

Thread-Count Tuning for Carbon-Efficient Multicore HPC Applications

Arthur Silveira¹, Matheus Costa¹, Carla Osthoff²,
Philippe O. A. Navaux¹, Arthur Lorenzon¹

¹ Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brazil

² National Laboratory for Scientific Computing (LNCC)
Petrópolis – RJ – Brazil

{acsilveira, matheus.costa, navaux, aflorenzon}@inf.ufrgs.br, osthoff@lncc.br

Abstract. *As multicore HPC systems expose increasingly large degrees of parallelism, using more hardware resources does not necessarily translate into more useful scientific computation. Once application scalability saturates, activating additional threads can provide little or no performance benefit while increasing energy consumption and the carbon cost of computation. Therefore, understanding this trade-off is essential for improving resource efficiency and advancing sustainable HPC. In this paper, we quantify how thread-count tuning affects performance and operational carbon emissions across 20 OpenMP applications and five multicore CPU platforms spanning 16–128 hardware threads. A full design-space exploration identifies performance-, carbon-, and joint-optimal configurations. Relative to maximum-thread execution, performance-optimal tuning improves geometric-mean runtime by up to 16.0%, while carbon-optimal tuning reduces emissions by up to 27.8%. Scalability-limited workloads benefit substantially more: *sp.C* reduces emissions by 69.5% on AMD128.*

1. Introduction

High-performance computing (HPC) has traditionally treated shorter execution time as the primary optimization objective. However, this view is increasingly incomplete as the energy required to execute scientific workloads directly affects operating cost and, depending on the electricity source, the operational carbon footprint of the computation. Consequently, a faster execution policy is not necessarily more sustainable, and a policy that activates more hardware resources is not necessarily more efficient.

Thread-level parallelism (TLP) is a representative example of this tension. The default OpenMP execution policy exposes all available hardware threads to an application. This policy is effective when additional threads continue to reduce execution time, but many parallel workloads stop scaling before reaching the machine limit. Synchronization, contention for shared caches and memory, off-chip bandwidth saturation, and simultaneous multithreading (SMT) contention can cause additional threads to provide little performance benefit or even increase execution time [Suleman et al. 2008, Subramanian et al. 2013, Raasch and Reinhardt 2003]. Those threads nevertheless keep cores and shared resources active, increasing the energy spent to complete the workload.

Thread count is a particularly attractive sustainability control knob, since it is software-defined, portable across multicore processors, and can be

changed without modifying the application algorithm or deploying specialized power-management infrastructure. Previous work has used concurrency control to optimize performance, power, energy, or Energy-Delay Product (EDP) [Suleman et al. 2008, Chadha et al. 2012, Marathe et al. 2015, Sensi et al. 2016, Sridharan et al. 2014, Lorenzon et al. 2017, Lorenzon et al. 2018]. However, the operational carbon consequence of these scalability effects has received substantially less attention, especially across processors with different core counts and when the electricity carbon intensity is considered explicitly.

Given the aforementioned scenario, this paper continues the discussion done by [Silveira and Lorenzon 2024] and asks a direct question: *How much carbon and performance are left on the table when an OpenMP application simply uses all hardware threads?* We answer it through a full thread-count design-space exploration of 20 applications on five multicore CPU platforms ranging from 16 to 128 hardware threads. For every application–platform pair, we identify the configuration that minimizes execution time, the one that minimizes operational carbon emissions, and a scale-independent configuration that balances both objectives.

This extensive evaluation leads to three main findings. First, maximum-thread execution is frequently suboptimal, and the optimization opportunity increases on the larger platforms. At the geometric-mean level, performance-oriented tuning improves runtime by up to 16.0%, whereas carbon-oriented tuning reduces emissions by up to 27.8%. Second, the largest gains occur in scalability-limited applications: on AMD128, *sp.C* reduces execution time by 67.3% and operational carbon emissions by 69.5% relative to the default execution. Third, hardware efficiency alone does not determine sustainability. Under the electricity-mix scenarios evaluated in this work, the same execution can differ by up to $37\times$ in operational carbon emissions depending on location.

The main contribution of this work is thus an experimental characterization of thread count as a carbon-efficiency mechanism for shared-memory HPC. Rather than proposing a new runtime, we establish the optimization potential, identify the workload behaviors that create it, and quantify how the resulting execution choices interact with architecture and electricity carbon intensity. Overall, we provide the empirical basis for future carbon-aware runtimes and schedulers that treat concurrency as an explicit optimization dimension.

The remainder of this paper is organized as follows. Section 2 presents the background and related work on energy-efficient and carbon-aware HPC. Section 3 describes the experimental platforms, workloads, measurement methodology, and optimization criteria. Section 4 presents the evaluation of thread-count effects on performance, energy consumption, and operational carbon emissions, including the multiobjective analysis and the impact of regional carbon intensity. Finally, Section 5 summarizes the main findings and discusses their implications for sustainable HPC.

2. Background and Related Work

2.1. Why More Threads Can Be Less Efficient

Increasing the number of active threads improves performance only while the application can effectively use the additional parallel resources. Four mechanisms are particularly

relevant to the workloads evaluated in this paper. First, synchronization introduces serialization and waiting: once the synchronization cost grows faster than the useful work assigned to each thread, additional parallelism increases overhead [Suleman et al. 2008]. Second, threads that communicate through shared cache or memory compete for common resources, increasing access latency. Third, memory-intensive applications can saturate off-chip bandwidth; once saturated, additional threads increase activity without increasing the data delivery rate. Finally, when SMT is used, threads sharing a physical core compete for issue bandwidth, execution units, and other core resources, which can increase imbalance and expose new critical threads [Raasch and Reinhardt 2003].

