# Energy Optimization and Analysis with EAR

1st Julita Corbalan
*Computer Science Dpt. (BSC)*
*Computer Science Dpt. (UPC)*
Barcelona, Spain
julita.corbalan@bsc.es

2nd Lluis Alonso
*Computer Science Dpt. (BSC)*
Barcelona, Spain
lluis.alonso@bsc.es

3rd Jordi Aneas
*Computer Science Dpt. (BSC)*
Barcelona, Spain
jordi.aneas@bsc.es

4th Luigi Brochard
*Energy Aware Solutions (EAS)*
Paris, France
luigi.brochard@eas4dc.com

***Abstract*—EAR is an energy management framework which offers three main services: energy accounting, energy control, and energy optimization. The latter is done through the EAR runtime library (EARL). EARL is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control. EARL optimises energy by selecting the *optimal* CPU frequency, based on the energy policy selected and application runtime characteristics without any application modification or user input. Currently EARL only works for MPI applications but EAR itself can still operate for non-MPI applications. It automatically (and transparently) identifies iterative regions (loops) and computes a set of metrics per iteration, *application signature*, and, together with the *system signature*, applies energy models to estimate the execution time and power for the CPU frequencies available. System signature is a set of coefficients per-node computed during EAR installation via a learning phase. Given time and power projections, EARL selects the best frequency based on policy settings.**

**This papers shows how to optimize energy using the EAR library with `min_time_to_solution` energy policy and how to analyse applications through EAR framework. Evaluation includes eight applications with different sizes and application signatures. Results show how EARL computes each application signature on the fly and applies the CPU frequency selected by the `min_time_to_solution` policy.**

***Index Terms*—Energy, System software, Optimization, Data centers**

## I. Introduction

This paper presents how to optimize energy by using EAR runtime library (EARL), one of the components of the EAR framework and the energy policy by default: `min_time_to_solution`. EARL measures a set of application metrics at runtime, the application signature, and based on energy models and policies it automatically selects the optimal frequencies for the specific run knowing EARL will select different frequencies if the application signature changes over time.

Tools like cpupower or schedulers like SLURM or PBS offer the possibility to change manually the CPU frequency of the node where the application is running, but the performance and power impacts of such change are totally out of reach of these tools. Figures 1 and 2show time, power and energy variation when running two of the applications used in this paper when varying the CPU frequency from 2.4Ghz to 1.8Ghz. The first application is doing an intensive CPU utilization and the second one exhibits an exhaustive utilization of main memory. Therefore the first application will report significant power savings when reducing its CPU frequency but at the expense of an increased time, while the second application will report significant power savings with little performance penalty leading to significant energy reduction. That's the analysis EARL is doing automatically and transparently to make the right decision depending on the application characteristics gathered by the application signature.

All the applications used in this paper present different behaviours when changing submission parameters such as number of MPI processes, threads, or input files, creating the need of a runtime solution to identify these characteristics. EARL is able to do that without any user input or application modification. Moreover, the rest of EAR's components offer a complete energy management framework, offering a simple way to apply an energy optimization policy and getting performance metrics for application evaluation and analysis.

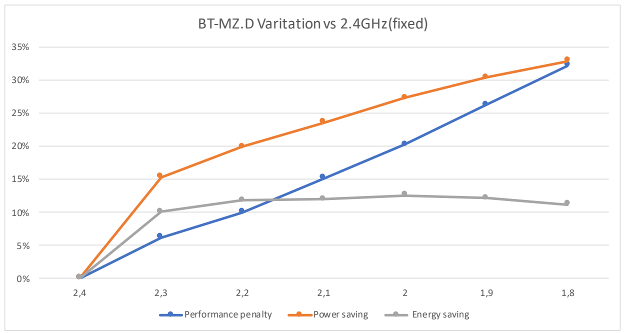

Fig. 1. Impact on BT-MZ.D when varying CPU frequency

This paper will evaluate and analyse a set of real applications when using the `min_time_to_solution` energy policy. We will show how simple it is to apply an energy policy and analyse application performance and power metrics when using EAR. Additional metrics which are valuable for energy efficiency classification, such as CPI or GFlops, are also reported, making it easy both for normal users and *sysadmins*

his work has been mostly funded by the BSC-Lenovo collaboration agreement and partially by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through PID2019-107255GB project, by the Generalitat de Catalunya (2017-SGR-1414)

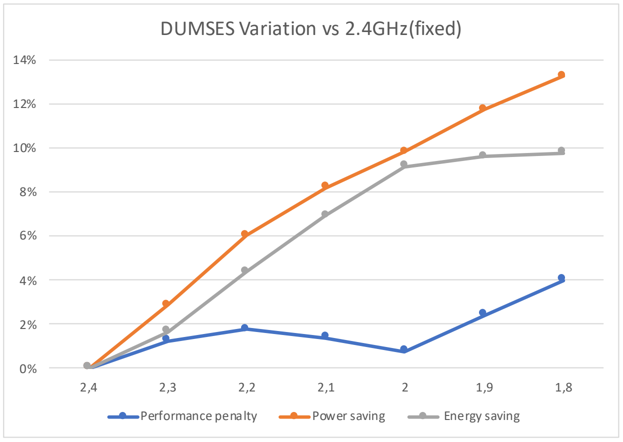

Fig. 2. Impact on DUMSES when varying CPU frequency

to do application analysis and workload characterization.

The evaluation section VII will present results on eight applications showing how GFlops/Watt is improved 9% in average when using EARL+`min_time_to_solution`.

The rest of the paper is organised as follows: Section II describes EAR as energy management framework. Section III presents EAR library in detail. Section IV presents the application signature and energy models used by EAR. Section VI presents the min_time_to_solution policy. Section VII shows how to run and get performance metrics with EAR. SectionVIII evaluates EARL in terms of performance and energy savings. Section IX presents the related work. And finally, section X presents conclusions and future work.

