To Pin or not to Pin: That is the question

Guilherme Galante[1], **Marcio Seiji Oyamada**[2]

[1]Programa de Pós-Graduação em Computação
Universidade Estadual do Oeste do Paraná - Unioeste
Cascavel – PR – Brazil

{guilherme.galante, marcio.oyamada}@unioeste.br

Abstract. *Hybrid CPU architectures, that combine high-performance and efficiency cores, pose new challenges for serial and parallel applications. With homogeneous processors, problems such as load imbalance, performance predictability, thread migration overhead and energy consumption are mitigated by CPU pinning. However, given the characteristics of hybrid processors, is CPU pinning a solution to the problems posed? Thus, our main goal is to answer the central question: to pin or not to pin? The results indicate that there is no universal answer, as it depends on the application and the execution context.*

Resumo. *Arquiteturas de CPU híbridas, que combinam núcleos de alto desempenho e eficiência energética, apresentam novos desafios para aplicações seriais e paralelas. Em processadores homogêneos, problemas como balanceamento de carga, previsibilidade de desempenho, sobrecarga gerada pela migração de threads e consumo de energia podem ser mitigados pela pinagem de CPU. No entanto, dadas as características dos processadores híbridos, a pinagem de CPU é uma solução eficiente para os problemas apresentados? Nesse contexto, o principal objetivo deste trabalho é responder à pergunta: fixar ou não fixar as threads? Os resultados indicam que não há uma resposta única, dependendo da aplicação e do contexto de execução.*

1. Introduction

Asymmetric multicore processors (Gonçalves et al. 2024), Heterogeneous Processors (Cunningham and Weaver 2024) or Hybrid processors (Yue and Mehta 2023) are processors that combine two or more different types of processing cores within a single chip to improve performance and energy efficiency. Typically, they include high-performance and energy-efficient cores.

A common example of this type of processor is the ARM big.LITTLE architecture, which is widely used in smartphones (Smejkal et al. 2024). The high-performance (big) cores feature high single-thread performance and high energy consumption, while the energy-efficient (LITTLE) cores present much lower single-thread performance at a lower power. More recently, Intel has adopted a similar approach in its Alder Lake Architecture that combines performance cores (P-cores) and efficiency cores (E-cores) (Rotem et al. 2022), addressing power and energy scaling with two core types that are very different in power and performance characteristics. In this paper, we use an Alder Lake CPU in the execution environment.

Ideally, these architectures can offer significant benefits in terms of energy efficiency and overall performance, however, they can also introduce some challenges for parallel applications, especially those designed with homogeneous cores in mind.

The first issue is *load imbalance* (Moori et al. 2023). In hybrid systems, performance cores (P-cores) are much more powerful than efficiency cores (E-cores). When a parallel application distributes work equally across all available cores, the P-cores may finish their tasks much earlier than the E-cores. This leads to idle time and inefficient resource usage, as the overall execution time can be determined by the slowest threads. Besides, this processing power difference also reduces *performance predictability*. Since the same code may run at very different speeds depending on whether it executes (P-core or an E-core), developers may face difficulties in debugging, profiling, collecting metrics, and optimizing parallel applications.

Another issue is related to *task scheduling* (Cunningham and Weaver 2024). If the operating system's scheduler is not adapted for hybrid architectures, it may assign performance-critical threads to E-cores, resulting in significant slowdowns. In contrast, less demanding background tasks might unnecessarily occupy P-cores, wasting valuable processing power. Some modern operating system schedulers, for example, Windows 11, address this issue (Smejkal et al. 2024; Saez and Prieto-Matias 2022). At the time of writing this work, it was not clear if/how the Linux operating system addresses this issue (diffuse and conflicting information). This problem can also be related to the *thread migration overhead* caused when threads are dynamically moved between cores of different types, increasing the execution time since a slower CPU may be chosen.

Finally, hybrid architectures can sometimes lead to *higher energy consumption* when using energy-efficient cores. Although these cores consume less power individually, their lower performance can result in longer execution times. As the execution time increases, the total energy consumed (which is a product of power and time) may end up being higher than if the same task were executed more quickly on a performance core (Smejkal et al. 2024). In other words, low power does not always mean low energy usage.

