Tutorial 1: Performance analysis for High Performance Systems

Objectives

- Yet another performance analysis tool

- Developping performance analysis features for your application/library

Contents

- Introduction
- Overview of EZTrace workflow
- Analyzing an MPI application
- Analyzing an MPI + OpenMP application
- Developping a plugin

Who are we ?

François Trahay
EZTrace project leader
Associate professor
Télécom SudParis

François Rue

Research Engineer
INRIA Bordeaux

Mathias Hastaran

Research Engineer
INRIA Bordeaux

Before we start

The materials for this tutorial are available here:

http://eztrace.gforge.inria.fr/eurompi2015

You should have received an email with information on your temporary account on the Plafrim cluster

Introduction

□ Modern HPC applications are complex
  - Complex hardware
    - NUMA architecture, hierarchical caches, accelerators
  - Hybrid programming models
    - MPI + [OpenMP | Pthread | CUDA]

→ Understanding the performance of such applications is difficult

→ Need for performance analysis tools

Performance analysis tools
Profiling tools

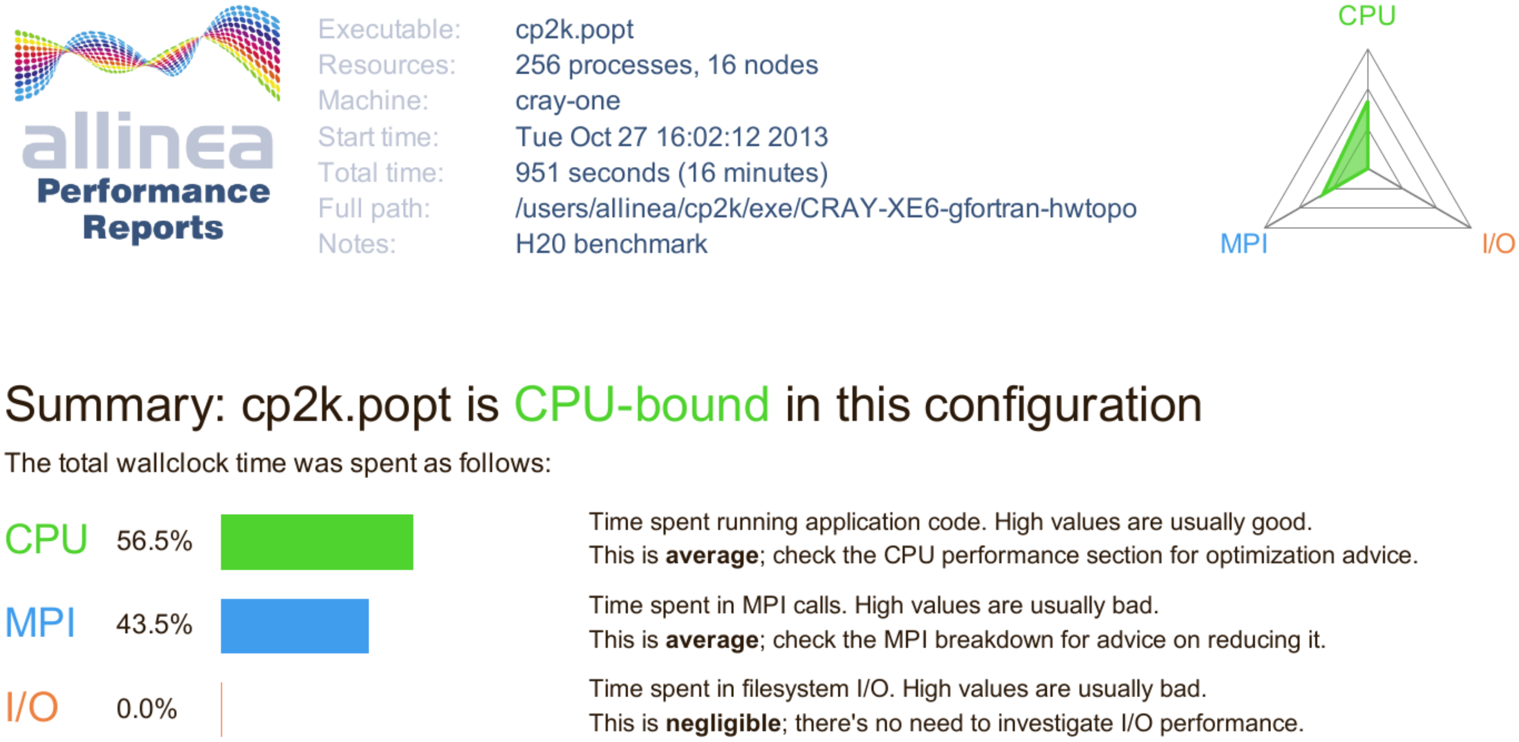
- □ Gather statistical information on the application
  - – Allinea MAP, gprof, mpiP, …

```
$ gprof ./sgefa_openmp
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 49.68      4.21     4.21     3283     0.00     0.00  sswap
 31.51      6.89     2.67     1107     0.00     0.00  msaxpy2
 17.47      8.37     1.48   511146     0.00     0.00  saxpy
  0.94      8.45     0.08        9     0.01     0.01  matgen
  0.47      8.49     0.04        3     0.01     0.50  sgefa
  0.00      8.49     0.00     3321     0.00     0.00  isamax
[...]
```

