Guiding Hardware-Driven Turbo with Application Performance Awareness

Daniel C. Wilson*†, Asma H. Al-rawi*, Lowren H. Lawson*, Siddhartha Jana*,
Federico Ardanaz*, Jonathan M. Eastep*, Ayse K. Coskun†
*Intel Corporation,†Boston University
Email: *{asma.h.al-rawi, lowren.h.lawson, siddhartha.jana, federico.ardanaz, jonathan.m.eastep}@intel.com,
†{danielcw, acoskun}@bu.edu

***Abstract*—Parallel programming across many CPU cores offers many challenges in software design, such as mitigating performance or efficiency loss in applications that reach synchronization points at varying times across the CPU cores. Existing solutions often aim to resolve this through clever optimizations in application design, or by reacting to the imbalance by throttling the CPU core frequency of the early-finishing cores at application run time.**

In this work, we propose a method to rebalance bulk-synchronous MPI applications by selectively *speeding up* the late-finishing cores throughout application run time. This algorithm makes use of the new *Intel® Speed Select Turbo Frequency* feature that enables software to guide the hardware toward increasing the turbo frequency limits of some cores in exchange for decreased turbo frequency limits in other cores. We demonstrate up to 40% energy reduction and 17% execution time reduction in a highly-imbalanced, compute-bound benchmark application and up to 21% energy reduction with 5% execution time reduction in an imbalanced real-world application.

***Index Terms*—energy-aware systems, power management**

I. Introduction

While computers continue to integrate into bigger and faster data centers, their global contribution to power consumption increases. Data centers consumed an estimated 3% of the global energy supply in 2021, twice as much as in the previous decade [1]. Efficiency improvements in data centers range from facility planning, to cluster job management, to chip design and software. We propose a method to guide a CPU's power management decisions for better high-performance computing (HPC) application performance, and to improve energy efficiency of servers as a result.

CPU designers have found several ways to make more efficient use of power-density-constrained systems [2], including *Turbo Boosting* to let a CPU temporarily achieve up to a maximum frequency when a single core is active [3]. When all cores are active, they are constrained to a lower *all-core* turbo frequency, which is applied uniformly across cores even when heterogeneous frequency limits are desired.

A. Heterogeneous Turbo Frequency Limits

The Ice Lake family of Intel CPUs supports new features that allow a subset of cores to reach increased turbo frequencies even when all cores are active [4]. *Intel® Speed Select Technology Core Power* (SST-CP) lets a user specify a priority level for each core in a CPU package, to guide the processor's throttling decisions when it cannot grant requested operating frequencies to all cores. *Intel® Speed Select Technology Turbo Frequency* (SST-TF) extends SST-CP by restricting low-priority cores to a lower operating frequency, allowing the high-priority cores to achieve a higher turbo frequency limit while maintaining CPU design constraints. Since the high-priority turbo frequency limit depends on the number of high priority cores, illustrated in Fig. 1, priority configuration of one core influences the performance of all other high-priority cores. We propose a technique to configure these features to reduce energy consumption and improve performance in imbalanced *bulk-synchronous parallel* (BSP) applications.

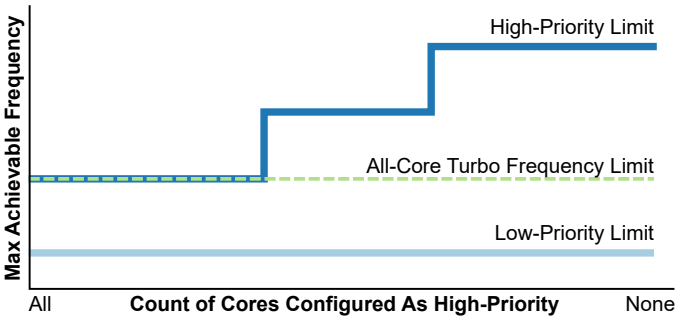

Fig. 1. SST-TF allows a user to exchange lower all-core turbo frequency limits on some cores for increased limits on other cores. The actual frequency limit trade-offs vary by part and can be read from the CPU.

B. Inefficiency Due to Application Imbalance

BSP applications include iterative solvers and simulators common to HPC workloads. Many processes independently solve local problems and halt at synchronization points until all processes reach those points. If processes reach a synchronization point at different times, then the application is *imbalanced*, reducing efficiency due to halted processes.

Workload imbalance may result from hardware and software causes that are not known in advance, such as hardware variation [5] and application input-driven behavior [6]. Applications often partition their work to address imbalance (examples in Section III-C). Middleware solutions often allocate computing resources to match the needs of a running application [7], [8],

Development of the GEOPM software package has been partially funded through contract B609815 with Argonne National Laboratory.

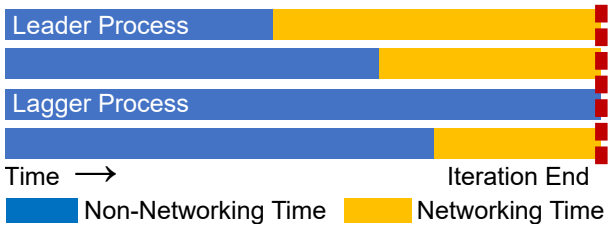

Fig. 2. Leading and lagging processes in an iteration of a bulk-synchronous loop. Leader processes have the shortest execution time per iteration in their compute phases of work. Lagger processes have the longest execution time in their compute phases.

seeking to minimize energy consumption with minimal performance degradation. However, prior work is still constrained by uniform turbo frequency limits even when a system is not constrained by power and thermal design limits.

