Exploiting the Table of Energy and Power Leverages

Issam Raïs, Laurent Lefèvre, Anne-Cécile Orgerie, Anne Benoit

Exploiting the Table of Energy and Power Leverages

Issam Raïs[1], Laurent Lefèvre[1], Anne-Cécile Orgerie[3], Anne Benoit[1,2]

[1] Laboratoire LIP, École Normale Supérieure de Lyon & Inria, France
[2] Georgia Institute of Technology, Atlanta, GA, USA
[3] Univ. Rennes, Inria, CNRS, IRISA, Rennes, France
firstname.lastname@inria.fr

Abstract. Large scale distributed systems and supercomputers consume huge amounts of energy. To address this issue, a large set of hardware and software capabilities and techniques (leverages) exist to modify power and energy consumption in large scale systems. Discovering, benchmarking and efficiently exploiting such leverages, remains a real challenge for most of the users. In this paper, we define leverages and the *table of leverages*, and we propose algorithms and predicates that ease the reading of the table of leverages and extract knowledge from it.

1 Introduction

Data centers worldwide consumed around 194 terawatt hours (TWh) of electricity in 2014, or about 1% of total demand [2]. This worrying consumption has direct financial and environmental consequences on data center managers, like Cloud providers and supercomputer operators. Several techniques have been developed in order to lower the electrical consumption of data centers. These techniques, that we call leverages, can improve the energy efficiency of data centers at different levels: hardware, middleware, and application. Hardware leverages include Dynamic Voltage and Frequency Scaling (DVFS) [11] and shutdown techniques [10]. At the middleware level, energy-efficient resource allocation policies for job managers are examples of leverages [7]. Finally, leverages at the application level include green programming [1].

While many of these leverages have been independently studied in the literature, few works consider the utilization of several leverages at the same time, and no more than two leverages. Yet, the utilization of a given leverage can impact both the utilization and the efficiency of another leverage. The variety of leverages is added to the data center's complexity, in terms of size and hardware heterogeneity, and makes energy efficiency complex to reach for the users who have access to multiple leverages.

In this work, we aim at extending the current state of the art, which is studying the influence of one or two leverages at maximum at the same time, thus ignoring the impacts incurred by the utilization of more leverages. Thus, we proposed a generic definition, combination and knowledge extraction of multiple leverages in order to fully explore their combined impacts.

We propose a first approach toward a completely automated process to characterize the leverages available on a data center node. The key idea of our contribution consists in providing hints to users about the most suitable solution

for their application from a defined score table with a value for each leverage combination and each studied metric. Through these tables could be derived knowledge about leverage combination and effects they incur on each other. From the definition of a table of leverages, a tool to help a user, a developer or an administrator to choose which leverage or leverage combination suits the best his objectives (here with a focus on energy or power metrics), the contribution of this paper consists in the algorithms proposed to extract knowledge about the interaction of leverages and their influence on a given metric.

The remaining of this paper is structured as follows. Section 2 formalizes the concept of leverages, and illustrates this formalism on the leverages under consideration in this paper. Section 3 defines and explains how to build the table of leverages. Section 4 presents the experimental setup and a first full example of table of leverages. Section 5 then shows how to exploit the raw data of the table of leverages and extract useful knowledge. Finally, Section 6 concludes this work and gives perspectives.[4]

2 Leverage definition

In this section, we first propose a formalization of a leverage. Second, we apply this formalism to the leverages that we selected for this paper.

Definition 1 *A* leverage L *is composed of* $S = \{s_0, s_1, \ldots, s_n\}$*, the set of available valid states of* L*, and* s_c*, the current state of* L*.*

Thus, an energy or power leverage is a leverage that has a high impact on the energy or power consumption of a device through its various states or through the modification of its current state. Switching from one state to another can have a cost in terms of time and energy. Yet, in the current work, we focus on studying the impacts of leverage combinations over a single intensive application phase [4], and thus we do not study the switching costs between states.

In this paper, we consider multiple leverages available on current hardware, namely multi-thread, computation precision and vectorization. These leverages belong to different categories of leverages: application level with computation precision and vectorization techniques, and middleware level with multi-threading. These leverages are described hereafter.

