2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 979-8-3315-2643-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/IPDPSW66978.2025.00066

Embracing Load Imbalance for Energy Optimizations: a Case-Study

Jelle van Dijk*, Gabor Zavodszky*, Ana-Lucia Varbanescu† and Andy Pimentel*
*University of Amsterdam, Amsterdam, The Netherlands
Email: {jelle.van.dijk, g.zavodszky, a.d.pimentel}@uva.nl
† University of Twente, Enschede, The Netherlands
Email: a.l.varbanescu@utwente.nl

***Abstract*—Scientific computing is a significant consumer of supercomputing resources, and, as a consequence, performance optimization has been a long-term goal of the high-performance computing (HPC) community. However, as the complexity and computational demands of modern scientific applications grow, optimizing energy efficiency becomes critical to balance computational throughput with power constraints.**

To address this challenge, we propose and evaluate a methodology to improve the energy efficiency of large-scale simulations running on multi-node computing systems. Our approach is based on a key observation: when load-imbalance during a large-scale simulation is difficult to avoid or fix, it can at least be exploited to reduce the energy consumption of the simulation. This can be achieved by reducing the CPU frequency of light-loaded nodes to reduce their energy consumption, while incurring minimal overhead and no overall increase in execution time.

We demonstrate this approach in practice through a case-study based on HemoCell, a large-scale scientific framework for cell-resolved blood flow simulation. We show that reducing the node frequency to match the workload proportion per node does reduce the overall energy consumption of the simulation, while only causing a negligible increase in its execution time. For our case-study we observe energy reductions of up to 23% and minimal performance loss compared to the same workloads without frequency scaling.

I. Introduction

The large energy cost of computation at scale has emerged as a critical challenge in high-performance computing (HPC) [1]. The Green500, a list of the top 500 most powerful supercomputers ranked by energy efficiency, demonstrates a yearly improvement in the energy efficiency of HPC platforms. However, efficient HPC platforms are only part of the solution: the other part is energy efficient application development.

The energy consumption of a simulation is proportional to its execution time, which in turn is a function of the parallel efficiency for a given execution. One of the major sources of decreased efficiency is workload imbalance, which leads to the idling of underutilized processes, and ultimately to increased execution time. Addressing this imbalance is not trivial, especially in scenarios where it changes dynamically during the execution. The necessary mitigation strategies typically incur additional computational and communication costs (e.g., associated with check-pointing and workload redistribution). Instead, we hypothesize that throttling down the CPU frequency in accordance to the estimated workload proportion of the underutilized nodes will lead to substantial energy cost reduction while not impacting runtime significantly.

One large-scale simulation affected by dynamic load imbalance is HemoCell, a coupled large-scale scientific framework for simulating cell-resolved blood flow [2], [3], [4]. HemoCell is capable of highly detailed simulations, that provide valuable insights into several health-care related questions where direct experimental methods are limited by current technology [5], [6], [7]. Alowayyed et al. [8], [9] have already shown that dynamic load-imbalance decreases performance in HemoCell. This previous work demonstrated that traditional load-balancing strategies require substantial engineering investments, and dynamic workload rebalancing can introduce significant overhead and performance disruptions. Because of this addressing the (dynamic) imbalance of HemoCell is non-trivial and has not been pursued so far.

For the cases when rebalancing is not feasible, we propose a method to improve the energy efficiency of imbalanced (HemoCell) simulations without redistributing the workload and without a significant increase in simulation wall-clock time. Our method reduces the energy footprint of load imbalance with the help of on-the-fly, *load-specific* frequency scaling of the CPU cores across the compute nodes. Specifically, we *reduce* the frequencies of underutilized CPU cores, thus reducing power consumption with minimal impact on the total runtime, resulting in an overall improvement in energy efficiency. Our method makes use of application insight to decide when and how to apply dynamic frequency scaling.

We apply our method to simulations running on a 16-node cluster, and measure the energy consumption with and without our frequency scaling for the two types of imbalance scenarios in HemoCell. Our results show up to 23% reduction in energy cost using our method.

The remainder of the paper is structured as follows. In Section II we provide a brief introduction to HemoCell. In Sections III and IV we describe the energy-efficiency optimization method, and introduce the experimental setup for the evaluation process. The results are shown in Section V. In Section VI we discuss our method's usability and implementation for HemoCell and feasible extensions to other computational applications. Finally, we provide a brief overview of related work in Section VII and conclude the paper in Section VIII.

