Outils d'analyse de performance pour le HPC

CSC 5001

Octobre 2019

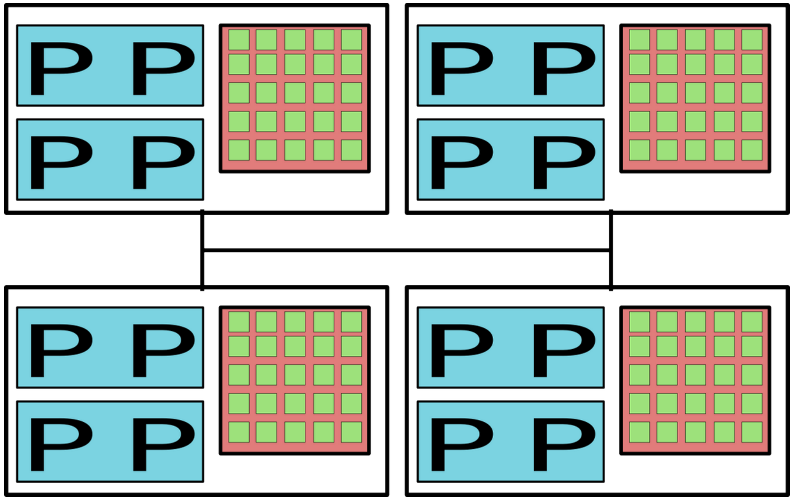
Introduction

- **Obsession dans le HPC : utiliser au mieux la puissance de calcul**

  → les applications doivent être optimisées

- **Problèmes :**
  - Matériel de plus en plus compliqué
    - Processeurs multicore, caches hiérarchiques, machines NUMA, etc.
  - On mélange les modèles de programmation
    - MPI, MPI+OpenMP, MPI+CUDA, etc.

- **Optimiser une application est extrêmement compliqué**



TELECOM SudParis

Optimiser une application

1. Executer l'application
2. si(performances satisfaisantes) return ;
3. Trouver une phase à optimiser
4. Optimiser
5. Goto 1

- **Partie compliquée : phase 3**
- **Nécessite de**
  - Comprendre le déroulement de l'application
  - Connaître la durée des différentes parties de l'application

Trouver les phases de l'application à optimiser

- **A la main**
  - Ajout d'appels à `gettimeofday()` dans le code source
  - But : identifier quelle partie du code prend le plus de temps
  - Inconvénient : fastidieux

```
for(i=0; i<iter; i++) {
  gettimeofday(&t[0], NULL);
  do_function_1();
  gettimeofday(&t[1], NULL);
  do_function_2();
  gettimeofday(&t[2], NULL);
  do_function_3();
  gettimeofday(&t[3], NULL);
  process_timers(t, 4);
}
```

Outils de profiling

Outils de profiling

- **But : décrire statistiquement le déroulement de l'application**
  - Temps passé dans chaque fonction (avec gprof)
    - Compiler avec `-pg`
    - Exécuter `./appli` (→ génère `gmon.out`)
    - `gprof ./appli`

```
$ gcc -pg -fopenmp -lm sgefa_openmp.c -o sgefa_openmp_gprof
$ ./sgefa_openmp_gprof
[...]
$ gprof ./sgefa_openmp_gprof
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 51.02      2.33     2.33     1980     0.00     0.00  sswap
 29.56      3.68     1.35      999     0.00     0.00  msaxpy2
 18.40      4.52     0.84   505497     0.00     0.00  saxpy
  0.66      4.55     0.03        3     0.01     0.01  matgen
  0.22      4.56     0.01        1     0.01     1.18  msgefa
  0.22      4.57     0.01        1     0.01     0.84  sgefa
  0.00      4.57     0.00     2997     0.00     0.00  isamax
[...]
```

