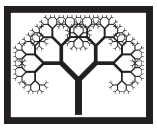
Paper 33



Proceedings of the Sixth International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering,
P. Iványi and B.H.V Topping (Editors)
Civil-Comp Press, Stirlingshire, Scotland

Fine-grained application tuning on OpenPOWER HPC systems

L. RIHA[1], **A. BARTOLINI**[2], **O. VYSOCKY**[1]
[1]IT4Innovations, VSB - Technical University of Ostrava, Czech Republic
[2]University of Bologna, Italy

Abstract

Energy consumption of HPC centers becomes major limitation in building a new peta or exascale system. In this paper we evaluate the approach of dynamic application tuning used to reduce energy consumption of HPC systems on the IBM's Power8+ system.

A D.A.V.I.D.E. HPC system installed in CINECA has been selected for these experiments due to its advanced power consumption monitoring system which provides power samples at high sampling rate of 1 kHz, that are stored in a stand-alone scalable database and can be read on request.

We have tuned two available hardware parameters: (1) Dynamic Voltage and Frequency Scaling and (2) concurrency throttling with focus on hyper-threading, which plays a significant role with respect to performance on Power8+ systems.

As a test application we have used the ESPRESO library, which is fully fledged application. Each part of such application may have its optimal configuration of the tuned parameters to utilize the system however at the same time without wasting the resources. This way we were able to reduce the application runtime about 23.7 % when tuning to reduce runtime or 27.3 % of energy with 9.9 % runtime savings when tuning for minimal energy consumption.

Keywords: high performance computing, energy efficient computing, performance analysis, DiG, Power Architecture, MERIC

1 Introduction

Energy consumption of HPC systems and applications is one of key metrics on the path to exascale. Several approaches have been presented to reduce it by trading-off performance and power consumption. These are based on reduction of available resources if the application cannot use them all. Haidar et al. [1] evaluate the impact of DVFS and power capping on supercomputing relevant kernels. Fraternali et al. [2] study the impact of DVFS and HW/SW variability in heterogeneous workloads. Bonati et al. [3] focuses on evaluation this trade-off in a multi-node multi-accelerator context. GEOPM [4] and Adagio [5] projects target the communication slack in complex application to save energy and/or improving their performance.

Horizon 2020 project READEX [6], described in more by [7], comes with the optimization technique that splits application into parts that shows different resources requirements and consequently it does dynamic switching of the resources limitation at the beginning and the end of such part.

Energy consumption cannot be reduced without access to information of energy consumed during the analyzed application run. Intel processors provide RAPL counters [8] that provide approximate energy consumption of processors and related memories. Similar counter are available also in AMD [9] and NVIDIA [10] hardware architectures.

Higher precision of the measurement can be provided by stand-alone hardware systems, connected not only to CPUs and memories but also to computational node itself. Examples of such system are HDEEM [11] or DiG [12]. DiG is available at D.A.V.I.D.E. system based on IBM's Power8+ processors. Its successor Power9 is currently at the heart of two the most powerful supercomputers Summit and Sierra [13].

We implement C/C++ open source MERIC library [14] providing application profiling and tuning of selected application parts according the READEX approach. Main goal of the library is to simplify the process of application analysis and provide maximum possible energy savings with minimum overhead caused by MERIC. MERIC has been extended to support D.A.V.I.D.E. system parameters tuning and reading DiG power samples. To our knowledge we are the first paper presenting energy consumption tuning on IBM Power architecture.

The paper is organized into the following sections. Section 2 presents used approach and the hardware which we use. Following Section 3 focus on the DiG energy measurement system and access the power samples from any application. Section 4 describes selected ESPRESO FEM that we have analyzed with MERIC. Significant gains when using MERIC on the D.A.V.I.D.E. system are presented in the Section 5. Finally, Section 6 summarize contribution of this paper.

2 Methodology

2.1 Dynamic Tuning Approach for Energy Optimization

Presented approach is based on dynamic tuning of selected hardware parameters, called *tuning parameters*, during the application runtime. The tuning parameters on Power8+ D.A.V.I.D.E. system are: (i) the CPU core frequency using Dynamic Voltage and Frequency Scaling (DVFS), (ii) number of active cores (using MPI processes) and (iii) number of active hyper-threads (using OpenMP threads). The concept of dynamic tuning is introduced by the Horizon 2020 [6] project and is described in more by [7].

In short, the process of dynamic tuning is performed in following three steps:

- *Step 1:* identify and annotate the significant regions of an application (see Figure 2 for regions that have been used for tuning of the ESPRESO FEM library).
- *Step 2:* find the optimal settings for the tuning parameters for each of these regions to minimize energy consumption.
- *Step 3:* dynamically switch to the optimal settings of tuning parameters during the application runtime.

Step 1 is done manually by a code developer. The goal is to identify the regions with different workloads (compute bound, memory bound, I/O bound, communication bound, etc.) and annotate them in the source code. Annotated regions, also called *significant regions*, are evaluated in step 2, which is called the analysis phase. During the analysis, the application is executed multiple times with all combinations of the tuning parameters to find the optimal settings for each significant region.

After manual annotation, the analysis is performed automatically using two tools: MERIC (performs parameter tuning and energy measurements) and RADAR (analyzes the measurements and creates the tuning model), see [14]. The optimal settings are then saved into a file called *tuning model*, which is used in step 3. Finally, the production runs are done in step 3, in which the dynamic tuning is performed by the MERIC tool. At application start-up, MERIC reads the tuning model from the file, and at the beginning of each significant region it changes the tuning parameters to the optimal settings for that particular region. At the end of the region the settings are changed back to the optimal ones for the parent region.