Performance analysis tools

Profiling tools

- Gather statistical information on the application
  - Allinea MAP, gprof, mpiP, …



Executable: cp2k.popt
Resources: 256 processes, 16 nodes
Machine: cray-one
Start time: Tue Oct 27 16:02:12 2013
Total time: 951 seconds (16 minutes)
Full path: /users/allinea/cp2k/exe/CRAY-XE6-gfortran-hwtopo
Notes: H20 benchmark

Summary: cp2k.popt is CPU-bound in this configuration

The total wallclock time was spent as follows:

CPU 56.5% Time spent running application code. High values are usually good.
This is **average**; check the CPU performance section for optimization advice.

MPI 43.5% Time spent in MPI calls. High values are usually bad.
This is **average**; check the MPI breakdown for advice on reducing it.

I/O 0.0% Time spent in filesystem I/O. High values are usually bad.
This is **negligible**; there's no need to investigate I/O performance.

Performance analysis tools

Tracing applications

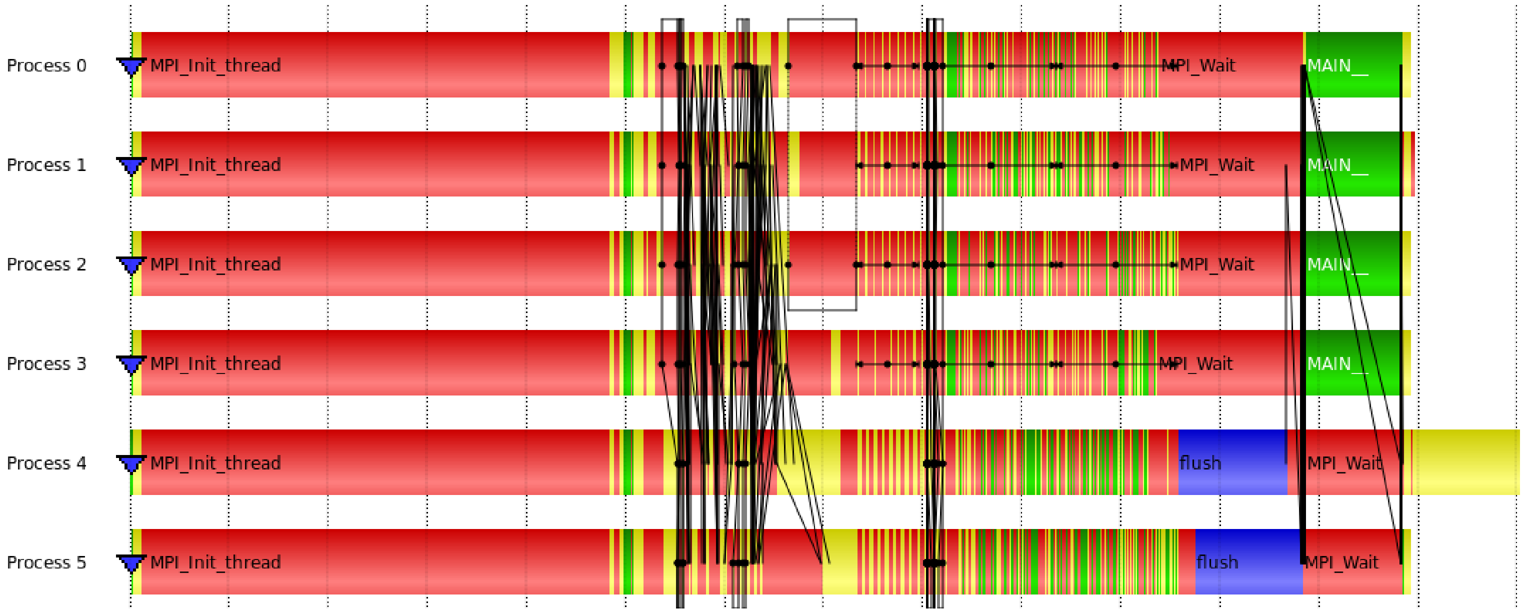
- Collect a list of timestamped events
  - Tau, VampirTrace, Scalatrace, Intel Trace Analyzer and Collector, EZTrace, …

```
#timestamp  #ThreadId  #Event
0.00175s        1      Enter function Foo(arg1=17)
0.20573s        1      Enter function Bar(n=42.23)
0.21248s        2      Enter function Baz(a=21, b=40)
0.31054s        2      Leave function Baz(a=21, b=40) return value=91
0.61057s        1      Leave function Bar(n=42.23) return value=124.89
[...]
```