## II. EAR OVERVIEW

EAR offers three main services concerning energy management as shown on Figure 3: Monitoring, Accounting, Control and Optimisation. Energy optimisation is offered by EARL and only enabled for MPI or hybrid MPI+OpenMP applications.

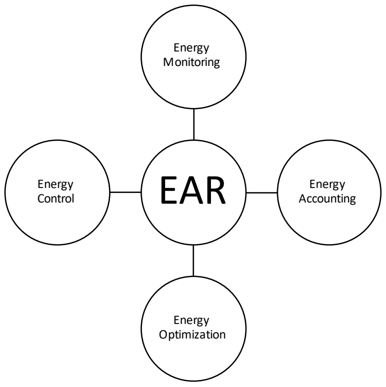

Fig. 3. Energy capabilities provided by EAR

Fig. 4 shows the EAR components and main interactions between them: EAR Daemon, EARL, EARDBD, EARGMD and EAR submission plugin (only SLURM plugin is available).

- Node monitoring and Job accounting is provided by the EAR Daemon, a Linux service running in compute nodes (one instance per compute node), the SLURM SPANK plugin (offering the basic EAR API to automatically identify new jobs (start/end of new JobIDs.StepIDs in SLURM are automatically detected), and the EAR DB manager. Multiple EAR DB managers can be used in the system to increase EAR scalability. The EAR DB manager offers replication automatically so no special hardware is needed to provide high availability.
- Optimization and per-application control is provided by EARL and deeply described in section III.
- Global control is offered by the EAR Global Manager (EARMG) and is out of the scope of this paper.

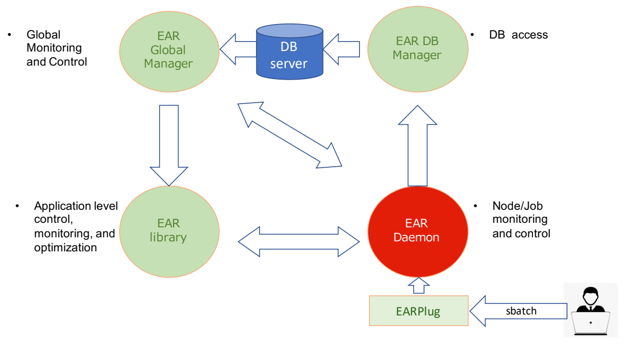
Global Monitoring and Control
EAR Global Manager
DB server
EAR DB Manager
DB access
Application level control, monitoring, and optimization
EAR library
EAR Daemon
Node/Job monitoring and control
EARPlug
sbatch

Fig. 4. EAR components

## III. EAR RUNTIME LIBRARY

EARL is a dynamic, transparent, and lightweight runtime library that provides energy optimisation and control. EARL identifies on the fly the application iterative structure existing in many parallel codes. It does it without any user intervention (without hints, code marks, tags etc).

Figure 5 shows EARL's lifecycle with its main components. One of the EAR research contributions is the DynAIS algorithm which detects outer loops on the fly without application modification which is quite unique compared with other runtime solutions. EARL includes two energy policies by default: `min_time_to_solution` targeted at improving the performance by increased frequency starting from a lower than nominal frequency and `min_energy_to_solution`, targeted at saving energy by reducing frequency starting from nominal.

Once DynAIS identifies the iterative structure of the application, EARL computes the application signature used by energy model and energy policy. The energy model and energy policy by default are specified in the ear configuration by the system administrator but it is possible to select a different policy than the default by using the `--ear-policy=policy_name` when submitting the job.

Fig. 5 shows the main internal EARL phases: (1) MPI interception, (2) Loop detection through DynAIS, (3) Application Signature computation, (4) Time and power models projections, and (5) Energy policy execution (using these projections).

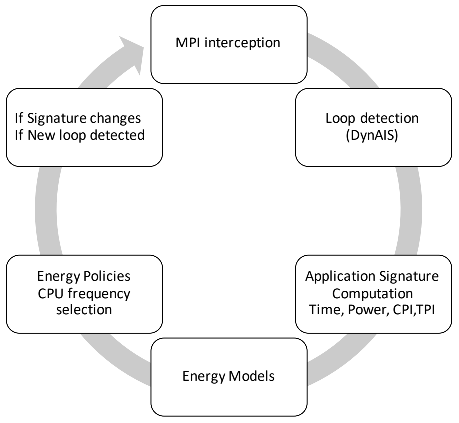
MPI interception
Loop detection (DynAIS)
Application Signature Computation Time, Power, CPI,TPI
Energy Models
Energy Policies CPU frequency selection
If Signature changes If New loop detected

Fig. 5. EARL internals

1) **MPI interception**, the standard MPI profiling in interface (PMPI)+LD_PRELOAD. EAR offers MPI symbols which are intercepted when pre-loading EARL before MPI library.
2) **DynAIS**, which dynamically (and transparently) detects the iterative regions (outer loops) of MPI applications (or hybrid MPI+OpenMP). EARL generates a runtime sequence of events based on MPI calls (together with its arguments) which is the input for DynAIS. DynAIS compares each event with the last N[1] events and returns the *state* corresponding to this event.
3) Based on DynAIS states, EARL computes the **application signature**, that uniquely identifies the application dynamic behaviour (which depends on specific nodes, input data, configuration, etc).
4) Application signature is part of the input for the **energy models**. The energy model predicts the time and power for each CPU frequency available in the node
5) Energy projections are used by the **energy policy** to select the optimal CPU frequency. Since EARL is aware of the application structure, it self-evaluates the frequency selected, and reverts it (or re-applies the policy) if needed. Moreover as EARL is aware of any change in the application by either detecting a new loop or by a change in its signature, it will re-apply the energy policy with the new context and metrics. We refer to this default mode as *DynAIS mode*

If the application is not iterative, DynAIS will not detect loops and EARL will not be able to apply steps 3 to 5. To support these cases EARL dynamically detects this situation and executes steps 3 to 5 every N seconds (configurable). We refer to this mode as *Periodic mode*.

