EAS
Energy Aware Solutions

EAR user guide

EAS S.L.

Contents

1 EAR4.1 user guide

Energy Aware Runtime (EAR) is a system software for energy management. EAR offers energy and performance node monitoring, job accounting and energy optimization. EAR library is the component responsible for energy optimization. EAR library is automatically loaded with MPI applications (Intel and OpenMPI), OpenMP and CUDA applications. For OpenMP and CUDA applications, it is required the application uses dynamic symbols and are submitted with srun.

Visit the complete EAR documentation from **official Wiki** for a complete description of EAR service including admin guide.

This user guide contains information on how to use the EAR library and how to gather EAR collected metrics using EAR commands.

1.1 License

EAR is an open source software and it is licensed under both the BSD-3 license for individual/non-commercial use and EPL-1.0 license for commercial use. Full text of both licenses can be found in COPYING.BSD and COPYING.EPL files.

Contact: support@eas4dc.com

2 Executing jobs with EAR

EAR library is automatically loaded with some applications when EAR is enabled by default. Check with the ear-info application if EAR library is on/off by default. If it's off, use "–ear=on" to enable it or any other EAR flag. This documentation applies to SLURM scheduler, for PBS contact with support@ eas4dc.com

3 Use Cases

3.1 MPI applications

EAR library is automatically loaded with MPI applications when EAR is enabled by defailt (check ear-info). EAR supports the utilization of both mpirun/mpiexec and srun commands. See next sections for examples and more details

3.2 Hybrid MPI +(OpenMP, CUDA, MKL) applications

EAR library automatically supports this use case. Check with the ear-info application if EAR library is on/off by default. If it's off, use "–ear=on" to enable it. mpirun/mpiexec and srun are supported.

3.3 Python (not MPI)

EAR Version 4.1 automatically executes the EAR library with python applications, so no actions are needed. Check with the ear-info application if EAR library is on/off by default. If it's off, use "–ear=on" to enable it. Srun must be used.

3.4 MPI+PYTHON applications

EAR library cannot detect automatically MPI symbols when python is used. In that case, one environment variable to specify intel or "open mpi" must be used.

```
export SLURM_EAR_LOAD_MPI_VERSION=intel or
export SLURM_EAR_LOAD_MPI_VERSION="open mpi"
```

Check with the ear-info application if EAR library is on/off by default. If it's off, use "–ear=on" to enable it. mpirun/mpiexec and srun are supported.

3.5 OpenMP, CUDA, MKL (non-MPI) with EAR library

To load the EAR library automatically with non MPI applications it is required to execute the application with srun and to have compiled the application with dynamic symbols. For example, for cuda application the `--cudart=shared` option must be used.

The EAR library is automatically loaded for OpenMP, MKL and CUDA programming models when symbols are dynamically detected.

3.6 Other application types or frameworks

For other programming models or sequential applications not suported by default, EAR library can be forced to be loaded by setting the environment variable SLURM_EAR_LOADER_APPLICATION defined with the application name. Srun is required for a complete application monitoring.

```
export SLURM_EAR_LOADER_APPLICATION=myapp
srun myapp
```

4 MPI + srun

Running MPI applications with the EAR library is automatic for SLURM systems when using srun. All the jobs are monitored by EAR and the library is loaded by default depending on the cluster configuration. To run a job with srun and the EAR library there is no need to load the EAR module. Even though it's automatic, there are few ear flags than can be selected at job submission. When using slurm commands for job submission both Intel and OpenMPI are supported.

4.1 EAR job submission flags

The following EAR options can be specified when running `srun` and/or `sbatch`, and are supported with `srun`/`sbatch`/`salloc`:

- `--ear=on/off` : Enables/disables EAR library loading with this job.
- `--ear-user-db=file` : Asks the EAR library to generate a set of CSV files with EAR library metrics. One file per node is generated with the average node metrics (node signature) and one file with multiple lines per node is generated with runtime collected metrics (loops node signatures).
- `--ear-verbose=value` : Specifies the level of verbosity {value=[0...1]}; the default is 0. Verbose messages are placed by default in stderr. For jobs with multiple nodes, this option can result in lots of messages mixed in stderr. If the user wants to ask for EAR verbosity, the SLURM_EARL_VERBOSE_PATH env var can be used. One file per node will be generated with EAR messages. The env var must be set with the path (folder name) where files will be generated. The folder is automatically created.