Analyzing this context, one alternative that immediately comes to mind to mitigate these issues is *CPU pinning* (CPU affinity or binding). This ensures that the operating system scheduler will run a thread on the designated CPU core (or set of cores) (Zhao et al. 2023). Traditionally, on multicore and NUMA systems, CPU pinning may improve performance by reducing context switching overhead, as processes remain on the same cores instead of being moved by the operating system scheduler. This also enhances cache efficiency by avoiding cache misses and maintaining warm data (Mazouz et al. 2013).

But given the characteristics of hybrid processors, is CPU pinning a solution to the questions posed? Is binding threads to P-cores sufficient to improve performance, or can restricting execution to E-cores serve as a strategy to reduce energy consumption? Thus, in this paper, we conduct an extensive set of experiments to analyze the impact of CPU pinning on performance interference and energy efficiency in both sequential and parallel OpenMP applications running on an Intel Alder Lake processor. Our goal is to address the central question: to pin or not to pin?

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 presents the experimental setup, while Section 4 describes the test cases used. The results are presented in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Recent advances in processor design have resulted in the extensive use of hybrid architectures. Although these architectures offer significant potential to optimize performance per watt, they also introduce new challenges. This section reviews the existing literature that investigates the issues associated with hybrid processors, providing context for the analysis presented in this work.

Regarding *scheduling*, Saez and Prieto-Matias (2022) evaluate how effectively the OS can drive scheduling decisions with Thread Director (TD) performance hints incorporating support in Linux to access TD facilities from the operating system kernel. They also build hardware-counter based prediction models for improving scheduling. The experiments used an Intel Core i9-12900K processor. The results show that the model provided better results than the use of TD.

In the same sense, Bilbao et al. (2023) propose PMCSched, an open source framework for the Linux kernel that enables rapid development of the OS-level support required to create custom scheduling and resource management schemes on symmetric and asymmetric multicore systems. In the tests, an Intel Core i9-12900K Alder Lake processor was used. The potential of the framework was proven by a set of experimental case studies.

Smejkal et al. (2024) present E-Mapper, a resource management approach integrated into Linux to improve execution on heterogeneous processors. The resource allocation decisions are based on high-level application descriptions that the user can attach to programs or that the system can learn automatically at runtime. The solution was tested in a Samsung Exynos 5422 and in an Intel Raptor Lake Core i9-13900K. The authors reported improvements in terms of execution time and energy consumption of 25% and 40% for the Intel Raptor Lake Core i9-13900K and 12% and 25% for the Odroid XU3-E.

Another issue explored in the literature is the *performance* in hybrid processors. Moori et al. (2023) propose HyTuning, a heuristic algorithm for optimizing the application performance requirements (execution time, energy, or EDP) by adjusting the number of threads on P-cores and E-cores of Alder Lake processors. Gonçalves et al. (2024) investigated the effects of process variability on core performance, power, and temperature on an Intel i9-12900KF processor when it is subjected to different workloads and core/uncore frequency levels.

Sundfeld et al. (2025) developed an asymmetry-aware workload distribution strategy for the PA-Star MSA bioinformatics application. They designed a two-level hash-function strategy to assign threads to cores in order to improve the execution time. The results present a reduction in the average execution time 2.97x faster for the x86_64 architecture and 2.46x faster for the ARM architecture.

As emphasized in related work, mapping applications to hybrid CPUs remains a relevant and current challenge. Our study contributes by providing additional empirical evidence and investigating how thread affinity affects both performance and energy consumption in parallel applications.

3. Experimental Setup

The experimental evaluation setup comprises an Intel Alder Lake i9-14900K, which has 8 P-cores and 16 E-cores, along with 128 GB of DDR5 RAM. The cache configuration is presented in Figure 1, generated via $lstopo$ Linux command. P-cores operate at a 3.2 GHz base frequency, reaching up to 6 GHz. In contrast, the E-cores operate at a 2.4 GHz base frequency, reaching up to 4.4 GHz. In our experiments, hyperthreading was disabled on P-cores to ensure a fair comparison, given that E-cores lack support for this feature, and also to prevent system threads from running on these cores.

