EAR quick user guide

EAR is a system software for energy management, accounting and optimization for super computers. This document provides a quick overview about how to use it services at the user level. The complete documentation can be found at the official EAR wiki. You can also find useful tutorials there.

Below there is a list of specific details about the EAR software used in this guide:

- Version: 5.1

> **Note** This user guide contains examples of job submission which may not fit exactly to submission instructions specified by your cluster documentation. Please, refer to it to adapt batch job scripts showed here.

Job submission

With the EAR SLURM SPANK plug-in, running an application with EAR is as easy as submitting a job with either `srun`, `sbatch` or `mpirun`. The EAR runtime Library (EARL) is automatically loaded with some applications when EAR is enabled by default.

Therefore, application performance metrics and power consumption can be gathered online without code modification. Moreover, you can enable energy optimization policies for your workloads.

Automatically full supported use cases

MPI applications (Including MPI + OpenMP/CUDA/MKL)

- Using **sbatch** + **srun**: The job submission with EAR is totally automatic.

There are some EAR options that can be requested at submission time:

```
srun --help | grep ear
```

If multiple steps are submitted in the same job, different flags for each one can be used.

The following example executes two steps. First one uses default flags and second one asks ear to report ear metrics in a set of csv files. `ear_metrics`/`app_metrics` is used as the root of filenames generated.

```
#!/bin/bash
#SBATCH -N 10
#SBATCH -e test.%j.err -o test.%j.out
```

```
#SBATCH --tasks-per-node=24 --cpus-per-task=1
#SBATCH --ear=on

module load mpi

mkdir ear_metrics
# run application with ear's default flags.
srun -n $SLURM_NTASKS application

# run application and store ear metrics in ear_metrics/app_metrics.*.csv
srun --ear-user-db=ear_metrics/app_metrics application
```

- Using Intel's **mpirun**: When running EAR with `mpirun` rather than `srun`, you must specify the utilisation of `slurm` as the bootstrap server.

Since Intel MPI-2019 version there are two environment variables for bootstrap server specification and arguments.

- `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.

The previous example batch script can be written (for an Intel MPI application) as:

```
module load impi

export I_MPI_HYDRA_BOOTSTRAP=slurm
export I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS="--ear-user-db=ear_metrics/app_metrics"
mpiexec.hydra -n 64 application
```

- Using OpenMPI's **mpirun**: It is recommened to use `srun` for OpenMPI applications.

If OpenMPI's `mpirun` is used instead, EAR will report just job accounting metrics (DC Node Power and execution time of the job). If you want to enable the EARL monitoring and optimization features, you must use EAR's `erun` command before running your application binary. In this case, you must **load the ear module** installed in the system.

The tool accepts same flags as `sbatch`/`srun` commands. In addition, the `--program` flag is used to specify the application you want to run. See the following example:

```
#!/bin/bash
#----------------------------------------------------
# Example SLURM job script with SBATCH requesting GPUs
#----------------------------------------------------
#SBATCH --job-name=gromacs
#SBATCH -o slurm_output.%j
```

```
#SBATCH -e slurm_error.%j
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=8
#SBATCH --time=02:00:00
#SBATCH --exclusive
#SBATCH --gres=gpu:4

module load GROMACS/2024.2-foss-2023b
module load ear

mpirun  erun --ear-verbose=1 --program="gmx_mpi mdrun -ntomp 8 -nb gpu
   -pme gpu -npme 1 -update gpu -bonded gpu -nsteps 100000 -resetstep
   90000 -noconfout -dlb no -nstlist 300 -pin on -v -gpu_id 0123"
```

Non-MPI applications (CUDA, OpenMP, MKL and Python)

In order to enable EAR monitoring and optimization features for non-MPI applications, it is required to run the application with the `srun` command. For *CUDA*, *OpenMP* and *MKL* applications, the binary must have been linked with dynamic symbols (e.g., `--cudart=shared`). Below there is an example enabling EAR with an OpenMP application.

```
#!/bin/bash

#SBATCH -N 1 -n 1 --cpus-per-task=64
#SBATCH --ear=on --ear-verbose=1

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun -n $SLURM_NTASKS -c $OMP_NUM_THREADS ./bt.D.x
```

