# Energy optimization with EAR*

Julita Corbalan[1,2]†, Lluis Alonso[3]‡, and Jordi Aneas[3]§

[1] Barcelona Supercomputing Center (BSC),
[2] Politechnic University of Catalonia (UPC), Barcelona, Spain
julita.corbalan@bsc.es
[3] Barcelona Supercomputing Center (BSC), Barcelona, Spain.
lluis.alonso@bsc.es
jordi.aneas@bsc.es

**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. 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 with the list of available frequencies. 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 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.

# Contents



*This work has been funded by the BSC-Lenovo collaboration agreement

†

‡

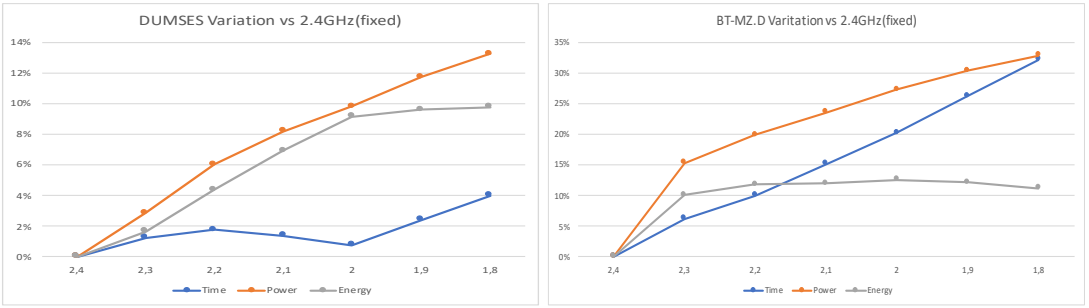
§

# 1 Introduction

This paper presents how to optimize energy by using one of EAR's components, the EAR runtime library, from here on known as EARL, when using one of the energy policies included with EAR by default: `min_time_to_solution`. EARL measures a set of application metrics at runtime called application signature, and using energy models and energy policies, automatically selects the optimal frequency for the specific run.

Figure 1 shows 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 is doing an exhaustive utilization of main memory, being less sensible to changes in the CPU frequency. First type of applications could report significant savings in energy but at the expense of excessive increments in execution time, and thus being not acceptable.

Table 1 in section 7 presents each application's characteristics, which help understand the performance results included in this paper. 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.




(a) Impact on DUMSES when varying CPU frequency

(b) Impact on BTMZ.D when varying CPU frequency

Figure 1: Time, Power and Energy variation when changing 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 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* to do application analysis and workload characterization.

The evaluation section will show time, power and energy as well as average frequency and GFlops/Watt. Results show 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 2 describes EAR as energy management

framework. Section 3 presents EAR library in detail. Section 6 shows how to run and get performance metrics with EAR. Section7 evaluates EARL in terms of performance and energy savings. Section 8 presents the related work. And finally, section 9 presents conclusions and future work.

## 2 EAR overview

EAR offers three main services concerning energy management (see Figure 2): Monitoring, Accounting, Control and Optimisation. Energy optimisation is offered by EARL and only enabled for MPI or hybrid MPI+OpenMP applications.


Figure 2: Energy capabilities provided by EAR

Fig. 3 shows the EAR components and main interactions between them: EAR Daemon, EAR Library, 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, 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.
- Optimization and per-application control is provided by EARL and deeply described in section 3.
- Global control is offered by the EAR Global Manager (EARMG) and out of the scope of this paper.

## 3 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 4 shows EARL's lifecycle with its main components. One of the EAR research contributions is the algorithm for outer loops detection on the fly without application modification, DynAIS. It is one the main strengths when compared with other runtime solutions. EARL includes two energy policies by default: `min_time_to_solution` and `min_energy_to_solution`, targeted to save energy by reducing frequency.