## IV. EAR RUNTIME LIBRARY OVERHEAD AND SCALABILITY

EARL main source of potential overhead is the DynAIS algorithm since it's applied for each MPI call. Because of that, we highly optimized it using AVX512 instructions. The cost to process each MPI call has been measured at 0,1 usecs. Even though it's small, it could be a problem for application doing an intense number of MPI calls per second. For example, SUSPENSE [7], one of the applications used in the evaluation, reports 188.460 MPI calls per second causing an 1.8% overhead. Even though SUSPENSE is an extreme use case, one of the acceptance criteria to install in SuperMUC-NG [37] was to report less than 1% of overhead. To address the applications where the number of MPI calls per second is extremely high, we introduced an internal mechanism to limit the overhead. EARL computes the overhead due to MPI calls based on the number of MPI calls per second measured during the signature computation. If the estimated overhead is greater than 0.5%, DynAIS is disabled and the last loop detected by DynAIS is used as reference until a signature change is detected. If none is detected, EARL assumes there is no need to reapply DynAIS algorithm. This mechanism is dynamic allowing DynAIS to be turned OFF and ON several times during the application execution.

EAR library by itself does not use global synchronizations. Synchronizations at application level only happen when a function or a policy needs it. For example, there is no global synchronization with the min_time_to solution policy but it is used by the application power capping policy which is under development

## V. APPLICATION SIGNATURE AND ENERGY MODELS

The application signature, describing application dynamic behaviour, includes: iteration time in seconds (**Time**), average DC node power (**Power**), main memory transactions per instructions (**TPI**), and cycles per instructions (**CPI**). Application signature together with the System signature are the inputs for the energy models. Application signature identifies applications metrics influencing energy variation as a function of the frequency.

EARL default energy model uses equations proposed in [29] and [30] and evaluated in [8]. The fact that EARL uses DC node power based on DC node energy and not only CPU energy is one of the differences between EARL metrics and other research works which focus only on CPU energy consumption.

System signature is computed per node during EAR installation via a learning phase. It is a set of six per-node matrices (A..F) where each of the (i,j) values are used to project time and power from $Freq_i$ to $Freq_j$. During the learning phase a set of pre-selected kernels are executed with different frequencies and the (A..F) coefficients are computed by a least square method. Kernels have been selected to cover a wide range of application characteristics, from CPU intensive applications to memory intensive ones. Currently, the learning phase is using the following applications: BT-MZ.C, SP-MZ.C, LU-MZ.C , EP.D, LU.C [9], and UA [10] from the NASPB benchmarks, DGEMM [11], and STREAM [12]. Equations (1),(2) and (3) show the default energy model used by EAR. Since this model

[1]N is part if the DynAIS configuration and it is called the Dynais window size

depends on the system and is critical to EAR accuracy, it can only be modified by the *sysadmin* in EAR's configuration.

$$Power(f) = A * Power(f_{\text{ref}}) * B * TPI(f_{\text{ref}}) + C \quad (1)$$

$$CPI(f) = D * CPI(f_{\text{ref}}) + E * TPI(f_{\text{ref}}) + F \quad (2)$$

$$Time(f) = Time(f_{\text{ref}}) * (CPI(f))/(CPI(f_{\text{ref}})) * f_{\text{ref}}/f_{\text{n}} \quad (3)$$

## VI. EARL ENERGY OPTIMIZATION POLICIES: MINIMIZE TIME TO SOLUTION

EARL has three energy policies: monitoring, min_time_to_solution and min_energy_to_solution. The monitoring policy collects and reports information on the system and the applications but does not perform any energy optimization. We focus here one the min_time_to_solution policy. The minimize time to solution policy runs applications at a frequency lower than nominal (the default frequency) and increases the frequency if the ratio of performance benefit vs frequency increase is greater than a given threshold ratio set by the sysadmin. In other words, this policy will increase the frequency of applications in which performance scales some how with frequency to justify an extra power consumption. For example, if `min_ratio=0.75`, EAR will not set upper frequencies if the ratio between performance gain and frequency gain do not improve at least 75% `(PerfGain >= FreqGain* min_ratio)`(4)(5).

$$PerfGain = (Time - Timenew)/Time \quad (4)$$

$$FreqGain = (Freqnew - Freq)/Freq \quad (5)$$

When executed with `min_time_to_solution` policy, applications start at a predefined but configurable CPU frequency lower than nominal. For example, given a system with a nominal frequency of 2.4GHz with a default frequency set to 2.0GHz, an application executed with `min_time_to_solution` will start with frequency $F_i$=2.0GHz. When the application signature is computed, EARL computes performance projection for $F_{i+1}$ together with PerfGain(4) and FreqGain(5). If PerfGain is greater or equal than `FreqGain*min_ratio`, the policy will process the next performance projection $F_{i+2}$. Otherwise, the policy will select the last frequency where the performance gain was satisfied, preventing any waste of energy.

The `min_ratio=0.75` is the policy arguments that manages how aggressive in terms of energy saving the policy is. The higher the value the higher the energy savings (but also the performance penalty).

## VII. EXECUTING WITH EAR AND GETTING PERFORMANCE METRICS

Executing an application with EARL is totally transparent on systems where EARL is enabled by default for all applications. In case EARL's usage is optional, it is as simple as setting the EAR flag (`--ear=on`) or using any of the other EAR flags to select this feature (`--ear-policy=min_time`).

