Energy consumption comparison of parallel linear systems solver algorithms on HPC infrastructure

Sofia Montebugnoli*
sofia.montebugnoli3@unibo.it
University of Bologna
Bologna, Italy

Anna Ciampolini*
anna.ciampolini@unibo.it
University of Bologna
Bologna, Italy

ABSTRACT

High-Performance Computing (HPC) systems today are gradually increasing in size and complexity due to the correspondent demand for ever-increasing computing needs, requiring more complicated tasks and higher accuracy. The growing energy needs of HPC systems require the urgent adoption of green HPC approaches to mitigate environmental impact and promote energy-efficient computing.

This paper proposes a monitoring solution for the energy consumption during the execution of parallel software focusing in particular on the solution of linear systems in HPC systems: the Inhibition Method and Gaussian Elimination from ScaLAPACK library. The main goal is to profile their execution from the energy consumption perspective. The approach follows a white-box paradigm, injecting the monitoring component into specific ranks. Despite a slight overhead compromise due to synchronization, this design permits accurate measurements.

KEYWORDS

Linear systems solver, HPC systems, energy efficiency, energy measurements, energy consumption, Inhibition Method, ScaLAPACK, Green HPC

ACM Reference Format:
Sofia Montebugnoli and Anna Ciampolini. 2023. Energy consumption comparison of parallel linear systems solver algorithms on HPC infrastructure. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3624062.3624266

1 INTRODUCTION

Nowadays the reduction of energy wasting is a core issue for many companies, especially for the most energy-consuming, like data centers. Since the advent of Green IT, i.e. the study and practice of environmentally sustainable computing or IT, the interest in achieving more efficient High-Performance Computing (HPC) systems has increased. For example, the Green 500 lists the world's most energy-efficient supercomputers, based on floating point operations per second (flops) per watt metric [1]. Consequently, there is a compelling need to substantially increase the budget allocation to procure the necessary energy to power these supercomputers. Thus, attaining an efficient power usage effectiveness index assumes paramount importance, as it not only fosters the development of more sustainable HPC systems but also contributes to tangible cost reductions [13]. Until 2013 the power budget of the Top500[2] for supercomputers has increased, but in the last years directly raising the performance had the effect of increasing the power budget, and consequently the economic budget to power these HPC systems. Thus, as it is impossible to increase energy consumption indefinitely, energy efficiency is a core issue.

The primary objective of this proposal is to design and test a monitoring solution to carefully track the energy consumption of linear systems resolution algorithms on HPC architectures. Through the integration of this solution, with the parallel code to be monitored, our aim is to gain profound insights into the energy utilization patterns during the execution of parallel algorithms. This approach will facilitate the identification of potential ways for energy optimization and efficiency enhancement, ensuring the execution of only indispensable instructions, thus mitigating the waste of computational resources and energy. The monitoring system will comprehensively capture and analyze data pertaining to energy consumption, allowing for a thorough assessment of the energy usage profile of these algorithms under diverse conditions. Being aware of these results, programmers could take informed decisions to augment the energy efficiency of linear systems resolutions, contributing to the development of more ecologically sustainable HPC systems and yielding tangible cost reductions.

Some of the most complex problems can be represented through matrices and their resolution is represented by the solution of the matrix as a linear system. Hence, the study of linear system solver algorithms is important to solve a wide set of problems from many fields with real-world applications. These linear systems involve thousands of equations, that can be managed efficiently from parallel architectures like HPC systems. With the purpose of improving this class of algorithms, it can be helpful to study their performances on the HPC systems by monitoring the execution of these tasks.

In particular, this work tests the monitoring solution on two different parallel algorithms for the resolution of linear systems: the Gaussian Elimination and the Inhibition Method IMe [10]. Both algorithms were tested with the proposed framework, in order to measure and compare either performance or energy efficiency, thus proposing a new perspective for the evaluation of linear systems solution methods.

*Both authors contributed equally to this research.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624266

Paper outline. Accordingly, the remainder of this work is structured as follows. Section 2 covers the definition of the proposed algorithm for linear systems resolution above an HPC architecture. Section 3 analyzes other linear programming solvers for HPC systems. Section 4 focuses on how the Message Passing Interface and the Performance API should integrate to monitor the designated metrics, moreover, it explains how the monitoring program is integrated and implemented in the code of the linear systems solver algorithms. Section 5 specifies the parameters of the performed tests, and the obtained results, through charts. The parameters consist of a set of different variables for the execution which are tested during the monitoring phase of the algorithm. Whilst, the results of the several executions are shown and reviewed in the second part of the section. Finally, section 6 concludes this work by summarising its main contribution and presenting eventual future developments.

2 BACKGROUND

The breakdown of the effectiveness of Dennard's scaling around 2006 led to the inability to significantly increase the clock frequencies, therefore most CPUs manufacturers have focused on multicore processors as an alternative way to improve performance. An increased core count benefits many workloads, but the increase in active switching elements from having multiple cores still results in increased overall power consumption and thus worsens CPU power dissipation issues [11] [14]. The result is that the total power consumption of each device limits the practical achievable performance. Moreover, due to the overheating of the cores, the cooling systems consume more additional power, and this impacts directly the power budget available for the data center. When a program is executed on HPC systems using a parallel paradigm, improving the efficiency of the algorithm is a core issue. In fact, all those problems which are CPUs intensive like linear systems resolution algorithms and aim to solve complex problems often require days of computation and consequently require a lot of energy, computational resources, and money.

This section offers a comprehensive exposition of key topics essential to our investigation. It delves into the IMe and ScaLAPACK algorithms and their consequent impact on energy consumption. Moreover, the section provides an overview of energy measurement and saving principles in HPC systems, along with an extensive list of monitoring tools dedicated to optimizing energy efficiency. These tools, such as Model Specific Registers (MSR), Running Average Power Limit (RAPL), and Performance Application Programming Interface (PAPI), enable the monitoring and evaluation of energy usage patterns, facilitating the identification of areas for improvement and energy optimization. Armed with this academic foundation, our research endeavours to explore energy consumption during linear systems resolution, contribute to energy-efficient computing, and promote the development of sustainable HPC systems.