As previously stated, the algorithm responsible of this application characterisation is DynAIS. 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

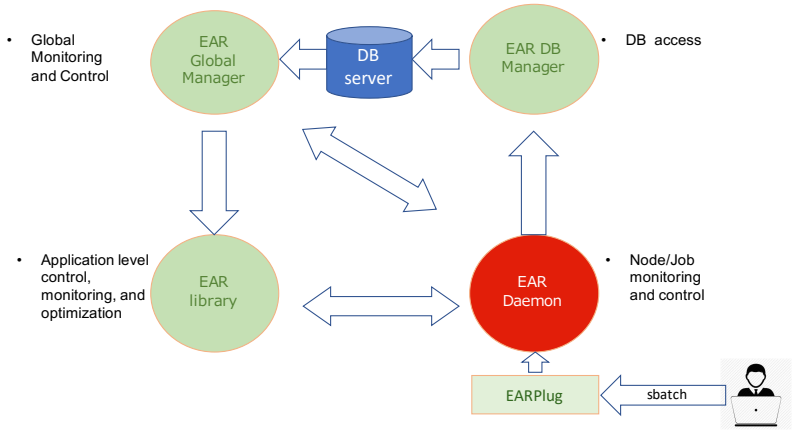



Figure 3: EAR components

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 why using the `--ear-policy=policy_name` when submitting the job.

Fig. 4 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).

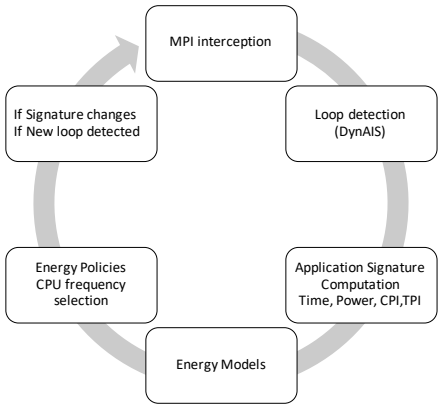



Figure 4: EARL internals

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*

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

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, EARL detects any change in the application, either detecting a new loop or a change in its signature, and applies the energy policy again with the new context and metrics.

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 that situation and executes steps 3 to 5 every N seconds (configurable). We refer to the default mode as *DynAIS mode* and this case as *Periodic mode*.

# 4 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 per-node matrix of coefficients (A..F) where each point in the matrix is a set of six coefficients used to project time and power from $\text{Freq}_\text{i}$ to $\text{Freq}_\text{j}$. During the training period a set of pre-selected kernels are executed with different frequencies. Kernels have been selected to guarantee different applications characteristics, from CPU intensive applications to memory intensive ones. Currently, the learning selected applications are: 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 said model depends on the system, it can only be modified by the *sysadmin* in EAR's configuration.

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

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

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

# 5 EARL energy optimization policies: Minimize time to solution

Minimize time to solution policy reduces the execution time while guaranteeing a minimum ratio between *performance benefit* and *frequency increment* that justifies this energy consumption. The policy uses a *minimum efficiency* ratio set by the sysadmin . For example, if `min_ratio=0.75`, EAR will prevent scaling to upper frequencies if the ratio between performance gain and frequency gain do not improve at least 75% `(PerfGain >= FreqGain* min_ratio)`(4)(5).

$$PerfGain = (Time - Timenew)/Time \tag{4}$$

$$FreqGain = (Freqnew - Freq)/Freq \tag{5}$$

When executed with `min_time_to_solution` policy, applications start at a default predefined frequency lower than its 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` would 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 go on with the next performance projection $F_{i+2}$. Otherwise, the policy will select the last frequency where the performance gain was enough, preventing any waste of energy.