These effects explain why the thread count that minimizes execution time can be lower than the hardware maximum, and why the energy-minimizing thread count can be even lower. From a carbon perspective, operational emissions are proportional to the energy consumed during execution and the carbon intensity of the supplied electricity. We model operational carbon as

$$C_{op} = E_{op} \times I_{grid}, \quad (1)$$

where E_{op} is execution energy in kWh and I_{grid} is electricity carbon intensity in gCO₂/kWh. The analysis intentionally focuses on operational emissions. Embodied emissions from hardware manufacturing, transportation, and end-of-life are outside the scope because they are not affected by the per-run thread-count decision studied here [Li et al. 2023, Kline et al. 2019, Gupta et al. 2022].

2.2. Concurrency, Energy, and Carbon-Aware Execution

Concurrency control has a long history as a performance- and energy-management technique. Feedback-Driven Threading adapts thread count according to synchronization and bandwidth behavior [Suleman et al. 2008]; LIMO, Conductor, NORNIR, and Varuna similarly control concurrency to improve execution efficiency [Chadha et al. 2012, Marathe et al. 2015, Sensi et al. 2016, Sridharan et al. 2014]. PUPiL combines software and hardware control under power constraints [Zhang and Hoffmann 2016], while other work considers power-budget-aware scheduling [Etinski et al. 2010]. LAANT and Aurora are particularly close to the mechanism studied here: both tune OpenMP concurrency at runtime, but their objectives are performance/EDP rather than carbon emissions [Lorenzon et al. 2017, Lorenzon et al. 2018].

More recent work broadens the optimization space through DVFS, power caps, runtime parameters, and resource management policies [Navarro Muñoz et al. 2021, Krzywaniak et al. 2022, Azhar et al. 2023, Cabrera et al. 2023, Chheda et al. 2024]. These studies underscore that execution configuration substantially affects energy efficiency, but carbon is generally not the primary optimization target. In parallel, sustainable computing research has developed methods to estimate the computational carbon footprint [Lannelongue et al. 2021] and carbon-aware scheduling policies that exploit spatial or temporal variation in electricity supply [Wang et al. 2018, Maji et al. 2023, Zhou et al. 2013].

Our work connects these two lines of research. We do not treat carbon solely as a transformed energy metric because the final environmental outcome also depends on the

Table 1. Target multicore platforms. C/T denotes physical cores/HW threads.

Platform	Processor	Microarch.	C/T	LLC	Memory	TDP
Intel16	Xeon E5-2620 v4	Broadwell	8/16	20 MB	256 GB	85 W
Intel40	Xeon E5-2650	Haswell	20/40	50 MB	128 GB	210 W
Intel48	Xeon Gold 5317	Ice Lake	24/48	36 MB	128 GB	300 W
Intel88	Xeon E5-2699	Broadwell	44/88	110 MB	256 GB	290 W
AMD128	EPYC 7763	Zen 3	64/128	256 MB	512 GB	280 W

provenance of electricity. Conversely, we do not treat location as the only carbon-aware decision, because application scalability determines how much energy is required before the grid mix is applied. Therefore, the evaluation isolates three interacting dimensions: application behavior, multicore architecture, and electricity carbon intensity.

3. Experimental Methodology

3.1. Applications

We evaluate 20 C/C++ OpenMP applications from established benchmark suites and scientific kernels. Eight applications come from Rodinia [Che et al. 2009]: BackPropagation (BP), Computational Fluid Dynamics (CFD), Hotspot (HS), Kmeans (KM), Leukocyte Tracking (LE), LU Decomposition (LUD), Streamcluster (SC), and Speckle Reducing Anisotropic Diffusion (SRAD). Six are from the NAS Parallel Benchmarks [Bailey et al. 1995]: bt.C, cg.C, ft.C, mg.C, sp.C, and ua.C. The remaining workloads are LULESH [Karlin et al. 2013], FFT [Brigham and Morrow 1967], HPCG [Heroux et al. 2013], Jacobi (JA), Poisson (PO), and STREAM (ST).

The set spans distinct scalability behaviors. Prior characterization associates JA, sp.C, and BP with synchronization overhead; SC, PO, mg.C, bt.C, and LUD with shared-memory contention; FFT, ST, HPCG, LULESH, and LE with off-chip bandwidth pressure; and HS, cg.C, ft.C, SRAD, CFD, and KM with SMT/resource contention effects [Suleman et al. 2008, Subramanian et al. 2013, Raasch and Reinhardt 2003]. This diversity is important because the benefit of thread-count tuning depends on the bottleneck that prevents useful scaling.

3.2. Execution Environment

Table 1 summarizes the five multicore platforms. They cover 8–64 physical cores and 16–128 hardware threads, including Intel Broadwell, Haswell, and Ice Lake processors and an AMD Zen 3 processor. All applications were compiled with GCC/G++ 12 using `-O3` and OpenMP 5.0. CPU frequency was managed by the Linux *ondemand* governor throughout the campaign, so frequency selection remained dynamic while thread count was the controlled variable.