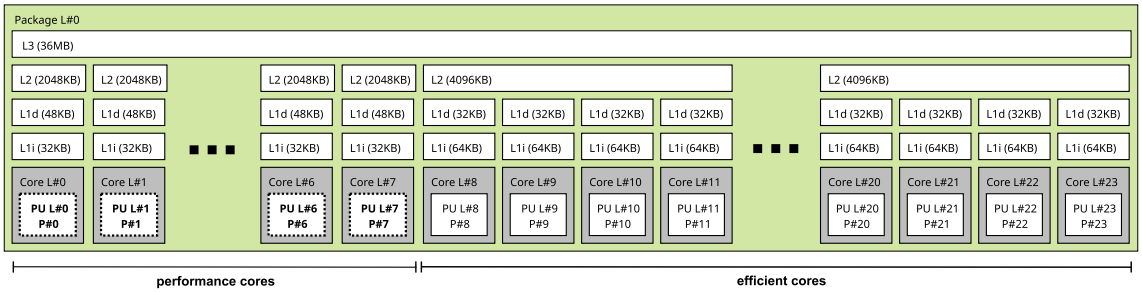

Figure 1. Intel i9-14900K configuration.

Regarding the software environment, we used Ubuntu Linux (various versions) as the operating system. We used three applications implemented in C/C++ with OpenMP as benchmarks: Lulesh 2.0 (*Livermore Unstructured Lagrange Explicit Shock Hydrodynamics*)(Karlin et al. 2013), a highly simplified application hard coded to solve the Sedov blast problem with analytical answers; Mandelbrot set area, an almost linearly scalable application from the *OpenMP Source Code Repository (OmpSCR)*, and an implementation of the LU reduction algorithm for a dense matrix, also from OmpSCR (Dorta et al. 2005). Lulesh incorporates both CPU- and memory-bound kernels, while Mandelbrot is primarily CPU-bound, and this LU decomposition is memory-bound. The parameters used in each application are presented in Table 1.

Application	**Parameters**	**Meaning**
Lulesh	-q -s 45	quiet execution, input size 45
Mandelbrot	500000	input size: 500000 points
LU	5000	input size: 5000 x 5000 matrix

Table 1. Applications parameters.

To collect the metrics, we used *perf*[1] an analysis tool available on Linux systems, designed to collect and analyze a wide range of hardware and software performance data. It provides access to performance counters in modern processors, allowing users to monitor metrics such as execution time, CPU cycles, instructions executed, cache hits and misses, context switches, energy consumption, and more.

We design our methodology to characterize performance variability while ensuring reproducible and unbiased measurements. Each experiment was run 10 times under identical conditions (no concurrent processes) to account for variability and to apply statistical analysis to ensure the robustness of our results. In the next section, we describe the different test scenarios.

[1] https://man7.org/linux/man-pages/man1/perf.1.html

4. Experimental Scenarios

In this section, we present the three experimental scenarios used in our evaluation. Each scenario is designed to assess a different aspect of the impact of hybrid processors. In Section 4.1, we evaluate the performance and energy consumption of applications executed serially. Section 4.2 focuses on the evaluation of parallel executions, analyzing their behavior under different configurations. Finally, in Section 4.3, we investigate the influence of hybrid CPUs on the performance of KVM virtual machines.

4.1. Scenario 1: Serial Applications

In Scenario 1, the execution of the serial versions of Lulesh, Mandelbrot and LU is evaluated. We compare ten execution configurations, described in Table 2. In this scenario, Ubuntu Linux 24.04 LTS kernel 6.11.0-29 is used.

Test Case #	Mapping	Governor
1	Linux scheduler	Performance
2	P-cores	Performance
3	P-cores pinning	Performance
4	E-cores	Performance
5	E-cores pinning	Performance
6	Linux scheduler	Powersave
7	P-cores	Powersave
8	P-cores pinning	Powersave
9	E-cores	Powersave
10	E-cores pinning	Powersave

Table 2. Experimental scenarios configurations.