C. Key Contributions

Our goal is to improve efficiency in imbalanced HPC applications by speeding up critical-path cores and slowing down other cores without increasing the CPU's power limit.

Prior works (discussed in Section V) improve the efficiency of imbalanced bulk-synchronous parallel workloads by throttling the frequency of CPUs that do not reside on an iteration's critical path. This work also *increases* the achievable frequency of CPUs on the critical path, by using application performance awareness to influence the configuration of SST-CP and SST-TF. Through the work in this paper, we observe energy reductions of up to 40% in a benchmark that also exhibits up to 17% reduction in execution time.

Our key contributions are:

- Guidance for using performance-guided turbo in software power management algorithms and application work rebalancers.
- An algorithm that balances bulk synchronous HPC applications by leveraging software-driven critical-path detection and awareness of hardware-imposed CPU frequency trade-offs.
- An evaluation of the energy and performance opportunities from rebalancing bulk-synchronous parallel MPI applications solely with P-State (voltage-frequency levels) control or with SST-TF, and from using both together.

In this paper, we describe an algorithm that improves efficiency by using application awareness to guide power management decisions while executing imbalanced applications. We evaluate performance and energy effects on benchmarks and a real-world application and discuss takeaways for application imbalance and power management algorithms.

II. Algorithm

Our proposed algorithm exchanges turbo frequency limits across CPU cores to balance a bulk-synchronous application. We balance the time spent outside of MPI function calls in an application, shown as *non-networking time* in Fig. 2. We refer to processes as *leaders* or *laggers* when they reach the synchronization point early or late, respectively. We guide CPU's power management unit with priority hints to balance leaders and laggers.

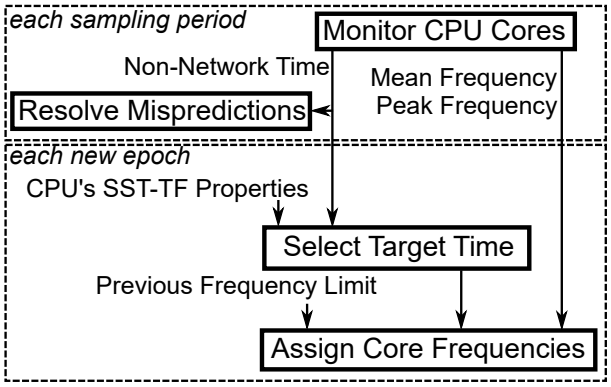

Fig. 3. The algorithm's data flow consists of monitoring and misprediction resolution in each sampling period, as well as predictive rebalancing whenever a new epoch is detected.

We insert a function call in the application's main outer loop of computations to measure performance. Each time this function is called across all processes in the application, the algorithm's *epoch* count increments. The algorithm reduces time between epochs by enabling higher turbo frequencies on the CPU cores that spend the most non-networking time per epoch.

The algorithm initializes all cores as high priority with no frequency limits to start the application with baseline performance. After initialization, a control loop samples program state once every 5 ms, empirically selected as a period that has low impact to our applications under test. When a new epoch is detected, the algorithm sets controls based on predicted performance trade-offs. In case of mispredictions due to changes in application behavior, controls are also adjusted in any sampling period that detects unexpected application state. The overview of these components is shown in Fig. 3. Each component is described in the following subsections.

A. Monitor CPU Cores

Every sampling period, we update a running counter of time spent in non-networking regions of the application. We use the *PMPI* profiling interface for MPI applications to measure time spent in MPI functions.

We monitor `IA32_APERF` and `IA32_MPERF` model-specific registers (MSRs) to compute achieved CPU frequencies as documented in our target processor's software development manual [9]. Average achieved frequency between two points in time is calculated as:

$$f_{\text{achieved}} = f_{\text{base}} * \frac{APERF_t - APERF_{t-1}}{MPERF_t - MPERF_{t-1}} \quad (1)$$

An epoch's peak achieved frequency is approximated as the maximum observed frequency between any two sampling periods within the epoch. The mean achieved frequency is calculated from the first sample of the monitored epoch and the first sample of the next epoch.

