SAND2017-11028C

Improving Energy Efficiency in Memory-constrained Applications Using Core-specific Power Control

Sridutt Bhalachandra
Department of Computer Science,
UNC Chapel Hill
sriduttb@cs.unc.edu

Allan Porterfield
Department of Computer Science,
UNC Chapel Hill
akp@cs.unc.edu

Stephen L. Olivier
Center for Computing Research,
Sandia National Laboratories
slolivi@sandia.gov

Jan F. Prins
Department of Computer Science,
UNC Chapel Hill
prins@cs.unc.edu

Robert J. Fowler
RENCI,
UNC Chapel Hill
rjf@renci.org

ABSTRACT

Power is increasingly the limiting factor in High Performance Computing (HPC) at Exascale and will continue to influence future advancements in supercomputing. Recent processors equipped with on-board hardware counters allow real time monitoring of operating conditions such as energy and temperature, in addition to performance measures such as instructions retired and memory accesses. An experimental memory study presented on modern CPU architectures, Intel Sandybridge and Haswell, identifies a metric, *TORo_core*, that detects bandwidth saturation and increased latency.

TORo_core is used to construct a dynamic policy applied at coarse and fine-grained levels to modulate per-core power controls on Haswell machines. The coarse and fine-grained application of dynamic policy shows best energy savings of 32.1% and 19.5% with a 2% slowdown in both cases. On average for six MPI applications, the fine-grained dynamic policy speeds execution by 1% while the coarse-grained application results in a 3% slowdown. Energy savings through frequency reduction not only provide cost advantages, they also reduce resource contention and create additional thermal headroom for non-throttled cores improving performance.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Hardware** → **Power and energy**; **Testing with distributed and parallel systems**; • **Software and its engineering** → **Runtime environments**;

KEYWORDS

Memory constrained applications, Energy efficiency, Dynamic Voltage and Frequency Scaling, Core-specific Power Control

E2SC'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5132-4/17/11…$15.00
https://doi.org/10.1145/3149412.3149418

ACM Reference Format:
Sridutt Bhalachandra, Allan Porterfield, Stephen L. Olivier, Jan F. Prins, and Robert J. Fowler. 2017. Improving Energy Efficiency in Memory-constrained Applications Using Core-specific Power Control. In *E2SC'17: E2SC'17: Energy Efficient Supercomputing, November 12–17, 2017, Denver, CO, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3149412.3149418

1 INTRODUCTION

Energy efficiency in high performance computing (HPC) will be critical to limit operating costs and carbon footprints in future supercomputing centers. For an Exascale machine to be delivered within a 20 Megawatt power budget, as suggested by the Exascale Computing study [16], an energy efficiency of about 50 GigaFlops/Watt is desired. Physical variations arising during the fabrication process induce variations in energy usage and performance among otherwise identical processors [24]. Software variations due to varying data locality, compiler optimizations, thread count among others cause energy variability of 20% in general and in extreme cases over 2x [22]. External factors like cooling also vary the energy consumed by an application. These factors suggest a need for dynamic power regulation to achieve energy efficiency.

Disparity between CPU and memory speeds and the latency of accessing DRAM across the system bus contribute significantly to CPU cycling while waiting on memory and wasting power. Memory operations are seldom explicit and coarse-grained enough for the operating system to stall (or switch off) cores and reduce power as memory accesses are serviced. For certain classes of applications that are memory-bound, reducing the processor speed or using related approaches like CPU throttling for power savings has shown little adverse impact on performance, or in some cases slight speedup from reduced contention [22, 29]. The present work focuses on detecting situations in which an application is memory-bound during execution and using core-specific power controls to reduce CPU power consumption.

Most research to regulate energy and performance in software has revolved around Dynamic Voltage and Frequency Scaling (DVFS) [14, 15, 25]. Because of past hardware limitations, these previous DVFS methods impacted all cores on a multi-core processor and potentially slowed the critical

path. With the introduction of per-core voltage regulators in Intel Haswell, each physical core (or pair of logical cores if using Hyper-Threading) can be independently controlled to allow target reduction of only non-critical threads.

Recent advancements in memory technologies have attempted to close the gap between CPU and memory processing times. It is important to ascertain the severity of the memory bottleneck in modern HPC systems, to see if there exist opportunities to pursue energy efficiency research addressing that bottleneck. We analyze the bandwidth and latency on modern CPU architectures like Intel Sandybridge and Haswell with different memory configurations. The study reveals bandwidth saturation and increased latency where CPU frequency can be reduced without affecting performance. We evaluate several on-board hardware counters to identify a usable metric that can capture the above opportunities during execution. The metrics are then used to characterize HPC applications based on their memory activity. Using the evaluation a new memory metric is proposed to construct a dynamic policy that can lower frequency of individual cores using per-core DVFS on Haswell machines, to reduce power and save energy with minimal performance impact.