For more information consult `srun --help` output or see configuration options sections for more detailed description.

4.2 CPU frequency selection

The EAR configuration file supports the specification of EAR authorized users who can ask for specific submission options. The more relevant are the possibility to ask for a speficic optimization policy and specific default CPU frequency. To become an EAR priviled user contact with your sysdamin/helpdesk.

- The `--ear-policy=monitoring` flag asks for monitoring.
- The `--ear-cpufreq=CPUfreq` (in KHz)' asks for a specific CPU frequency.

5 MPI + mpirun

To provide an automatic loading of the EAR library, the only requirement from the MPI library is to be coordinated with the scheduler.

5.1 Intel MPI

Recent versions of Intel MPI offers two environment variables that can be used to guarantee the correct scheduler integrations:

- I_MPI_HYDRA_BOOTSTRAP sets the bootstrap server. It must be set to **slurm**.
- I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS sets additional arguments for the bootstrap server. These arguments are passed to slurm.

Click here to read Intel env vars guide.

5.2 OpenMPI

- For OpenMPI and EAR it is highly recommened to use slurm. When using mpirun, as OpenMPI is not fully coordinated with the scheduler, the EAR library is not automatilly loaded on all the nodes. If mpirun is used, the EAR library will be disabled and only basic energy metrics will be reported.

5.3 MPI4PY

- To use MPI with python applications, the EAR loading cannot be automcatically done since the EAR loader cannot detect the symbols to classify the application as Intel or OpenMPI. In order to specify it, the user has to define the SLURM_LOAD_MPI_VERSION with the values intel or ompi. It is recommented to add in python modules to make it easy for final users.

5.4 Using additional MPI profiling libraries/tools

EAR uses the LD_PRELOAD mechanism to be loaded and the PMPI API for a transparent loading. In order to be compatible with other profiling libraries EAR is not replacing the MPI symbols, it just calls the next symbol in the list. So it is compatible with other tools or profiling libraries. In case of conflict, the EAR library can be disabled by selecting `--ear=off` at submission time.

5.5 Examples

5.5.1 `srun` examples

Having an MPI application asking for one node and 24 tasks, the following is a simple case of job submission. If EAR library is on by default, no extra options is needed to load the EAR library. To check if the EAR library is on by default load the EAR module and execute the `ear-info` command. EAR verbose is set to 0 by default (no messages).

```
srun -J test -N 1 -n 24 --tasks-per-node=24 application
```

- The following executes the application showing EAR messages, including EAR configuration and node signature in stderr.

```
srun --ear-verbose=1 -J test -N 1 -n 24 --tasks-per-node=24 application
```

EARL verbose messages are generated in the *stderr*. For jobs using more than 2 or 3 nodes messages can be overwritten. If the user wants to have EARL messages in a file, the `SLURM_EARL_VERBOSE_PATH` environment variable must be set with a folder name. One file per node, job and step will be generated with EARL messages.

```
export SLURM_EARL_VERBOSE_PATH=logs
srun --ear-verbose=1 -J test -N 1 -n 24 --tasks-per-node=24 application
```

- The following asks for EAR library metrics to be generated in csv file after the application execution. Two files per node, job an step will be generated: one with the avg signatures and another with the loop signatures. `filename` text is used as the root for generated files.

```
srun -J test -N 1 -n 24 --tasks-per-node=24 --ear-user-db=filename application
```

- For EAR authorized users only: the following executes the application with CPU frequency 2.0GHz :

```
srun --ear-cpufreq=2000000 --ear-policy=monitoring -J test -N 1 -n 24
     --tasks-per-node=24 application
```

5.6 sbatch + EAR library + srun

When using `sbatch`, EAR options can be specified in the same way. If more than one `srun` call is included in the job submission, EAR options can be inherited from `sbatch` to the different `srun` instances or they can be specifically modified in each individual `srun`.