Performance analysis tools

Tracing applications

- □ Collect a list of timestamped events
  - Tau, VampirTrace, Scalatrace, Intel Trace Analyzer and Collector, EZTrace, …



EZTrace

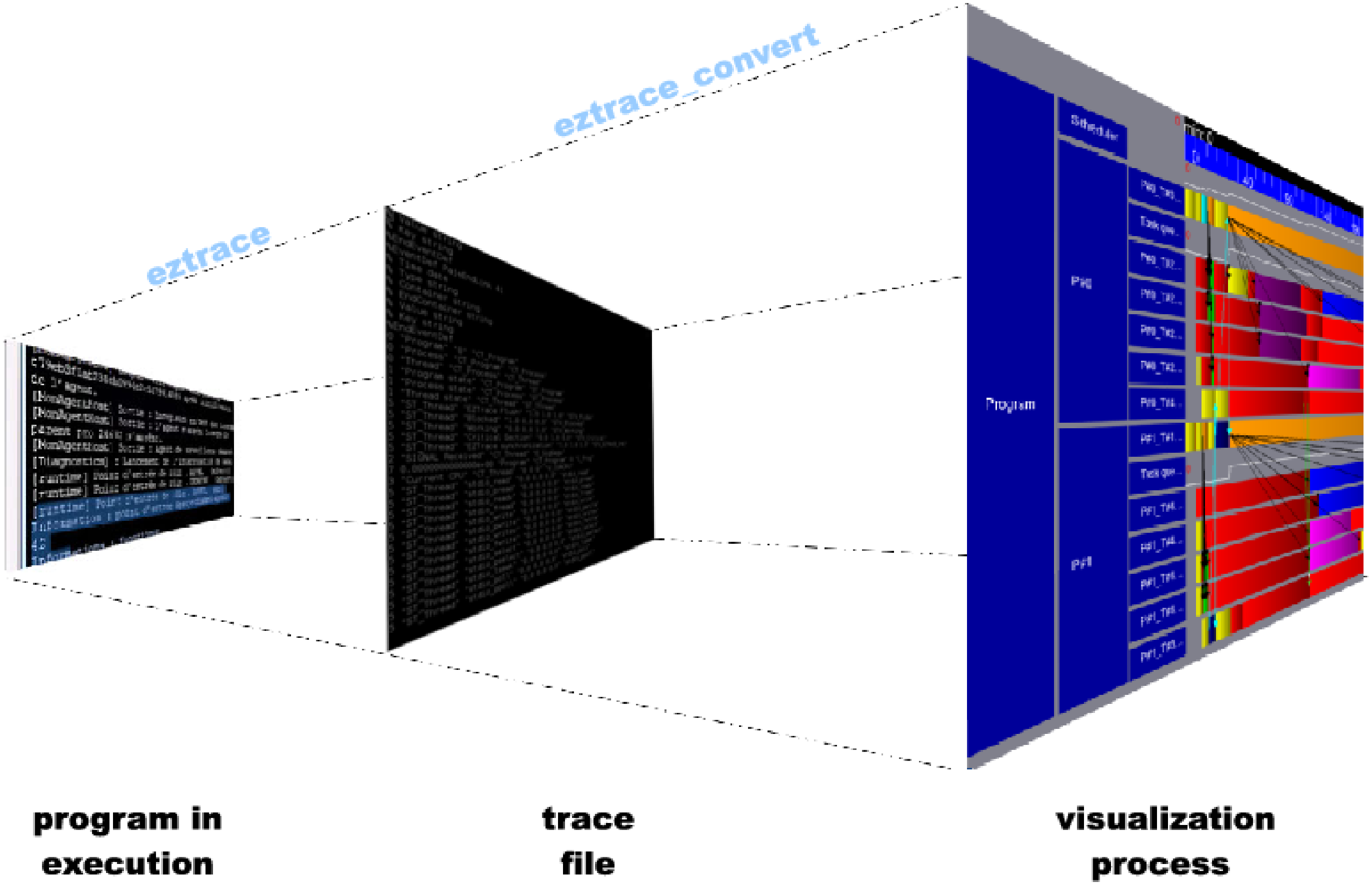
- Framework for performance analysis
  - Provides tracing facilities
  - Provides pre-defined modules (MPI, OpenMP, CUDA, etc.)
  - Allows external modules
    - Develop your own module
    - Use a module shipped with a library (eg. PLASMA)
  - Uses standard file formats (OTF, Pajé)
  - Open source (~BSD license)

