2014 Energy Efficient Supercomputing Workshop

On the Feasibility of Dynamic Power Steering

Kevin J. Barker and Darren J. Kerbyson
Performance and Architecture Lab
Pacific Northwest National Laboratory
Richland, WA USA
{Kevin.Barker,Darren.Kerbyson}@pnnl.gov

Eric Anger
College of Computing
Georgia Institute of Technology
Atlanta, GA USA
eanger@gatech.edu

***Abstract*— While high performance has always been the primary constraint behind large-scale system design, future systems will be built with increasing energy efficiency in mind. Mechanisms such as fine-grained power scaling and gating will provide tools to system-software and application developers to ensure the most efficient use of tightly constrained power budgets. Such approaches to-date have been focused on node-level optimizations to impact overall system energy efficiency. In this work we introduce *Dynamic Power Steering*, in which power can be dynamically routed across a system to resources where it will be of most benefit and away from other resources to maintain a near-constant overall power budget. This, a higher-level algorithmic approach to improving energy efficiency, considers the whole extent of a system being used by an application. It can be used for applications in which there is load-imbalance that varies over its execution. Using two classes of applications, namely those that contain a wavefront type processing, and a particle-in-cell, we quantify the benefit of Dynamic Power Steering for a variety of workload characteristics and derive some insight into the ways in which workload behavior affect Power Steering applicability.**

Keywords—Energy-efficient Computing, Large-scale Systems, Dynamic Power Steering, Dynamic Workloads

I. INTRODUCTION

The High Performance Computing (HPC) landscape is evolving rapidly on the way towards Exascale computing. Whereas ultimate performance was previously the sole metric for computing platform success, future systems will be required to achieve unprecedented levels of performance within tightly constrained power budgets. For instance, the current drive to Exascale systems has a power budget goal of 20MW [1]. This new emphasis on energy efficiency within the context of high performance will necessitate new approaches to optimizing power at all levels from the underlying technology, up through the software-stack, to and including the applications.

Sea changes are imminent in both system hardware and software architecture as well as application design in response to the pressures imposed by this new emphasis on energy efficiency. On the system side, restrictive power budgets imply that it may be the case that not all architectural components may be utilized at their full capabilities simultaneously. In turn, fine-grained power allocation and measurement capabilities will allow system software and applications to closely monitor power consumption across the system and adapt the power distribution to characteristics of the executing workload. In other words, the parallel system may be used in a "throttled-down" configuration, or, alternatively, an asymmetric power distribution may be employed across the machine.

On the application side, it is expected that applications will become increasingly more adaptive and asynchronous, varying over time and across the parallel system in ways that are input dependent. The evolving nature of the ongoing simulation will lead to dynamic and input-dependent load imbalance, making static performance prediction and power allocation impossible and thus necessitating the need for dynamic mechanisms to respond to the evolving workload.

In response to these needs we introduce here *Dynamic Power Steering* whose goal is to maximize performance within a fixed power budget and thereby optimizing energy consumption. Dynamic Power Steering routes power to resources (processor-cores) that are assigned the most work and thus lie along the performance-critical path of the application. Such situations arise in most applications that exhibit load-imbalance and that often require complex load balancing that approximately equalizes work across the system resources but at the cost of increasingly complex load-balancing operation and with extensive data movement. In order to keep the overall system within the prescribed power budget, power is diverted away from other resources in such a way as to not negatively impact overall performance.

Of particular interest is the following open question: what application characteristics will enable Dynamic Power Steering approaches to be most successful? This work explores this question through the use of two representative workloads, namely those that contain a wavefront type processing, and a particle-in-cell that is used to simulate a charged field between two electrical plates. A third is used as a control case containing work that is a randomly distributed across the system. These represent many applications and are implemented in the form of a synthetic workload in which sequential and parallel workload characteristics can be explicitly controlled. The use of a synthetic workload addresses the "chicken and egg" problem that this feasibility study faces: we are exploring future system and application architectures, neither of which currently exists. Dynamic Power Steering is a critical enabling technology for Exascale systems and must be developed concurrently with future applications and architectures.

The main contributions of this work are as follows:

- We introduce Dynamic Power steering that is a system-wide approach to optimize performance within a fixed power budget and thereby optimizes energy consumption
- We explore Dynamic Power steering for a range of application workload characteristics on a medium sized, 1K-processor-core system, that allows for the emulation of a constrained power-budget
- We demonstrate that Dynamic Power Steering can improve performance and hence improve energy efficiency by up to 40%, by keeping the power budget near-constant for a wide-range of workload characteristics

Our initial feasibility study of Dynamic Power Steering clearly illustrates that such an approach can be used to optimize power use across a system for load-imbalanced applications when there is a power constraint to the system operation. It utilizes a current system that is power constraint, but instead mimics a power-constraint by setting the default power-state (p-state) to be at a mid-point, and allowing the p-state to vary up or down under control of Dynamic Power Steering depending on the availability of work. The approach is general and can be applied to future power-constrained systems.

The rest of this paper is organized as follows. Section 2 details related work in the field of large-scale energy efficiency and dynamic energy optimization. Section 3 details the principles and approach of Dynamic Power Steering. Section 4 describes the three test-case workloads. The experimental setup including the software infrastructure and the parallel system utilized for our experiments is detailed in Section 5. Section 6 provides the results of our exploration and a discussion. Conclusions drawn from this work are given in Section 7.