II. HemoCell

HemoCell is a simulation code used for cell resolved blood flow modeling. In HemoCell blood is simulated as a dense cellular suspension flow. Blood plasma is modeled as a fluid using the Lattice Boltzman Method (LBM) with the Palabos library [10]. The movement of particles, i.e., red blood cells (RBCs), are modeled separately using a discrete element method [2]. These two models are coupled together via the immersed boundary method [2].

A. Parallelization

The HemoCell *simulation domain* is decomposed into N subdomains, where N is the number of MPI processes. Each MPI process computes its assigned subdomain, and all particles that are located within that subdomain. The size of a HemoCell domain is expressed in lattice units (LU), the conversion from LU to µm is 1 LU = $0.5\,\mu\text{m}$. During the simulation, neighboring processes communicate with each other through MPI messages. The communication consists of the outer layer of each subdomain, and the particles cross subdomain.

B. Workload Imbalance and Quantification

HemoCell, as a coupled simulation code, contains two separate numerical methods: the fluid computation, employing regular (structured) spatial data structures, and the particle computation, which relies on an unstructured numerical grid. These numerical methods have different compute and communication characteristics, and thus experience distinct computational imbalances. In contrast to the particle workload, the fluid workload will not change at runtime and is determined by the details of the specific domain decomposition that is chosen (see e.g. [9]). Alowayyed et al. [8], [9] have already shown that the workload imbalance in the particle simulation model both impacts performance and can increase at runtime. Van Dijk et al. [11] have shown and modeled the impact of static fluid imbalance on the performance of HemoCell.

To quantify the amount of workload imbalance in a simulation, we can use the fractional load imbalance model that describes the amount of load imbalance as a fraction of the average workload [12]. Alowayyed et al. [9] built on the fractional imbalance model and applied it to HemoCell. We briefly revisit the notation here.

The starting point of the fractional overhead model is that the execution time T_p of a parallel application with p processes can be expressed as:

$$T_p = \frac{T_1}{p} + \sum_j T_j^{Overhead} \tag{1}$$

where $T_j^{Overhead}$ is extra cost of performance overheads j, e.g., communication or load imbalance, and $\sum_j$ sums over all overheads. The speedup, S_p, and efficiency, ϵ_p, this application can be written as:

$$S_p = \frac{T_1}{T_p} = \frac{p}{1+\sum_j f_j} \tag{2}$$

$$\epsilon_p = \frac{S_p}{p} = \frac{1}{1+\sum_j f_j} \tag{3}$$

where f_j are fractional overheads, which are written as:

$$f_j = \frac{T_j^{Overhead}}{(T_1/p)} \tag{4}$$

We can use the concept of fractional overheads to express the amount of computational load imbalance f_{li} in a parallel application. We assume that the application has no communication cost and that the time it takes for process i to finish its computation is t_i. Because runtime is dominated by the slowest processors, that is, the highest value of t_i we can write f_{li} as:

$$f_{li} = \left(\frac{t^{max}}{\langle t\rangle}\right) - 1 \tag{5}$$

where t^{max} and $\langle t\rangle$ are the highest and average value from $\{t_0, ...t_{p-1}\}$ respectively.

To interpret with other words, fractional load imbalance is a quantification of the amount of imbalance in a workload. It describes the relation between the most overutilised process compared to the average workload in the system. A fractional load imbalance of zero, i.e., $f_{li} = 0$, indicates no variation in the workload between processes, and thus a balanced system.

III. Methodology

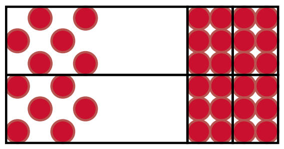
In this section we discuss our method for improving the energy efficiency of imbalanced HemoCell simulations (see Section III-A) and the setup for evaluating its feasibility and success (see Section III-B).

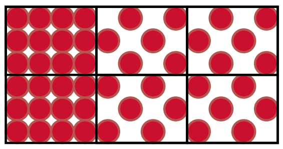
A. Energy Optimization

The proposed energy optimization method is based on two observations: (1) In previous work, van Dijk et al. [11] demonstrated that the overall performance of HemoCell is constrained by the longest-running process. Consequently, the total simulation runtime remains unchanged when modifying the execution times of (other) processes with shorter runtimes, provided that the duration of the longest-running process remains unaltered. (2) Reducing the operational frequency of a CPU core has a dual-effect: a proportional reduction in computational throughput and a reduction in power consumption [13]. Thus, by selectively lowering some CPU-cores' frequencies, we increase the time spent on compute for the selected processes, but reduce their energy footprint. At the same time, the runtime of the longest-running process remains unaltered, thus we effectively reduce the overall power costs of the simulation without compromising the performance. In other words, instead of lowering the runtime of the longest-running process to match the average, we raise the average to meet the maximum time, while saving energy due to lower CPU core frequencies.