These configurations were defined by varying both the thread-to-core mapping strategy and the CPU frequency scaling policy (Linux Power Governor). We evaluated five mapping schemes: (1) the default Linux scheduler; (2) mapping threads to a set of P-cores; (3) pinning each thread to a specific P-core; (4) mapping threads to a set of E-cores; and (5) pinning each thread to a specific E-core. For each assignment scheme, experiments were conducted with two governor policies: Performance (favoring maximum performance) and Powersave (favoring energy efficiency).

4.2. Scenario 2: Parallel Applications

In Scenario 2, we evaluate the execution of parallel versions of Lulesh and Mandelbrot. We compare the same ten configurations described in Table 1 using 2 and 8 threads.

4.3. Scenario 3: KVM Virtual Machines

In Scenario 3, the focus is on assessing the impact of hybrid processors in KVM virtual machines. To this end, experiments were conducted using a virtual instance with Ubuntu Linux 24.04 LTS kernel 6.11.0-29.

Even when a VM is instantiated with a kernel version that supports hybrid processors, KVM virtualization abstracts the underlying processor details, even when CPU passthrough is enabled. As a result, all virtual CPUs are exposed to the guest with identical characteristics, masking the distinction between P-cores and E-cores. Consequently,

the guest operating system's scheduler loses the ability to properly distribute computational loads, leaving the responsibility for thread scheduling entirely to the host operating system's scheduler. In this context, we analyze how different strategies for allocating VM cores affect the performance of the Lulesh application when running in a virtualized environment.

5. Results and Discussion

In this section, we present the experimental results obtained from evaluating the impact of thread-to-core mappings and frequency scaling strategies on the performance and energy efficiency of applications running on Intel's Alder Lake processor. The analysis is organized to highlight performance trends, quantify energy tradeoffs, and provide insights into the impact of different scheduling strategies on sequential and parallel workloads.

5.1. Scenario 1

Table 3 shows the results of the serial execution of the Lulesh application. It shows the execution wall-clock time (seconds), the average processor frequency (GHz), the number of migrations, the cache misses, and the energy consumption (joules)[2]. All data were collected directly in perf tool output, which, when executed with r repetitions (using $-r$ parameter), directly provides averages and standard deviations.

Lulesh	Exec. Time (s)		avg. Freq. (GHz)	Migrations	Cache misses	Energy (J)
Test Case	avg	std.dev				
#1	45.073	0.026	5.976	1	1376989875	2215.08
#2	45.189	0.043	5.957	1	1394953159	2220.25
#3	56.165	0.015	4.690	1	1356548507	2021.04
#4	81.899	0.074	4.390	0	2097211501	1788.89
#5	82.366	0.155	4.390	1	2119812554	1746.04
#6	65.419	0.026	3.991	1	1311504638	760.07
#7	65.492	0.027	3.991	1	1339252718	758.49
#8	65.610	0.009	3.991	1	1342128338	805.77
#9	90.584	0.211	3.978	0	2086250040	1152.18
#10	90.045	0.198	3.978	1	2085943032	1125.09

Table 3. Serial Lulesh algorithm results.

In terms of performance, the best execution times were achieved in test cases #1 and #2, where the threads were executed exclusively on a P-core running at an average frequency close to the processor's maximum limit. An interesting behavior was already observed in test case #3: When the thread was concentrated on a single P-core, the core temperature increased, triggering CPU throttling and leading to performance degradation. As expected, cases #4 and #5 present inferior results compared to tests #1 - #3, as they only use E-cores, which have lower frequencies and smaller caches, leading to more cache misses. Test cases #6 – #10 employ the powersave governor. In these scenarios, the operating frequencies are reduced and become nearly identical for both P-cores and E-cores. Nevertheless, execution times remain superior when P-cores are utilized, a result primarily explained by the lower number of cache misses observed in these cases.

[2]It is important to emphasize that *perf* measures energy consumption "system-wide" and not just the energy used to run the application.

