cilk (Frigo et al., 1998) with extension to combine steal requests in order to reduce overhead in stealing (Tchiboukdjian et al., 2013). Moreover, XKaapi computes dependencies on steal request, which is a perfect application of the work first principle to report overhead in task creation to critical path. The XKaapi-based OpenMP runtime also has support to some OpenMP extensions such as task affinity (Virouleau et al., 2016) that allows to schedule tasks on NUMA architecture and to increase performance by reducing memory transfer and thus memory energy consumption.

libKOMP (Gautier and Virouleau, 2015) is a redesign of Broquedis et al. (2012) on a top of the Intel runtime libOMP. It includes following features coming mainly from XKaapi: the dequeue management and work stealing with request combining; task affinity specific work stealing heuristic; a dynamically resized hash map that avoids high conflicts when finding dependencies for large tasks' graph; and tracing tool based on the OpenMP OMPT API; and finally, a task CW extension with a Clang modification[2] to provide the OpenMP directive clause. This latter extension allows better parallelism and it is similar to the *task reduction* feature currently under discussion in the OpenMP architecture review board.

3.3. Discussion

In our study of the mentioned OpenMP runtime systems, none of them include energy leverage such as thread throttling or DVFS. Nevertheless, their different task scheduling algorithms may impact energy efficiency. The main dequeue accesses in libGOMP serialize threads using a POSIX mutex. On Linux, the mutex will block waiting threads after short period of active polling which ensure that few core cycles will be wasted in the synchronization.

On the other hand, libOMP, XKaapi, and libKOMP work stealing actively poll dequeues until the program ends or a task is found. In order to reduce activity during polling, libOMP and libKOMP may block threads after an unsuccessful search of work by 200 ms (default value). Once work is found, all threads are woken up.

4. Tools and methods

This section details the hardware configuration we experimented on and the OpenMP runtime systems we compared. We also give hints about the methodology used to process the collected data using statistical tools R.

4.1. Evaluation platform

Our experimental platform was an SGI UV 2000 machine composed of eight NUMA nodes with one Intel Xeon E5-4617 (Sandy Bridge) processor each (total eight processors) and six cores per processor (48 cores total) running at 2.9 or 3.2 GHz with Turbo Boost and 512 GB of main memory. The processor has Turbo Boost 2.0 technology, and six idle states of C-States available: POLL C1-SNB C1E-SNB C3-SNB C6-SNB C7-SNB. The operating system was a Debian with Linux kernel 4.9.0-1 with the Intel P-State driver, which supports two CPUFreq governors: powersave and performance. Both DVFS governors have frequency limits from 1.20 GHz to 3.40 GHz.

4.2. Software description

4.2.1. Benchmarks. We used kernels from two benchmark suites: the KASTORS (Virouleau et al., 2014) benchmark suite and the OpenMP-based PLASMA benchmark. Both benchmark suites tackle the same computational problems but use different algorithms in some cases.

KASTORS was built from PLASMA version 2.6.0 (released in December 2013) that was implemented over a specific task management library called QUARK. The OpenMP-based PLASMA is a new version of PLASMA over OpenMP directives instead of QUARK library. We compare OpenMP-based PLASMA version 82f89ee.[3]

We focused our study on three dense linear algebra kernels:

- a Cholesky factorization (`dpotrf`);
- an LU factorization (`dgetrf`); and
- a QR factorization (`dgeqrf`).

All these linear algebra kernels we used rely on the BLAS routines; we used the implementation of OpenBLAS version 0.2.19. Cholesky and QR algorithms are the same in both KASTORS and PLASMA. Nonetheless, LU had three versions: KASTORS from original PLASMA version, KASTORS extended with CW access, and PLASMA from the OpenMP-based version.

4.2.2. Runtime systems. We compared the following runtime systems during our experiments:

- libGOMP: The OpenMP implementation from GNU that comes with GCC 6.3.0.
- libOMP: A port of the Intel OpenMP open-source runtime to LLVM release 4.0.
- libKOMP (Gautier and Virouleau, 2015): A research runtime system, based on the Intel OpenMP runtime, developed at INRIA. It offers several nonstandard extensions to OpenMP. We evaluate the CW feature in our experiments coupled with Cilk T.H.E work stealing protocol. We make experiments with version `54f7a28`.[4]
- XKaapi (Gautier et al., 2013): Research runtime system developed at INRIA. It has lightweight task

creation overhead, and it offers several nonstandard extensions to OpenMP (Broquedis et al., 2012). We evaluate its version `efa5fdf`.[5]

- OmpSs (Bueno Hedo et al., 2012): A runtime system developed at the Barcelona Supercomputing Center. The reported results are based on the 17.12 version.[6]

In all experiments, we set `OMP_WAIT_POLICY` environment variable to *passive*, which means that threads should not consume CPU power while waiting. OmpSs has an equivalent option to Nano++ through `--enabled-sleep`. XKaapi runtime does not implement a waiting policy.

In our experiments, we compared the runtime systems using KASTORS benchmark suite, except for PLASMA configuration that is based on the OpenMP-based version running over GCC libGOMP runtime.

4.3. Energy measurement methodology

Since several metrics have to be considered depending on the objective, we consider performance (GFlop/s) and energy consumption (energy-to-solution). GFlop/s is measured by the each benchmark itself: It corresponds to the algorithmic count of the number of floating point operations over the elapsed time, using fact that matrix–matrix product does not rely on a fast algorithm such as Strassen-like algorithm. Times are get using the Linux `clock_gettime` function with `CLOCK_REALTIME` clock.

We employed the Intel running average power limit (RAPL) feature as source of data acquisition for energy measurement. It exposes the energy consumption of several components on the chip (such as the processor package and the DRAM) through model specific registers (MSRs). Due to access limitation of MSRs on the tested system, we designed a small tool querying periodically the RAPL counters based on LIKWID (Treibig et al., 2011): energy consumption for the whole package (`PWR_PKG_ENERGY`), for the cores (`PWR_PP0_ENERGY`), for the DRAM (`PWR_DRAM_ENERGY`), as well as the core temperature (`TEMP_CORE`). The tool gets the counter values periodically and associates them with a time stamp.

4.4. Experimental methodology

All benchmarks are composed of two steps: The first allocates and initializes a matrix; the second step is the computation. We report execution time only from the computation step. Each experiment was repeated at least 30 times, each computation on a newly random matrix (as implemented by the benchmark). In parallel of the computation, we monitored the system by collecting various energy counters from RAPL.