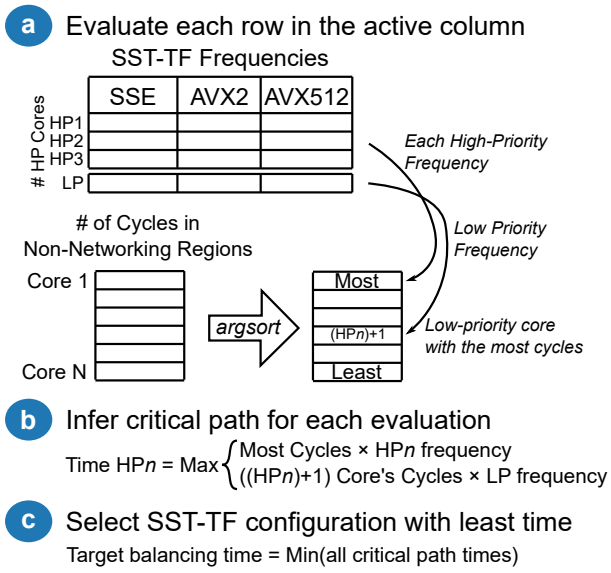

Fig. 4. Target time selection searches for the SST-TF configuration that is estimated to minimize the time spent in an application's critical path. HP*n* bins indicate the available SST-TF high-priority core counts. LP refers to the low-priority SST-TF properties.

B. Select Target Time

When a new epoch is observed in a sample, we select a new target non-networking time for upcoming epochs based on awareness of application performance and hardware frequency trade-offs. The steps to select a target time are described below and depicted in Fig. 4.

The performance risks of poor target time selection are asymmetric. Over-targeting harms performance but under-targeting merely misses opportunities while achieving baseline performance. The algorithm mitigates risk in time selection by ensuring that at least one core is not throttled in each CPU package. Target time is selected from the *expected* non-network time on unthrottled reference cores.

Expected non-network time is determined from the expected frequency of the reference core, which depends on the turbo license level (i.e., the SSE, AVX2, and AVX512 frequency limits reported by the SST-TF interface). The reference core's achievable frequency also depends on the total count of cores configured as high priority in SST-TF. We search for an SST-TF configuration of CPU core priorities that minimizes the expected critical path execution time.

The SST-TF configuration search begins with a target time equal to the non-network time of the reference core. We iterate over each of the high-priority core counts that result in a different maximum achievable frequency (i.e., each row in part *(a)* of Fig. 4). We estimate the critical path time at each stepping point (part *(b)*), and select the SST-TF configuration with the least estimated non-networking time (part *(c)*).

The critical path may move to a different core if the formerly most-lagging core speeds up significantly. Our goal is not to maximize the frequency of the former most-lagging core, but to minimize the time of the new most-lagging core. To achieve that goal, each critical path time is approximated as the greater of times spent in the expected most-lagging high-priority core and the most-lagging low priority core. We apply a simplifying assumption that the most-lagging cores are those with the most non-networking compute cycles, and that the inverse of their non-networking time scales linearly with frequency, as defined below.

$$T_{\text{estimated,core}} = T_{\text{measured}} * \frac{f_{\text{measured,core}}}{f_{\text{expected,core}}} \quad (2)$$

C. Estimating Achievable Frequency

The time estimates described in Section II-B assume that we know the maximum frequency that each core can achieve. The achievable frequency of each core depends both on how many cores are configured as high priority, and on which turbo license level (i.e., SSE, AVX2, AVX512) primarily limits our achievable frequency. The high priority core count is known because we configure the core priorities in each iteration of the algorithm. For the current high-priority core count, we look up the maximum achievable frequencies of each license level, as reported in the SST-TF programming interface. We assume that our workload's license level is the nearest one greater than the core's maximum frequency in the previous epoch. A workload may execute with a mix of license levels within an epoch, but this assumption allows us to avoid over-restricting frequency limits during bursts of license levels with higher achievable frequencies.

D. Assign Core Frequencies

We compute a set of per-core *desired frequencies* that are expected to make each core spend the target non-networking time in the coming epoch. This computation applies the same linearity assumption as eq. (2). We scale the previous epoch's average *measured frequency* by the ratio of *measured non-networking time* to *desired non-networking time*, biased higher by the length of the sampling period with respect to the desired time. The added bias exists to reduce the risk of performance loss due to the impacts of noise over a small count of samples. The desired frequency is calculated as:

$$f_{\text{desired,core}} = f_{\text{measured,core}} * \frac{T_{\text{non\_net,core}}}{T_{\text{desired}}} * \left(1 + \frac{T_{\text{period}}}{T_{\text{desired}}}\right) \quad (3)$$

Target time selection in Section II-B assumes that there is always at least one unthrottled core. We ensure that invariant by scaling the highest-limited core frequency all the way to the processor's maximum frequency. The remaining cores are assigned lower frequencies based on how much less time they spent in non-networking regions of code.

The desired frequency is controlled by either P-States, or SST-TF, or both in combination. Desired frequencies between P-States are rounded up to the next P-State, then applied via the `IA32_PERF_CTL` MSR. When SST-TF is in use, the desired frequencies are used to guide the selection of core priorities for use in the SST-CP interface.