Regarding power consumption, no significant differences were observed when the performance governor was applied. However, in test cases #6 – #10 (powersave governor), the results reveal a counterintuitive behavior: executions restricted to P-cores (#6, #7, and #8) exhibited lower power consumption compared to those running exclusively on E-cores (#9 and #10). This outcome contrasts with expectations, as E-cores are generally assumed to offer better energy efficiency.

Tables 4 and 5 show the results of the executions of the Mandelbrot and LU applications, respectively. The results shown in these tables are consistent with those obtained in the Lulesh experiments, with no significant differences observed.

Mandelbrot	**Exec. Time (s)**		**avg. Freq. (GHz)**	**Migrations**	**Cache misses**	**Energy (J)**
Test Case	**avg**	**std.dev**				
#1	46.882	0.021	5.979	1	270276	1908.70
#2	46.910	0.018	5.975	1	345848	1913.81
#3	46.841	0.014	5.984	1	284001	1690.44
#4	80.860	0.004	4.390	0	333210	1588.80
#5	80.867	0.002	4.390	1	373315	1650.79
#6	73.240	2.980	3.829	1	277895	731.34
#7	70.250	0.001	3.991	1	319899	681.30
#8	70.249	0.001	3.991	1	324191	709.70
#9	88.956	0.003	3.991	0	330368	968.74
#10	88.959	0.011	3.991	1	383916	964.18

Table 4. Serial Mandelbrot algorithm results.

LU	**Exec. Time (s)**		**avg. Freq. (GHz)**	**Migrations**	**Cache misses**	**Energy (J)**
Test Case	**avg**	**std.dev**				
#1	65.957	0.109	5.915	1	2851228366	3407.19
#2	65.771	0.065	5.915	1	2853126206	3510.62
#3	65.880	0.026	5.915	1	2869917036	3143.43
#4	112.953	0.081	4.390	0	5455336425	2764.96
#5	112.963	0.026	4.390	1	5448414184	3018.88
#6	95.834	0.071	3.991	1	2581244763	1231.49
#7	95.939	0.065	3.991	0	2587569429	1212.91
#8	95.871	0.078	3.991	1	2195002366	1210.11
#9	124.454	0.196	3.962	0	5379033992	1657.64
#10	124.475	0.031	3.962	1	5376811059	1660.35

Table 5. Serial LU algorithm results.

In summary, the results presented in this section indicate that the operating system scheduler is effective in mapping application threads, achieving performance comparable to that obtained with explicit pinning. However, from an energy perspective, pinning threads in E-cores under the powersave governor is not advisable, since their lower performance results in longer execution times and demands more energy. This is in accordance with what is stated by Smejkal et al. (2024).

5.2. Scenario 2

In this section, we present the results of Lulesh and Mandelbrot applications and investigate whether the patterns observed in their serial counterparts also apply to multithreaded

executions. Experiments with LU application were conducted, however, since the results were very similar to those obtained with Lulesh, they were omitted.

Table 6 reports the results of executing Lulesh with 2 threads, using the same experimental configurations as described in Table 2.

Lulesh (2th)	Exec. Time (s)		avg. Freq. (GHz)	Migrations	Cache misses	Energy (J)
Test Case	avg	std.dev				
#1	33,380	0,032	5,800	141	2817575000	2333,98
#2	34,125	0,052	5,708	2	2822575887	2315,91
#3	34,496	0,039	5,593	2	2791312808	2088,08
#4	57,744	0,005	4,333	2	3265669001	1499,06
#5	59,112	0,027	4,334	2	3471180328	1566,75
#6	48,139	0,091	3,777	3499	2960993483	1382,13
#7	47,836	0,013	3,827	2	2762125393	1243,20
#8	48,280	0,052	3,822	2	2953521883	1412,64
#9	64,266	0,047	3,847	2	3265025871	1450,74
#10	66,471	0,115	3,816	2	3771994269	1517,57

Table 6. Parallel Lulesh results - 2 threads.

