Performance and Energy Improvement of the ECP Proxy App SW4lite under Various Workloads

Xingfu Wu
Argonne National Laboratory
The University of Chicago, USA
Email: xingfu.wu@anl.gov

Valerie Taylor
Argonne National Laboratory
The University of Chicago, USA
Email: vtaylor@anl.gov

Zhiling Lan
Computer Science Department
llinois Institute of Technology, USA
Email: lan@iit.edu

***Abstract*—Energy efficient execution of scientific applications requires insight into how HPC systems affect the performance and energy of the applications. In this paper, we conduct experiments to evaluate the performance of SW4lite under various workloads with two different memory modes on Cray XC40 Theta at Argonne National Laboratory. We use MuMMI and ensemble learning to build the performance-counter-based performance and power models, and we identify performance and power bottlenecks based on the insights from these performance and power models. Then we improve the performance and energy of SW4lite with a focus on memory-centric optimizations and code modifications. The experimental results show that our performance counter-guided application optimization strategies result in up to 26.97% performance improvement and up to 19.44% energy saving for SW4lite with various workloads on up to 16,384 cores.**

***Index Terms*—performance optimization, energy improvement, hardware performance counters, performance modeling, power modeling, ensemble learning**

I. Introduction

Understanding the performance and energy consumption of large-scale applications is important not only for optimizing the applications but also for specifying next-generation high performance computing (HPC) systems. With next-generation HPC systems implementing more but less powerful cores with access to smaller per-core resources, optimization will become a more important and effective means of increasing performance. Energy efficient execution of scientific applications requires insight into how HPC systems affect the performance and energy of the applications. This insight generally results from significant experimental analysis and possibly the development of performance and power models. To balance power and performance for energy efficiency, one must understand the relationships between runtime, power, and the unique characteristics of a large-scale scientific application. Insights about these relationships provide guidance for application optimizations to increase performance and to reduce energy. The potential optimizations may involve application code revision, system tuning, or a combination of both. In this paper, we use the performance and power modeling tool MuMMI (Multiple Metrics Modeling Infrastructure) [1], [2] which is an infrastructure that facilitates systematic measurement, modeling, and prediction of performance and power consumption, and performance-power tradeoffs and optimization, and ensemble learning [3] which combines several machine learning models to create a new ensemble model, to explore the combinations with the focus on memory-centric optimizations.

SW4lite [4] is one of DOE Exascale Computing Project (ECP) proxy applications [5], [6], and it is a bare bone version of SW4 [7], [8] (Seismic Waves, 4th order accuracy) intended for testing performance in a few important numerical kernels of SW4. SW4 implements substantial capabilities for 3-D seismic modeling with a free surface condition on the top boundary, absorbing super-grid conditions on the far-field boundaries, and an arbitrary number of point force and/or point moment tensor source terms. It uses a fourth order in space and time finite-difference discretization of the elastic wave equations in displacement formulation.

SW4lite is often used to evaluate and compare different hardware solutions [6], [9], [10], and to find solutions to overcome exascale challenges in co-designs. In [6], [11], the real application SW4 and proxy application SW4lite are evaluated and characterized, and SW4lite is representative of its parent SW4 with respect to computation and memory behavior and similar communication patterns. Further, in [12] the MPI communication patterns of SW4lite were analyzed in detail.

While SW4lite is often used to evaluate the performance on different architectures, the performance optimization of SW4lite itself has not been considered seriously. To address this gap, we investigate performance and energy of SW4lite under various workloads with two different memory modes on Cray XC40 Theta [13] at Argonne National Laboratory, and we use hardware performance counter-based performance and power modeling to identify performance and power bottlenecks based on the insights from the performance and power models, and then improve the performance and energy of SW4lite.

Hardware performance counter-based performance and power modeling [1] based on different system configurations and problem sizes provides a broader understanding of an application's usage of the underlying architecture. The modeling is used not only to predict performance and power but also to identify performance and power bottlenecks based on the insights from performance counter-based models.

In our previous work [14], we used MuMMI to model the application performance and power of LULESH [15] and then identified the most important performance counters. Based on the insights learned from our models and measurements under various workloads, we significantly improved performance and energy of the LULESH code. In our recent work [3], we used single-object and multiple-objects ensemble learning to model the four metrics: runtime, node power, CPU power and memory power to identify the most important counters to improve parallel cancer deep learning application performance and energy. We found ensemble learning provides the most robust counter ranking to identify the most important performance counters for potential application optimizations.

In this paper, we conduct experiments to evaluate the performance of SW4lite with two different memory modes (cache, flat) on Theta and then collect the data for 91 different configurations. Based on the dataset, we use MuMMI and ensemble learning to build the performance-counter-based models to identify the most important performance counters for performance and energy improvement. Then we improve the performance and energy of SW4lite with the focus on memory-centric application optimizations and the code modifications. The experimental results show that our performance counter-guided application optimization strategies result in up to 26.97% performance improvement and up to 19.44% energy saving for SW4lite with various workloads on up to 16,384 cores.

The remainder of this paper is organized as follows. Section 2 discusses main kernels and problem sizes of SW4lite. Section 3 analyzes the performance of SW4lite on a single node. Section 4 describes the data collection of SW4lite with various workloads. Section 5 discusses performance and power modeling using MuMMI and ensemble learning to identify the most important performance counters for potential application optimizations. Section 6 presents the experimental results in performance and energy improvement. Section 7 summarizes this paper and briefly discusses some future work.