The following example will execute twice the `application`. Both instances will have the verbosity set to 1. As the job is asking for 10 nodes, we have set the SLURM_EARL_VERBOSE_PATH env var set to the `ear_log` folder. Moreover, the second step will create a set of csv files placed in the `ear_metrics` folder. The nodename, jobid and stepid texts are part of the filename for a better identification.

```
#!/bin/bash
#SBATCH -N 10
#SBATCH -e test.%j.err
#SBATCH -o test.%j.out
#SBATCH --tasks-per-node=24
#SBATCH --cpus-per-task=1
#SBATCH --ear-verbose=1

export SLURM_EARL_VERBOSE_PATH=ear_logs
mkdir ear_metrics
srun  application
srun --ear-user-db=ear_metrics/app_metrics application
```

5.7 EAR library + mpirun

5.7.1 Intel MPI

When running EAR with `mpirun` rather than `srun`, we have to specify the utilisation of `srun` as bootstrap.

Version 2019 and newer offers two environment variables for bootstrap server specification and arguments.

```
export I_MPI_HYDRA_BOOTSTRAP=slurm
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS="--ear-user-db=ear_metrics/app_metrics"
mpiexec.hydra -n 10 application
```

5.7.2 OpenMPI

Bootstrap is an Intel® MPI option but not an OpenMPI option. For OpenMPI `srun` must be used for an automatic EAR support. In case OpenMPI with mpirun is needed, EAR offers the `erun` comman explained below.

5.8 erun

`erun` is a program that simulates all the SLURM and EAR SLURM Plugin pipeline. To use the `erun` load the ear module.

You can launch `erun` with the `--program` option to specify the application name and arguments.

```
mpirun -n 4 erun --ear-verbose=1 --program="hostname --alias"
```

```
> erun --help

This is the list of ERUN parameters:
Usage: ./erun [OPTIONS]

Options:
    --program=<arg> Sets the program to run.
    --clean     Removes the internal files.

SLURM options:
...
```

6 EAR metrics

When using the option –ear-user-db EAR reports application metrics in csv files. Two type of files are generated, one with average metrics per jobid, stepid, and nodename and a second type with runtime collected metrics, also identified by jobid, stepid and nodename but in this case there are some extra columns at the end with the runtime identifications used by EAR : loopid, and iteration number. The columns generated in the csv files concerning mertics are:

- AVG.CPUFREQ: Average CPU frequency (includes all the cores of the node). In KHz
- AVG.IMCFREQ: Average Memory frequency (include the two sockets). In KHz
- DEF.FREQ: default CPU frequency used at the beginning of the application
- TIME: Execution time (in seconds)
- CPI: Cycles per Instructions

- TPI: Main memory transactions per Instruction
- GBS: Main memory bandwith (GB/sec)
- IO_MBS: IO memory bandwith in the node (reads+writes, in MB/sec)
- P.MPI. Average percentage of MPI time vs computational time in the node.
- DC-NODE-POWER: Average DC node power (in Watts)
- DRAM-POWER. Average DRAM power, including the 2 sockets (in Watts)
- PCK-POWER. Average CPU power, including the 2 sockets (in Watts)
- CYCLES. Total Cycles consumed by application processes in the node.
- INSTRUCTIONS. Total instructions executed by application processes in the node.
- GFLOPS: GFlops generated by application processes in the node.
- L1_MISSES: L1 cache misses generated by application processes in the node.
- L2_MISSES: L2 cache misses generated by application processes in the node.
- L3_MISSES: L3 cache misses generated by application processes in the node.
- SP_SINGLE: Floating point operations Single precission 64 bits consumed by application processes in the node.
- SP_128: Floating point operations Single precission 128 bits consumed by application processes in the node.
- SP_256 Floating point operations Single precission 256 bits consumed by application processes in the node.
- SP_512: Floating point operations Single precission 512 bits consumed by application processes in the node.
- DP_SINGLE: Floating point operations Double precission 64 bits consumed by application processes in the node.
- DP_128: Floating point operations Double precission 128 bits consumed by application processes in the node.
- DP_256: Floating point operations Double precission 256 bits consumed by application processes in the node.
- DP_512: Floating point operations Double precission 512 bits consumed by application processes in the node.