For instance, the experiments used for this paper have been done using one of three equivalent approaches shown below. Option 1 is valid for Intel MPI versions older than 2019. Option 2 is valid for current Intel MPI versions. Option 3 is valid for all the Intel MPI versions as well as for other MPI libraries such as OpenMPI which does not offer a specific option to pass additional options to SLURM. All the experiments done in this paper have been done using Intel MPI version 2019.

```
 /* Option 1 : Using mpirun*/
export $MPIS=80
mpirun -l -n $MPIS  -bootstrap slurm
-bootstrap-exec-args="--ear-policy=min_time"\
 ./bt-mz.D.$MPIS

/* Option 2 : Using mpirun*/
export $MPIS=80
export I_MPI_HYDRA_BOOTSTRAP=slurm
export I_MPI_HYDRA_BOOTSTRAP_\
EXEC_EXTRA_ARGS="--ear-policy=min_time"
mpirun -l -n $MPIS ./bt-mz.D.$MPIS

/* Option 3: Using srun */
export $MPIS=80
srun -l --ntasks=$MPIS \
--ear-policy=min_time ./bt-mz.D.$MPIS
```

Once the application is completed successfully, it is possible to check its metrics by using the `eacct` command. This command accepts different flags to specify the amount of details reported, or to select the output (stdout or csv file).

TABLE I
EACCT -J 277 OUTPUT: PER JOB DEFAULT OUTPUT FOR A SPECIFIC JOB

| Job | Pol | Freq. | Time | Power | GBs | CPI | KJ | GF/W |
|---|---|---|---|---|---|---|---|---|
| 277.12 | MO | 2,32 | 371 | 321 | 8,9 | 0,66 | 1909 | 2,88 |
| 277.3 | MT | 2,14 | 380 | 282 | 8,6 | 0,67 | 1735 | 3,19 |

For example, Table I shows two rows and a subset of the columns reported with the `eacct` when executing GROMACS application. We have selected one stepid with `monitoring` and another with `min_time_to_solution` for comparison. This output can be easily loaded in any spreadsheet tool. [2] When using `-j` option, columns reported are the following ones:

- Job: Job id and Step id assigned by SLURM (SLURM_JOB_ID and SLURM_STEP_ID). eacct shows the average metrics for all the nodes executing this jobid.step except the energy, which is accumulated.
- User (not shown in Table 1): username.
- Application (not shown in Table 1): Job Name set by the user when submitting the job or executable name when not provided.
- Pol: Energy policy. NP= No energy policy. MT=min_time, MO=monitoring only.

[2]We have hidden some columns because lack of space.

- Nodes (not shown in Table 1): Number of nodes used by the jobid.stepid.
- Freq (in GHz): Average frequency of all the nodes running this jobid.stepid. Moreover, the per-node frequency used to compute the average is, in his turn, the average frequency of all the cores. It is not the CPU freq set in the governor but the effective CPU frequency computed using aperf/mperf msr registers.
- Time: jobid.stepid Execution time in seconds.
- Power: Average power of all the nodes running this jobid.stepid. In Watts.
- GBs: Average memory bandwith of all the nodes running this jobid.stepid. In GBytes/second.
- CPI: Average CPI (Cycles per Instructions) of all the nodes running this jobid.stepid.
- KJ: Accumulated energy (in Kilo Joules) of all the nodes running this jobid.stepid (eacct reports energy in Joules, we have converted to KJ to reduce the space).
- GF/W (GFlops/Watts): Total GFlops (floating point operations per second) per Watts. It's the average of the accumulated GFlops for all the nodes running this jobid.stepid and the accumulated Watts.
- MaxPower (not shown in Table 1): During the jobid.stepid execution, the power is computed periodically. This value is the maximum value detected during these periodic metrics.

With this view, we can compare the application characteristics between different executions. For instance, in Table I we can compare the GROMACS execution time when executed with `min_time_to_solution` vs `monitoring_only` and see it leads to 2% of performance penalty but also to 14% of power saving, 10% of energy saving and 10% improvement in GFlops/Watts. This view presents also the average cpu frequency for the whole application. It is an average across all the nodes and the whole execution, so differences between nodes can be hidden but it's a good indicator of EARL optimizations. If more details are needed, for instance between nodes, the `-l` flag can be set. Table II presents this output.

TABLE II
EACCT -J 277.3 -L OUTPUT: LONG FORMAT FOR A SPECIFIC JOB ID AND STEP ID

| Job | Node | Freq | Time | Power | GBs | CPI | KJ | VPI |
|---|---|---|---|---|---|---|---|---|
| 277.3 | n25 | 2,14 | 380 | 282 | 8,8 | 0,65 | 108 | 0,44 |
| 277.3 | n26 | 2,14 | 380 | 273 | 8,9 | 0,77 | 105 | 0,40 |
| 277.3 | n27 | 2,14 | 380 | 273 | 8,8 | 0,67 | 105 | 0,45 |
| 277.3 | n28 | 2,14 | 380 | 276 | 8,9 | 0,67 | 106 | 0,46 |
| 277.3 | n29 | 2,14 | 380 | 280 | 8,7 | 0,66 | 107 | 0,46 |
| 277.3 | n30 | 2,14 | 380 | 286 | 8,6 | 0,66 | 110 | 0,46 |
| 277.3 | n31 | 2,14 | 380 | 282 | 8,9 | 0,66 | 108 | 0,44 |
| 277.3 | n32 | 2,14 | 380 | 274 | 8,7 | 0,66 | 105 | 0,46 |
| 277.3 | n33 | 2,14 | 380 | 275 | 8,8 | 0,67 | 108 | 0,46 |
| 277.3 | n34 | 2,14 | 380 | 275 | 8,8 | 0,67 | 108 | 0,46 |
| 277.3 | n35 | 2,14 | 380 | 280 | 8,7 | 0,67 | 109 | 0,46 |
| 277.3 | n36 | 2,14 | 380 | 276 | 8,9 | 0,67 | 105 | 0,45 |
| 277.3 | n37 | 2,14 | 380 | 281 | 8,8 | 0,65 | 106 | 0,45 |
| 277.3 | n38 | 2,14 | 380 | 285 | 8,8 | 0,66 | 106 | 0,36 |
| 277.3 | n39 | 2,14 | 380 | 273 | 8,9 | 0,68 | 105 | 0,45 |
| 277.3 | n40 | 2,14 | 380 | 275 | 8,7 | 0,64 | 105 | 0,45 |