II. Main Kernels and Problem Sizes of SW4lite

SW4 [8] solves the elastic wave equation in second order formulation and uses a fourth order accurate finite difference method with the explicit time stepping. SW4lite [4] was extracted from SW4 and is used for testing performance options and is distributed as one of ECP proxy applications. SW4lite consists of five main kernels as follows:

- **BC comm**: communication among MPI tasks for exchanging boundary conditions
- **BC phys**: imposing physical boundary conditions
- **Scheme**: evaluating the difference scheme for divergence of the stress tensor
- **Supergrid**: evaluating supergrid damping terms
- **Forcing**: evaluating the forcing functions

It uses **Total** to stand for the total runtime for the time step loop. In the rest of this paper, we use the term to report the application performance (total runtime).

The SW4lite version 1.1 package [4] provides five different types of input problems: LOH.1, LOH.2, topo, cartesian, and pointsource. The problem LOH.1 is from the SCEC (Southern California Earthquake Center) test suite [16]. It sets up a grid with a spacing h (=50, 100) over a domain (X x Y x Z) 30000 x 30000 x 17000. It will run from time t=0 to t=9. The material properties are given by the block commands. They describe a layer on top of a half-space in the z-direction. A single moment point source is used with the time dependency being the Gaussian function.

The problem LOH.2 is also from the SCEC test suite as well. It sets up a grid with a grid spacing (h) of 50 or 100 over a domain (X x Y x Z) 30000 x 30000 x 17000. It will run from t=0 to t=9. The material properties are given by the block commands. They describe a layer on top of a half-space in the z-direction. A large number of moment point sources are used with the time dependency given by the Brune function but with different offsets in time (depending on location). This models a fault plane in the y-z-plane. The base grid has grid size h=50 or 100. The material properties are extrapolated to the ghost point and the grid point on the mesh refinement interface (extrapolate=2).

topo stands for Gaussian hill analytical topography which includes Gaussian hill dataset and skinny dataset. cartesian provides the very small input problems for LOH.1. The small problem pointsource solves the elastic wave equation for a single point source in a half space.

In summary, SW4lite is strong scaling with all these problems. In order to analyze the performance and scalability of SW4lite, we mainly use the medium and large problems such as topo, LOH.1, and LOH.2 for SW4lite in our experiments.

III. Performance Analysis of SW4lite

We conduct our experiments on Cray XC40 Theta [13] at Argonne National Laboratory. Each Cray XC40 node has 64 compute cores with the support of 4 threads per core (one Intel Phi Knight Landing (KNL) 7230 with the thermal design power (TDP) of 215 W), shared L2 cache of 32 MB (1MB L2 cache shared by two cores), 16 GB high-bandwidth in-package memory Multi-Channel DRAM (MCDRAM), 192 GB DDR4 RAM, and a 128 GB SSD. Cray XC40 [17], [18] provides power management to operate more efficiently by monitoring, profiling, and limiting power usage in order to increase system stability by reducing heat dissipation, reduce system cooling requirements, reduce utility costs by minimizing power usage when rates are the highest and calculate the actual power cost for individual users and/or jobs. In this work, we use simplified PoLiMEr [19] which utilizes Cray's CapMC [17] and Intel's RAPL [20] to measure power consumption for the node, CPU and memory at node level on Theta. The power sampling rate used is approximately 2 samples per second (default).

MCDRAM is a high bandwidth (480GB/s), low capacity (16GB) memory. The DDR4 has the low bandwidth of 90GB/s and the large capacity of 192GB. MCDRAM can be configured as a shared last level cache L3 (cache mode) shown in Figure 1 or as a distinct NUMA node memory (flat mode) in Figure 2 or somewhere in between. For the flat mode, the default memory allocation preference is DDR4 first, then MCDRAM. With the different memory modes by which the system can be booted,

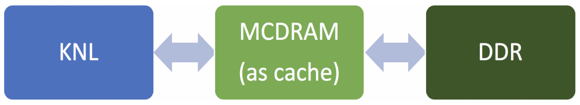

Fig. 1. Cache memory mode on Cray XC40 Theta

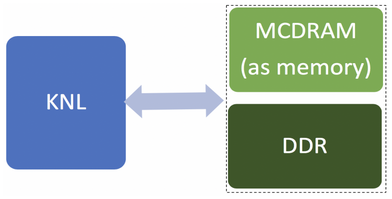

Fig. 2. Flat memory mode on Cray XC40 Theta

it becomes very challenging from a software perspective to understand the best mode suitable for an application [13], [21].

Figure 3 shows the performance of the SW4lite with the small problem pointsource for the cache and flat memory modes using up to 256 OpenMP threads on a single Theta node. For each memory mode (cache or flat), we use the total application runtime for utilizing 64 OpenMP threads per node with one thread per core as the baseline to normalize them to calculate the normalized ratio for using different numbers of threads per node. Notice that using 256 OpenMP threads per node means using 4 threads per core because each node has 64 cores. Therefore, the normalized ratio for using 64 threads per node with both cache and flat memory modes are 1. We observe that using 64 threads per node with one thread per core results in the best performance for both memory modes in Figure 3.