An example running a Python application:

```
#!/bin/bash

#SBATCH -N 1 -n 1 --cpus-per-task=64
#SBATCH --ear=on --ear-verbose=1

srun -n $SLURM_NTASKS -c $SLURM_CPUS_PER_TASK python script.py
```

Other use cases supported

Python MPI applications

EAR can't detect MPI symbols when Python is used, so an environment variable is needed to specify which MPI flavour is being used.

```
module load ompi

export EAR_LOAD_MPI_VERSION="open mpi"

srun -n 64 --ear-user-db=ear_metrics/app_metrics python script.py
```

For applications which use IntelMPI, the value must be *intel*.

Other application types or frameworks

For other programming models or sequential apps not supported by default, EAR can be loaded by setting `EAR_LOADER_APPLICATION` enviroment variable:

```
export EAR_LOADER_APPLICATION=/full/path/to/my_app

srun --ear-user-db=ear_metrics/app_metrics my_app
```

Job accounting (eacct)

The `eacct` command shows accounting information stored in the EAR DB for jobs (and step) IDs. You must first load the **ear** module. Here we list the most useful command flags:

- `-j <job_id>[.step_id]`: Specify the job (and optionally, the step) you want to retrieve information.
- `-a <job_name>`: Specify the application name that will be retrieved.
- `-c <filename>`: Store the output in csv format in <filename>. Fields are separated by `;`.
- `-l`: Specify you want job data for each of the used computation nodes.
- `-r`: Request loop signatures instead of global application metrics. **EAR loop reporting must be enabled through `EARL_REPORT_LOOPS` environment variable if not automatic loop reporting is enabled in your system.** Just set it to a non-zero value.
- `-s <YYYY-MM-DD>`: Specify the minimum start time of the jobs that will be retrieved.
- `-e <YYYY-MM-DD>`: Specify the maximum end time of the jobs that will be retrieved.

Examples

The basic usage of eacct retrieves the last 20 applications (by default) of the user executing it. The default behaviour shows data from each job-step, aggregating the values from each node in said job-step. If using SLURM as a job manager, a sb (sbatch) job-step is created with the data from the entire execution. A specific job may be specified with `-j` option:

- [user@host EAR]$ eacct –> Shows last 20 jobs (maximum) executed by the user.
- [user@host EAR]$ eacct -j 175966 –> Shows data for jobid = 175966. Metrics are averaged per job.stepid.
- [user@host EAR]$ eacct -j 175966.0 –> Shows data for jobid = 175966 stepid=0. Metrics are averaged per job.stepid.
- [user@host EAR]$ eacct -j 175966,175967,175968 –> Shows data for jobid = 175966, 175967, 175968 Metrics are averaged per job.stepid.

Eacct shows a pre-selected set of columns. Some flags sligthly modifies the set of columns reported:

- JOB-STEP: JobID and Step ID. sb is shown for the sbatch.
- USER: Username who executed the job.
- APP=APPLICATION: Job's name or executable name if job name is not provided.
- POLICY: Energy optimization policy name (MO = Monitoring).
- NODES: Number of nodes which ran the job.
- AVG/DEF/IMC(GHz): Average CPU frequency, default frequency and average uncore frequency. Includes all the nodes for the step. In KHz.
- TIME(s) : Step execution time, in seconds.
- POWER Average node power including all the nodes, in Watts.
- GBS : CPU Main memory bandwidth (GB/second). Hint for CPU/Memory bound classification.
- CPI : CPU Cycles per Instruction. Hint for CPU/Memory bound classification.
- ENERGY(J) : Accumulated node energy. Includes all the nodes. In Joules.
- GFLOPS/WATT : CPU GFlops per Watt. Hint for energy efficiency.
- IO(MBs) : IO (read and write) Mega Bytes per second.
- MPI% : Percentage of MPI time over the total execution time. It's the average including all the processes and nodes.
- GPU metrics
  - G-POW (T/U) : Average GPU power. Accumulated per node and average of all the nodes.
    * T= Total (GPU power consumed even if the process is not using them).
    * U = GPUs used by the job.
  - G-FREQ : Average GPU frequency. Per node and average of all the nodes.
  - G-UTIL(G/MEM) : GPU utilization and GPU memory utilization.