http://eztrace.gforge.inria.fr/

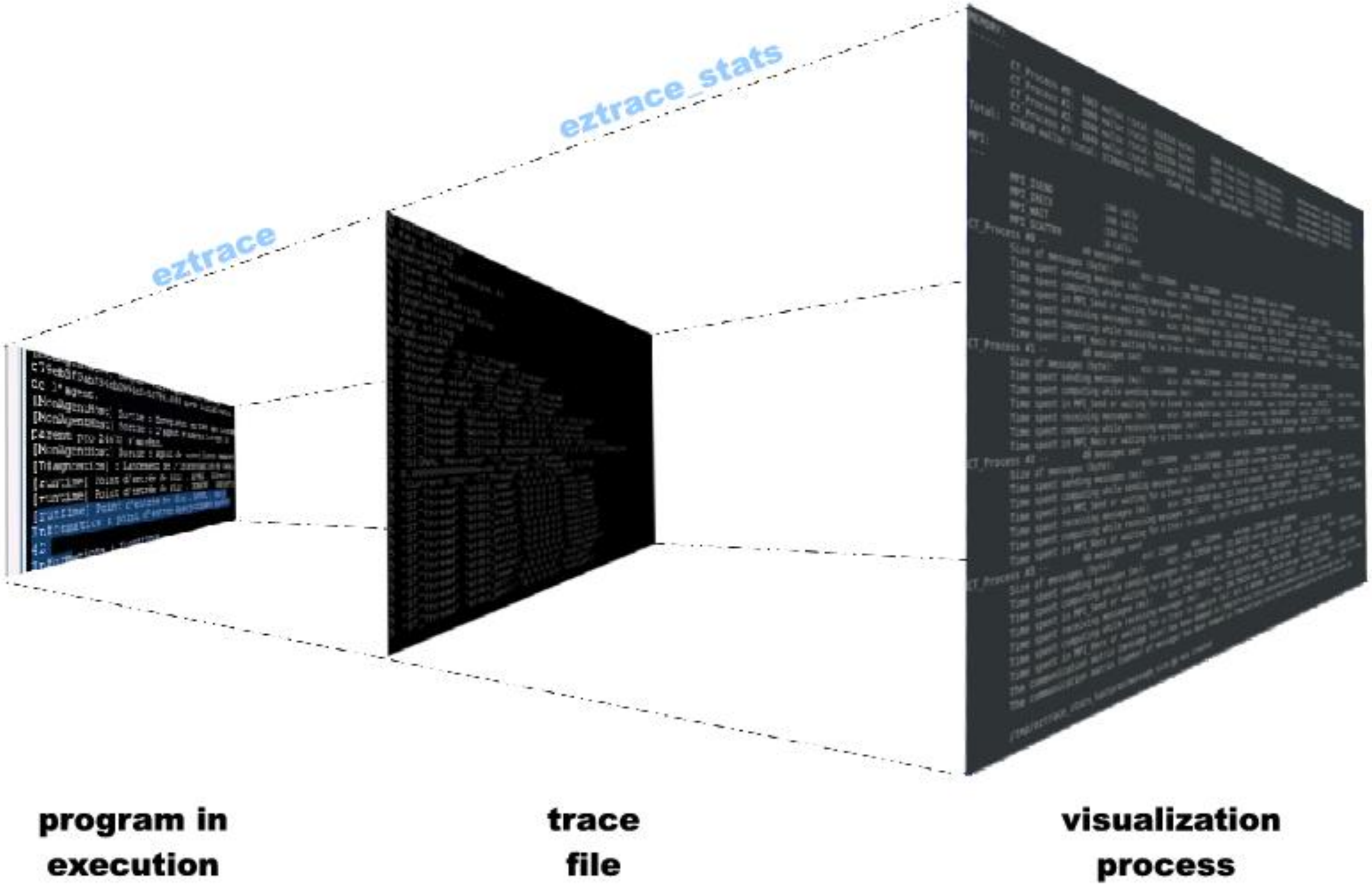
Contents

- □ Introduction
- □ Overview of EZTrace workflow
- □ Analyzing an MPI application
- □ Analyzing an MPI + OpenMP application
- □ Developping a plugin

Overview of EZTrace workflow



Overview of EZTrace workflow



Running an application with EZTrace

- Select the modules to load

```
$ eztrace_avail
3       stdio   Module for stdio functions (read, write, select, poll, etc.)
2       pthread Module for PThread synchronization functions (mutex, semaphore, spinlock, etc.)
1       omp     Module for OpenMP parallel regions
4       mpi     Module for MPI functions
5       memory  Module for memory functions (malloc, free, etc.)
6       papi    Module for PAPI Performance counters
7       cuda    Module for cuda functions (cuMemAlloc, cuMemcopy, etc.)
10      starpu  Module for the StarPU framework

$ export EZTRACE_TRACE="pthread"

$ eztrace_loaded
2       pthread Module for PThread synchronization functions (mutex, semaphore, spinlock, etc.)
```

Running an application with EZTrace

- □ Run the application

```
$ eztrace ./heat_pthread  100 100 50 1
Starting EZTrace... Done
[...]
Stopping EZTrace... saving trace  /tmp/trahay_eztrace_log_rank_1

$ eztrace.preload ./heat_pthread 100 100 50 1
Starting EZTrace... Done
[...]
Stopping EZTrace... saving trace  /tmp/trahay_eztrace_log_rank_1
```