The Gaussian Elimination also known as row reduction is the most efficient algorithm for solving systems of linear equations both in a parallel and in a sequential form with an arithmetic complexity of $2/3n^3 + O(n^2)$. Whereas, the Inhibition method in the last available version has reached a complexity of $3/2n^3 + O(n^2)$ so far [15]. However, recently it was proved [7] that IMe has a good integrated low-cost multiple fault tolerance, which is more efficient than the checkpoint/restart technique usually applied in Gaussian Elimination linear systems resolution. Therefore, deep analysis and understanding of IMe can lead to significant contributions to the available algorithms for linear systems resolution and further optimizations. Both the Gaussian Elimination and the Inhibition Method have a parallel version which is implemented by the Netlib organization in the ScaLAPACK library in the first case, and by ENEA and University of Bologna in the second case.

2.1 Inhibition Method

The Inhibition Method was proposed in 1963[10] to simplify the analysis of complex electric circuits. Then it turned out to be useful also in the resolution of physical linear systems and square matrix inversion. This algorithm is general because it can be applied to every linear system or non-linear system, and it is independent from the physical nature of the system itself. The Inhibition Method (IMe) is an iterative, exact, non-inverting method to solve any linear system. It derives from the Cross method, from which IME inherits the fundamental characteristic of decomposing the problem into easier-to-solve sub-problems, even though the cross method is non-exact[6]. IME produces a hierarchical sequence of sub-systems, at the end of which only elementary systems can be found. Hence, they can be solved rapidly with the minimum of knowledge, consequently, only a few program code lines are needed.

IMe starts from considering the linear system with n equations and n unknowns in its matrix form: $Ax = b$, where A is the $n \times n$ matrix of coefficients, b is the vector of constant terms and x contains the unknowns. First, it prescribes to compute a matrix T (n), called inhibition table, and a vector $h(n)$ of elements, called auxiliary quantities. $T(n)$ is built using only the $a_{i,j}$ coefficients from A as follows:

$$T^{(n)} = \left[\begin{array}{cccc|cccc} \frac{1}{a_{1,1}} & 0 & \dots & 0 & 1 & \frac{a_{2,1}}{a_{1,1}} & \dots & \frac{a_{n,1}}{a_{1,1}} \\ 0 & \frac{1}{a_{2,2}} & \dots & 0 & \frac{a_{1,2}}{a_{2,2}} & 1 & \dots & \frac{a_{n,2}}{a_{2,2}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \frac{1}{a_{n-1,n-1}} & 0 & \vdots & \vdots & \dots & \frac{a_{n,n-1}}{a_{n-1,n-1}} \\ 0 & \dots & 0 & \frac{1}{a_{n,n}} & \frac{a_{1,n}}{a_{n,n}} & \dots & \frac{a_{n-1,n}}{a_{n,n}} & 1 \end{array}\right]$$

The algorithm is divided into two parts: the first part is the INITIME procedure. It handles the initialization of the $T(n)$ matrix. The second part reduces iteratively the number of rows and columns. The matrix $T(n)$ and the vector $h(n)$ can be seen as a decomposition of the original problem into n sub-problems (one for each row). One of the best ways to compute an algorithm faster is to parallelize parts which are independent from the others, ascertaining that the amount of data replicated and exchanged by the nodes is minimized. In the following procedure, we will use $t^{(l)}$,j and $t^{(l)}$ i, to address the j-th column and the i-th row of $T^{(l)}$, respectively, and N as number of nodes, considering N-1 slaves and one master. Under certain conditions the fundamental formula allows an independent computation of each element of T. Precisely, three different parallelization schemes are possible:

i) column-wise, entailing that the node computing the last column $t^{(l)}_{*,n+l}$ should make it available to all the others, and all the nodes should share $h^{(l)}$;
ii) row-wise, symmetrically, the node computing the last row $t^{(l)}_{l,*}$ should make it available to all the others and $h^{(l)}$ is shared;
iii) block-wise, combining row-wise and column-wise parallelization.

As a matter of fact, the scheme used in the Inhibition Method Parallelized IMeP is column-wise because its characteristic fits the integration with the fault tolerance requirements better than the others. Each computing node works on a subset $T^{'}$ of T, by iteratively applying the fundamental formula. At every level it is also necessary to broadcast from the master to the slaves h, whilst the node in charge of the computation of the last column $t_{*,n+l}$ should broadcast it to all the other nodes, and besides only the n elements of the last row which result modified after the application of the fundamental formula must be sent to the master.

One of the concerns of parallelization is memory usage: by spreading the execution of the algorithm in N nodes the memory occupation increases from $2n^2 + 3n$ to $moIMeP = 2n^2 + 2nN + 3n$, whereas the flops remain the same. Furthermore, the distributed environment forces message exchange, which has a cost. The traffic generated by the exchanged messages is measured by the number of messages, and volume, meaning the number of floating points. The total number and volume of messages exchanged is:

$$M_{IMeP} = n^2 + 2(N-1)n + 2(N-1)$$

,

$$V_{IMeP} = (N+2)n^2 + 2(N-1)n$$

.

Which is the sum of:

– the broadcast of the last column $t_{*,2n}$ from the master to all the slaves, for the initialization.
– h is broadcasted from the master to all slaves.
– the node that oversees of the last column $t_{*,n+l}$ broadcasts it to all the other nodes.
– all the slaves send the last entry of their columns to the master. Only n elements of the last row are exchanged, because all the others are certainly 0.

2.2 Gaussian Elimination by ScaLAPACK

ScaLAPACK, an abbreviation for Scalable Linear Algebra PACKage [5], is a high-performance library of linear algebra routines designed for distributed memory computers that support the Message Passing Interface (MPI). The capabilities of ScaLAPACK encompass solving dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems. This library incorporates several key concepts to enhance its performance:

– A block cyclic data distribution for dense matrices and a block data distribution for banded matrices, which can be parametrized at runtime, allows efficient data distribution across distributed memory systems.
– Block-partitioned algorithms are utilized to promote significant data reuse, leading to improved computational efficiency.
– The library is thoughtfully designed with well-structured low-level modular components that facilitate the parallelization of high-level routines. This design maintains uniformity in the source code between sequential and parallel implementations.