In general, the performance trends observed in the execution using 2 threads closely match those of serial execution. When using the performance governor, test cases #1 – #3 consistently deliver similar execution times and significantly outperform the E-core restricted configurations (#4 and #5). These results reinforce the findings that P-cores provide a significant advantage in sustaining high performance compared to E-cores, even in parallel executions. Conversely, the exclusive use of E-cores introduces a significant performance penalty that can only be justified in scenarios where energy efficiency is prioritized over execution time (using the performance governor). Under the powersave governor, however, configurations that use only P-cores still deliver superior performance. In this case, they also achieve better energy efficiency, as all cores operate at lower frequencies.

Table 7 shows the results of executing Lulesh with 8 threads. Here, under the performance governor, the behavior of the application is similar to the serial version running with two threads. However, under powersave mode, some of the previously observed trends were not confirmed. Although the average frequencies of P-cores and E-cores were relatively close, execution times exhibited greater variability, with a slight advantage for configurations using only P-cores. Case #8 produced a particularly adverse result, which remained consistent even after re-running the experiment. The previously observed trend towards energy efficiency did not persist here. In this scenario, better energy performance was achieved with E-cores.

In both Tables 6 and 7 we can observe in the tests without explicit affinity (#1 and #6), the number of thread migrations increases significantly. This behavior reflects the scheduler's attempt to mitigate the increase in core temperature and prevent CPU throttling. However, such migrations come at the cost of reduced cache locality, which can increase the number of cache misses. Also in terms of cache performance, executions restricted to E-cores show a higher number of misses, which can be attributed to the reduced cache capacity available on these cores. It can also be observed that increasing the number of threads generally leads to a reduction in the average operating frequency.

This effect results from the fact that the processor cannot maintain higher frequencies when a larger number of cores are active simultaneously.

Lulesh (8th)	Exec. Time (s)		avg. Freq. (GHz)	Migrations	Cache misses	Energy (J)
Test Case	avg	std.dev				
#1	18,288	0,160	4,877	2886	2612315255	2086,35
#2	18,155	0,002	4,968	8	2544510910	2147,49
#3	18,364	0,156	4,796	8	2580966436	1988,61
#4	25,555	0,024	4,132	8	3005432398	1369,13
#5	27,310	0,065	4,123	8	3373994502	1401,25
#6	26,043	0,090	3,444	20904	2889387860	1672,79
#7	22,975	0,028	3,500	8	2575187489	1359,34
#8	45,130	2,360	3,413	8	2892967800	1870,97
#9	28,719	0,049	3,614	8	3000912461	1251,00
#10	31,217	0,090	3,551	8	3455313422	1119,82

Table 7. Parallel Lulesh results - 8 threads.

Tables 8 and 9 show the results of executing Mandelbrot with 2 and 8 threads, respectively, using the same experimental configurations described in Table 2. In general terms, the performance and energy patterns observed with the Mandelbrot application match those reported in Lulesh experiments.

Mbrot (2th)	Exec. Time (s)		avg. Freq. (GHz)	Migrations	Cache misses	Energy (J)
Test Case	avg	std.dev				
#1	25,580	1,180	5,706	3	245065	1484,82
#2	24,559	0,001	5,786	2	274717	1460,01
#3	24,741	0,001	5,688	2	225835	1358,89
#4	39,837	0,015	4,390	2	340490	1051,50
#5	39,816	0,001	4,390	2	328961	1029,45
#6	36,710	1,440	3,908	2	223782	558,95
#7	37,770	2,500	3,854	2	260967	554,94
#8	35,266	0,001	3,991	2	235869	559,39
#9	43,897	0,078	3,991	2	326825	690,35
#10	43,835	0,025	3,991	2	314724	682,40

Table 8. Parallel Mandelbrot results - 2 threads.

Mbrot (8th)	Exec. Time (s)		avg. Freq. (GHz)	Migrations	Cache misses	Energy (J)
Test Case	avg	std.dev				
#1	6,525	0,012	5,451	14	211089	891,52
#2	7,762	0,001	5,347	8	298423	1100,39
#3	6,861	0,010	5,178	8	240861	848,03
#4	10,054	0,000	4,390	8	315402	645,84
#5	10,053	0,000	4,390	8	285143	626,15
#6	8,9204	0,004	3,991	13	255370	501,53
#7	8,9106	0,001	3,991	8	265576	429,64
#8	8,9117	0,001	3,991	8	331449	665,98
#9	11,0847	0,019	3,991	8	301638	436,82
#10	11,0666	0,001	3,991	8	298500	523,84

Table 9. Parallel Mandelbrot results - 8 threads.