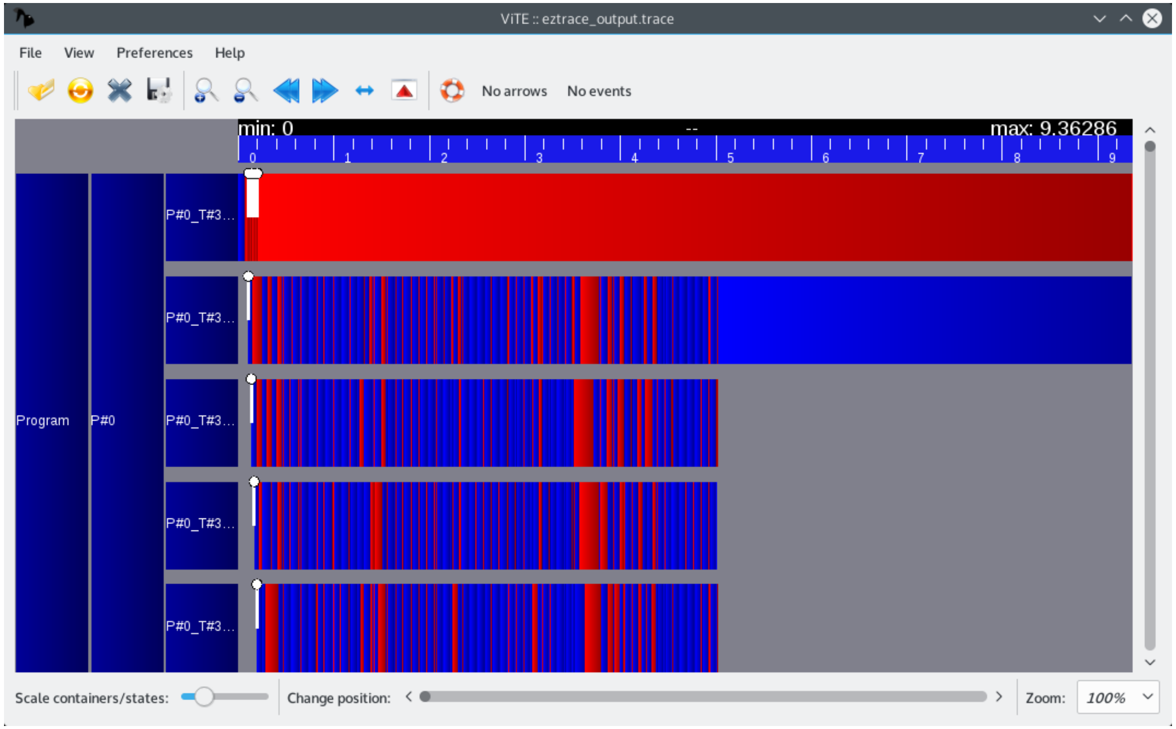
- Intercept the calls to a set of functions
  - Intercept calls to shared libraries (using LD_PRELOAD)
  - Modify the binary to insert hooks (only with `eztrace`)
- Record timestamped events in trace files
- Create one file per process

Post-mortem analysis

□ Visualizing the trace

```
$ eztrace_convert /tmp/trahay_eztrace_log_rank_1
module pthread loaded
1 modules loaded
no more block for trace #0
833 events handled
$ vite eztrace_output.trace
```

- Read the traces and interpret events
- Creates the output file: `eztrace_output.[trace|otf]`
- Visualize the trace with standard tools (Vampir, ViTE, etc.)



Post-mortem analysis

□ Getting statistics

```
$ eztrace_stats /tmp/trahay_eztrace_log_rank_1
PThread:
-------
CT_Process #0:
        semaphore 0x0x601f40 was acquired 4 times. total time spent waiting: 0.089913 ms.
        barrier 0x0x601f00 was acquired 400 times. total time spent waiting: 4.499698 ms.
Total: 2 locks acquired 404 times
Thread P#0_T#3711915776
        time spent waiting on a semaphore: 0.089913 ms
Thread P#0_T#3665626880
        time spent waiting on a barrier: 1.159355 ms
Thread P#0_T#3514812160
        time spent waiting on a barrier: 1.159498 ms
Total for CT_Process #0
        time spent waiting on a semaphore: 0.089913 ms
        time spent waiting on a barrier: 4.499698 ms
PTHREAD_CORE
------------
Thread P#0_T#3711915776:
        time spent in pthread_join   : 9.158800 ms
        time spent in pthread_create: 0.044299 ms
Total for CT_Process #0
        time spent in pthread_join   : 9.158800 ms
        time spent in pthread_create: 0.044299 ms
812 events handled
```

Hands-on

- Connection to plafrim

```
$ emacs ~/.ssh/config
Host formation
    ForwardAgent yes
    ForwardX11 yes
    User eurompi2015-trahay
    ProxyCommand ssh -A -l login@formation.plafrim.fr -W plafrim:22
$ ssh formation
```

- Accessing a node of the cluster

```
(plafrim) $ module load slurm
(plafrim) $ salloc --share -N 4
(plafrim) $ echo $SLURM_JOB_NODELIST
miriel[078-081]
(plafrim) $ ssh miriel078
```

- http://eztrace.gforge.inria.fr/eurompi2015
  - Exercice 1: *Introduction to EZTrace*