II. Related Work

Much of our work has been driven by the observation that future systems will be power constrained, that is not all resources will be active at any one time as insufficient power will be available. This has been noted at the processor socket level, and the term dark-silicon has been analyzed by several including work that points to how such power constraints may impact future architecture design [2]. Power constraints have also been considered in the context of job scheduling to make best use of an available power budget [3]. However, current systems do not have power constraints and so most current research is focused on improving energy efficiency using both application independent and application driven approaches.

Current work on improving energy efficiency includes the Adagio runtime system for slack prediction and provides energy optimization for applications including UMT2K and Paradis [4]. However, the approach assumes Bulk Synchronous Parallelism (BSP) in which the model of execution consists of multiple steps each containing a compute followed by communication that is often global. This does not provide the best energy efficiency for applications with complex and time-dependent processing patterns including those in wavefront applications that have benefited from the Energy Template approach [5]. Approaches for the static analysis of power use by dividing applications into phases have also been used [6]. A framework that exploits barriers for energy efficiency has also been explored [7]. Green building blocks and methodologies for hybrid execution for providing energy efficiency in the runtime system was proposed in [8] and [9]. Several techniques that use Dynamic Voltage Scaling, including a just-in-time method [10], enable a processor-core to slow down if the assigned computation is lower than on others. Energy efficiency for one-sided communication runtime systems has also been proposed [11] that considered the use of DVFS and interrupt-driven execution. Methods for designing energy efficient collective communication primitives using MPI have been studied for a range of applications [12]. Recent work includes designing energy efficient runtimes for hybrid programming models [13], and graph algorithms [14]. However, these approaches are either reliant on power reduction [15] or more focused towards reduction in runtime, which will have an automatic effect on energy consumption.

Our work is rather different from that on improving energy efficiency alone. It provides a higher-level view of power-optimization across a system that has an overall power constraint. It trades the principle of load balancing for power balancing whilst also aimed at satisfying an overall system power budget.

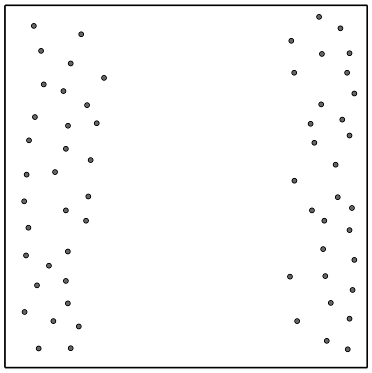
III. Dynamic Power Steering Approach

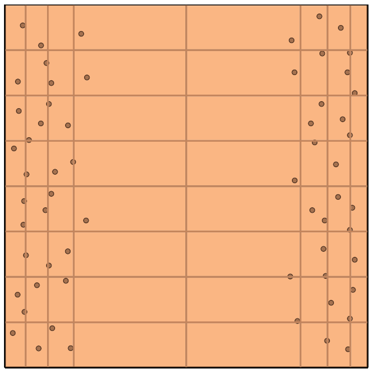
A. *Overview of Dynamic Power Steering*

Dynamic Power Steering addresses a key challenge envisioned with future extreme scale systems: restrictive power budgets will mean that not all system components may be fully utilized simultaneously. Combined with increased software complexities that will lead to adaptive and dynamic applications, whose behavior and resource requirements will evolve with simulation progression, will create the need for novel methodologies that will require the optimization of energy in order to extract maximum performance. Our Dynamic Power Steering approach addresses these challenges by routing power to components across a system where it provides the most benefit in response to changing demands imposed by the executing workload.

Traditional techniques to optimize execution time typically involve dynamic load balancing, in which computation tasks or data are migrated from over-loaded processors to under-loaded ones in an attempt to dynamically smooth out variations in workload across a parallel system. Thus processors will step through the execution of an application at approximately the same rate. This often requires significant movement of data from one memory domain to another, a cost which increases with system scale. In addition, carefully constructed data distributions may be altered to the detriment of performance. In successful load balancing techniques, the cost (in terms of time) of both evaluating a load-balancing decision-making algorithm as well as data movement is less than the idle time lost to load imbalance and thus provides an overall reduction in runtime. However, the complexity for determining optimum (or even reasonable) balanced distributions, as well as determining which data to migrate, is increasingly complex for irregular data such as that used in many adaptive applications.

Given the tight power constraints of future Exascale systems, there will be more resources available than can be actively powered at any one time. Thus there is a clear need for active power management to ensure that maximum power thresholds are not exceeded.


(a) Non-uniform distribution of particles across a domain


(b) Uniform load-balancing of particles across processors (denoted by grid lines marking sub-domains) with uniform power distribution

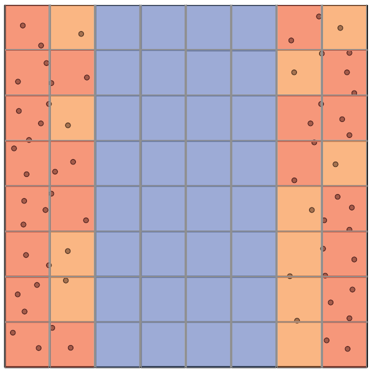

(c) power distribution for Dynamic Power Steering in which particles are left in-place across processors and power allocation is optimized

Figure 1: Example distribution of particles within a 2-D domain showing both load-balancing and Dynamic Power Steering (color indicates relative power allocation from low to high: blue to orange to red)

Our Dynamic Power Steering approach optimizes the power consumption in two primary ways:

1. Minimization of the power associated with data movement by eliminating the load-balancing of data between computation resources
2. Determining how power can be assigned to those resources that have more work to perform.