Regarding the Gaussian Elimination method, to address the numerical instability arising from roundoff errors, the introduction of the Partial Pivoting technique becomes essential. This technique involves swapping rows such that the diagonal element $A_{(i,i)}$ is the largest in its column. Numerical instability arises when the diagonal element $A_{(i,i)}$ becomes exceedingly small (though not precisely zero), potentially leading to erroneous outcomes even if the algorithm terminates.

2.3 Tools and APIs for performance-energy power tracing

All the Intel CPUs offer power management interfaces that are not architectural but address the power management needs of several platforms' specific components. RAPL (Running Average Power Limit) interfaces provide mechanisms to enforce power consumption limits. RAPL interfaces consist of non-architectural Model Specific Registers MSR. The counters are 32-bit registers that indicate the energy consumed since the processor was booted up. The counters are updated approximately once a millisecond (due to jitter). The MSRs can be accessed directly on Linux using the MSR driver in the kernel. For direct MSR access, the MSR driver must be enabled, and the read access permission must be set for the driver. Reading RAPL domain values directly from MSRs requires detecting the CPU model and reading the RAPL energy units before reading the RAPL domain (i.e., PKG, PP0, PP1, etc.) consumption values.

Once the CPU model is detected, the RAPL domains can be read per package of the CPU by reading the corresponding 'MSR status' register.

Performance Application Programming Interface PAPI is an interface for accessing performance counters on different platforms in a common way. As each processor vendor defines different processor interfaces to the performance counters, PAPI was built to solve this problem and to handle requests to these counters in a comfortable way.

As for the development of PAPI the main goal was a common and convenient way to access performance counters on different platforms: PAPI is built upon different layers for a better abstraction of different tasks found in each layer as shown in Figure 1. The main layers are the Portable Layer which offers an API for tool and application developers and the Machine Specific Layer used to access performance counters on a given platform. A given platform consists possibly of a certain processor architecture, a certain operating system, available libraries or a combination of these.

The Portable Layer consists of the PAPI Low Level-API enabling a developer to access all core functions of PAPI and direct interaction with the counter interface on a given platform. The PAPI High Level-API defines only a fraction of functions compared to the

PAPI Low Level-API to access the counters, but these functions are enough to extract performance data using pre-sets events defined by PAPI.

The Machine Specific Layer handles all direct access to a given platform. The term direct access is meant to express access either to the counters on a platform directly or by using an operating system interface for accessing these processor-specific functions. The Machine Specific Layer also limits PAPI in its functionality, as PAPI supports many different platforms whereas some platforms do not support specific functionalities.

Between the Portable Layer and the Machine Specific Layer is the core functionality of PAPI with support for managing the counter access. Memory allocation, thread binding and event-related issues are handled here, invisible for the developer of a tool or application for performance counter instrumentation [9].

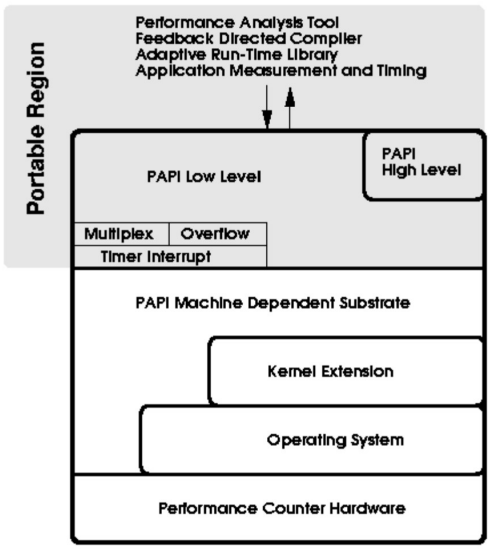

Figure 1: PAPI architecture, hierarchy of power domains [9]

3 RELATED WORKS

Alonso et al. [4] present a comprehensive framework aimed at profiling, monitoring, modelling, and analyzing power dissipation in parallel MPI and multi-threaded scientific applications. The framework comprises a custom-designed device for internal DC power consumption measurement and a package that offers a user-friendly interface to interact with this design, as well as compatibility with commercial power meters. However, their study does not include a comparative assessment of the ScaLAPACK algorithm's performance against other libraries. On a different note, McCraw et al. present a study [16] introducing two new PAPI components that enable power and energy monitoring for Intel Xeon Phi co-processors and the IBM Blue Gene/Q system, with an application on ScaLAPACK algorithms. This particular study focuses solely on the mentioned hardware architectures, omitting an overview of the hardware architecture used in our paper to test IMe and ScaLAPACK.

In another relevant work, Tan et al. [21] summarize widely deployed power management techniques in HPC systems, presenting power and energy models and two fundamental types of power management: static and dynamic approaches. Additionally, they review research on power and energy efficiency for high-performance numerical linear algebra algorithms, covering aspects like profiling, performance trade-offs, static and dynamic saving techniques. Although their study is comprehensive, they do not evaluate and test existing monitoring solutions on HPC architectures, which leaves room for further investigation.

The paper by Ilya Meignan–Masson et al.[17] introduces Colmet a monitoring framework, that bridges the gap between system monitoring and profiling. It emphasizes its dynamic sampling and reconfigurability as groundbreaking features of this work. Similarly, in the work presented by Bartolini et al. [8], they present a fine-grained monitoring solution that comprehends the application of AI techniques to measure and control power and performance. Agelastos et al. [3] developed a continuous, synchronous, whole-system monitoring on an HPC system. In this work, they especially focused on analytic and visualization techniques, that they used to characterize application and system resource usage under production conditions for a Rashti et al. [19] presents WattProf, a versatile power monitoring platform designed for precise and efficient power and energy monitoring across hardware components in HPC clusters. WattProf includes a programmable PCIe expansion card, component-specific sensors, a host runtime system, and a user-friendly API. The authors showcase WattProf's capabilities through two benchmark applications. However, these works do not tackle linear systems solver algorithms, indeed they represent general-purpose solutions for performance monitoring on exascale supercomputing systems.