B. HemoCell Imbalance Cases

As discussed in Section II, the case-study simulation has two distinct forms of load imbalance: particle imbalance and fluid imbalance. To comprehensively test our energy optimization method, we define three test cases that systematically explore the possible combinations of computational imbalance:


(a) C1: Fluid Imbalance


(b) C2: Particle Imbalance

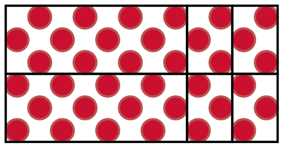

(c) C3: Fluid + Particle Imbalance

Fig. 1: Load imbalance overview in the test cases. Note: the examples here represent a simplified visualization in two-dimensions, the actual simulations run in three dimensions

C1 Fluid imbalance: The imbalance is generated by changing the size of some of the subdomains, but keeping the number of particles per subdomain the same (Figure 1a).

C2 Particle imbalance: The imbalance in this case is generated by changing the number of particles per subdomain, while keeping the size of each subdomain the same (Figure 1b).

C3 Fluid + Particle imbalance: The imbalance in this case is generated by chancing the size of the subdomains, while keeping the concentration of particles (also referred to as hematocrit) of each subdomain the same (Figure 1c).

IV. Experimental setup

In this section we describe the details of the experiment setup used to evaluate the energy optimization method. The code used to run the simulations is split over two publicly available GitHub repositories, one containing the HemoCell case [14], the other contains all the results, scripts for artifact generation, and setup for each scenario [15].

A. Platform

Our experiments are conducted on 16 nodes of a Tier-2 institute HPC cluster [16]. Each node is equipped with a 24-core *AMD EPYC 7402P* processor and 128 GB of memory. On this platform we can control the CPU frequency using *Likwid-setFrequencies* [17]. The platform supports three discrete frequency levels: 1.5 GHz, 2.4 GHz, and 2.8 GHz.

B. Performance and Energy Measurements

Hemocell is instrumented with ScoreP [18], [19] to obtain fine-grained performance metrics. ScoreP instrumentation provides detailed timings of individual code sections, enabling identification of performance bottlenecks and characterization of computational patterns. For energy consumption measurements, we utilize Likwid [17], which interfaces with *Running Average Power Limit* (RAPL) to collect power draw and energy consumption data at the package level for each node. In our analysis, we ignore operations such as initialization and I/O. As such, we restrict our time measurements exclusively to the simulation phase. The corresponding energy consumption is computed by multiplying the average power draw by the measured simulation time. We report total runtime, compute time per process, observed fractional load imbalance, and energy consumption.

C. HemoCell Test cases

The total size of each simulation domain ($600 \times 400 \times 200$ *LU*), is decomposed in multiple ways to create load imbalance. In Table I we list the decomposed domain sizes and the hematocrit value for each scenario. To replicate physiological conditions, the average hematocrit in each case is 40%. To limit variation in the results, each simulation is executed for 500 iterations, and is repeated 3 times.

For each test case, we aim for an imbalance of $f_{li} = 1$, that is, we aim for the maximal compute cost across all processes to be twice as high as the average compute cost. Due to technical limitations, we are unable to create an imbalance scenario C3 with $f_{li} = 1$. Therefore, in this case we create $f_{li} < 1$, and C3-2 with $f_{li} > 1$, apart from this, the two scenarios are identical. We consistently assign high-workload processes to nodes 1 and 2, out of the 16 available nodes. Along with the imbalanced scenarios, we also report the performance and energy of the balanced version of each case, where each process is assigned the same amount of work. The balanced scenario is similar across all the cases, with the same domain size, decomposition, and hematocrit.

TABLE I: Description of the imbalance decomposition for each imbalance scenario. Largest (LD) and smallest (SD) subdomain in lattice units (LU), average hematocrit (H), and maximal (MH) and smallest (SH) hematocrit across the domains

Case	LD [LU]	SD [LU]	H [%]	MH [%]	LH [%]
C1	250x50x50	31x50x50	40	8	65
C2	50x50x50	50x50x50	40	99	33
C3	100x50x50	45x50x50	40	40	40
C3-2	150x50x50	40x50x50	40	40	40

D. Energy Optimization Strategies

Each scenario is run in six different configurations (see Table II): (1) balanced with the default CPU frequencies, (2) imbalanced with the default CPU frequencies, (3) imbalanced at a fixed 2.8 GHz, (4-6) with one of the three optimization

TABLE II: Node Frequency Assignments. Balanced (B), imbalanced (IB), imbalanced at 2.8 GHz (IB@2.8), and optimization strategy 1-3.

