# Energy Aware Runtime (EAR) documentation

---

# User guide

This document is part of the Energy Aware Runtime (EAR) framework. It has been created in the context of the BSC-Lenovo Cooperation project.


Contact

BSC Julita Corbalan julita.corbalan@bsc.es

Lenovo Luigi Brochard lbrochard@lenovo.com


2017 BSC-Lenovo internal. All rights reserved 

# Energy Aware Runtime (EAR) user guide

---

Energy Aware Runtime (EAR) is designed to provide a simple solution for MPI applications to be energy efficient. EAR is designed to work independently of any scheduler, however, we provide a SLURM plugin to make easy the EAR utilization. This document describes main EAR features, how to automatically use EAR in SLURM systems and how to use it in not-SLURM systems.

## 1 Executing applications with EAR

EAR core components are the EAR library and the EAR daemon. EAR library is dynamically loaded with MPI applications in order to provide a transparent and automatic solution for applications, and configured through environment variables. EAR daemon is an external process that must be launched (one per node) before application is started. To automatize application execution, EAR distribution includes:

- a SLURM plugin which adds some options to the srun command
- a bash file with four functions that will simplify the process of executing applications with EAR for non-SLURM systems. These functions can be found in $EAR_INSTALL_PATH/ear_scripts/tests/EAR_utils_functions.sh file. In the same folder there are some examples that use these functions.

Sections 1.1..1.3 presents environment variables that configures the EAR behavior. Section 3 describes the bash functions included in the EAR_utils_functions.sh file. Figure 1 shows main inputs and outputs for EAR. EAR environment variables are automatically set when using srun options.

With these environment variables we can select the power policy (EAR_POWER_POLICY), thresholds to be used by this policy (depends on the policy, see section 1.1 ), the verbosity level (EAR_VERBOSE), application name to be used when reporting metrics (EAR_APP_NAME), and the file pathname for user metrics summaries (EAR_USER_DB_PATHNAME). [1] In figure 1 the my_mpi_app will be executed with EAR configured with MIN_ENERGY_TO_SOLUTION policy, with a maximum performance degradation set to 10%, the application name used when reporting metrics will be my_mpi_app, the verbose level will be set to 1, and the user database file name use will be $HOME/EAR_outputs/my_db.

[1] To see the whole list of environment variables check the EAR reference document