2.2 Power8+ CPU on D.A.V.I.D.E. system

D.A.V.I.D.E. [15] computing nodes are derived from IBM Power8+ System S822LC. Each node feature dual sockets IBM Power8+ with NVlink and four NVIDIA Tesla P100 HSXM direct liquid cooled. It uses Open Rack Enclosure with integrated piping and power distribution and each node is modified to fit the OCP form-factor. Each compute node has a peak performance of 22 TFlops and a power consumption of less than 2 KW. The IBM Power8+ CPUs on D.A.V.I.D.E. are 8 core chips with eight hardware threads each, for a total of 64 threads per CPU. Each socket is connected to eight memory buffers (Centaur chips) for a total of 1 TB of memory per node, with an aggregated 128 MByte L4 cache and 230 GB/s sustained memory bandwidth in and out of the processor.

2.3 Selected Hardware Parameters for Tuning

After examination of the possible hardware knobs on the D.A.V.I.D.E. system we have selected ones that can be potentially accessible in the user-space of the Linux operating system. Following set was identified:

1. *DVFS* – OpenPOWER supports DVFS tuning using CPUfreq kernel driver [16]. Power8+ CPU supports 60 different frequencies that can be set in range from 2.06 to 4.02 GHz with a step of 30 MHz. To limit the evaluation time we use coarse grained steps and run tests for full range but 9 frequencies only.

2. *hyper-threading* – This parameter is controlled using OpenMP threads and correct thread pinning. Each Power8+ CPU core can run up to 8 hyper-threads. For benchmarking we use 1, 2, 4 and 8 threads.

The utilization of CPU cores is controlled using MPI processes pinned to the particular cores. In this paper we use always all 16 CPU cores. The MPICH version 3.2 MPI library has been used for testing. To ensure that MPI processes and OpenMP threads are not allowed to migrate and use only proper resources we need to control the process/thread placement as well as binding. We use properties of the *mpirun* command of the MPICH 3.2 library and

its parameters "-bind-to" and "map-by" are set to "hwthread:$OMP_NUM_THREADS" and "hwthread:8", respectively, as shown in the following listing:

```
mpirun
  -n 16
  -bind-to hwthread:$OMP_NUM_THREADS
  -map-by  hwthread:8
  ./application ...
```

This setup means that each MPI process will have access to continuous 8 hyper-threads per 1 physical core. Following mapping will be used: MPI rank 0 runs on core 0 of socket 0, rank 1 runs on core 1 of socket 0, ..., rank 8 runs on core 0 of socket 1, ..., and finally rank 15 runs on core 7 of socket 1.

3 Energy Measurements on D.A.V.I.D.E.

3.1 DiG Power Measurement System

Each D.A.V.I.D.E. compute node features a dedicated high quality out-of-band monitoring device, namely a Dwarf in a Giant (DiG) [12]. DiG connects to the inlet power supply of the node with a dedicated power sensor, and to the the node's telemetry through dedicated AMESTER out-of-band commands [17]. The DiG provides an accurate and fine-grain sampling of performance, power and energy consumption, it is completely out-of-band and can be deployed in any hardware architecture/large-scale datacenter at a low cost. It supports (i) fine-grained power monitoring up to 20 μs; (ii) below 1 % of uncertainty on power measurements and (iii) high-precision time-stamping (sub-microsecond), with no impact on the computing resources. High frequency power samples are averaged internally in the DiG to 1 ms and to 1 s to reduce the amount of data transmitted. AMESTER data are sampled every 10 s. AMESTER values are sampled at higher frequency and represent average of the sample interval.

This metric is used for fine grained tuning presented in this paper.

3.2 Addition Power Sensors on D.A.V.I.D.E. System

These metrics are monitored by the On Chip Controller (OCC) and exported to Examon either through Intelligent Platform Management Interface (IPMI) or through the DiG by mean of custom AMESTER commands. In the second case the measurements are average values in the reading interval. These metrics are sampled on significantly lower sampling rate, one sample every 10 s, but provide more insight on intra-node power consumption. These are: FANs, GPUs, CPU#0, CPU#1, DRAMs, Centaur chips, PCIEs power consumption.

3.3 Examon and Kairos DB

On D.A.V.I.D.E. the monitored metrics are collected by Examon[18, 19]. Examon, combines distributed monitoring agents which independently collects power, performance and usage metrics from the different system's components (i.e. Computing Element, IPMI, DiG, Liquid Cooling, PSUs and Job Scheduler) with a lightweight and standard communication protocol

and big data technology for storage and processing in real-time the acquired data. Data are labeled and structured to give flexibility on the analysis which can be conducted on that. Examon uses the MQTT protocol for exchanging data which has emerged as a lightweight and scalable data-exchange protocol. It is characterized by a small packet header composed of only two bytes. It implements the publisher-subscriber communication protocol and allows to trade-off latency, overhead and transmission quality by mean of three QoS levels [20]. Data are aggregated through information broker and inserted in a scalable database as time-series. This is done by combining KairosDB and Cassandra Database [21]. KairosDB is used to provide a time-series abstraction in the Cassandra storage layer. These services (MQTT broker, KairosDB and Cassandra) runs on management node of the D.A.V.I.D.E.

The KairosDB is single point of contact between MERIC and the entire energy monitoring system which significantly simplifies the implementation of MERIC Examon plug-in.

3.4 C++ Client for KairosDB

In order to develop a MERIC plug-in for Examon system, we had to first implement a C++ client for KairosDB that is able to send queries to the database and process the JSON formatted responses. The client uses two external libraries: (i) REST client for C++ [22] and (ii) JSON for Modern C++ [23].

The key parts of the client is shown in Listing 1. The code shows how to connect to KairosDB, and compile query that requests a set of samples with corresponding time stamps for given time range. The time range is defined by *start_t* and *stop_t* which are given in Unix time in milliseconds. In addition user has to specify the name of the node, here *davide1* and the name of the metric he wants to query, here *power* (Please note: the bbb_pub plugin contains only single metric, for other metrics from OCC or IPMI the query needs to be updated, see Section 3.2.).