Outils de profiling

- **But : décrire statistiquement le déroulement de l'application**
  - Utilisation des caches (avec cachegrind)
  - `valgrind --tool=cachegrind ./appli`
  - `kcachegrind cachegrind.out.5055`

```
$ valgrind --tool=cachegrind ./appli
[...]
==5055== I   refs:      48,000,410
==5055== I1  misses:         1,818
==5055== LLi misses:         1,557
==5055== D   refs:      22,700,314  (21,338,206 rd   + 1,362,108 wr)
==5055== D1  misses:        34,891  (    31,269 rd   +     3,622 wr)
==5055== LLd misses:         4,005  (     2,534 rd   +     1,471 wr)
==5055== D1  miss rate:        0.2% (       0.1%     +       0.3%   )
==5055== LLd miss rate:        0.0% (       0.0%     +       0.1%   )
==5055==
==5055== LL refs:           36,709  (    33,087 rd   +     3,622 wr)
==5055== LL misses:          5,562  (     4,091 rd   +     1,471 wr)
==5055== LL miss rate:         0.0% (       0.0%     +       0.1%   )
```

Outils de profiling

- **But : décrire statistiquement le déroulement de l'application**
  - Chemins d'appels des fonctions (avec callgrind)
  - `valgrind --tool=callgrind ./appli`
  - `kcachegrind callgrind.out.5056`

```
$ valgrind --tool=callgrind ./appli
[...]
==5055== I   refs:      48,000,410
==5055== I1  misses:         1,818
==5055== LLi misses:         1,557
==5055== D   refs:      22,700,314  (21,338,206 rd   + 1,362,108 wr)
==5055== D1  misses:        34,891  (    31,269 rd   +     3,622 wr)
==5055== LLd misses:         4,005  (     2,534 rd   +     1,471 wr)
==5055== D1  miss rate:        0.2% (       0.1%     +       0.3%  )
==5055== LLd miss rate:        0.0% (       0.0%     +       0.1%  )
==5055==
==5055== LL refs:           36,709  (    33,087 rd   +     3,622 wr)
==5055== LL misses:          5,562  (     4,091 rd   +     1,471 wr)
==5055== LL miss rate:         0.0% (       0.0%     +       0.1%  )
```

