# EZTrace: a generic framework for performance analysis

Francois Trahay

University of Tokyo – Riken

# Introduction

- Understanding the performance of an application is critical
- More and more hybrid programming models
  - MPI+OpenMP
  - MPI+CUDA

  → Need for a modern framework for performance analysis

# Profiling an application

- Manual instrumentation of source code
  - Not convenient for large source codes

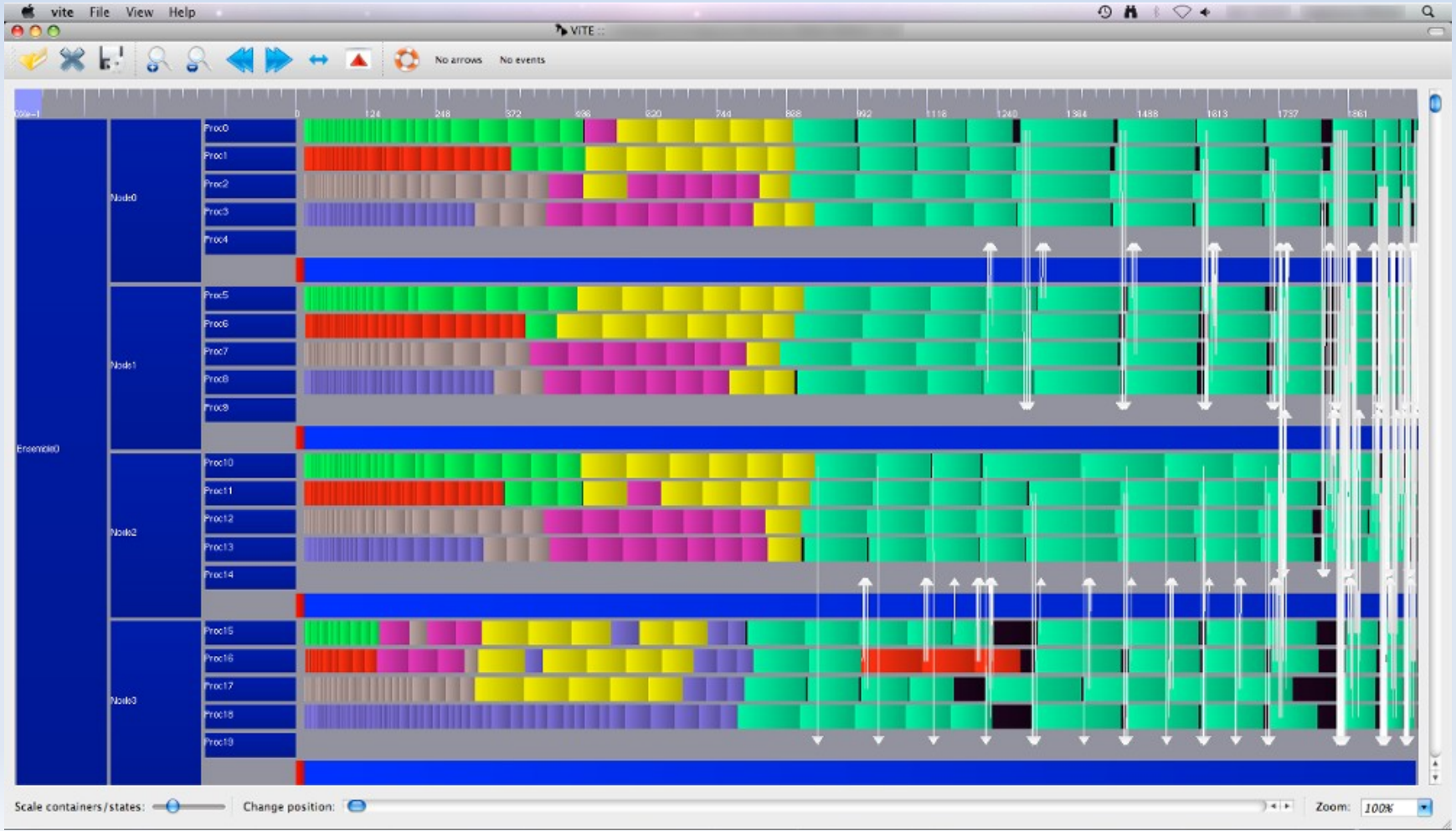
```
START_RECORD("MPI_Send");
MPI_Send(...);
STOP_RECORD("MPI_Send");
```