We selected three CPU-level optimizations (Orgerie et al., 2014) in order to evaluate their impact on performance and energy over the runtime systems: processor features Turbo Boost and C-States (Rotem et al., 2012), and CPU DVFS through Linux CPUFreq governors. Intel Turbo Boost is an overclocking mechanism that allows to the processor to raise core frequencies as long as the thermal and power limits are not exceeded. The C-States feature corresponds to CPU idle states. Deeper C-States offer more power savings but at the cost of longer latency to enter and exit the C-State. On Linux, CPUFreq allows the control of P-States, which defines the frequencies at which a processor can operate, by governors that choose a frequency for the processor to use. The available governors on our experimental platform were *powersave* that chooses the lowest frequency and *performance* that chooses the highest frequency.

For each computation, we collected performance (GFlop/s) time stamped by the beginning and the end of the computation. This two time stamps were used in data post-processing to compute energy consumed by the computation between the two time stamps. Values were interpolated by linear function if missing in the collected energy values sampled periodically. Post-processing employs R script to compute energy per computation and to output basic statistic for each configuration. In our experimental results, energy values were the mean computed among the at least 30 computations of each configuration.

5. Experimental results

The goal of our experiments is to evaluate performance and energy consumption of OpenMP runtime systems and the impact of three CPU-level optimizations. Our objectives are as follows:

1. evaluate the runtime impact on performance and energy, as well as the CPU-level techniques (Section 5.1);
2. analyze the correlation coefficient of CPU optimizations over performance and energy (Section 5.2); and
3. assess the impact of runtime extensions on performance of LU (Section 5.3).

The presented runtime systems have been experimented on the two benchmark suites presented in Section 4.2.1. We build two configurations of libKOMP using two sets of options (Gautier and Virouleau, 2015). On the following, *libKOMP* refers to the runtime version configured with T.H.E Cilk work stealing queue and requests combining protocol; and *libKOMP+CW* is the same configuration than libKOMP with addition to support CW extension used in the KASTORS LU code dgetrf (Virouleau et al., 2014). In addition, *PLASMA* refers to the OpenMP-based PLASMA implementation compiled with GCC and the libGOMP runtime.

5.1. Runtime impact

Figure 2 shows performance results with a matrix size of 32,768 $\times$ 32,768 and over all machine resources available,

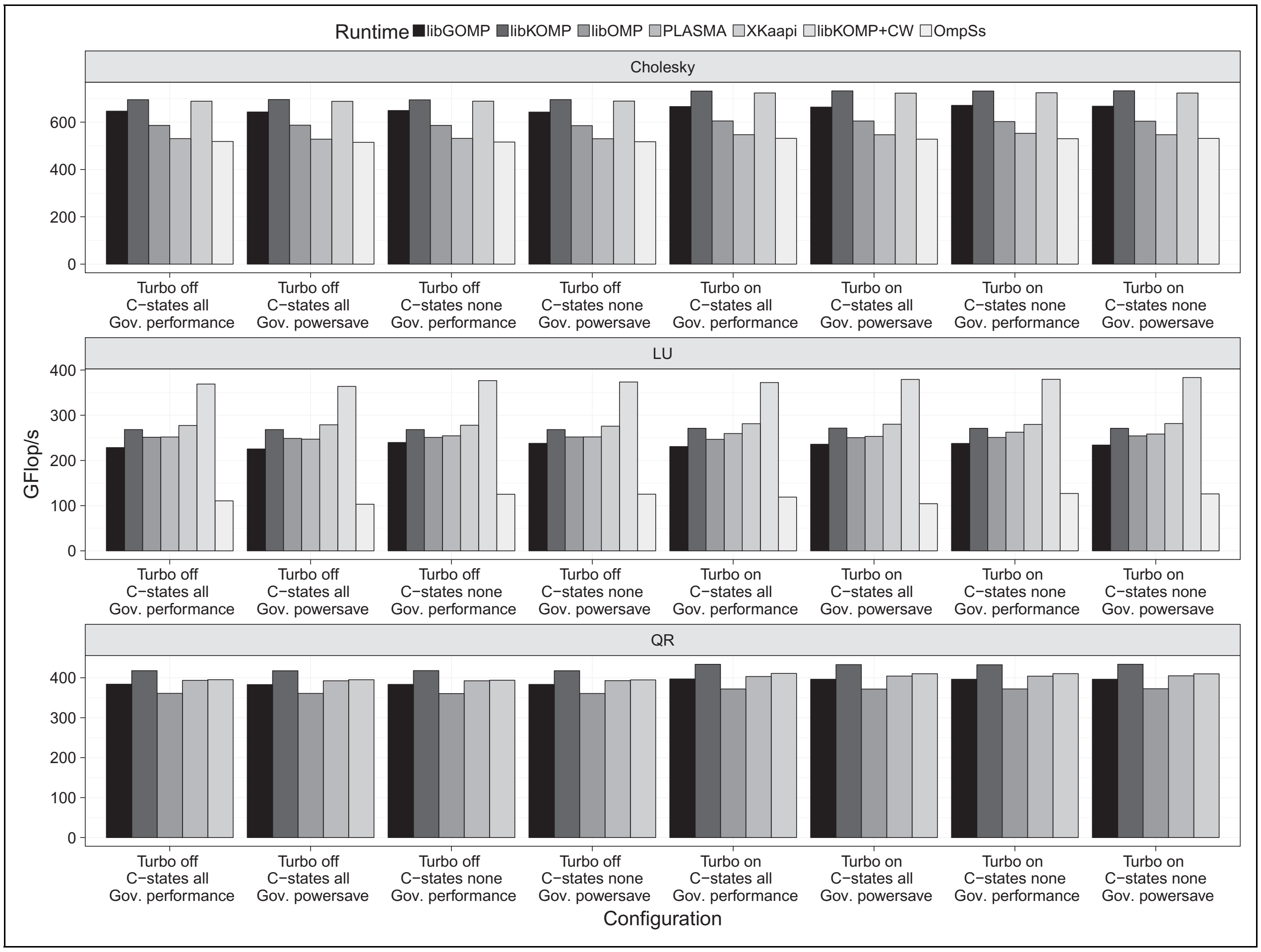

Figure 2. Performance results of Cholesky, LU, and QR over the UV2000 machine. The matrix size was 32,768 $\times$ 32,768 with 352 $\times$ 352 of block size.

and Figure 3 illustrates energy results from RAPL memory (DRAM) and processor (PKG) counters. Each configuration color represents the combination of the three CPU-level optimizations Turbo Boost (*on* and *off*), C-States (*all* enabled and *none*), and CPUFreq governor (*performance* and *powersave*). Table 2 summarizes the best results over each metric collected, that is, performance, DRAM energy counter, PKG energy counter, and DRAM+PKG energy counters.