Multi-thread leverage. The first studied leverage is a middleware-level leverage that permits the usage of multiple cores during computation. OpenMP [5], a well-known application programming interface abstraction for multi-threading, can be used to exploit this intra-node parallelism of multi-cores. It consists of a set of directives that modifies the behavior of the executed code, where a master thread forks a specific number of slave threads that run concurrently.

[4] This work is supported by the ELCI project, a French FSN project that associates academic and industrial partners to design and provide software environment for very high performance computing. Experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities (https://www.grid5000.fr).

This multi-thread leverage increases the CPU utilization of the node. Consequently, because of the non-power proportionality of current hardware architectures [10], this leverage can improve the energy efficiency of the node. In the rest of the paper, the multi-thread leverage is denoted by *nbThreads* with the set of states $\{1, \ldots, n_{max}\}$, where 1 means that one OpenMP thread is used, and n_{max} corresponds to the maximum number of threads that could be launched simultaneously on the node. In this work, only the extreme states, 1 and n_{max}, are explored.

Computation precision leverage. The second leverage belongs to the application level and exploits the various computation precision options available on actual hardware (i.e., int, float, double). Such a leverage alters the precision of the results computed by the application, but lower precision translates into shorter data representation and so, less computation and less energy consumption. At the application level, the user can specify a desired Quality-of-Service that can be expressed as accessible computation precision states.

This precision leverage is denoted by $Precision$, and the set of states is {int, float, double}, corresponding to the data format for the application. For each of these states, a different code version is provided.

Vectorization leverage. Finally, the last studied leverage concerns the application level. Current CPUs allow the usage of vectorization capabilities to exploit intra-core parallelism. On Intel architectures, it started with MMX instruction in Pentium P5 architectures in 1997 [9]. It was then extended to SSE [6]. SSE was then extended to SSE2, SSE3, SSSE3 and finally SSE4. AVX [8] then introduces new instructions, followed by AVX2 and finally AVX512 available in XeonPhi architecture. In this paper, we focus on SSE3 and AVX2, which are representative of the SSE and AVX families. These instruction sets permit single instruction on multiple data (SIMD) at application level.

This vectorization leverage is denoted by $Vectorization$. The set of states is {none, SSE3, AVX2}, where *none* means that no vectorization is used. For each of these states, a different code version is provided using the specific intrinsics and adequate compilation flags for each version.

The proposed leverage formalism described above is used in the rest of the paper to easily describe the state of each considered leverage and the possible combinations of leverages. The three leverages studied here are chosen to be representative examples of available leverages on modern architectures and frequently used during HPC applications. The methodology proposed in this paper is designed to be applied to any number and any type of leverages.

3 The table of leverages

We describe the table of leverages, which relies on metrics and benchmarks to characterize the performance and energy impact of each leverage combination on a given node. For each metric and each benchmark, a score is attributed to a given leverage combination. The table is then used to extract knowledge about

each leverage and evaluate impacts of leverage combinations in order to help the users to utilize their computing infrastructure in a more energy-efficient way.

Metrics. Leverages may influence the quality of service or performance of an application. For instance, shutdown techniques may induce latency in waking up the required nodes. Consequently, for these leverages, users need to determine their acceptable trade-off between energy-related metrics and performance metrics. The table of leverages relies on three different metrics that represent both energy and performance constraints. These metrics are measured for a given period of time corresponding to the time spent during benchmark execution.

The two first metrics are energy and power related metrics. To define them, we introduce the following notations: $T = \{t_0, \ldots, t_N\}$ is the set of time stamps of energy consumption measurements of a given run; t_0 and t_N represent the starting and ending timestamps (with a distance of one second), respectively; p_j, $j \in [0, N]$, represents the power consumption (in Watt), of the considered node for the timestamp t_j. **Metric 1:** The average power consumption of an executable is denoted $avrgWatt$, and it is defined as $avrgWatt = \sum_{j \in [0,N]} p_j/(N+1)$. **Metric 2:** The energy consumption of an executable is denoted $Joules$. It represents the energy consumption of the complete node used between t_0 and t_N. It is defined as $Joules = \sum_{j \in [0,N-1]} (t_{j+1} - t_j) \times p_j$. **Metric 3:** The last metric concerns the performance of the run, and is expressed as the execution time, denoted $Time$. It includes whole execution time of an executable, including initialization.