4 OUR PROPOSAL

The impact of performance is of paramount importance for Green HPC goals and, consequently, a runtime monitoring system is needed to grant the collection of the energy values. In this section, we propose a monitoring solution for linear system solver algorithms by monitoring the energy consumption of CPU packages 0 and 1, as well as DRAM 0 and 1. These metrics respectively represent the total energy consumption of the CPU and the power consumed by the RAM.

To effectively monitor the entire execution process, the monitoring program must exhibit high portability, enabling seamless adaptation to various algorithms. Additionally, it should accommodate both white-box and black box approaches, introducing only minimal modifications. Ensuring the efficiency of the adopted monitoring solution is crucial, as any overhead caused by the monitoring libraries should not significantly impact the algorithm's performance, or if possible, should maintain a negligible impact. Moreover, a key requirement is the modularity of the monitoring system. Specifically, the system should be structured to initiate monitoring through a function call and conclude monitoring with another function call. While these parts work on the same arrays of events to modify values, direct interactions between them should be avoided.

The testing framework needs to cater to both simple and complex tests, offering flexibility in its support. Moreover, it is imperative that the framework automatically collects and stores results in a

human-readable format for subsequent review and analysis. It is essential that the testing process does not disrupt the structure of the tested algorithms or compromise overall performance. The test framework should be adaptable to different algorithms and function effectively across various scenarios. Furthermore, considering that tests will run on multiple nodes, and each node may exhibit different energy values, comprehensive data collection is vital. The proposed solution should demonstrate scalability to efficiently manage the execution on diverse nodes, ensuring reliable and accurate measurements.

The proposed solution follows a white-box approach by designating a single rank in each node to execute the monitoring task. This approach enables the collection of metrics pertaining to the overall energy consumption of the CPU package, processor die, and DRAM. Simplifying the node and rank relationship, where each node represents a CPU and each rank represents a core, dictates that only one rank per processor should be responsible for the monitoring process.

MPI communicators are utilized to establish groupings of ranks belonging to a single node, creating sub-communicators accordingly. By implementing sub-communicators, a specific rank can be appointed for the monitoring task. These designated ranks execute the monitoring process distinctly from the other ranks. Specifically, they initiate monitoring by initializing PAPI, commence measurements of the selected metrics, and then proceed with the assigned part of the linear system solver algorithm, similar to all other ranks. Subsequently, after all the ranks on the same node complete their respective computations, the monitoring ranks conclude the measurements. To ensure synchronization and alignment of execution across ranks within the same group, an MPI barrier of synchronization is introduced, requiring the monitoring ranks to wait for the processing ranks in their communicator.

It is crucial to emphasize that both the beginning and end of the monitoring process are consistently preceded by MPI synchronization barriers for ranks on the same node, enhancing the accuracy and correctness of the measurements. The adopted approach follows the "white-box" paradigm as the monitoring component implemented through PAPI is introduced into a specific rank for each node, necessitating the grouping of ranks to subsequently assign a monitoring rank.

This solution maintains its modular and portable structure while enhancing efficiency through differentiated rank execution. However, to achieve accurate measurements, a compromise is made regarding the time spent on synchronization, which, in turn, results in slower program execution and adds some overhead, not directly to the linear system solver algorithm, but to the overall execution.

An additional noteworthy characteristic pertains to the PAPI initialization. In the first solution, it occurs before the MPI initialization, when the program is still in a sequential state. In contrast, the second solution initializes PAPI exclusively in the designated monitoring ranks.

Figure 2 illustrates the general execution flow for each rank of the involved processors. Initially, ranks are categorized into monitoring and non-monitoring groups. The process of selecting monitoring ranks involves designating the rank with the highest value on each node as the monitoring rank. Following this step, node synchronization occurs to enable the monitoring ranks to commence collecting energy values from the processors. Subsequently, a general execution synchronization is performed to align the ranks for the linear system solver execution phase. This results in an additional node synchronization phase at the conclusion of the algorithm execution, which is necessary to halt the collection of energy values. Finally, all ranks are synchronized once more.

The proposed solution entails the designation of one rank per node to perform effective monitoring. After MPI_init() a new communicator for each is created through the method MPI_Comm _split_type(), through the constant split type MPI_COMM_TYPE _SHARED the ranks are automatically divided into a group based on their rank. The following step is to designate the monitoring rank, which is always the one which has the highest rank value in the communicator. If the rank is designated for monitoring it calls start_monitoring() from papi_monitoring.h. The values are passed as a reference because also the function end_monitoring() needs the event's values to process the monitoring measurements. The method start_monitoring() uses PWCAP_plot_init() to initialize PAPI. The functions for PAPI initialization perform the library initialization, the thread initialization, the creation of the event set and the addition of all the desired events. Eventually, start_monitoring() calls PAPI_start_AND_time() which starts PAPI monitoring. The algorithm is executed between start_monitoring() and stop_monitoring(). Before stopping the whole monitoring, ranks that run on the same node are synchronized to the MPI_Barrier(). When all the ranks have terminated the execution of the linear system solver algorithm the PAPI monitoring procedure is ended by the monitoring ranks. The function end_monitoring() from papi_monitoring.h stops PAPI event counters with PAPI_stop_AND _time(), and then it creates one file for each processor with file _management(). In each file are saved the values of PAPI event counters for the processor in which the node has run. The function PAPI_term() cleans up and destroys PAPI Event set. After that, the monitoring is completed and MPI_Finalize() is performed.

Since most of the RAPL events of interest are included in powercap event set, which also adds the power capping functionalities, the monitored events will belong only to powercap event set offered by PAPI. Therefore, the array event_names which is used to list the names of the monitored events in papi_monitoring.h will contain all the powercap event set displayed by PAPI. This array is a parameter of papi_event_name_to_code which translates the names into the macros of the PAPI library. Code available at [18].