The major work and ideas presented in this paper are:

- An experimental memory study on modern CPU architectures, Intel Sandybridge and Haswell, with different memory configurations.
- A memory characterization of HPC applications using on-board hardware counters to identify metrics that can guide application of power controls.
- A dynamic policy with coarse and fine-grained application that uses TORo_core metric with per-core DVFS during program execution. Policy validation is performed using six mini-apps (*HPCCG, AMG, CloverLeaf, miniFE, miniGhost, miniMD*). The evaluation shows the best energy savings with coarse and fine-grained versions is 32.1% and 19.5% respectively. Also, the performance is seen to improve in a few cases, more often with the fine-grained version.

2 MEMORY PERFORMANCE

The memory study undertaken is similar to the one in our previous work [19] on Dual and Quad-socket AMD Opterons and Intel Nehalem. Recent HPC systems are evaluated to better understand the improvement in memory performance over previous generations. The goal is to determine when memory bottlenecks exist.

2.1 PCHASE Benchmark

PCHASE [4] is specifically used in a mode to test memory throughput under carefully controlled degrees of concurrent access. Each thread executes a loop with a controllable number of independent "pointer chasing" operations per iteration. Each sequence of pointer addresses is pseudo-random and designed to defeat hardware prefetching while limiting TLB misses. A wrapper script around PCHASE is added that iterates over different numbers of memory reference chains (thereby controlling memory concurrency) and threads for each PCHASE run. This forces sequential memory accesses within a thread. References from different chains can be issued concurrently. By varying the number of chains, number of threads and placement of threads, contention in most parts of the downstream path and components can be measured.

2.2 Experimental Setup

The memory performance is obtained by running PCHASE on Sandybridge (SB16) and Haswells (HW20, HW32).

SB16 is an M620 Dell Blade with two Intel Xeon E5-2680 CPUs at 2.7GHz containing eight cores each and 64 GB of main memory. It has 20MB cache per CPU (40MB total). Each DDR3 DIMM is 4GB in size, with 8 out of the 12 available slots for each CPU populated. This configuration allows the memory to run at its maximum speed of 1600MHz.

HW20 is an Dell PowerEdge R730 with two Intel E5-2650v3 CPUs at 2.3GHz containing 10 cores. The LLC size is 25MB per CPU (50MB total). The DDR4 memory with a chip size of 4GB is placed in 4 out of the 12 slots available for each of the CPU (32GB total). The maximum memory speed is 2133MHz. Hyperthreading has been disabled on SB16 and HW20, and both run Linux kernel 2.6.32.

The 32-core Haswell machine (HW32) runs Linux kernel 3.17.8. It has two Intel(R) Xeon(R) E5-2698 V3 CPUs, each with 16 cores at 2.3GHz and hyperthreading enabled. The total memory is 128GB and cache size is 50MB on each. The DDR4 16GB chip is populated in 4 out of the 8 slots available and clock at maximum speed of 2133MHz.

2.3 Results

Figure 1 shows total memory bandwidth (in MB/s) and effective latency (in ns) with varying outstanding memory references. For each system, the lowest bandwidth and latency occurs when there is one memory reference per core. Bandwidth increases almost linearly without increasing latency as the system becomes better utilized at two references per core. As the references per core increases beyond two, queuing and contention within the system increases latency.

A considerable improvement in bandwidth is observed from SB16 to HW20 & HW32 emphasizing the benefit from DDR3 to DDR4 memory. Better bandwidth does not always guarantee better latency, which is subject to core count on chip (Figure 1). HW20 is seen to be the fastest in terms of latency above three outstanding references per core. The gap increases slowly as the number of outstanding references increases. In contrast, HW32 is not only slower than HW20, but also SB16. It is slower at all concurrency levels and increases as contention increases. Core count and type/number of DIMMs is still an important factor while determining the overall memory performance. An poorly provisioned newer chip can have higher latency than an older chip.

Peak bandwidth on all three machines is achieved with 5-7 outstanding memory references. Beyond this, the memory bandwidth no longer increases, but the effective latency continues to increase. For example, at 24 memory references

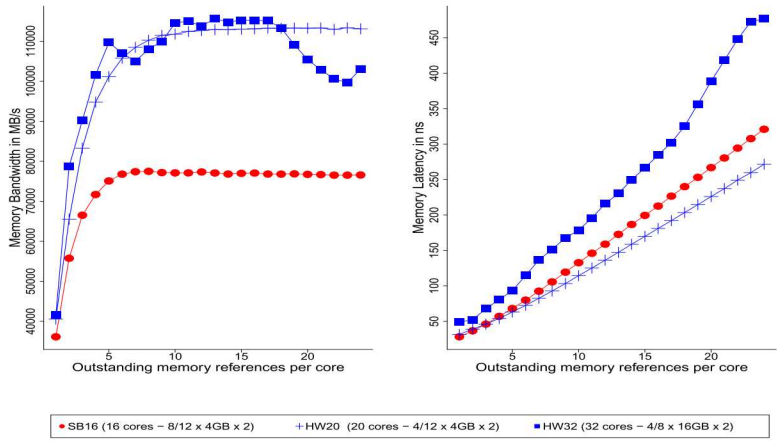

Figure 1: Memory Bandwidth & Latency using PCHASE on Sandybridge (SB16) and Haswell (HW20, HW32)

per core the latency (476.83ns) on HW32 is about 10 times that of at one memory reference (49.13ns). Even though hyperthreads weren't used, a non uniform memory bandwidth curve is seen for HW32 beyond 5 memory references. The low level at which memory bandwidth reaches saturation and consequent increased latency presents an opportunity to improve energy efficiency by reducing cpu frequency when the offered memory load exceeds saturation level.