In all cases, libKOMP attained the best performance results, followed by XKaapi and libGOMP for Cholesky, XKaapi and libKOMP (without CW) for LU, and PLASMA and XKaapi for QR. OmpSs had similar performance to PLASMA on Cholesky and performed worst than others on LU. Nonetheless, we were not able to evaluate OmpSs with QR due to a runtime error. The performance gains of libKOMP over libGOMP were 9.7% for Cholesky, 63.8% for LU, and 9.4% for QR. In addition, the CPU-level optimizations for these results were Turbo Boost enabled, powersave governor, and all C-States disabled. Clearly, the CW feature of LU contributed to the significant gain of libKOMP.

In energy, libKOMP had better energy results with Cholesky and LU, while QR had lower energy consumption with PLASMA implementation and libGOMP. The CPU-level optimizations for these results were Turbo Boost disabled, performance governor, and all C-States enabled for Cholesky and QR, and all disabled for LU.

The cost of energy-efficient cases in performance was not significant on the three benchmarks. Comparing the best cases of energy over the best cases in performance, Cholesky had a reduction of 5.12% in performance, while LU and QR had 1.76% and 9.31% reduction, respectively. Although, energy reduction was of 8.51%, 10.88%, and 11.71% for Cholesky, LU, and QR, respectively.

Regarding the CPU-level techniques, it seems that Turbo Boost and C-States contributed to the performance gains, and CPUFreq governors had more impact on energy than performance. It was expected that the best performance cases had Turbo enabled and all C-States disabled; still, those cases had powersave as CPU DVFS governor. Besides, energy-efficient cases had in most cases Turbo disabled, all C-States enabled, and performance as CPU frequency governor.

In order to investigate the performance gains of powersave governor, we analyzed the core temperature counter from RAPL over Cholesky with libKOMP runtime as

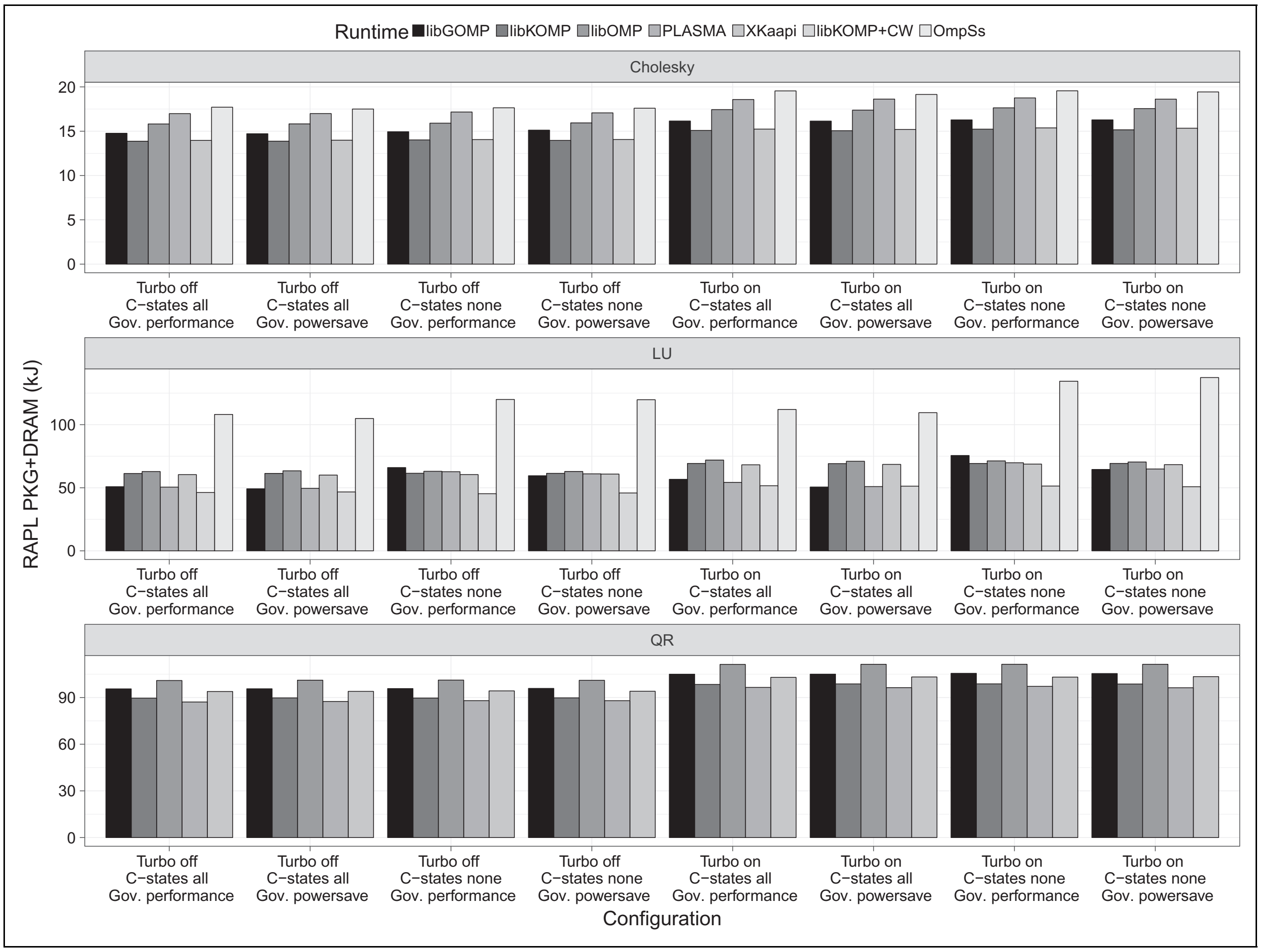

Figure 3. Energy results (kJ) of Cholesky, LU, and QR over the UV2000 machine. The matrix size was 32,768 × 32,768 with 352 × 352 of block size. The reported results are the sum of RAPL counters PKG and DRAM from all processor sockets. RAPL: running average power limit.