We select a high-priority SST-CP class of service for a core if the desired frequency exceeds the expected low priority frequency (using the license level inference method described

TABLE I
SYSTEM PROPERTIES

Operating System	CentOS Linux 7, Kernel 5.10.0
CPU Model	Intel Xeon Gold 6338T
CPU Packages Per Node	2
Cores Per CPU Package	24
Thermal Design Power	165 W
Min Frequency Limit	0.8 GHz
Base Frequency	2.1 GHz
Max All-Core Turbo Frequency	2.7 GHz
Max Single-Core Frequency	3.4 GHz

TABLE II
SYSTEM PACKAGE SST-TF FREQUENCY LIMITS

	Max Frequency (GHz)		
High-Priority Core Count	SSE	AVX2	AVX512
up to 8	3.3	3.3	3.2
up to 12	3.0	3.0	2.9
up to 16	2.8	2.8	2.7
(Low Priority)	2.1	1.8	1.5

in Section II-B). Otherwise, a low priority class of service is selected. In the SST-CP interface, priorities range from 0 to 3, with 0 as the highest priority. We use priorities 0 and 3 for high and low priority, respectively. The interface allows us to specify ranges of desired frequencies for each priority level. We conservatively set the low-priority level's limit to the CPU's base frequency. SST-TF may enforce a lower limit, depending on workload properties. We set the high-priority level to allow the entire turbo range of frequencies.

E. Resolve Mispredictions

Mispredictions of target time and target frequencies may occur from simplifying assumptions about application behavior and due to rapid changes in application imbalance. We detect and react to those cases in each control period.

The algorithm applies a simplifying assumption that each epoch consists of a single networking region and a single non-networking region. In reality, a single epoch may enter and exit networking regions many times. By considering only the aggregate time spent in the region of interest, we may miss opportunities if significant time is spent in the intermediate networking regions within an epoch. We mitigate this risk by deprioritizing cores that are in a networking region for multiple samples in a row. We only apply this heuristic when *multiple* such samples are observed, based on performance evaluations by Cesarini et al. [8].

If all the frequency-unlimited cores are in networking regions for multiple samples, this may indicate either that we are within an intermediate networking region or that we over-restricted the frequency of our low-priority cores. In that case, we remove the frequency restrictions applied to the cores that are still in non-networking regions of code.

III. EXPERIMENTAL SETUP

A. Computing Environment

Our experiments require Intel Ice Lake or later CPUs that support the *Intel® Speed Select Technology - Turbo Frequency* feature. We use a single server with properties outlined in Table I.

The processor reports its SST-TF frequency limits to the user through a Linux driver [4]. Table II illustrates the maximum all-core turbo frequency under different workload types, as reported by the driver on our experimental platform. The frequency limits of low-priority cores are constant across different counts of high-priority cores.

B. Measurement Tools

Measurement and control of our server are implemented in the Global Extensible Power Manager (GEOPM) job runtime [7]. GEOPM offers *IO groups* to sample and control system state, and *agents*, which implement power and performance management algorithms. GEOPM uses `msr-safe` [10] to read and write MSRs in batch operations.

We introduce a new IO group to interact with SST-CP and SST-TF from GEOPM, and we add a new *frequency_balancer* agent to balance imbalanced MPI applications using SST-TF. The IO Group interacts with the SST driver [4]. We grant the Linux `CAP_SYS_ADMIN` capability to our evaluated applications so we can use this driver for our experiments.[1]

We use the existing *monitor* agent to measure baseline energy and performance. This agent reads requested metrics without modifying system power and performance controls. We reserve one core to execute the monitoring infrastructure separate from the applications in this work.

C. Applications

We evaluate how the power management policies respond to imbalanced benchmarks from the NAS Parallel Benchmarks suite [11], the Mantevo suite [12], and a micro-benchmark with controllable imbalance. We evaluate a real-world application in the form of a simulation in LAMMPS [6].

1) NPB IS: The NPB IS benchmark sorts a collection of integers by partitioning numbers into buckets, which are sorted in parallel before merging into the result. Imbalance manifests when buckets end up with significantly different sizes after partitioning the collection. The benchmark partially addresses imbalance by combining small buckets in MPI processes.

2) NPB BT-MZ: The NPB BT-MZ benchmark solves a partial differential equation by dividing the space into a mesh of zones that can be solved in parallel, with periodic exchanges of boundary values between neighboring zones. The benchmark statically balances the work by bundling smaller zones in a process. If the MPI process count is sufficiently large, there may not be enough small zones to match the amount of work performed by the larger zones, causing imbalance. We execute class *B*, which exhibits high imbalance on our server.

3) Mantevo MiniFE: MiniFE serves as a proxy for the main phases of unstructured finite element applications, with most of its time spent performing sparse matrix multiplication. MiniFE

[1] Future work may configure the recently released `geopmd` service to enable unprivileged access to individual components of the SST-TF interface.

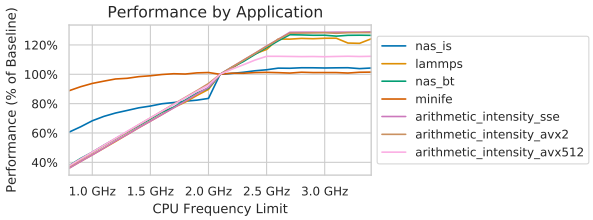

Fig. 5. Performance of selected benchmarks under various frequency limits. Performance (inverse of execution time) is shown relative to performance at the CPU's base frequency.