3 INFRASTRUCTURE

The modern Intel architectures can monitor performance of both core and uncore components.

3.1 Energy measurement using RAPL

Starting with the SandyBridge implementation [1], Intel extended the X86-64 architecture to include "Running Average Power Limit" (RAPL) interface that provides model specific registers (MSRs) to measure and control microprocessor energy/power utilization. Included in the interface is a **MSR_PKG_PERF_STATUS** counter which records the energy since boot in 15.3 nanojoule increments. It is a 32-bit counter and rollover occurs frequently. Counting the number of rollovers gives a very fine-grained measurement.

3.2 Uncore Subsytem

The uncore sub-system of Intel architecture consists of caching agent (CBo), power controller unit (PCU), integrated memory controller (iMC) and home agent (HA) among other components. It facilitates per-component performance monitoring (PMON) through sets of counter registers. In this work, we use the counter registers that are part of the CBo to monitor all core transactions that access the Last Level Cache (LLC). The transaction monitoring is supported using 44-bit wide counters (**Cn_MSR_PMON_CTR{3:0}**) [2].

3.3 RCRdaemon

RENCI has developed the Resource Centric Reflection (RCR) daemon [21] to provide hardware performance counter information to any program interested in how a system is performing. It focuses on counters for resources that are shared by more than one hardware core (e.g. energy consumption, Last Level Cache performance). RCRdaemon runs at root level for supervisor-privilege or can use *msr-safe* [26]. A simple API delineates a code region for measurement with a start and end call. Performance information is provided to various clients through a shared memory region containing no pointers using simple volatile loads and stores.

3.4 System

A thirty-two blade partition of the Shepard Advanced Systems Technology Test Bed at Sandia National Laboratories is used for all memory characterization and policy experiments. This development partition exposes power and energy instrumentation along with control of clock frequency and modulation through *msr-safe* and other kernel modules. The Penguin nodes are connected with Mellanox Fourteen Data Rate InfiniBand and have configuration similar to HW32. The hyperthread enablement has no effect on our experiment as benchmarks are run only on one thread per core and frequency is changed by same amounts for both logical threads. The cluster runs Red Hat Enterprise Linux Server 6.8 with Mpich 3.2 and is scheduled by Slurm 2.3.3-1.18chaos.

3.5 Measurement Techniques

All reported power, energy and temperature numbers are obtained with RAPL through the RCRdaemon. RCRdaemon has been extended to provide additional performance related metrics associated with frequency, instructions retired and cache accesses. Modern systems have enough internal heterogeneity that execution times often vary by several percent run to run [20]. The average is taken over 12 test runs for

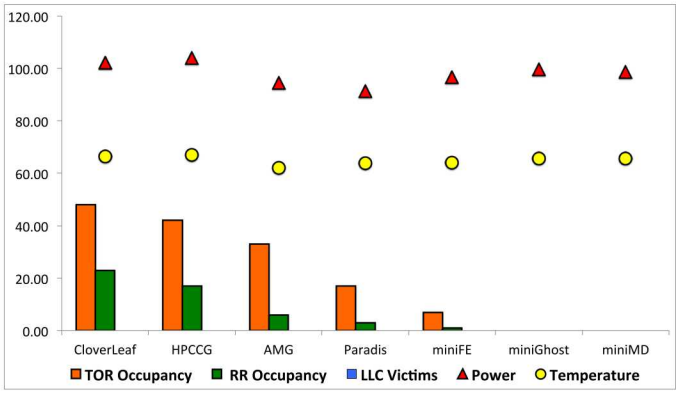

Figure 2: Memory characterization of applications on 32 Haswell Nodes (1024 cores)

each power and software setting. To avoid energy variation with temperature, each test script ignored results from the first several minutes until the system temperature was stable.

3.6 Applications

A number of DOE MPI mini-apps from the Mantevo suite [10] and NERSC-8/Trinity benchmarks [3] were selected to simulate a variety of loads on HPC systems.

CloverLeaf simulates the material behavior under stress using a 2-D Eulerian formulation. The input used is the provided "clover_bm512_short.in".

HPCCG is an approximation to an unstructured implicit finite element simulation but generates a synthetic linear system. The chosen problem size is 90x120x150.