5 EVALUATION

To facilitate comprehensive evaluations of the proposed monitoring solution, various aspects of the IMe and ScaLAPACK algorithms are rigorously tested, considering a multitude of parameters. hests are performed on CINECA infrastructure using supercomputer Marconi which mounts Intel OmniPath Cluster architecture, with 17PB of local storage. The performance of Marconi A3 comprehends forty-five racks, 3188 nodes and each one has 2 x 24-core Intel Xeon 8160 CPU (Skylake) at 2.10 GHz, therefore the total cores per node are 48, whilst a 192 GB of DDR4 RAM is provided to every node. A single node can reach a peak performance of 3.2 TFlop/s, whereas the overall MARCONI A3 architecture can achieve a 10 PFlop/s peak [20]. The supercomputer batch job submission is managed

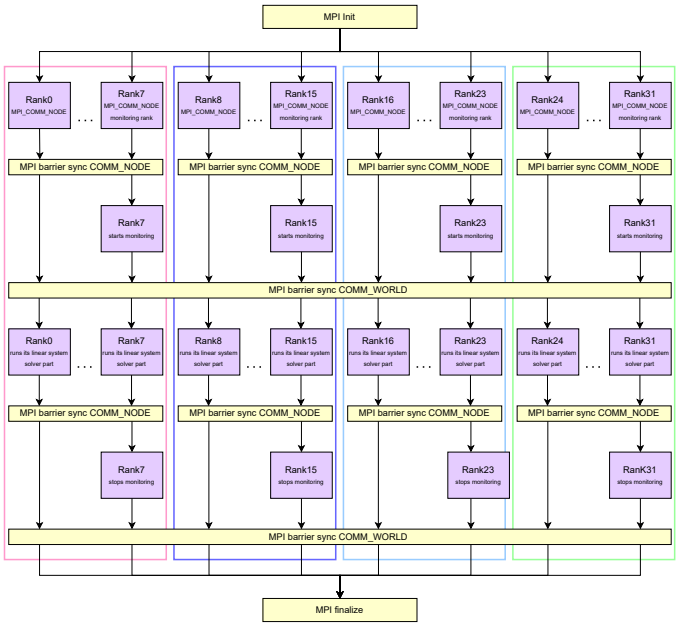

Figure 2: Structure of the MPI program in the common nodes solution

through Slurm, thus the collected energy values concern only the processors directly involved in the computation.

5.1 Tested parameters

The monitoring process primarily focuses on measuring energy status values. However, there are many combinations of the parameters that can affect the measurements.

Regarding the optimization level of the compiler (CFLAG), all tests are set at O3, the highest achievable level. While higher optimization levels (e.g., -O3) increase compilation time and executable efficiency, they decrease execution time significantly.

The matrices allocation is tested in a contiguous form, which enhances processing speed by reducing head movements through buffered I/O and consecutive memory block reads by the operating system.

The input linear system is not generated at runtime but loaded from a file to ensure consistent input data for repetitive measurements, particularly when executing a large number of tests. Four different matrix dimensions (8640, 17280, 25920, and 34560) are utilized to observe energy trends with fixed dimensions for ranks and nodes.

The chosen linear system solver algorithms (IMe and ScaLAPACK) are tested under identical conditions, and to achieve realistic values for comparison, ten repetitions for each job are performed.

The algorithm is divided into two phases: matrix allocation and execution. Monitoring the entire execution, including allocation, deallocation, and execution, yields an estimation of energy consumption for allocation and deallocation, conserving resources not crucial for this study.

The number of nodes and ranks significantly impact computation when the matrix dimension is fixed. There is a trade-off between execution time (increasing with load per rank) and energy consumption (powering more ranks on different nodes). Evaluating the energy-saving potential between assigning 48 ranks per node versus 24 ranks per processor across doubled nodes is essential.

Ranks	Nodes	Ranks per Node	Sockets	Ranks x Socket	
144	3	48	2	24	24
	6	24	1	24	0
	6	24	2	12	12
576	12	48	2	24	24
	24	24	1	24	0
	24	24	2	12	12
1296	27	48	2	24	24
	54	24	1	24	0
	54	24	2	12	12

Table 1: test configurations for nodes, ranks and sockets

The tests also explore socket distribution inside the processor, with 48 cores divided into 2 sockets of 24 cores each. Different configurations with 12 cores in each socket or only one socket utilized while the other remains idle are examined.

The strong scalability observation focuses on how energy consumption varies with the number of processors for a fixed problem size. For each matrix dimension, three specific rank values (144, 576, and 1296) are used, which are related to the matrix dimension and fulfil IMe's square number of ranks requirement. The table in the paper summarizes hardware configurations with the number of processors, ranks, and included sockets. The smallest number of nodes is computed by dividing ranks by 48 and multiplying the result by two for 24 cores per processor, while the most parallelized involves 54 nodes with 24 ranks each. Table 1 shows the result of the combination of the parameters presented above.

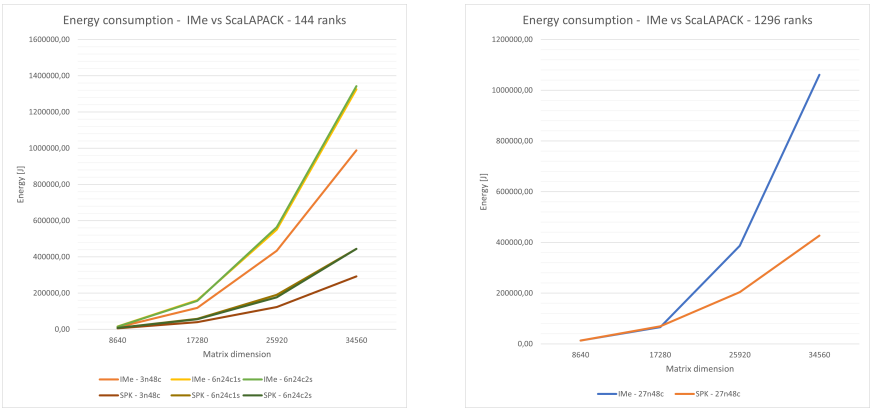

Figure 3: Comparison between full loaded processors and half loaded processors

5.2 Test results

The objective of this section is to highlight the trends in energy consumption and duration across various configurations while underscoring the distinctions between IMe and ScaLAPACK. As the results did not reveal significant differences between the monitored phases, the graphs exclusively present the values from the general execution phase of both IMe and ScaLAPACK. Given the extensive number of diverse tests and configurations, the graphs exhibit a condensed representation of more substantial data aggregations through combined charts.