The following example shows how to submit a job with EAR monitoring enabled. It also shows how to enable loop signatures reporting and finally how to request the data.

```
#!/bin/bash
#SBATCH -J test
#SBATCH -N 1
#SBATCH --ntasks=112
#SBATCH --cpus-per-task=1


#SBATCH --ear=on
#SBATCH --ear-user-db=metrics


module purge
module load impi

export EARL_REPORT_LOOPS=1
srun ./bt-mz.D.impi
```

Using *eacct* to retrieve loop signatures:

```
[user@host bin]$ module load ear
[user@host bin]$ eacct -j 3180887 -r
   JOB-STEP NODE ID   DATE     POWER(W) GBS/TPI CPI   GFLOPS/W TIME(s)
      AVG_F/F  IMC_F IO(MBS) MPI% G-POWER(T/U)  G-FREQ G-UTIL(G/MEM)
3180887-0   gs02r3b66 09:08:12 825.6    156/17  0.277 0.619    1.013
   2.52/2.0 1.81  0.0     4.2  0.0   /   0.0 0.00   0%/0%
3180887-0   gs02r3b66 09:08:24 969.7    157/17  0.277 0.527    1.240
   2.51/2.0 1.81  0.0     3.6  0.0   /   0.0 0.00   0%/0%
3180887-0   gs02r3b66 09:08:47 906.7    157/17  0.277 0.563    1.127
   2.51/2.0 1.81  0.0     3.8  0.0   /   0.0 0.00   0%/0%
3180887-0   gs02r3b66 09:09:09 909.1    157/17  0.277 0.561    1.126
   2.51/2.0 1.81  0.0     3.7  0.0   /   0.0 0.00   0%/0%
```

Using *eacct* to retrieve job signature:

```
[user@host bin]$ eacct -j 3180887
    JOB-STEP USER      APPLICATION POLICY NODES AVG/DEF/IMC(GHz) TIME(s
       ) POWER(W) GBS    CPI  ENERGY(J) GFLOPS/W IO(MBs) MPI% G-POW (T/
       U) G-FREQ G-UTIL(G/MEM)
3180887-sb   user           test        NP     1     2.61/2.00/---
   120.00  874.49   ---    ---  104939    ---      ---     ---  ---
            ---    ---
3180887-0    user        test        MO     1     2.52/2.00/1.81
   97.72   913.51   157.12 0.28 89268     0.5578   0.0     3.7
   0.00/---    ---    ---
```

As through *srun* command both Intel MPI and OpenMPI implementations are compatible, below you can see a very similar example script which runs an OpenMPI application:

```
#!/bin/bash
#SBATCH -J test
#SBATCH -N 1
#SBATCH --ntasks=112
#SBATCH --cpus-per-task=1

#SBATCH --ear=on

module purge
module load openmpi/4.1.5

srun ./bt-mz.D.ompi
```