TABLE I
USING 64 THREADS PER NODE WITH CACHE AND FLAT MEMORY MODES

Problem	pointsource	LOH.1-h100
flat/cache ratio	1.06	1.08

Figure 4 shows the performance of SW4lite with the large problem LOH.1-h100 for the cache and flat modes using up

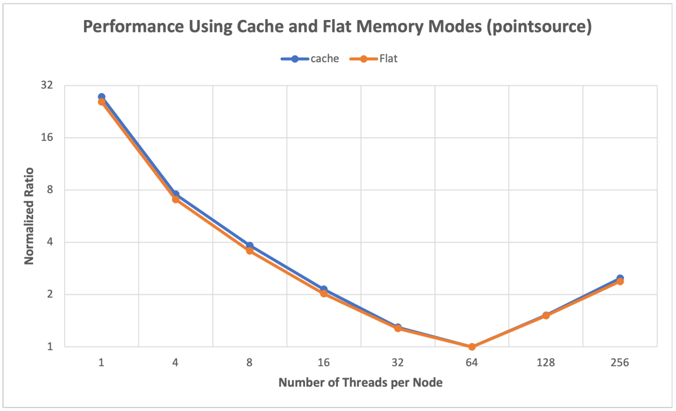

Fig. 3. Performance using cache and flat memory modes for the small problem pointsource

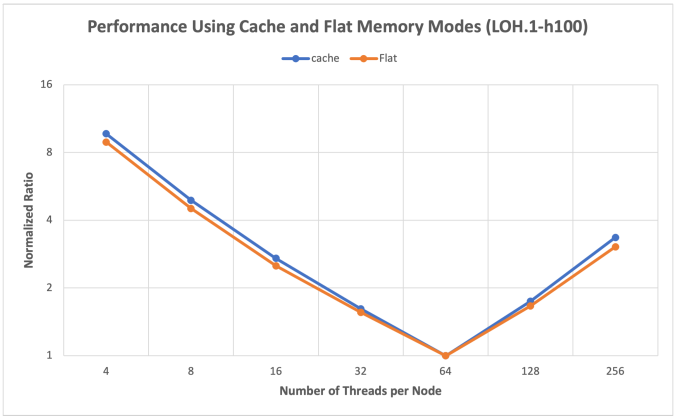

Fig. 4. Performance using cache and flat memory modes for the large problem LOH.1-h100

to 256 threads on a single Theta node. Similarly, we observe that using 64 OpenMP threads per node with one thread per core results in the best performance for both memory modes.

Overall, using 64 OpenMP threads per node with one thread per core results in the best performance for both cache and flat memory modes for SW4lite with the small problem pointsource and the large problem LOH.1-h100. Table I shows the performance comparison using 64 threads per node with cache and flat memory modes. For each problem, the flat/cache ratio is calculated as the ratio of the runtime for the flat mode and the runtime for the cache mode. We observe that for SW4lite, using the cache mode outperforms using the flat mode because the application with all the problem sizes used fits into the high bandwidth MCDRAM. Therefore, in the rest of this paper, we use 64 OpenMP threads per node and one thread per core with the cache mode for all of our experiments.

IV. Data Collection of SW4lite with Various Workloads

Based on the performance analysis in the previous section, we use one MPI process per node with 64 OpenMP threads per node and one thread per core and the cache mode to conduct all our experiments for SW4lite to collect performance and power data on Theta. We use the medium and large problems such as topo (gaussianHill.in, skinny.in), LOH.1 (LOH.1-h100.in, LOH.1-h50.in), and LOH.2 (LOH.2-h100.in, LOH.2-h50.in) to collect the data on up to 1024 nodes. SW4lite with these problems are strong scaling.

We use MuMMI [1], [2] with PAPI [22] to instrument the SW4lite code to collect 26 available performance counters. The 26 performance counters are TOT_CYC (total cycles), TOT_INS (total instructions completed), BR_CN (conditional branch instructions), BR_NTK (conditional branch instructions not taken), L1_TCM (L1 total cache misses), L1_LDM (L1 load misses), L1_DCM (L1 data cache misses), L1_ICA (L1 instruction cache accesses), L1_ICH (L1 instruction cache hits), L1_ICM (L1 instruction cache misses), L2_TCM (L2 total cache misses), L2_TCA (L2 total cache assesses), L2_TCH (L2 total cache hits), L2_LDM (L2 load misses),

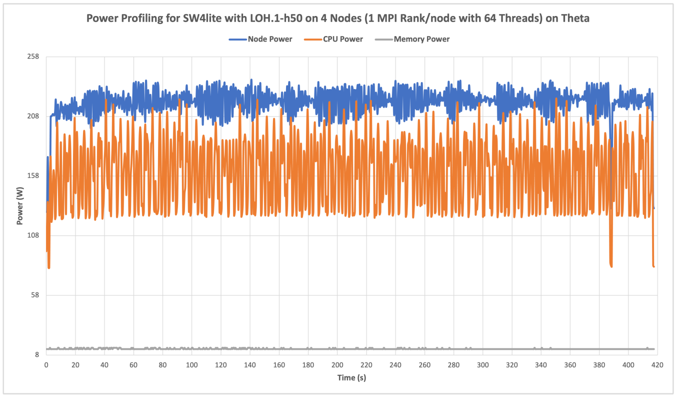

Fig. 5. Power consumption over time for SW4lite with the problem LOH.1-h50 on 4 nodes