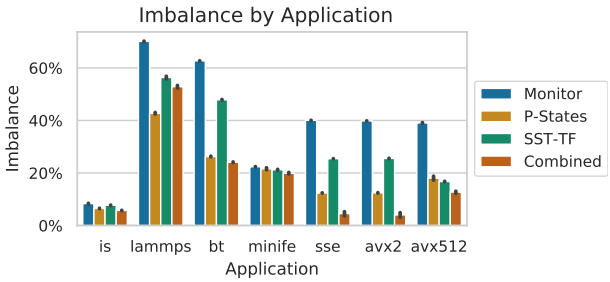

Fig. 6. Imbalance of the selected benchmarks under monitor-only runs and under frequency-controlled runs. Frequency control is evaluated with P-States only, SST-TF only, or a combination of the two controls. 0% indicates perfectly balanced, 100% indicates a lack of concurrent work. Error bars indicate the 95% confidence interval over 5 trials.

is well-balanced by default, but it has a command-line option to simulate imbalanced scenarios. We set this option to 50.

4) LAMMPS: LAMMPS (Large-Scale Atomic/Molecular Massively Parallel Simulator) partitions its simulation space into subdomains that are assigned to processes. Imbalance is likely to surface when the input problem cannot be evenly distributed among processes, or when particle density changes throughout simulation. We simulate a collapsing water column in a container with obstacles. The simulation begins in a balanced state but rapidly changes the distribution of work across CPUs as simulation progresses. The rapid changes in this application's imbalance are a motivating factor for the adjustments described in Section II-E.

5) Arithmetic Intensity Micro-benchmark: Prior work in cluster-wide rebalancing [13] introduces a set of micro-benchmarks[2] that trigger different workload-dependent frequency limits, or *license levels* [14]. These micro-benchmarks are executed in an imbalanced configuration with twice as much work on 8 processes per CPU package so that only the application's 8 slowest-progressing processes need to be configured as high-priority in SST-CP. If our algorithm places at least one core in the wrong configuration, then the application's performance will be limited.

D. Measured Properties

We measure the imbalance and frequency sensitivity of each application to establish our expectations for opportunities to rebalance these workloads using performance-guided turbo.

1) Frequency Sensitivity: Fig. 5 shows the frequency sensitivity of each application. We execute each application under different CPU frequency limits. Performance is measured as the inverse of the time spent in each application's main compute loop and compared to the performance of each application with a frequency limit set to the CPU's base frequency.

The drop in performance for IS near 2.0 GHz occurs because the power control unit selects a lower uncore frequency setting when *all* cores are configured with low core frequency controls. The algorithm ignores this behavior since it always has at least one unthrottled core per CPU package.

Applications with peak performance near the all-core turbo frequency limit may perform better with increased turbo limits. As expected, IS and MiniFE benchmarks do not achieve much improved performance beyond the processor's base frequency; they are documented to be more memory-sensitive applications. The varying level-off points of the frequency-sensitive applications result from multiple effects: first, they spent different amounts of frequency-sensitive time on their critical paths, and second, the different vectorization types have different all-core turbo frequency limits [3].

2) Imbalance: We calculate imbalance similar to prior work in load imbalance detection [15], as $\frac{T_{max}-\bar{T}}{T_{max}} * \frac{n}{n-1}$ for the time T each CPU core spends in non-networking sections of code, over n CPUs used by the application. 0% imbalance indicates a perfectly-balanced application where all processes completed their work in the same amount of time, whereas 100% means that a single process performed all the work.

Fig. 6 shows the measured imbalance of each application. The applications with greater imbalance offer more opportunity for rebalancing. If they are not CPU-frequency-bound, as shown in Fig. 5, then we can only expect energy savings. Otherwise, we also expect opportunities to improve performance with guided turbo.

IV. Results

Each application responds differently to each of the three frequency-control variants described in Section II. One variant only applies the desired frequency control setting through P-States. Another only configures SST-CP classes of service with SST-TF enabled. Lastly, a combined variant applies both sets of controls. These are all compared to baseline performance from monitoring-only runs, as described in Section III-B.

A. Achieved Frequencies

Fig. 7 shows achieved core frequencies in each application. The monitored runs of most applications are able to achieve the system's all-core turbo frequency of 2.7 GHz, while the other policies achieve varied ranges of core frequencies.

P-States alone often cannot increase peak frequency even when they heavily throttle some of the cores, as in `nas_bt`, since all-core turbo is already achievable. But the runs with SST-TF successfully increase the peak achievable frequency.

The *AVX512* micro-benchmark is frequently throttled due to voltage regulator design constraints, so it spends a significant

[2]https://github.com/dannosliwcd/arithmetic-intensity

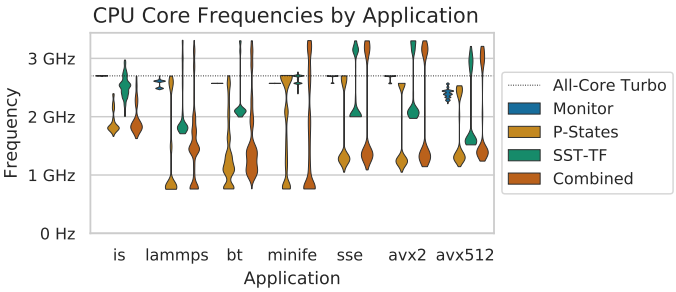

Fig. 7. Distributions of average frequency across CPU cores and across time in monitor-only runs and under frequency-control variants of the algorithm described in Section II.