Benchmarks. A benchmark corresponds to a self-contained application that is representative of typical applications or portions of applications. The benchmark is compiled before the run, and once launched, the metrics previously defined are collected during its execution.

Here, for the sake of clarity, we evaluate only one benchmark for a set of embedded leverages. We chose to focus on a well-known CPU intensive code: the line per line matrix multiplication (LpL MM) of dense random large squared matrices (8192 as dimension size). The same algorithm is implemented for the various leverage combinations. The considered leverages are multi-thread, computation precision and vectorization. For the last two leverages, a different state means a different version of code, here generated by hand using dedicated intrinsics and compilation flags (-O3 -msse3 -mavx2). We deactivated the auto vectorization of the compiler (-fno-tree-vectorize) to have a control over the chosen intrinsics and because auto generation of vectorizable code is not one of the focused leverage in this paper.

Formalization of the table of leverages. Here, we describe how to compute the score associated to each metric for each leverage. Let X, Y, Z be the sets of available states of three leverages χ, ψ, ω (corresponding to S, the set of states for a given leverage L, from definition 1): $X = \{x_0, \ldots, x_{n_x}\}$, $Y = \{y_0, \ldots, y_{n_y}\}$, and $Z = \{z_0, \ldots, z_{n_z}\}$. Let $g_1, \ldots, g_m$ be the measured metric functions, as for instance $avrgWatt$, $Joules$, and $Time$. For all u $(1 \leq u \leq m)$, $g_u(x_i, y_j, z_k)$ is the value of metric g_u for the states x_i, y_j, z_k for the leverages χ, ψ, ω.

In the table of leverages, each line corresponds to a combination of states for each leverage and the columns correspond to the measured metrics. We normalize each value on the minimum value for each metric. These normalized values constitute the scores indicated in the table of leverages. Let $h_1, \ldots, h_m$ be the normalized versions of $g_1, \ldots, g_m$. So, we have, for $1 \leq u \leq m$, $h_u(x_i, y_j, z_k) = \frac{g_u(x_i, y_j, z_k)}{\min_{x_{i'} \in X, y_{j'} \in Y, z_{k'} \in Z} g_u(x_{i'}, y_{j'}, z_{k'})}$, with $h_u(x_i, y_j, z_k)$ being the value in the table of leverages in column of metric u and corresponding to the line for the states x_i, y_j, z_k respectively for the leverages χ, ψ, ω.

For application-level leverages, here *Precision* and *Vectorization*, the chosen benchmarks correspond to a different combination of application leverage states. Leverage *nbThreads* changes its state through environment variable. When all states are covered, the table of leverages is complete for the considered benchmark. Reducing the creation time of such a table is not the focus of this paper.

4 Building and analyzing the table of leverages

In this section, we present the table of leverages built on a node from our experimental testbed, Grid'5000 [3]. Grid'5000 deploys clusters linked with dedicated high performance networks in several cities in France. As our focus is on energy and performance related metrics, we used the Lyon site, where the energy consumption of every computing node is monitored through a dedicated wattmeter, exposing one power measurement per second with a 0.125 Watts accuracy. The Nova cluster from Lyon is used in the following. This cluster contains Dell PowerEdge R430 with 2 CPU E5-2620 v4 of 8 cores each, 32 GB of memory, 2 HDD disks of 300 GB each.

We applied our previous methodology for the three chosen leverages to the CPU intensive benchmark. This allows us to explore all possible states of chosen leverages, and thus to build a complete table of leverages. The table has the following format: the first three columns present the states of the $nbThreads$, $Precision$, and $Vectorization$ leverages respectively, while the last three columns show the normalized results of the three metrics $avrgWatt$, $Joules$, and $Time$, respectively, for every combination of leverage. As can be seen in Table 1 (first six columns), a line represents results of all gathered metrics for the execution of a representative load for a chosen combination of leverages. The results are normalized as explained before. The table of leverages gathers the knowledge of a Nova node, for a given workload done for multiple states of leverages combined.