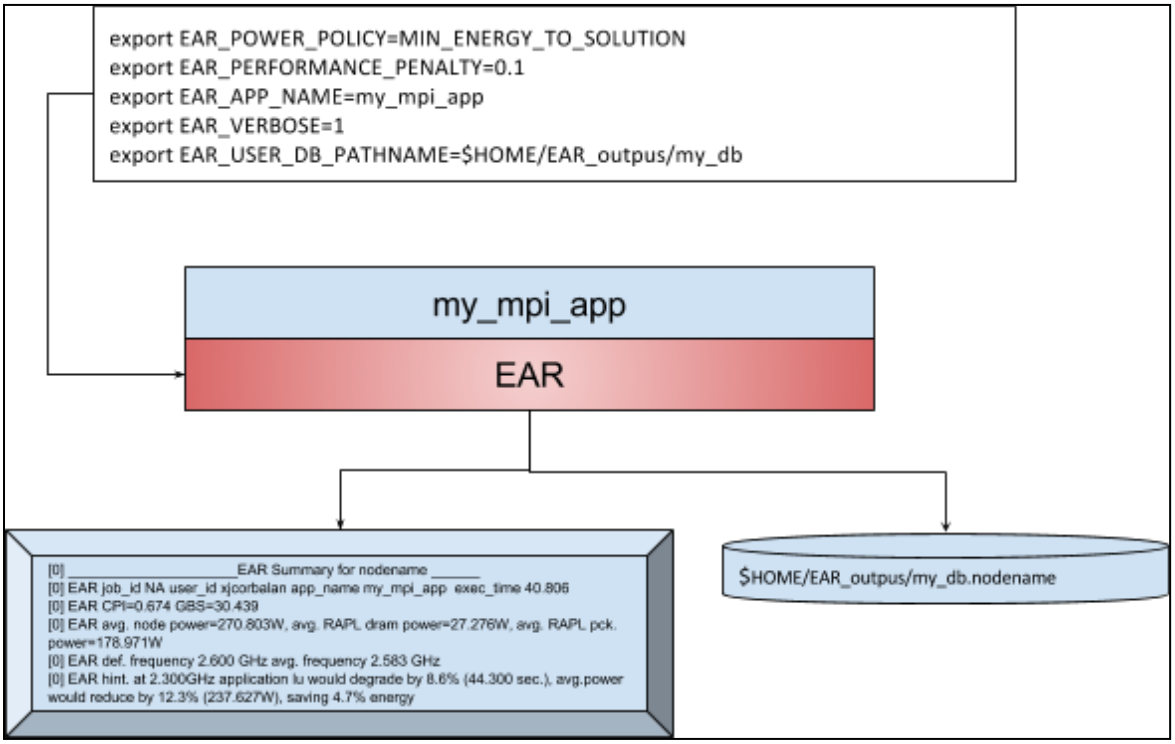


Figure 1: EAR user inputs and outputs

## 1.1 Energy policy selection and configuration

EAR offers three energy policies: MIN_ENERGY_TO_SOLUTION, MIN_TIME_TO_SOLUTION and MONITORING_ONLY. This last option is not a power policy, it is used for application monitoring and CPU frequency is not modified.

The energy policy is selected by setting the EAR_POWER_POLICY user environment variable.

- MIN_ENERGY_TO_SOLUTION: The goal is to minimize the energy consumed with a limit to the performance degradation. The limit in the performance degradation is set in the user defined environment variable EAR_PERFORMANCE_PENALTY. The MIN_ENERGY_TO_SOLUTION policy will select the optimal frequency that minimizes energy enforcing (performance_degradation <= EAR_PERFORMANCE_PENALTY). When executed with MIN_ENERGY_TO_SOLUTION policy, applications starts at nominal frequency.

- MIN_TIME_TO_SOLUTION: The goal is to improve the execution time while guaranteeing a minimum performance efficiency that justifies that energy consumption. The user can specify a percentage indicator for efficiency by setting EAR_MIN_PERFORMANCE_EFFICIENCY_GAIN environment variable. For example, if is set with a value of 0.15, EAR will prevent scaling to upper frequencies if the performance do not improve at least 15% (performance_gain >= EAR_MIN_PERFORMANCE_EFFICIENCY_GAIN). When executed with

MIN_TIME_TO_SOLUTION policy, applications starts at a predefined frequency lower than nominal (EAR_MIN_P_STATE defined at EAR installation time). For example, given a system with a nominal frequency of 2.3GHz and EAR_MIN_P_STATE set to 3, an application executed with MIN_TIME_TO_SOLUTION will start with frequency $F_i$=2.0Ghz (3 p_states less than nominal). When application metrics are computed, the library will compute performance projection for $F_{i+1}$ and will compute the performance gain as shown in figure 2. If performance gain is greater or equal than EAR_MIN_PERFORMANCE_EFFICIENCY_GAIN, the policy will check with the next performance projection $F_{i+2}$. If the performance gain computed is less than EAR_MIN_PERFORMANCE_EFFICIENCY_GAIN, the policy will select the last frequency where the performance gain was enough, preventing the waste of energy.

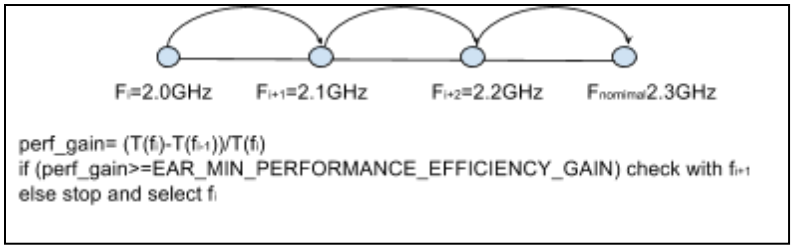


Figure 2: MIN_TIME_TO_SOLUTION uses EAR_MIN_PERFORMANCE_EFFICIENCY_GAIN as the minimum value for the performance gain between between Fi and Fi+1

- MONITORING_ONLY: When selecting this value, EAR is not changing the CPU frequency. This policy is used only with monitoring purposes. When using this policy, the EAR_P_STATE environment variable defines the p_state to be used (default value 1).

