ARCS: Adaptive Runtime Configuration Selection for Power-Constrained OpenMP Applications

Md Abdullah Shahneous Bari*, Nicholas Chaimov†, Abid M. Malik*,
Kevin A. Huck†, Barbara Chapman*, Allen D. Malony† and Osman Sarood‡
*Department of Computer Science, University of Houston, {mshahneousbari, ammalik3, bchapman}@uh.edu
†Department of Computer Science, University of Oregon, {nchaimov,khuck,malony}@cs.uoregon.edu
‡Yelp Inc, {osarood}@yelp.com

***Abstract*—Power is the most critical resource for the exascale high performance computing. In the future, system administrators might have to pay attention to the power consumption of the machine under different work loads. Hence, each application may have to run with an allocated power budget. Thus, achieving the best performance on future machines requires optimal performance subject to a power constraint. This additional performance requirement should not be the responsibility of HPC (High Performance Computing) application developers. Optimizing the performance for a given power budget should be the responsibility of high-performance system software stack. Modern machines allow power capping of CPU and memory to implement power budgeting strategy. Finding the best runtime environment for a node at a given power level is important to get the best performance.**

This paper presents ARCS (Adaptive Runtime Configuration Selection) framework that automatically selects the best runtime configuration for each OpenMP parallel region at a given power level. The framework uses OMPT (OpenMP Tools) API, APEX (Autonomic Performance Environment for eXascale), and Active Harmony frameworks to explore configuration search space and selects the best number of threads, scheduling policy, and chunk size for a given power level at run-time. We test ARCS using the NAS Parallel Benchmark, and proxy application LULESH with Intel Sandybridge, and IBM Power multi-core architectures. We show that for a given power level, efficient OpenMP runtime parameter selection can improve the execution time and energy consumption of an application up to 40% and 42% respectively.

I. Introduction

Power consumption has become a critical design factor for a large scale HPC system. If we are to build an exascale machine out of today's hardware, it would require a dedicated power plant [1]. The U.S. Department of Energy's goal of reaching exascale computing within 20 megawatts of power implies that power is going to be the biggest constraint for future systems. This constraint will filter down to job-level power constraints. The goal at the job-level will be to optimize performance subject to a prescribed power budget.

Recent advances in processor and memory hardware designs have made it possible for the user to control the power consumption of the CPU and memory through software, e.g., the power consumption of Intel Sandy Bridge family of processors can be user-controlled through the Running Average Power Limit (RAPL) interface [2]. This ability to constrain the maximum power consumption of the subsystems allows a user to run an application within a given power budget. However, capping a processor at a lower power level reduces its execution performance as it may decrease the frequency or clock gate the capped processor. Performance improvement per node level at various power caps is important to get the overall best performance on future machines subject to a power constraint.

OpenMP is the *de facto* programming model for intra-node parallelism. To get the best performance out of an individual power capped node, one needs to study the execution behavior of OpenMP in power constrained systems. The OpenMP programming model provides certain performance adjustment runtime parameters that can be used to control the OpenMP execution environment [3]. Different OpenMP parallel regions or regions[1] have different execution behavior. Therefore, a runtime execution environment that gives the best performance at a certain power level also differs for various OpenMP regions. If the OpenMP runtime parameters are not properly selected, one may see a severe performance degradation. Usually, application developers choose to use the default parameter settings provided by an OpenMP runtime library. As a result, one gets sub-optimal performance for most of the existing OpenMP applications.

HPC users are already overstretched with ensuring correctness and maintaining sufficient performance of HPC applications. Therefore, the task of enforcing the job level power constrained performance should be left to HPC system software. The system software is in a good position to dynamically configure applications for the best performance subject to a power constraint. In this paper, we present the ARCS (Adaptive Runtime Configuration Selection) framework that chooses the best OpenMP runtime configurations for parallel loops in an HPC application. We define OpenMP configurations as: (1) Number of Threads, (2) Scheduling Policy, and (3) Chunk Sizes. We test ARCS using the NAS Parallel Benchmark, and a proxy application LULESH. We show that for a given power level, efficient OpenMP runtime parameter selection can improve the execution time and energy consumption of an application up to 40% and 42% respectively.

The major contributions of this work are listed below:

- We present ARCS framework that selects the best OpenMP runtime configurations for OpenMP regions to

[1]We use *OpenMP parallel regions* and *regions* synonymously

optimize HPC applications under a power constraint.

- To the best of our knowledge, ARCS is the first fully automatic framework that chooses OpenMP runtime configurations with no involvement of the application programmer.
- ARCS chooses and adapts OpenMP runtime configurations dynamically based on OpenMP region and underlying architecture characteristics, resulting in efficient execution on a number of applications under a power constraint across different architectures.

II. Motivation

OpenMP programming model is an integral part of many important HPC legacy codes in the form of hybrid programming models (e.g., - MPI + OpenMP). Therefore, tuning an OpenMP code to get a better per node performance for a given power budget is an important research problem. In this section, we motivate a reader about the need of ARCS for power-constrained OpenMP applications. The need for ARCS like framework depends on the following questions:

- *Does the best configuration for a given OpenMP region remain same across different power levels and workloads?*
- *Does the performance gain due to the best configuration persist across all power caps?*

We took an OpenMP region from the BT benchmark application and ran it with different power levels or power caps[2] using different number of threads, scheduling policies, and chunk sizes (150 different configurations). The region belongs to the `x_solve` function, and has coarse grain parallelism, i.e., the outermost loop is parallelized with **`#pragma omp parallel for`** OpenMP directive.

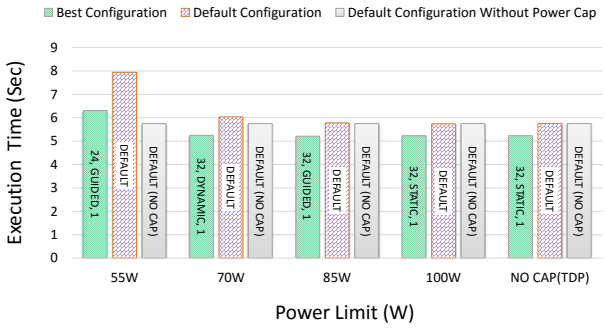


Fig. 1: Execution time comparison for the **x_solve** region of BT using different OpenMP runtime configurations at different power levels. Smaller value is better. The function was run on Intel Sandy Bridge.