For each application–platform pair, we sweep the OpenMP thread count from low concurrency to the maximum number of hardware threads using the even-valued thread-count sequence employed by the experimental campaign. The maximum-thread configuration is denoted *Default*. Each configuration is executed ten times. We report mean execution time and mean operational carbon; the standard deviation was below 0.5% in all measured configurations.

Energy is reported by the measurement mechanism available on the corresponding experimental platform. The same measurement backend is kept fixed across all thread configurations on a given platform, making the within-platform comparisons used by the design-space exploration directly comparable. Operational carbon is then computed using Eq. 1. The baseline experiments use 432 gCO₂/kWh for Intel16/40/48/88 and 410 gCO₂/kWh for AMD128, following the carbon-intensity values from the original measurement campaign and obtained via CarbonTracker [Anthony et al. 2020]. Section 4.3 replaces these values with region-specific intensities while holding execution energy constant.

3.3. Optimization Objectives

For each application and platform, the design space exploration (DSE) produces three configurations. *Opt-Perf* is the thread count with minimum execution time; *Opt-Carbon* is the thread count with minimum operational carbon; and *Opt-Balance* captures a compromise between the two objectives. To avoid a unit-dependent distance between seconds and grams of CO₂, time and carbon are independently min–max normalized within each application–platform DSE:

$$\hat{T}_i = \frac{T_i - T_{min}}{T_{max} - T_{min}}, \quad \hat{C}_i = \frac{C_i - C_{min}}{C_{max} - C_{min}}. \quad (2)$$

The balanced score is the Euclidean distance to the ideal point (0, 0),

$$B_i = \sqrt{\hat{T}_i^2 + \hat{C}_i^2}, \quad (3)$$

and *Opt-Balance* is the configuration with minimum B_i . This formulation gives equal scale to both metrics while preserving the application-specific shape of the performance–carbon trade-off.

4. Evaluation

4.1. Impact of Thread Count on Performance and Carbon

The first result is that the default maximum-thread policy is not a reliable optimum for either metric. Table 2 reports the thread counts selected independently for performance and carbon. The optimum is strongly application- and architecture-dependent, and the two objectives often diverge. On Intel48, for example, the FFT achieves its best execution time with 38 threads but its lowest carbon footprint with only 2 threads. On Intel88, PO, BP, FFT, and HPCG achieve their best performance with 20, 20, 14, and 16 threads, respectively, despite 88 hardware threads being available. In contrast, cg.C and KM frequently remain at or near maximum concurrency, showing that tuning is not universally equivalent to reducing parallelism.

Across the 100 application–platform pairs, maximum-thread execution fails to minimize runtime in 52 cases and fails to minimize operational carbon in 71. The carbon-optimal thread count is lower than the performance-optimal count in 54 pairs, identical in 45, and higher in only one. The trend becomes stronger as concurrency grows: on AMD128, 16 of 20 workloads minimize carbon below the 128-thread default, and 15

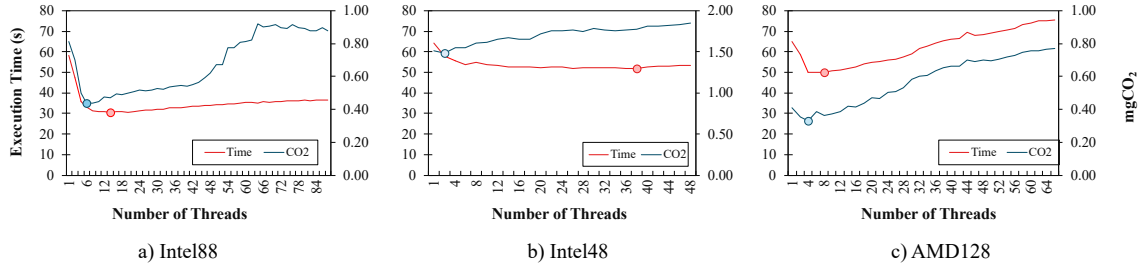


Figure 1. FFT execution time and operational carbon across thread counts. Markers identify the best configuration for each metric.

of 20 select fewer threads for carbon than for performance. Thus, carbon optimization is not merely recovering the same scalability optimum; it shifts many workloads toward a lower-concurrency operating point. This divergence is a direct consequence of scalability. Figure 1 shows FFT on three architectures. Once memory-system pressure limits useful scaling, increasing concurrency no longer reduces runtime proportionally, while energy and carbon continue to accumulate. On Intel48, reducing concurrency from the 48-thread default to the carbon-optimal two-thread configuration reduces carbon by 20.0%; the performance-optimal 38-thread configuration is only 3.2% faster than the default. The effect is even clearer on AMD128, where the FFT achieves its best runtime with 12 threads and its best carbon result with 4 threads.

Table 3 summarizes the geometric-mean improvement relative to the maximum-thread default. Performance-optimal tuning improves runtime by 3.2% on Intel16 but by 16.0% on AMD128. Carbon-optimal tuning reduces emissions by 6.5% on Intel16, 22.4% on Intel88, and 27.8% on AMD128. The larger machines offer a broader concurrency range, allowing applications to saturate shared resources before reaching the hardware limit, increasing the opportunity to avoid unproductive activity. Application-level gains

Table 2. Thread counts selected by the DSE for minimum execution time (P) and minimum operational carbon (C). The default is the maximum hardware thread count for each platform.