Data visualization

ear-job-visualizer

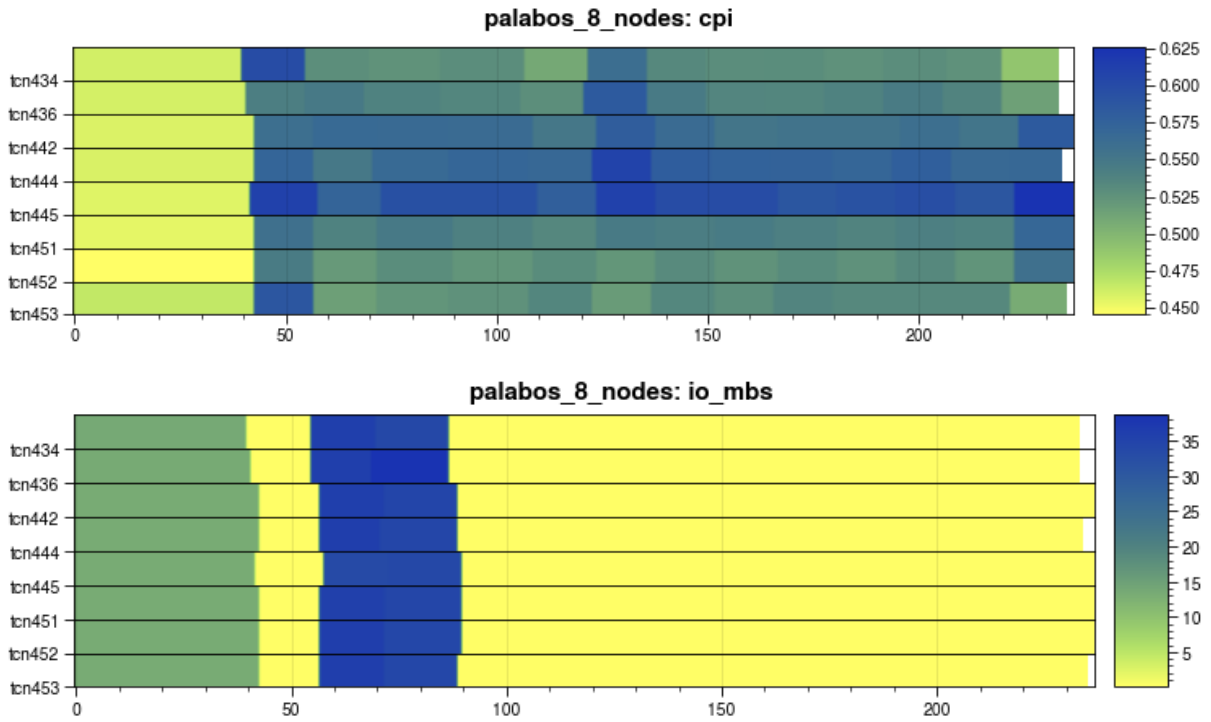
This is a tool which lets you generate either **static images** or **Paraver trace files** directly from EAR data. If it is installed on a system with EAR full installed, the tool calls internally the *eacct* command to retrieve and build timelines for the requested job and step id. The complete user guide can be found at the official, maintained repository.

```
$> module load eas-tools
$> cpu_metrics="cpi gflops avg_cpufreq avg_imcfreq gbs dc_power"
$> gpu_metrics="gpu_power gpu_freq gpu_memfreq gpu_util gpu_memutil"
$> ear-job-visualizer --format runtime -j 6043213 -s 0 -t
    palabos_8_nodes -o palabos_8.png -m $cpu_metrics $gpu_metrics
```

After that, you will get the following image files:

```
$> ls *palabos_8*
runtime_cpi-palabos_8.png  runtime_dc_power-palabos_8.png  runtime_gbs-
    palabos_8.png  runtime_gflops-palabos_8.png  runtime_io_mbs-
    palabos_8.png  runtime_pck_power-palabos_8.png  runtime_perc_mpi-
    palabos_8.png
```

Graphs look like this:



Below there is an example on how to generate a **Paraver trace** for the of the same Job and Step:

```
$> ear-job-visualizer --format ear2prv -j 6043213 -s 0 -o palabos_8
$> ls
palabos_8.pcf palabos_8.prv palabos_8.row
```

You can download CPU metrics configuration file and GPU metrics configuration file.

Grafana

You can read on the wiki how to visualize EAR metrics in Grafana Dashboards.