These parameters are then passed to the *get_samples(...)* function which will fill the provided two vectors with time-stamps and power samples in the given range. These arrays are then processed by the D.A.V.I.D.E. plug-in for MERIC as described in the following section.

3.4.1 Query Processing Performance

We have implemented and evaluated the performance for two different queries:

- *Type 1:* Get average power value of power samples for particular sensor in particular node in particular time range
- *Type 2:* Get all samples for particular sensor in particular node in particular time range.

In case of Type 1 query the calculation of the average value (P_{avg}) is done by the KairosDB and only single values is sent back to the client. In this scenario MERIC would submit this query for every call of every significant region and this situation can occur up to several thousand times during the application runtime. Therefore a low latency is essential if this approach should be used.

In case of Type 2 query, all the samples for the entire application runtime are transferred to client and the calculation of average value (P_{avg}) for every regions is done from the power samples. The main disadvantage of this approach is that if application runtime is long the

```
#include <connection.h>      // Rest client headers
#include <restclient.h>      // access to KairosDB
#include <nlohmann/json.hpp> // JSON library used to
using json = nlohmann::json; // parse JSON responses
RestClient::Connection* conn;
string url = "http://KAIROSDB_SERVER_IP_ADDR:KAIROSDB_PORT/api/v1"+
             "/datapoints/query";

void init_kairos() {
 RestClient::init();
 conn_samples = new RestClient::Connection(url);
 conn_samples->SetBasicAuth("username", "password");}

void close_kairos() { RestClient::disable(); }

void get_samples( // get samples in the time interval
     long long start_time, long long stop_time, string node_n,
     string metric, std::vector<long long> & time_samples,
     std::vector<double> & power_samples) {
 string querry = "{\"cache_time\":0,\"end_absolute\":"+
 to_string(stop_time) +",\"metrics\":[{\"group_by\":[{\"name\":"+
 "\"tag\",\"tags\":[\"node\",\"cluster\",\"org\",\"plugin\"]}],"+
 "\"name\":\"" + metric + "\",\"tags\":{\"node\":[\"" + node_n +
 "\"],\"plugin\":[\"bbb_pub\"],\"ts\":[\"1ms\"],\"chnl\":[\"data"+
 "\"],\"cluster\":[\"davide\"],\"org\":[\"e4\"]}}],\"time_zone\":"+
 "\"Europe/Rome\",\"start_absolute\":"+to_string(start_time)+"}";
 RestClient::Response r = conn_samples->post("", query);
 json j = json::parse(r.body); //parse JSON
 json v = j["/queries/0/results/0/values"_json_pointer];
 time_samples.resize(v.size()); //get time stamps
 power_samples.resize(v.size()); //get power samples
 for (int ts = 0; ts < v.size(); ts++) {
  time_samples[ts]  = (long long) v[ts][0];
  power_samples[ts] = (double)    v[ts][1];
 }
}

int main(int argc, char const *argv[]) {
 string node_n     = "davide1";     // node name
 string metric     = "power";       // metric name
 long long stop_t  = 1529332500200; // Unix time [ms]
 long long start_t = 1529332500100; // Unix time [ms]
 std::vector<long long> t_samples;  // time samples
 std::vector<double> p_samples; // power samples
 init_kairos();   // setup connection to KairosDB
 get_samples(start_t, stop_t, node_n, metric, t_samples, p_samples);
 close_kairos();  // close connection to KairosDB
 return 0;
}
```

Listing 1: A key parts of the C++ KairosDB client in MERIC

client will request large amount data to be transferred. For this type the throughput is key performance indicator.

The performance evaluation is shown in Table 1. The observations are:

- the minimal latency of Type 1 query is at least ~350 ms – in this case KairosDB is not efficient for short regions (length of several samples),
- for regions shorter than ~2 s (2048 ms) the response time is very similar for both type of queries,
- for application that runs approximately 9 minutes KairosDB needs 10 seconds to provide all samples to the client.

Based on these results we have implemented the Examon client for MERIC to transfer all samples (Type 2 query) from KairosDB to client and calculate the P_{avg} in the client. The details are described in the next section.

Interval length [ms];[min]	**Type 1** Time to get P_{avg} [ms]	**Type 2** Time to get samples [ms]
1	357	327
2	370	326
4	344	338
8	337	404
16	343	375
32	325	360
64	377	381
128	356	376
256	408	348
512	379	409
1024	379	395
2048	733	753
4096	710	822
8192	802	950
16384	765	1096
32768	867	1292
65536 ; 1.1	965	1857
131072 ; 2.2	1660	2593
262144 ; 4.4	2222	5067
524288 ; 8.7	3806	10070

Table 1: Performance evaluation of the KairosDB for Type 1 and Type 2 query for single node of D.A.V.I.D.E.

3.5 Examon plug-in for MERIC

The Examon plug-in for MERIC is designed to process time-series of power samples that are downloaded from KairosDB using our C++ client.

During the runtime of the application MERIC records beginning and end time of each significant regions (at this time it does not measure energy). This approach has minimal negative effect on the application performance and it is designed particularly for out-of-band monitoring systems such as Examon. Once the application finishes, MERIC takes the time-stamp of the beginning and the end of the *main* function and queries the KairosDB for all samples in this interval. During the decoding of the JSON response of the KairosDB the time-stamp for each power sample is checked and if a missing sample is detected its value is interpolated from surrounding samples and correctly saved into a simple Energy Readings Structure (ERS) in the MERIC. The ERS contains (i) time-stamp of the first sample, called initial time-stamp t_i and (ii) vector of power samples p_s which length is equal to the duration of the *main* function in milliseconds. This way plug-in ensures that there are no missing samples and MERIC does not need to store time-stamps of all the samples.

This improves the performance of the per region energy consumption calculation, because plug-in does not need to search an array of time-stamps to find the first and the last energy sample of a region. Instead, required power samples can be directly accessed due to fact that each power sample has been taken every 1 ms.