	Nodes	
Strategy	1-2	3-16
Balanced	1.5 - 3.35 GHz	1.5 - 3.35 GHz
Imbalanced	1.5 - 3.35 GHz	1.5 - 3.35 GHz
Imbalanced@2.8GHz	2.8 GHz	2.8 GHz
Strategy 1 (S1)	2.8 GHz	1.5 GHz
Strategy 2 (S2)	2.8 GHz	2.4 GHz
Strategy 3 (S3)	2.4 GHz	1.5 GHz

strategies. With the optimization strategies, the goal is to get the average compute time close to the maximal compute time, without increasing this maximum. Because our experiments are setup with a fractional computational imbalance of around 1, where the average amount of work is half that of the most over-utilized processes. We expect that S1 shows the best energy efficiency, because the difference between the lowest and highest frequencies in S1 are closest to 2.

V. Results

In this section we evaluate the energy optimization method for the three imbalanced scenarios discussed in Section III-B. For each case, we report the total runtime, the compute time per process, the observed fractional imbalance, and the energy cost. All results and code needed to reproduce the results are publicly available [15].

C1: Fluid Imbalance

In Figure 2 we show the results for the fluid imbalance case C1. The load imbalance has a clear impact on performance. A fractional imbalance of 1.1, which means that the difference between the average compute time and the maximal compute time is 2.1, results in an almost 80% increase in total execution time and energy cost. Out of the optimization strategies, S3 is the most effective and reduces the energy cost by 12.6% compared to the optimized run.

In one case, the total runtime showed unexpected behavior. We assumed that the total runtime would not be affected by the frequency changes as long as the slowest process did not slow down. Strategy S1 did not increase the compute time of the slowest processes, but it did increase the overall runtime of the simulation. This is a clear sign that the frequency changes had some impact on communication costs. This effect will need to be taken into account in the development of future strategies.

Even considering the slightly increased total runtime, each strategy is able to reduce the energy cost of the imbalanced simulation.

C2: Particle Imbalance

Figure 3 shows the results for the particle imbalance case C2. The impact of particle imbalance on the execution time and the energy cost is significantly smaller than the impact of the same amount of fluid imbalance. A fractional particle imbalance of 1.1 only increases the execution time by 23%, in

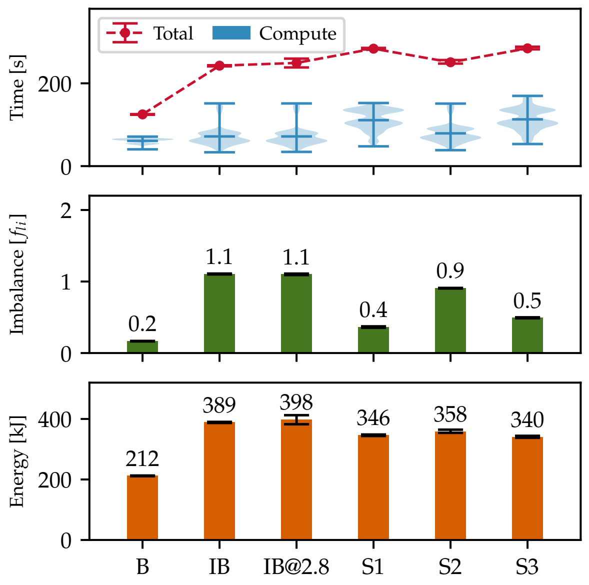


Fig. 2: Results for C1. Min/max/mean of total runtime, distribution of compute, energy costs, and measured fractional imbalance across different configurations: Balanced (B), Imbalanced (IB), and Imbalanced at 2.8 GHz, (IB@2.8), and the strategies S1, S2, and S3

contrast to the 80% increase observed in C1. Analysis of the performance reveals that this difference is mostly attributed to a change in communication overheads, because we observe that the computational times behave similar across all the scenarios. The energy optimization strategies are still very effective, achieving an energy reduction of 23% compared to the imbalanced run with S1.

Similar to C1, we observe a slight increase in the total runtime when applying the optimization strategies. However, still all strategies perform better as compared to the imbalanced run. Notably, S1 consumes approximately the same amount of energy as the balanced run, even with a higher total runtime.

C3: Fluid + Particle Imbalance

In Figure 4 we show the results for C3. We again observe that the imbalance impacts the performance of HemoCell. However, C3 is the first situation where S1 does not reduce the energy cost compared to the imbalanced run. The optimization strategy is effective in removing the imbalance, as we can see from the observed fractional imbalance. However, there is a significant increase in the total runtime. The result is that only S2 reduces the energy cost compared to the imbalanced situation. We expect that this is because the amount of fractional imbalance we observe in C3 is less than 1, i.e., the highest observed computation time is less than double of the mean computation time per process. To verify this in Figure 5 we show the results for case C3-2. C3-2 is set up similarly to C3, but with more imbalance.