Explanation of the table: A lot of unexpected results, at first sight, are detected in Table of leverage 1, like the combination with *int* being better than *float* and *double* when *1* and *none* are the chosen state for the *nbThread* and *Vectorization* leverages, with this trend being reversed with *nbThreads=32*.

From the set of combination with *1* as the chosen state for leverage *nbThreads*, it is logic to see that *int* is quicker than *float* then *double* from a cache usage perspective. Indeed, more data can be brought into the cache to compute without the need to fetch new data compared to float or double representation that

Table 1: Normalized table of leverage states and ranked impact for line per line matrix multiplication (LpL MM) benchmark on a Nova node.

Leverage states			Table of leverages			Ranked impact		
nbThreads (T)	Prec. (P)	Vector. (V)	avrgWatt	Joules	Time	avrgWatt	Joules	Time
1	int	none	1.05	65.09	61.89	P,T,V	P,T,V	P,T,V
1	int	SSE3	1.06	28.26	26.56	P,V,T	V,P,T	V,P,T
1	int	AVX2	1.06	29.32	27.67	P,V,T	V,P,T	V,P,T
1	float	none	1.05	72.97	69.67	P,V,T	P,T,V	P,T,V
1	float	SSE3	1.06	33.8	31.89	V,P,T	V,P,T	V,P,T
1	float	AVX2	1.05	36.8	34.89	P,V,T	V,P,T	V,P,T
1	double	none	1.06	81.59	76.89	P,T,V	P,T,V	P,T,V
1	double	SSE3	1.07	58.52	54.89	V,P,T	V,P,T	V,P,T
1	double	AVX2	1.06	57.72	54.22	P,V,T	V,P,T	V,P,T
32	int	none	1.43	13.48	9.44	P,T,V	T,P,V	T,P,V
32	int	SSE3	1.4	4.68	3.33	P,V,T	T,V,P	T,V,P
32	int	AVX2	1.0	1.0	1.0	P,V,T	T,V,P	T,V,P
32	float	none	1.45	7.4	5.11	P,T,V	T,P,V	T,P,V
32	float	SSE3	1.41	3.76	2.67	V,P,T	T,P,V	T,P,V
32	float	AVX2	1.56	3.11	2.0	P,V,T	T,V,P	T,V,P
32	double	none	1.53	8.34	5.44	P,T,V	T,P,V	T,P,V
32	double	SSE3	1.53	8.52	5.56	V,T,P	T,P,V	T,P,V
32	double	AVX2	1.54	7.0	4.56	P,T,V	T,V,P	T,V,P

need more space for the same amount of elements. As for the *SSE* and *AVX* combinations, we have tremendous gain while using it compared to *None*, as it uses vectorial capabilities of the used core. Using a leverage usually comes with a cost. This statement is also true for the *Vectorization* leverage. An operation on vectors has costs, even if it is low. For instance, it is known that loading and saving vectors has a non null cost. With only one active thread, the current architecture, Broadwell here, allows turbo boost, a technology that permits to reach a much higher frequency that the available ones (here it can reach 3.0 GHz, when average frequency is 2.1 GHz). Also, when the OS detects too much load on a core, it context switches the running process and runs it on another core. Hence, the kernel saves the states (stack, registers) of the current process and loads it on another core, implying a storing and loading cost of the given process. This phenomenon can happened several times during a second. Thus, saving and charging states can create a lot of cache misses, which could be dramatical with usage of vectorization, where loading and saving vectors is not free. As *AVX* has longer vectors, its operation costs on vectors can be longer than SSE. Thus, it starts to be beneficial only when comparing *double* combinations for such a *Vectorization* leverage.