# 6 Executing with EAR and getting performance metrics

To execute an application with EARL is totally transparent in systems where the 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 in this paper have been done using one of the approaches shown below (which are equivalent). 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.

```
/* 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 executed, it is possible to check its metrics by using the `eacct` command. This command accepts different flags to add filters, to specify the amount of details reported, or to select the output (stdout or csv file).

For example, Figure 5 shows the `eacct` output when executing POP application. This output can be easily loaded in any spreadsheet tool.

eacct -j 125414

| JOB-STEP | USER | APPLICATION | POLICY | NODES# | FREQ(GHz) | TIME(s) | POWER(Watts) | GBS | CPI | ENERGY(J) | GFLOPS/WATT | MAX POWER(W) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 125414-sbatch | xjcorbalan | pop384 | NP | 1 | 2.04 | 9437.00 | 293.20 | NO-EARL | NO-EARL | 2766927.98 | NO-EARL | 293.20 |
| 125414-3 | xjcorbalan | pop384 | MT | 10 | 2.09 | 1540.82 | 275.61 | 61.89 | 1.54 | 4246590.32 | 0.09 | 340.68 |
| 125414-2 | xjcorbalan | pop384 | MT | 10 | 2.09 | 1573.30 | 273.98 | 60.75 | 1.54 | 4310492.04 | 0.09 | 337.78 |
| 125414-1 | xjcorbalan | pop384 | MO | 10 | 2.38 | 1516.81 | 313.94 | 62.90 | 1.63 | 4761907.02 | 0.09 | 368.15 |
| 125414-0 | xjcorbalan | pop384 | MO | 10 | 2.38 | 1557.90 | 311.65 | 61.49 | 1.64 | 4855262.25 | 0.08 | 362.18 |

Figure 5: eacct: jobid filter (-j)

When using `-j` option, columns reported are the following ones:

- JOB-STEP: 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: username
- Application: Job Name set by the user when submitting the job or executable name when not provided
- Policy: Energy policy. NP= No energy policy. MT=min_time, MO=monitoring only. ME=min_energy (not used in this example)
- Nodes: Number of nodes used by the jobid.stepid
- Freq(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, it is the effective CPU frequency computed using aperf/mperf 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.
- Energy. Accumulated energy (in Joules) of all the nodes running this jobid.stepid
- 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. During the jobid.stepid execution, the power is computed periodically. This value is the maximum value detected during these periodic metrics.

eacct -j 125414.0 -l

| JOBID-STEPID | NODE | USER | APP | FREQ | TIME | POWER | GBS | CPI | ENERGY | START TIME | VPI | MAX POWER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 125414-0 | cmp2501 | xjcorbalan | pop384 | 2.38 | 1566.23 | 316.35 | 62.93 | 1.52 | 495485.56 | 21/4/2020 21/4/2020 | 0.00 | 347.57 |
| 125414-0 | cmp2502 | xjcorbalan | pop384 | 2.38 | 1556.97 | 313.15 | 66.42 | 1.74 | 487560.21 | 21/4/2020 21/4/2020 | 0.00 | 347.27 |
| 125414-0 | cmp2503 | xjcorbalan | pop384 | 2.38 | 1556.98 | 327.08 | 66.52 | 1.71 | 509253.55 | 21/4/2020 21/4/2020 | 0.00 | 362.18 |
| 125414-0 | cmp2504 | xjcorbalan | pop384 | 2.38 | 1556.96 | 321.96 | 65.97 | 1.79 | 501271.56 | 21/4/2020 21/4/2020 | 0.00 | 358.53 |
| 125414-0 | cmp2505 | xjcorbalan | pop384 | 2.38 | 1556.98 | 325.57 | 65.63 | 1.77 | 506910.89 | 21/4/2020 21/4/2020 | 0.00 | 359.32 |
| 125414-0 | cmp2509 | xjcorbalan | pop384 | 2.38 | 1556.98 | 316.33 | 65.42 | 1.70 | 492520.65 | 21/4/2020 21/4/2020 | 0.00 | 352.16 |