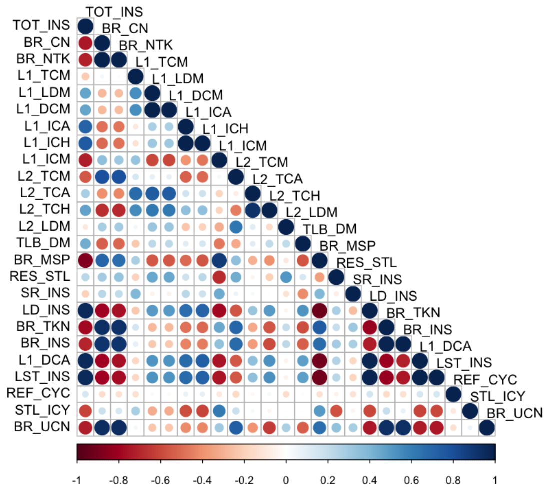

Fig. 6. Counter correlation matrix for SW4lite

TLB_DM (data translation lookaside buffer misses), BR_MSP (conditional branch instructions mispredicted), RES_STL (cycles stalled at any resource), SR_INS (store instructions), LD_INS (load instructions), BR_TKN (conditional branch instructions taken), BR_INS (branch instructions), L1_DCA (L1 data cache accesses), LST_INS (Load/store instructions completed), REF_CYC (reference clock cycles), STL_ICY (cycles with no instructions issued), and BR_UCN (unconditional branch instructions). Then TOT_CYC is used to normalize all the performance counters.

In this paper, the metrics for performance and power are runtime, node power, CPU power, and memory power. For instance, Figure 5 shows the power consumption over time for SW4lite with the problem LOH.1-h50 executed on 4 nodes, where the blue curve stands for the node power over time; the orange curve stands for the CPU power over time; and the grey curve stands for the memory power over time. This indicates that the CPU power is the dominant factor in the node power for SW4lite, and the frequent communication for exchanging boundary conditions causes the burst power patterns for the CPU power and node power.

We collected the data for 91 different configurations with 26 performance counters and four metrics per configuration plus application problem name and parameters for the six problem sizes from topo, LOH.1 and LOH.2. We use the dataset to analyze the pairwise correlations among 25 normalized hardware performance counters and four target objects. Figure 6 shows the performance counter correlation matrix for 25 counters. This indicates that most counters are not correlated each other. Figure 7 presents the correlation matrix for the four metrics: runtime, node power (power_sys), CPU power (power_cpu), and memory power (power_mem). It indicates that runtime is correlated with power. node power is highly correlated with CPU power because node power includes the CPU power, and CPU power is poorly correlated with memory power.

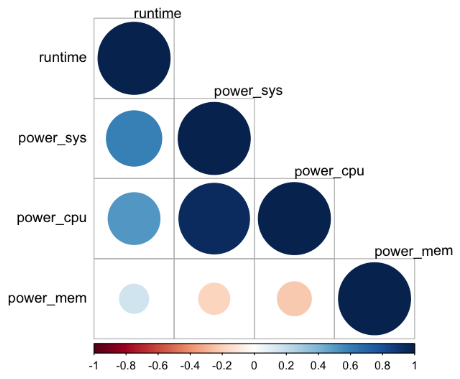

Fig. 7. Object correlation matrix for SW4lite

V. Performance and Power Modeling Using MuMMI and Ensemble Learning

In our previous work [1], [2], [23], [24], we used MuMMI to model four metrics: runtime, node power, CPU power, and memory power based on hardware performance counters with less than 7% prediction error rate, and we identified the most important counters to improve scientific application performance and energy. In our recent work [3], we used single-object and multiple-objects ensemble learning to model the four metrics to predict the results and to identify the most important counters to improve parallel cancer deep learning application performance and energy, and we found ensemble learning provides the most robust counter ranking to identify the most important performance counters for potential application optimizations. In this paper, we mainly focus on the four metrics as multiple objects to use MuMMI and ensemble learning to generate the performance and power models using the dataset from 91 different system configurations and problem sizes, thus providing a broader understanding of the application's usage of the underlying architecture. This in turn results in more knowledge about an application's performance and energy consumption on the given architecture. Because the focus of this paper is on identifying the most important counters to improve performance and energy of SW4lite using the modeling methods, it is not necessary to present the predicted results.

Based on the dataset for 91 different configurations, we split the data into the training and test datasets based on the 80/20% rule, and find out what the training and test datasets are by setting the seed 3456 of R's random number generator set.seed() so that creating the random objects can be reproduced. Then, we apply the same training and test datasets to MuMMI, ensemble learning and unsupervised recursive feature elimination to identify the most important counters from each model.

TABLE II
TOP 3 PERFORMANCE COUNTERS FOR EACH MODEL USING MUMMI

Models	Top 3 performance counters		
Runtime	REF_CYC	BR_INS	
Node Power	L2_TCH	TLB_DM	LD_INS
CPU Power	L2_TCH	LD_INS	TLB_DM
Memory Power	BR_MSP	BR_UCN	STL_ICY

We use MuMMI to build the models for runtime, node power, CPU power, and memory power. Table II lists the top 3 performance counters for each model. For the runtime model, REF_CYC (reference clock cycles) is dominant; for the node and CPU power models, L2_TCH (L2 total cache hits) is dominant; for the memory model, BR_MSP (conditional branch instructions mispredicted) is dominant.