Traces d'exécution

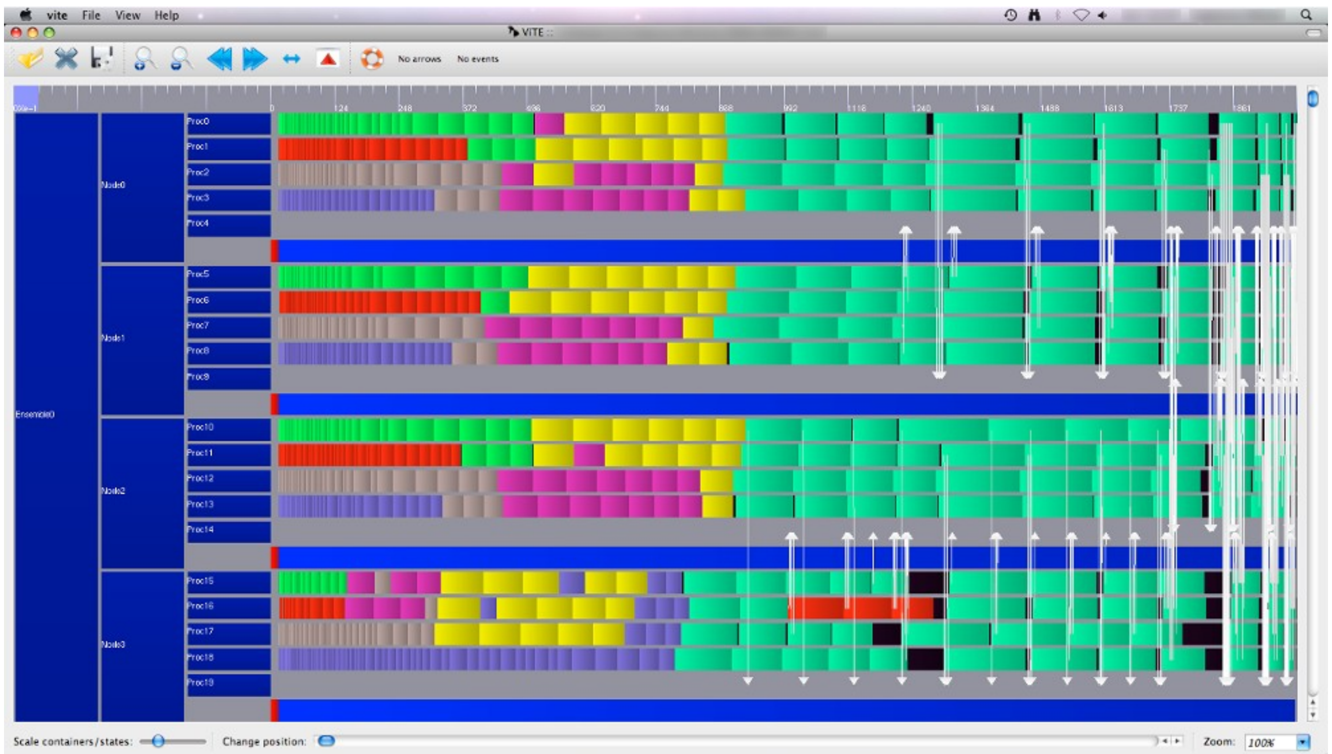
Traces d'exécution

- **But : décrire dynamiquement le déroulement de l'application**
- **Trace d'exécution : liste d'événements horodatés**

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

Visualisation de traces d'exécution

- **Représentation graphique des événements stockés dans une trace**



TELECOM SudParis

Les outils

- **Génération de traces d'exécution**
  - VampirTrace
  - **EZTrace**
  - Tau
- **Visualisation de traces d'exécution**
  - Vampir
  - **ViTE**
- **Tout en un**
  - Scalasca
  - OpenSpeedShop
  - Intel Trace Analyzer and Collector

En gras : les outils installés en B313

Liste non exhaustive

Utiliser EZTrace

Générer une trace avec EZTrace

- **Lister les modules disponibles**

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

- **Exécuter l'application**

```
$ eztrace -t ''module1 module2'' ./mon_programme
$ mpirun -np 2 eztrace -t ''module1 module2'' ./mon_programme
```

  - Génère des fichiers de trace (`/tmp/${USER}_eztrace_log_rank*`, etc.)

- **Visualiser la trace obtenue**

```
$ eztrace_convert /tmp/${USER}_eztrace_log_rank_*

$ vite eztrace_output.trace
```

Problèmes classiques avec EZTrace

- **EZTrace ne capture pas les événements OpenMP**
  - Besoin de compiler l'application avec `eztrace_cc`
  - `$ eztrace_cc gcc -o mon_appli mon_appli.c`

```
[EZTrace] The buffer for recording events is full. Stop recording. The trace will be truncated
```

`$ export EZTRACE_BUFFER_SIZE=1073741824`

ou `$ export EZTRACE_FLUSH=1`

Des tutoriels sont disponibles en ligne

`http://eztrace.gforge.inria.fr/tutorials/index.html`

Utiliser les compteurs matériels

PAPI

Performance Application Programming Interface

- **Le processeur fourni un ensemble de compteurs matériels**
  - Nombre de cache hits/miss
  - Nombre d'instructions exécutée
  - ...
- **PAPI : bibliothèque facilitant l'accès à ces compteurs**

```
$ papi_avail -a
    Name        Code    Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  No   Level 2 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  No   Level 3 cache misses
PAPI_L3_LDM  0x8000000e  No   Level 3 load misses
PAPI_TLB_DM  0x80000014  No   Data translation lookaside buffer misses
[...]
```

Utiliser PAPI

- **Avec EZTrace**

```
$ export EZTRACE_PAPI_COUNTERS=PAPI_L2_TCM:PAPI_FP_OPS
```

- **Attention ! Consultation de 2 compteurs max**

TELECOM SudParis