## 1.2 Application name

For monitoring purposes, users can specify an application name defining the EAR_APP_NAME environment variable. This name will be used to report summarized metrics.

## 1.3 Configuring the EAR output: metrics and log messages

### 1.3.1 Stderr messages

EAR users can select the “verbose” level by setting the EAR_VERBOSE environment value with the appropriate value. All the verbose messages are generated in the stderr.

**0)** Only errors are reported

**1)** Application granularity is selected. EAR initial configuration is reported (policy,thresholds, application name, etc). Basic information such as frequency changes or application signature changes detected.

**2)** Loop granularity is selected to report information (not per-iteration information)

**3)** Loop granularity with EAR lifecycle changes (EAR states) and metrics

**4)** Iteration and function call granularity is selected. This level is targeted to EAR debugging and validation purposes.

At application end, independently of the verbose level selected, EAR reports in the stderr a summary of metrics only for mpi rank 0. Next figure shows an example of output.

```
[0] _____EAR Summary for nodename _____
[0] EAR job_id job_number user_id user1 app_id  app_name exec_time 40.806
[0] EAR CPI=0.674 GBS=30.439
[0] EAR avg. node power=270.803W, avg. RAPL dram power=27.276W, avg. RAPL pck.
power=178.971W
[0] EAR def. frequency 2.600 GHz avg. frequency 2.583 GHz
[0] EAR hint. at 2.300GHz application lu would degrade by 8.6% (44.300 sec.), avg.power would
reduce by 12.3% (237.627W), saving 4.7% energy
[0] _____
```

When selecting MONITORING_ONLY EAR provides a hint for uses. EAR computes the “optimal” frequency in case MIN_ENERGY_TO_SOLUTION was selected as power policy with a maximum performance degradation of 10%.

## 1.3.2 Per-user metrics DB

EAR also updates a per-user database at application end with summarized metrics. The per-user database is a csv text file with only jobs executed by current user. It can be used for benchmarking or performance analysis. It includes the following fields: USERNAME;JOB_ID;NODENAME;APPNAME;AVG.FREQ;TIME;CPI;TPI;GBS;DC-NODE-POWER;DRAM-POWER;PCK-POWER;DEF.FREQ;POLICY;POLICY_TH

- USERNAME;JOB_ID;NODENAME;APPNAME are reported for application identification.
    - Username as reported by LOGNAME environment variable
    - job_id as reported by SLURM_JOB_ID environment variable (NA otherwise)
    - nodename reported by gethostname system call
    - application name is either EAR_APP_NAME (if available) or executable name.
- FREQ is average frequency computed by ear
- TIME is application execution time. It will be the same in all the nodes
- CPI;TPI;GBS;DC-NODE-POWER;DRAM-POWER;PCK-POWER are average values for the current node
- DEF.FREQ: Is the default frequency at which the application was started

- POLICY;POLICY_TH: In order to a better understanding of metrics, EAR reports the power policy at which application was executed and the policy threshold selected. For MONITORING_ONLY, policy threshold is set to 0.

Each node generates its own information in order to avoid overheads and synchronization. EAR creates one file per node with the name $EAR_USER_DB_PATHNAME.nodename. Merging the different files and using job_id as identifier, the user can compute average power, total energy consumed, etc. The value of $EAR_USER_DB_PATHNAME is used as the prefix for file, adding the nodename automatically. If $EAR_USER_DB_PATHNAME includes subfolder, they must be created before starting the application.