TABLE III
TOP 3 PERFORMANCE COUNTERS FOR EACH MODEL USING MVTBOOST

Models	Top 3 performance counters		
Runtime	L1_ICM	SR_INS	L1_TCM
Node Power	BR_CN	BR_UCN	LD_INS
CPU Power	LD_INS	BR_UCN	L1_ICM
Memory Power	REF_CYC	BR_UCN	TLB_DM

Boosted decision tree ensembles such as gradient boosting machine (gbm) [25] are powerful ensemble learning algorithms, allowing dependent variables to be nonlinear functions of predictors. mvtboost (multivariate tree boosting) [26] extends gbm to multivariate, continuous object variables by fitting a separate univariate model of a common set of predictors to each object variable. This accounts for covariance in the object variables as in seemingly unrelated regression. This joint analysis of several object variables can be informative when we consider the four metrics for application improvement. We use a multivariate tree boosting method mvtboost to model performance and power and to rank the performance counters based on multiple objects. Table III shows that the dominant performance counters are L1_ICM (L1 instruction cache misses) in the runtime model, BR_CN (conditional branch instructions) in the node power model, LD_INS (load instructions) in the CPU power model, and REF_CYC in the memory power model.

TABLE IV
TOP 3 PERFORMANCE COUNTERS FOR EACH MODEL USING RFE

Models	Top 3 performance counters		
Runtime	BF_MSP	L1_ICM	BR_NTK
Node Power	BR_CN	BR_UCN	BR_NTK
CPU Power	BR_TKN	LD_INS	L1_ICM
Memory Power	TLB_DM	L1_DCA	REF_CYC

Then, we use and compare an unsupervised feature selection method with the model-based feature selections using MuMMI and mvtboost. For the unsupervised feature selection, we use recursive feature elimination (RFE) with random forests [27] to identify the most important performance counters. Table IV shows that the dominant performance counters are BR_MSP in the runtime model, BR_CN in the node power model, BR_TKN (conditional branch instructions taken) in the CPU power model, and TLB_DM (data translation lookaside buffer misses) in the memory power model.

Overall, based on the insights from these important performance counters with the focus on memory-centric application optimizations, our potential optimization efforts are to address the dominance of REF_CYC in the runtime model using MuMMI and in the memory power model using mvtboost, to improve the cache utilization (L1_ICM, L2_TCH) and to reduce TLB_DM.

VI. PERFORMANCE AND ENERGY IMPROVEMENT

In this section, based on the insights from the important performance counters identified by MuMMI and ensemble learning in the preceding section, our potential optimization efforts are threefold: to address the dominance of REF_CYC in the runtime model using MuMMI and in the memory power model using mvtboost, to improve the cache utilizations (L1_ICM, L2_TCH) and to reduce TLB_DM.

A. Looking into the SW4lite source code

The dominance of REF_CYC in the runtime and memory power models indicates that something else other than CPU performance limits the performance and power of SW4lite. Based on the collected performance data, the main kernels Scheme, Supergrid and Forcing in the SW4lite source code are dominated in the application execution time. the Scheme entails evaluating the difference scheme for divergence of the stress tensor; and Supergrid entails evaluating supergrid damping terms, and Forcing entails evaluating the forcing functions. So we focus on looking into the kernels for some hints.

When we look into their source codes with the focus on the large loops, we find that one issue is to apply "#pragma omp for" to the for statement "for $(k = 1; k \leq 6; k{+}{+})$" in the five files: rhs4sg.C, rhs4sg_rec.C, rhs4sg_recNW.C, rhs4sgcurv.C and rhs4sgcurv_rev.C. This limits the number of OpenMP threads to up to 6 per node for the "for statement" in these codes. For instance, the original code segment in the five files described above is as follows:

```
#pragma omp for
for( k=1 ; k<= 6 ; k++ )
   for( j=jfirst+2; j<=jlast-2; j++ )
      #pragma ivdep
      for( i=ifirst+2; i<=ilast-2; i++ )
      {
         ...
      }
```

It means that using 64 OpenMP threads per node with one thread per core for SW4lite results in only 6 threads with the workload and the others are idle. This is a big resource waste. We fix the problem by moving the OpenMP directive to the next loop level as follows:

```
for( k=1 ; k<= 6 ; k++ )
   #pragma omp for
   for( j=jfirst+2; j<=jlast-2; j++ )
      #pragma ivdep
      for( i=ifirst+2; i<=ilast-2; i++ )
      {
         ...
      }
```

TABLE V
PERFORMANCE COMPARISON FOR THE ORIGINAL AND REVISED CODES OF SW4LITE WITH THE PROBLEM LOH.1-H100

Number of Nodes	Original	Revised	Improvement (%)
1	96.38	78.71	18.33
2	54.19	46.30	14.56
4	36.99	31.51	14.81
8	22.97	20.22	11.97

After we revised these loops in the SW4lite code, we use the problem LOH.1-h100 to do initial performance test. Table V shows that we achieved between 11.97% and 18.33% performance improvement for the revision because of better utilization of using all 64 cores. This is a good performance improvement. Because of strong scaling and the increased communication time, the improvement percentage decreases with the increase of number of nodes.