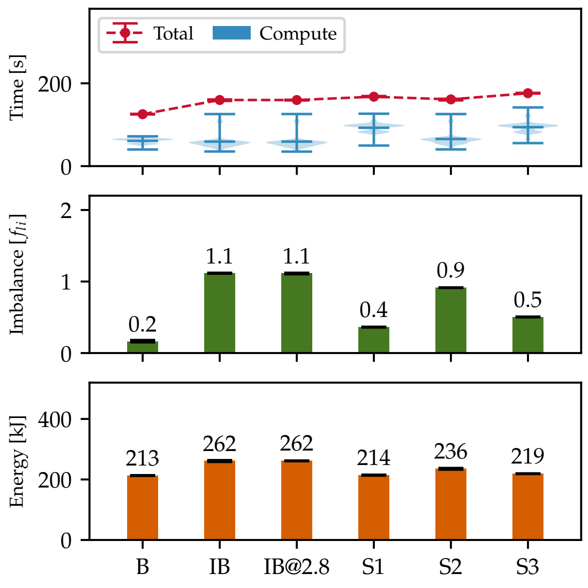


Fig. 3: Results for C2. Min/max/mean of total runtime, distribution of compute, energy costs, and measured fractional imbalance across different configurations: Balanced (B), Imbalanced (IB), and Imbalanced at 2.8 GHz, (IB@2.8), and the strategies S1, S2, and S3

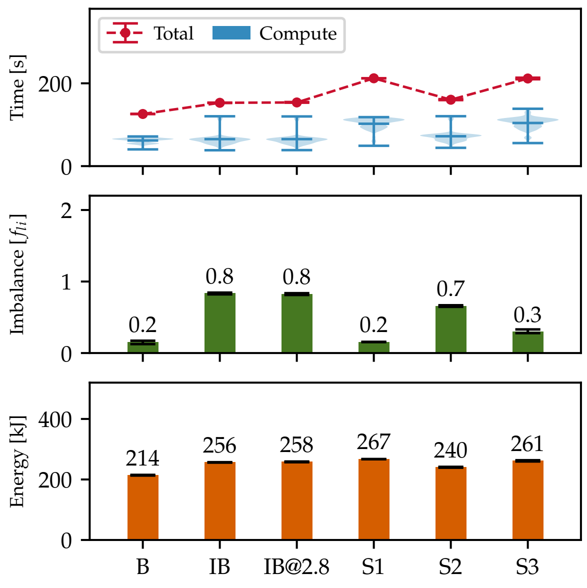


Fig. 4: Results for C3. Min/max/mean of total runtime, distribution of compute, energy costs, and measured fractional imbalance across different configurations: Balanced (B), Imbalanced (IB), and Imbalanced at 2.8 GHz, (IB@2.8), and the strategies S1, S2, and S3

The results for C3-2 are similar to the results of C1 and C2. All strategies reduce energy consumption compared to the imbalanced situation. S3 again shows the largest impact on energy consumption, with an energy cost reduction of 14% compared to the imbalanced case.

VI. Impact considerations

In this section, we discuss the strategy for the implementation of a generic modular framework that is able to detect and address workload imbalance, and improve energy efficiency on-the-fly. We then discuss how this framework could be applicable beyond our use-case.

A. The Framework/Implementation

There are three core components to the framework, (1) Detection, (2) Policies, and (3) Actuation. In this section we describe each and their interaction, and outline the necessary future work to implement this framework as a generic energy optimization tool.

Detection: On-the-fly energy optimization requires real-time detection of workload distribution across processes. In this work, we use use process-level execution time measurements as the detection mechanism. Although this direct measurement approach is very accurate, in production it might incur instrumentation overhead. Another detection approach is modeling the workload imbalance. Previous work has shown that modeling the per-process computational cost of HemoCell is feasible with relatively high accuracy [11].

Policies: The policies are a set of rules that decide when and which actions need to be taken. These decisions are made based on both the input from the imbalance detection, and the limitations of the HPC platform that is used.

The policy outputs a new desired frequency for each node or process. The development of efficient policies will require further effort beyond this work, including a better understanding of the impact of frequency scaling on all overheads (e.g., communication overheads).

Actuation: The actuation component is responsible for enforcing the policies. It is a layer between the abstract policies and the underlying HPC platform-specific frequency management interfaces. This layer ensures framework portability across diverse HPC environments while abstracting the complexity of platform-dependent frequency control implementations from the policy layer.