Although the overall patterns of Lulesh and Mandelbrot are similar, some application-specific behaviors can be observed. For example, we can observe that the number of migrations and cache misses is much lower in the Mandelbrot application. Another important observation is that the average frequency in Mandelbrot is higher than that recorded for Lulesh.

Thus, understanding the application's profile becomes essential to explore the performance of this type of processor. For instance, CPU throttling is directly influenced by the characteristics of the running application, especially in terms of CPU intensity, memory access patterns, and parallelism. CPU-bound applications, which require high processing power per core, tend to trigger throttling mechanisms faster when temperature or power limits are reached, especially on high-performance cores (P-cores). The degree of parallelism also plays a crucial role since increasing the number of simultaneous threads can overload multiple cores, which increases power consumption and temperature, and leads to frequency reduction to ensure safe operation. In addition, workloads with frequent cache misses or irregular memory accesses can trigger frequency fluctuations to balance performance and thermal efficiency.

5.3. Scenario 3

This experiment was performed on a KVM[3] virtual machine configured with 16 GB of memory and 8 vCPUs. The vCPUs were mapped using different approaches, and for each configuration, the execution time of the Lulesh application with 4 threads was measured. Table 10 shows the results. This number of threads was chosen to allow partial utilization of the available cores, allowing different mapping possibilities.

Mapping	**Lulesh execution time**	**std. dev.**
8 P-Cores	18,505	0,085
8 P-Cores (pinned)	18,395	0,101
8 E-Cores	33,963	0,108
P-Cores and E-Cores (scheduler)	18,511	0,088
4 P-Core and 4 E-Cores (pinned)	34,145	0,438

Table 10. Lulesh results in different vCPU mapping approaches.

Assigning the 8 vCPUs to 8 P-cores, whether pinned individually or not, results in the best performance, with an execution time of approximately 18 seconds. When the decision is left exclusively to the scheduler, without explicit affinity, the vCPUs are also allocated to P-cores, producing results very similar to those obtained through explicit P-core assignment.

In contrast, mapping the 8 vCPUs to E-cores results in an execution time of approximately 34 seconds, roughly 1.8x slower than the best case. A comparable slowdown is observed when pinning the vCPUs across both core types (4 on P-cores and 4 on E-cores)[4], where the inclusion of E-cores negatively impacts overall application performance.

[3] `https://linux-kvm.org/page/Main_Page`

[4] Cores 4 to 11 were allocated, where cores 4–7 correspond to P-cores and cores 8–11 to E-cores.

6. Conclusion

This study highlights the impact of hybrid CPU architectures on the execution of serial and parallel applications and shows that application characteristics and mapping strategies play a decisive role in overall performance and energy efficiency. But in the end, how do we answer the initial question? *To pin or not to pin?*. The results indicate that no single answer applies universally, as it depends on the application and execution context.

In general, the Linux scheduler proved to be effective in mapping application threads to the hybrid CPU cores, usually favoring P-cores and thereby ensuring good execution performance. Thus, in this scenario, relying on the scheduler's decisions may represent a reasonable choice. Obviously, if the goal is to grant the best application performance, you can choose to pin it to the P-cores, even at the risk of incurring CPU throttling. From an energy perspective, the results confirm that pinning applications to E-cores does not always yield the best results, as their lower performance leads to longer execution times, which may result in higher energy consumption. In virtualized environments, CPU affinity is equally important. Since the virtualization layer abstracts the underlying hardware from the guest operating system, the correct assignment of vCPU to cores can have a significant impact on the performance of virtualized applications. In such cases, explicit pinning of virtual CPUs becomes necessary.