Further, to improve the cache utilization, we put "#pragma unroll(6)" directive right before the loop to unroll the revised loop to improve the cache performance as follows:

```
#pragma unroll(6)
for( k=1 ; k<= 6 ; k++ )
   #pragma omp for nowait
   for( j=jfirst+2; j<=jlast-2; j++ )
      #pragma ivdep
      for( i=ifirst+2; i<=ilast-2; i++ )
      {
         ...
      }
```

The "#pragma unroll(6)" directive is a compiler optimization for loop unrolling. This pragma instructs the compiler to replicate the body of the loop six times and to reduce the loop control overhead. Table VI shows up to 9.40% performance improvement for the revised code with the unrolling and the nowait clause for "#pragma omp for" because of the enhancement of ILP (Instruction-Level Parallelism) which means more operations for the processor to push into processing pipeline without the worry about the for loop condition for each iteration in the unrolled version.

TABLE VI
PERFORMANCE COMPARISON FOR THE REVISED CODES OF SW4LITE WITH THE PROBLEM LOH.1-H100

Number of Nodes	Revised	Revised + unrolling	Improvement (%)
1	78.71	74.83	4.93
2	46.30	44.71	3.43
4	31.51	31.40	0.35
8	20.22	18.32	9.40

TABLE VII
PERFORMANCE COMPARISON USING DIFFERENT HUGE PAGE SIZES FOR THE REVISED CODE OF SW4LITE WITH THE PROBLEM LOH.1-H100

Huge Page Sizes	1	2	4	8
None	78.71	46.30	31.51	20.22
2MB	68.88	41.57	29.33	17.98
4MB	68.67	41.88	29.07	18.07
8MB	68.54	41.21	29.13	18.02
16MB	68.87	41.86	29.12	17.99
32MB	69.07	41.69	29.02	18.10
128MB	68.96	41.29	29.23	18.16
512MB	71.01	47.56	29.11	18.00
2GB	68.69	41.52	28.90	18.33

B. *Utilizing huge pages*

TLB_DM occurs in several power models from Tables II, III, and IV. It indicates that we can utilize huge page sizes to reduce TLB_DM so that the cache performance and the overall application performance may be improved. Cray XC40 Theta supports the system modules for the huge page sizes of 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB and 2GB [13]. The default page size in Linux is 4KB. In order to reduce the TLB data miss (TLB_DM), the main kernel address space is mapped with huge pages—a single 8 MB huge page requires only a single TLB entry, while the same memory in 4 KB pages would need 2048 TLB entries. Utilizing huge pages decreases the system page table overhead so that the overall memory performance may be improved. However, the question is which huge page size will result in the best performance improvement. We load the system module for each huge page size to measure them to determine which huge page size will be used for further improvement.

In Table VII, we observe that 8MB huge page results in the consistent good performance on 1 to 8 nodes. This is the same to what we found in our recent work [3], where 8MB huge page resulted in the best performance for the CANDLE benchmarks [28]. We also observe that the next best one is 4MB huge pages. In the following further optimizations, we utilize 8MB huge page in our experiments.

TABLE VIII
RUNTIME (S), AVERAGE NODE POWER (W), ENERGY SAVING, AND PERFORMANCE IMPROVEMENT COMPARISON BETWEEN THE ORIGINAL AND IMPROVED CODES OF SW4LITE WITH THE PROBLEM LOH.1-H100 ON UP TO 512 CORES

Number of Nodes	Original		Improved		Energy Saving (%)	Perf. Improv (%)
	Runtime	Node Power	Runtime	Node Power		
1	96.38	222.23	70.39	248.07	18.47	26.97
2	54.19	211.86	41.95	223.94	18.17	22.59
4	36.99	194.27	29.00	211.97	14.46	21.60
8	22.97	187.07	17.57	203.20	16.91	23.51

TABLE IX
RUNTIME (S), AVERAGE NODE POWER (W), ENERGY SAVING, AND PERFORMANCE IMPROVEMENT COMPARISON BETWEEN THE ORIGINAL AND IMPROVED CODES OF SW4LITE WITH THE PROBLEM LOH.1-H50 ON UP TO 16,384 CORES

Number of Nodes	Original		Improved		Energy Saving (%)	Perf. Improv (%)
	Runtime	Node Power	Runtime	Node Power		
1	1620.11	241.67	1310.53	259.27	13.22	19.11
2	804.95	226.30	660.84	238.31	13.55	17.90
4	419.09	219.67	366.67	227.04	9.57	12.51
8	219.02	216.93	197.52	222.82	7.37	9.82
16	124.69	206.61	112.70	210.04	8.12	9.62
32	69.51	204.87	64.03	207.15	6.86	7.88
64	47.53	188.84	44.99	189.43	5.05	5.34
128	31.16	201.17	28.97	200.81	7.19	7.03
256	24.50	182.22	23.57	181.53	4.16	3.80

TABLE X
RUNTIME (S), AVERAGE NODE POWER (W), ENERGY SAVING, AND PERFORMANCE IMPROVEMENT COMPARISON BETWEEN THE ORIGINAL AND IMPROVED CODES OF SW4LITE WITH THE PROBLEM LOH.2-H100 ON UP TO 512 CORES