| 125414-0 | cmp2510 | xjcorbalan | pop384 | 2.38 | 1556.98 | 316.41 | 64.07 | 1.71 | 492640.93 | 21/4/2020 21/4/2020 | 0.00 | 347.10 |
| 125414-0 | cmp2511 | xjcorbalan | pop384 | 2.38 | 1556.98 | 314.51 | 63.44 | 1.69 | 489677.74 | 21/4/2020 21/4/2020 | 0.00 | 349.26 |
| 125414-0 | cmp2512 | xjcorbalan | pop384 | 2.38 | 1556.98 | 316.41 | 62.65 | 1.64 | 492646.24 | 21/4/2020 21/4/2020 | 0.00 | 346.61 |
| 125414-0 | cmp2517 | xjcorbalan | pop384 | 2.39 | 1556.98 | 248.75 | 31.86 | 1.18 | 387294.92 | 21/4/2020 21/4/2020 | 0.00 | 269.66 |

Figure 6: eacct: jobid, stepid filter (-j) with per-node extended information (-l)

Figure 6 shows the output when using the extended information option, where per-node metrics are shown. These columns are a subset of information available in the DB. To ask for all the metrics a `-c` flag is provided (csv file is then generated). In the figure you can see the per-node (each row is one node) metrics for jobid=125414 and stepid=0. Most of the columns show the same information than in the previous example but per-node rather than the average (we have shorten some names to fit in the page). Different columns are:

- Node: Nodename
- StartTime: The date and time the jobid.stepid started
- VPI. VPI is the percentage of SIMD instructions (AVX512) executed over the total number of instructions (Vectorial Per Instructions). It gives us an insight about the utilization of AVX512 instructions.

# 7 Evaluation

## 7.1 Applications

We have evaluated 7 real applications and 1 benchmark with `min_time_to_solution` energy policy and we have computed 3 metrics:

- Increment of the execution time (Time in graphs). The lower the better
- Reduction of average Power (Power). The higher the better
- Reduction of Energy. (Energy). The higher the better

In all the cases the, metrics are computed as the variation against application executed at a fixed frequency 2.4GHz. The default frequency selected is 2.1GHz and the minimum efficiency ratio applied has been 70%.

Table 1 shows specific application configuration in terms of number of nodes, tasks, and cores as well as main application characteristics (CPI and GBs). Given these different behaviours, we present a total of 8 use cases. Results show how EARL is able to identify each use case dynamically and adapt the frequency for each application. This table shows metrics for the whole applications.

- GROMACS. GROningen MAchine for Chemical Simulations [20] is a molecular dynamics package mainly designed for simulations of proteins, lipids and nucleic acids .

- BQCD. Berlin quantum chromodynamics program [1] is a Hybrid Monte-Carlo program for simulating lattice QCD with dynamical Wilson fermions.
- SUSPENSE [7] is an improved method for computing incompressible viscous flow around suspended rigid particles using a fixed and uniform computational grid.
- DUMSES [19] is a 3D MPIOpenMP & MPI/OpenACC Eulerian second-order Godunov (magneto)hydrodynamic simulation code in cartesian, spherical and cylindrical coordinates .
- ECMWF OpenIFS [21] is a scientific outreach programme that provides an portable version of the IFS in use at ECMWF for operational weather forecasting.
- BT-MZ class D. Block Tri-diagonal solver from the NAS-PB [9]
- 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].
- POP [35] is the open source Parallel Ocean Model version 2 developed by Los Alamos National Lab