- Automatic instrumentation
  - Compiler-based (ex: VampirTrace)
  - Only works for predefined functions

- Both generate an execution trace

# Analyzing an execution trace

- Using visualisation tools
  - Vampir, ViTE



# Issues when profiling an application

- Hard to profile custom functions
  - Require source code instrumentation
- Need to recompile the application using a specific tool

# EZTrace: a framework for profiling applications

- Based on plugins
  - Allows to profile custom functions
- No need to recompile
- 2 separate phases:
  - Trace collection
    - Record events during the execution of the application
  - Trace analysis
    - Interpret the collected events

# Phase 1: Recording events

- Simply run the application with EZTrace

```
$ eztrace ./my_app param1 param2
$ mpiexec -np X `eztrace -e ./my_app` param1 param2
```

- Load plugins
  - MPI, GNU OpenMP, or custom plugins
- Intercept function calls
  - Record events in a raw format
- Generate one trace file per process

# Phase 2: Interpreting the events

- Convert and merge the trace files
- Load plugins
  - Possibly different from the recording phase
- Each event is interpreted
  - Generate a trace file
    - Can be visualized with ViTE or Vampir
  - Compute statistics
    - Number of calls to a function
    - Min/max/average duration of a function
    - ...

# Writing a plugin

- Possibility to create a plugin for custom functions
  - In C
  - Or with an easy to use script language
- EZTrace can interpret a script and compile a plugin

# Writing a plugin
## module definition

- Give some information to EZTrace
  - Module name
  - Description of the module
  - Language of the functions

```
NAME example_module
DESC "module for the example library"
LANGUAGE C
```

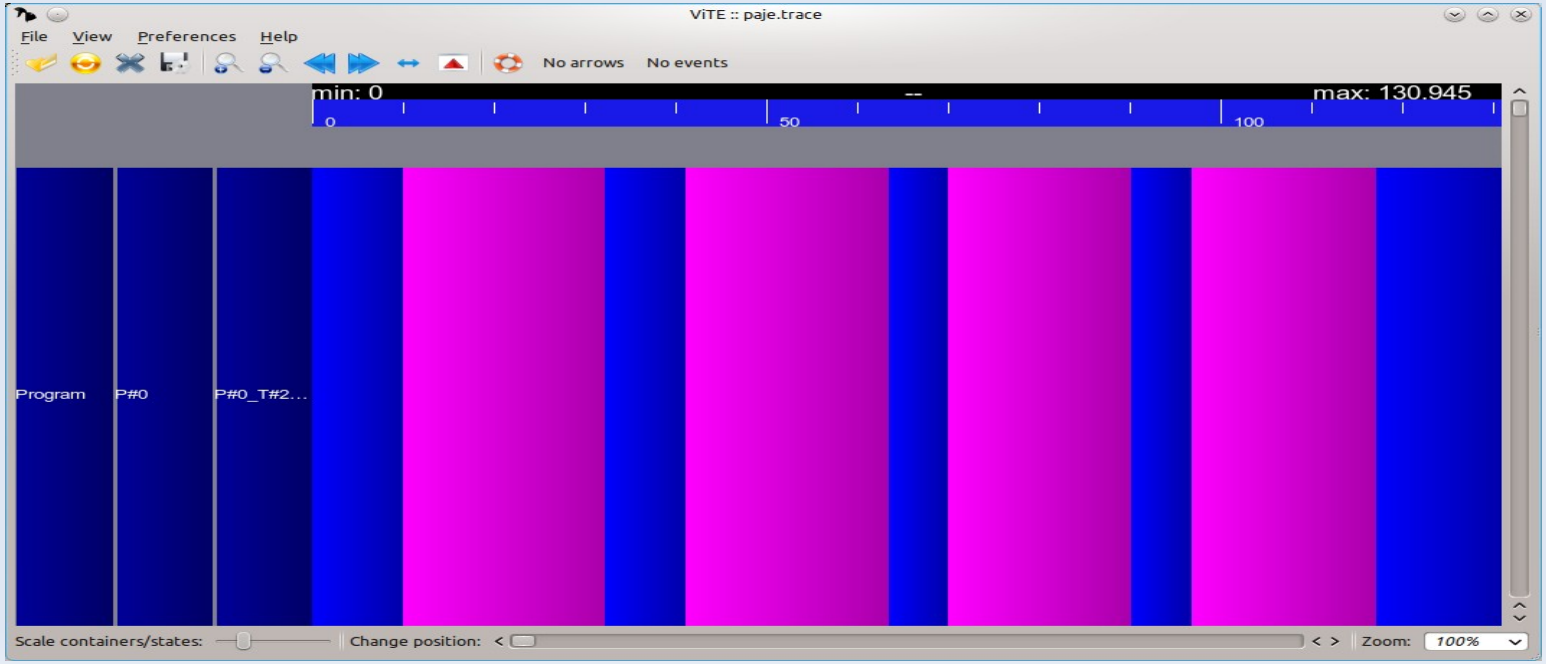
# Writing a plugin
## tracing a function