For a matter of size, some columns are not included. In the table the per-node metrics for jobid=277 and stepid=3 are reported one per row. Most of the columns show the same information than in the previous example but per-node rather than the average (we have shortened some names to fit in the page). The extra columns not listed when the `-l` flag is not used are:

- Node: Nodename
- StartTime (not shown in the table): the date and time when the jobid.stepid started
- VPI: VPI (Vectorial Per Instructions) is the percentage of the FLOP-AVX512 performance counter value over the total number of instructions. It provides an insight about the utilization of AVX512 instructions which have a major impact on the application performance and power consumption.

In this view the per-node metrics are reported for the whole execution of the jobid.stepid. The VPI extra column shows us the weight of AVX512 instructions over the total. This ratio shows if the code is highly vectorized which is very important since AVX512 instructions have a different behavior with respect to CPU frequency and power consumption than non-AVX512 instructions.

These columns are a subset of information available in the DB. To get all the metrics the `-c` flag has to be used and it generates a csv file. With the `-c` flag, columns such as total instructions, total cycles and the individual FLOPS event counter are reported.

## VIII. EVALUATION

### A. Applications

We have evaluated executed seven real applications and one kernel benchmark with `min_time_to_solution` energy policy.

- GROMACS. GROningen MAchine for Chemical Simulations [20] is a molecular dynamics package mainly designed for simulations of proteins, lipids and nucleic acids . We used 640 MPI processes, 16 nodes.
- BQCD. Berlin quantum chromodynamics program [1] is a Hybrid Monte-Carlo program for simulating lattice QCD with dynamical Wilson fermions. We used 400 MPI processes, 10 nodes.
- SUSPENSE [7] is an improved method for computing incompressible viscous flow around suspended rigid particles using a fixed and uniform computational grid. We used 1000 MPI processes, 25 nodes.
- DUMSES [19] is a 3D MPIOpenMP & MPI/OpenACC Eulerian second-order Godunov (magneto)hydrodynamic simulation code in cartesian, spherical and cylindrical coordinates. We used 1024 MPI processes, 26 nodes.
- ECMWF OpenIFS [21] is a scientific outreach programme that provides an portable version of the IFS code in use at ECMWF for operational weather forecasting. We used 40 MPI processes with 4 threads, 10 nodes.

- BT-MZ class D. Block Tri-diagonal solver from the NAS-PB [9]. We used 160 MPI processes, 4 nodes.
- AFiD. AFiD is a highly parallel application for Rayleigh-Benard and Taylor-Couette flows. It is developed by Twente University, SURFsara and University of Rome ”Tor Vergata” [33]. We used 576 MPI processes, 15 nodes.
- POP [35] is the open source Parallel Ocean Model version 2 developed by Los Alamos National Lab. We used 384 MPI processes, 10 nodes.

Table III shows for each of the eight use cases the execution time, DC node power, CPI, GBS and GF/W reported by EARL with monitoring_only policy, i.e. without any energy optimization used. This table shows metrics for the whole application.

TABLE III
APPLICATIONS

| Application | CPI | GBs | Exec.Time (sec) | DC Power/node | GFlops/Watt |
|---|---|---|---|---|---|
| GROMACS | 0.66 | 9 | 371 | 321 | 2.88 |
| BQCD | 0.63 | 15 | 222 | 325 | 0.31 |
| BT-MZ.D | 0.40 | 25 | 214 | 343 | 0.34 |
| OpenIFS | 0.67 | 28 | 195 | 291 | 0.18 |
| AFiD | 0.99 | 75 | 262 | 332 | 0.12 |
| POP | 1.64 | 61 | 1565 | 324 | 0.08 |
| SUSPENSE | 1.22 | 90 | 269 | 334 | 0.08 |
| DUMSES | 1.09 | 65 | 550 | 314 | 0.09 |

We see that GROMACS, BQCD, BT-MZ and OpenIFS are cpu bound applications. These applications have high values for GFlops/Watts, corresponding with lower values of CPI (less than 0.7). On the other hand, we note that AFiD, POP, SUSPENSE and DUMSES are memory bound applications with higher CPI values (more than 1), and corresponds with cases with low values of GFlops/Watt, being less energy efficient.

Applications have been executed in a cluster of Lenovo ThinkSystem SD530 nodes where each node includes two Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz (20c) per node (Hyper-threading is activated but we are not using it), 40 cores in total and 12*8GB dual rank DIMMs per node. For all the experiments, three runs have been executed and we are using the average of all three. For a fair comparison, all the executions for each application have been done using the same set of nodes.