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 (Hyperthreading 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 1: Applications

| Application | Num.Nodes | MPIs | Threads PerProcess | Total cores | Avg.CPI | Avg.GBs | Exec.Time (sec) | Avg.DC Power/node | GFlops/Watt |
|---|---|---|---|---|---|---|---|---|---|
| GROMACS | 16 | 640 | 1 | 640 | 0.66 | 9 | 737 | 327 | 2.92 |
| BQCD | 10 | 400 | 1 | 400 | 0.63 | 15 | 222 | 325 | 0.31 |
| BT-MZ.D | 4 | 160 | 1 | 160 | 0.40 | 25 | 214 | 343 | 0.34 |
| OpenIFS | 10 | 40 | 4 | 160 | 0.67 | 28 | 195 | 291 | 0.18 |
| AFiD | 15 | 576 | 1 | 576 | 0.99 | 75 | 262 | 332 | 0.12 |
| POP | 10 | 384 | 1 | 384 | 1.64 | 61 | 1565 | 324 | 0.08 |
| SUSPENSE | 25 | 1000 | 1 | 1000 | 1.22 | 90 | 269 | 334 | 0.08 |
| DUMSES | 26 | 1024 | 1 | 1024 | 1.09 | 65 | 550 | 314 | 0.09 |

We can see how GROMACS, BQCD, BT-MZ and OpenIFS are cpu bound applications. These applications have high values for GFlops/Watts, corresponding also with lower values of CPI (less than 0.7). AFiD, POP, SUSPENSE and DUMSES are memory bound applications with higher CPI values (more than 0.9), and corresponds with cases with low values of GFlops/Watt, being less energy efficient.

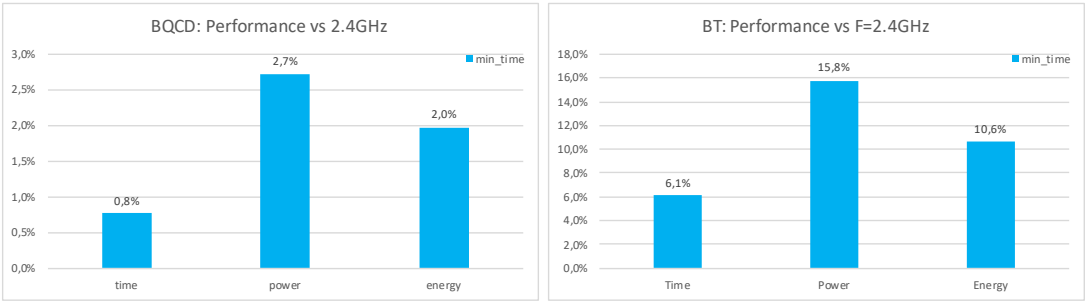
Figures 7a to 10b present the impact of EAR with min_time_to_solution policy on these applications. Figure 11a presents the frequency EAR selected for each application. Figure 11b presents the improvment in GFlops/Watt the min_time_to_solution policy provided.

Figures 7a,7b,8a and 8b show time, power and energy variation for cpu bound applications. These applications do an intensive use of the CPU and are more sensible to frequency variations,

being difficult to provide energy savings without performance penalty since execution time in this type of applications is linear with the CPU frequency.

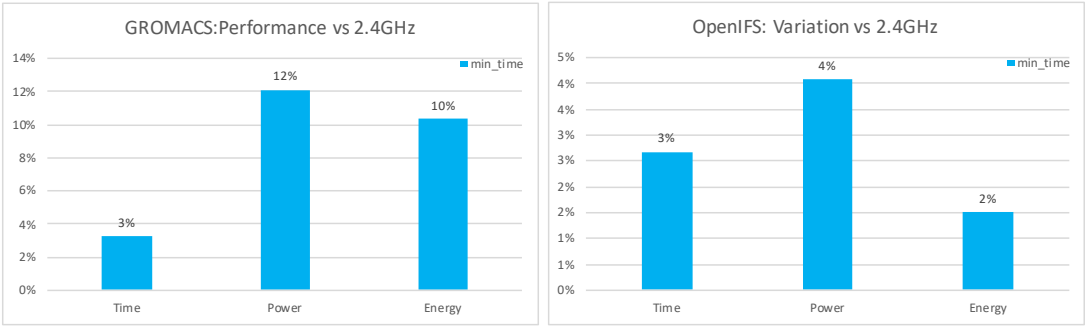
In the case of BQCD, BT-MZ, GROMACS and OpenIFS EARL automatically identifies the application signature and `min_time_to_solution` boosts applications depending on the signature, but anyway increasing the default frequency. The goal of this energy policy is not to save energy, is to do an efficient utilization of energy, accelerating up to the nominal frequency applications were the execution time scales with the frequency.

The case of GROMACS needs a special mention. GROMACS does an intense utilization of AVX512 instructions. These instructions are very efficient in terms of GFlops/Watt (GROMACS reports 2.92GFlosp/Watt whereas the other applications report less than 1). Applications using AVX512 are very good candidates for an optimal energy utilization with EAR since our framework automatically identifies them. Moreover, the maximum frequency for this type of instructions, AVX512, is automatically managed by the hardware and depending on the number of actives cores and the percentage of utilization of these instructions their maximum is lower than the nominal frequency. EAR measures this percentage and adapts the CPU frequency to take that into consideration, resulting in the 10% of energy reduction for GROMASCS.




(a) BQCD (b) BT

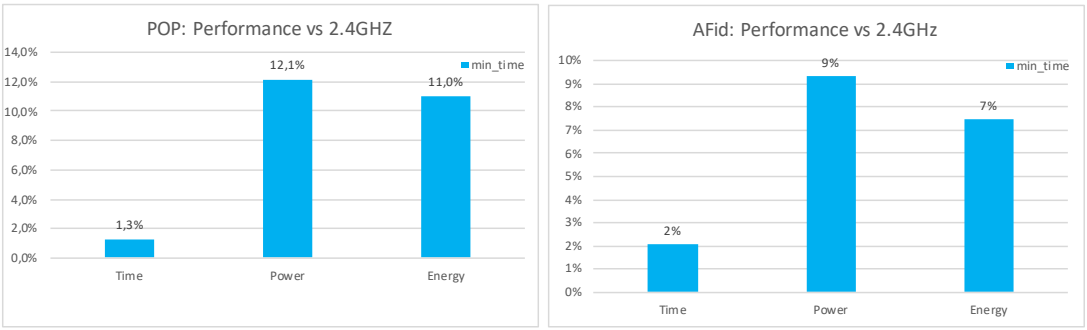
Figure 7: Performance evaluation of BTMZ and BQCD with min_time_to_solution




(a) GROMACS (b) OpenIFS

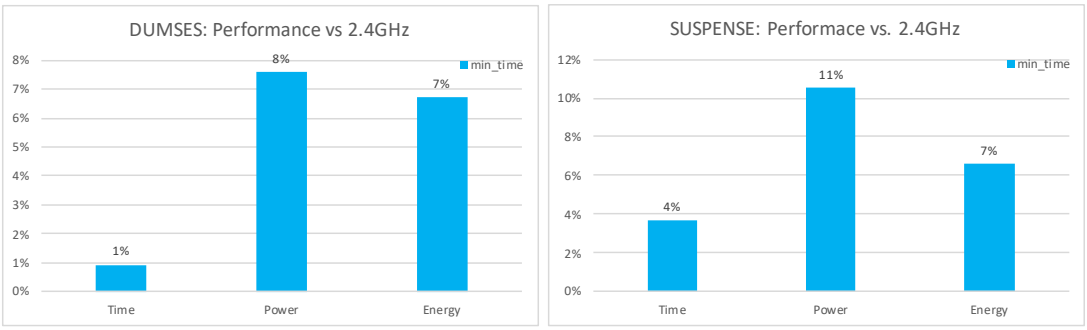