App.	Intel16		Intel40		Intel48		Intel88		AMD128	
	P	C	P	C	P	C	P	C	P	C
bt.C	16	8	40	20	46	46	88	80	128	56
cg.C	16	16	40	40	24	24	88	88	128	128
ft.C	16	16	40	32	24	24	42	42	56	52
mg.C	8	4	20	20	24	24	42	42	12	12
sp.C	6	6	20	4	24	4	42	10	28	12
ua.C	16	16	40	40	48	48	88	88	56	128
PO	14	10	40	40	48	48	20	12	128	128
ST	12	10	40	40	48	48	88	88	128	56
FFT	8	6	12	12	38	2	14	6	12	4
HPCG	8	8	20	18	10	8	16	16	16	4
JA	8	6	20	10	24	8	88	14	48	12
LULESH	8	8	20	14	10	8	20	16	20	12
HS	14	14	40	34	48	30	88	74	56	44
LUD	16	16	40	20	36	34	88	78	128	28
SC	16	12	36	18	42	38	46	18	96	32
BP	16	16	30	8	44	44	20	20	124	116
CFD	16	16	40	38	48	48	88	72	128	112
KM	16	16	40	40	48	48	86	86	128	128
LE	16	16	40	36	48	42	60	50	56	40
SRAD	16	16	40	40	48	48	88	88	128	112

can be substantially larger than the platform geometric mean. On AMD128, sp.C reduces runtime from 75.08 s to 24.55 s (67.3%) and carbon from 1.18 to 0.36 gCO₂ (69.5%). LULESH reduces carbon by 65.4% on Intel88 and 62.7% on Intel48. FFT reduces carbon by 53.3% on AMD128 and 50.0% on Intel88. These workloads are precisely the cases in which additional threads amplify synchronization or memory-system pressure rather than useful throughput. Conversely, workloads such as cg.C and KM often select the default concurrency, which indicates that the DSE does not yield when scaling remains effective.

4.2. Balancing Performance and Operational Carbon

Carbon-optimal configurations sometimes trade substantial runtime for a relatively small additional emission reduction in emissions, while performance-optimal configurations can incur energy overhead. Figure 2 exposes this trade-off at the platform level by aggregating all applications with the geometric mean. Each orange point represents a tested concurrency; the highlighted points indicate Default, Opt-Perf, Opt-Carbon, and Opt-Balance.

Two patterns emerge. First, the balanced configuration is usually located much closer to the performance and carbon optima than the default point. Second, the relative cost of pursuing the absolute carbon minimum is architecture-dependent. Intel40 is illustrative, since the carbon-optimal configuration lowers emissions but incurs a runtime penalty relative to the default, whereas the balanced configuration retains most of the carbon benefit with a much smaller performance cost. On Intel88 and AMD128, the trade-off is more favorable because the default is inefficient enough that a single tuned configuration can improve both metrics simultaneously.

Figure 3 normalizes the three optimized policies to Default and makes the cross-platform behavior explicit. Opt-Perf is always the fastest policy, while Opt-Carbon consistently produces the lowest emissions. Opt-Balance remains below the default carbon level across all platforms and below the default runtime on four of the five platforms. This is an important practical result, in which thread-count tuning does not require that sustainability be treated as a pure performance sacrifice. In many cases, avoiding scalability-induced waste simultaneously improves time-to-solution and operational emissions.

The application curves in Figure 4 show why the balanced point changes with workload behavior. JA and SC on Intel88 reach minimum carbon well before their minimum runtime; in both cases, the normalized balance therefore favors a lower thread count. For sp.C and HS on AMD128, runtime falls rapidly before flattening, and the balance shifts toward the performance optimum. The relevant decision is not simply *use fewer*

Table 3. Geometric-mean improvement over the maximum-thread default. Runtime uses *Opt-Perf*; carbon uses *Opt-Carbon*.

Platform	Runtime reduction	Carbon reduction
Intel16	3.2%	6.5%
Intel40	3.1%	7.5%
Intel48	7.5%	10.6%
Intel88	7.3%	22.4%
AMD128	16.0%	27.8%