B. Limitations

The evaluation done in this work indicates that the proposed framework can be a powerful tool for energy optimization. However, there are two notable limitations to consider.

First, the framework's energy optimization capabilities are constrained to computational scenarios where workload rebalancing is not feasible. Our evaluation demonstrates that in situations where load balancing is an option, our method does not provide energy efficiency improvements.

Moreover, imbalance still increases the total runtime as compared to a balanced run. Therefore, even with optimal

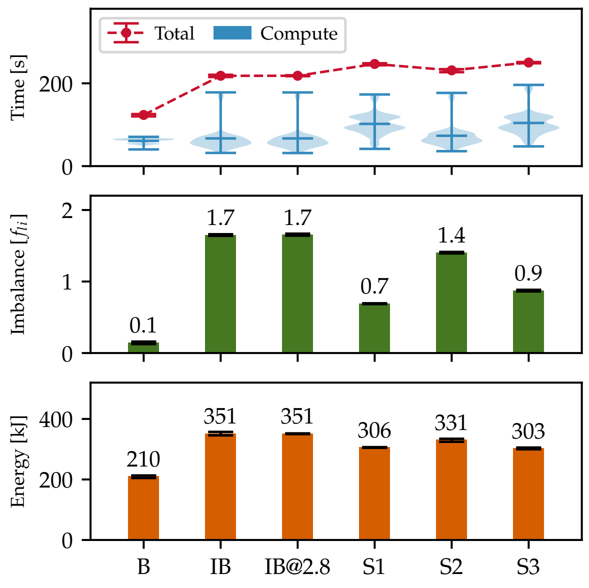


Fig. 5: Results for C3-2. Min/max/mean of total runtime, distribution of compute and energy costs, and measured fractional imbalance across different configurations: Balanced (B), Imbalanced (IB), and Imbalanced at 2.8 GHz, (IB@2.8), and the strategies S1, S2, and S3

energy optimization, the total runtime of an optimized run is still longer compared to a perfectly balanced run.

Second, the framework is dependent on the permissions and tools of the underlying HPC platform. Specifically, on platforms that restrict user control of CPU frequencies, our solution will not be practically feasible.

C. Beyond the use-case

While this work has focused on energy efficiency optimization of HemoCell, the method could potentially be a general-purpose optimization tool for a wide range of large-scale applications across various HPC platforms. The proposed architecture separates the application-specific detection, from the abstract general policies, and the platform-specific policy actuation. This method can be especially beneficial for complex simulations, where the cost and complexity of rebalancing functionality might be prohibitively high.

Our evaluation has shown that the method is robust across multiple types of imbalance in HemoCell, suggesting the potential of broader applicability to large-scale scientific applications. The frameworks modular design enables policy portability, localizing application specificity to the detection component. Further validation across diverse application domains will be necessary to comprehensively evaluate the framework's portability, though initial results demonstrate promising adaptability to varying computational patterns.

VII. Related Work

In this section we provide an overview of existing methods and tools that leverage frequency scaling for energy optimization in HPC environments, highlighting how they differ from our approach.

Dynamic Voltage and Frequency Scaling (DVFS) has long been employed as an energy optimization strategy across various computing domains, including cloud computing [20], [21], networking [22], [23], and embedded systems [24], [25].

In the HPC domain, DVFS remains a widely used technique for reducing energy consumption. Existing DVFS-based tools in HPC environments can be broadly categorized into application-agnostic solutions and specialized approaches. A good example of an application-agnostic solution is the Energy Aware Runtime-system (EAR) [26], [27], a runtime system that sits atop an HPC platform and enables automatic energy optimization for any application using DVFS. EAR is capable of intercepting MPI calls, allowing it to automatically detect iterations within an application. Based on one of two predefined energy strategies, EAR optimizes either energy efficiency or performance by sweeping through a range of frequencies and observing the impact on performance and energy efficiency.

A few other examples of application-agnostic tools include GEOPM [28], a framework for power management and optimization; Adagio [29], a runtime system for optimization of scientific applications; and Cuttlefish [30], a C/C++ library/runtime for energy optimization on Intel processors. All these tools observe the application from the outside, without any prior application knowledge, and tweak frequencies to achieve better energy efficiency and/or performance. Our approach can also be included as a policy in such systems/libraries, but can also act within the application itself (our current approach). However, in the latter case, our approach has the ability to react immediately without the need to first observe a reduction in performance.

Various examples of scheduling policies applicable for HPC systems and applications are included in a detailed survey [31]. Also in this case, our approach can be regarded as an additional, finer-grain DVFS policy.