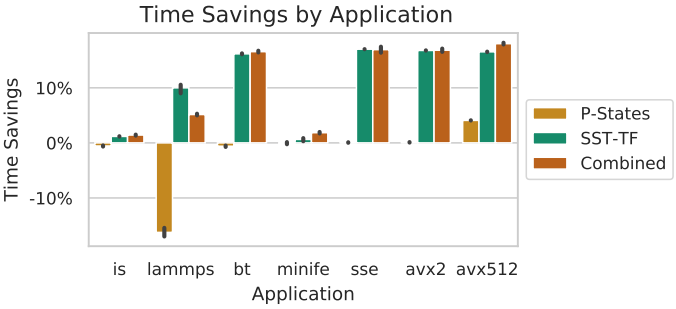

Fig. 8. Time savings for imbalanced applications under different frequency-control variants of the algorithm described in Section II. Savings are relative to monitor-only runs. Error bars indicate the 95% confidence interval over 5 trials.

amount of samples at lower frequencies. While the upper end of achieved frequencies is lower with the P-State-only approach, the distribution is shifted such that the cores on the critical path are throttled less often.

B. Execution Time Savings

Time savings are summarized in Fig. 8. We see improvement in each of the imbalanced application configurations, with one case showing potential for future enhancement.

All three of the arithmetic intensity micro-benchmarks are able to achieve nearly the same performance improvement with SST-TF because their frequency-sensitive critical paths have about 2.5x as much work as the rest of their processes, and they operate in the highest-frequency SST-TF configuration for the duration of their runs. Although the BT-MZ class B input results in more imbalance than the amount configured in the micro-benchmarks, we do not seize additional performance improvement since we are already achieving the maximum-allowed frequency under this server's SST-TF properties.

P-State-only control of LAMMPS is the only case with significant performance degradation. The degraded performance comes from the rapidly-changing imbalance in that application, where around 8 application epochs execute between each algorithm sampling period. For example, the algorithm may observe little to no non-networking time on a CPU core in one period, so it assigns a low frequency limit to that core. But new work may shift to that core by the next period, impacting the performance of multiple epochs in the meantime. Although

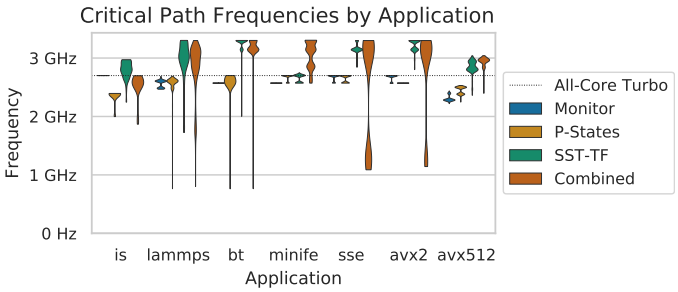

Fig. 9. Distributions of average achieved CPU core frequency for cores that spend the most non-networking time in each iteration of the application.

the *combined* algorithm variant has the same limitation, it is able to mitigate the performance loss by speeding up the cores that are consistently given a lot of work by the application.

The P-State-only variant avoids performance degradation in all the applications that exhibit steady imbalance. This is expected since most of the applications reach the all-core turbo frequency without throttling any cores. The AVX512 micro-benchmark exhibits performance *improvement* because it is often throttled when all cores are in the high-power-consuming region of interest. The P-State-only solution throttles the leader cores, creating headroom for the lagger cores to achieve higher frequencies. The SST-TF and combined algorithm variants achieve more performance improvement because they enable higher peak frequencies in the lagging cores' region of interest.

Fig. 9 shows achieved frequencies on critical-path cores across application iterations. In the AVX512 example, we see that proactive throttling enables us to *increase* the average frequency on critical-path cores. Furthermore, SST-TF enables greater peak frequency on critical paths.

In Section III-D1, we observe that MiniFE benchmark is insensitive to higher frequencies above the processor's base frequency. However, increased peak turbo frequencies on the application's critical path does offer some performance improvement opportunities for MiniFE. This benchmark spends a significant amount of time in sparse matrix vector multiplication regions of code. While those regions are typically non-compute bound, they can exhibit moderate temporal locality as values in the vector are repeatedly accessed for the computation. This indicates that there may be performance opportunities for selectively higher turbo frequencies in imbalanced applications even if they do not appear to be largely core-frequency-sensitive at a high level.

C. Energy Savings

Energy savings in Fig. 10 result from both power reductions and performance improvements. The P-State-only variant reduces energy despite showing little to no execution time improvement in Fig. 8, indicating power-based savings. The additional energy savings on the combined variant are from the reduction in execution time that comes with higher achievable frequencies.