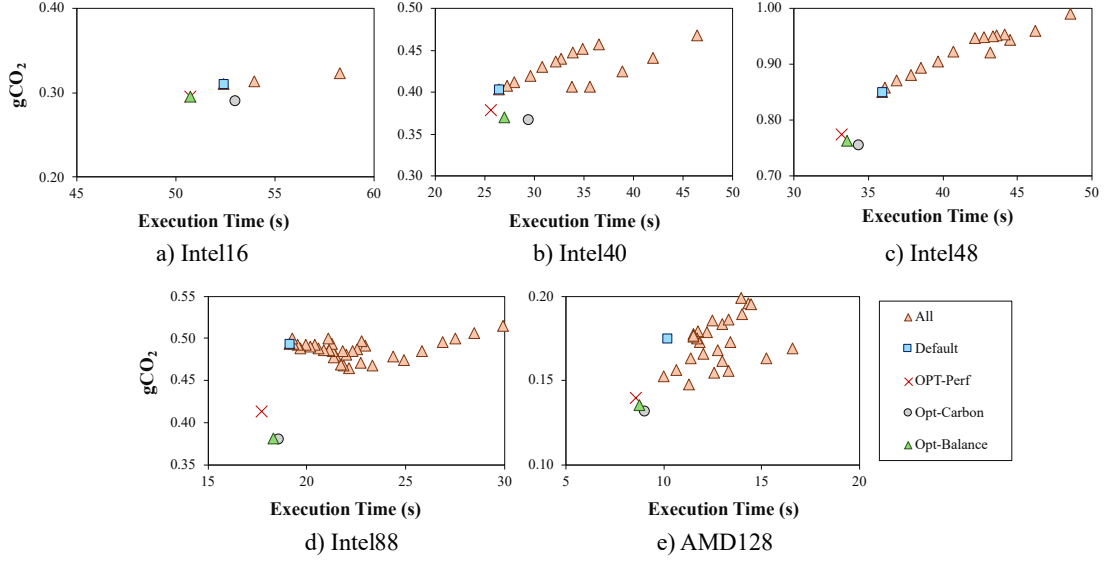


Figure 2. Platform-level performance-carbon trade-off across the explored thread counts. Lower-left is better.

threads, but rather stop increasing concurrency when the marginal performance return no longer compensates for the additional energy and carbon.

4.3. Impact of Electricity Mix on Carbon Footprint

While thread tuning changes how much energy a workload requires, electricity provenance determines how much carbon is associated with that energy. To isolate this second effect, we recompute the operational emissions of the Opt-Balance executions using nine electricity carbon-intensity scenarios representative of regions hosting major HPC facilities. Table 4 lists the values used in the analysis. The scenarios combine electricityMap and Ember data [Tomorrow 2024, Climate 2024] with statistics for North America [IEA 2023, EIA 2023, IESO 2023], Brazil and France [ONS 2023, RTE 2023], and Italy and the UK [Terna 2023, National Grid ESO 2023].

Figure 5 shows the resulting emissions for each architecture. The ranking of hardware configurations does not change because execution time and energy are held fixed, but the absolute environmental cost changes substantially. For Intel88, the same balanced workload mix produces 0.108 gCO₂ under the Switzerland scenario and 3.954 gCO₂ under the Japan scenario, a difference of approximately 37×. The result separates two complementary optimization levels, in which local concurrency control reduces a job’s energy demand, while carbon-aware placement reduces the carbon intensity associated with that demand.

From a systems perspective, this suggests a two-stage policy for infrastructures with placement flexibility. The runtime or node-level policy should first avoid wasteful concurrency for the selected architecture; the scheduler can then consider temporal or spatial carbon intensity when deciding where or when to execute the job. Prior work has demonstrated the potential of carbon-aware scheduling [Maji et al. 2023, Zhou et al. 2013]; our results show that such policies should not assume that a job’s energy demand is fixed independently of its thread configuration.

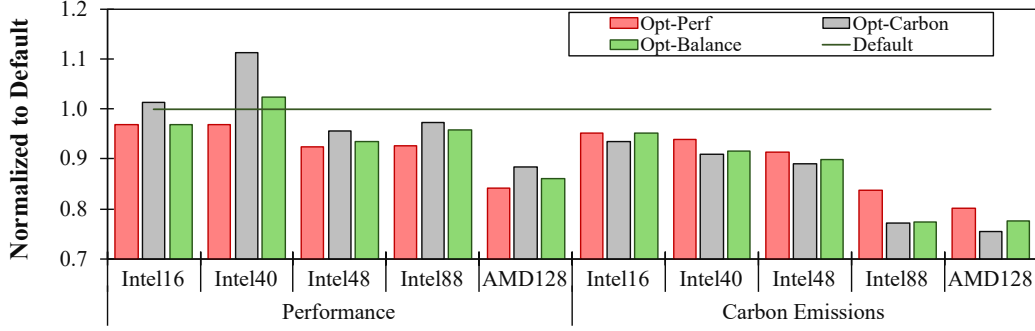


Figure 3. Performance and carbon of Opt-Perf, Opt-Carbon, and Opt-Balance normalized to maximum-thread execution. Values below 1.0 improve over the default.

4.4. Implications and Limitations

Three implications follow from the experiments. First, thread count should be treated as an execution parameter rather than a constant derived from machine size. The default is particularly risky on high-core-count processors, where the gap between useful application parallelism and available hardware parallelism can be large. Second, the bottleneck class provides a useful qualitative signal, where synchronization- and memory-limited workloads expose the largest optimization headroom, whereas well-scaling compute workloads often remain close to maximum concurrency. Third, a thread configuration determines energy demand, but final operational emissions also depend on the electricity mix.

Moreover, we identify four main limitations in our study. The carbon analysis covers only operational emissions and does not account for embodied hardware emissions. However, it is very hard to account for all embodied hardware emissions, which is a natural extension of this paper. The regional experiment is a scenario analysis based on fixed carbon-intensity values rather than a time-varying trace. The evaluated systems are CPU-based shared-memory platforms, so the conclusions should not be extrapolated directly to accelerator-dominated workloads. Finally, the DSE is offline and exhaustive; a production system would need to predict or learn the useful thread count with substantially fewer exploratory executions. These limitations do not affect the main comparison between maximum-thread execution and tuned execution, but they define the scope for runtime and scheduler-level extensions.