# 2 Executing applications with EAR: SLURM systems

EAR package includes a SLURM [slurm] plugin to simplify the execution of applications it is used as queueing system. This plugin extends the SLURM options to automatically define EAR environment variables. Running an application with both could be done by calling the programs '*sbatch'* and/or '*srun'*.

'sbatch' command receives a script file to execute. This script includes commands like the definition of new environment variables, or any other action that defines the current environment. Finally a call to the 'srun' [srun], which is the SLURM's basic scheduling and execution program, and the responsable of serialize and send the environment to the distributed context, is made, scheduling therefore our binary in the cluster nodes.

Take a look to the section 2.1 to get the information of the parameters added to 'srun' and to the section 8 of the reference manual for a complete list of environment variables to take into account prior to execution.

Take into account that the '*srun*' parameters are converted to the same environment variables, but they exist to simplify that task. So you can configure an execution just modifying the value of the EAR environment variables. An example of a '*sbatch*' shell script could be seen below (figure 3).

```
#!/bin/bash

# Edit architecture values
export MPIS=28
export OMP_NUM_THREADS=1

# Execution
srun --ntasks $MPIS --ear-policy=MIN_ENERGY_TO_SOLUTION \
      --ear-verbose=1 --ear-user-db=$HOME/summaries/sum      \
      --job-name=TEST mpi_test_program
```

Figure 3: Executing an application with EAR using srun ear options

We have highlighted in read EAR flags added to srun options. In this example, EAR is executed with *MIN_ENERGY_TO_SOLUTION* policy, *EAR_VERBOSE* is set to level 1 and the summary file is appended (or just saved if no other summary were found) in '$HOME/summaries/' with the prefix *sum*.

But, as previously said, the 'srun' program, can be used as standalone. It's most used options are described in the following table:

| Argument | Description |
|---|---|
| -n<value>, --ntasks=<value> | Number of tasks of our program to be scheduled.. |
| -N<value>, --nodes=<value> | Number of different nodes in the cluster where the processes will be scheduled. |
| --tasks-per-node=<value> | Number of tasks of our program per node. |
| --job-name=<name> | Defines the job name and also overwrites the environment variable EAR_APP_NAME. |
| --cpu-freq=<value> | Defines the base frequency, in kilohertzs, and is converted in a P_STATE for the EAR_P_STATE environment variable. |

For example, if we want to launch 20 instances of a program, distributed in two nodes with 10 instance in each node, with the policy MONITORING_ONLY and their frequency to 2.5 GHz (P_STATE = 2), the command would be:

‘*srun -N2 -n20 --tasks-per-node=10 --ear-power-policy=MONITORING_ONLY --cpu-freq=2500000 <path.to.program>*’

You can list a complete set of srun options by typing the argument ‘--help’. Also, if you are used to use your MPI job scheduling command, you could launch SLURM to allocate the resources through the bootstrap argument:

‘*mpirun -n $MPIS -hosts node1 -bootstrap slurm -bootstrap-exec /path_to_slurm/srun -bootstrap-exec-args="--exclusive --ear-verbose=1 --ear-user-db=$HOME/summaries/sum" mpi_test_program*‘

These are the widely used commands when mixing MPI job launcher with SLURM:

| Argument | Description |
|---|---|
| -bootstrap <type> | A bootstrap server is the basic remote node access mechanism that is provided by the system. *rsh*, *ssh* and *slurm* are it’s common options. Check your MPI distribution documentation to get a complete list. |
| -bootstrap-exec <path> | Use this option to set the executable to be used as a bootstrap server. Only needed when using a private slurm installation. |
| -bootstrap-exec-args=<args> | Use this option to provide the additional parameters to the bootstrap server executable file |

## 2.1 EAR parameters for SLURM users

These parameters defines the running application with EAR behavior. In fact, all these parameters are converted to environment variables prior to the execution. You can see a complete list of this user oriented parameters below.