The energy consumption of a region, that is identified by start time-stamp t_s, end time-stamp t_e and duration $d = t_e - t_s$ in milliseconds, is than calculated using ERS, where the index of the first power sample is $t_e - t_i$ and the index of the last power sample is $t_e - t_i$. Then a mean value, $P_{avg} = 1/d \sum_{n=t_s-t_i}^{t_e-t_i} p_s(n)$, of all samples between these indexes is calculated and the energy is calculated from P_{avg} and d in seconds. This process is repeated for every region. This approach can be used for any sensor which readings are stored in the KairosDB. One just need to update the sampling frequency of the sensor.

For sensors from OCC which sampling rate is significantly slower (one sample every 10 seconds) , an interpolation of time series must be used. This is described in the next section.

3.5.1 Interpolation for Short Regions

Interpolation is used for energy evaluation of regions which length is close to the sampling period of the Examon system. We have manually set the threshold to use interpolation if number of samples is smaller than 10.

The plug-in supports two interpolation techniques: (1) linear interpolation and (2) spline interpolation, see Figure 1. Both techniques are implemented in the cubic spline interpolation library [24]. The complexity of spline interpolation is $O(N)$ to generate a spline and $O(log(N))$ to evaluate spline at a single point, where N is the number of input data points. In order to eliminate negative effect of interpolation on the boundaries we use extra sample before the start of a region and after the end of a region to generate the spline. The up-sampling ration is set 10, i.e. we get the 10 times more samples for the region evaluation. From these samples an average value of input power is calculated and the total region energy consumption is calculated using this average value and region duration.

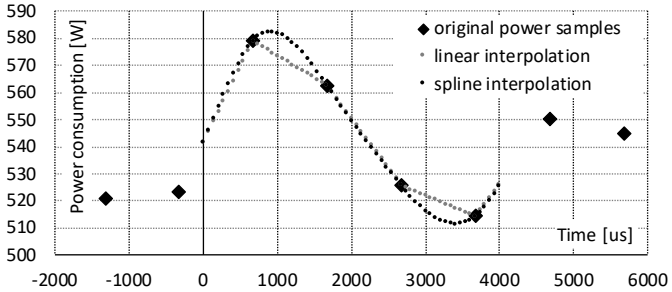


Figure 1: Example of two interpolation methods implemented in the Examon plugin for MERIC.

4 Benchmarking

4.1 ESPRESO FEM library description

The ESPRESO FEM [25][26] library is a combination of Finite Element (FEM) and Boundary Element (BEM) tools and TFETI/HTFETI solvers. It supports FEM and BEM (using the BEM4I library [27]) discretization for Advection-diffusion equation, Stokes flow and Structural mechanics. Real engineering problems are imported from Ansys Workbench or OpenFOAM. A C API allows ESPRESO FEM to be used as a solver library for third party applications. It has been used for integration with CSC ELMER. For large scale tests, the library also contains a multiblock benchmark generator.

The ESPRESO solver is a parallel linear solver which includes a highly efficient MPI communication layer [28] designed for massively parallel machines with thousands of compute nodes. The parallelization inside a node is done using OpenMP. Three versions of the solver are developed:

- *ESPRESO CPU* uses sparse matrices and sparse direct solvers to process the system matrices. This version also supports the second generation (KNL) of Intel Xeon Phi many-core processors.

- *ESPRESO MIC* is an Intel Xeon Phi accelerated version designed for the first generation (KNC) of Intel Xeon Phi accelerator attached to PCI-Express bus and using an offloading approach;

- *ESPRESO GPU* is a GPU accelerated version which supports dense structures using the LSC method and sparse structures using cuSolver. This paper evaluates several methods that executes key kernels of the LSC method on GPU in order to select the most optimal one for different scenarios (different problem size, different decomposition, etc.).

ESPRESO FEM has been manually instrumented for the purposes of the READEX project, see Figure 2 for the diagram, call tree, of the annotated regions. We evaluate only the leaf regions for dynamic tuning which are marked in red. Some of these regions have a very short runtime, bellow 1 % of total runtime, so we do not use them for this paper.

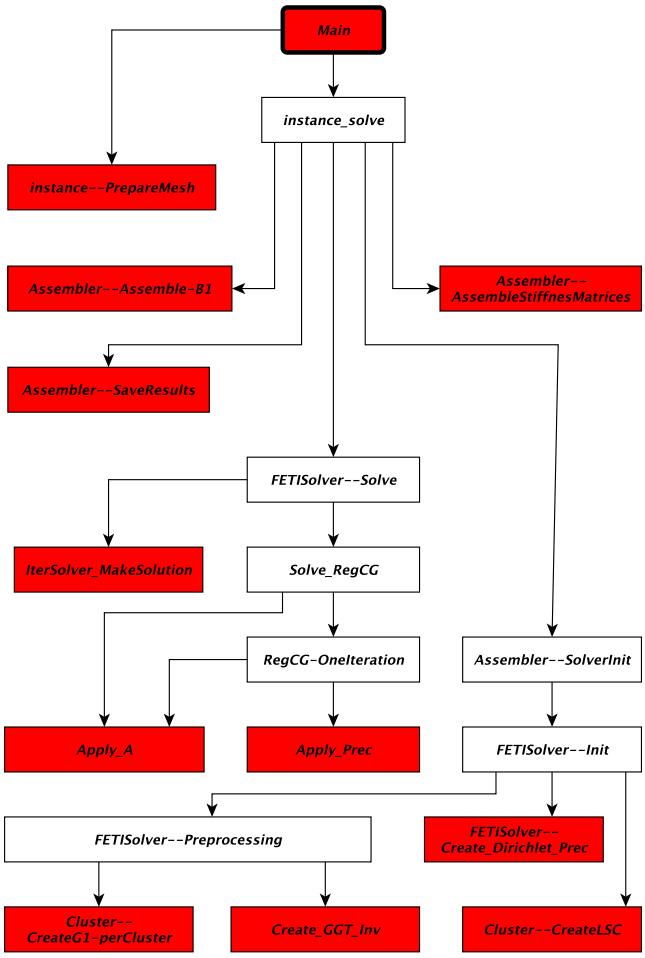


Figure 2: Diagram of significant regions on the ESPRESO FEM library. The red regions are used for dynamic tuning.

