Memory Throttling on BG/Q: A Case Study with Explicit Hydrodynamics

Bo Li
Virginia Tech
bxl4074@vt.edu

Edgar A. León
Lawrence Livermore National Laboratory
leon@llnl.gov

Abstract

Power and energy efficiency are major concerns in future supercomputing systems. We expect that applications will be constrained to operate under a power budget and achieving the expected levels of performance will be challenging. Understanding how power is consumed by an application throughout its different phases will be necessary to shift power to those resources on the critical path. In this paper, we identify opportunities for shifting power between components for a representative kernel of explicit hydrodynamics codes. Based on a linear regression model, we dynamically throttle the memory system in regions with low memory bandwidth requirements on an energy-efficient supercomputer. Our results show that we can save a significant amount of power that could be used on resources on the critical path and, thus, maximize performance under the operating power budget.

1 Introduction

Power and energy efficiency are major concerns in future supercomputing systems. We expect that high-performance computing (HPC) applications will be constrained to operate under a power budget and, thus, achieving the expected levels of application performance will be challenging. Furthermore, because of the complexity of these applications, the computational requirements within a given code may vary on a per-physics and per-phase basis dynamically and across different problem sizes and input sets. Given this environment, we need a better understanding of the computational requirements of applications so that we can distribute and reallocate power selectively to those system components that an application really needs.

In this work, we identify and analyze opportunities for distributing power among system components on a supercomputer system to improve the performance of a representative kernel of explicit hydrodynamics codes at the U.S. Department of Energy (LULESH). Using an energy efficient supercomputer architecture, IBM Blue Gene/Q (BG/Q), we throttle the memory system on a per-region basis according to a linear regression model based on performance counters. We envision that in future machines, advanced runtime systems will shift power from the memory system to the CPU, for example, in regions with high locality and low demand for memory bandwidth. Although BG/Q does not allow redirecting power from one subsystem to another, our objective is to identify opportunities where this may be possible. We demonstrate that even with a simple linear regression model we can save up to 20% of dynamic power with a marginal effect on performance. This power could be used by other system components on the critical path and, thus, maximize performance under the given power budget.

2 Related work

Adjusting operating voltage and frequency is a well-known approach to improve power and energy efficiency. Curtis-Maury et al. [3] optimized power consumption via CPU DVFS and dynamic concurrency throttling using a performance prediction model based on hardware counters. Lim et al. [9] also used CPU DVFS in a similar manner. Miftakhutdinov et al. [11] extended this approach by considering a realistic memory system with a streaming prefetcher. Deng et al. [5] proposed a scheme to reduce energy consumption by applying DVFS to the memory controller and DFS to memory channels. David et al. [4] also studied memory DVFS and its impact on power consumption. These two memory studies relied on simulation or replying an analytical model because of the lack of an interface to dynamically change memory speeds. Chang et al. [2] investigated the impact on both performance and energy consumption from memory bandwidth scaling, but not from DVFS directly. Malladi et al. [10] used mobile DRAM memory in data centers to improve energy efficiency. In our work, we adjust the speed of memory dynamically based on an application's needs. To the best of our knowledge, this is the first study that employs memory throttling on a supercomputer system for explicit hydrodynamics codes.

Code phase based analysis is becoming useful for optimizing performance. Servat et al. [12] presented a mechanism to map source code to performance. Similarly, our work characterizes program behavior from hardware counters; we use this information to predict the minimal memory frequency per code region. However, our application regions are driven by domain scientist knowledge. In general, the behavior of complex physics packages is difficult to predict and, thus, hints from domain scientists are necessary.

Our work is closely related to Bertran's [1] and Felter's [6]. Bertran et al. analyzed the potential power and performance benefits of implementing gating mechanisms in the network and memory links for a set of HPC benchmarks. The authors quantified memory and network idle times and estimated, analytically, the benefits of shifting power to the processor. Using simulation, Felter et al. used power shifting to improve system efficiency, reduced power or increased performance, for a set of SPEC benchmarks. In our work, we employ real memory throttling on a supercomputer system to identify power shifting opportunities for HPC hydrodynamics codes. Our throttling policy is based on a linear regression model of performance and applied on a per application's phase basis.

3 LULESH

The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) mini-app [7] provides a simplified source code that contains the data access patterns and computational characteristics of larger hydrodynamics codes. It uses an unstructured hexahedral mesh with two centerings and solves the Sedov problem. We chose it for this study because explicit hydrodynamics can consume up to one third of the compute cycles at the U.S. Department of Defense data centers.