Table 2. Overview of the best results.[a]

Method	Runtime	Turbo Boost	C-States	Governor	GFlop/s	RAPL PKG	RAPL DRAM
Performance							
Cholesky	libKOMP	Enabled	None	Powersave	732.45	13.02	2.13
LU	libKOMP+CW	Enabled	None	Powersave	383.40	42.83	7.98
QR	libKOMP	Enabled	None	Powersave	433.77	83.36	15.31
RAPL PKG+DRAM							
Cholesky	libKOMP	Disabled	All	Performance	694.95	11.62	2.24
LU	libKOMP+CW	Disabled	None	Performance	376.64	37.12	8.16
QR	PLASMA	Disabled	All	Performance	393.40	71.42	15.70
RAPL PKG							
Cholesky	libKOMP	Disabled	All	Performance	694.95	11.62	2.24
LU	libGOMP	Disabled	All	Powersave	225.41	36.60	12.53
QR	PLASMA	Disabled	All	Performance	393.40	71.42	15.70
RAPL DRAM							
Cholesky	XKaapi	Enabled	None	Powersave	723.23	13.22	2.11
LU	libKOMP+CW	Enabled	None	Powersave	383.40	42.83	7.98
QR	PLASMA	Enabled	None	Powersave	404.92	81.07	15.19

[a]Higher is better for performance and lower is better in energy.
RAPL: running average power limit.

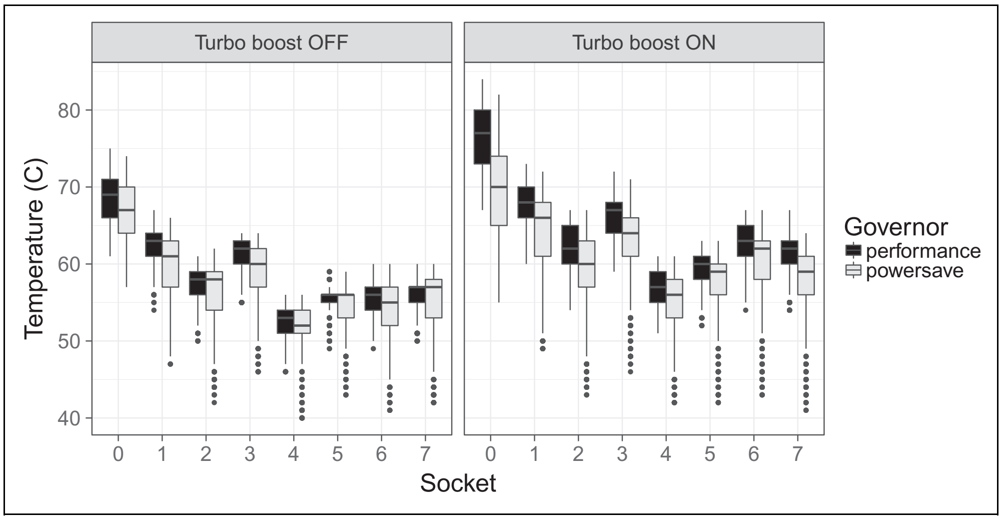

Figure 4. Processor temperature of each socket with Cholesky and libKOMP runtime. We compared the impact of CPUFreq governor and Turbo Boost. C-States were disabled.

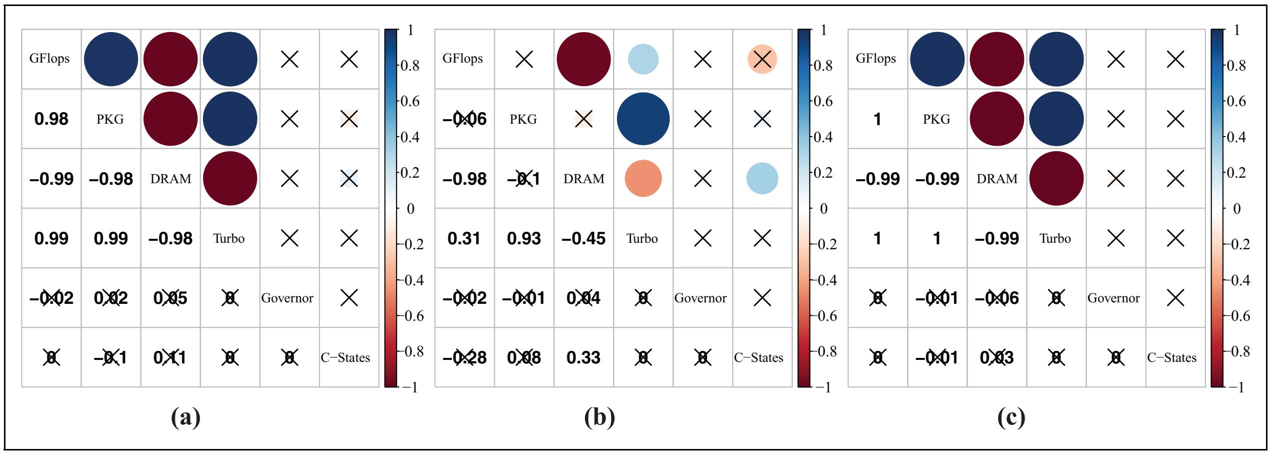

Figure 5. Correlation coefficients on the three benchmarks based on the best performance results. The × symbol means that the significance level (or *p*-value) of the correlation is under 5%. (a) Cholesky with libKOMP, (b) LU with libKOMP+CW, and (c) QR with libKOMP. CW: concurrent write.

illustrated in Figure 4. We collected each socket temperature through a series of Cholesky executions, including an interval between executions of 30 s, and computed the mean of all readings per socket. The performance governor had greater temperature readings than powersave on all sockets using Turbo Boost, while both governors had similar temperature readings without Turbo Boost. It seems that tuning all CPU-level optimizations to target performance reached the thermal limits of the processor sockets and degraded performance.

5.2. Correlation analysis

We used the Pearson correlation coefficient to test the correlation between two variables X and Y in order to identify the CPU-level techniques and their relation with performance and energy. This coefficient has values between -1 and $+1$, where $+1$ means a perfect positive linear correlation, 0 is no linear correlation, and -1 a total negative linear correlation. In addition, we added a significance test of the correlation with confidence interval of 95%.

Figure 5 shows a correlation graph for the best performance cases on the three benchmarks. We employ the following numerical values for each CPU-level optimization: Turbo Boost on $(+1)$ and off (-1), C-States all enabled $(+1)$ and all disabled (-1), and CPUFreq governor performance $(+1)$ and powersave (-1).

It seems that Turbo Boost parameter had direct impact on performance and energy with an almost perfect correlation on Cholesky and QR. An exception was LU with a lower correlation of 0.31 on performance. In all cases, the negative correlation between Turbo Boost and RAPL DRAM means that it reduced DRAM energy when enabled while increased PKG consumption.