When threads are up to 32, data is more likely to be shared between caches of various used cores. Without the previous struggles from caches for one core and because it is also well known that floating points operations(*float* and *doubles* here) are well optimized on current architectures and perform better than integers, {32, float, none} and {32, double, none} perform better than {32, int,

none}. All threads are sharing data on separated cache, *SSE* and *AVX* outperforms the none configuration, with *AVX* always outperforming *SSE* for a fixed combination. Due to this data repartition between caches implied by the chosen configuration of the *nbThreads* leverage, there is enough computation to overcome costs of larger vector operations, here *AVX* for all combinations.

Note that the best combination for all metrics used here is always the {32, int, AVX2} combination. This result is the best combination to choose only if we have no constraints about leverage choices. It is expected to see variation, as leverages highly modulate the usage of nodes, either from intensity of usage for example of caches, core usage, availability of specific leverages (like seen with turbo boost with one thread). Results of metrics from combination of leverages is thus complicated to fully understand without a detailed knowledge of the architecture, the underlying used leverages and their influences on a given context. We propose predicates that helps a user underline such interesting points of interest from the table of leverages. For example, this table could help a user to choose a combination taking into account a fixed leverage state. Or to answer the following question: is there a leverage or a state of leverage that is always better for a given metric?

5 Exploiting the table of leverages

In this section, we describe the main contribution of this paper: a methodology to exploit the table of leverages and to extract useful knowledge, such as the influence and impact of one or multiple leverages on a given metric or set of metrics. We propose two focuses for extracting a score for each leverage. The first one corresponds to the actual table: it normalizes the results of a given metric for every explored configuration. The second one computes a ratio of contribution for each leverage in order to expose the most relevant leverage (the one with the largest contribution to the considered metric). We define four exploitation predicates that ease the analysis of the table, and answer questions. We illustrate these predicates and the answers of these questions on the selected table (Table 1). These questions target a single metric, h_u.

Question 1: Is a selected combination of leverages states the best one for metric h_u? If a given combination is always the best, it means it should always be applied, if possible, if one wants to optimize h_u. Consider a combination of states x_a, y_b, z_c of leverages χ, ψ, ω for metric h_u. We need to check whether for all $i \in [0, \ldots, n_x] \backslash \{a\}, j \in [0, \ldots, n_y] \backslash \{b\}$, and $k \in [0, \ldots, n_z] \backslash \{c\}$, we have $h_u(x_a, y_b, z_c) \leq h_u(x_i, y_j, z_k)$. On Nova nodes and for the three leverages (Table 1), the best combination for all three studied metrics is {32, int, AVX2}.

Question 2: When I fix a state, do I always improve metric h_u? Consider state x_a of leverage χ. We want to check whether for all $i \in [0, \ldots, n_x] \backslash \{a\}$, for all $l, j \in [0, \ldots, n_y]$, and for all $m, k \in [0, \ldots, n_z]$, we have $h_u(x_a, y_l, z_m) \leq h_u(x_i, y_j, z_k)$. On the example of Table 1, for the $Joules$ and $Time$ metric, only the n_{max} (here, 32) state of *nbThreads* leverage answers this predicate, meaning that using this state will always be beneficial. No specific results can be obtained

with this question for the $avrgWatt$ metric, meaning that no leverage state is always better for this metric when used.

Question 3: If some states are fixed for a subset of leverages, is a given state for the remaining leverages the best choice to optimize h_u? Consider that the state of leverages ψ, ω is fixed to y_b, z_c. We are asking whether state x_a of leverage χ is the best choice for metric h_u. Therefore, we need to check whether for all $i \in [0, \ldots, n_x] \backslash \{a\}$, we have $h_u(x_a, y_b, z_c) \leq h_u(x_i, y_b, z_c)$, which tells for instance that for the fixed combination {32, SSE3}, the best state for the *Precision* leverage is *float*, when considering the $Joules$ or $Time$ metric (Table 1). Although, when focusing on $avrgWatt$ as the studied metric, for the {32, SSE3} fixed combination, the best state for the *Precision* metric is *int*.