From an application developer's perspective, LULESH can be divided into code phases or regions according to the physics performed [7]. This code breakdown allows us to understand changes in performance, power, and energy from one region to the next. On a multi-physics code, in addition to regions, we would be interested in changes between physics packages.

In this work, we study five regions that account for over 90% of the execution time of LULESH and present a diverse set of properties: *Region 1* performs the stress calculation routines; *Region 2* performs the hourglass calculation; *Region 3* consists of two memory-bound loops that update the velocity and position of nodes; *Region 4* includes `CalcKinematicsForElems` and `CalcMonotonicQForElems`, which gather values from nodes to element centers followed by compute-intense calculations; and *Region 5* includes `MonotonicQforRegions` and `MaterialApply`, which have a significant amount of control flow instructions and instructions with dependencies.

4 Memory throttling on BG/Q

In this section, we introduce BG/Q's memory throttling and its impact on system performance. A BG/Q compute node consists of a chip with 16 A2 user cores and 16 GB of SDRAM-DDR3 memory. Each core has four SMT threads and runs at 1.6 GHz. Each chip has a 32-byte, 1.33 GHz interface to memory for a peak read+write bandwidth of 42 GB/s.

BG/Q provides a built-in hardware feature on the DDR controller that introduces higher delays or *idle cycles* between each read and write to DDR. A user application can adjust this number, to slowdown the memory bus (and save power), as a parameter to the function `Kernel_SetPowerConsumptionParam`. This value ranges between 0 and 126 and can be specified at a node granularity. Each idle cycle is 1 DDR cycle at 1.33 GHz.

4.1 Impact on bandwidth and runtime

We ran the STREAM benchmark (copy, scale, add, and triad) with 16 threads on a BG/Q node to quantify the impact of memory throttling on bandwidth. The maximum bandwidth achieved under different memory speeds is shown in Figure 1. In the *baseline* case, we do not throttle the memory system (memory at full speed). As expected, as we add more idle cycles to the memory controller, memory bandwidth decreases.

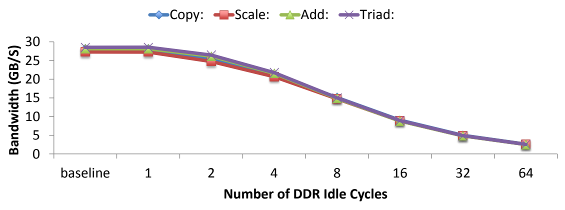

Figure 1: Effect of throttling on memory bandwidth.

We also measured the impact of throttling on the execution time of LULESH. Although we run experiments for all five regions, Figure 2 focuses on regions 3 and

4, which exemplifies the different computational characteristics across regions. As this figure shows, region 3 is more sensitive to memory bandwidth than region 4. As explained in Section 3, region 3 consists of two memory-bound loops that update the velocity and position of nodes. Region 3 shows degradation in performance between 8 and 16 idle cycles, while region 4 between 32 and 64. We will show later that we can use a simple model, based on performance counters, to predict the highest number of idle cycles to add, on a per-region basis, with a marginal effect on performance. The saved power could be shifted to other subsystems to improve performance.

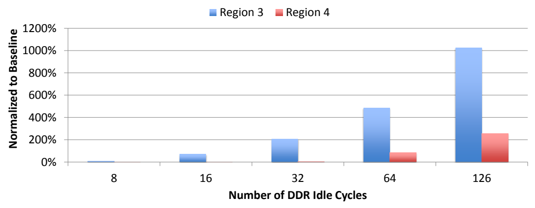

Figure 2: Effect of throttling on LULESH's execution time for input size 90 (90^3 elements) and 48 threads.

4.2 Memory throttling latency

We now quantify how long it takes for an application to observe changes in memory speed as a result of executing `Kernel_SetPowerConsumptionParam`. This is important to understand when memory throttling can and cannot be used. If the latency is too long, throttling may not be applicable to short-lived regions.

We used STREAM again to exercise the memory system and allocated a separate tracer thread to measure memory bandwidth at regular intervals of 1, 10, 100, and 1000 μs. Using the tracer thread, we throttled the memory system and quantified the number of intervals it took to observe a difference in bandwidth. We found that an effective change in bandwidth occurs between 1 to 10 μs.

