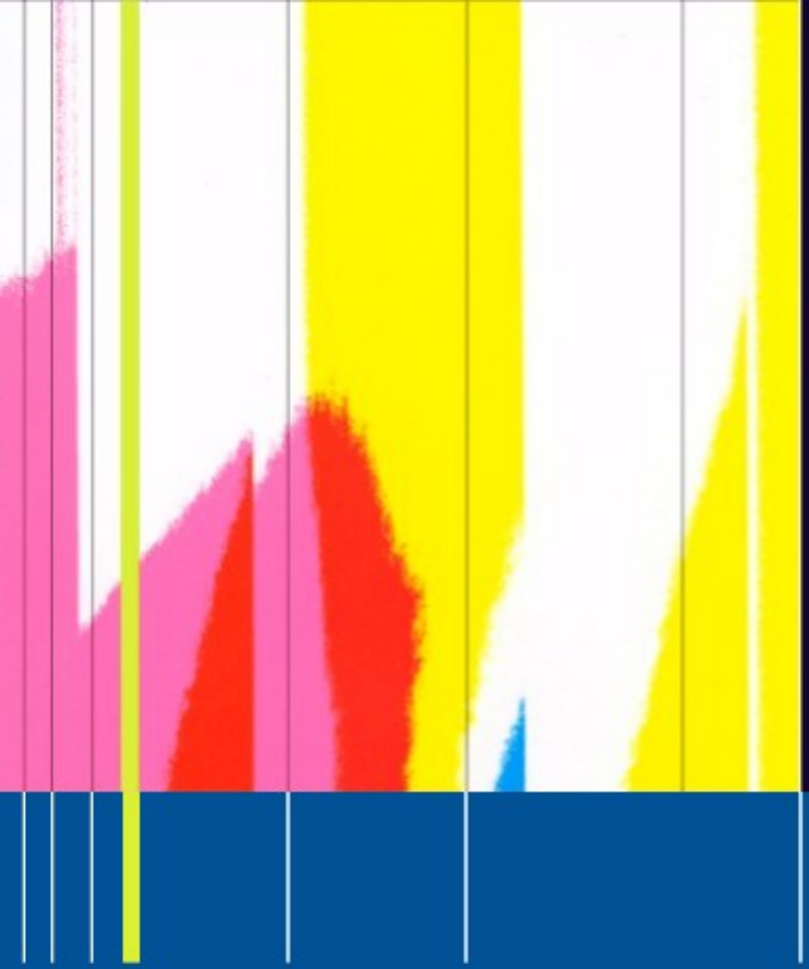


# EZTrace

*Générateur de traces*

Rué François

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD-OUEST

1. L'historique
2. Le fonctionnement
3. Démonstration / Exemple

1. L'historique
2. Le fonctionnement
3. Démonstration / Exemple

# Générer une trace d'exécution

Générateurs de trace d'exécution : outils permettant **d'analyser un code de calcul.**