| **Argument** | **Description** |
|---|---|
| --ear | Enables Energy Aware Runtime library. It's disabled by default. It is used when no other EAR parameter is used, and uses the default configuration defined in ear.conf SLURM file. |
| --ear-policy=<type> | Selects an energy policy. These policies types could be:<br>- MONITORING_ONLY, this type is just for monitoring purposes<br>- MIN_ENERGY_TO_SOLUTION to reduce energy consumption<br>- MIN_TIME_TO_SOLUTION to reduce the execution time.<br>The default type is MIN_ENERGY_TO_SOLUTION. |
| --ear-policy-th=<value> | Sets a threshold whose function depends on the selected policy. In case MIN_ENERGY policy is selected, the threshold means a performance degradation percentage [0,1] (default: 0.2) above which the next P_STATE frequency wouldn't be selected. In case of MIN_TIME, means a performance scaling percentage [0,1] (default: 0.75) below which the next P_STATE frequency wouldn't be selected, preventing any waste of energy in case the scaling performance isn't worth. |
| --ear-verbose=<value> | Sets the level of verbosity, a number between 0 and 4. More the level more debugging messages. The default value is 0. |
| --ear-user-db=<file pathname> | Sets the path of the summary file. This file is an historic of reports of programs launched next to EAR library. |
| --ear-traces | Generates application traces at runtime with performance and power metrics and application structure internals details detected by EAR. EAR uses a private format (text mode) that can be easily converted to any other format. EAR package includes a tool to convert EAR traces to Paraver traces [paraver] |
| --job-name=<name> | Specifies the name of the application. It's just for monitoring purposes. |
| --cpu-freq=<freq> | Specifies the frequency to use in kilohertzs when using MONITORING_ONLY policy. It is converted to P_STATE by the SLURM plugin. |

# 3 Executing applications with EAR without scheduler support

The ear_scripts/test folder includes some script files that execute well known benchmarks (bt-mz and lu from [naspb-mz] and [naspb-mpi]) with and without EAR. To make it easy, we also offer a simple bash file with two functions to automatically execute an application with EAR: EAR_utils_functions.sh. It offers three functions to automatically execute an application with EAR with the MIN_ENERGY_TO_SOLUTION, MIN_TIME_TO_SOLUTION policies and MONITORING_ONLY. The three functions assumes the user will manually define the EAR_APP_NAME, EAR_USER_DB_PATHNAME and EAR_VERBOSE variables.

**IMPORTANT**: These functions automatically starts one EAR component, the ear_daemon, who requires root privileges (it is executed with sudo). They automatically sets the LD_PRELOAD environment variable to dynamically load the EAR library. Current EAR version only supports MPI applications.

| Function | Description |
|---|---|
| mpirun_min_energy<br>args: binary total_mpis ppn | Executes the application specified in binary with total_mpis MPI processes, ppn process per node. One ear_daemon process is started per node Energy policy is set to MIN_ENERGY_TO_SOLUTION and maximum performance degradation is set to 10%. |
| mpirun_min_time<br><br>args: binary total_mpis ppn | Executes the application specified in binary with total_mpis MPI processes, ppn process per node. One ear_daemon process is started per node Energy policy is set to MIN_TIME_TO_SOLUTION and min performance efficiency gain is set to 75%. |
| mpirun_ear_monitoring<br><br>args: binary total_mpis ppn | Executes the application specified in binary with total_mpis MPI processes, ppn process per node. One ear_daemon process is started per node Energy policy is set to MONITORING_ONLY, CPU frequency is not changed and default frequency is set to the nominal frequency. |
| mpirun_noear<br><br>args: binary total_mpis ppn | Executes the application specified in binary with total_mpis MPI processes, ppn process per node. This function is provided just to compare, it only calls mpiexec.hydra with the specific arguments |

The following code ( figure 4) shows the code for the test_bt-mz.ear_min_energy.sh example included in the $EAR_INSTALL_PATH/ear_scripts/test folder. The script executes the bt-mz kernel included with the EAR distribution with as many MPI processes as total cpus in a single node. This script must be modified to specify the hardware configuration (sockets, cores per socket etc).