5 Measuring power

BG/Q systems are capable of measuring power and energy at a node-board granularity. Each board includes 32 compute nodes and 2 direct current amperage (DCA) modules. Each DCA has a microprocessor unit that measures current and voltage of at most seven domains. The compute nodes can read power/energy data via the EMON2 (Environmental Monitoring version 2) API. Through this API, a user application is able to retrieve cumulative energy consumption at a given point in time. With two snapshots, the EMON2 library computes the energy difference and the average power consumption for the given interval. In all of our experiments, we capture power and energy consumption every 10 ms of all seven domains. We focus on the processor and memory since the other domains consume mostly the same amount of power regardless of utilization (e.g., network links are always on). More information on BG/Q's high-resolution power infrastructure and the proportions of static and dynamic power can be found elsewhere [1, 8].

6 Predicting optimal memory speed

On a given application, some code regions may be memory intensive while some others computation intensive. In those regions where computation is high and memory utilization is low, we can throttle the memory system and save power. If we can find the minimal memory speed for a given region without decreasing performance, we could reduce power and energy consumption.

The five code regions of LULESH have different computation and memory characteristics. So is the case when changing problem size[1] and level of concurrency (number of threads). In Figure 3, we show the lowest amount of throttling that can be applied to LULESH while maintaing at least 95% of its performance compared to the baseline (full memory speed). The top graph shows that the minimal memory throttling is region and concurrency dependent, while the lower graph shows that the minimal throttling for region 4 is also problem size dependent.

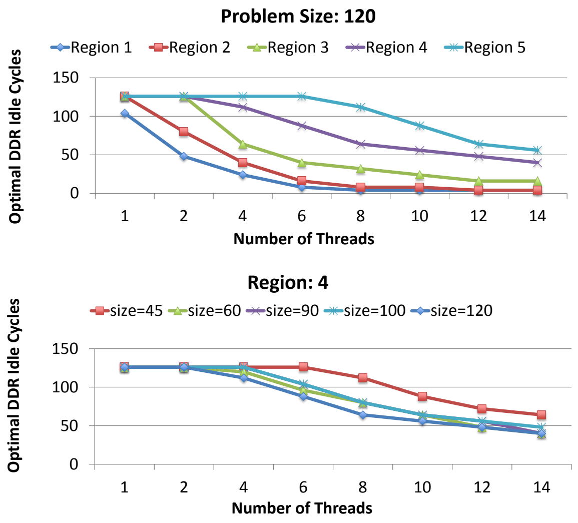

Figure 3: Optimal memory throttling as a function of concurrency for the LULESH regions and several problem sizes.

As shown in Figure 3, small problem sizes and low

[1]Problem size = X refers to an input of X^3 elements in LULESH.

levels of concurrency do not require high memory speeds (low number of idle cycles). Regions 1, 2, and 3 are more sensitive to memory bandwidth, while region 5 is the least sensitive as it can absorb a high number of idle cycles without affecting performance.

Thus, choosing the minimal memory speed is a function of several parameters including code region, problem size, and concurrency. To predict this *optimal* memory speed dynamically, we build a model based on performance counters that captures the compute and memory requirements of LULESH.

6.1 Linear regression model

There are several hardware counters that can capture the computation and memory behavior of an application. The counters we chose include the number of CPU cycles, number of floating-point instructions, number of load and store commands that missed the L1 data cache, number of executed instructions, L2 cache misses, misses in the prefetch buffer, and number of loads and stores to main memory.

These counters are classified as either CPU or memory related. The changes in problem size and level of concurrency can be observed through these counters. For example, increasing number of threads can cause an increase in memory bandwidth. We use these counters to predict the optimal (minimal) memory speed f_{min} for each region within an application. The model is defined as:

$$f_{min} = \sum_{i=1}^{N} w_i * \frac{c_i}{cyc} \tag{1}$$

where c_i is a hardware counter value, w_i is the coefficient achieved by offline training, and *cyc* is the total number of machine cycles executed. Since we consider multiple counters in this model, we use multivariate regression analysis to build the prediction model. Note that f_{min} corresponds to the number of idle cycles needed to throttle the memory.

To train the model (offline), we ran LULESH over a number of problem sizes, threads, and regions collecting more than 400 samples. We focused on problem sizes of 45 and higher, because smaller problem sizes result in regions (especially region 3) that are too small to quantify their power draw.

Figure 4 shows the observed optimal memory speed versus the one predicted from our model. Our model's accuracy is reasonable with an R^2 value of 0.67, although for some data samples the error is significant because the relationship between LULESH's execution time (characterized by the chosen performance counters) and memory speed is not necessarily linear. However, most predicted data points are close to the observed values.