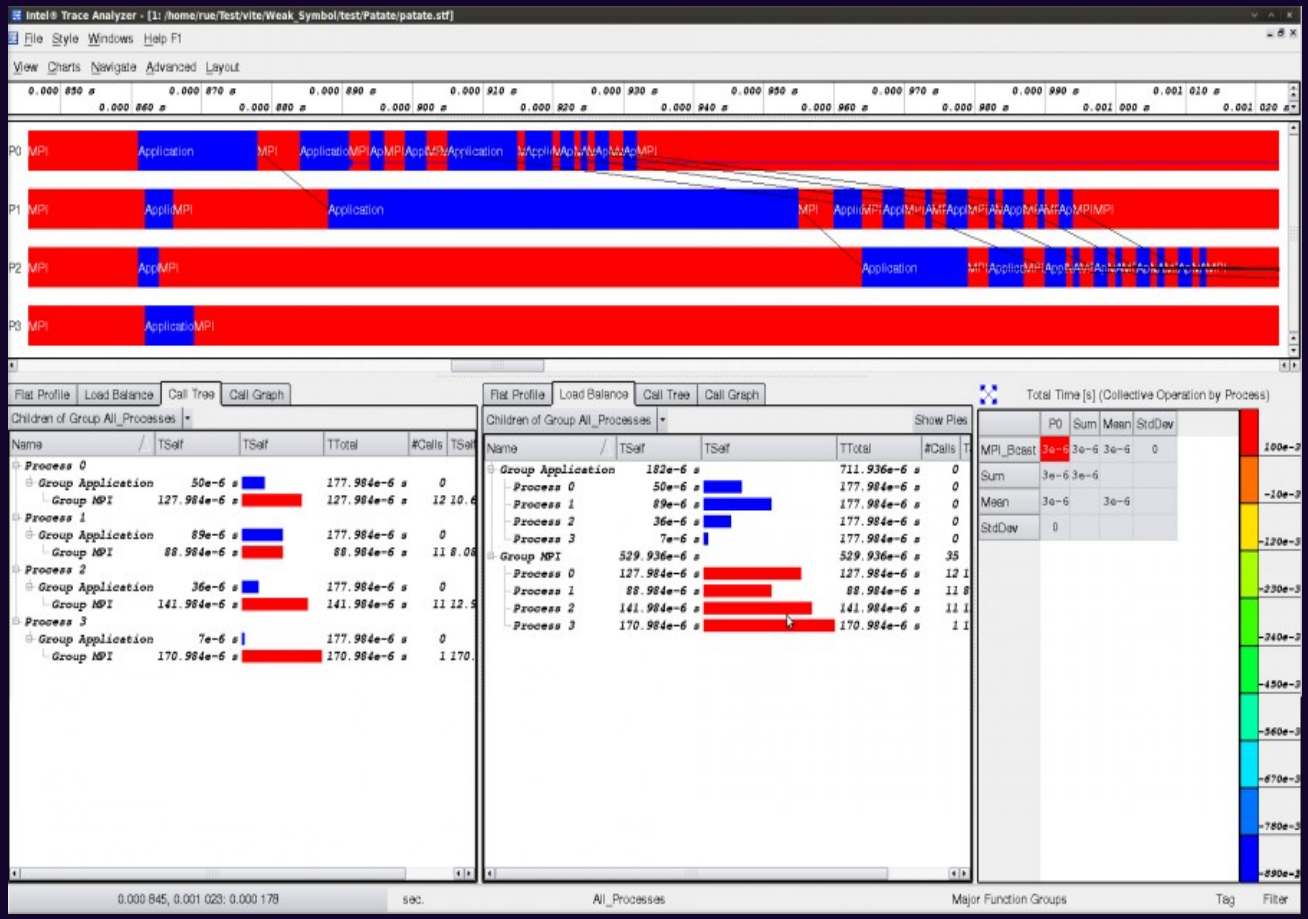
sur le marché :

1. ITAC : Intel Trace Analyser & Collector
2. MPITrace
3. OpenSpeedShop
4. Intel Thread Checker
5. Etc …

et maintenant EZTrace !