Figure 1 shows the comparison of execution time using the optimal configuration[3] and the default configuration at different power levels. The default configuration uses maximum number of available threads, static scheduling, and chunk sizes calculated dynamically by dividing total number of loop iterations by number of threads. The figure clearly shows that the optimal configuration is different from the default configuration at all the power levels. It also shows that the optimal configuration improves the execution time of the region up to 20% compared to the default configuration at the same power level. Also, we can see that the optimal configuration at a lower power level gives better execution time performance than the default configuration with maximum power level prescribed by the manufacturer or Thermal Design Power (TDP). For example, the optimal configuration at 70W power cap improves execution time by almost 9% as compared to the default configuration at TDP (115W in our case).

We also experimented with OpenMP regions from other NAS Parallel benchmark applications using different runtime configurations. We observed that a significant number of the OpenMP regions showed similar behavior. We observed these OpenMP regions to have poor load balancing and cache behavior with the default configuration. We also saw that these poor behaviors persist across different power levels and workloads for these kernels with the default configuration. As a result, irrespective of power level or workload size an optimal configuration always shows consistent performance improvement compared to the default configuration for these kernels. However, we observed that the optimal configurations for these kernels change across different power levels and workloads.

In the future HPC facility, the load of applications may change dynamically. If the facility is working under a power constraint, the resource manager may add/remove number of nodes and adjust their power level dynamically. To get the best per node performance at each power level, the runtime configurations need to be changed dynamically. Our ARCS framework can do this efficiently.

III. Framework

The ARCS runtime is composed of two key software components. The first component is a modified OpenMP runtime. The second component is the APEX instrumentation and adaptation library. APEX integrates the Active Harmony search engine, integrated as part of the APEX library. Figure 2 shows the integration of the components in the ARCS runtime.

A. OpenMP runtime with OMPT

A broad group of interested parties has been working on extending the OpenMP specification to include a formal performance and debugging tool interface [4]. In order to provide support for both instrumentation (event-based) and sampling based tools, OMPT includes both events and states. The OMPT draft specification is available as a Proposed Draft Technical Report at the OpenMP Forum website [5]. The key OMPT design objectives are to provide low overhead observation of OpenMP applications and the runtime in order to collect performance measurements, provide stack frame support for sampling tools and incur minimal overhead when not in use. OMPT specifies support for a large set of events and

[2]We use the two words synonymously in this paper.
[3]The configuration that gives the best execution time.

states, covering the OpenMP 4.0 standard. In addition, OMPT specifies additional insight into the OpenMP runtime in the form of data structures populated by the runtime itself. These data structures include the parallel region and task identifiers, wait identifiers and stack frame data. A reference OpenMP runtime implementation with OMPT support based on the open-source Intel runtime is available[4] and OMPT has been integrated into performance tools such as TAU [6] and APEX.

B. APEX

We have implemented a measurement and runtime adaptation library for asynchronous multitasking runtimes called *Autonomic Performance Environment for eXascale (APEX)* [7], [8]. The APEX environment supports both introspection and policy-driven adaptation for performance and power optimization objectives. APEX aims to enable autonomic behavior in software by providing the means for applications, runtimes, and operating systems to observe and control performance. Autonomic behavior requires both performance awareness (introspection), and performance control. APEX can provide introspection from timers, counters, node- or machine-wide resource utilization data, energy consumption, and system health, all accessed in real-time. The introspection results are analyzed in order to provide the feedback control mechanism.

The most distinguishing component in APEX is the *policy engine*. The policy engine provides controls to an application, library, runtime, and/or operating system using the aforementioned introspection measurements. Policies are rules that decide on outcomes based on the observed state captured by APEX. The rules are encoded as callback functions that are periodic or triggered by events. The policy rules access the APEX state in order to request profile values from any measurement collected by APEX. The rules can change runtime behavior by whatever means available, such as throttling threads, changing algorithms, changing task granularity, or triggering data movement.

APEX was originally designed for use with runtimes based on the ParalleX [9] programming model, such as HPX [10] or HPX-5 [11]. However, the APEX design has proven to be flexible enough to be broadly applied to other thread-concurrent runtimes such as OpenMP.

APEX integrates the auto-tuning and optimization search framework *Active Harmony* [12]. In APEX, Active Harmony is directly integrated into the library to receive APEX performance measurements and suggest new parametric options in order to converge on an optimal configuration. Active Harmony implements several search methods, including exhaustive search, *Parallel Rank Order* and *Nelder-Mead*. In this work, we used the exhaustive and *Nelder-Mead* search algorithms. In our experiments, the *ARCS-Offline* method uses an exhaustive search to find the best configuration during one execution, then executes again with that optimal configuration. Only the second execution with the optimal configuration is measured. The *ARCS-Online* method uses the *Nelder-Mead* search algorithm to search for and use an optimal configuration in the same execution.

[4]https://github.com/OpenMPToolsInterface/LLVM-openmp

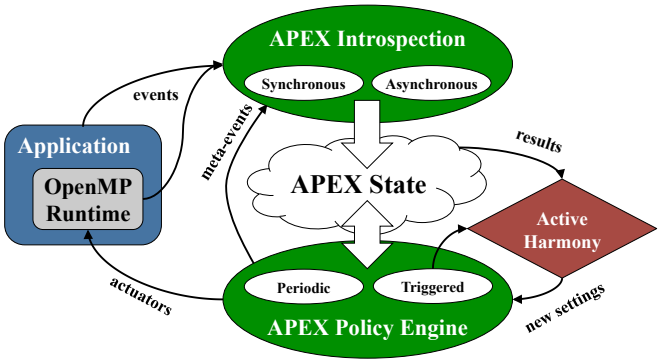


Fig. 2: ARCS framework, based on the original APEX design.