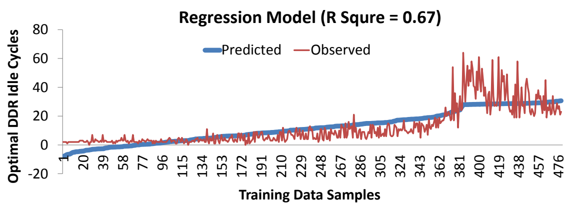

Figure 4: Accuracy of our linear regression model.

7 Impact on performance and power

In this section, we quantify the impact of memory throttling using our model on both performance and power. We show that the power consumption of LULESH can be reduced significantly with a marginal effect on performance. Since the granularity of our power and energy measurements is a node-board, we ran LULESH on 32 nodes: 1 process per node and 4 threads per core. We dedicated one core per node for a tracer thread that measures power. In addition, we only report the *dynamic* portion of power since we cannot change the amount of standby power. We expect that standby power will use a smaller fraction of the overall power in future memory technologies (e.g., PCM).

7.1 Effect on performance

When we trained the regression model, we used a threshold of 0.01. If our model is perfect, we should see no more than 1% performance loss compared to the *baseline* running at full memory speed. Our experiments consist of running LULESH with our regression model, which determines the number of idle cycles to inject on a per-region basis. In the first three iterations of each region (LULESH is an iterative code), we collect hardware counter data that is fed to the model to predict the *optimal* number of idle cycles. We use this number to set the memory speed for the remaining iterations. In Figure 5, we show the performance overhead of our approach compared to the execution time of the baseline configuration.

The problem sizes range from 70 to 140 and the number of threads from 44 to 60. In general, most of our predictions with higher number of threads are within 2% error. The performance loss is close to our desired threshold value of 1%. Higher inaccuracies occur with the smaller number of threads, i.e. 44 threads, because the observed data points are not linearly distributed (see Figure 4). In future work we plan on using other models that can capture non-linear behavior. However, even this simple regression model provides a small amount of performance degradation for most cases.

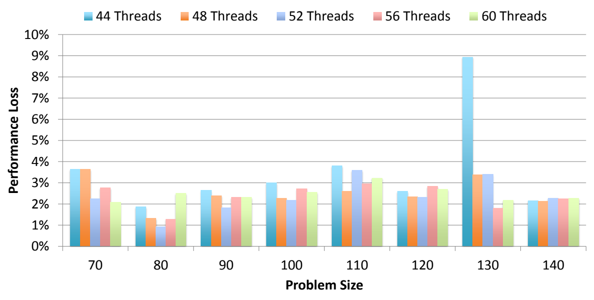

Figure 5: Runtime performance degradation by throttling memory according to our model. Each data point includes the aggregated overhead across all five regions.

7.2 Power and energy analysis

Our goal is to reduce as much power consumption as possible by lowering the memory speed until the performance loss reaches the given threshold. In this section, we first show that memory speed can significantly affect the power consumption of each LULESH region. Then, we demonstrate that both memory power and the total node-board power can be reduced based on our model predictions under various LULESH configurations. Finally, we discuss the tradeoffs between performance and power.

Figure 6 shows the power consumption of regions 1 and 4 under different memory speeds. We observe that total board power, CPU power, and memory power all drop significantly as we decrease memory speed. The difference in power may be as large as 300 Watts for the total board power. Although there is great potential to reduce power, we must consider performance (or energy) as well.

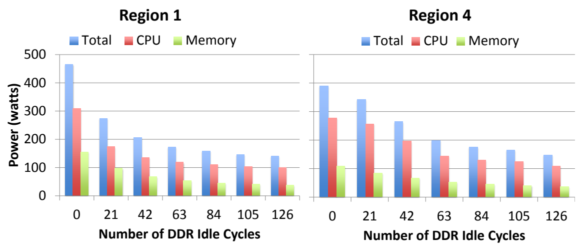

Figure 6: Memory throttling's effect on power for LULESH.

We now show what happens to the power consumption, along with performance, using our model-based prediction to throttle the memory of each region. Figure 7 shows how the performance, memory power, and total board power change with different number of threads based on our model's memory speed prediction for a problem size of 130. This figure shows that on the 44 threads configuration, our model prediction causes a significant decrease in performance (less than 10%) with almost no power savings. The total energy consumption in this case is worse than the baseline. For the other four thread configurations, our model prediction leads to small performance loss (within 3%) and large memory and total power savings. The total energy consumption is significantly lower for these cases.