The correlation results also show that CPUFreq governors and C-States had no relation with performance and energy. It appears that CPUFreq governor parameter was not relevant in our comparison of performance and energy

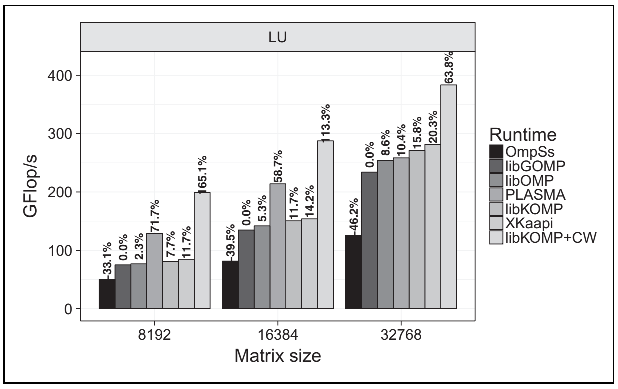

Figure 6. Performance (GFlop/s) results of LU. All percentages on top of bar plots are the difference of current runtime over GNU C compiler (GCC) libGOMP runtime.

in Table 2. On the other hand, C-States had a correlation of 0.33 with RAPL DRAM over LU, that is, increasing DRAM energy consumption with all C-States enabled. It explains the best energy case of LU with C-States disabled in Table 2.

5.3. Focus on LU factorization

Figure 6 shows performance results for LU factorization with different matrix sizes. We used as reference the GCC libGOMP runtime to compute the difference over the other four runtime systems and the OpenMP-based PLASMA, represented on the bar plots by a percentage value. The LU algorithm with CW showed significant improvement compared to other runtime systems (up to 165.1% over libGOMP), followed by PLASMA LU algorithm.

Thanks to the CW mode, the LU algorithm with libKOMP+CW runtime had more parallelism than other runtime systems due to the CW algorithm extension based on KASTORS (Virouleau et al., 2014). Figure 7 illustrates a Gantt execution from the LU factorization using PLASMA, libKOMP, and libKOMP with CW.

On the LU factorization, even if the CW version generates more parallelism, the algorithm has poor efficiency and threads are frequently idle. The Gantt diagram on all the 48 cores illustrates long periods of inactivity. libGOMP is the only runtime where threads lock common dequeue to get task. Linux will put these lightweight process idle. If we do not consider libKOMP+CW's algorithmic variant, then PLASMA algorithm with libGOMP is the best runtime in terms of energy consumption for LU factorization. This is not true for runtime systems based on task scheduling by work stealing such as libKOMP, libOMP, or XKaapi which have very active threads that consume energy.

6. Discussion

Majority of the best configurations from Figures 2 and 3, which are summarized in Table 2, were runtime systems using work-stealing–based scheduling. On fine grain problems, libKOMP and XKaapi were generally better. These results can be explained by the smaller task creation overhead on XKaapi and libKOMP than others.

The difference between libOMP and libKOMP is the new features we add into the original Intel libOMP runtime: the lightweight work stealing algorithm from Cilk and the request combining protocol from XKaapi. These features not only impact performance but also impact the way tasks are scheduled: It suppresses the bounded dequeue limitation that may degenerate task creation into task serialization. It means that at runtime a thread may be forced to execute immediately tasks for which no or less affinity exist. Without bounded size dequeue, a thread that completes a task will always activate one of the successors following a data flow relationship producer–consumer, thus sharing a data resident into cache or the thread becomes idle and tries to steal tasks. We will investigate by more finer experiments the exact impact of these additions in libKOMP.

On LU factorization where algorithmic variant libKOMP+CW was the best, it was followed by XKaapi and libKOMP on performance. LU factorization is a relevant code with inactivity sections from the dependencies imposed by the algorithm, mainly due to a search of pivot and swap of elements. This optimized algorithm allowed to increase performance while energy is decreased due to libKOMP+CW runtime and CW OpenMP extension (Virouleau et al., 2014). Nevertheless, the platform characteristic, and especially its memory network, had also an impact on both performance and energy consumption.

Without these algorithm variants, LU factorization code consumes less energy using GCC libGOMP runtime. In libGOMP, the synchronization between threads on the shared task dequeue resource wastes less cycles. A work-stealing–based runtime may have interest to incorporate part of Ribic and Liu (2014), which is used to lower the speed of threads that are not in the critical path with a warranty on performance. One of the biggest challenges is the design of adaptive OpenMP runtime capable to saving energy on short delays of inactivity.

Our findings on CPU-level optimizations lead us to believe that the three CPU-level optimizations impact performance and energy of OpenMP runtime systems. The two processors parameters, Turbo Boost and C-States, had significant influence over experimental results. The former increased performance significantly at energy cost, while the latter reduced energy at idle states. The impact of C-States over work-stealing–based runtime systems was not clear because our correlation test (Figure 5) showed no relation between C-States and the experimental results. Nevertheless, we acknowledge that runtime systems based on work stealing may take advantage of steal phases at the beginning and the end of the computation to enter in idle state and reduce energy consumption (Tchiboukdjian et al., 2013).

CPUFreq governors had more impact at experimental cases with Turbo Boost disabled. Our experimental results suggest that tuning all three CPU-level optimizations to