- For each function to trace, specify
  - The function prototype
  - The interpretation of the function in the output trace

```
int function0(int arg1, char* arg2)
BEGIN
  RECORD_STATE("doing function0")
END
```

# Writing a plugin
## tracing a function

```
int function0(int arg1, char* arg2)
BEGIN
  RECORD_STATE("doing function0")
END
```



# Writing a plugin
## tracing a function

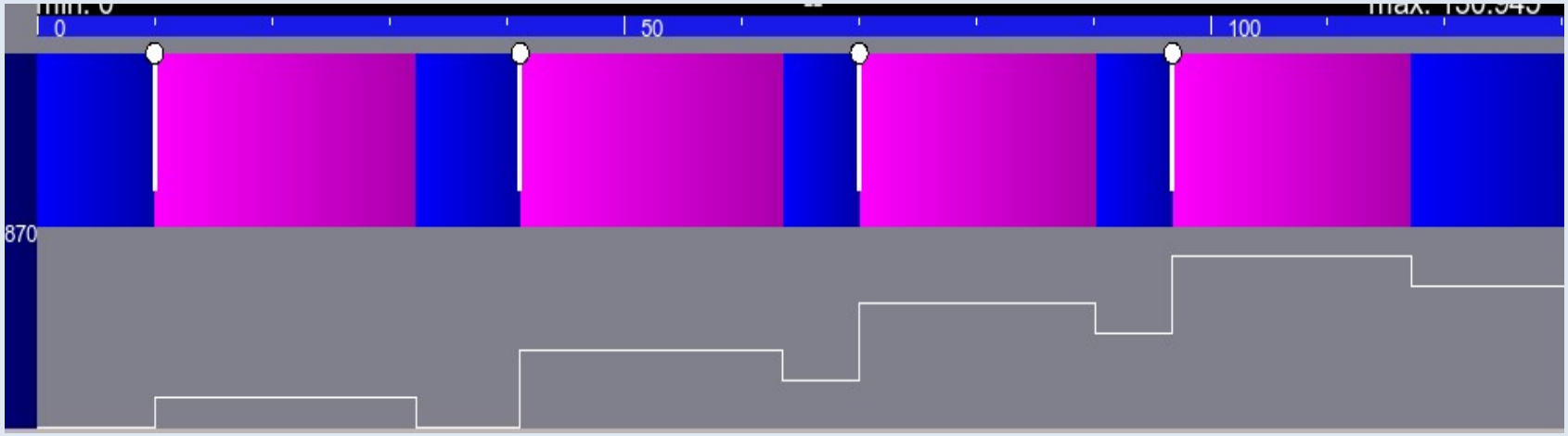
- Possibilities of interpretation:
  - Modify the state of the calling thread
    - `RECORD_STATE("doing function X")`
    - `PUSH_STATE("doing function X")`
    - `POP_STATE()`
  - Notify an event
    - `EVENT("function X")`
  - Modify a counter
    - `SET_VAR("number of jobs", value)`
    - `ADD_VAR("number of jobs", value)`
    - `SUB_VAR("number of jobs", value)`