Table IV present the energy , power and time variation when using EAR with the min_time_to_solution policy for the cpu bound applications BQCD, BT-MZ, GROMACS and OpenIFS and the memory bound applications POP, DUMSES, AFiD and SUSPENSE. Figure 6 presents the frequency EAR selected for each application. Figure 7 presents the improvement in GFlops/Watt. In all cases the metrics presented in the graphs are the delta of the time, power and energy measurements when the applications are executed with EARL and min_time_to_solution policy vs when they are executed at nominal frequency 2.4 GHz. The default frequency selected is 2.1GHz and the minimum efficiency ratio is set to 70%. In all experiments, we have computed three metrics:

- Increment of the execution time (labeled ”Performance penalty” in table). The lower the better
- Reduction of average Power (labeled ”Power savings” in table). The higher the better
- Reduction of Energy. (labeled ”Energy savings” in table). The higher the better

TABLE IV
APPLICATION EVALUATION WITH MIN_TIME_TO_SOLUTION

| Appplication | Energy saving | Power Saving | Performance penalty |
|---|---|---|---|
| BT | 1% | 2% | 1% |
| BQCD | 2% | 3% | 1% |
| GROMACS | 10% | 12% | 3% |
| OpenIFS | 2% | 4% | 3% |
| POP | 14% | 15% | 1% |
| DUMSES | 7% | 8% | 1% |
| AFiD | 7% | 9% | 2% |
| SUSPENSE | 7% | 11% | 4% |

First four applications do an intensive use of the CPU (as shown by a CPI less or equal than one) and are more sensible to frequency variations, such it is difficult to provide energy savings without performance penalty. This is due to the fact that the performance of applications with a low CPI scales linearly with frequency. This explains that EAR decisions lead to little performance, power and energy variations since larger power savings would lead to larger performance degradations. This will be also highlighted in Figure 6.

Applications with a high AVX512 content and a high VPI ( VPI larger than 0.3) are an exception to this rule. While their performance scale with frequency, it is still possible to reduce their energy with little performance penalty due to the high power consumption of AVX512 instructions. GROMACS is a very good example of such behavior and EARL is able to take benefit of it. GROMACS does an intense utilization of AVX512 instructions (as shown in Table II column VPI). These instructions are very efficient in terms of GFlops/Watt (GROMACS reports 2.9 GFlosp/Watt whereas the other applications report less than 1). AVX512 instructions run at frequencies determined automatically by the processor. As these instructions are complex and use a lot of power, they run at frequencies lower than the base non-AVX512 frequency. For example the processor 6148 as a base non-AVX512 frequency of 2.4 GHz but the base AVX512 frequency is 1.6 GHz and the max frequency with all cores active is 2.2 GHz. Due to this behavior and the introduction of the AVX512 instruction impact in the energy model, EAR is able to find the right compromise between high frequency and energy savings leading to 10% energy savings and only 3% of performance penalty.

Last four applications in table IV have high CPI and high GBS making an intensive use of main memory are less sensible to frequency variations than cpu bound applications. Due to this characteristic, EAR is able to find significant energy savings with up to 11% for POP and 7% for the others, highlighting that the frequency influence on performance decreases as CPI gets higher, as shown in Table III.

Figure 6 shows the average frequency computed when running with `min_time_solution` policy. We clearly see how EARL dynamically selects the frequency according to the application characteristics, resulting in lower frequencies for memory intensive applications and frequencies close to the maximum (2.4Ghz) for CPU intensive applications with the exception of highly vectorized applications. It is worth mentioning that the average frequency reported by EAR is computed using the msr registers "aperf" and "mperf" which report always a frequency which is a few KHz lower than the frequency set by the userspace governor. For instance, an average frequency of 2.37 or 2.38 corresponds to a 2.4GHz CPU frequency.

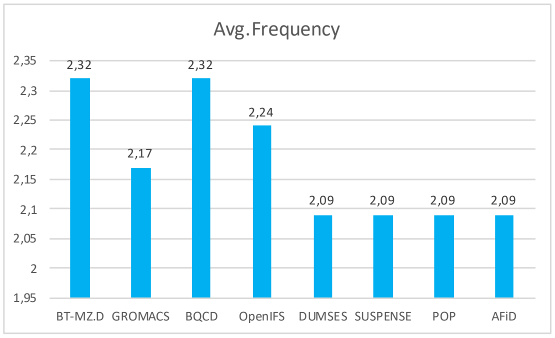

Fig. 6. Average frequency

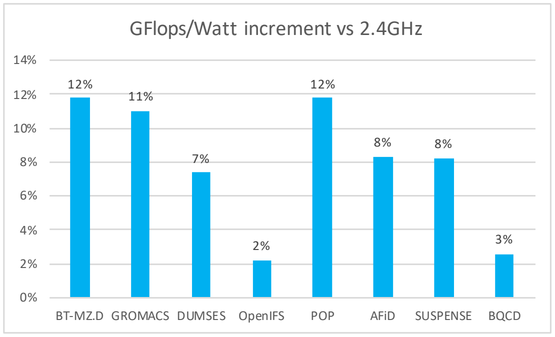

Fig. 7. GFlops/Watt increment

Figure 7 shows the increment in GFlops/Watt computed when using EAR with `min_time_solution` policy.

## IX. Related work

Some other recent works have also been tackling the power/energy management. Reference [13] presents a complementary approach to EARL. READEX tool suite is a set of tools targeted to help application developers to create an energy efficient version of their applications. It requires much more user intervention since applications must be compiled and tuned to be able to construct the model that will be used later at runtime. Moreover, they rely on external system software, such as the SLURM [15] or HDEEM [14] to compute energy consumption. EAR framework doesn't need any external software to calculate the metrics for energy management. Moreover, EAR is not only providing energy optimization as it provides also a more general framework offering energy/power control, accounting as well as hints about the system reliability and performance whereas READEX has a scope similar to EARL.