Dynamic Power Steering is most suited where the static calculation of an ideal power distribution is impossible such as to applications that are naturally load-imbalanced and whose load varies dynamically over time in an input-dependent manner. Further, applications whose performance is impacted by changes to the node or core p-state are most amenable to this approach; routing more power to over-loaded resources should have caused a significant improvement in performance.

For example, consider the example non-uniform distribution of particles across a 2-D domain in Figure 1. Using a classical load-balancing operation the assignment of particles to processors could result in the mapping as shown in Figure 1(b) with a uniform power distribution (as indicated by uniform color). However, if a constant domain decomposition of the space is assumed, as shown in Figure 1(c), Dynamic Power Steering can be used to assignment more power (shown in red) to processors with more work (or particles), and less (shown in blue) to processors with little or no work. With Dynamic Power Steering, there is no load balancing, but the power will be assigned to processors in proportion to the amount of work (particles).

Dynamic Power Steering results in a *power-optimized* system in which power is directed to the work being performed instead of to the data movement associated with load-balancing. This goes some way towards allowing applications to be tolerant of power constraints while still enabling the optimization of performance and thereby increasing energy efficiency.

B. Power Assignment to Processor-cores

Central to Dynamic Power Steering is the routing of power to processor-cores that are assigned more work. This can be achieved at a local-level by assigning each processor-core a suitable p-state that is in proportion to the amount of work whilst also satisfying the constraint of not exceeding a global-power budget. Determining the correct p-state to use for each processor-core depends on several elements. The first is the notion of the global power budget, the system-wide quantity of power capable of being assigned to the system at any one point. Only the available power in the global power budget may be allocated across the system. By lowering the p-state of under-utilized cores, the savings in power draw may be reallocated to other processor-cores with more assigned work which improves the performance of those processor-cores that are in the performance critical-path. The processor-core with the highest work will always be the one limiting the performance of the application, due to all other cores waiting idle at a global synchronization. If enough power is freed by other ranks to allow this rank to move to a higher p-state, overall execution time will improve.

The heuristic shown below is used in this work to select the p-state for all processor-cores in an iteration of an application. The maximum amount of work is calculated over all processors P_i (step 3), and its associated time cost using a performance model or empirical measurements (step 4). The *p-state* for all other processor-cores is calculated as being the slowest that does not impact the overall execution time (step 5). If the global power is exceeded then the *p-state* of the highest loaded processor is reduced and the assignment heuristic is repeated from step 3.

Power assignment heuristic used in this work

```
Start
1. $PWR_{max}$ = maximum globally available power
2. $p\text{-}state_{max}$ = fastest $p\text{-}state$
3. $N_{work\_max} = \max(N_{work\_i})\ \forall\ i \in \{ P_i \}$
4. $t_{work\_max} = N_{work\_max} \cdot t_{work}(p\text{-}state_{max})$
5. $\forall\ i \in \{P_i \mid P_i <> P_{work\_max}\}$ find the slowest p-state
   such that $t_{work\_i} < t_{work\_max}$
6. $PWR_i = t_{work\_i}(p\text{-}state_i)$
7. $PWR_{global} = \sum_{i=0}^{p} PWR(p\text{-}state_i)$
8. If $PWR_{global} > PWR_{max}$ then reduce $p\text{-}state_{max}$ and repeat 3.
9. Assign p-state calculated to each processor-core
End
```

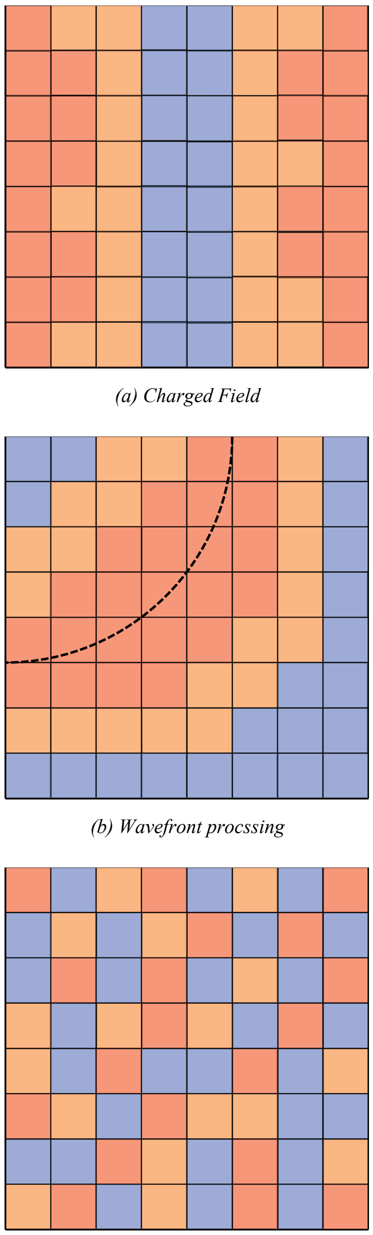
This algorithm optimizes the selection of p-states so that the most heavily loaded rank has the highest performance possible. Since applications are typically synchronized per iteration, the slowest rank is the limiting factor in the overall performance of the application. Most processor-cores will enter a polling state when they reach a global synchronization. This strategy reduces the latency of collective operations, but drastically increases processor utilization and by extension power. For this work, we also estimate how long each iteration will require and put any ranks that reach the barrier early to sleep, as below:

$$T_{Sleep} = T_{MaxWork} - T_{LocalWork} \qquad (1)$$

In addition, every processor-core enters the lowest-power p-state before sleeping. This ensures that all ranks are consuming as little power as possible while they wait to proceed. Lowering power consumption at collectives allows us to explore the maximum potential benefit of power steering.