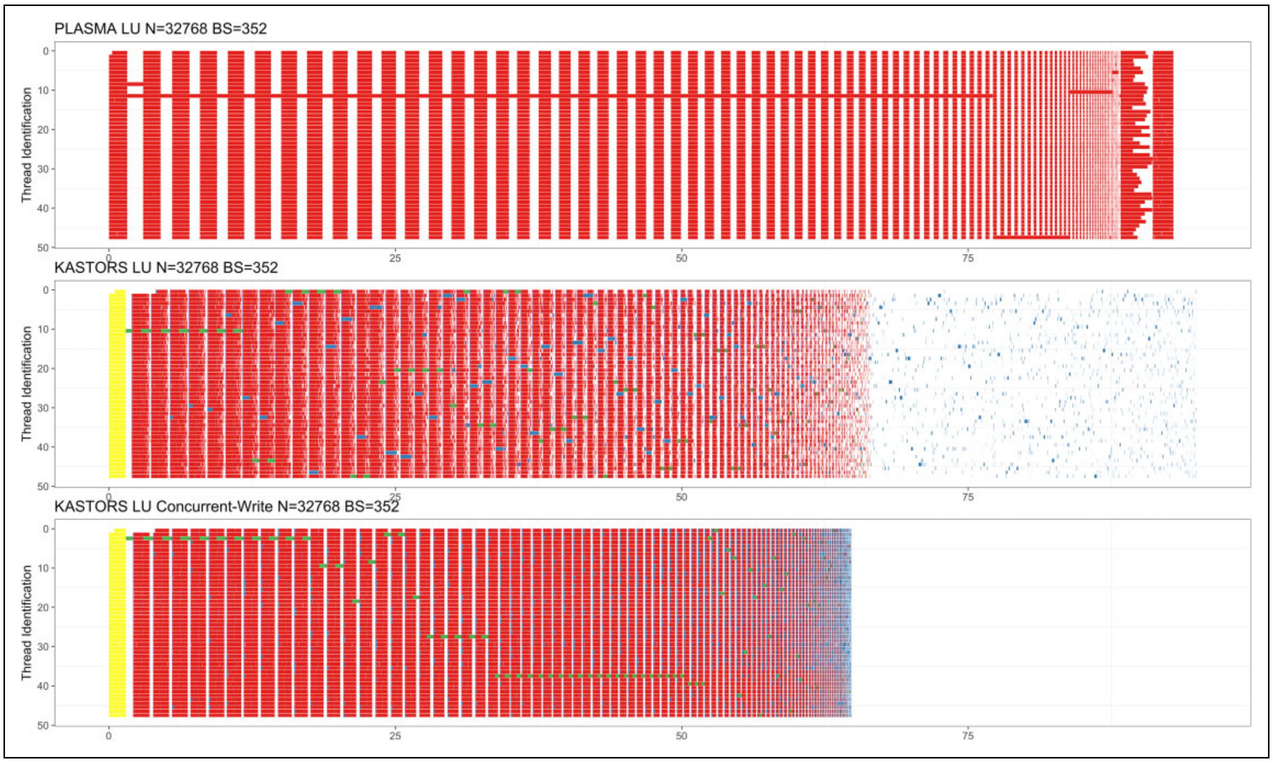

Figure 7. Gantt of LU algorithm from PLASMA (top), KASTORS and libKOMP (middle), and KASTORS with libKOMP and concurrent write (bottom). The matrix size was 32,768 $\times$ 32,768 with 352 $\times$ 352 of block size.

target performance, mostly DVFS governor and Turbo Boost, degraded performance due to CPU thermal limitations. Still, the powersave governor was able to sustain performance while consuming less energy. Other works show experimental results on techniques using DVFS, as discussed in Section 2.

7. Conclusion

In this article, experiments with five production-based OpenMP runtime systems and three CPU-level optimizations (Turbo Boost, C-States, and Linux CPUFreq governors) on the three main kernels in dense linear algebra were conducted on an NUMA platform. We showed that OpenMP runtime is a new leverage for controlling energy, and Turbo Boost, as well as C-States, impacted significantly performance and energy. Our experimental results suggest that small algorithmic and runtime improvements may allow performance gains up to 63% and thus reducing the energy by 29% over LU factorization from PLASMA using GCC libGOMP runtime. Runtime systems based on work stealing were more efficient in performance; although, it was not clear the impact of C-States at idle phases in order to reduce energy consumption.

Future works include an extension of our experimental comparison over a wide range of architectures, including Intel KNL many-core; experiments using other numerical benchmarks such as the NAS benchmark (Griebler et al., 2018); and finer control of architectural features through high-level tools such as libmsr[7] and Mammut (De Sensi et al., 2017). In addition, we will evaluate the impact on performance and energy of idle states at steal phases of work stealing scheduler.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the ELCI project, a French FSN (Fond pour la Société Numérique) project that associates academic and industrial partners to design and provide a software environment for very high-performance computing, and the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

ORCID iD

João Vicente Ferreira Lima https://orcid.org/0000-0002-2670-6963

Notes

1. OpenMP does not allow variable renaming to suppress output and anti-dependencies.

2. http://gitlab.inria.fr/openmp/clang
3. Mercurial hash from https://bitbucket.org/icl/plasma
4. Git repository: https://gitlab.inria.fr/openmp/libkomp
5. Available at: http://kaapi.gforge.inria.fr
6. Available at: https://pm.bsc.es
7. https://github.com/LLNL/libmsr

References

Bari MAS, Chaimov N, Malik AM, et al. (2016) Arcs: adaptive runtime configuration selection for power-constrained OpenMP applications. In: *2016 IEEE international conference on cluster computing (CLUSTER)* (eds Dongarra J, Matsuoka S and King C-T), Taipei, Taiwan, 12–16 September 2016, pp. 461–470. Washington, DC: IEEE Computer Society. DOI: 10.1109/CLUSTER.2016.39.

Benoit A, Lefevre L, Orgerie AC, et al. (2017) Shutdown policies with power capping for large scale computing systems. In: *Europar 2017: International European conference on parallel and distributed computing* (eds Rivera FF, Pena TF and Cabaleiro JC), 28 August–1 September 2017, pp. 134–146. Santiago de Compostela, Spain: Springer International Publishing.

Bergman K, Borkar S, Campbell D, et al. (2008) Exascale computing study: technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO): Technical Report* 15. Notre Dame, IN: DARPA IPTO, Air Force Research Labs.

Bhalachandra S, Porterfield A and Prins JF (2015) Using dynamic duty cycle modulation to improve energy efficiency in high performance computing. In: *2015 IEEE international parallel and distributed processing symposium workshop* (eds Barua S and Govindarajan R), Hyderabad, India, 25–29 May 2015, pp. 911–918. Washington, DC: IEEE Computer Society. DOI: 10.1109/IPDPSW.2015.144.

Broquedis F, Gautier T and Danjean V (2012) LibKOMP, an efficient OpenMP runtime system for both fork-join and data flow paradigms. In: *Proceedings of the 8th international conference on OpenMP in a heterogeneous world*, IWOMP '12 (eds Chapman BM, Massaioli F, Muller MS, et al), Rome, Italy, 11–13 June 2012, pp. 102–115. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-642-30960-1.

Bueno Hedo J, Planas J, Duran A, et al. (2012) Productive programming of GPU clusters with OmpSs. *IEEE Computer Society*, 557–568. ISBN 978-1-4673-0975-2.

Cicotti P, Tiwari A and Carrington L (2014) Efficient speed (ES): adaptive DVFS and clock modulation for energy efficiency. In: *2014 IEEE international conference on cluster computing (CLUSTER)* (ed Perez MS), 22–26 September 2014, pp. 158–166. Washington, DC: IEEE Computer Society. DOI: 10.1109/CLUSTER.2014.6968750.