5. Concluding Remarks and Future Work

As HPC systems continue to increase in core count, power density, and architectural complexity, improving sustainability requires more than deploying energy-efficient processors

Table 4. Electricity carbon-intensity scenarios used for the location analysis.

Region	gCO ₂ /kWh	Region	gCO ₂ /kWh
Switzerland	13	UK	200
Brazil	46	Italy	257
France	56	Germany	371
Canada	110	Japan	476
USA (TN)	124		

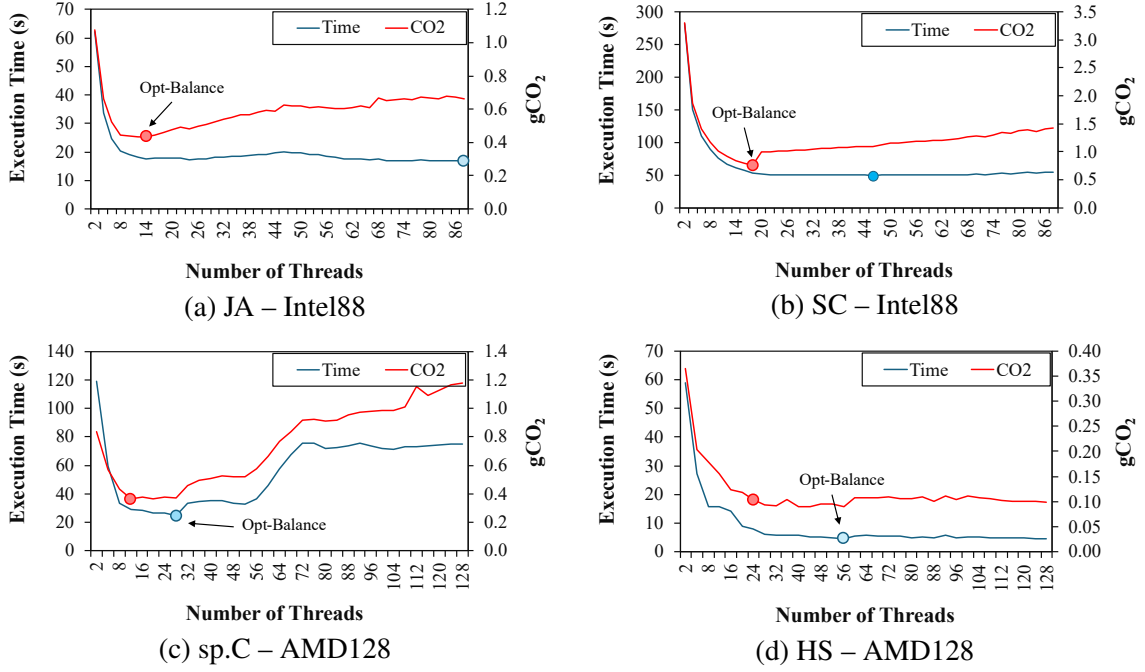


Figure 4. Representative application-level trade-offs. The annotated Opt-Balance point varies with the shapes of the runtime and carbon curves.

or operating systems in regions with cleaner electricity. It also requires ensuring that allocated computing resources are effectively converted into useful scientific progress. In this context, maximizing resource occupancy is not necessarily equivalent to maximizing efficiency. A system may activate more hardware threads while delivering little or no additional performance, thereby increasing energy consumption and operational carbon without increasing useful computation. Sustainable HPC must consider not only peak performance, but also how much scientific work is delivered per unit of energy, i.e., its effective *science per watt*.

Our results provide quantitative evidence that thread count is an important control dimension in this problem. Across 20 OpenMP workloads and five multicore processors, using all available hardware threads is frequently neither the fastest nor the most carbon-efficient configuration. Performance-oriented tuning improves geometric-mean runtime by up to 16.0%, while carbon-oriented tuning reduces geometric-mean operational emissions by up to 27.8%. For scalability-limited workloads, the effect is substantially larger. sp.C reduces runtime by 67.3% and operational carbon by 69.5% on AMD128. The multiobjective analysis further shows that performance and sustainability are often aligned, demonstrating that reducing resource overprovisioning can simultaneously shorten execution time and decrease environmental cost.

This observation is particularly relevant as modern HPC processors expose an increasing number of cores and hardware threads. Under these conditions, inefficient parallel scaling becomes not only a performance problem but also an energy-efficiency problem. Allocating additional execution resources beyond an application’s effective scalability can reduce the useful work obtained per watt consumed. Conversely, selecting an appropriate concurrency level can improve resource utilization and increase scientific