IV. Workload Case Studies

We consider two types of workload that represent the underlying processing characteristics of many large-scale applications: the processing of a wavefront whose position varies over time, and a particle-in-cell processing for a charged field between two electrical plates. In addition a third random assignment of work across the processors in a parallel system is used as a control case. To explore the impact of Dynamic Power Steering, each of the workloads are kept as



(a) Charged Field

(b) Wavefront procssing

(c) Random distribution

Figure 2: Maps of processor load levels for the three workloads on an 8x8 processor configuration (color indicates relative processor load from blue to orange to red)

general as possible without reference to a particular application. Their processing characteristics are generated through a workload generator that allows for compute, load-imbalance as well as temporal aspects to be easily changed. We assume that the processing flow in each step of the workload consists of: concurrent computation performed by each processor-core, followed by a global synchronization. This flow corresponds directly to most large-scale applications. Each processor-core determines its own work assignment locally, which is then shared globally; *p-state* settings are determined based upon the local work in relation to the global work state to optimally assign power. *p-states* are then set according to each processor-cores assigned work, and the processing of the step begins.

Work is assigned based on the three workload distributions, is shown in Figure 2 for an 8x8 processor configuration; two represent patterns typically manifest in HPC workloads, while the third captures a control case. The first HPC pattern represents the density of charged particles in two-dimensional space placed between two charged plates. Due to the electric field caused by the charged plates, particles move over time toward the plates. Given that the global spatial domain is equally decomposed across the available, the initial load of randomly distributed particles is well balanced. However, as the simulation progresses and particles migrate towards the charged plates, the load becomes imbalanced. In Figure 2(a), this load imbalance is indicated using color; the red shows processor cores with the highest concentration of particles. These processor-cores are the ones that would most benefit from an increase in performance afforded by an additional power allocation.

Rather than perform the actual calculations of this workload, an analytic model is used. As time approaches infinity, points will follow a Gaussian distribution in distance from the closer of the two plates:

$$Position(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2)$$

Work assignment to each processor-core is determined by weighing a sample from each distribution that puts more weight on the even distribution earlier in time and on the Gaussian distribution later in time, as in Equation 3.

$$\begin{aligned} Work = Weight \times Position(x, \mu, \sigma) + \\ (1 - Weight) \times SteadyState \end{aligned} \quad (3)$$

The *Weight* value monotonically increases with each application iteration.

The second type of workload replicates the propagation of a wavefront through a two-dimensional space (Figure 2(b)). The wavefront begins in the upper-left corner of the two-dimensional global domain and expands with each subsequent iteration. As with the charge field, the space is partitioned along two dimensions with a single subdomain assigned to each available processor-core. The amount of work assigned to each core is inversely proportional to the distance of that core's subdomain from the wavefront according to the Gaussian distribution curve defined in Equation 2.

The final type of workload serves as a "control case" and assigns a random distribution of work across all processor-cores (Figure 2(c)). Note that this is not a random uniform distribution of particles, but is instead a random distribution of load across processor-cores, meaning that some are assigned proportionally higher or lower values of work in a random fashion.

The specific parameter space we explore is controlled by two parameters. The first is termed *computational intensity* and reflects the ratio of computation to memory access within the sequential computation. The computational intensity is uniform across all processor-cores. The computational intensity is able to reflect the characteristics of a multitude of applications; as the value is increased more work is performed in the core. Performance of a compute-bound application is more sensitive to changes in core *p-state* than an application that is memory bound. The second parameter is load-imbalance. The degree of load-imbalance across the system can be varied so as to again represent multiple applications. The lower the load-imbalance, the closer each processor-core's assigned work is to a set maximum per-core load value.

V. Experimental Setup

Since current systems are not power constrained we utilize an exiting system under an assumed power constraint. As we describe below, the power constraint is set to be the power use when setting all processor-cores' p-states to the mid-point of the available p-states. This allows for some cores to be allocated more power (when more heavily loaded), and at the same time for some cores to be allocated less power (when less loaded) with the same overall power budget.

For this work we utilized the PAL cluster located at Pacific Northwest National Laboratory (PNNL). This system contains power instrumented Power Distribution Unit (PDU) rails with which power consumption can be measured on a per-outlet basis at a frequency of 0.3 Hz. The cluster contains 144 nodes in total, each comprised of two sockets of AMD Opteron 6272 Interlagos processors with a maximum clock frequency of 2.1 GHz. Each socket consists of a dual-chip module with each chip containing four AMD Bulldozer dual-core modules (i.e., eight cores in total per chip). Because each dual-core Bulldozer module contains only a single floating-point unit, our experiments utilize a total of 16 of the available 32 cores per node. Each 8-module chip contains 8x64KB instruction caches, as well as 16x16KB L1 data cache, a 4x2MB L2 data cache, and a 2x8MB L3 data cache. Each node has 64GB DDR3 memory connected to the processors via a 3200MHz front-side bus. The Thermal Design Point (TDP) of each processor socket is 115W.

PAL is housed in three racks, with each rack containing 48 nodes physically organized as 12 quad-node units. Each quad-node contains two power supplies and is supplied by two outlets from the instrumented PDU rails. No direct power measurement of the InfiniBand switches was performed. Monitoring and storage of the power use of all nodes was enabled at the system's normal operating frequency of 0.3 Hz. PAL's installed O/S was Red Hat Enterprise Linux version 5.7 distribution containing the Linux kernel version 2.6.32. Pathscale version 4.0.10 compilers were used, along with the OpenMPI version 1.5.4 MPI libraries.