AMG is a parallel algebraic multigrid solver. A 27-point stencil on a cube with size 60x60x60 and PCG with diagonal scaling as solver is run on 16, 8 and 8 cores along three axes.

miniFE approximates an unstructured implicit finite element simulation. The problem size is 225x375x525 with 'load_imbalance' factor at 100 to exploit max load imbalance.

miniGhost simulates highly structured stencil operations (halo exchange pattern). A problem size of 30x30x30 is spread across 1024 cores with 16, 8 and 8 cores along the three axes.

miniMD is a light-weight molecular dynamics application (LAMMPS). The Lennard-Jones input file has problem size of 160x160x160 and runs for 100 steps on 1024 cores.

4 CHARACTERIZING MEMORY

As uncore performance monitoring can capture numerous events in various components, choosing an appropriate metric that detects memory saturation and latency increase presents a challenge. With shared LLC, each core has to create requests for memory locations not in its private cache into the Table of Requests (TOR). Hence, it was reasonable to begin the search to find our metrics by analyzing the events associated with LLC, particularly TOR. From the many events monitored in the CBo, three memory metrics captured using **Cn_MSR_PMON_CTR{3:0}** were chosen. The metric value from each core is summed for the entire length of application and then divided by the number of memory updates that happened in the RCRdaemon during this period.

The first metric, TOR occupancy (TORo) captures all valid requests in TOR, including that reside even for a short time, such as LLC Hits that do not need to snoop cores or requests that get rejected and have to be retried through one of the ingress queues.

$$TORo = \frac{\sum_0^t \sum_{i=0}^n (TOR\_Occupancy_i / Clock\ cycles_i)}{Number\ of\ Memory\ updates}$$

The second metric, RR occupancy (RRo) is the average number of entries in the Ingress Request Queue (IRQ) on Address (AD) Ring. The AD Ring is associated with core read/write requests and Intel QPI Snoops. It also carries Intel QPI requests and snoop responses from core to Intel QPI. This metric is supposed to be a subset of TORo such that applications with high R×R_Occupancy generally have higher TORo.

$$RRo = \frac{\sum_0^t \sum_{i=0}^n (RR\_Occupancy_i / Clock\ cycles_i)}{Number\ of\ Memory\ updates}$$

The final metric, LLCv is the average number of lines that are victimized on a LLC fill.

$$LLCv = \frac{\sum_0^t \sum_{i=0}^n (LLC\_Victims_i / Clock\ cycles_i)}{Number\ of\ Memory\ updates}$$

In the above two equations; n is the number of cores per socket, t is the running time of the application. The memory value in the RCRdaemon is updated every 1ms.

The results for memory characterization of the chosen applications on 32 Haswell nodes (1024 cores) is shown in Figure 2 along with average power consumption and temperature. TORo, RRo and LLCv are dimensionless ratios.

The metrics TORo and RRo classify the applications along a spectrum with cache/memory bound applications on the left of the plot and compute bound applications to the right. For *miniGhost* and *miniMD* that are completely compute intensive applications the two metric values are 0. LLCv is not a viable metric for characterizing application as compute or memory bound as even for highly memory constrained

applications – *CloverLeaf, HPCCG and AMG* the value for LLCv is 0. Overall both TORo and RRo are able to support characterization/classification of the applications. However, TORo has a better resolution w.r.t applications in the middle of the spectrum. TORo is used in the policies next to capture the opportunities as seen in the evaluation with PCHASE.

5 RUNTIME POLICY

The insights from characterization were used to formulate a dynamic policy based on TORo to control power. The policy is implemented within the MPI profiling interface (PMPI) and requires no application code changes. Calls to `MPI_Init`, `MPI_Finalize` are intercepted to start collection of power, memory and other metrics using RCRdaemon, while other MPI calls like `MPI_Send`, `MPI_Recv` among others are intercepted to collect TORo values and make policy decisions. Since the policies are core-specific, the TORo value collected by the RCRdaemon at a core level between the MPI calls is used (denoted by *TORo_core* hereafter) eliminating the need for communication with other MPI ranks. The application has to link against our MPI library in addition to the standard MPI library. For controlling power using DVFS, the *acpi-cpufreq* or other applicable kernel modules need to be loaded.

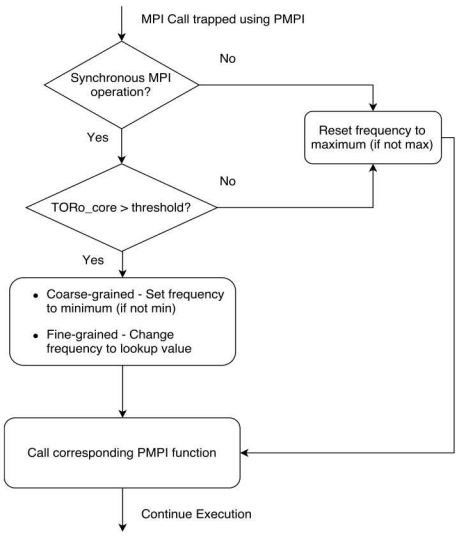

Figure 3: Flowchart of the dynamic policy using TORo_core as the metric to throttle power using per-core DVFS

Figure 3 describes actions taken at each MPI call by the dynamic policy in coarse and fine-grained versions. For coarse-grained, the threshold value is empirically obtained and set at the beginning of execution. The fine-grained dynamic policy uses multiple threshold values to change the core frequency.