In [22] authors propose Adagio which is very interesting approach for power management. Adagio is a runtime library targeted to save as much energy as possible minimising the performance penalty. Adagio takes its frequency decisions at runtime and doesn't need user intervention. EARL is similar to Adagio as it targets also energy optimization at runtime with on-line collected information. However, it differs in the approach and the scope of the work. EARL supports two energy policies and plans for the near future to support a clear interface to simplify the introduction of new policies (such as the power balancing policies used in Adagio). EARL has been used in production with a significant number of cores and hybrid applications on SuperMUC-NG system at LRZ with 6480 nodes demonstrating its robustness and reliability [37]. Moreover, our scope coves not only power optimization but also global energy accounting and control with the EAR Global Manager EARGM. Reference [16] presents Conductor, a run-time system that intelligently distributes available power to nodes and cores to improve performance. Conductor and EAR are not directly comparable since EAR doesn't provide instantaneous power capping but both targets energy management (or power) in a dynamic way. Conductor is also based on detecting critical paths using more power in these parts. However Conductor requires the user to mark the end of the iterative timestep while DynAIS does it automatically. Conductor targets MPI+OpenMP applications and it can be seen as an extension or improvement of Adagio since it exploits similar concepts in a different context. Conductor exploits the thread level by reconfiguring the number of threads and later redistributing the power to speedup the critical path. Conductor and Adagio assume a per-job power bound while EAR assumes a global energy limit with no per-job limits. Conductor uses a global scheduler after application reconfiguration for power reallocation. It does global synchronization for a few time steps, expecting the applications is running so many time step that this overhead will become negligible. EARL is not applying global synchronisations in order to minimize the overhead. Authors reports an overhead of 35 usec. per MPI call with Conductor while EAR algorithm needs only 0,1 usec.. Our experience with applications doing millions of MPI calls demonstrate that 35 usec can lead to serious overheads.

GEOPM [23] from Intel also presents an open source framework for power management. GEOPM implements a hierarchical design to support Exascale systems. GEOPM is an extensible framework where new policies can be added at the node level or application (MPI) level. Some GEOPM policies requires application modifications but some others don't. However in this case one additional process is created and metrics used for frequency selection cannot be associated with specific parts of the application structure (for instance outer loops). GEOPM doesn't include the EARL capability to dynamically detect the application structure offered by DynAIS. Energy control in GEOPM is not included as part of the GEOPM framework but it offers an API to be used by the resource manager while energy control is part of the EAR framework.

DVFS has been also used in other works to reduce the power consumed by applications. In the context of MPI applications, DVFS have been used extensively inside the MPI library to reduce the power consumed during communication periods [24] [25] [26]. The goal of these proposals was to reduce the power consumed inside the MPI library without introducing a significant performance degradation in the application execution. EAR does not only consider communication parts of an application but analyzes the outer loop of the application to determine the best frequency to minimize the energy for the whole iteration.

Power capping has been also targeted at the scheduler and resource manager level, for instance in SLURM [27] or PBS [28] to control the total power consumption. EAR provides also energy capping which means EAR will ensure that a given maximum total power consumption will not be reached on the average over a given period of time.

Frequency selection at the cluster was used in the Energy Aware Scheduling feature of the IBM LSF scheduler [29] [30]. With this feature, frequency selection was statically done for the all job, based on a user provided "energy tag" at job submission and historic information gathered during the execution of previous jobs run with the same tag. But relying on user information is a source of error which limited the success of this feature.

Energy Efficiency is a larger topic that the one addressed by EAR and the other tools and this work about energy management [36] is a good presentation of all aspects of energy efficiency from the servers to the data centers, from hardware to software.

## X. Conclusions and Future Work

This paper shows how EAR provides automatic and transparent support for energy optimization for MPI applications. Optimization is done through the the EAR library based on the energy policies included by default. In this paper we have evaluated seven real applications being representative use cases running on up to 1024 cores (26 nodes) with millions of MPI calls. EAR is used at the Leibniz Supercomputing Center (LRZ) on the 6480 compute nodes SuperMUCNG supercomputer, running a lot of real applications on a very large number of core and threads [37].

EAR library is able to dynamically compute the application signature used by energy models and energy policies to select the optimal CPU frequency on the fly without any user information or application modification.

The EAR command eacct is a simple command line tool to recover applications metrics computed by EAR library which simplifies the application analysis.

This paper is focussing on energy optimization but EAR has other values like hints about the node reliability and applications performance leading to a better utilization of the system and its workload.

EAR is currently ported on Intel CPUs. Next development steps will bring support for AMD CPU and NVIDIA GPU as well as instantaneous power capping and support for non-MPI applications.

It's also worth to mention the information gathered by EARL for applications is really valuable since it includes signatures computed at runtime and global application signatures. All these signatures characterizes system workload with meaningful data associated with relevant parts of applications. This information could be used for system optimization, specific energy models, application classification etc.

## Acknowledgment

Authors want to thank Peter Mayes from Lenovo HPC and AI Application Team for his valuable help in doing the EAR installation and tuning of the Lenovo cluster used for this evaluation, the Application Support Department of the Leibniz Supercomputing Center (LRZ) of the Bavarian Academy of Sciences and Humanity for valuable comments and suggestions on designing EAR. Authors also wants to thanks Prof. Uhlmann for allowing us to use SUSPENSE in this evaluation, and Kim Serradell from Earth Sciences department for the guidance on how to use OpenIFS. We also wants to thank the SURF Open Innovation Lab, in particular Dr Axel Berg and Sagar Dolas with them help in evaluating POP and AFiD and also to Marc Joos for his help with DUMSES application.