Figure 8: Performance evaluation of GROMACS and OpenIFS with min_time_to_solution

Figure 10 shows the Time, Power and Energy variation for memory bound applications. These applications have high CPI and make an intensive use of main memory and are not so sensible to frequency variations as *CPU* applications.




(a) POP (b) AFiD

Figure 9: Performance evaluation of POP andAFiD with min_time_to_solution




(a) DUMSES (b) SUSPENSE

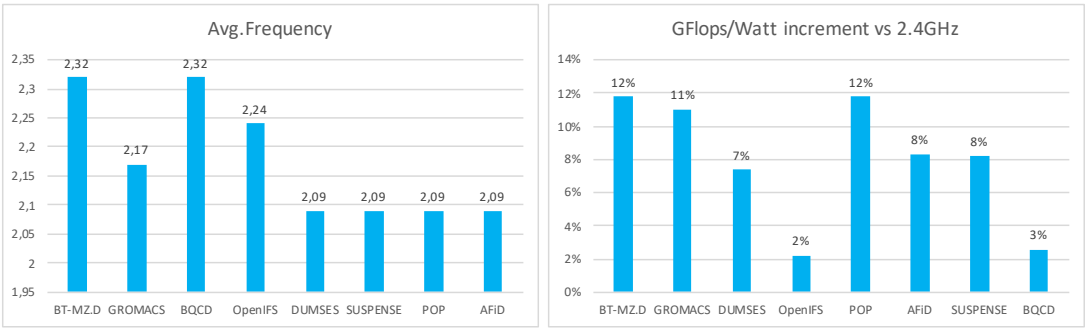
Figure 10: Performance evaluation of DUMSES and SUSPENSE with min_time_to_solution

Figure 11a shows the average frequency computed when running with `min_time_solution` policy. We can clearly see how EARL with the energy policy 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. It is worth mentioning that the average frequency is computed using msr aperf/mperf registers by EAR being always a few KHz lower than the frequency set in the userspace governor. For instance, an average frequency of 2.37 or 2.38 corresponds to a 2.4GHz CPU frequency.

Figure 11b shows the increment in GFlops/Watt computed when using EAR with `min_time_solution` policy. Results show how EAR provides an increment of up to 12% in the energy efficiency in the case of POP with an average improvement of 8%.

# 8 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 much be compiled and tuned to be able to construct the model that will be used later at runtime. Moreover, they rely on external system




(a) Average Frequency with min_time_to_solution (b) GFlops/Watt increment

Figure 11: Average frequency and GFlops/Watt increment when using EARL

software, such as the SLURM [15] or HDEEM [14] to compute energy consumption. EAR framework doesn't need any external software to calculate metrics for energy management. Moreover, EAR is not only targeted to energy / power optimization. It is a more general framework offering energy/power control and accounting whereas READEX is limited to a similar scope to EARL.

In [22] authors propose Adagio. A 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. EAR is similar to Adagio in the sense we target energy optimization at runtime with on line collected information. However, we differ in the approach and the scope of the work. EARL supports two energy policies and our plan for the future work is to specify a clear interface to make it easy the introduction of new policies (such as the power balancing policies used in Adagio). EARL has been evaluated with a significant number of cores and hybrid applications, being ready for production systems. Moreover, our scope coves not only power optimization but also energy accounting and control with the coordination with the EARGM. Reference [16] presents Conductor. It is run-time system that intelligently distributes available power to nodes and cores to improve performance. It is also based on detecting critical paths and using more power in these parts, however, Conductor requires the user to mark the end of the iterative timestep. Conductor targets MPI+OpenMP applications and it can be envision as an extension or improvement of Adagio since it exploits similar concepts in a different context. Conductor exploits the thread level by doing a reconfiguration of the concurrency level and later redistributing the power to speedup the critical path. They assume a context where there is a per-job power bound while we are assuming there are global limits but not per-job