5.1 Coarse-grained application

The coarse-grained dynamic policy (CDP) targets applications that are well-known to be memory intensive like *HPCCG* (Figure 2). For empirically determining the threshold value for evaluation, we ran HPCCG and miniMD run on HW32 with different TORo_core threshold values as shown in Figure 4. Initially, the applications run as-is in their default (D) configuration with power, execution time and energy consumption recorded. Cores for which the TORo_core value is above the specified value are forced to run at their lowest frequency (1.2GHz). The threshold value is varied from 0 (almost all cores run at lowest frequency for the entire length of execution) to 10 (almost all cores run at max frequency).

For *HPCCG*, which is heavily memory constrained (Figure 2), there is very low performance impact of running the application at lowest frequency for the entire duration. Consequently, at TORo_core=0, the application power is at its lowest point at 91.1W (30.6% reduced) along with the energy consumption (3091.4J – 25.8% reduction) as the increase in execution time is marginal at 3.7% (33.94s). At TORo_core=4, the application power consumption increases drastically to 108.4W and the execution time increase at this point is reduced to 1.6%. This suggests that for a highly memory-bounded application like *HPCCG* majority of its memory references are concentrated at a value of less than four on the TOR_core scale. Between a TORo_core value of five to eight, the power and performance is almost flat followed by drastic increase in power at TOR_core=9 where the values are comparable to default. This indicates there exists a second level of concentration for the memory references.

The compute-bound application (Figure 2), *miniMD*, sees a steep increase of about 16% in execution time at TORo_core=0. Even though the power is reduced by 30.4%, the energy increases by 8%. Each increase in threshold level thereafter is seen to reduce the performance impact with increasing power consumption. At TORo_core=10, the values match that of the default configuration. The prevailing phenomenon is indicative of the direct impact of frequency reduction on performance of a compute bound application. For highly compute bound applications like *miniMD*, the TOR_core scale is not suitable to improve energy efficiency and such applications are best left alone by our proposed schemes.

For HPCCG the lowest energy point is TORo_core=0. The PCHASE evaluation (Figure 1) shows that the memory bandwidth is no longer linear beyond two outstanding memory references. As the energy consumption curve is almost flat upto TORo_core=3, we choose a value of 1.0 (one outstanding request in the TOR per core). The value of TORo_core=1.0 also allows some headroom for the frequency decisions avoiding overfitting. However, the value for TORo_core can be set to any desired value at runtime using environment variables.

5.2 Fine-grained application

Figure 4 reveals that memory references are concentrated on multiple levels of the TORo_core scale with steps at TORo_core=4 and TORo_core=9. The fine-grained dynamic policy (FDP) aims to take advantage of this information to have a much finer control on frequency to make the TOR_core-scale based policy broad. This further helps the policy to

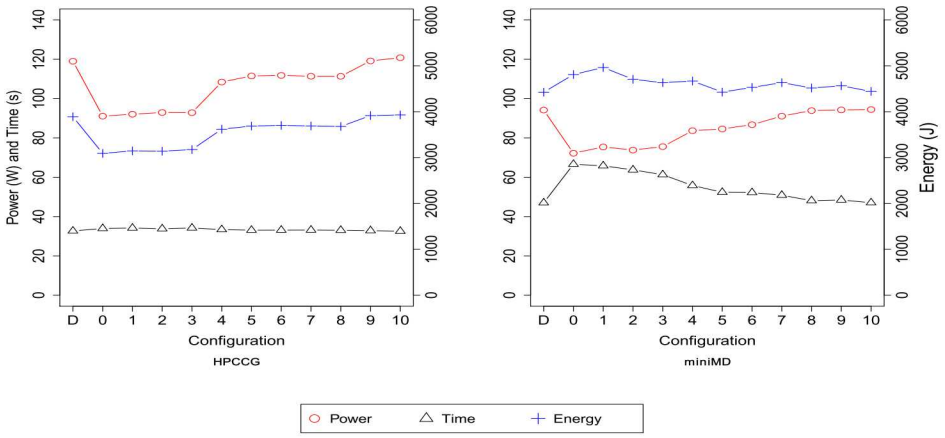

Figure 4: Effect of using different TOR_core threshold values on HPCCG (memory constrained) and miniMD (compute bound) application energy and execution time within coarse-grained dynamic policy (CDP). Note that the values are discrete with links only serving as visual cues.

target applications that lie between the extremes of the memory characterization spectrum by controlling degradation (Figure 2). The memory references for a memory intensive application are spread across different values on the TORo_core scale with regions of concentration. The DVFS operational range levels are mapped to TORo_core values and the frequency is set to the value at a particular TORo_core value during execution. The effective average TORo_core value at which the core has to be slowed down is determined during execution. For example, in our evaluation between a TORo_core value of 0 to 0.5 the mapped core frequency is 2.3GHz (max non-turbo frequency supported). Beyond a TORo_core value of 9.0, the mapped core frequency is 1.2GHz. During execution the core frequency ranges from [1.2, 2.3] GHz. The effective frequency at which the core needs to run for an application is adaptive, making the performance deterioration lower. In contrast, with **CDP** the frequency reduction is fixed and is set at the beginning of execution.