Prior to running the examples with the framework, the NPB 3.3-OMP-C OpenMP benchmarks were exhaustively parameterized to explore the full search space for the OpenMP environment variables `OMP_NUM_THREADS` and `OMP_SCHEDULE` (schedule type and chunk size). From that initial dataset, the search space was manually reduced. Unlike the initial parameter search, ARCS can tune the settings for each OpenMP parallel region independently. The reduced set of search parameters was used to limit the search space that had to be explored at runtime. The final ranges explored by ARCS are listed in Table I.

TABLE I: Set of ARCS search parameters for OpenMP parallel regions.

Parameter	Set of values
Number of threads (Crill)	2, 4, 8, 16, 24, 32, default
Number of threads (Minotaur)	10, 20, 40, 80, 120, 160, default
Schedule Type	dynamic, static, guided, default
Chunk Size	1, 8, 16, 32, 64, 128, 256, 512, default

Using the policy engine, we designed a policy to tune OpenMP thread count, schedule, and chunk size based upon the reduced search space described above. At program initialization, the policy registers itself with the APEX policy engine, and receives callbacks whenever an APEX timer is started or stopped. The OMPT interface starts a timer upon entry to an OpenMP parallel region and stops that timer upon exit. When a timer is started for a parallel region which has not been previously encountered, the policy starts an Active Harmony tuning session for that parallel region. When a timer is stopped, the policy reports the time to complete the parallel region. When a timer is started for a parallel region which has been previously encountered, the policy sets the number of threads, schedule, and chunk size to the next value requested by the tuning session, or, if tuning has converged, to the converged values. When the program completes, the policy saves the best parameters found during the search. When the same program is run again in the same configuration in the future, the saved values can be used instead of repeating the search process.

C. Overhead

The main overhead of ARCS can be characterized into three different types.

- **Configuration changing overhead**: ARCS changes the runtime configuration each time a region is executed. To change these configurations, ARCS uses the OpenMP runtime library routine `omp_set_num_threads()` and `omp_set_schedule()`. Time consumed during these routine calls adds some extra overhead. We call this overhead *Configuration Changing overhead*. This overhead is present in both Online and Offline strategies. In Crill, we calculated this overhead to be about 0.0008 sec in each region call. If a region is large enough, this overhead becomes insignificant. However, if the the region time is not large enough this overhead can become a significant factor.
- **APEX instrumentation overhead**: Overhead incurred due to APEX runtime instrumentation. Just like *Configuration changing overhead*, the impact of this overhead is also dependent on the region execution time. It is present in both Online and Offline strategies.
- **Search overhead**: In the online search strategy, finding the optimal configuration requires ARCS to test several runtime configurations before converging. Many of these configurations are not optimal, and as a result these sub-optimal configurations incur extra execution time. This additional execution time can be termed as *Search overhead*. This overhead is only present in the Online strategy. It is not present in *Offline strategy*, because in *Offline strategy* ARCS does not search for the the optimal configuration, it reads it from the history file only once during the whole application lifetime. We observed this overhead to vary across regions based on how fast they converge to the optimal configuration. During our experimentation, we observed this overhead to reach as high as 10% of the total execution time.

IV. Experimentation

A. Test System

We evaluated our framework on two different systems, Crill and Minotaur. These systems differ in architecture, number of cores, memory size and power consumption.

Crill (hosted at the University of Houston) is a dual socket machine with two 2.4 GHz quad-core Intel® Xeon® E5-2665 processors (Sandy Bridge architecture). It has a total of 16 cores (32 hyper-threaded threads) and 16 GB of memory. It runs on OpenSUSE 13.1 and has a TDP limit of 115W.

Minotaur (hosted at the University of Oregon) is an IBM® S822LC system equipped with two 10-core IBM POWER8® processors that operate at 2.92 GHz. It has support for 160 hardware threads (8 per core) and 256 GB of memory. It is running Ubuntu Linux, version 15.04.

B. Compiler & Libraries

We used GCC compiler version 4.9.2, the reference OpenMP runtime with OMPT support for our experimentation, and libmsr [13], a library that facilitates access to MSRs via RAPL interface for energy measurement and power capping.

C. Benchmarks

We used three proxy applications, LULESH 2.0, BT and SP to evaluate ARCS. We selected these benchmarks because they exhibit performance and load balancing behavior typical for a broad range of HPC applications.

LULESH 2.0 [14] is a shock hydrodynamics computational kernel from Lawrence Livermore National Laboratory. It approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. It is built on the concept of an unstructured hex mesh. It is one of the most used proxy applications in the HPC area, and it shows excellent load balancing and cache behavior. We used mesh sizes of 45 and 60 for our experimentation.

BT is a simulated CFD computational kernel that uses an implicit algorithm to solve 3-dimensional (3-D) compressible Navier-Stokes equations. The finite differences solution to the problem is based on an Alternating Direction Implicit (ADI) approximate factorization that decouples the x, y and z dimensions. The resulting systems are Block-Tridiagonal of 5×5 blocks and are solved sequentially along each dimension. This application shows good load balancing behavior. We used data set sizes B ($102 \times 102 \times 102$) and C ($164 \times 164 \times 164$) with custom 1000 time steps.

SP is a simulated CFD computational kernel that has a similar structure to BT. The finite differences solution to the problem is based on a Beam-Warming approximate factorization that decouples the x, y and z dimensions. The resulting system has Scalar Pentadiagonal bands of linear equations that are solved sequentially along each dimension. It shows good load balancing behavior but poor cache behavior. For SP, we also used data set sizes B ($102 \times 102 \times 102$) and C ($164 \times 164 \times 164$) with custom 1000 time steps.

Both ***BT*** and ***SP*** are from from NAS parallel benchmark suite [15], version 3.3-OMP-C.