The characteristics of PAL are summarized in Table 1.

Table 1: Summary of the PAL cluster

Per-Socket Info	
Total Cores	16
Cores Used	8
Max. Frequency (GHz)	2.1
Power Gating Available	No
DVFS Available	Yes
Idling Cores Available	Yes
Per-Node Info	
Total Sockets	2
Memory (GB)	64
Nodes/Rack	48
Software Info	
Linux Kernel	2.6.32
Compiler	Pathscale v4.0.10
MPI Library	OpenMPI v1.5.4
Network	4xQDR InfiniBand
Power Info	
Socket TDP (W)	115
Rack Idle Power (KW)	10.4
Rack Active Power (KW)	14.2
Measurement Rate (Hz)	0.3

DVFS is available on PAL at available clock frequencies of 2.1, 1.9, 1.7, 1.5, and 1.4 GHz. The frequency domain is a single dual-core Bulldozer module. A "baseline" power budget is assumed corresponding to all processor-cores set at a 1.7 GHz. This baseline defines the power budget which may not be exceeded across and enables improved performance on some cores by increasing the clock frequency to 2.1 GHz at the expense of requiring other cores to be reduced to 1.4 GHz to compensate.

In order to analyze the impact on power consumption caused by changing the processor-core *p-state*, we measured the active power on a per-core basis under load. This was executed for each p-state and Figure 3 shows the power consumption for a single core. Initial core idleness can be seen in the first 30s, followed by the sharp increase in power consumption corresponding to the execution of the benchmark. On benchmark completion, the power drops to the idle level. It can be clearly seen that higher frequencies correspond to higher power draws, and also to shorter runtimes. However, the impact on runtime is dependent on the sensitivity of the benchmark to clock frequency.

Of particular note is that the power consumption groups into three bands consisting of the highest frequency (2.1 GHz), the middle three frequencies, and the lowest frequency (1.4 GHz). For this reason we consider only three *p-states*: 2.1, 1.7, and 1.4 GHz in this work. From this, we derive the per-core *p-state* power draw values given in Table 2.

Table 2: Per-core power utilization for each *p-state*

Frequency (GHz)	Core Active Power (W)
2.1	21.1
1.7	18.0
1.4	15.6

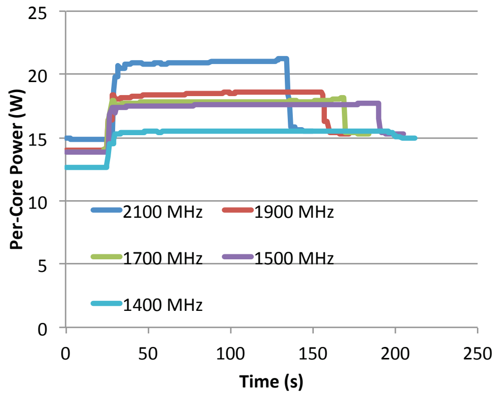


Figure 3: Measured power states for an AMD Interlagos core on the PAL system

VI. Results and Discussion

We conducted a set of experiments using PAL and the three workload case studies as detailed in Section IV. For all application configurations, 576 processor-cores were utilized across 36 nodes of the PAL cluster. Results are presented relative to executing the workloads without Dynamic Power Steering thus enabling its impact to be assessed. Positive values indicate improvement over the baseline and thus the advantage of using Dynamic Power Steering. Below the results of our experiments are presented and a discussion of the more general observations is provided.

Figure 4 indicates the improvement in runtime improvement when using Dynamic Power Steering for all three application configurations with varying degrees of load imbalance and compute intensity. For each graph, the *x-axis* denotes the level of computational intensity, on a scale from zero (entirely compute bound, executing in cache) to one (memory bound with very few arithmetic operations). The *y-axis* denotes the load-balance across processor-cores in the parallel system, again with zero indicating a high degree of load-imbalance and one indicating a perfect load-balance across all processor-cores.

In the case of the Charge Field synthetic workload, Figure 4(a), a clear trend emerges. First, when using Dynamic Power Steering and allowing each of the processor core's *p-state* rto change results in a greater change in performance when the workload exhibits a higher level of computational intensity. This is due to the fact that, on PAL, changing processor core *p-state* equates with altering clock frequency. Workloads that are bound by memory performance are sensitive to processor-core frequency, and thus less impacted by changes in *p-state*. Compute intensive workloads therefore benefit from increased clock frequency on those processor cores that are overloaded.

Second, the performance improvement increases with increased load-imbalance, although the effect is more dramatic than for increased computational intensity. One reason for this is the coarse granularity of the available *p-states* on the PAL system. The workload has to exhibit a fairly high level of imbalance before lowering a processor core's *p-state* does not place it onto the performance critical path, thereby increasing overall runtime.