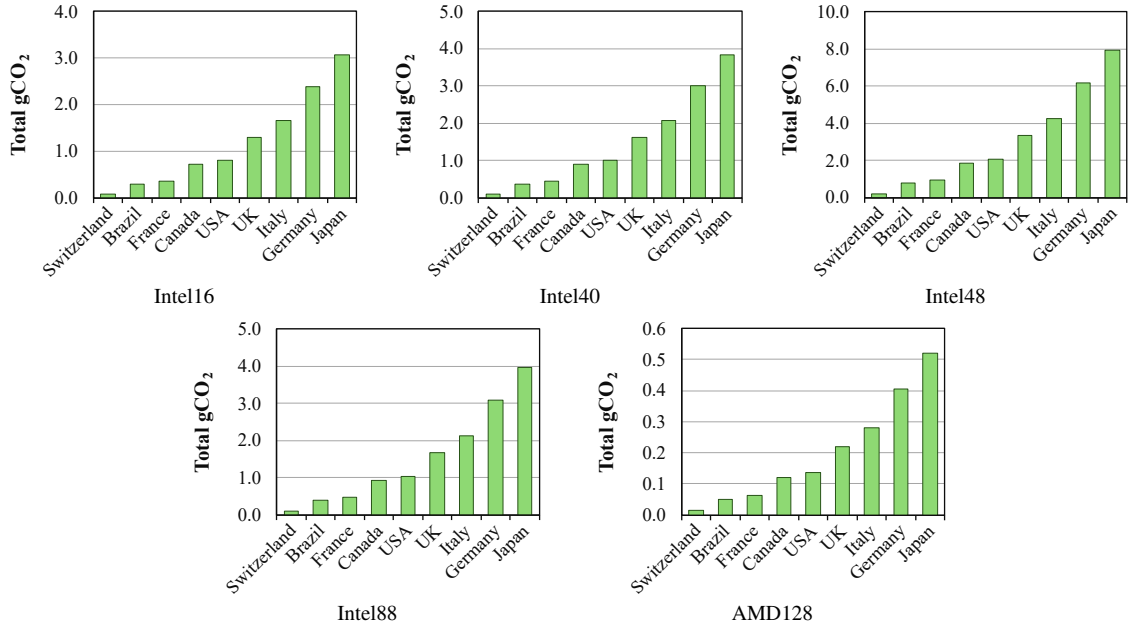


Figure 5. Operational carbon of Opt-Balance under different electricity carbon-intensity scenarios.

throughput per unit of energy without requiring changes to the application algorithm or underlying hardware. The regional analysis also demonstrates that execution efficiency represents only one layer of the sustainability problem. Once workload behavior and architecture determine energy consumption, the electricity mix can change the associated operational carbon emissions by up to $37\times$ across the evaluated scenarios. Improving the sustainability of HPC requires coordinated decisions across multiple levels: application-level concurrency, architecture and resource selection, system scheduling, and the carbon intensity of the electricity supplying the computation.

Future work will extend this perspective beyond thread-count selection by jointly exploring concurrency and dynamic voltage and frequency scaling (DVFS), enabling runtime systems to adapt both the amount of active parallelism and the processor’s operating point to workload behavior. We also plan to incorporate heterogeneous resource selection and time- and location-dependent carbon-intensity signals into this optimization process. More broadly, future analyses should move beyond operational emissions and account for the embodied carbon associated with manufacturing, provisioning, and amortizing computing infrastructure. This is particularly relevant when comparing resource configurations with different processor generations, node counts, utilization levels, and hardware lifetimes, since minimizing the carbon emitted during execution does not necessarily minimize the total life-cycle carbon cost of computation. Combining operational and embodied carbon with performance and energy metrics can enable a more complete notion of *science per watt* and, ultimately, *science per unit of life-cycle carbon*, providing a foundation for runtime and scheduling policies that maximize scientific output under explicit energy, power, and environmental constraints.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This study was financed in part by CAPES – Finance Code 001, FAPERGS – PqG 24/2551-0001388-1, and CNPq.

References

- Anthony, L. F. W., Kanding, B., and Selvan, R. (2020). Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems. arXiv:2007.03051.
- Azhar, M. W., Manivannan, M., and Stenström, P. (2023). Approx-rm: Reducing energy on heterogeneous multicore processors under accuracy and timing constraints. *ACM Trans. Archit. Code Optim.*, 20(3).
- Bailey, D., Harris, T., Saphir, W., Van Der Wijngaart, R., Woo, A., and Yarrow, M. (1995). The nas parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center.
- Brigham, E. O. and Morrow, R. E. (1967). The fast fourier transform. *IEEE Spectrum*, 4(12):63–70.
- Cabrera, A., Almeida, F., Castellanos-Nieves, D., Oleksiak, A., and Blanco, V. (2023). Energy efficient power cap configurations through pareto front analysis and machine learning categorization. *Cluster Computing*, 27(3):3433–3449.
- Chadha, G., Mahlke, S., and Narayanasamy, S. (2012). When less is more (LIMO): Controlled parallelism for improved efficiency. In *CASES*, pages 141–150, USA.
- Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54.
- Chheda, S., Verma, G., Tian, S., Chapman, B., and Doerfert, J. (2024). Evaluating tuning opportunities of the llvm/openmp runtime. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 919–929.
- Climate, E. (2024). Global electricity review 2024. <https://ember-climate.org/insights/research/global-electricity-review-2024/>. Accessed: 2024-04-01.
- EIA (2023). State electricity profiles - tennessee. <https://www.eia.gov/electricity/state/tennessee/>. U.S. Energy Information Administration. Accessed: 2024-04-01.
- Etinski, M., Corbalan, J., Labarta, J., and Valero, M. (2010). Optimizing job performance under a given power constraint in hpc centers. In *International Conference on Green Computing*, pages 257–267.
- Gupta, U., Elgamal, M., Hills, G., Wei, G.-Y., Lee, H.-H. S., Brooks, D., and Wu, C.-J. (2022). Act: designing sustainable computer systems with an architectural carbon

- modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 784–799, New York, NY, USA. Association for Computing Machinery.
- Heroux, M. A., Dongarra, J., and Luszczek, P. (2013). Hpcg benchmark technical specification.
- IEA (2023). Electricity information 2023: Overview. <https://www.iea.org/reports/electricity-information-overview>. International Energy Agency. Accessed: 2024-04-01.
- IESO (2023). Ontario's electricity data. <https://www.ieso.ca/en/Power-Data>. Independent Electricity System Operator. Accessed: 2024-04-01.
- Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. (2013). Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)*, Boston, USA.
- Kline, D., Parshook, N., Ge, X., Brunvand, E., Melhem, R., Chrysanthis, P. K., and Jones, A. K. (2019). Greenchip: A tool for evaluating holistic sustainability of modern computing systems. *Sustainable Computing: Informatics and Systems*, 22:322–332.
- Krzywaniak, A., Czarnul, P., and Proficz, J. (2022). Depo: A dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing. *Software: Practice and Experience*, 52(12):2598–2634.
- Lannelongue, L., Grealey, J., and Inouye, M. (2021). Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science*, 8.
- Li, B., Basu Roy, R., Wang, D., Samsi, S., Gadepally, V., and Tiwari, D. (2023). Toward sustainable hpc: Carbon footprint estimation and environmental implications of hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA. Association for Computing Machinery.
- Lorenzon, A. F., De Oliveira, C. C., Souza, J. D., and Beck, A. C. S. (2018). Aurora: Seamless optimization of openmp applications. *IEEE transactions on parallel and distributed systems*, 30(5):1007–1021.
- Lorenzon, A. F., Souza, J. D., and Beck, A. C. S. (2017). Laant: A library to automatically optimize edp for openmp applications. In *DATE*, pages 1229–1232. IEEE.
- Maji, D., Pfaff, B., PR, V., Sreenivasan, R., Firoiu, V., Iyer, S., Josephson, C., Pan, Z., and Sitaraman, R. K. (2023). Bringing carbon awareness to multi-cloud application delivery. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, pages 1–6.
- Marathe, A., Bailey, P. E., Lowenthal, D. K., Rountree, B., Schulz, M., and de Supinski, B. R. (2015). A run-time system for power-constrained hpc applications. In Kunkel, J. M. and Ludwig, T., editors, *High Performance Computing*, pages 394–408, Cham. Springer.

- National Grid ESO (2023). Carbon intensity api and national grid dashboard. <https://carbon-intensity.github.io>. National Grid Electricity System Operator. Accessed: 2024-04-01.
- Navarro Muñoz, A., F. Lorenzon, A., Ayguadé Parra, E., and Beltran Querol, V. (2021). Combining dynamic concurrency throttling with voltage and frequency scaling on task-based programming models. In *Proceedings of the 50th International Conference on Parallel Processing, ICPP '21*, New York, NY, USA. Association for Computing Machinery.
- ONS (2023). Boletim mensal de geração - brasil. <https://www.ons.org.br/aceso-a-informacao/dados-abertos/geracao>. Operador Nacional do Sistema Elétrico. Accessed: 2024-04-01.
- Raasch, S. E. and Reinhardt, S. K. (2003). The impact of resource partitioning on smt processors. In *PACT*, pages 15–25.
- RTE (2023). Bilan Électrique 2023. <https://www.rte-france.com>. Réseau de Transport d'Électricité. Accessed: 2024-04-01.
- Sensi, D. D., Torquati, M., and Danelutto, M. (2016). A reconfiguration algorithm for power-aware parallel applications. *TACO*, 13(4):43:1–43:25.
- Silveira, A. and Lorenzon, A. (2024). Análise da pegada de carbono em aplicações paralelas. In *Anais da XXIV Escola Regional de Alto Desempenho da Região Sul*, pages 111–112, Porto Alegre, RS, Brasil. SBC.
- Sridharan, S., Gupta, G., and Sohi, G. S. (2014). Adaptive, efficient, parallel execution of parallel programs. In *ACM SIGPLAN PLDI*, pages 169–180, NY, USA. ACM.
- Subramanian, L., Seshadri, V., Kim, Y., Jaiyen, B., and Mutlu, O. (2013). Mise: Providing performance predictability and improving fairness in shared main memory systems. In *IEEE HPCA*, pages 639–650.
- Suleman, M. A., Qureshi, M. K., and Patt, Y. N. (2008). Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on cmps. *SIGPLAN Not.*, 43(3):277–286.
- Terna (2023). Dati statistici sull'energia elettrica in italia. <https://www.terna.it>. Terna - Rete Elettrica Nazionale. Accessed: 2024-04-01.
- Tomorrow (2024). electricitymap - live CO₂ emissions of electricity consumption. <https://www.electricitymap.org>. Accessed: 2024-04-01.
- Wang, C., Zink, M., and Irwin, D. (2018). Energy-agile design for parallel hpc applications. *Sustainable Computing: Informatics and Systems*, 19:123–134.
- Zhang, H. and Hoffmann, H. (2016). Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *SIGARCH Comput. Archit. News*, 44(2):545–559.
- Zhou, Z., Liu, F., Xu, Y., Zou, R., Xu, H., Lui, J. C., and Jin, H. (2013). Carbon-aware load balancing for geo-distributed cloud services. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 232–241.