The use of hybrid architectures can impact other aspects of parallel programming not addressed in this paper. Considering that P-cores and E-cores exhibit different performance and energy profiles, a full CPU cores usage can generate a natural load imbalance. To address such imbalances, developers can implement custom load balancing algorithms within the application or rely on scheduling mechanisms provided by parallel programming models. In OpenMP, for example, approaches such as dynamic and guided scheduling enable a more adaptive allocation of tasks, mitigating workload imbalances and improving overall processor utilization.

While the results provide relevant insights, their generalization is limited by factors such as the use of a single CPU model, one Linux distribution, and a relatively small benchmark set. Future work should expand these dimensions to strengthen the validity of the findings. Thus, future research could explore more advanced scheduling strategies, such as adaptive runtime or AI-based heuristics, to improve performance and energy efficiency on hybrid CPUs. Extending the analysis to larger, real-world workloads, including HPC applications, data science, and machine learning, would help to verify whether the trends observed in benchmarks persist in more complex scenarios. Investigating dynamic load balancing techniques to distribute work more evenly across heterogeneous cores is also a relevant research opportunity.

Acknowledge

The authors would like to thank the Laboratório de Processamento e Prototipação (LAPP-Unioeste) for providing access to the machine used in the experiments.

References

[Bilbao et al. 2023] Bilbao, C., Saez, J. C., and Prieto-Matias, M. (2023). Flexible system software scheduling for asymmetric multicore systems with pmcsched: A case for intel alder lake. *Concurrency and Computation: Practice and Experience*, 35(25).

[Cunningham and Weaver 2024] Cunningham, W. E. and Weaver, V. M. (2024). Performance measurement on heterogeneous processors with papi. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 1551–1561. IEEE.

[Dorta et al. 2005] Dorta, A., Rodriguez, C., de Sande, F., and González-Escribano, A. (2005). The openmp source code repository. In *13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 244–250.

[Gonçalves et al. 2024] Gonçalves, T. D. S., Beck, A. C. S., and Lorenzon, A. F. (2024). Investigating the influence of process variability on asymmetric multicore processors. In *2024 37th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, page 1–5. IEEE.

[Karlin et al. 2013] Karlin, I., Keasler, J., and Neely, R. (2013). Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, LLNL.

[Mazouz et al. 2013] Mazouz, A., Touati, S.-A.-A., and Barthou, D. (2013). Dynamic thread pinning for phase-based openmp programs. In Wolf, F., Mohr, B., and an Mey, D., editors, *Euro-Par 2013 Parallel Processing*, pages 53–64, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Moori et al. 2023] Moori, M. K., Rocha, H. M. G. d. A., Lorenzon, A. F., and Beck, A. C. S. (2023). Searching for the ideal number of threads on asymmetric multiprocessors. In *2023 XIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, page 1–6. IEEE.

[Rotem et al. 2022] Rotem, E., Yoaz, A., Rappoport, L., Robinson, S. J., Mandelblat, J. Y., Gihon, A., Weissmann, E., Chabukswar, R., Basin, V., Fenger, R., Gupta, M., and Yasin, A. (2022). Intel alder lake cpu architectures. *IEEE Micro*, 42(3):13–19.

[Saez and Prieto-Matias 2022] Saez, J. C. and Prieto-Matias, M. (2022). Evaluation of the intel thread director technology on an alder lake processor. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '22, page 61–67. ACM.

[Smejkal et al. 2024] Smejkal, T., Khasanov, R., Castrillon, J., and Härtig, H. (2024). E-Mapper: Energy-efficient resource allocation for traditional operating systems on heterogeneous processors.

[Sundfeld et al. 2025] Sundfeld, D., Teodoro, G., and Melo, A. C. M. A. (2025). Pa-star2: Fast optimal multiple sequence alignment for asymmetric multicore processors. In *2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 146–153.

[Yue and Mehta 2023] Yue, A. and Mehta, S. (2023). An application-oriented approach to designing hybrid cpu architectures. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, page 92–102. IEEE.

[Zhao et al. 2023] Zhao, J., Lim, K., Anderson, T., and Enright Jerger, N. (2023). The case of unsustainable cpu affinity. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, New York, NY, USA. Association for Computing Machinery.