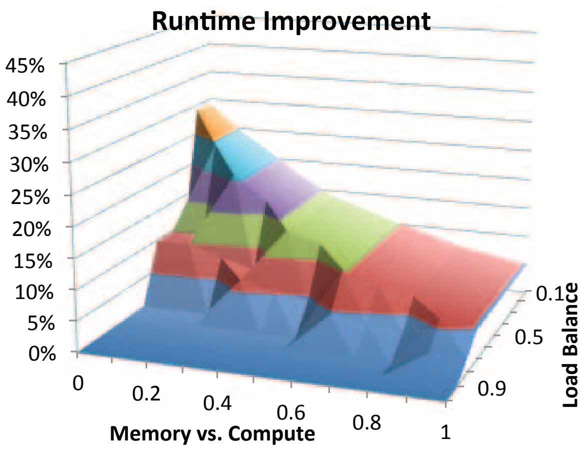
We note that the performance improvement for the Wavefront workload is non-zero even in cases of load balance. This is due to the load not being perfectly distributed due to work being distributed according to a Gaussian distribution for a given distance from the wavefront; however, the Gaussian distribution always assigns more work along the wavefront itself so that a degree of load imbalance always exists. Given a smooth load distribution, we feel that the performance improvement for a well-balanced load distribution would be negligible.

Although the Random load distribution distributes load levels across processor cores randomly, ranging from under-loaded to over-loaded load levels, there is still sufficient load imbalance across the system (in some configurations) to afford a performance improvement of over 25% (Figure 4(c)). An notable side-effect of the Dynamic Power Steering methodology is that, by eliminating the data movement typically associated with dynamic load balancing, data locality is preserved to the greatest extent possible. In the case of the Random load distribution example, this is especially relevant as data may be required to migrate a great distance away from its originating processor core to evenly distribute the load.

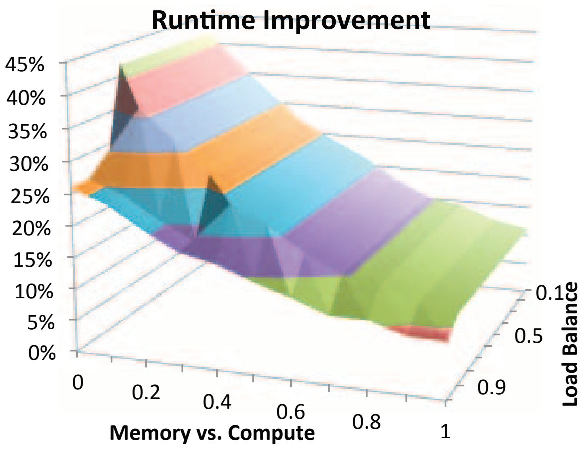
The Dynamic Power Steering approach provides improved performance for workloads executing on a parallel system under power budget constraints. It is important to ensure that power allocation is within a global limitation. Figure 5 denotes the change in power when using Dynamic Power Steering relative to default execution. Because of the coarse granularity of processor-core p-states and the fact that Dynamic Power Steering aims to keep overall power consumption below a specified threshold, it is often the case that an execution of all three workloads actually consumes less power than a default execution.

In some cases, however, the power budget is exceeded by less than 5%. The reason is that the models used to predict execution time and power consumption for workload execution under different processor *p-states* have a small margin of error. This results in an overall prediction within power constraints, but measurements resulting in slightly higher power consumption. As the focus of this research is in assessing the viability of the Dynamic Power Steering approach, we do not see this as a limitation in the methodology. Future work will be in developing accurate power modeling approaches that can be integrated with the Dynamic Power Steering methodology.

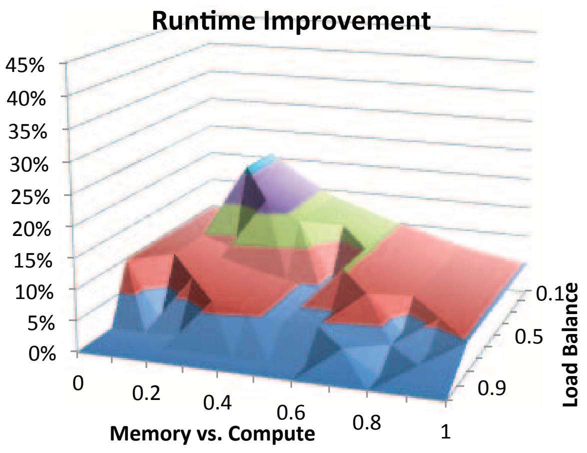
Improved performance combined with nearly equivalent power consumption will result in improved energy efficiency, and that is indeed what is shown in Figure 6. For all three workload types, energy efficiency is improved; in the case of the Wavefront workload, energy efficiency is improved by nearly 45%, for the charged field by 27%, and for the random load by up to 20%. The greatest improvement in general corresponds to the cases of greatest compute intensity and load-imbalance.