Analyzing an MPI application with EZTrace

- Run the application with eztrace

```
$ export EZTRACE_TRACE=mpi
$ mpirun -np 4 eztrace ./application arg1 arg2
or
$ mpirun -np 4 eztrace -t mpi ./application arg1 arg2
or

$ mpirun -np 4 $(eztrace.preload -t mpi ./application arg1 arg2)
```

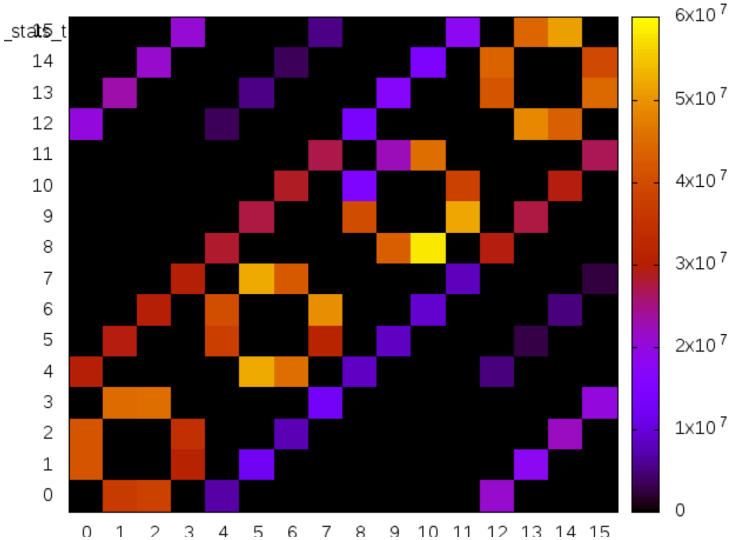
  - Generates one trace per process
  - Each MPI process write in its `/tmp` directory

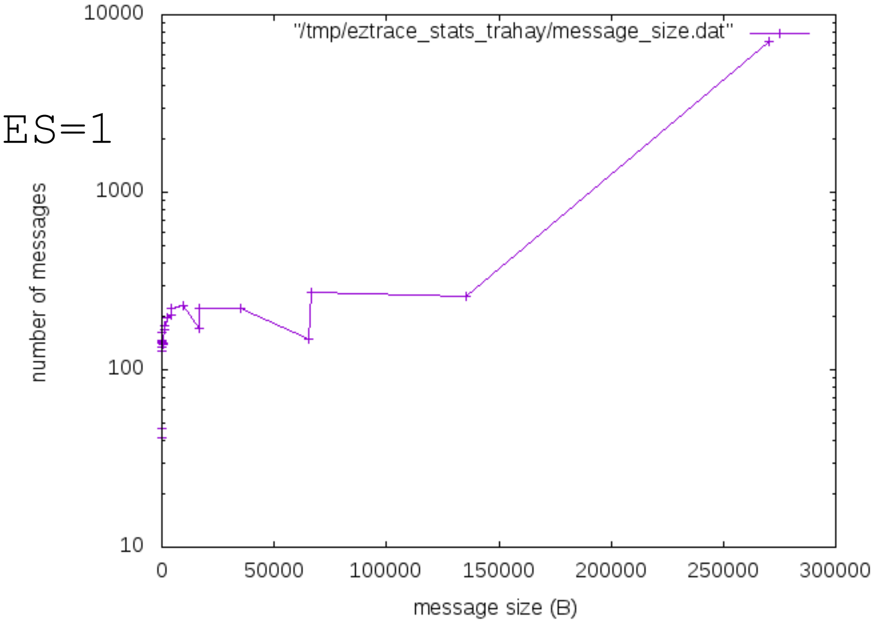
    → `export EZTRACE_TRACE_DIR=$PWD`

MPI statistics

- `eztrace_stats` dumps information on MPI messages
  - Communication matrix
  - Distribution of message sizes
  - List of *all* the messages

→ `export EZTRACE_MPI_DUMP_MESSAGES=1`





Analyzing an OpenMP application with EZTrace

- OpenMP relies on compiler directives
  - Need to recompile the application with `eztrace_cc`

```
$ make CC=’’eztrace_cc gcc’’
[...]
$ eztrace −t omp ./application
```

Analyzing an MPI+OpenMP application

□ Simply select the `mpi` and `omp` modules

```
$ make MPICC=’’eztrace_cc mpicc’’
[...]
$ mpirun −np 4 eztrace −t ’’mpi omp’’ ./application
```