# Writing a plugin
## example

```
int function0(int arg1, char* arg2)
BEGIN
   PUSH_STATE("doing function0")
   EVENT("Calling function0")
   ADD_VAR("number of jobs", arg1)
   CALL_FUNCTION
   SUB_VAR("number of jobs", 2)
   POP_STATE()
END
```

# Writing a plugin
## example (output)

```
int function0(int arg1, char* arg2)
BEGIN
   PUSH_STATE("doing function0")
   EVENT("Calling function0")
   ADD_VAR("number of jobs", arg1)
   CALL_FUNCTION
   SUB_VAR("number of jobs", 2)
   POP_STATE()
END
```



# Writing a plugin
## tracing a function

- Possibility to take the result into account

```
int submit_job(int job_id)
BEGIN
  ret = CALL_FUNCTION
  IF(ret == SUBMIT_SUCCESS)
    ADD_VAR("Number of jobs", 1)
  FI
END
```

# Performance evaluation

- Cluster0
  - 2 x dual core Opteron 2214 HE (2.2 GHz)
  - Myri-10G NICs
  - Evaluation on 4 nodes (16 cores)

- Comparison between
  - OpenMPI (development branch)
  - OpenMPI + EZTrace

# Performance evaluation

## NAS Parallel Benchmarks

- Low / no overhead on execution time

| Class=C, Nprocs=16 | | | | | |
|---|---|---|---|---|---|
| | OpenMPI | EZTrace | Overhead | File size | Nb events |
| **BT** | 218.41 s | 219.17 s | **0.35%** | 23MB | 464 032 |
| **CG** | 55.08 s | 51.34 s | **-6.79%** | 52MB | 1 118 816 |
| **EP** | 38.20 s | 37.95 s | **-0.65%** | 128KB | 192 |
| **FT** | 76.65 s | 85.65 s | **11.74%** | 192KB | 1 472 |
| **IS** | 4.96 s | 4.57 s | **-7.86%** | 192KB | 1 211 |
| **LU** | 252.08 s | 252.94 s | **0.34%** | 344MB | 7 772 564 |
| **MG** | 22.92 s | 23.65 s | **3.20%** | 12MB | 258 864 |
| **SP** | 382.49 s | 387.95 s | **1.43%** | 44MB | 924 768 |

# Conclusion

- Easy to use tracing tool
- No need to modify/recompile the application
- Support for hybrid model
- Pre-defined plugins
  - MPI, GNU OpenMP, PLASMA, PThread synchronisation functions
- Script language for generating plugins
- Low overhead

# Future works

- Implement the script parser
- Extend the script language
- Add support for performance counters
- Add support for CUDA/OpenCL

# Thank you !

- EZTrace:
  - Freely available (GPL2)
  - INRIA Bordeaux, U. Tokyo, U. Tennessee
  - http://eztrace.gforge.inria.fr/
- ViTE:
  - Freely available (Cecill-A)
  - INRIA Bordeaux
  - http://vite.gforge.inria.fr/