6 RESULTS

The result for mini-applications with the two policies is shown in Figure 5. The energy consumptions and execution times are normalized and the default values for each application is provided. At the onset, the policies are seen to target applications that are only memory constrained and not compute-bound. The most memory constrained application – HPCCG is seen to have largest reduction in energy with both policies. While the least memory constrained – miniGhost and miniMD are mostly untouched. This establishes the validity of TORo_core as the right metric that captures the conducive behavior required to improve energy efficiency seen in Figure 1.

For *HPCCG* using FDP, the energy saving is 19.5% and program execution time is increased by 1.9%. With CDP the increase in execution time is slightly more at 2.4%, but the reduction in energy is 32.1% owing to large power reduction of 33.7%. The large energy efficiency improvement with CDP can be attributed to its high memory-intensive nature.

CDP not only reduces energy consumption (17.8%) for *AMG*, but also improves performance by 2.2%. Even with FDP the performance improves by 0.6% saving 6.4% in energy. These cases evince that energy optimization improves performance in certain scenarios.

CloverLeaf is an atypical example where we see that even though its average TOR occupancy is the highest (Figure 2), the energy savings is much lesser than *HPCCG*. It reduces the power with CDP by up to 24.7%, however the execution time is increased by 3.6% resulting in energy savings of 24.7%. Using FDP, we are able to reduce the performance degradation up to 0.4% saving 12.0% in energy. This shows the benefit of FDP to control performance deterioration within permissible levels of run-to-run variation. Profiling *CloverLeaf* revealed a large presence of asynchronous calls (`MPI_Isend` and `MPI_Irecv`). The frequency at which these calls run determines execution time. To overcome this, both policies were slightly modified to run the asynchronous calls at 1.8GHz instead of 2.3GHz. The energy reduction without slowdown in case of FDP suggests that even though asynchronous calls are able to hide memory latency to an extent, there exist further opportunities to improve energy efficiency.

Applications *miniFE*, *miniGhost* and *miniMD* are mostly cache/compute bound (Figure 2). Neither policy reduces power or adversely impacts performance, as necessary for a policy meant to only target memory-constrained scenarios.

miniFE sees a slight increase in energy (2.1%) owing to increase in power (1.4%) and execution time (0.7%) using CDP. With FDP the increase in execution time is nullified and the execution time and power is close to default. It is important to note here that even though the load_imbalance=100, there

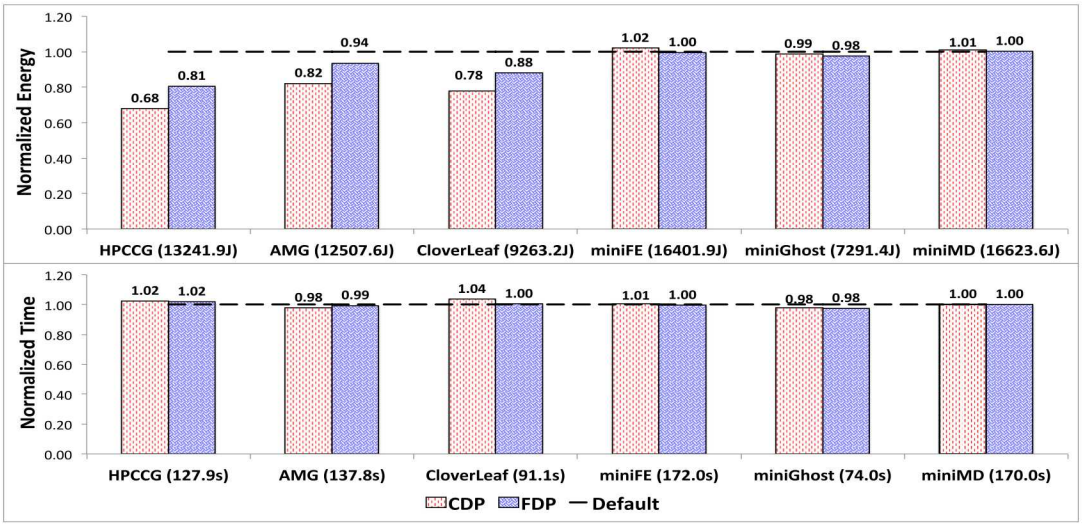

Figure 5: Energy consumption and execution times using coarse (CDP) and fine-grained (FDP) dynamic policy for mini applications

is no change in its memory footprint. It continues to remain compute-bound and is unaffected by the dynamic policy.

miniGhost performance improves with both CDP (2.0%) and FDP (2.5%). The power slightly increases with CDP (0.6%) therefore the energy saving is 1.4%. With FDP power is on par with default and energy savings commensurate with speedup (2.5%), showing yet another scenario showing energy optimization leading to performance improvement.