4.2 Benchmark setup

For our experiments the ESPRESO FEM has been built with the following compilers:

- MPICH version 3.2
- GCC compiler version 7.1.0

In addition following numerical libraries has been used:

- PARDISO sparse direct solver version 6.1
- reference LAPACK version 3.8.0 from NETLIB compiled with optimized IBM ESSL BLASS
- IBM ESSL library version 5.4 (sequential version)

5 Results

In this section we present results for both static and dynamic tuning of the selected regions.

5.1 Static tuning

We first evaluate the potential savings for the static tuning, i.e. one configuration is used for the entire application runtime. The summary of the results is shown in Table 2.

The static savings are defined against the *default configuration*, which is:

- 1 hyper-thread per core
- 4.02 GHz CPU core frequency (the highest available)

In the HPC environment using hyper-threads is not default behavior because of less floating point units, than available computational threads, which does not speed-up the application but may even cause slowing down of the compute bound parts of the application. Also when application cause cache pressure, with hyper-threading it may generate more cache-miss situations [29].

By finding the static optimum ESPRESO FEM can save 18.7 % of runtime and 21.6 % of energy. If we use the optimal settings for energy the runtime is also reduced by 9.8 %. Since most of the application is compute bound, despite 8 hyper-threads are available, the best static configuration for time uses 4 and for energy only 2 of them. For static tuning the difference between 2 and 4 hyper-threads is not very significant, but this is not the case for particular regions as described in the next section.

	Default settings	**Default values**	**Static optimum**	**Static savings**
Tuning for energy consumption	1 h-thread 4.02 GHz	97.3 kJ	2 h-thread 3.22 GHz	21038 J 21.6%
Tuning for runtime	1 h-thread 4.02 GHz	140.8 s	4 h-thread 4.02 GHz	26.3 s 18.7 %
Runtime for optimal energy configuration	-10.8 s 90.2 % of runtime with default settings			

Table 2: Evaluation of the static saving of the ESPRESO solver with respect to time and energy.

5.2 Dynamic tuning

The summary of results for dynamic tuning of the longest regions is presented in Table 3. From global point of view when tuning to reduce application energy consumption all the regions but one (*Apply_Prec*) have optimal frequency equal to 3.22 GHz. For these regions the dynamic savings come from tuning of hyper-threads only.

The *Apply_A* region call forward and backward substitution of the PARDISO sparse direct solver. From its analysis we observe:

- The runtime for 4 and 8 hyper-threads is identical but since 8 hyper-threads utilize more hardware units the energy consumption is higher by 1.6% for the optimal frequency 3.2 GHz.
- The difference in energy consumption between static optimum, 2 hyper-threads, and dynamic optimum, 8 hyper-threads, is 1.47 kJ or 10.7 %.

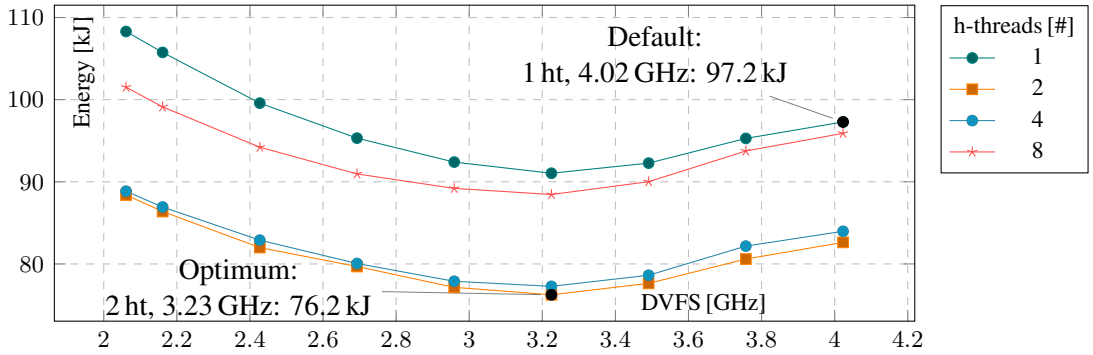
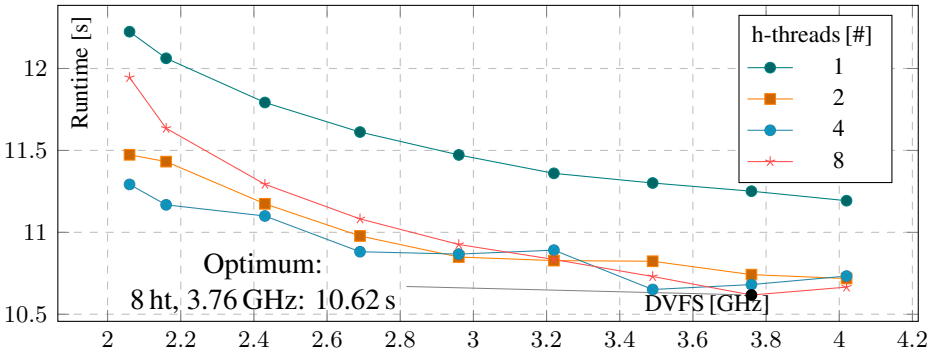


Figure 3: Evaluation of the effect of DVFS and hyper-threading on runtime and energy consumption of the Main region of ESPRESO FEM library.