D. Experimental Details

We carried out extensive experiments to evaluate the impact of ARCS. We considered both the execution time and energy consumption during the evaluation. An optimal OpenMP runtime configuration for a region is dependent on the region's characteristics, power cap level, workload size, and architecture. For that reason, we designed our experiments in such a way that they cover all these scenarios. We tested ARCS on five different power levels, two different workloads, and two distinct architectures (Intel Sandy Bridge and IBM POWER8).

As mentioned before, our primary experimental resource Crill is equipped with Intel Sandy Bridge processors, and our secondary resource Minotaur with IBM POWER8 architecture. In Crill, we had power capping privilege and access to the energy counters. For that reason we were able to evaluate the impact of ARCS at different power levels. We experimented on 55W, 70W, 85W, 100W and 115W (TDP for this processor)

power level. We only limited the processor power (package power). A package consists of cores, caches and other internal circuitry. We used maximum power for other components (DRAM, Network card, etc.), because we did not have capping capability on these subsystems. We used RAPL for power capping and collecting energy information. We tried to tackle known issues of RAPL such as counter update frequency and the warm up period after enforcing a power cap during the experimentation to get reliable energy readings. We ran each experiments three times. We report the average of these runs for Crill(Sandybridge) as it was a dedicated resource. However, we report the minimum of these three runs for Minotaur(Power8) as it was a shared resource. We did this to minimize any interferance.

As Minotaur is a relatively new resource, we did not have energy counter access nor power capping privilege. Therefore all the experiments conducted on this machine were using the default (TDP) power level of this machine. Also, all the evaluation done on this machine is based on execution time only. We evaluated both Online and Offline ARCS strategies in the above-mentioned environments.

V. Results and Analysis

In this section we present our experimental results. Through these results we show the impact of ARCS on different types of OpenMP applications. As mentioned previously, we evaluated ARCS on three different OpenMP applications. These applications vary in scalability, load balancing, and cache behavior. ***LULESH*** is a well-balanced application with good cache behavior. ***BT*** is also fairly well balanced with good cache behavior. ***SP*** is well balanced but shows poor cache behavior. We mainly concentrated on scalability, load balancing and caching because these are the behaviors that impact OpenMP performance the most.

In an OpenMP application with loop level parallelism, these behaviors can be controlled by the number of threads, scheduling policy and chunk sizes. The number of threads has a significant impact on scalability while scheduling policy and chunk sizes are very important for good load balancing and cache behavior. These behaviors not only affect the execution time performance, but they also impact energy consumption. Load balancing and cache behavior of an application are two of the main factors that define an application's energy profile.

Applications with bad cache behavior tend to consume more energy [16]. If there is a cache miss, the system has to do the extra work of fetching the data from the next level of cache or memory and in the process use I/O path which leads to extra energy consumption.

On the other hand, load balancing affects the energy consumption in a different way. Poor load balancing of an application leads the cores to wait in idle states in the synchronization points (barriers). Lightly loaded threads wait for highly loaded threads to finish their work. Even though current processors do a decent job at saving energy by entering the sleep state while waiting, entering and exiting sleep states incurs non-trivial overheads and can cause negative savings if the idle duration is short [17]. In OpenMP regions, the waiting time is usually short. Therefore, improving the load balancing behavior is crucial to improving the energy profile of an OpenMP application. Not only that but also these behaviors impact an OpenMP application's power profile, as power is the ratio of the energy consumption and execution time.

Moreover, cores and caches are the main power consuming components of a processor [18]. The total power of a processor is divided between these two components. So when a power cap is imposed on a processor, it not only affects the performance of the cores but also impacts the cache performance. As a result, the load balancing and cache behavior also change with the change of the power cap.

Furthermore, these behaviors vary across different regions of an application. Therefore, choosing an optimal configuration (number of threads, scheduling policy, and chunk sizes) for each regions separately is no trivial task. But we show through extensive analysis that ARCS is able to do this job very proficiently.

In the following discussion, we analyze each application separately. We show that ARCS can potentially improve performance across different types of applications. We also demonstrate the effect of ARCS strategies at both application and region level using detailed analysis of dynamic features. We show the performance behavior across different power caps and different workload sizes. Finally, we show the ARCS performance across different architectures.

We compare the performance of ARCS strategies with the default configuration. The default configuration uses maximum number of available threads, static scheduling, and chunk sizes calculated dynamically by dividing total number of loop iterations by number of threads. We concentrate on both online and offline strategies for ARCS. Results shown here is based on Crill, unless mentioned otherwise. The same applies for the power cap; if nothing is mentioned, that means we are using the highest power cap (TDP).

A. SP

SP is an application which shows a good load balancing behavior and poor cache behavior with the default configuration. SP consists of 13 loop based OpenMP regions. However, almost 75% of it's execution time is spent on four regions (`compute_rhs`, `x_solve`, `y_solve` and `z_solve`). Among them, `compute_rhs` has a poor load balancing and cache behavior, `x_solve`, `y_solve` and `z_solve` regions have good load balancing behavior but show poor cache behavior. To improve these regions' performance, their load balancing and cache behavior has to be improved. Therefore, we need to find configurations that improve the load balancing and cache behavior of these regions. To find such configurations we applied ARCS on this application. Table II shows the optimal configuration chosen by ARCS-Offline strategy for these regions at TDP power.

In Figure 3 we show the feature comparison between the default configuration and the configurations chosen by *ARCS-Offline*, the best ARCS strategy. We compare the L1

TABLE II: Optimal configuration chosen by ARCS-Offline strategy for SP regions.

Region	Optimal Configuration (Thread, Schedule, Chunk)
compute_rhs	16, guided, 8
x_solve	16,guided, 1
y_solve	8, static, default
z_solve	4, static, 32

cache miss rate in Figure 3a, L2 cache miss rate in Figure 3b, L3 cache miss rate in Figure 3c and OpenMP barrier (OMP_BARRIER) time in Figure 3d. The L1, L2 and L3 cache miss rates show the cache behavior of these regions. The OMP_BARRIER time shows the load balancing behavior; greater OMP_BARRIER time is a symptom of poor load balancing. For all of these metrics, lower values indicate better performance.