If only state z_c for leverage ω is fixed, and we consider states x_a and y_b of leverages χ and ψ respectively, we check whether for all $i \in [0, \ldots, n_x]$ and for all $j \in [0, \ldots, n_y]$, we have $h_u(x_a, y_b, z_c) \leq h_u(x_i, y_j, z_c)$. Concerning the $Joules$ metric (Table 1) for the fixed state *float* of the *Precision* leverage, the best combination for the *nbThreads* and *Vectorization* leverages is {32, AVX2}. However, for the $avrgWatt$ metric, fixing again the state *float* of the *Precision* leverage, the best combination is now {32, SSE3}.

Applying this predicate allows us to extract some unexpected results. Concerning the $Joules$ and $Time$ metrics, for the *Precision* and *Vectorization* leverages, no state emerges as the best one. In fact, it highly depends on the chosen state of other leverages. One could for instance expect *int* to always be the best state, but when comparing the {32, double, none} with {32, int, none}, we see that the *double* combination is more effective than the *int* combination. Similar conclusions can be drawn when the *Vectorization* leverage is used. *AVX2* has larger vectors than *SSE3*, thus we would expect it to be always more efficient. However, when *nbThreads* state is equal to 1, {1, float, SSE3} is more effective than {1, float, AVX2}, leading to a different best choice when combined to the n_{max} state (here, 32), where {32, float, AVX2} is more effective than {32, float, SSE3}. Note that this combination emerges as the best one when *SSE3* is fixed.

Concerning the $avrgWatt$ metric, we also get unexpected knowledge. In opposition to the $Joules$ and $Time$ metrics, no state emerges as the best one for none of the studied leverages. As *AVX2* has larger vectors than *SSE3*, we would expect it to always stress more the CPU, thus always having higher values for this metric. It is the case with the {32, float} and {32, double} combinations. However, it is not observed with other combinations. When *nbThreads=1*, *int* is always the best choice to minimize this metric, whatever the chosen state for *Precision* and *Vectorization* leverages. Moreover, when *Vectorization* and *nbThreads* are set to any studied states, *int* is also always the best choice to minimize the $avrgWatt$ metric.

Question 4: Given a combination for all the leverages, how can we rank the states in terms of contribution for metric h_u? To answer this question, we consider a set of states x_a, y_b, z_c of leverages χ, ψ, ω. Then, for each state $w \in \{x_a, y_b, z_c\}$, we compute the contribution score $mc(w)$ for this state on

metric h_u as follows. For state x_a of leverage χ, $mc(x_a) = \frac{h_u(x_a,y_b,z_c)}{\max\limits_{i\in[0,\dots,n_x]} h_u(x_i,y_b,z_c)}$. We define similarly the contribution of states for the other leverages ψ and ω. Then, we rank the contribution scores $mc(x_a)$, $mc(y_b)$, $mc(z_c)$ in ascending order to answer the question.

Table 1 (last three columns) presents the scoring related to the table of leverages. For the best combination {32, int, AVX2}, the ranking goes as follows for the *Joules* metric: "T,V,P" or "*nbThreads, Vectorization, Precision*", meaning that the chosen state for T here is the most contributing state in this combination, followed by the V, and then P states. Thus, for this combination, the precision leverage with the *int* position has the lowest contribution.