Although our approach is portable to other applications, it is more akin to application-specific methods, since it utilizes integration with the application through either detailed active monitoring, or via a performance model. In the same class of approaches, Freeh et al. [32] proposed DFVS based on specific phases of the application, while Hsu et al. [33] detected regions of an application and changed frequencies based on the performance of these regions. However, their policies are global rather than local. To our knowledge there is currently no solution that focuses specifically on exploiting node-level load-imbalance to improve energy efficiency of large-scale scientific applications.

VIII. Conclusion and Future Work

In this work we demonstrate the effectiveness of exploiting load-imbalance to reduce the energy consumption of HemoCell, a representative scientific application, through frequency-scaling simulations. Intuitively, our method enables light-loaded nodes to reduce their frequency (and therefore slow-down execution) sufficiently to save energy, with minimal effect to the overall application performance.

Our empirical analysis shows that, with sufficient computational imbalance (relative to the available frequency range), our approach may lead to significant energy savings - as much as 23% compared to the imbalanced, nominal frequency case. The method proves robust across different types of computational imbalance, arising from computational components of different characteristics in the complex use-case simulations (i.e., fluid and particle computations). The amount of energy saved relies only on quantification of the computational imbalance and sufficient range for frequency down-scaling.

Our future work focuses on the design and development of the complete framework around this method. The goal is to observe load imbalance at runtime, use application-independent policies to determine optimal frequencies, and have a portability layer that is able to enact these policies on a wide range of HPC platforms. When combined with various runtime systems, such a framework can enhance energy efficiency for a broad range of large-scale scientific applications.

References

[1] S. Heldens, P. Hijma, B. V. Werkhoven, J. Maassen, A. S. Z. Belloum, and R. V. Van Nieuwpoort, "The Landscape of Exascale Research: A Data-Driven Literature Analysis," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 23:1–23:43, Mar. 2020.

[2] G. Závodszky, B. van Rooij, V. Azizi, and A. Hoekstra, "Cellular Level In-silico Modeling of Blood Rheology with An Improved Material Model for Red Blood Cells," *Front Physiol*, vol. 8, 2017.

[3] G. Zavodszky, B. van Rooij, V. Azizi, S. Alowayyed, and A. Hoekstra, "Hemocell: A high-performance microscopic cellular library," *Procedia Computer Science*, vol. 108, pp. 159–165, 2017.

[4] G. Zavodszky, C. Spieker, B. Czaja, and B. van Rooij, "Cellular blood flow modelling with HemoCell," May 2023.

[5] C. J. Spieker, G. Závodszky, C. Mouriaux, M. van der Kolk, C. Gachet, P. H. Mangin, and A. G. Hoekstra, "The Effects of Micro-vessel Curvature Induced Elongational Flows on Platelet Adhesion," *Ann Biomed Eng*, vol. 49, no. 12, pp. 3609–3620, Dec. 2021.

[6] C. J. Spieker, K. Asteriou, and G. Zavódszky, "Simulating Initial Steps of Platelet Aggregate Formation in a Cellular Blood Flow Environment," in *Comput. Sci. – ICCS 2023*, ser. Lecture Notes in Computer Science, J. Mikyška, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. Sloot, Eds. Cham: Springer Nature Switzerland, 2023, pp. 323–336.

[7] C. J. Spieker, G. Závodszky, C. Mouriaux, P. H. Mangin, and A. G. Hoekstra, "Initial platelet aggregation in the complex shear environment of a punctured vessel model," p. 2023.05.11.540363, Jun. 2023.

[8] S. A. Alowayyed, *Patterns for Multiscale Computing*, 2018, ISBN: 978-94-6323-409-2.

[9] S. Alowayyed, G. Závodszky, V. Azizi, and A. G. Hoekstra, "Load balancing of parallel cell-based blood flow simulations," *Journal of Computational Science*, vol. 24, pp. 1–7, Jan. 2018.

[10] J. Latt, O. Malaspinas, D. Kontaxakis, A. Parmigiani, D. Lagrava, F. Brogi, M. B. Belgacem, Y. Thorimbert, S. Leclaire, S. Li, F. Marson, J. Lemus, C. Kotsalos, R. Conradin, C. Coreixas, R. Petkantchin, F. Raynaud, J. Beny, and B. Chopard, "Palabos: Parallel Lattice Boltzmann Solver," *Computers & Mathematics with Applications*, Apr. 2020.

[11] J. van Dijk, G. Zavodszky, A.-L. Varbanescu, A. D. Pimentel, and A. Hoekstra, "Building a Fine-Grained Analytical Performance Model for Complex Scientific Simulations," in *Parallel Process. Appl. Math.*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2023, pp. 183–196.