From these figures, we observe that all four regions show better cache and load balancing behavior with the ARCS strategy. Using the configuration chosen by ARCS, the OMP_BARRIER time is decreased by more than 50% in all four regions compared to the default configuration, shown in Figure 3d. The best improvement, which is more than 80% is achieved in the z_solve region while a relatively smaller improvement (around 50%) is achieved in compute_rhs.

We also observed L1, L2 and L3 cache miss rate improvement. Although L1 and L2 cache behaviors show good improvement, the biggest improvement (up to 90%) is visible in L3 cache behavior. This is important for performance because L3 cache misses have the highest cache miss penalty. The improvement also shows that these configurations enabled different cores to maximize their use of the shared L3 cache.

The above analysis shows that ARCS strategies can improve the cache behavior and load balancing of SP regions. This leads to the question: how much do these improvements affect the overall application's execution time and energy consumption? In Figure 4 we show the execution time and energy consumption comparisons between the default and ARCS strategies (*ARCS-Online* and *ARCS-Offline*). We show the results for five different power levels. We compare both execution time (in Figure 4a) and energy consumption (in Figure 4b). In Figure 4a we see that all the strategies in all five power levels outperform the default configuration by a large margin. The improvement varies between 26-40%. We observe similar behavior in energy consumption, shown in Figure 4b with the highest improvement touching 40% limit.

We were able to achieve so much improvement using ARCS because most of these time-consuming regions have a slight load imbalance and poor cache behavior. However there are applications which may have a very good load balance and cache behavior. In those kind of applications, the improvement will likely not be that significant, because there is very little room for ARCS to work on. In the later part of this section, we will look into such applications as well.

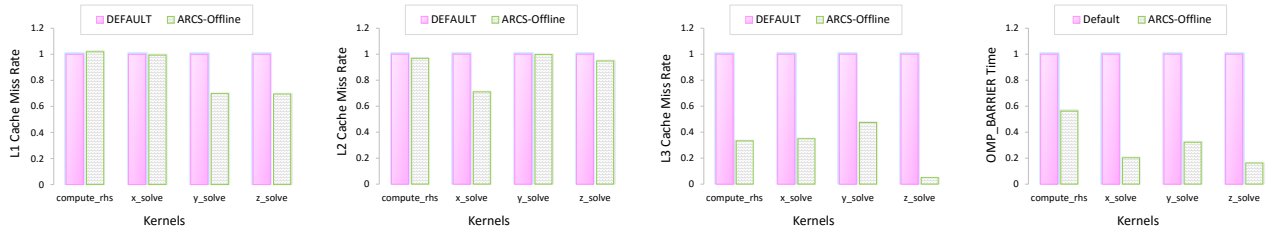
We discussed in Section II that the behavior of a region changes across different workloads. To see how efficient ARCS in choosing optimal configurations across workloads, we used ARCS on data set C of SP. Dataset C is four times larger than data set B. Figure 5 shows the execution time and energy consumption improvement at TDP (highest power cap). Even in this workload, we achieve execution time improvement of up to 40% and energy consumption improvement of up to 42% using ARCS strategies. It shows that ARCS can find optimal configurations across different workloads. We also observed that the configurations of the regions from SP differed across workloads which also proves the claim we made in Section II. To validate ARCS's consistency across different architectures, we used ARCS on a new architecture, IBM POWER8 (Minotaur). Minotaur differs significantly compared to Crill. Even so, when we ran SP with data set B in Minotaur, we observed 37% execution time improvement compared to the default strategy. This result demonstrates ARCS's versatility across architectures.

B. BT

BT is an application with good load balancing and cache behavior. BT is very similar to SP in structure although the approximate factorization is different. Like SP, majority of its execution time is also dependent on four regions (compute_rhs, x_solve, y_solve and z_solve). However, the behavior of these regions is slightly different. Three of these regions (x_solve, y_solve and z_solve) show very good load balancing and cache behavior in the default configuration. Only compute_rhs shows poor scaling, load balancing, and cache behavior. As a result, ARCS has a limited opportunity to improve the performance of this application. compute_rhs is the only region where ARCS strategies can have a significant effect, as all other regions already perform very well with the default strategy. In addition, compute_rhs is algorithmically hard to optimize due to its long stride memory access. Specifically, the second-order stencil operation in rhsz uses the $K \pm 2$, $K \pm 1$ and K elements of the solution array to compute RHS for the z direction:

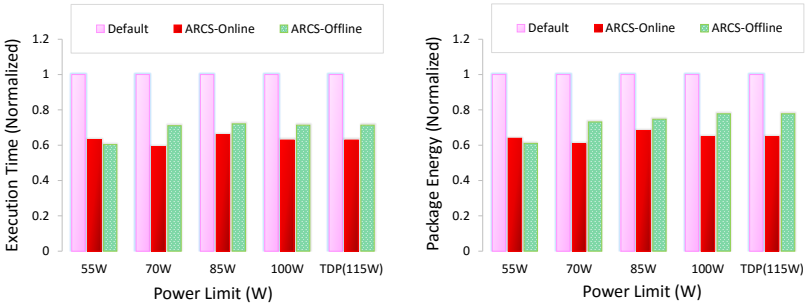
$$\begin{aligned} RHS(I,J,K) = & A * U(I,J,K-2) + \\ & B * U(I,J,K-1) + C * U(I,J,K) + \\ & D * U(I,J,K+1) + E * U(I,J,K+2) \end{aligned}$$

Such memory accesses are not cache friendly, so finding an optimal configuration for such a region is not trivial. However, ARCS does a very good job in finding an optimal configuration (24, guided, 1) for compute_rhs that improves the OMP_BARRIER and cache behavior of the region. The comparison between the *ARCS-Offline* and default strategy is shown in Figure 6. We compare the cache (L1, L2 and L3 cache miss rate) and load balancing (OMP_BARRIER time) behavior. We are only showing the result for compute_rhs region, because in other regions the improvement is negligible. For compute_rhs, the ARCS configuration shows a significant load balancing behavior improvement which is demonstrated by 80% OMP_BARRIER time improvement. It



(a) L1 cache miss rate (b) L2 cache miss rate (c) L3 cache miss rate (d) OMP_BARRIER time

Fig. 3: Feature comparison between the default and *ARCS-Offline* strategy at TDP power level. Comparison is done on four of the most time consuming regions of SP. Y-axis shows the normalized feature value. Smaller value is better.