./test_bt-mz.ear_min_energy.sh

We have highlighted in red specific lines for EAR. Lines 7-9 defines application name and EAR output file for this application. Line 13 loads the bash file with EAR bash functions and line 15 executes the application with the MIN_ENERGY_TO_SOLUTION preconfigured energy policy. This code is a simplified version since some MPI variables are additionally defined. The script is available at $EAR_INSTALL_PATH/ear_scripts/test/test_bt-mz.ear_min_energy.sh

```
1.  #!/bin/bash
2.
3.  export MPIS=28
4.  export BENCHMARK=$EAR_INSTALL_PATH//bin/kernels/bt-mz.C.$MPIS
5.  export OMP_NUM_THREADS=1
6.
7.  export EAR_APP_NAME=bt-mz
8.  mkdir -p $HOME/.EAR/EAR_OUTS
9.  export EAR_USER_DB_PATH_NAME=$HOME/.EAR/EAR_OUTS/${EAR_APP_NAME}
10.
11. echo "Executing bt-mz(with EAR, MIN_TIME_TO_SOUTION power policy) with $MPIS mpis and $THREADS threads "
12. #adding EAR bash utils functions
13. source ./EAR_utils_functions.sh
14.
15. mpirun_min_time $BENCHMARK $MPIS $MPIS
```

Figure 4: Application is executed with MIN_ENERGY_TO_SOLUTION policy with a maximum performance degradation set to 0.1.

The following code (figure figure 5) shows the code for the test_bt-mz.ear_min_time.sh example included in the $EAR_INSTALL_PATH/ear_scripts/test folder. The script executes the bt-mz kernel included with the EAR distribution with as many MPI processes as total cpus in a single node. This test receives the same arguments than the previous one. It executes bt-mz with MIN_TIME_TO_SOLUTION energy policy with a predefined minimum performance efficiency gain of 75%.

```
1.  #!/bin/bash
2.
3.  export MPIS=28
4.
5.  export BENCHMARK=$EAR_INSTALL_PATH/bin/kernels/bt-mz.C.$MPIS
6.  export OMP_NUM_THREADS=1
7.
8.  export EAR_APP_NAME=bt-mz
9.  mkdir -p $HOME/.EAR/EAR_OUTS
10. export EAR_USER_DB_PATHNAME=$HOME/.EAR/EAR_OUTS/${EAR_APP_NAME}
11.
12. echo "Executing bt-mz(with EAR, MIN_TIME_TO_SOUTION power policy) with $MPIS mpis and $THREADS threads"
13. #adding EAR bash utils functions
14. source ./EAR_utils_functions.sh
15.
16. mpirun_min_time $BENCHMARK $MPIS $MPIS
```

Figure 5: Application is executed with MIN_TIME_TO_SOLUTION policy with a minimum performance gain set to 0.75

Figure 6 shows the same example but using the function to use EAR only for monitoring.

```
1.  #!/bin/bash
2.
3.  export MPIS=28
4.
5.  export BENCHMARK=$EAR_INSTALL_PATH/bin/kernels/bt-mz.C.$MPIS
6.  export OMP_NUM_THREADS=1
7.
8.  export EAR_APP_NAME=bt-mz
9.  mkdir -p $HOME/.EAR/EAR_OUTS
10. export EAR_USER_DB_PATHNAME=$HOME/.EAR/EAR_OUTS/${EAR_APP_NAME}
11.
12. echo "Executing bt-mz(with EAR, MIN_TIME_TO_SOUTION power policy) with $MPIS mpis and $THREADS threads"
13. #adding EAR bash utils functions
14. source ./EAR_utils_functions.sh
15.
16. mpirun_ear_monitoring $BENCHMARK $MPIS $MPIS
```

Figure 6: Application is executed with EAR but CPU frequency is not changed, EAR is only used for monitoring purposes

# 4 References

[slurm] https://slurm.schedmd.com/overview.html

[srun] https://slurm.schedmd.com/srun.html

[naspb-mz] https://www.nas.nasa.gov/assets/pdf/techreports/2003/nas-03-010.pdf

[naspb-mpi] https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf

[paraver] https://www.bsc.es/discover-bsc/organisation/scientific-structure/performance-tools