(a) Charge Field performance improvement

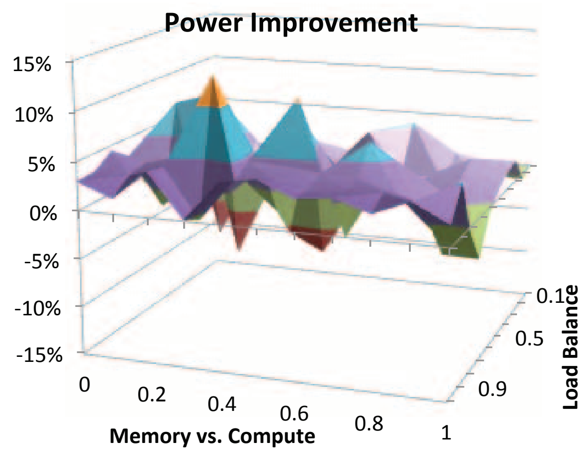


(b) Wavefront performance improvement

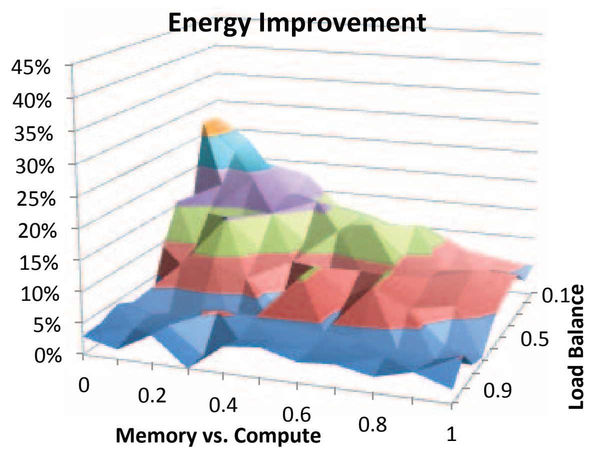


(c) Random Load performance improvement

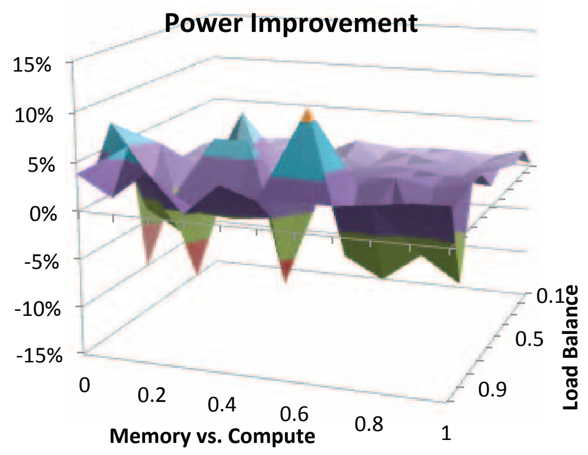
Figure 4: Relative runtime improvement for all three workload types when using Dynamic Power Steering



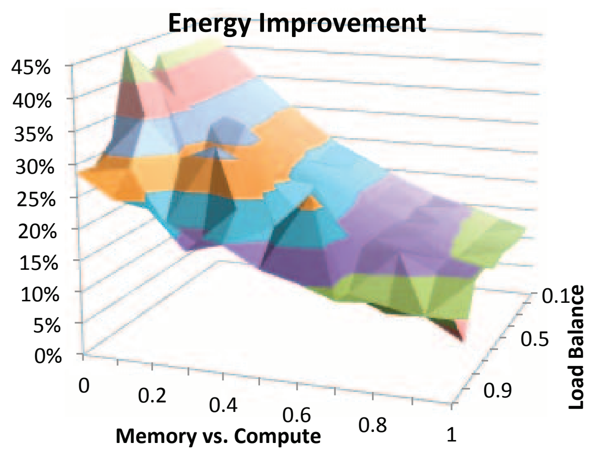
(a) Charge Field power improvement



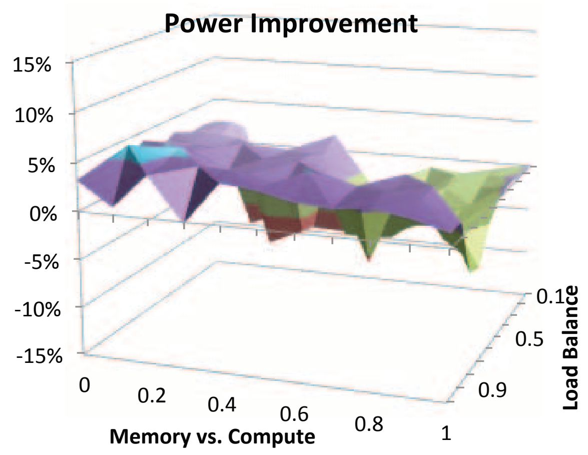
(a) Charge Field energy improvement



(b) Wavefront power improvement

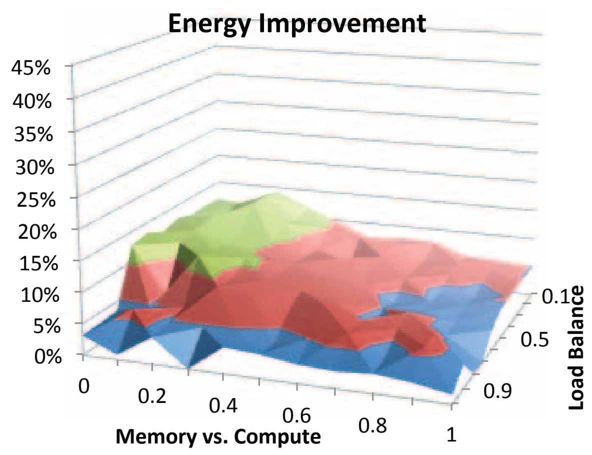


(b) Wavefront energy improvement



(c) Random Load power improvement

Figure 5: Relative power improvement for all three workloads when using Dynamic Power Steering



(c) Random Load energy improvement

Figure 6: Relative energy improvement for all three workloads when using Dynamic Power Steering

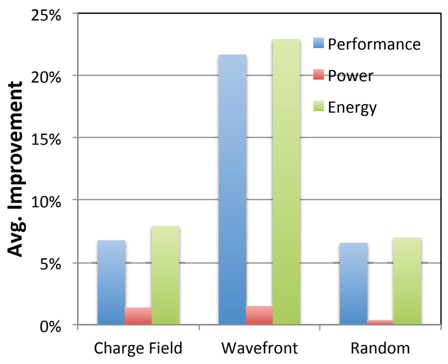


Figure 7: Average performance, power, and energy consumption results for all workloads

A summary of the average improvement in performance, in power consumption, and in energy efficiency for all three workloads is shown in Figure 7. This is taken over all computational intensity and load-balance cases shown in Figures 4-6. Clearly, performance is improved in all cases. Combined with slight improvements in power consumption, this results in slightly greater improvements in overall energy efficiency. The Wavefront workload exhibits greater improvements because as a slight degree of load imbalance persists in all cases allowing for improved performance as discussed earlier.