(a) Execution Time (b) Package energy

Fig. 4: Application level execution time and package energy comparison among the default and ARCS strategies in SP at data set B. Comparison is done on five different power levels. Smaller value is better.

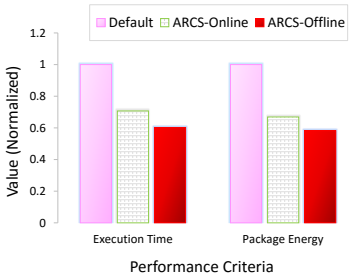


Fig. 5: Execution time and energy consumption comparison of ARCS strategies and the default strategy in data set C of SP. Smaller value is better.

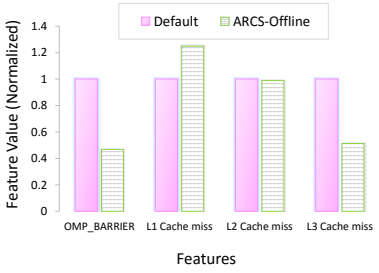


Fig. 6: Feature comparison between the default and *ARCS-Offline* strategy at TDP power level for `compute_rhs` region of BT. Smaller value is better.

also shows good L3 cache miss rate improvement indicating better cache utilization among different cores.

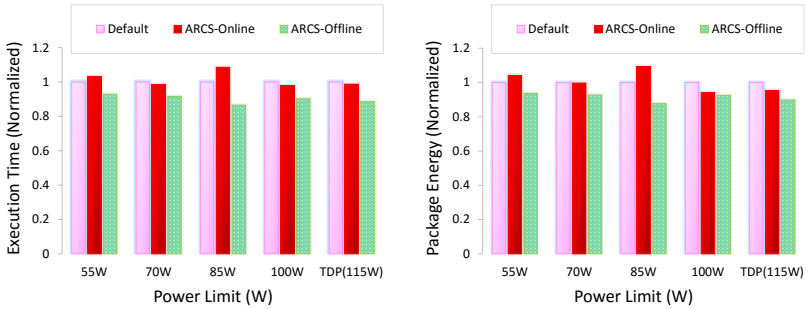
The impact of these behaviors is also visible in the overall application level execution time and energy consumption comparison in Figure 7. Here, we compare the execution time(7a) and energy consumption(7b) among the default and ARCS strategies(***ARCS-Online and ARCS-Offline***). We show the results for all five power levels. We observe that the execution time improvement is small across all power levels, with the highest improvement recorded is 13% at 85W power cap with *ARCS-Offline* strategy. In some cases ARCS actually performs worse than the default strategy (e.g., ARCS-Online at 85W). This is because in those cases small improvement achieved by ARCS is offset by the overhead. Similar behavior is visible for package energy in Figure 7b.

We also observed similar trend at Power8 architecture. Only the *ARCS-Offline* strategy was able to achieve an application level improvement of 18%.

C. LULESH 2.0

In Figure 8 we show the comparison of execution time and energy consumption comparison between the default strategy and *ARCS-Online* and *ARCS-Offline* strategies on both Crill and Minotaur. In Minotaur, We achieved a 40% execution time improvement using the *ARCS-Offline* strategy, while with *ARCS-Online* we achieved around a 4% improvement.

However, in Crill, the improvement is not evident. With *ARCS-Offline* strategy, we achieved about 3% execution time



(a) Execution Time (b) Package energy

Fig. 7: Application level execution time and package energy comparison among the default and ARCS strategies in BT with data set B. Comparison is done on five different power levels. Smaller value is better.

improvement in the smallest (55W) and the highest (115W) power levels. However, we lost performance on other three power levels. We achieved energy consumption improvement in all five power levels with maximum of 26% coming in 85W power level. As for *ARCS-Online* strategy, we observed a degradation in both execution time and energy consumption for every power levels as compared to default.

To understand why ARCS is performing poorly with LULESH on Crill, we did an extensive analysis. We used TAU [19] for our analysis. We profiled LULESH running with the default configuration at the highest power cap. In Figure 9 we show the top five regions based on total time (inclusive time). Through three OMPT events we show how these regions spent their time. These OMPT events are,

- `OpenMP_IMPLICIT_TASK`, it reports the total time spent by an implicit task, in other words it shows the overall execution time of the region.
- `OpenMP_LOOP` reports the execution time that is spent only on the loop body.
- `OpenMP_BARRIER/OMP_BARRIER` reports the time spent on the implicit and explicit barriers.

We observe from Figure 9 that in terms of `OpenMP_IMPLICIT_TASK` the most time consuming region is `EvalEOSForElems_1`. But most of its time is spent on `OpenMP_BARRIER`. Only a small portion of time is spent on real computation which can be attributed by `OpenMP_LOOP` time. The same applies for the `CalcPressureForElems_1` region. Both of these regions have a very small execution time per region call, `EvalEOSForElems_1` with 0.000828 sec and `CalcPressureForElems_1` with 0.000139 sec. And as we explained in the Overhead section, for each region run ARCS has a **Configuration changing overhead** of around 0.0008 sec. For these regions this overhead becomes a huge issue. In fact the overhead becomes almost 100% and 600%. Combined with APEX instrumentation overhead, ARCS looses a significant amount of performance in these tiny regions and in the process adds a fair amount of extra execution time.