IMe vs ScaLAPACK full loaded processors and half loaded processors. The charts displayed at 3 show the behaviour of energy consumption in the full and half-loaded processor. The fully loaded processor involves all 48 cores in the computation, each of them is assigned to one core. Whereas, the half-loaded is deployed in two different ways: in one case one socket is full with 24 cores running 24 ranks and the other socket is empty, in the other case there are 12 ranks per socket. From this graph is possible to note the difference in terms of energy consumption between the three configurations for IMe and ScaLAPACK. The full load configuration always consumes less than the other ones. Moreover, there are only slight differences between the configuration that deploys 24 cores on one socket and the one that distributes 24 cores on two sockets. In fact, the lines overlap multiple times and for both IMe and ScaLAPACK, thus is impossible to determine the best one.

Total energy consumption and time duration for fixed ranks sizes. The total energy consumption and the duration of the execution increase with the dimension of the input matrix. In particular, in these charts, there are the values obtained for the 48 core deployments on 3 nodes, 12 nodes and 27 nodes. It is evident that the energy consumption of IMe is always equal to or higher than ScaLAPACK. The trend of energy consumption seems to be exponential, it increases faster with the linear progression of the matrix dimensions, for both the algorithms. From these charts 4 is possible to note the dependency between the energy consumption and the duration, which clearly follows the same course for all the ranks deployments.

Total Energy consumption and time for different matrices dimensions. These charts 5 compare the power consumption of the three different node configurations for a given matrix size. In particular, in these charts, there are the values obtained for the 48 core deployments on 3 nodes, 12 nodes and 27 nodes. In this case, there is not a clear pattern for the energy values. On one hand, ScaLAPACK tends to assume a linear trend, whereas IMe values do not follow a specific trend. However, these charts clearly display the strong scalability behaviour of the problem for both algorithms. In fact, the time duration decreases with the increase of the number of ranks on which is deployed the algorithm. The course of the duration is inversely proportional. As far as a comparison between IMe and ScaLAPACK is concerned, it is clear that ScaLAPACK is faster in the more dense computations, whilst IMe is faster than ScaLAPACK in more distributed computations, like for 576 and 1296 ranks for matrix dimensions 8640 and 17280.

Total energy consumption and power for different ranks. These charts 6 compare the power consumption and the energy consumption of the three different rank configurations, by varying the matrix dimension. Since the power consumption is obtained by dividing the energy in Joules, with the duration of the execution, the result is a constant almost horizontal line between the various matrix sizes. As a matter of fact, power values, represented by the lines, reveal the actual difference between IMe and ScaLAPACK per second. With reference to the values of the secondary vertical axis, the power values of IMe and ScaLAPACK differ by 12% to 18%.

Total energy consumption and power for different matrices dimensions. The charts in 7 show the trend of the energy and power consumption by varying the number of ranks for a fixed matrix dimension. The energy consumption, in this case, does not show a clear trend. However, it is clear the dependency of power from the deployed number of ranks. The values of power consumption of IMe and ScaLAPACK are similar for the different rank values and strongly follow a directly proportional course. Hence, it can be noticed that the power values enhance the real trend of energy consumption.