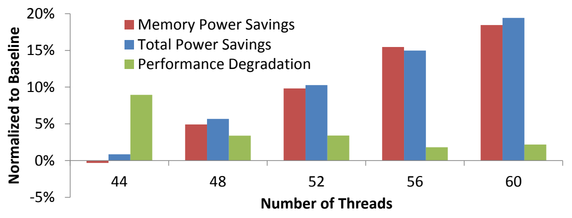

Figure 7: Performance and dynamic power tradeoffs with memory throttling using our regression model.

8 Conclusions and future work

In this work, we investigate opportunities for shifting power between system components on a supercomputer architecture for explicit hydrodynamics codes. Because of the different computational characteristics within a single application, we employ memory throttling on a per region basis to save power on code regions with low memory bandwidth requirements. To find the optimal (lowest) memory speed for a given region without affecting performance, we use a linear regression model based on performance counters. Our results show that memory throttling can save up to 20% of dynamic power with an average performance loss of 3%. This indicates that power shifting on explicit hydrodynamics present significant opportunities to improve performance within a fixed power budget. Future work includes quantifying power shifting opportunities on a range of HPC applications via memory throttling and employing machine learning techniques to capture non-linear behavior such as artificial neural networks.

9 Acknowledgments

We would like to thank Ian Karlin from Lawrence Livermore National Laboratory for lending his time and expertise with LULESH to this project. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-657811.

References

[1] Bertran, R., Sugawara, Y., Jacobson, H. M., Buyuktosunoglu, A., and Bose, P. Application-level power and performance characterization and optimization on IBM Blue Gene/Q systems. *IBM Journal of Research and Development 57*, 1/2 (Jan-Mar 2013).

[2] Chang, D. W., Son, Y. H., Ahn, J. H., Kim, H., Ahn, M., Schulte, M. J., and Kim, N. S. Dynamic bandwidth scaling for embedded DSPs with 3D-stacked DRAM and wide I/Os. In *International Conference on Computer-Aided Design* (San Jose, CA, Nov. 2013), ICCAD'13, IEEE/ACM.

[3] Curtis-Maury, M., Shah, A., Blagojevic, F., Nikolopoulos, D. S., de Supinski, B. R., and Schulz, M. Prediction models for multi-dimensional power-performance optimization on many cores. In *International Conference on Parallel Architectures and Compilation Techniques* (Toronto, Canada, Oct. 2008), PACT'08, ACM.

[4] David, H., Fallin, C., Gorbatov, E., Hanebutte, U. R., and Mutlu, O. Memory power management via dynamic voltage/frequency scaling. In *International Conference on Autonomic Computing* (Karlsruhe, Germany, June 2011), ICAC'11, ACM.

[5] Deng, Q., Meisner, D., Ramos, L., Wenisch, T. F., and Bianchini, R. MemScale: Active low-power modes for main memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, CA, Mar. 2011), ASPLOS XVI, ACM.

[6] Felter, W., Rajamani, K., Keller, T., and Rusu, C. A performance-conserving approach for reducing peak power consumption in server systems. In *International Conference on Supercomputing* (Cambridge, MA, June 2005), ICS'05, ACM.

[7] Karlin, I., McGraw, J., Keasler, J., and Still, B. Tuning the LULESH mini-app for current and future hardware. In *Nuclear Explosive Code Development Conference* (Livermore, CA, Oct. 2012), NECDC'12.

[8] León, E. A., and Karlin, I. Characterizing the impact of program optimizations on power and energy for explicit hydrodynamics. In *International Parallel & Distributed Processing Symposium; Workshop on High-Performance, Power-Aware Computing* (Phoenix, AZ, May 2014), HPPAC'14, IEEE.

[9] Lim, M. Y., Porterfield, A., and Fowler, R. SoftPower: Fine-grain power estimations using performance counters. In *International Symposium on High Performance Distributed Computing* (Chicago, IL, June 2010), HPDC'10, ACM.

[10] Malladi, K. T., Nothaft, F. A., Periyathambi, K., Lee, B. C., Kozyrakis, C., and Horowitz, M. Towards energy-proportional datacenter memory with mobile DRAM. In *International Symposium on Computer Architecture* (Portland, OR, June 2012), ISCA'12, ACM/IEEE.

[11] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N. Predicting performance impact of DVFS for realistic memory systems. In *International Symposium on Microarchitecture* (Vancouver, BC, Canada, Dec. 2012), MICRO-45, IEEE/ACM.

[12] Servat, H., Llort, G., Gonzalez, J., Gimenez, J., and Labarta, J. Identifying code phases using piece-wise linear regressions. In *International Parallel & Distributed Processing Symposium* (Phoenix, AZ, May 2014), IPDPS'14, IEEE.