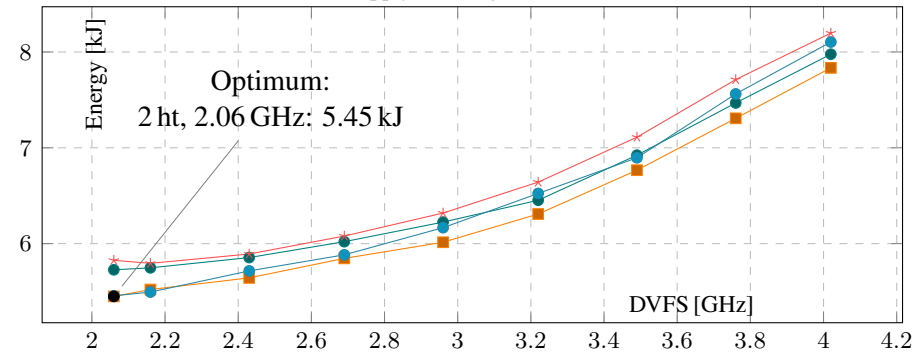
Region name	**Best dynamic configuration**	**Energy**	**Time**
		total [kJ]; savings [kJ]	**total [s]; savings [s]**
% of default application runtime	for energy / for time	default configuration (no tuning)	
		static tuning for energy	
		dynamic tuning for energy	
		static tuning for time	
		dynamic tuning for time	
Main 100 % (static optimum)	2 h-thread 3.22 GHz / 4 h-thread 4.02 GHz	97.3; – (–)	140.8; – (–)
		76.3; 21.0 (21.6 %)	114.5; 26.3 (18.7 %)
		–	–
		84.0; 13.3 (13.7 %)	137.1; 3.7 (2.6 %)
		–	–
Apply_A 18.5 %	4 h-thread 3.22 GHz / 4 h-thread 4.02 GHz	18.1	26.1
		13.7; 4.4 (24.3 %)	24.2; 1.9 (7.3 %)
		12.3; 5.8 (32.0 %)	20.4; 5.7 (21.8, %)
		13.3; 4.8 (26.5 %)	17.4; 8.7 (33.3 %)
		13.3; 4.8 (26.5 %)	**17.4; 8.7 (33.3 %)**
Apply_Prec 8.5 %	2 h-thread 2.06 GHz / 8 h-thread 4.02 GHz	7.98	11.2
		6.31; 1.67 (26.4 %)	10.8; 0.4 (3.5 %)
		5.45; 2.53 (31.7 %)	11.5; -0.3 (-2.6)
		8.10; -0.12 (-1.5 %)	10.7; 0.5 (4.5 %)
		8.19; -0.21 (-2.6 %)	**10.6; 0.6 (5.4 %)**
Assemble Stiffnes Matrices 10.2 %	4 h-thread 3.22 GHz / 4 h-thread 4.02 GHz	9.64	14.4
		6.66; 2.98 (30.9 %)	12.3; 2.1 (14.6 %)
		5.64; 4.00 (41.5 %)	9.86; 4.54 (31.5 %)
		6.07; 3.57 (37.0 %)	8.34; 6.06 (42.1 %)
		6.07; 3.57 (37.0 %)	**8.34; 6.06 (42.1 %)**
Create Dirichlet Prec 37.8 %	2 h-thread 3.22 GHz / 2 h-thread 4.02 GHz	37.4	53.3
		30.1; 7.3 (19.5 %)	53.4; -0.1 (-0.2 %)
		30.1; 7.3 (19.5 %)	53.4; -0.1 (-0.2 %)
		36.4; 0.7 (2.7 %)	49.3; 4.0 (7.5 %)
		32.1; 5.3 (14.2 %)	**44.4; 8.9 (16.7 %)**
Total consumption in:			
- default configuration:		110.3 s; 76.7 kJ	
- dynamic tuning for time:		84.2 s; 62.1 kJ (+10.1 % of energy)	
- dynamic tuning for energy:		99.4 s; 55.8 kJ (+15.3 % of time)	

Table 3: Evaluation of dynamic saving for the most significant regions of the application. The significant regions cover 78.1 % of the default runtime of the ESPRESO FEM library. All remaining parts are tuned using best static configuration.

The *Apply_Prec* region calls mainly the DGEMV function of the ESSL library, so this a strictly memory bound region. This is clearly visible from Figure 4 where we can see that the lowest CPU frequency is the most efficient while the runtime changes are very small with respect to DVFS tuning. To run this region at its fastest configuration cost 33.4 % more energy while shortening the runtime by 7.8 % only.



(a) Apply_Prec region runtime



(b) Apply_Prec region energy consumption

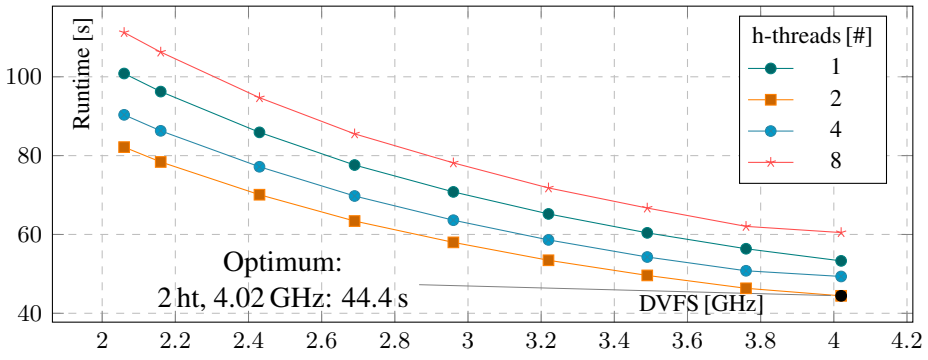
Figure 4: Energy consumption and runtime in relation to specified CPU frequency and number of active hyper-threads of Apply_Prec region.

The *Assemble_Stiffness_Matrices* region calculates the element matrices and combines these into final sparse CSR stiffness matrix for every domain. For this region the most energy efficient configuration can save up to 41.5 % against the default settings and 31.5 % of runtime. The best runtime configuration can save up to 42.1 % of runtime and 37.0 % of energy against default settings.