For *miniMD* using CDP, power is increased slightly by 0.8% and execution time by 0.1% to increase energy consumption by 0.9%. The FDP runs at same speed as that of default and consumes similar power.

Overall, **FDP** saves considerable amount of energy at lower performance degradation even with smaller power reduction compared to **CDP**. It is more suitable for applications whose memory behavior is not well known, as CDP may adversely impact performance. The best energy savings with CDP and FDP are 32.1% and 19.5% for *HPCCG*. The cases where the policies have shown to improve performance may be attributed to lowering of frequency decreasing resource contention or adding thermal headroom for non-throttled cores.

7 RELATED WORK

Software research in energy efficient computing fundamentally involves characterization of power/energy consumption of applications, architectures and infrastructure with an intent to design novel methods that promote optimal utilization of computational components at all levels of hierarchy. System-software approach that leverages accurate workload characterization via a unique synthesis of hardware performance counters in order to determine when and how to use DVFS to improve power efficiency while strictly maintaining performance has been proposed [12]. A fine-grained microarchitecture-driven DVFS mechanism that scales down voltage and frequency upon individual off-chip memory accesses using on-chip regulators is proposed in [8]. Runtime Energy Saving Technology (REST) [18] utilizes DVFS to modify frequencies of cores at runtime without prior knowledge using memory versus non-memory phase detection system on Xeon architectures. Our previous work [29] shows chip-wide DVFS has high power state switching overhead, preventing its use when a more fine-grained technique is necessary. Low transition overhead of CPU clock modulation is leveraged and applied to fine-grained OpenMP parallel loops.

Research involving slack reclamation to amortize the effect of uneven work distribution have been proposed [13–15]. Green Queue [28] automates the process of finding phases and optimal frequencies using power models. Automatic tuning of applications based on software level performance-related tunables has also been explored[23]. A number of efforts use hardware performance counters [7, 17, 27] to compute optimal off-line settings. Several projects estimate energy based on hardware counters with direct correlation including cache access [9], MIPS [11] and CPU stall cycles [12]. Moving beyond DVFS duty cycle modulation [20], power capping [24] along with similar mechanisms on IBM Power 6 and 7 (capping) and AMD Bulldozer have been explored.

While addressing memory bottlenecks, mostly chip-wide DVFS has been used to develop coarse-grain solutions that may suffer from the pitfalls of DVFS chip-wide effect. Some approaches use static analysis and may not always be flexible to adapt to dynamic changes. Our previous works [5, 6] attempt to go beyond chip-wide DVFS and propose dynamic policies that use core-specific power controls. However, these works target scenarios with computational workload imbalances. The current work targets memory-constrained applications proposing a dynamic policy that is core-specific in nature and makes decisions during execution.

8 CONCLUSION & FUTURE WORK

Despite faster memory transfer rates and simultaneous transfers through multiple memory controllers, the memory bottleneck remains a frequent limitation in HPC. As *Big Data* and other memory-intensive applications become more main stream in HPC, the bottleneck is likely to stay. There is likely to remain an opportunity to save energy in memory-bound applications. The TORo_core metric proposed identifies memory behavior exhibited by the applications conducive to constructing runtime energy saving policies.

Further understanding of advantages and disadvantages of the proposed metric and policy is needed. This can involve finding memory-constrained applications for assessment, and identifying when and where each of the two policies is better suited. Other metrics like instructions per cycle may also aid the policy decisions to further improve energy efficiency.

9 ACKNOWLEDGEMENT

The authors thank Anirban Mandal, RENCI for helping with PCHASE and Nathan Gauntt, SNL for helping with Shepard cluster. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This work was performed under the auspices of the U.S. Department of Energy XPRESS project under Contract DE-SC0008704 and Office of Science SciDAC SUPER Institute on grant DE-SC0006925.

REFERENCES

[1] Intel Corporation. *IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, 2015.*

[2] Intel Corporation. *Intel Xeon Processor E5 and E7 v3 Family Uncore Performance Monitoring Reference Manual, 2015.*

[3] NERSC-8/Trinity Benchmarks. *http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks.*

[4] pChase benchmark page. *https://github.com/tycho/pChase.*

[5] Sridutt Bhalachandra, Allan Porterfield, Stephen L Olivier, and Jan F Prins. 2017. An Adaptive Core-Specific Runtime for Energy Efficiency. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International.* IEEE, 947–956.

[6] Sridutt Bhalachandra, Allan Porterfield, and Jan F Prins. 2015. Using Dynamic Duty Cycle Modulation to Improve Energy Efficiency in High Performance Computing. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International.* IEEE, 911–918.