VII. Conclusions

In this work, we have explored the feasibility of Dynamic Power Steering, a method by which performance and energy efficiency of load-imbalanced applications may be optimized on power-constrained large-scale systems. In contrast with traditionally dynamic load balancing, data migration is eliminated, conserving the power expenditure associated with data movement. Instead, power is dynamically routed to processing resources with the greatest load allowing them to execute at a higher rate of performance. To compensate, power is routed away from under-loaded resources, thereby minimizing idle-time.

To assess its feasibility, we use three workloads that represent many large-scale applications and allow us to "dial in" the characteristics of a wide range of workloads. We simulate a power-constrained system by restricting the global power budget across a state-of-the-art cluster; by specifying the nominal *p-state* to be that of each processor core executing at 1.7 GHz, the middle *p-state* available. This allows each processor core room to improve performance by increasing the power allocating to the core, requiring other cores in the system to reduce their power consumption by utilizing a *p-state* of lower performance and power.

In this work, we have focused on two application characteristics that are the most significant: computational intensity and degree of load imbalance. Our results indicate that significant performance gains are possible in the cases in which the application is significantly load-imbalanced and compute intensive. Large degrees of load imbalance provide the most opportunity for power routing, as there are under-loaded processors able to "donate" power to over-loaded cores in a power constrained system. Compute intensity, in turn, determines the impact increases in *p-state* will have on core performance; processors that are bound by compute performance, as opposed to memory performance, are more likely to exhibit a change in performance when core frequency is altered. Across the three workloads, we demonstrate an average performance improvement of greater than 11%, resulting in an average improvement in energy efficiency of 13%.

In future systems with more available *p-states* and a finer degree of control in power routing, we expect the Dynamic Power Steering approach to offer greater improvements.

Acknowledgements

This research was supported in part by Advanced Scientific Computing Research (ASCR) office of the DOE Office of Science. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

References

[1] "Scientific Grand Challenges: Architectures and Technology for Extreme Scale Computing," Report from DOE workshop, San Diego, Dec 8-10, 2009. (http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Arch_tech_grand_challenges_report.pdf)

[2] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling", in Proc. 38th Int. Symp. on Computer Architecture (ISCA'11), pp. 365-376.

[3] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Parallel Job Scheduling for Power Constrained HPC Systems," Parallel Computing, 28(12):615-630, Dec. 2012.

[4] B. Rountree, D.K. Lowenthal, B.R. deSupinski, M. Schulz, V.W. Freeh and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in proc. Int. Conf. on Supercomputing (ICS), New York, 2009, pp. 460–469.

[5] D. J. Kerbyson, A. Vishnu, K. J. Barker, "Energy Templates: Exploiting Application Information to Save Energy", in proc. IEEE Cluster, Austin, Sept. 2011.

[6] V.W. Freeh and D.K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in proc. ACM Symp. on Principles and Practice of Parallel Programming (PPoPP), 2005, pp. 164–173.

[7] C. Liu, A. Sivasubramaniam, M. Kandemir and M.J. Irwin, "Exploiting Barriers to Optimize Power Consumption of CMPs," in proc. Int. Parallel and Distributed Processing Symp. (IPDPS), Denver, April 2005.

[8] D.S. Nikolopoulos, "Green Building Blocks - Software Stacks for Energy-Efficient Clusters and Data Centres," ERCIM News 79, 2009.

[9] D. Li, D.S. Nikolopoulos, K. Cameron, B.R. deSupinski and M. Schulz, "Hybrid MPI/OpenMP Power-Aware Computing," in proc. Int. Parallel and Distributed Processing Symp. (IPDPS), Atlanta, GA, April 2010.

[10] N. Kappiah, V.W. Freeh and D.K. Lowenthal, "Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," in proc. IEEE/ACM Supercomputing (SC'05), Seattle, WA, Nov. 2005.

[11] A. Vishnu., S. Song, A. Marquez, K.J. Barker, D.J. Kerbyson, K. Cameron, and P. Balaji, "Designin Energy Efficient Communication Runtime Systems for Data Centric Programming Models", in proc. IEEE Int. Conf. on Green Computing and Communications, Hangzhou, China, Dec. 2010.

[12] K. Kandalla, E.P. Mancini, S. Sur and D.K. Panda, "Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters," in proc. Int. Conf. on Parallel Processing (ICPP), Sept. 2010, pp. 218–227.

[13] D. Li, B. de Supinski, M. Schulz, D. Nikolopolous, K. Cameron, "Strategies for Energy Efficient Resource Management of Hybrid Programming Models", IEEE Transactions on Parallel and Distributed Systems 24(1):144-157, Jan. 2013.

[14] N. Satish, C. Kim, Chugani, P. Dubey, "Large-scale Energy-efficient Graph Traversal: A Path to Efficient Data-intensive Supercomputing", in proc. IEEE/ACM Supercomputing (SC'12), Salt Lake City, Nov. 2012.