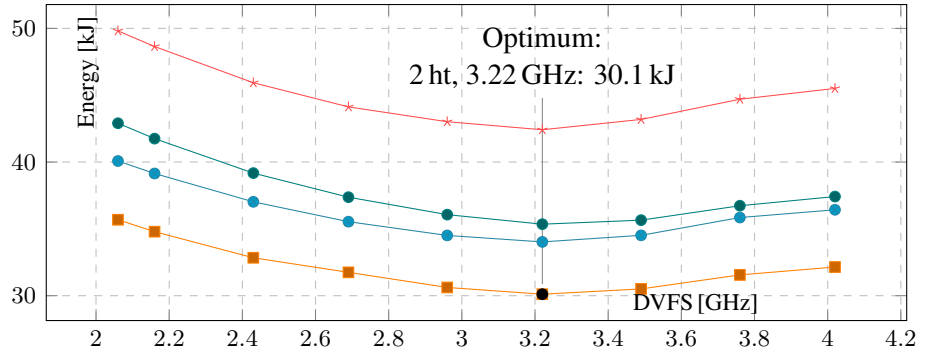
The *Create_Dirichlet_Prec* at first permutates the stiffness matrix and then calculate Schur complement using PARDISO. In this case however the size of the final Schur complement is significantly higher that in the previous region. As this region runs 37.8 % of the application runtime it mostly defines optimal its static configuration. There the static and dynamic configuration is the same. As shown at the Figure 5 we can save 19.5 % of energy and -0.2 % of runtime for optimal energy configuration or 16.7 % of runtime and 14.2 % of energy for the optimal time configuration.

All these regions covers 78.1 % of the runtime in the default settings, which means 110 s while consuming 76.7 kJ of energy. If we use the configuration for the best runtime we can reduce the runtime to 84.2 s (23.7 % time savings) and energy to 62.1 kJ (19.0 % energy savings),

on the other hand the configuration for the best energy consumption results in runtime of 99.4 s (9.9 % time savings) and the energy consumption will be 55.8 kJ (27.3 % energy savings).



(a) Create_Dirichlet_Prec region runtime



(b) Create_Dirichlet_Prec region energy consumption

Figure 5: Energy consumption and runtime in relation to specified CPU frequency and number of active hyper-threads of Create_Dirichlet_Prec region.

When we compare these two optimal settings we can observe these trade-offs:

- **against default settings we can save up to 23.7 % of runtime** for these regions,
- **against default settings we can save up to 27.3 % of energy** for these regions,
- **the fastest runtime consumes 10.1 % more energy than the most energy efficient run**, and
- **the most energy efficient run requires extra 15.3 % of runtime than the fastest run**.

6 Conclusions

In this paper we have presented application of the READEX approach using the MERIC library for HPC applications tuning from energy consumption point of view. To support new architecture MERIC has been extended about the possibility to tune IBM Power8+ system parameters and implemented an interface for accessing power samples from the DiG, stand alone, high frequency energy measurement system, that is available at UNIBO D.A.V.I.D.E. system.

As a benchmark we have presented fully fledged ESPRESO FEM library, with several regions showing different resources requirements, which implies possibility to dynamically tune the parameters and save an energy accordingly.

For ESPRESO FEM we were able to save up to 27.3 % of energy in compare to default system settings, and at the same time it results in 9.9 % shorter runtime. We were also able to tune the application for the runtime, and reached 23.7 % savings.

Most of the saving both time and energy comes from the applying optimal number of hyper-threads for each region of the application. It is not usual to use hyper-threads in HPC, since it may even cause application slowdown in the phases of an application that are computational or memory bound. In case of using hyper-threading it is also always important to take care about the thread placement and their migration even more than in case of usual dual socket systems without hyper-threading.

Acknowledgement

The work has been performed under the Project HPC-EUROPA3 (INFRAIA-2016-1-730897), with the support of the EC Research Innovation Action under the H2020 Programme; in particular, the author gratefully acknowledges the support of University of Bologna, DEIS and the computer resources and technical support provided by CINECA.

This work was supported by the READEX project - the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671657.

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project „IT4Innovations excellence in science - LQ1602“ and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project „IT4Innovations National Supercomputing Center – LM2015070“. This work was supported by the ESF in „Science without borders“ project, reg. nr. CZ.02.2.69/0.0./0.0./16_027/0008463 within the Operational Programme Research, Development and Education. This work was partially supported by the SGC grant No. SP2018/134 "Development of tools for energy-efficient HPC applications", VSB - Technical University of Ostrava, Czech Republic.

References

[1] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, J. Dongarra, "Investigating power capping toward energy-efficient scientific applications", *Concurrency and Computation: Practice and Experience*, page e4485, URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4485`.

[2] F. Fraternali, A. Bartolini, C. Cavazzoni, L. Benini, "Quantifying the Impact of Variability and Heterogeneity on the Energy Efficiency for a Next-Generation Ultra-Green Supercomputer", *IEEE Transactions on Parallel and Distributed Systems*, 29(7): 1575–1588, July 2018, ISSN 1045-9219.

[3] C. Bonati, E. Calore, M. D'Elia, M. Mesiti, F. Negro, S.F. Schifano, G. Silvi, R. Tripiccione, "Early Experience on Running OpenStaPLE on DAVIDE", in R. Yokota, M. Weiland, J. Shalf, S. Alam (Editors), *High Performance Computing*, Lecture Notes in Computer Science, pages 387–401. Springer International Publishing, 2018, ISBN 978-3-030-02465-9.

[4] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, S. Jana, "Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions", in *ISC*, pages 394–412. Springer International Publishing, Cham, 2017, ISBN 978-3-319-58667-0.