[7] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. 2004. Dynamic Voltage and Frequency Scaling Based on Workload Decomposition. In *Proc. of the 2004 Intl. Symposium on Low Power Electronics and Design.*

[8] Stijn Eyerman and Lieven Eeckhout. 2011. Fine-grained DVFS using on-chip regulators. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 1 (2011), 1.

[9] R. Ge, X. Feng, W. Feng, and K.W. Cameron. 2007. CPU Miser: A Performance-directed, Run-time System for Power-aware Clusters. In *ICPP 2007: 36th Intl. Conference on Parallel Processing.* IEEE.

[10] M Heroux and R Barrett. 2012. Mantevo Project Homepage. (2012).

[11] C. Hsu and W. Feng. 2005. A Power-aware Run-time System for High-performance Computing. In *SC05: Proc. of the 2005 ACM/IEEE Conference on High Performance Networking and Computing.* IEEE Computer Society.

[12] S. Huang and W. Feng. 2009. Energy-efficient Cluster Computing Via Accurate Workload Characterization. In *CCGrid 2009: Proc. of the 9th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid.* IEEE Computer Society.

[13] Jaeyeon Kang and Sanjay Ranka. 2010. Dynamic Slack Allocation Algorithms for Energy Minimization on Parallel Machines. *J. Parallel and Distrib. Comput.* 70, 5 (2010).

[14] Nandini Kappiah, Vincent W Freeh, and David K Lowenthal. 2005. Just in Time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing.* IEEE Computer Society, 33.

[15] Hideaki Kimura, Mitsuhisa Sato, Yoshihiko Hotta, Taisuke Boku, and Daisuke Takahashi. 2006. Emprical Study on Reducing Energy of Parallel Programs Using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster. In *CLUSTER 2006: Proc. of the 2006 IEEE Intl. Conference on Cluster Computing.* IEEE.

[16] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, W Carson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, et al. 2008. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. (2008).

[17] Charles W. Lively, Xingfu Wu, Valerie E. Taylor, Shirley Moore, Hung-Ching Chang, Chun-Yi Su, and Kirk W. Cameron. 2012. Power-aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems. *Computer Science - R&D* 27, 4 (2012).

[18] Kelly Livingston, Nicolas Triquenaux, Thibault Fighiera, Jean Christophe Beyler, and William Jalby. 2014. Computer using too much power? give it a rest (runtime energy saving technology). *Computer Science-Research and Development* 29, 2 (2014), 123–130.

[19] Anirban Mandal, Rob Fowler, and Allan Porterfield. 2010. Modeling memory concurrency for multi-socket multi-core systems. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on.* IEEE, 66–75.

[20] Allan Porterfield, Rob Fowler, Sridutt Bhalachandra, and Wei Wang. 2013. OpenMP and MPI Application Energy Measurement Variation. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing.* ACM, 7.

[21] Allan Porterfield, Rob Fowler, and Min Yeol Lim. 2010. *RCRTool Design Document; Version 0.1.* Technical Report RENCI Technical Report TR-10-01.

[22] Allan K Porterfield, Stephen L Olivier, Sridutt Bhalachandra, and Jan F Prins. 2013. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International.* IEEE, 884–891.

[23] Shah Mohammad Faizur Rahman, Jichi Guo, Akshatha Bhat, Carlos Garcia, Majedul Haque Sujon, Qing Yi, Chunhua Liao, and Daniel Quinlan. 2012. Studying the Impact of Application-level Optimizations on the Power Consumption of Multi-core Architectures. In *Proceedings of the 9th conference on Computing Frontiers (CF '12).* 123–132.

[24] Barry Rountree, Dong H. Ahn, Bronis de Supinski, David K. Lowenthal, and Martin Schulz. 2012. Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound. In *HPPAC 2012: Proc. of the 8th Workshop on High Performance, Power-Aware Computing.*

[25] Barry Rountree, David K. Lowenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler K. Bletsch. 2009. Adagio: Making DVS Practical for Complex HPC Applications. In *ICS '09: Proc. of the 23rd Intl. Conference on Supercomputing.*

[26] Kathleen Shoga, Barry Rountree, Martin Schulz, and Jeff Shafer. 2014. Whitelisting MSRs with msr-safe. In *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14.*

[27] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. 2009. Koala: A Platform for OS-level Power Management. In *Proc. of the 2009 EuroSys Conference.*

[28] Ananta Tiwari, Michael Laurenzano, Joshua Peraza, Laura Carrington, and Allan Snavely. 2012. Green Queue: Customized Large-scale Clock Frequency Scaling. In *CGC '12: Proc. of the 2nd Intl. Conference on Cloud and Green Computing.*

[29] Wei Wang, Allan Porterfield, John Cavazos, and Sridutt Bhalachandra. 2015. Using Per-Loop CPU Clock Modulation for Energy Efficiency in OpenMP Applications. In *Proceedings of the 2015 44th International Conference on Parallel Processing (ICPP).* IEEE Computer Society, 629–638.