Hands-on part 2: MPI

- Connection to plafrim

```
$ emacs ~/.ssh/config
Host formation
    ForwardAgent yes
    ForwardX11 yes
    User eurompi2015-trahay
    ProxyCommand ssh -A -l login@formation.plafrim.fr -W plafrim:22
$ ssh formation
```

- Accessing a node of the cluster

```
(plafrim) $ module load slurm
(plafrim) $ salloc --share -N 4
(plafrim) $ echo $SLURM_JOB_NODELIST
miriel[078-081]
(plafrim) $ ssh miriel078
```

- http://eztrace.gforge.inria.fr/eurompi2015
  - Exercice 2: Using EZTrace for MPI applications

EZTrace thrid-party modules

- EZTrace is a framework for performance analysis
  - Allow third-party modules
    - Analyze your application/library
    - Ship an EZTrace module with your library (eg. PLASMA)

- An EZTrace module consists of
  - A library that intercepts a set of functions and record events
  - A library that interprets events

Module generator

- `eztrace_plugin_generator`
  - Search for symbols in a binary application (C / Fortran)
  - Search for the prototypes of the functions
  - Generates a `.tpl` file for these functions

```
$ eztrace_plugin_generator heat_mpi
Creating the plugin script heat_mpi.tpl
        Found 'void ghosts_swap (MPI_Comm comm, MPI_Datatype col, const int *neighbours, int 
size_x, int size_y, double *u)'
        Found 'void print_mat (int size_x, int size_y, const double *u)'
        Found 'void save_mat (const char *filename, int size_x, int size_y, const double *u)'
        Found 'void set_bounds (const int *coo, int nc_x, int nc_y, int size_x, int size_y, 
double *u)'
        Found 'void usage (char *argv[])'
5 symbols found

Generating the plugin...
        $ eztrace_create_plugin -o plugin_heat_mpi heat_mpi.tpl

Compiling the plugin...
        $ make -C plugin_heat_mpi
```

Creating a module from a .tpl file

- Describe the module
  - Name/description
  - List of functions to intercept
  - Actions to perform for each function
    - `EVENT("Do function foo“)`
    - `PUSH_STATE("doing function foo")`
    - `POP_STATE()`
    - `SET_VAR("var_name", value)`
    - `ADD_VAR("var_name", value)`
    - `SUB_VAR("var_name", value)`

```
BEGIN_MODULE
NAME heat_mpi
DESC "Module for the heat_mpi program"

void print_mat (const double *u)

void save_mat (const char *f, const double *u)
BEGIN
 ADD_VAR("variable name", 1)
END

int foo(int a, int b)
BEGIN
  PUSH_STATE("Doing function foo")
  CALL_FUNC
  POP_STATE()
END

END_MODULE
```

```
$ eztrace_create_plugin -o plugin_heat_mpi heat_mpi.tpl
$ make -C plugin_heat_mpi
$ export EZTRACE_LIBRARY_PATH=$PWD/plugin_heat_mpi
```

Tuning a module

- Objective: collect the exact information you're looking for
  - eg. average duration of function `void foo(int a, int b)` when `b>a`

- Edit the `eztrace_convert_*.c` file generated by `eztrace_create_plugin`

- Per-thread/per-process statistics

Hands-on part 3: creating EZTrace modules

- Connection to plafrim

```
$ emacs ~/.ssh/config
Host formation
    ForwardAgent yes
    ForwardX11 yes
    User eurompi2015-trahay
    ProxyCommand ssh -A -l login@formation.plafrim.fr -W plafrim:22
$ ssh formation
```

- Accessing a node of the cluster

```
(plafrim) $ module load slurm
(plafrim) $ salloc --share -N 4
(plafrim) $ echo $SLURM_JOB_NODELIST
miriel[078-081]
(plafrim) $ ssh miriel078
```

- http://eztrace.gforge.inria.fr/eurompi2015
    - Exercice 3: Creating an EZTrace module

Thank you !

- EZTrace is open-source
  - CeCILL-B (~BSD) license
  - Contribution / collaboration are welcome !

http://eztrace.gforge.inria.fr/

eztrace-devel@lists.gforge.inria.fr