Number of Nodes	Original		Improved		Energy Saving (%)	Perf. Improv (%)
	Runtime	Node Power	Runtime	Node Power		
1	102.61	219.32	76.26	237.73	19.44	25.68
2	57.55	209.00	46.31	218.81	15.75	19.53
4	38.53	197.04	31.42	203.32	15.85	18.45
8	22.97	189.67	20.11	192.80	11.01	12.45

TABLE XI
RUNTIME (S), AVERAGE NODE POWER (W), ENERGY SAVING, AND PERFORMANCE IMPROVEMENT COMPARISON BETWEEN THE ORIGINAL AND IMPROVED CODES OF SW4LITE WITH THE PROBLEM LOH.2-H50 ON UP TO 16,384 CORES

Number of Nodes	Original		Improved		Energy Saving (%)	Perf. Improv (%)
	Runtime	Node Power	Runtime	Node Power		
1	2248.09	242.20	1901.94	244.93	14.44	15.40
2	1202.57	211.73	932.33	243.26	10.93	22.47
4	553.82	216.84	470.28	228.74	10.42	15.08
8	286.98	215.66	245.34	223.50	11.40	14.51
16	148.95	213.87	131.37	217.84	10.17	11.80
32	82.51	200.44	75.52	202.42	7.57	8.47
64	52.83	193.80	49.56	195.56	5.34	6.19
128	32.92	197.09	30.74	196.52	6.89	6.62
256	24.47	183.39	22.65	183.87	7.20	7.44

C. *Combination of all improvement strategies*

In the section, we apply the combination of the revised code with the unrolling and utilizing 8MB huge page (called the improved code) to the large problem sizes such as LOH.1-h100, LOH.1-h50, LOH.2-h50, and LOH.2-h50. We compare the improved code with the original code for these problem sizes to investigate how much performance improvement and energy saving we can achieve on up to 16,384 cores on Theta.

We summarize the experimental results for SW4lite with the problem LOH.1-h100 in Table VIII, where "Original" stands for the original SW4lite code; and "Improved" stands for the improved code for the combination of all improvement strata-gies. We achieve up to 26.97% in performance improvement and up to 18.47% in energy saving on up to 512 cores. This is a large performance improvement and energy saving. As we discussed above, the main performance improvement from the revised code results in the large energy saving.

For the experimental results of SW4lite with the large problem LOH.1-h50 shown in Table IX, we achieve up to 19.11% in performance improvement and up to 13.55% in energy saving on up to 16,384 cores. With increasing the number of nodes, the performance improvement percentage decreases because this application is strong scaling and our application optimization effort focused on the computation. The decreased workload per core results in the smaller total runtime but the low-power communication overhead increases. This is why the power consumption decreased in the overall trend.

Table X shows the experimental results of SW4lite with the problem LOH.2-h100. We achieve up to 25.68% in performance improvement and up to 19.44% in energy saving on up to 512 cores.

Table XI shows the performance improvement and energy saving results of SW4lite with the large problem LOH.2-h50 on up to 16,384 cores. We achieve up to 22.47% in performance improvement and up to 14.44% in energy saving. Similar to what we found in Table IX, with increasing the number of nodes, the performance improvement percentage decreases.

Overall, our performance counter-guided application optimization strategies result in up to 26.97% performance improvement and up to 19.44% energy saving for SW4lite with various workloads on up to 16,384 cores.

VII. CONCLUSIONS

We conducted the experiments to evaluate the performance of SW4lite with two memory modes (cache and flat) on Theta and found that using 64 OpenMP threads with the cache mode resulted in the best performance for SW4lite. Then we used the medium and large problems for SW4lite to collect the data for 91 different configurations which provided a broader understanding of the application's usage of the underlying architecture, analyzed the performance counter and metrics correlation matrix, and applied MuMMI and ensemble learning to the dataset to build the performance and power models to identify the most important performance counters for the potential optimization efforts. Then we improved the performance and energy of SW4lite with the focus on the memory-centric application optimizations such as cache memory mode, loop unrolling and 8MB huge page and the code modifications. Our performance counter-guided application optimization strategies resulted in up to 26.97% performance improvement and up to 19.44% energy saving for SW4lite with various workloads on up to 16,384 cores.

For the future work, we will apply our ytopt autotuning framework [29], [30] to further tune the performance and energy of SW4lite and other ECP proxy applications under various workloads for energy efficiency.

Acknowledgments

This work was supported in part by DOE ECP PROTEAS-TUNE, in part by DOE ASCR RAPIDS2, and in part by NSF grants CCF-1801856 and CCF-2119203. We acknowledge the Argonne Leadership Computing Facility (ALCF) for use of Cray XC40 Theta under ALCF project EE-ECP. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] X. Wu, V. Taylor, J. Cook, and P. Mucci, "Using performance-power modeling to improve energy efficiency of HPC applications," *IEEE Computer*, vol. 49, no. 10, pp. 20–29, Oct. 2016.

[2] X. Wu, V. Taylor, C. Lively, H. Chang, B. Li, K. Cameron, D. Terpstra, and S. Moore, "MuMMI: Multiple metrics modeling infrastructure," in *Tools for High Performance Computing*. Springer, 2014.

[3] X. Wu and V. Taylor, "Utilizing ensemble learning for performance and power modeling and improvement of parallel cancer deep learning CANDLE benchmarks," *Concurrency and Computation: Practice and Experience*, vol. e6516, 2021.

[4] "SW4lite Version 1.1, ECP Proxy Application," https://github.com/geodynamics/sw4lite (Downloaded in Oct. 2019).