Curtis Maury M, Dzierwa J, Antonopoulos CD, et al. (2006) Online strategies for high-performance power-aware thread execution on emerging multiprocessors. In: *Proceedings 20th IEEE international parallel distributed processing symposium* (eds Spirakis P and Siegel HJ), 25–29 April 2006, pp. 8–15. Washington, DC: IEEE Computer Society. DOI: 10.1109/IPDPS.2006.1639598.

De Matteis T and Mencagli G (2017) Proactive elasticity and energy awareness in data stream processing. *Journal of Systems and Software* 127(C): 302–319. DOI: 10.1016/j.jss.2016.08.037.

De Sensi D, Torquati M and Danelutto M (2017) Mammut: high-level management of system knobs and sensors. *SoftwareX 6*: 150–154. DOI: 10.1016/j.softx.2017.06.005. Available at: http://www.sciencedirect.com/science/article/pii/S2352711017300225 (accessed 14 May 2018).

Duran A, Teruel X, Ferrer R, et al. (2009) Barcelona OpenMP tasks suite: a set of benchmarks targeting the exploitation of task parallelism in OpenMP. In: *Proceedings of the 2009 international conference on parallel processing*, ICPP '09 (eds Tjoa AM and Takizawa M), Vienna, Austria, 22–25 September 2009, pp. 124–131. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-3802-0. DOI: 10.1109/ICPP.2009.64.

Etinski M, Corbalan J, Labarta J, et al. (2010) Utilization driven power-aware parallel job scheduling. *Computer Science-Research and Development* 25(3–4): 207–216.

Freeh VW, Lowenthal DK, Pan F, et al. (2007) Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems* 18(6): 835–848. DOI: 10.1109/TPDS.2007.1026.

Frigo M, Leiserson CE and Randall KH (1998) The implementation of the Cilk-5 multithreaded language. *SIGPLAN Not* 33(5): 212–223.

Gautier T and Virouleau P (2015) New libKOMP library. Available at: http://gitlab.inria.fr/openmp/libkomp (accessed 14 May 2018).

Gautier T, Lima JVF, Maillard N, et al. (2013) Xkaapi: a runtime system for data-flow task programming on heterogeneous architectures. In: *Proceedings of the 2013 IEEE 27th international symposium on parallel and distributed processing*, IPDPS '13 (eds Herbordt M and Weems CC), Boston, MA, 20–24 May 2013, pp. 1299–1308. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-4971-2.

Griebler D, Loff J, Mencagli G, et al. (2018) Efficient NAS benchmark kernels with C++ parallel programming. In: *Proceedings 26th Euromicro international conference on parallel, distributed and network-based processing (PDP)* (eds Merelli I and Kotenko PLI), Cambridge, UK, 21–23 March 2018, pp. 733–740.

Lagravière J, Ha HP and Cai X (2015) *Evaluation of the power efficiency of UPC, OpenMP and MPI: Technical Report 2015-76*. Tromsø, Norway: Institutt for informatikk Tromsø.

Laros JH III, Pedretti KT, Kelly SM, et al. (2012) Energy based performance tuning for large scale high performance computing systems. In: *Proceedings of the 2012 symposium on high performance computing*, HPC '12 (ed ElAarag H), Orlando, Florida, 26–30 March 2012, pp. 6:1–6:10. San Diego, CA, USA: Society for Computer Simulation International. ISBN 978-1-61839-788-1.

Li D, de Supinski BR, Schulz M, et al. (2010) Hybrid MPI/OpenMP power-aware computing. In: *2010 IEEE* *international symposium on parallel distributed processing (IPDPS)* (ed Bader DA), Atlanta, GA, 19–23 April 2010, pp. 1–12. Washington, DC: IEEE Computer Society. DOI: 10.1109/IPDPS.2010.5470463.

Lima JVF, Raïs I, Lefevre L, et al. (2017) Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms. In: *International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)* (ed Azevedo R), Campinas, SP, Brazil, 17–20 October 2017, pp. 7–12. Washington, DC: IEEE Computer Society.

Lively C, Wu X, Taylor V, et al. (2011) Energy and performance characteristics of different parallel implementations of scientific applications on multicore systems. *International Journal of High Performance Computing Applications* 25(3): 342–350. DOI: 10.1177/1094342011414749.

Marathe A, Bailey PE, Lowenthal DK, et al. (2015) A run-time system for power-constrained HPC applications. In: *High performance computing: 30th international conference, ISC high performance 2015, proceedings, chapter* (eds Kunkel JM and Ludwig T), Frankfurt, Germany, 12–16 July 2015, pp. 394–408. Cham, Switzerland: Springer. ISBN 978-3-319-20119-1. DOI: 10.1007/978-3-319-20119-1_28.

Nandamuri A, Malik AM, Qawasmeh A, et al. (2014) Power and energy footprint of OpenMP programs using OpenMP runtime API. In: *Proceedings of the 2nd international workshop on energy efficient supercomputing*, E2SC '14 (eds Cameron K, Hoisie A, Kerbyson D, et al), New Orleans, LA, 16 November 2014, pp. 79–88. Piscataway, NJ, USA: IEEE Press. ISBN 978-1-4799-7036-0. DOI: 10.1109/E2SC.2014.11.

Orgerie AC, Assuncao MD and Lefevre L (2014) A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Computing Surveys* 46: 47:1–47:31. DOI: 10.1145/2532637.

Porterfield AK, Olivier SL, Bhalachandra S, et al. (2013) Power measurement and concurrency throttling for energy reduction in OpenMP programs. In: *2013 IEEE international symposium on parallel distributed processing, workshops and PhD forum* (eds Herbordt M and Weems CC), Cambridge, MA, 20–24 May 2013, pp. 884–891. Washington, DC: IEEE Computer Society. DOI: 10.1109/IPDPSW.2013.15.

Ribic H and Liu YD (2014) Energy-efficient work-stealing language runtimes. *SIGARCH Computer Architecture News* 42(1): 513–528. DOI: 10.1145/2654822.2541971.

Rotem E, Naveh A, Ananthakrishnan A, et al. (2012) Power-management architecture of the Intel microarchitecture code-named sandy bridge. *IEEE Micro* 32(2): 20–27. DOI: 10.1109/MM.2012.12.

Rountree B, Lownenthal DK, de Supinski BR, et al. (2009) Adagio: Making DVS practical for complex HPC applications. In: *Proceedings of the 23rd international conference on supercomputing, ICS '09* (eds Gschwind M and Nicolau A), Yorktown Heights, NY, 08–12 June 2009, pp. 460–469. New York, NY, USA: ACM. ISBN 978-1-60558-498-0. DOI: 10.1145/1542275.1542340.