## References

[1] BQCD. https://www.rrz.uni-hamburg.de/services/hpc/bqcd
[2] Intel Node Manager. https://www.intel.com/content/www/us/en/data-center/data-center-management/node-manager-general.html
[3] SLURM SPAN plugin. https://slurm.schedmd.com/spank.html
[4] ThinkSystem SD650 Direct Water Cooled Server https://lenovopress.com/lp0636-thinksystem-sd650-direct-water-cooled-server
[5] J.R. Deller, Jr., J. G. Proakis, and J.H. Hansen. 1993. Discrete Time Processing of Speech Signals (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA
[6] F. Freitag, J. Corbalan, and J. Labarta. "A dynamic periodicity detector: Application to speedup computation." Parallel and Distributed Processing Symposium., Proceedings 15th International. IEEE, 2001.
[7] M. Uhlmann, "An immersed boundary method with direct forcing for the simulation of particulate flows,Journal of Computational Physics", Volume 209, Issue 2, 2005,Pages 448-476,ISSN 0021-9991, https://doi.org/10.1016/j.jcp.2005.03.017. http://www.sciencedirect.com/science/article/pii/S0021999105001385
[8] A. Auweter, A. Bode,M. Brehm,L. Brochard,N. Hammer,H. Huber,R. Panda,F. Thomas,T. Wilde . "A case study of energy aware scheduling on super-muc." International Supercomputing Conference. Springer, Cham, 2014
[9] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. "The NAS parallel benchmarks." The International Journal of Supercomputing Applications 5.3 (1991): 63-73. https://www.nas.nasa.gov/assets/npb/NPB3.3.1-MZ.tar.gz

[10] H. Feng, R. VanderWijngaart, R. Biswas, C. Mavriplis. ”Unstructured Adaptive (UA) NAS Parallel Benchmark. Version 1.0.” (2004).
[11] Intel Math kernel libray. https://software.intel.com/en-us/mkl/documentation/code-samples
[12] The stream_mpi benchmark. https://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.c
[13] READEX project. https://www.readex.eu/
[14] T. Ilsche, R. Schne, J.Schuchart, D. Hackenberg, M. Simon, Yi. Georgiou and W. E. Nagel. ”Power Measurement Techniques for Energy-Efficient Computing: Reconciling Scalability, Resolution, and Accuracy” In: Second Workshop on Energy-Aware High Performance Computing (EnA-HPC). 2017. DOI: 10.1007/s00450-018-0392-9
[15] SLURM. https://slurm.schedmd.com/
[16] A. Marathe, P.E. Bailey, D.K. Lowenthal, B. Rountree, M.Schulz, B.R. de Supinski. (2015) A Run-Time System for Power-Constrained HPC Applications. In: Kunkel J., Ludwig T. (eds) High Performance Computing. ISC High Performance 2015.Lecture Notes in Computer Science, vol 9137. Springer, Cham https://e-reports-ext.llnl.gov/pdf/789054.pdf
[17] EAR web page. https://www.bsc.es/research-and-development/software-and-apps/software-list/energy-aware-runtime-ear
[18] EAR user guide. https://gateway.bsc.es:11003/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear_guide.pdf
[19] DUMSES. https://github.com/marcjoos-phd/dumses-hybrid
[20] GROMACS. http://www.gromacs.org/
[21] ECMWF. IFS Documentation CY41R1: IFS documentation. Part III: Dynamics and Numerical Procedures. https://www.ecmwf.int/sites/default/files/elibrary/2014/9203-part-iii-dynamics-and-numerical-procedures.pdf
[22] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. 2009. Adagio: making DVS practical for complex HPC applications. In Proceedings of the 23rd international conference on Supercomputing (ICS ’09). ACM, New York, NY, USA, 460-469. DOI: https://doi.org/10.1145/1542275.1542340
[23] 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.” International Supercomputing Conference. Springer, Cham, 2017.
[24] A.Venkatesh, A.Vishnu, K.Hamidouche, N. Tallent, D. (DK) Panda, D.Kerbyson, and A. Hoisie ”A case for application-oblivious energy-efficient MPI runtime.” Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2015
[25] M. Etinski, J. Corbalan, J. Labarta, M. Valero and A. Veidenbaum. ”Power-aware load balancing of large scale MPI applications.” Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009.
[26] M. Y. Lim, V. W. Freeh and D. K. Lowenthal. ”Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs.” SC 2006 conference, proceedings of the ACM/IEEE. IEEE, 2006.
[27] SLURM Power Adaptive Computing. https://slurm.schedmd.com/SLUG15/Power_Adaptive_final.pdf
[28] PBSpro Green computing http://www.pbsworks.com
[29] R.Bell, L.Brochard, D.DeSotta, R.Panda, F.Thomas. Energy-Aware job scheduling for cluster environments. Patent US 8,527,997 B2. November 2011
[30] L.Brochard, R.Panda, D.DeSota, F.Thomas, and R.H. Bell, Jr. ”Power and energy-aware processor scheduling.” ACM SIGSOFT Software Engineering Notes. Vol. 36. No. 5. ACM, 2011.
[31] The MariaDB Foundation https://mariadb.org/
[32] PAPI. http://icl.cs.utk.edu/papi/
[33] Erwin P. van der Poel and Rodolfo Ostilla-Mnico and John Donners and Roberto Verzicco.”A pencil distributed finite difference code for strongly turbulent wall-bounded flows”. Computers & Fluids, vol. 116,pages 10-16. Year 2015. doi https://doi.org/10.1016/j.compfluid.2015.04.007. http://www.sciencedirect.com/science/article/pii/S004579301500116
[34] GNU FreeIPMI https://www.gnu.org/software/freeipmi/
[35] POP http://www.cesm.ucar.edu/models/ccsm4.0/pop/
[36] L. Brochard, V. Kamath, J. Corbalan, S. Holland, W. Mittelbach, M. Ott. ”EnergyEfficient Computing and Data Centers”. ISTE Ltd 2019. ISBN:9781786301857. DOI:10.1002/9781119422037
[37] EAR at SuperMUC-NG: https://doku.lrz.de/display/PUBLIC/Energy+Aware+Runtime