The BT-MZ benchmark shows greater energy savings than the other evaluated applications due to its heavy imbalance. We

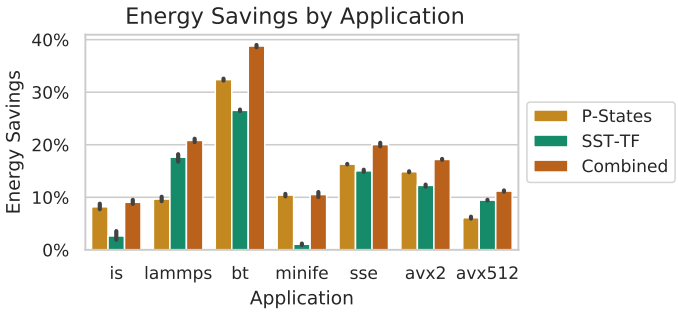

Fig. 10. Energy savings for imbalanced applications under different frequency-control variants of the algorithm described in Section II. Savings are relative to monitor-only runs. Error bars indicate the 95% confidence interval over 5 trials.

execute the benchmark's class *B* input, which cannot be evenly distributed across all application processes on our server. The most-lagging process receives about 4 times as much work as the process that finishes each iteration first. As a result, many of the cores executing this application can be throttled to very low frequencies. We observed that BT-MZ did not achieve more performance improvement than the imbalanced micro-benchmarks in Section IV-B because we were already executing the application's critical path at the maximum-allowed SST-TF frequency. However, we do observe greater energy-saving opportunity through throttling the non-critical-path CPU cores.

A useful takeaway from this comparison is that compute-bound, imbalanced applications stand to *reduce* their energy consumption by making use of performance-guided turbo. Although the less compute-bound imbalanced applications are able to achieve most of their energy savings from P-State throttling alone, we are able to convert some energy savings back to performance improvement when bursts of turbo utilization may improve performance.

D. High-Level Results Takeaways

We see a few takeaways with respect to the impact of turbo guidance in power management algorithms as well as with respect to the applications where such algorithms may benefit.

1) Power Management Algorithms: The algorithm variants discussed in this paper utilize the same core rules to determine critical path and to select the CPU core frequency limits. Although the same decisions are applied across variants, there are differences in behavior by applying different combinations of P-State control and SST-TF configuration.

The P-State-only approach is typically limited to reducing energy consumption while limiting performance degradation. However, in cases where application performance is limited by power constraints (*power-bound*) such as the AVX512 micro-benchmark, proactively throttling some cores enables speedups on other cores. Aside from workloads that are naturally power-bound, this may be useful for algorithms intended to rebalance applications on power-capped systems.

The SST-TF-only approach improves performance by increasing the peak turbo frequency on an application's critical path. Our algorithm applies a conservative limit to the low-priority cores. As a result, the SST-TF-only approach responds to the rapidly-changing CPU demands of the LAMMPS experiment with less risk to performance degradation. However, the conservative approach leaves behind more imbalance (Fig. 6) compared to the solutions with more aggressive throttling on the low-priority end, and ultimately leaves some energy savings on the table.

2) Applications: Our experiments cover applications with multiple characteristics of imbalance. We have the greatest opportunity when the critical path is co-located on a CPU package with non-critical paths that can be slowed down. Some applications (e.g., LAMMPS) have application-level work rebalancers that are aware of the hardware topology. On systems capable of performance-guided turbo configuration, application-level rebalancers could consider how to spread any unresolved imbalance across CPU packages if it is not in conflict with other resource-oriented rebalancing objectives.

V. Related Work

We introduce a method to combine P-State frequency control with mixed turbo frequency limits in a CPU package, in order to improve performance in imbalanced applications. Related works in P-State control, turbo boosting, and application imbalance explore a variety of techniques to mitigate the effects of imbalance but do not leverage performance trade-offs of user-configurable limits to CPU frequency boosting.

A. P-State Control

Countdown [16] throttles frequency when in a long-lasting network region. This and some of its related works use non-MPI time as a metric of interest when making decisions, as we do here. Later work [8] separately models slack time, networking wait time, and networking copy time. The model is used to balance the non-networking time and to select efficient configurations for networking code through frequency control.

Other works, such as Cuttlefish [17] and EAR [18], search for core and uncore frequency limits that achieve efficiency objectives in one or more regions of an application, based on the region's sensitivity to those limits. Our work assumes a simple linear relationship between performance and frequency. Future iterations may seek to model region-specific sensitivity to achieve energy savings in already-balanced workloads.

B. Turbo Boosting

Recent work demonstrates how SST-BF can be used to give high base frequencies to cores running higher-priority workloads [19]. SST-BF exchanges *base* frequency on *platform-defined* sets of CPU cores, whereas SST-TF exchanges *turbo* frequency ranges on *user-defined* sets of CPU cores. Our solution targets servers running one bulk-synchronous HPC workload at a time, where we define core priorities to adjust for existing imbalance.

The Poseidon algorithm [20] increases turbo frequencies of OpenMP workloads by packing work into a smaller number of cores, allowing unused cores to enter deeper C-States so the

remaining active cores can achieve higher turbo frequencies. Instead of controlling the load on each core, our solution influences achievable core frequency through core prioritization.