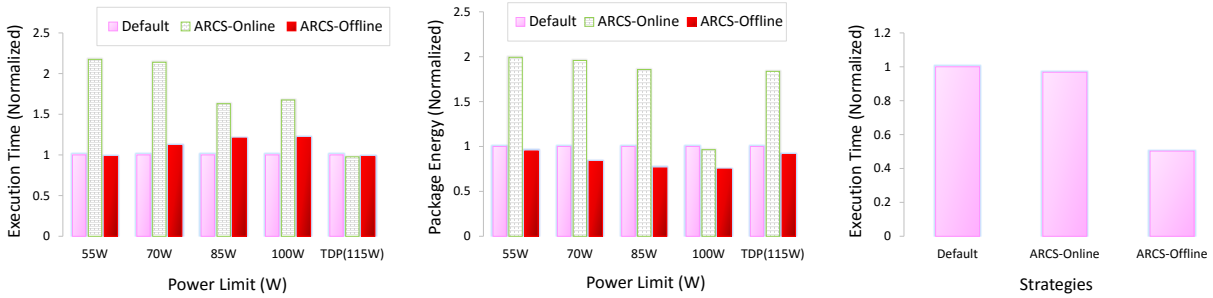
As for other three regions in Figure 9, although they have reasonable region time (execution time per region call), `CalcKinematicsForElems_1` and `CalcMonotonicQGradientsForElems_1` show near perfect load balancing behavior with only 1.8% and 0.26% of their total execution time spent in `OpenMP_BARRIER`. So there is not much ARCS can do to improve these regions' performance. However, the `CalcFBHourglassForceForElems_1` region shows slightly worse load balancing behavior with 16% of its total execution time spent in `OpenMP_BARRIER`, so ARCS can have some impact on its performance. ARCS was able to do so, which is evident in Figure 10. The figure shows OpenMP_BARRIER, L1, L2 and L3 cache miss rate comparison between the default and *ARCS-Offline* strategy on CalcFBHourglassForceForElems_1 region. From the figure we can see that the configuration (4, guided, 32) chosen by the *ARCS-Offline* strategy is able make the OpenMP_BARRIER time almost zero. It also shows that the configuration also improved the L1 and L3 cache miss rate significantly.

But execution time improvement from just this region was not enough to offset the overhead incurred by those tiny regions in Crill. However, these overheads are not energy hungry computation, that's why we still achieved overall energy improvement in all power levels.

As for Minotaur, we achieved execution time improvement for the following reason: Minotaur can support up to 160 threads without oversubscribing, which causes a bit more load imbalance in larger regions. As a result, ARCS improvement in those regions overcomes the overhead incurred by the smaller ones, which in turn results in overall application level improvement.

VI. Related Work

The paper by Bull et al. [20] is one of the first which provides an insight into the choices of the number of threads, scheduling policy and synchronization on an OpenMP application's performance. They show that selecting the best number of runtime parameters is not a trivial task as different applications behave differently. Suleman et al. [21] proposed a framework to dynamically control the number of threads



(a) Execution time(Crill) (b) Package energy (Crill) (c) Execution time (Minotaur)

Fig. 8: Application level execution time and package energy comparison among the default and ARCS strategies in LULESH, for mesh size 45. It shows results in both architectures. Smaller value is better.

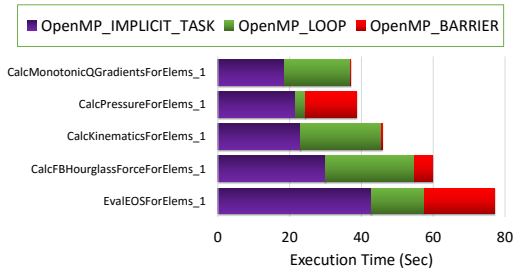


Fig. 9: OpenMP events data for top 5 time consuming regions from LULESH.

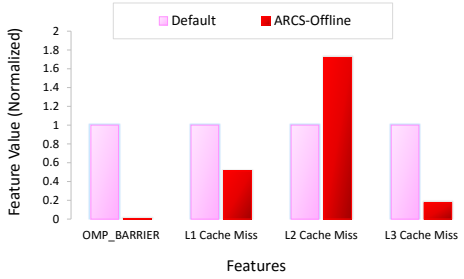


Fig. 10: Feature comparison among default and ARCS strategies on `CalcFBHourglassForceForElems_1` region.

using run-time information. It uses Feedback-Driven Threading (FDT) to implement Synchronization Aware Threading (SAT), which predicts the optimal number of threads using the amount of data synchronization. However, neither of these works consider power or energy consumption in their analysis, only execution time.

As the number of threads and processor frequency have a significant impact on performance and energy consumption of a given OpenMP application, many researchers have studied energy efficient performance prediction models for parallel applications. The work by Curtis-Maury et al. [22], [23] falls under this category. They employ dynamic voltage and frequency scaling (DVFS), dynamic concurrency throttling (DCT) and simultaneous multithreading (SMT) to implement various online and offline configuration selection strategies for OpenMP applications. Their main goal was to decrease energy consumption without loosing execution time. However, the work does not consider power budget. Peter Baily et al. [24] implemented an adaptive configuration selection scheme for both homogeneous and heterogeneous power constrained systems. It considers only two parameters – number of threads and processor frequency. Although the system selects these parameters for a given power budget, more than 10% of the time it violates the given power budget. The approach is not useful for a system working under a strict power budget. Dong Li et al. [25], [26] used DVFS and DCT to select energy efficient configurations for threads and operating frequency for MPI/OpenMP hybrid applications. They also did not consider a power budget. Their main target was to save energy without loosing execution time. The work by Wei et al. [27] shows the impact of optimal operating frequency on energy consumption improvement for parallel loops. It uses different operating frequency across different loops using frequency modulation techniques. In contrast to these works, ours concentrates on a complete set of runtime parameters on a strict power constrained system.

Power has become a limiting factor for large scale HPC centers. As a result, research on over-provisioned systems with a strict power budget is gaining popularity in the HPC community. Work by Rountree et al. [28] is one of the first to explore the impact of power capping. They investigate how different power levels impact the performance of different types of applications. Work by Patki et al. [2] explores the impact of hardware over-provisioning on the system level performance. The main contribution of their work was to select the number of nodes, number of cores per node, and power cap per node. Work by Aniruddha et al. [29] and Bailey et al. [30] consider only two parameters, DVFS and number of threads, as configuration options. They focus on overall system level performance on a MPI/OpenMP hybrid application. Compared to these works, our work concentrates on single node OpenMP performance given a power budget to that node.