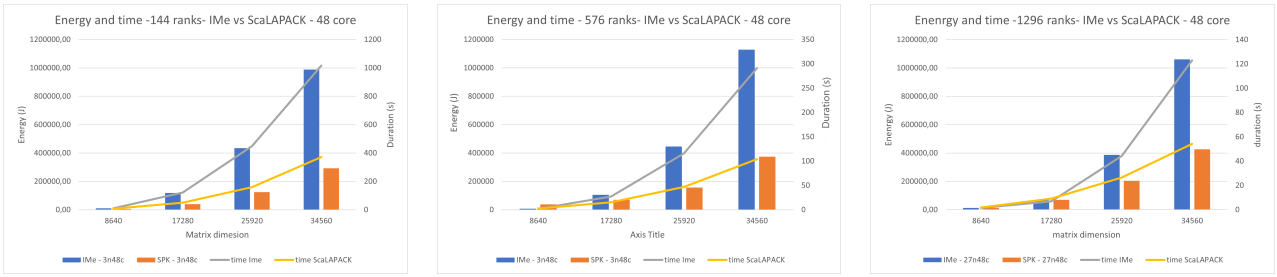

Figure 4: IMe and ScaLAPACK energy and time at fixed ranks size

Figure 5: IMe and ScaLAPACK energy and time at fixed matrix size

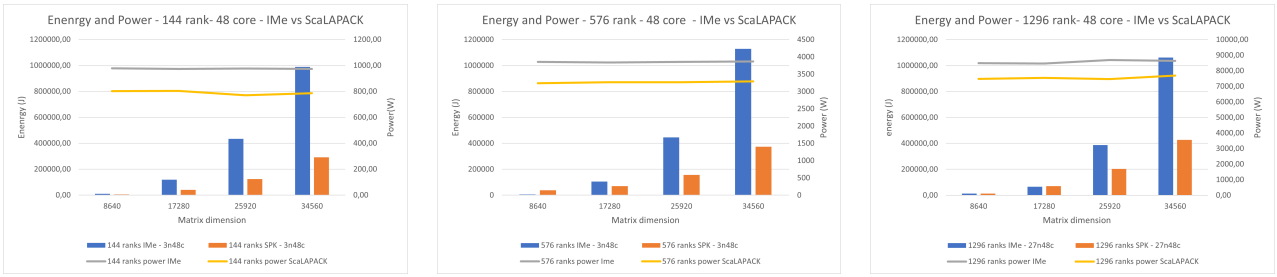

Figure 6: Energy and power consumption of IMe and ScaLAPACK at fixed ranks size

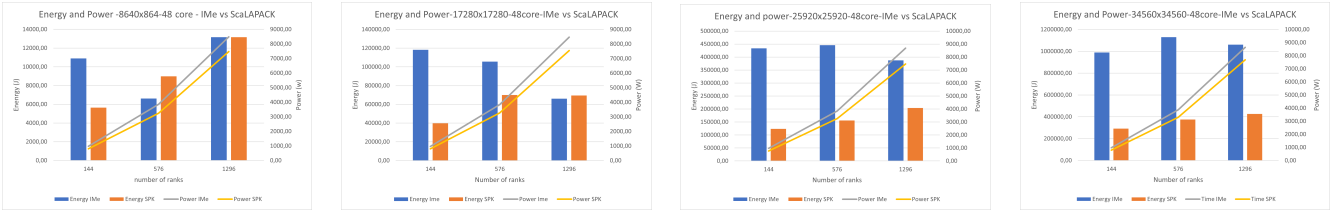

Figure 7: Energy and power consumption of IMe and ScaLAPACK at fixed matrix size

5.3 General Observations

The data pertaining to the general execution and the computation phase of the algorithm do not exhibit significant differences. In some cases, the execution of the algorithm alone consumes even more energy than the entire execution process. This discrepancy could be attributed to variations in the processors used for each execution, thereby limiting the precision of such comparisons. To enhance measurement accuracy, working consistently on the same nodes and controlling other parameters of the surrounding environment would have been beneficial. Surprisingly, the energy values for the general monitoring and the computational phase pertaining to the allocation and deallocation of matrices (which heavily impact DRAMs) do not demonstrate a marked difference.

Computations performed on 48 cores are more energy-efficient compared to the execution with 24 cores per node. The anticipated

increase in computation time for 48 cores per node due to interactions among multiple tasks sharing the same resources is evident only in the case of IMe for the 34560x34560 matrix in the 144 ranks deployment. However, the data in other scenarios do not consistently corroborate this behaviour; at times, the energy values for 48 cores computations are similar to those for 24 cores deployments. Nonetheless, in most cases, working with a full load per node proves more energy-efficient than half load per node deployment.

Unexpectedly, the behaviour of deployments on one socket or two sockets for the 24 cores solutions is quite similar, with insignificant variations. While the one-socket deployment utilizes only one package of the processor, the energy consumption of the second socket was expected to be low. However, this is not the case, and instead, the energy consumption of one socket is 50-60% lower than the other. This observation raises some doubts about the effectiveness of the Slurm directives. Additionally, the energy values for package 0 and package 1 in the 12 cores per socket solution were expected to be similar and significantly less than the energy consumption of package 0 in the one-socket deployment.

5.4 Summary Comparison between IMe and ScaLAPACK

Regarding the duration, the values for IMe and ScaLAPACK are heavily influenced by the number of ranks. When the matrix is distributed across many ranks, with each receiving a small part of the problem, IMe performs better than ScaLAPACK. However, if each task on each rank has a larger dimension, ScaLAPACK outperforms IMe.

In terms of total energy consumption, ScaLAPACK consumes less energy than IMe, with a consistent gap of 50% to 60%, except for a few cases where the values are quite similar. The gap tends to decrease with an increase in the number of ranks and a decrease in the matrix dimension, but the overall trend indicates that ScaLAPACK is more energy-efficient.

Notably, the gap in power consumption between the DRAMs of IMe and ScaLAPACK is even more significant, ranging from 12% to 18%. ScaLAPACK appears to respond better to deployments with fewer ranks. For instance, with ranks set to 144, the gap in DRAM power consumption between IMe and ScaLAPACK reaches 42%. In the 24 cores deployments, both with one and two sockets, the gap is around 33

Both algorithms demonstrate a considerable gap between package 0 and package 1 energy consumption in configurations involving 24 cores deployments. While this behavior affects ScaLAPACK only in terms of packages, IMe experiences a similar impact on both packages and DRAMs. Nonetheless, the difference in energy consumption between package 0 and package 1 is approximately 50

In summary, IMe exhibits higher energy consumption compared to ScaLAPACK, which can be partly attributed to its longer execution duration. However, the computation of power consumption confirms the gap between the two, albeit at a reduced margin of around 12% to 18%.

6 CONCLUSION

The primary objective of this research was to measure and analyze the energy consumption of parallel linear systems solvers on HPC systems. This goal has been achieved by developing a testing framework based on the PAPI library that has allowed us to obtain experimental data about the energy consumption of two linear system solver algorithms, IMe and ScaLAPACK, during execution on HPC systems. The study aimed to identify the algorithm with better energy efficiency and explore the influence of factors such as the number of ranks, nodes, and sockets on energy consumption.

The proposed solution grants observability of the energy values, ensures portability, accommodates the white-box approach and maintains modularity. It prioritizes efficiency to minimize the overhead caused by monitoring libraries, while a testing framework collects and stores results automatically for analysis. The solution designates a single rank per node for monitoring, employing MPI communicators to group ranks and synchronize measurements. The approach follows the white-box paradigm, injecting the monitoring component into specific ranks. Despite a slight overhead compromise due to synchronization, this design permits accurate measurements. In this way we can obtain valuable insights into energy efficiency and algorithm behaviour, thanks to a robust monitoring framework for comprehensive evaluations.

The results revealed unexpected behaviours of the energy values, as the energy consumption during the execution phase and the entire algorithm, including matrix allocation and deallocation, exhibited minimal differences. Deployments on fully loaded processors consistently proved less energy-consuming than half-loaded ones, particularly evident in DRAM energy values. However, patterns in the distribution of ranks on half-loaded processors (one socket or two sockets) were less pronounced, with the first socket generally consuming more energy than the second. Duration-wise, fully loaded deployments were quicker for smaller matrices, while larger matrices favoured half-loaded configurations due to reduced data interactions and more distributed computations.