[5] B. Rountree, D.K. Lowenthal, B.R. de Supinski, M. Schulz, V.W. Freeh, T.K. Bletsch, "Adagio: making DVS practical for complex HPC applications", ICS '09, pages 460–469. ACM, New York, NY, USA, 2009, ISBN 978-1-60558-498-0, URL http://doi.acm.org/10.1145/1542275.1542340.

[6] READEX, "Horizon 2020 READEX Project", https://www.readex.eu, 2018.

[7] J. Schuchart, M. Gerndt, P.G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U.S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg, W.E. Nagel, "The READEX formalism for automatic tuning for energy efficiency", *Computing*, 99(8): 727–745, Aug 01, 2017, ISSN 1436-5057, URL https://doi.org/10.1007/s00607-016-0532-7.

[8] M. Hähnel, B. Döbel, M. Völp, H. Härtig, "Measuring Energy Consumption for Short Code Paths Using RAPL", *SIGMETRICS Perform. Eval. Rev.*, 40(3): 13–17, Jan. 2012, ISSN 0163-5999, URL http://doi.acm.org/10.1145/2425248.2425252.

[9] AMD, "Preliminary Processor Programming Reference (PPR) for AMD Family 17h Models 00h-0Fh Processors", https://www.amd.com/system/files/TechDocs/54945_PPR_Family_17h_Models_00h-0Fh.pdf, 2018.

[10] NVIDIA, "Nvidia NVML API Reference Manual", 2019.

[11] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W.E. Nagel, M. Simon, Y. Georgiou, "HDEEM: High Definition Energy Efficiency Monitoring", in *2014 Energy Efficient Supercomputing Workshop*, pages 1–10, Nov 2014.

[12] A. Libri, A. Bartolini, L. Benini, "DiG: Enabling Out-of-Band Scalable High-Resolution Monitoring for Data-Center Analytics, Automation and Control", Nov. 2018, URL https://www.research-collection.ethz.ch/handle/20.500.11850/306925.

[13] TOP500.org, "TOP500: November 2018", https://www.top500.org/lists/2018/11/, 2018.

[14] O. Vysocky, M. Beseda, L. Riha, J. Zapletal, V. Nikl, M. Lysaght, V. Kannan, "Evaluation of the HPC Applications Dynamic Behavior in Terms of Energy Consumption

", in P. Ivanyi, B.H.V. Topping, G. Varady (Editors), *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. Civil-Comp Press, Stirlingshire, UK, 2017, Paper 3, 2017. doi:10.4203/ccp.111.3.

[15] W.A. Ahmad, A. Bartolini, F. Beneventi, L. Benini, A. Borghesi, M. Cicala, P. Forestieri, C. Gianfreda, D. Gregori, A. Libri, F. Spiga, S. Tinti, "Design of an Energy Aware Petaflops Class High Performance Cluster Based on Power Architecture", in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 964–973, May 2017.

[16] D. Brodowski, "Linux CPUFreq", URL `https://www.kernel.org/doc/Documentation/cpu-freq/index.txt`.

[17] T. Rosedah, C. Lefurgy, M. Broyles, "Measuring and Managing Energy in OpenPOWER", in M. Taufer, B. Mohr, J.M. Kunkel (Editors), *High Performance Computing*, pages 255–267. Springer International Publishing, Cham, 2016, ISBN 978-3-319-46079-6.

[18] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, D. Gregori, S. Tinti, C. Gianfreda, P. Altoè, "The D.A.V.I.D.E. Big-data-powered Fine-grain Power and Performance Monitoring Support", in *Proceedings of the 15th ACM International Conference on Computing Frontiers*, CF '18, pages 303–308. ACM, New York, NY, USA, 2018, ISBN 978-1-4503-5761-6, URL `http://doi.acm.org/10.1145/3203217.3205863`.

[19] F. Beneventi, A. Bartolini, C. Cavazzoni, L. Benini, "Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools", in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1038–1043, March 2017, ISSN 1558-1101.

[20] U. Hunkeler, H.L. Truong, A. Stanford-Clark, "MQTT-S: A publish/subscribe protocol for Wireless Sensor Networks", in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pages 791–798, Jan 2008.

[21] KairosDB, "KairosDB: Fast Time Series Database on Cassandra", `https://kairosdb.github.io/`, 2015-2019.

[22] D. Schauenberg, "REST client for C++", `http://code.mrtazz.com/restclient-cpp`, 2018.

[23] N. Lohmann, "JSON for Modern C++", `https://github.com/nlohmann/json`, 2019.

[24] T. Kluge, "Cubic Spline interpolation in C++", `http://kluge.in-chemnitz.de/opensource/spline/`, 2016.

[25] "ESPRESO - Exascale Parallel FETI Solver, http://espreso.it4i.cz", URL `http://espreso.it4i.cz`.

[26] L. Riha, T. Brzobohaty, A. Markopoulos, O. Meca, T. Kozubek, "Massively Parallel Hybrid Total FETI (HTFETI) Solver", in *Platform for Advanced Scientific Computing Conference, PASC*. ACM, 2016, ISBN 978-1-4503-4126-4/16/06,

URL http://espreso.it4i.cz/wp-content/uploads/2016/05/RIHA_PASC2016.pdf.

[27] J. Zapletal, M. Merta, L. Malý, "Boundary element quadrature schemes for multi- and many-core architectures", *Computers & Mathematics with Applications*, 74(1): 157 – 173, 2017, ISSN 0898-1221, URL http://www.sciencedirect.com/science/article/pii/S0898122117300482, 5th European Seminar on Computing ESCO 2016.

[28] L. Riha, T. Brzobohaty, A. Markopoulos, M. Jarosova¡, T. Kozubek, D. Horak, V. Hapla, "Implementation of the efficient communication layer for the highly parallel total FETI and hybrid total FETI solvers", *Parallel Computing*, 2016.

[29] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, R. Rooholamini, "An Empirical Study of Hyper-Threading in High Performance Computing Clusters", 2002.