VII. Conclusion and Future Work

Application power budgeting with over-provisioned systems is becoming an attractive solution to handle the power chal-

lenge in future HPC platforms. Previous work in this area only looks at distributed programming models. However, intra-node performance at different power levels is also important. OpenMP API is mostly used to exploit parallelism for shared memory processors. In this paper, we presented the ARCS framework that selects the best run-time configurations under imposed power constraints for OpenMP applications. Our framework handles a larger configuration search space as compared to prior work. We show that our framework is practical with varying data sets as well as architectures. We tested ARCS using three proxy applications, SP, BT and LULESH. We show that for a given power level, efficient OpenMP runtime parameter selection can improve the execution time and energy consumption of an application up to 40% and 42% respectively.

In future work, we plan to improve ARCS to enable selective tuning for OpenMP regions to avoid overheads on the smaller regions. We also intend to account for memory power in addition to processor power. Currently, we are not looking into the DVFS (Dynamic Voltage Frequency Scaling) strategy. We plan to include this policy in the future. We also aim to extend the power management policy of the framework for heterogeneous nodes.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under SI2-SSI grants CCF-1148052 and OCI-1148346. Support for developing APEX was provided through the the Advanced Computing (SciDAC) programs funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under award number DE-SC0008638. The Minotaur POWER8 test system was supported by an IBM equipment donation and Faculty Award.

REFERENCES

[1] The ASCAC Subcommittee on Exascale Computing, "The opportunities and challenges of exascale computing," United States Department of Energy, Tech. Rep., Fall 2010.

[2] T. Patki *et al.*, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 173–182.

[3] Z. Wang *et al.*, "Mapping parallelism to multi-cores: a machine learning based approach," in *ACM Sigplan Notices*, vol. 44, no. 4. ACM, 2009, pp. 75–84.

[4] A. E. Eichenberger *et al.*, "OMPT: An OpenMP tools application programming interface for performance analysis," in *IWOMP 2013*, 2013, vol. 8122, pp. 171–185.

[5] A. Eichenberger *et al.*, "OMPT and OMPD: OpenMP tools application programming interfaces for performance analysis and debugging," 2014, (OpenMP 4.0 draft proposal).

[6] K. A. Huck *et al.*, "Integrated Measurement for Cross-Platform OpenMP Performance Analysis," in *IWOMP 2014: Using and Improving OpenMP for Devices, Tasks, and More*. Springer International Publishing, 2014, pp. 146–160.

[7] K. Huck *et al.*, "An Early Prototype of an Autonomic Performance Environment for Exascale," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '13. New York, NY, USA: ACM, 2013, pp. 8:1–8:8. [Online]. Available: http://doi.acm.org/10.1145/2491661.2481434

[8] K. Huck *et al.*, "An Autonomic Performance Environment for Exascale," *Supercomputing Frontiers and Innovations*, vol. 2, no. 3, 2015. [Online]. Available: http://superfri.org/superfri/article/view/64

[9] H. Kaiser *et al.*, "Parallex an advanced parallel execution model for scaling-impaired applications," in *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*. IEEE, 2009, pp. 394–401.

[10] H. Kaiser *et al.*, "Hpx: A task based programming model in a global address space," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. ACM, 2014, p. 6.

[11] M. Anderson *et al.*, "A dynamic execution model applied to distributed collision detection," in *Supercomputing*. Springer, 2014, pp. 470–477.

[12] C. Ţăpuş *et al.*, "Active harmony: Towards automated performance tuning," in *2002 ACM/IEEE Conference on Supercomputing*, ser. SC '02. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=762761.762771

[13] B. R. Scott Walker, Kathleen Shoga *et al.*, "libmsr - a wrapper library for model-specific registers," https://github.com/LLNL/libmsr, 2014.

[14] I. Karlin *et al.*, "Lulesh 2.0 updates and changes," Tech. Rep. LLNL-TR-641973, August 2013.

[15] S. N. U. Center for Manycore Programming, "Implementation of NAS parallel benchmark using C," http://aces.snu.ac.kr/SNU_NPB_Suite.html, 2013.

[16] C.-L. Su *et al.*, "Cache designs for energy efficiency," in *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, vol. 1. IEEE, 1995, pp. 306–315.

[17] M. Arora *et al.*, "Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on cpu-gpu integrated systems," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 366–377.

[18] R. Sen *et al.*, "Cache power budgeting for performance," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2013.

[19] S. S. Shende *et al.*, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[20] J. M. Bull, "Measuring synchronisation and scheduling overheads in OpenMP," in *Proceedings of First European Workshop on OpenMP*, vol. 8, 1999, p. 49.

[21] M. A. Suleman *et al.*, "Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on cmps," *ACM Sigplan Notices*, vol. 43, no. 3, pp. 277–286, 2008.

[22] M. Curtis-Maury *et al.*, "Online strategies for high-performance power-aware thread execution on emerging multiprocessors," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[23] M. Curtis-Maury *et al.*, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 250–259.

[24] P. E. Bailey *et al.*, "Adaptive configuration selection for power-constrained heterogeneous systems," in *Parallel Processing (ICPP), 2014 43rd International Conference on*. IEEE, 2014, pp. 371–380.

[25] D. Li *et al.*, "Strategies for energy-efficient resource management of hybrid programming models," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 144–157, 2013.

[26] D. Li *et al.*, "Hybrid MPI/OpenMP power-aware computing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[27] W. Wang *et al.*, "Using per-loop cpu clock modulation for energy efficiency in OpenMP applications," *Energy*, vol. 1, no. 1.4, pp. 1–6, 2015.

[28] B. Rountree *et al.*, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 947–953.

[29] A. Marathe *et al.*, "A run-time system for power-constrained hpc applications," in *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings*, vol. 9137. Springer, 2015, p. 394.

[30] P. E. Bailey *et al.*, "Finding the limits of power-constrained application performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 79.