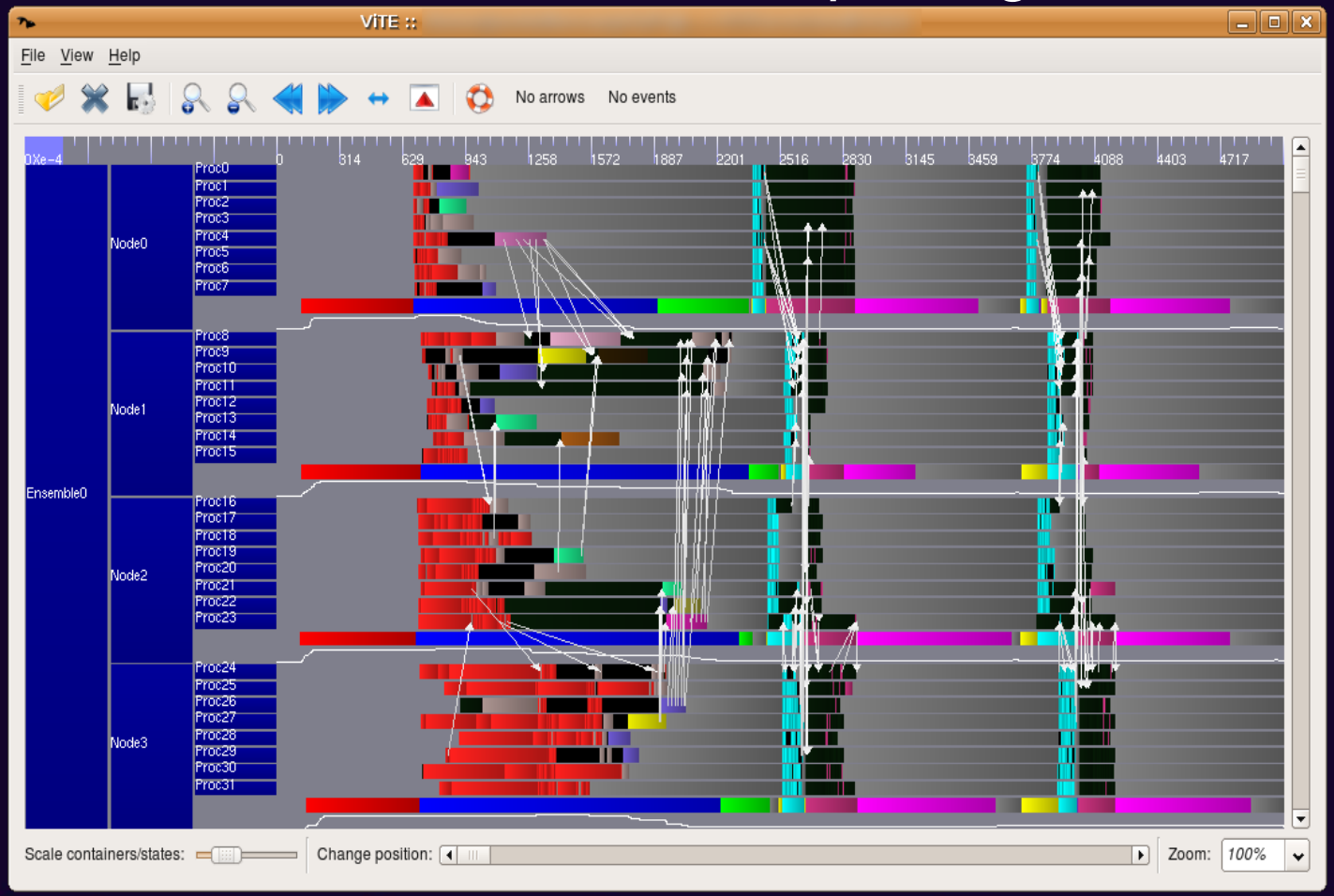
# Générer une trace d'exécution

Avec divers outils, les résultats obtenus :



# Générer une trace d'exécution

## Pour EZTrace le visualiseur privilégié est Vite



# Historique des besoins

**Les besoins :**

- Tracer le comportement d'un code OpenMP
- Ne rien instrumenter dans le code
- Que la trace soit facile à générer
- Que la trace soit facile à visualiser
- Qu'on puisse rajouter tout et n'importe quoi (plus tard) qui puisse être utile ….

**Approche simple :**

1. Intercepter les appels
2. Surcharger ces appels
3. Générer de la trace (OpenMP, MPI)
4. Utiliser des outils « maison »

# Les solutions

The Open Group Base Specifications Issue 6

IEEE Std 1003.1, 2004 Edition



**NAME**

dlsym - obtain the address of a symbol from a dlopen object

**SYNOPSIS**

```
#include <dlfcn.h>

void *dlsym(void *restrict handle, const char *restrict name);
```

# L'exemple

```
#include <stdio.h>
#include <dlfcn.h>
  int main (int argc, char ** argv)
{
    void * handle;
    double (* cosinus)(double);
    char * erreur;
    handle = dlopen ("/lib/libm.so", RTLD_LAZY);
    if (! handle) {
        fputs (dlerror (), stderr);
        exit(1);
    }
    cosinus = dlsym (handle, "cos");
    if ((erreur = dlerror()) != NULL)  {
        fprintf (stderr, "%s0, error);
        exit(1);
    }
    printf ("%f\n", (*cosinus) (2.0));
    dlclose(handle);
}
```

Fonction de la bibliothèque mathématique que l'on cherche à tracer.