Wamhoff et al. improve efficiency in multi-threaded workloads by throttling threads that are waiting for locks to speed up threads that hold locks [21]. Our work similarly aims to increase achievable frequency on the critical path, but on bulk-synchronous MPI applications. Our work both increases the achieved critical-path frequency by proactive throttling, *and* increases the peak achievable frequency with SST-TF.

C. Application-Level Work Balancing

Applications often rebalance their own work. For example, the NPB benchmarks evaluated in these experiments statically balance their work by decomposing the problem into many smaller sub-problems of varying size and distributing the sub-problems across processes to balance the work per process. For example, the LAMMPS application distributes sections of its simulation space across MPI processes by recursive coordinate bisection [6]. The balancer can be invoked dynamically as part of input script configuration.

VI. Conclusion

In this work, we present an algorithm that combines application performance awareness with knowledge of hardware frequency limit trade-offs to reduce energy *and* increase performance in imbalanced bulk-synchronous MPI workloads. Our evaluations with online sampling-based profiling show energy reductions up to 40% with performance improvements of up to 17% in highly-imbalanced and compute-bound benchmarks. We also demonstrate up to 21% energy reduction accompanied by 5% performance improvement in a real-world application that exhibits compute-bound imbalance.

Our evaluation of the results discusses further opportunities for improvement to the algorithm, and highlights takeaways that may be useful to other designers of algorithms in applications and in software power management. The presented algorithm locally balances cores in an individual server, but future work may extend the algorithm to balance across multiple servers.

References

[1] "ACM TechBrief: Computing and climate change," *ACM Technology Policy Council*, Nov. 2021.

[2] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC Design Automation Conference 2012*, 2012, pp. 1131–1136.

[3] "Optimizing performance with Intel® Advanced Vector Extensions," https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf, Sep. 2014.

[4] S. Pandruvada. (2020) Intel(R) speed select technology user guide. [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/pm/intel-speed-select.html

[5] A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree, "An empirical survey of performance and energy efficiency variation on intel processors," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, ser. E2SC'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3149412.3149421

[6] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[7] 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 Performance Computing*, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, Eds. Cham: Springer International Publishing, 2017, pp. 394–412.

[8] D. Cesarini, A. Bartolini, A. Borghesi, C. Cavazzoni, M. Luisier, and L. Benini, "Countdown slack: A run-time library to reduce energy footprint in large-scale mpi applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2696–2709, 2020.

[9] "Intel® 64 and ia-32 architectures software developer's manual, volume 3b: System programming guide, part 2," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, Apr. 2022.

[10] M. J. McFadden, K. S. Shoga, S. Brink, B. L. Rountree, T. Patki, C. Cantalupo, D. Guttman, B. Geltz, and B. Allen, "msr-safe," aug 2019. [Online]. Available: https://doi.org/10.11578/dc.20200513.3

[11] "NAS parallel benchmarks," https://www.nas.nasa.gov/software/npb.html, Jan. 2022.

[12] P. S. Crozier, H. K. Thornquist, R. W. Numrich, A. B. Williams, H. C. Edwards, E. R. Keiter, M. Rajan, J. M. Willenbring, D. W. Doerfler, and M. A. Heroux, "Improving performance via mini-applications." 9 2009. [Online]. Available: https://www.osti.gov/biblio/993908

[13] D. C. Wilson, S. Jana, A. Marathe, S. Brink, C. M. Cantalupo, D. R. Guttman, B. Geltz, L. H. Lawson, A. H. Al-rawi, A. Mohammad *et al.*, "Introducing application awareness into a unified power management stack," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2021.

[14] "Intel® 64 and ia-32 architectures optimization reference manual," https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html, Feb. 2022.

[15] L. DeRose, B. Homer, and D. Johnson, "Detecting application load imbalance on high end massively parallel systems," in *Euro-Par 2007 Parallel Processing*, A.-M. Kermarrec, L. Bougé, and T. Priol, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 150–159.

[16] D. Cesarini, A. Bartolini, P. Bonfà, C. Cavazzoni, and L. Benini, "Countdown: A run-time library for application-agnostic energy saving in mpi communication primitives," in *Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems*, ser. ANDARE '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3295816.3295818

[17] S. Kumar, A. Gupta, V. Kumar, and S. Bhalachandra, "Cuttlefish: Library for achieving energy efficiency in multicore parallel programs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476163

[18] J. Corbalan, O. Vidal, L. Alonso, and J. Aneas, "Explicit uncore frequency scaling for energy optimisation policies with ear in intel architectures," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 572–581.

[19] P. Veitch, J. J. Browne, and C. MacNamara, "Resource tuning for energy efficient slicing," in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2021, pp. 100–104.

[20] S. M. Marques, T. S. Medeiros, F. D. Rossi, M. C. Luizelli, A. C. S. Beck, and A. F. Lorenzon, "Synergically rebalancing parallel execution via DCT and turbo boosting," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 277–282.

[21] J.-T. Wamhoff, S. Diestelhorst, C. Fetzer, P. Marlier, P. Felber, and D. Dice, "The TURBO diaries: Application-controlled frequency scaling explained," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 193–204. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/wamhoff