limits. Conductor uses a global scheduler after application reconfiguration for power reallocation. It does global syncrhonization for a few time steps, expecting the applications is running so many time step that this overhead will become negligible. We are not applying global synchronisations in EARL to minimize the application interferences. Moreover this approach has a limitation in the power cap for the node, so there is a limit in the amount of power that can be reallocated. Authors reports an overhead of 35 usec. per MPI call. Our algorithm is much more efficient (0,1 usec.) and our experience with the applications evaluated with millions of MPI calls demonstrate that 35usec. is too much for most of them.

GEOPM [23] from Intel also presents an open source framework for power management. GEOPM implements a hierarchical design to support Exascale systems. However, GEOPM

doesnt have the technology to dynamically detect the application structure. GEOPM is an extensible framework where new policies can be added at the node level or application (MPI) level. Some of the policies requires application modifications but some others don't have this requirement. 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 of dynamic application characterisation offered by DynAIS to EARL. Energy control in GEOPM is not included as part of GEOPM framework, it offers an API to be used by the resource manager. In the case of EAR, a complete framework is provided. 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 considers not only communication parts, it takes into account all the code executed by the application, reducing the frequency considering the application as a whole, being able to detect, for instance, memory bounded applications which present good opportunities for energy savings. 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. Frequency selection at the cluster was used in the LSF scheduler [29][30]. In this implementation, frequency selection was statically done at the job submission based on user provided information at job submission and historic information. Relying on user information is a source of errors, limiting the success of the proposal.

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.

# 9 Conclusions

This paper shows how EAR provides automatic and transparent support for energy optimization for MPI applications. Optimization is done through the the EAR library using some of the energy policies included. In this paper we have evaluated 8 real applications with up to 1024 cores (26 nodes), with millions of MPI calls, being representative use cases, not benchmarks. 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 set of EAR commands include eacct, a simple command line tool to recover applications metrics computed by EAR library, being really easy the application analysis and evaluation.

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

# 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 in the evaluation platform, the Application Support Department of the Leibniz Supercomputing Center 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 our evaluation, and Kim Serradell from Earth Sciences deparment 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. Finally, 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/PBSSolution.aspx?v=1&i=6&n=Green-Computing

[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