La bibliothèque où on trouve les symboles à intercepter.

Notre nouvelle fonction Cosinus … qui est en fait La même

1. L'historique
2. Le fonctionnement
3. Démonstration / Exemple

# Les bibliothèques nécessaires

## FxT

- Ce projet ne fait pas partie du projet GNU.

  https://savannah.nongnu.org/projects/fxt

- FxT représente FKT (Fast Kernel Tracing) et FUT (Fast User Tracing). Cette bibliothèque fournit un support efficace pour l'enregistrement de traces.

## gtg

- Gtg – Generic Trace Generator – a pour but de fournir une interface simple de génération de trace d'exécution dans différents formats (Paje, OTF ...)

  http://gforge.inria.fr/projects/gtg

# La bibliothèque

**src**

- gomp, pthread, mpi ... blas
- Petit descriptif :

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,in
{
        int size;
        /* retrieve the size of the datatype so that we can compute the message
        MPI_Type_size(datatype, &size);
        EZTRACE_EVENT3 (FUT_MPI_START_SEND, count*
        int ret = libMPI_Send(buf, count, datatype, dest, tag,
        EZTRACE_EVENT3 (FUT_MPI_STOP_SEND, cou
        return ret;
}
```

MPI.C

```
void libinit(void)
{
...
INTERCEPT("MPI_Send", libMPI_Send);
…
}
```

EZTACE.H