In terms of energy consumption, ScaLAPACK generally outperformed IMe, especially in lower-rank deployments. The total energy consumption gap between the two algorithms was significant due to longer IMe executions, but when comparing power consumption, the gap was reduced.

This research effectively highlights the differences between IMe and ScaLAPACK in various configurations. However, it raises concerns regarding experiment repeatability, as the tests depend on the specific architecture and nodes used, which change for each test. While substantial differences and clear trends are evident, caution should be exercised when interpreting mild differences, and further research should investigate such cases.

The next phase of this work could involve the application of power caps to restrict power consumption during execution, aiming to achieve more efficient computations and investigate the behaviour of IMe and ScaLAPACK under different power configurations. Moreover, since we are aware that the accuracy of PAPI measurements is less than those we could obtain with external power meters we plan to integrate our analysis with external "ground truth" measurements [12].

ACKNOWLEDGMENTS

This work was supported by the project "Foundations of Trustworthy AI – Integrating Reasoning, Learning and Optimization – TAILOR", funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 952215. The authors want to acknowledge also the CINECA Consortium for the availability of the HPC resources and the technical support provided in the framework of the IscraC project En-FRLS. The authors wish to thank Dr. Daniela Loreti for helpful discussions on measurements of HPC software energy consumption.

REFERENCES

[1] [n. d.]. https://www.top500.org/lists/green500/2023/06/
[2] [n. d.]. Top500. https://www.top500.org/lists/top500/. https://www.top500.org/lists/top500/
[3] Anthony Agelastos, Benjamin Allan, Jim Brandt, Ann Gentile, Sophia Lefantzi, Steve Monk, Jeff Ogden, Mahesh Rajan, and Joel Stevenson. 2015. Toward Rapid Understanding of Production HPC Applications and Systems. In *2015 IEEE International Conference on Cluster Computing*. 464–473. https://doi.org/10.1109/CLUSTER.2015.71
[4] Pedro Alonso, Rosa M. Badia, Jesus Labarta, Maria Barreda, Manuel F. Dolz, Rafael Mayo, Enrique S. Quintana-Ortí, and Ruym'n Reyes. 2012. Tools for Power-Energy Modelling and Analysis of Parallel Scientific Applications. In *2012 41st International Conference on Parallel Processing*. 420–429. https://doi.org/10.1109/ICPP.2012.57
[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999. *LAPACK Users' Guide* (third ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA.
[6] Marcello Artioli and F. Filippetti. 2001. IME: a general method to analyse linear systems and electric circuits. In *Transactions on Engineering Sciences vol 31, WIT Press, 2001.* https://www.witpress.com/Secure/elibrary/papers/ES01/ES01014FU.pdf
[7] Marcello Artioli, Daniela Loreti, and Anna Ciampolini. 2019. Fault tolerant high performance solver for Linear Equation Systems. *2019 38th Symposium on Reliable Distributed Systems (SRDS)* (2019). https://doi.org/10.1109/srds47363.2019.00022
[8] Andrea Bartolini, Andrea Borghesi, Antonio Libri, Francesco Beneventi, Daniele Gregori, Simone Tinti, Cosimo Gianfreda, and Piero Altoè. 2018. The DAVIDE big-data-powered fine-grain power and performance monitoring support. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*. 303–308.
[9] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. 2000. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications* 14, 3 (2000), 189–204. https://doi.org/10.1177/109434200001400303
[10] Filippo Ciampolini. 1063. Un metodo di soluzione dei circuiti lineari,". In *L'Elettrotecnica, vol. L, no. 10, 1963.* https://doi.org/10.1007/978-3-642-30829-1_15
[11] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. *Proceeding of the 38th annual international symposium on Computer architecture - ISCA '11* (2011). https://doi.org/10.1145/2000064.2000108
[12] Muhammad Fahad, Arsalan Shahid, Ravi Reddy Manumachu, and Alexey Lastovetsky. 2019. A Comparative Study of Methods for Measurement of Energy of Computing. *Energies* 12, 11 (2019). https://doi.org/10.3390/en12112204
[13] S. Hemmert. 2010. Green HPC: From Nice to Necessity. *Computing in Science &; Engineering* 12, 06 (nov 2010), 8–10. https://doi.org/10.1109/MCSE.2010.134
[14] Joel Hruska. 2012. The death of CPU scaling: From one core to many - and why we're still stuck - page 3 of 3. https://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck/3. https://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck/3
[15] Daniela Loreti, Marcello Artioli, and Anna Ciampolini. 2020. Solving Linear Systems on High Performance Hardware with Resilience to Multiple Hard Faults. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 266–275.
[16] Heike McCraw, James Ralph, Anthony Danalis, and Jack Dongarra. 2014. Power monitoring with PAPI for extreme scale architectures and dataflow-based programming models. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 385–391.
[17] Ilya Meignan et al. 2022. *Bridging the gap between profiling and monitoring in HPC systems with dynamically reconfigurable fine-grain data collection.* Ph. D. Dissertation. Université Grenoble Alpes.
[18] Sofia Montebugnoli. 2022. IMe vs ScaLAPACK: an energy monitoring framework. https://github.com/sofiamontebugnoli/monitoring-ime-master-PAPI_MPI.git.
[19] Mohammad Rashti, Gerald Sabin, David Vansickle, and Boyana Norris. 2015. WattProf: A flexible platform for fine-grained HPC power profiling. In *2015 IEEE International Conference on Cluster Computing*. IEEE, 698–705.
[20] Elda Rossi. 2016. Ug3.1: Marconi UserGuide. https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide. https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide
[21] Li Tan, Shashank Kothapalli, Longxiang Chen, Omar Hussaini, Ryan Bissiri, and Zizhong Chen. 2014. A survey of power and energy efficient techniques for high performance numerical linear algebra operations. *Parallel Comput.* 40, 10 (2014), 559–573. https://doi.org/10.1016/j.parco.2014.09.001