This ranking points out unexpected results for the *Joules* metric. We notice a switch between two positions of a given leverage for the fixed combination of other leverage states: {32, double}. In fact, when comparing the scoring of {32, double, SSE3} with {32, double, AVX2}, we get respectively "T,P,V" and "T,V,P". In the first case, *double* and *SSE3* have the same worst possible score, 1.0, meaning that it is the worst state of this leverage for this combination. In the second case, *AVX2* scores better than *SSE3* and thus, it is above *double*. When *nbThreads=1*, we note that combinations including *SSE3* and *AVX2* states always have the *Vectorization* leverage state as the most contributing one, which leads to the conclusion that it is always better to use *SSE3* and *AVX2* states for the *Vectorization* leverage. For the {32, float, SSE3} combination, we get the scoring "T,P,V". *float* gets a better score and thus a better position than *SSE3* because it is the best leverage state for the {32, SSE3} combination, leading to the conclusion that choosing *float* instead of other *Precision* leverage states contributes more than choosing *SSE3* instead of other *Vectorization* leverage states for this combination. For the *avrgWatt* metric, scoring underlines the fact that when choosing *int* as a state of *Precision* leverage, and for a fixed state of the *Vectorization* leverage, the sorting is always the same. In fact, {32, int, none}, {32, int, SSE3} and {32, int, AVX2} get the exact same sorting of contribution that {1, int, none}, {1, int, SSE3} and {1, int, AVX2}, respectively. Moreover, *int* is always the most contributing leverage state, which shows that *int* is always a good choice to improve this metric. This scoring also underlines the fact that in order to minimize the *avrgWatt* metric, a user should better focus on P and V leverages, asT is never the most contributing one. This scoring highlights results that would have been difficult to notice just by looking at the table. It allows a user to quantify how much a leverage position used in a combination contributes to the overall performance for a given metric.

6 Conclusion

Energy efficiency is a growing concern. In the context of HPC and datacenters where the size of infrastructures grows drastically, energy consumption has to be taken into account as a high expense. There is a wide range of techniques, that we formally define as leverages, that permits to modulate the computing capabilities and/or the energy/power used by a device. We propose a generic solution to extract fine grain knowledge and hints from the table of leverages,

thanks to the defined predicates. Our solution underlines new knowledge about leverages alone and about combinations of leverages. Thus, it allows us to extract influences of leverages on each other and understandable knowledge by the user.

Knowledge could be extracted from a table on CPU-intensive workload. For example, our solution underlines the fact that if *Precision* is set to the *double* state, it is always better to use it with *AVX2* state for the *Vectorization* leverage to minimize the *Joules* metric. Also, for *Vectorization* fixed to the *SSE3* state, our solution tells us that *float* is the best state to minimize the *Joules* metric. We also underline the fact that some unexpected behavior can be seen when combining leverages. For example, we underline the fact that changing *float* or *int* to *double* for *Precision*, and keeping the *SSE3* state activated for *Vectorization* state, turns out to be counterproductive for the *Joules* metric.

The first short term future work is the parallelization of the creation of the table of leverages in order to improve the time needed to build it. Then, we plan to apply this methodology on other non CPU-intensive phases, such as IO, HDD, and RAM-intensive phases with appropriate leverages for every phase. Finally, a future working direction would be to extend this methodology to costly transition leverage states, as for instance shutdown policies. Also, we would like to investigate how to reduce the completion time for building such a table. In fact, the time to solution here could be greatly reduced, for example by predicting which run is not needed to know values of relevant metrics using learning or prediction techniques.

References

1. H. Acar, G. I. Alptekin, J.-P. Gelas, and P. Ghodous. Towards a Green and Sustainable Software. In *Concurrent Engineering*, pages 471–480, 2015.
2. I. E. Agency. Digitalization & Energy. White paper, 2017.
3. D. Balouek et al. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367, pages 3–20. Springer, 2013.
4. G. L. T. E. A. Chetsa. A user friendly phase detection methodology for hpc systems' analysis. In *IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, 2013.
5. L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, pages 46–55, 1998.
6. B. Gallas and V. Verma. Embedded Pentium (R) processor system design for Windows CE. In *Wescon/98*, pages 114–123. IEEE, 1998.
7. Y. Georgiou, D. Glesser, K. Rzadca, and D. Trystram. A scheduler-level incentive mechanism for energy efficiency in hpc. In *CCGrid*, pages 617–626, 2015.
8. C. Lomont. Introduction to intel advanced vector extensions. *Intel White Paper*, pages 1–21, 2011.
9. A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE micro*, 16(4):42–50, 1996.
10. I. Rais, A.-C. Orgerie, and M. Quinson. Impact of Shutdown Techniques for Energy-Efficient Cloud Data Centers. In *ICA3PP*, Dec. 2016.
11. D. Suleiman, M. Ibrahim, and I. Hamarash. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In *International Conference on Electrical and Electronics Engineering*, 2005.