```
/* interception function
func and store its previous value
into var */
#define INTERCEPT(func, var)
        do {
          void *__handle = RTLD_NEXT;
          var = (typeof(var)) (uintptr_t)
        dlsym(__handle, func);
                TREAT_ERROR();
        } while(0)
```

# Que faire pour que ca marche ?

**bin**

- Eztrace et Eztrace_convert
- Eztrace
  - Usage : (mpirun -np N) eztrace executable
  - En fait : LD_PRELOAD=$* executable
- Eztrace_convert
  - Convertir les traces générées sur le /tmp
  - eztrace_convert -o mypaje /tmp/eztrace_log_rank*

# Les limitations

**README**

Known limitations

---

* EZTrace relies on dlsym for interception various function calls. Thus, it only works with dynamically linked libraries.

* If EZTrace is compiled with a particular MPI implementation (OpenMPI for instance) it won't work if you run your application with another (MPICH2 for instance). So make sure your application uses the same MPI implementation as EZTrace.

1. L'historique
2. Le fonctionnement
3. Démonstration / Exemple

# Exemple MPI

## PING PONG

- time -p mpirun -np 2 ./pong

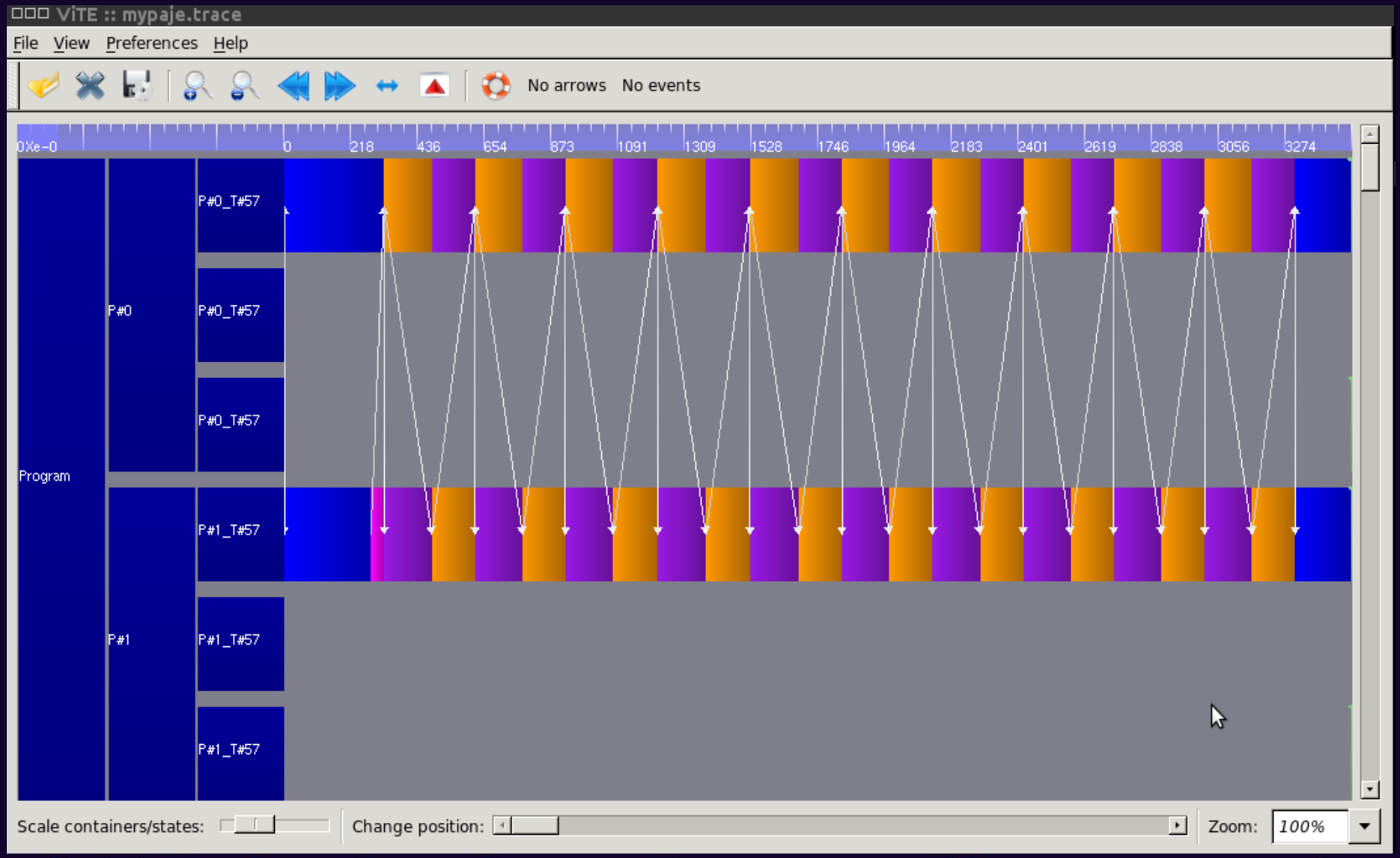
  résultat :

  real 5.15 user 9.44 sys 0.63
- Time -p mpirun -np 2 `eztrace -e ./pong`

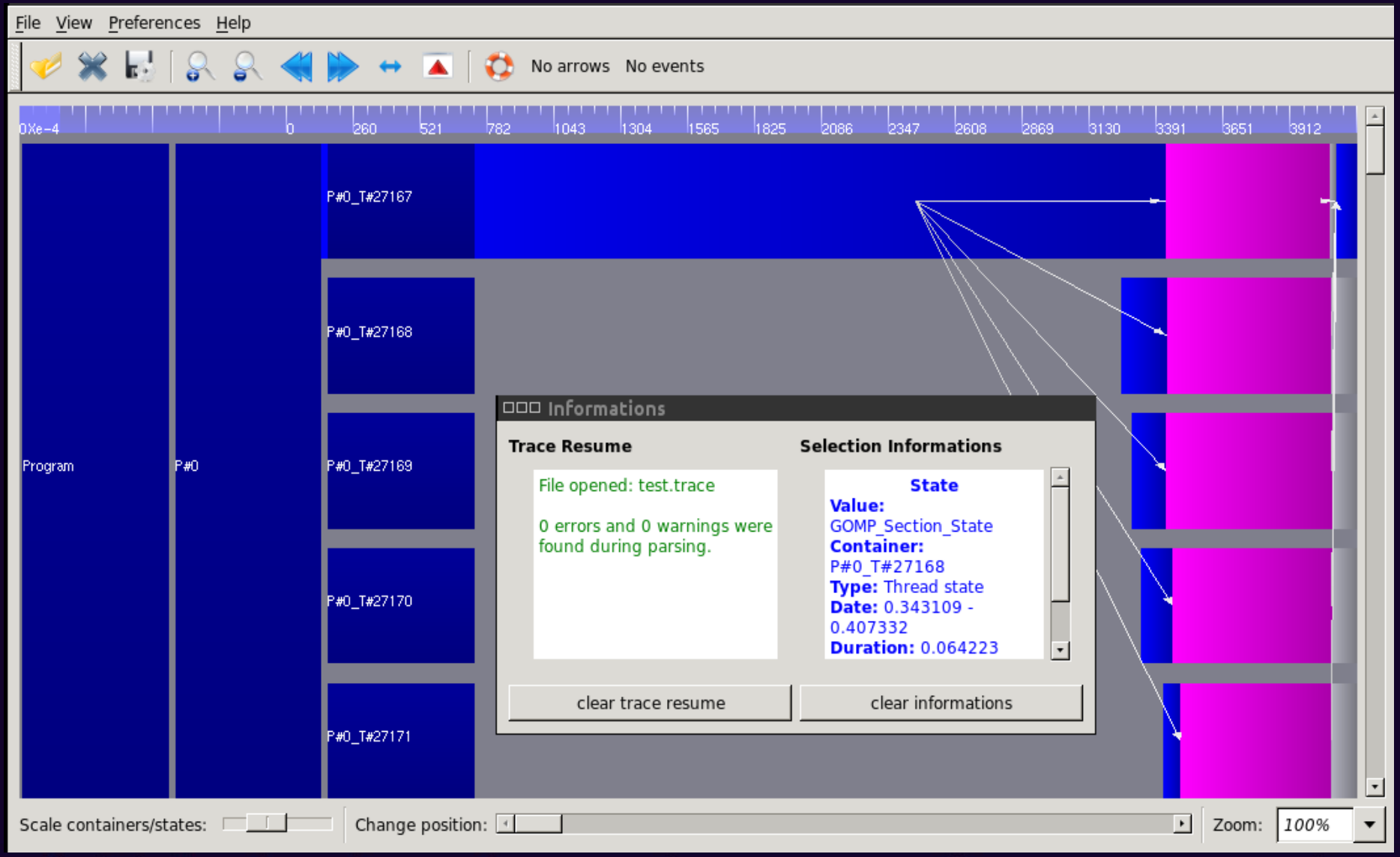
  résultat :

  real 5.23 user 7.57 sys 0.73

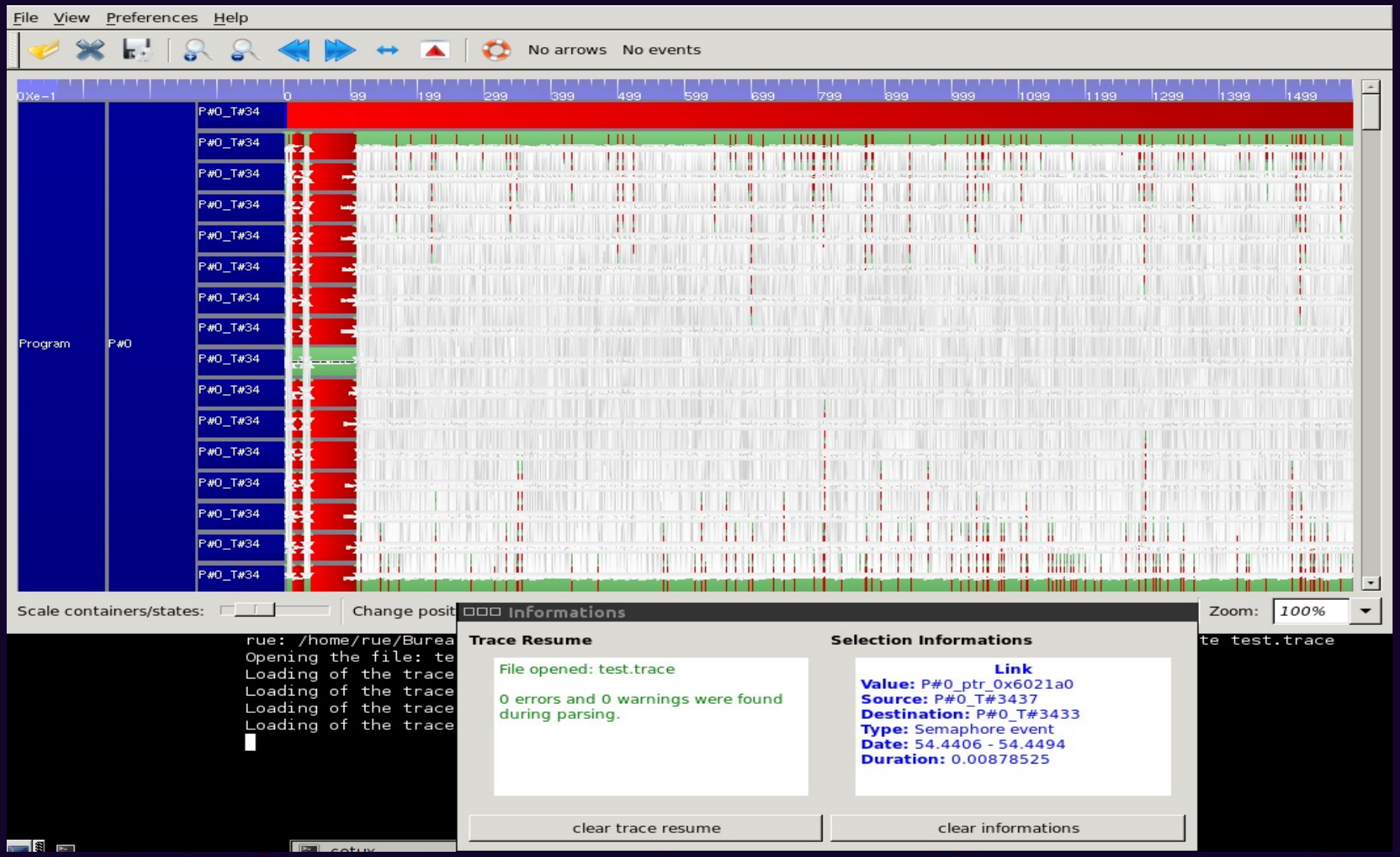
# Exemple MPI


ViTE :: mypaje.trace
File View Preferences Help
No arrows No events
0Xe-0
0
218
436
654
873
1091
1309
1528
1746
1964
2183
2401
2619
2838
3056
3274
Program
P#0
P#1
P#0_T#57
P#0_T#57
P#0_T#57
P#1_T#57
P#1_T#57
P#1_T#57
Scale containers/states:
Change position:
Zoom:
100%

# Exemple OpenMP



# Exemple pthread



# Les bibliothèques nécessaires

## Inconvénients

- Pas de link statique
- Pas de folklore compilation / usage des bibliothèques MPI

## Avantages

- Hybridation gérée
- Pas d'instrumentation nécessaire
- Évolutivité très élevée
- Produit local

# La feuille de route

**Fin 2010 : Porter EZTrace dans GTG –**

**Augmenter le nombre de formats de traces issues de  EZTrace pour avoir accès à un nombre plus large de logiciels de visualisation-**

**Fin 2010 :. Suivi allocation de mémoire**

**Quelle quantité de mémoire est utilisée par le logiciel en temps réel. Permettre de mieux gérer la mémoire de l'application et donc les performances-**

**Fin 2010 : Support de PLASMA : gestion des appels PLASMA**

**PLASMA : Bibliothèque de calcul numérique. Tracer les appels à cette bibliothèque. Objectif : offrir des fonctionnalités complémentaires-**

**Fin 2011 : Support PAPI : insertion Compteurs Hardware -**

**Donner des indications aux développeurs sur les performances de l'application (visualisation des problèmes)-**

**Fin 2011 : Support CUDA &/ou OpenCL**

**Permet d'exécuter une partie du programme sur carte graphique-**

# Questions ?