[12] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, and A. G. Hoekstra, "Performance evaluation of a parallel sparse lattice Boltzmann solver," *Journal of Computational Physics*, vol. 227, no. 10, pp. 4895–4911, May 2008.

[13] S. Hajiamini, B. Shirazi, A. Crandall, and H. Ghasemzadeh, "A Dynamic Programming Framework for DVFS-Based Energy-Efficiency in Multicore Systems," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 1, pp. 1–12, Jan. 2020.

[14] J. van Dijk, "UvaCsl/hemocell-performance-benchmarks: Release for PDSEC 2025," Zenodo, Mar. 2025, DOI: 10.5281/zenodo.15003102.

[15] ——, "Yelvd/PDSEC25-artifacts: PDSEC25 camera ready," Zenodo, Mar. 2025, DOI: 10.5281/zenodo.15022493.

[16] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term," *Computer*, vol. 49, no. 5, pp. 54–63, May 2016.

[17] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments," in *2010 39th Int. Conf. Parallel Process. Workshop*, Sep. 2010, pp. 207–216.

[18] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir," in *Tools High Perform. Comput. 2011*, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer, 2012, pp. 79–91.

[19] R. Schöne, R. Tschüter, T. Ilsche, J. Schuchart, D. Hackenberg, and W. E. Nagel, "Extending the Functionality of Score-P Through Plugins: Interfaces and Use Cases," in *Tools High Perform. Comput. 2016*, C. Niethammer, J. Gracia, T. Hilbrich, A. Knüpfer, M. M. Resch, and W. E. Nagel, Eds. Cham: Springer International Publishing, 2017, pp. 59–82.

[20] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, "The Cloud Adoption Toolkit: Supporting cloud adoption decisions in the enterprise," *Softw Pract Exp*, vol. 42, no. 4, pp. 447–465, Apr. 2012.

[21] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems*, vol. 37, pp. 141–147, Jul. 2014.

[22] D. Zoni, F. Terraneo, and W. Fornaciari, "A control-based methodology for power-performance optimization in NoCs exploiting DVFS," *Journal of Systems Architecture*, vol. 61, no. 5, pp. 197–209, May 2015.

[23] M. Mahmoudi, A. Avokh, and B. Barekatain, "SDN-DVFS: An enhanced QoS-aware load-balancing method in software defined networks," *Cluster Comput*, vol. 25, no. 2, pp. 1237–1262, Apr. 2022.

[24] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, "Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems," in *8th Int. Symp. Qual. Electron. Des. ISQED07*, Mar. 2007, pp. 204–209.

[25] A. Yeganeh-Khaksar, M. Ansari, S. Safari, S. Yari-Karin, and A. Ejlali, "Ring-DVFS: Reliability-Aware Reinforcement Learning-Based DVFS for Real-Time Embedded Systems," *IEEE Embed. Syst. Lett.*, vol. 13, no. 3, pp. 146–149, Sep. 2021.

[26] J. Corbalan and L. Brochard, "EAR: Energy management framework for supercomputers," in *Barc. Supercomput. Cent. BSC Work. Pap.*, 2019, p. 10.

[27] J. Corbalan, L. Alonso, J. Aneas, and L. Brochard, "Energy Optimization and Analysis with EAR," in *2020 IEEE Int. Conf. Clust. Comput. Clust.*, Sep. 2020, pp. 464–472.

[28] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, "Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions," in *High Perform. Comput.*, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, Eds. Cham: Springer International Publishing, 2017, pp. 394–412.

[29] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS practical for complex HPC applications," in *Proc. 23rd Int. Conf. Supercomput.*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 460–469.

[30] S. Kumar, A. Gupta, V. Kumar, and S. Bhalachandra, "Cuttlefish: Library for achieving energy efficiency in multicore parallel programs," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–14.

[31] B. Kocot, P. Czarnul, and J. Proficz, "Energy-Aware Scheduling for High-Performance Computing Systems: A Survey," *Energies*, vol. 16, no. 2, p. 890, Jan. 2023.

[32] V. W. Freeh and D. K. Lowenthal, “Using multiple energy gears in MPI programs on a power-scalable cluster,” in *Proc. Tenth ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, ser. PPoPP ’05. New York, NY, USA: Association for Computing Machinery, Jun. 2005, pp. 164–173.

[33] C.-H. Hsu and U. Kremer, “The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction,” in *Proc. ACM SIGPLAN 2003 Conf. Program. Lang. Des. Implement.*, ser. PLDI ’03. New York, NY, USA: Association for Computing Machinery, May 2003, pp. 38–48.