[5] "ECP Proxy Applications Suite," https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.

[6] D. F. Richards, O. Aaziz, J. Cook, S. Moore, D. Pruitt, and C. Vaughan, "Quantitative performance assessment of proxy apps and parents," Report for ECP Proxy App Project Milestone ADCD-504-9, April 30, 2020.

[7] B. Sjogreen and N. Petersson, "A fourth order accurate finite difference scheme for the elastic wave equation in second order formulation," *Journal of Scientific Computing*, vol. 52, no. 1, pp. 17–48, 2012.

[8] B. Sjogreen, "SW4 final report for iCOE," Technical Report LLNL-TR-759417, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, October 3, 2018.

[9] S. Hammond, C. Vaughan, and C. Hughes, "Evaluating the intel skylake xeon processor for hpc workloads," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, 16-20 July 2018.

[10] C. Hillairet, "Assessing seismic wave modelling on aws graviton2 with sw4lite," https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/assessing-seismic-wave-modelling-on-aws-graviton-2-with-sw4lite, September 9, 2020.

[11] O. Aaziz, J. Cook, J. Cook, T. Juedeman, D. Richards, and C. Vaughan, "A methodology for characterizing the correspondence between real and proxy applications," in *2018 IEEE International Conference on Cluster Computing (IEEE CLUSTER)*, Sep. 2018.

[12] N. Sultana, M. Rufenacht, A. Skjellum, P. Bangalore, I. Laguna, and K. Mohror, "Understanding the use of message passing interface in exascale proxy applications," *Concurrency and Computation: Practice and Experience*, vol. 33:e5901, 2021.

[13] "Theta, Cray XC40 system," https://www.alcf.anl.gov/theta.

[14] X. Wu, V. Taylor, J. Cook, and T. Juedeman, "Performance and power characteristics and optimizations of hybrid mpi/openmp lulesh miniapps under various workloads," in *SC2017 Workshop on Energy Efficient Supercomputing (E2SC'17)*, 2017.

[15] "LULESH: Livermore unstructured lagrangian explicit shock hydrodynamics," https://asc.llnl.gov/CORAL-benchmarks/Summaries/LULESH_Summary_v1.pdf.

[16] S. Day and et al., "Tests of 3d elastodynamic codes, final report for lifelines project 1a01, pacific earthquake engineering center," 2001.

[17] S. Martin, D. Rush, M. Kappel, M. Sandstedt, and J. Williams, "Cray xc40 power monitoring and control for knights landing," in *Proceedings of the Cray User Group (CUG) Conference*, 2016.

[18] "Cray, monitoring and managing power consumption on the cray xc system," Tech Report, S-0043-7204.

[19] I. Marincic, V. Vishwanath, and H. Hoffmann, "PoLiMEr: An energy monitoring and power limiting interface for hpc applications," in *SC2017 Workshop on Energy Efficient Supercomputing*, 2017.

[20] H. David, E. Gorbatov, U. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *ISLPED'10*, January 2010, pp. 189–194.

[21] N. Butcher, S. L. Olivier, J. Berry, S. D.Hammond, and P. M. Kogge, "Optimizing for knl usage modes when data doesn't fit in mcdram," in *2018 International Conference on Parallel Processing(ICPP2018)*, 2018.

[22] "PAPI (Performance API)," http://icl.cs.utk.edu/papi/.

[23] C. Lively, X. Wu, V. Taylor, S. Moore, H. Chang, C. Su, and K. Cameron, "Power-aware predictive models of hybrid (mpi/openmp) scientific applications on multicore system," *Comp. Sci. – Res. and Dev.*, no. 4, 2012.

[24] C. Lively, V. Taylor, X. Wu, H. Chang, C. Su, K. Cameron, S. Moore, and D. Terpstra, "E-AMOM: An energy-aware modeling and optimization methodology for scientific applications on multicore systems," *Comp. Sci. – Res. and Dev.*, vol. 29, no. 3, 2014.

[25] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, pp. 1189–1232, 2001.

[26] P. Miller, "mvtboost example," https://cran.r-project.org/web/packages/mvtboost/vignettes/mvtboost_vignette.html, Dec. 5, 2016.

[27] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. Springer, 2013.

[28] X. Wu, V. Taylor, J. M. Wozniak, R. Stevens, T. Brettin, and F. Xia, "Performance, energy, and scalability analysis and improvement of parallel cancer deep learning CANDLE benchmarks," in *Proceedings of 48th International Conference on Parallel Processing*, 2019.

[29] X. Wu, M. Kruse, P. Balaprakash, H. Finkel, P. Hovland, V. Taylor, and M. Hall, "Autotuning PolyBench Benchmarks with LLVM Clang/Polly loop optimization pragmas using Bayesian optimization," in *Proceedings of SC20 Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, ser. PMBS'20. Washington, DC, USA: IEEE Computer Society, 2020.

[30] X. Wu, A. Marathe, S. Jana, O. Vysocky, J. John, A. Bartolini, L. Riha, M. Gerndt, V. Taylor, and S. Bhalachandra, "Toward an end-to-end auto-tuning framework in HPC PowerStack," in *Proceedings of Energy Efficient HPC State of Practice 2020 (EE HPC SOP 20)*. Washington, DC, USA: IEEE Computer Society, 2020.