Schöne R and Molka D (2014) Integrating performance analysis and energy efficiency optimizations in a unified environment. *Computer Science—Research and Development* 29(3): 231–239. DOI: 10.1007/s00450-013-0243-7.

Schöne R, Ilsche T, Bielert M, et al. (2016) Software controlled clock modulation for energy efficiency optimization on Intel processors. In: *2016 4th international workshop on energy efficient supercomputing (E2SC)* (eds Cameron K, Kerbyson Hoisie, D, Nikolopoulos Lowenthal, DS, et al), Salt Lake City, UT, 14 November 2016, pp. 69–76. Washington, DC: IEEE Computer Society. DOI: 10.1109/E2SC.2016.015.

Su C, Li D, Nikolopoulos DS, et al. (2012) Model-based, memory-centric performance and power optimization on NUMA multiprocessors. In: *2012 IEEE international symposium on workload characterization (IISWC)* (ed Carter J), La Jolla, CA, 4–6 November 2012, pp. 164–173. Washington, DC: IEEE Computer Society. DOI: 10.1109/IISWC.2012.6402921.

Sundriyal V, Sosonkina M and Zhang Z (2014) Automatic runtime frequency-scaling system for energy savings in parallel applications. *The Journal of Supercomputing* 68(2): 777–797. DOI: 10.1007/s11227-013-1062-0.

Tchiboukdjian M, Gast N and Trystram D (2013) Decentralized list scheduling. *Annals of Operations Research* 207(1): 237–259.

Treibig J, Hager G and Wellein G (2010) LIKWID: Lightweight performance tools. In: *International conference on competence in high performance computing* (eds Bischof C, Hegering H-G, Nagel WE, et al), Schloss Schwetzingen, Germany, 22–24 June 2010, pp. 165–175. Berlin, Heidelber: Springer.

Virouleau P, Broquedis F, Gautier T, et al. (2016) Using data dependencies to improve task-based scheduling strategies on NUMA architectures. In: *Proceedings of the 22nd international conference on euro-par 2016: parallel processing, vol. 9833* (eds Dutot P-F and Trystram D), Grenoble, France, 24–26 August 2016, pp. 531–544. Berlin, Heidelberg: Springer. ISBN 978-3-319-43658-6. DOI: 10.1007/978-3-319-43659-3_39.

Virouleau P, Brunet P, Broquedis F, et al. (2014) Evaluation of OpenMP dependent tasks with the KASTORS benchmark suite. In: *10th international workshop on OpenMP, IWOMP '14* (eds DeRose L, de Supinski BR, Olivier SL, et al), Salvador, Brazil, 28–30 September 2014, pp. 16–29. Berlin, Heidelberg: Springer. DOI: 10.1007/978-3-319-11454-5_2.

Wang W, Porterfield A, Cavazos J, et al. (2015) Using per-loop CPU clock modulation for energy efficiency in OpenMP applications. In: *2015 44th international conference on parallel processing* (eds Yang Y and Wu J), Beijing, China, 1–4 September 2015, pp. 629–638. Washington, DC: IEEE Computer Society. DOI: 10.1109/ICPP.2015.72.

Yang X, Zhou Z, Wallace S, et al. (2013) Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems. In: *Proceedings of the international conference on high performance computing, networking, storage and analysis*, SC '13 (ed Gropp W), Denver, CO, 17–22 November 2013, pp. 60:1–60:11. New York, NY, USA: ACM. ISBN 978-1-4503-2378-9. DOI: 10.1145/2503210.2503264.

YarKhan A, Kurzak J, Luszczek P, et al. (2016) Porting the plasma numerical library to the OpenMP standard.

International Journal of Parallel Programming 45: 1–22. DOI: 10.1007/s10766-016-0441-6.

Zhang X, Dwarkadas S and Shen K (2009) Hardware execution throttling for multi-core resource management. In: *Proceedings of the 2009 conference on USENIX annual technical conference, USENIX '09*, pp. 23–23. Berkeley, CA, USA: USENIX Association. Available at: http://dl.acm.org/citation.cfm?id=1855807.1855830 (accessed 14 May 2018).

Zhuravlev S, Saez JC, Blagodurov S, et al. (2013) Survey of energy-cognizant scheduling techniques. *IEEE Transactions on Parallel and Distributed Systems* 24(7): 1447–1464. DOI: 10.1109/TPDS.2012.20.

Author biographies

João Vicente Ferreira Lima received a joint PhD degree in Computer Science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, and the MSTII Doctoral School at the Grenoble University, France. He received a BSc degree in Computer Science in 2007 from the Federal University of Santa Maria (UFSM), Brazil, and an MSc degree in Computer Science in 2009 from the Federal University of Rio Grande do Sul (UFRGS), Brazil. He is an Associate Professor at the Federal University of Santa Maria (UFSM), Brazil, since 2014. His research interests are high-performance computing, runtime systems for HPC, parallel programming for accelerators, and distributed computing.

Issam Raïs is a PhD student since 2015 at ENS Lyon, France. He received an MSc degree in Computer Science and a BSc degree in Computer and Information Sciences at the University of Orleans, France. He works in the AVALON team at the LIP laboratory (Laboratoire de l'Informatique du Parallélisme) advised by Laurent Lefèvre, Anne Benoit, and Anne-Cécile Orgerie.

Laurent Lefèvre obtained his PhD degree in Computer Science in January 1997 at the LIP Laboratory in ENS-Lyon (École Normale Supérieure), France. From 1997 to 2001, he was an Assistant Professor in Computer Science in Lyon 1 University and a member of the RESAM Laboratory (High Performance Networks and Multimedia Application Support Lab). Since 2001, he has been a Research Associate in computer science at INRIA (the French Institute for Research in Computer Science and Control). He is a member of the INRIA AVALON team (Algorithms and Software Architectures for Distributed and HPC systems) at the LIP laboratory in Lyon, France. His research interests focus on green and energy efficient computing and networking. He has organized several conferences in high-performance networking and computing and he is a member of several program committees. He has coauthored more than 100 papers published in refereed journals and conference proceedings. He participates in several national and European projects on energy efficiency. For more information, see http://perso.ens-lyon.fr/laurent.lefevre/.

Thierry Gautier received his Dipl.-Ing., MS, and PhD degrees in computer science at the INPG in 1996. He is a full-time researcher at INRIA (the French National Institute for Computer Science and Control), with the AVALON project team of the LIP laboratory in Lyon, France, and has held a postdoctoral position at ETH Zürich (1997). He conducts research in high-performance computing, runtime systems for HPC, parallel algorithms, parallel programming, OpenMP, and multicore architectures.