



# Performance analysis : Hands-on

## **time**

- Wall/CPU
- parallel context

## **gprof**

- flat profile/call graph
- self/inclusive
- MPI context

## **VTune**

- hotspots, per line profile
- advanced metrics :
  - general exploration, snb-memory-access, concurrency ...
- parallel context



# Performance analysis : Hands-on

---

## ■ **Scalasca**

- Load imbalance
- PAPI counters

## ■ **Vampir**

- trace

## ■ **A memory instrumentation**



# Hands on Environment

---



# The platform : Poincare

## Architecture

- 92 nodes
- 2 Sandy Bridge x 8 cores
- 32 Go

## Environnement

- Intel 13.0.1
- OpenMPI 1.6.3

## Job & resources manager

- Today : Max **1** node / job

- Compile on interactive nodes :

```
[mdlslx181]$ poincare
```

- Run on :

```
[poincareint01]$ llinteractif 1 clallmds 6
```



**Hwloc : lstopo**



# The code : Poisson

## Poisson - MPI @ IDRIS

- C / Fortran

## Code reminder

- Stencil :

```
u_new[ix,iy] = c0 * (
    c1 * ( u[ix+1,iy] + u[ix-1,iy] )
    + c2 * ( u[ix,iy+1] + u[ix,iy-1] )
    - f[ix,iy] );
```

- Boundary limits :

```
u = 0
```

- Convergence criterion :

```
max | u[ix,iy] - u_new[ix,iy] | < eps
```

## MPI

- Domain decomposition
- Exchanging ghost cells



## The code : Poisson (2)

### Data size

```
$ cat poisson.data  
480  
400
```

### Validation :

- *compile on an interactive node :*  
`[poincareint01]$ make read`  
`[poincareint01]$ make calcul_exact`
- *Run on a compute node*  
`[poincare001]$ make verification`  
...  
BRAVO, Vous avez fini



# Basics

---



## time : Elapsed, CPU

### Command lines :

```
$ time mpirun -np 1 ./poisson.mpi
```

### Sequential results :

...

Convergence apres 913989 iterations en **425.560393** secs

...

**MPI\_Wtime : Macro instrumentation**

**real 7m6.914s Time to solution**

**user 7m6.735s Resources used**

**sys 0m0.653s**



## time : MPI

---

### Command lines :

```
$ time mpirun -np 16 ./poisson.mpi
```

### MPI results :

...

Convergence apres 913989 iterations en **38.221655** secs

...

**real 0m39.866s**

**user 10m27.603s**

**sys 0m1.614s**



# time : OpenMP

## Command lines :

```
$ export OMP_NUM_THREADS=8  
$ time mpirun -bind-to-socket -np 1 ./poisson.omp
```

## OpenMP results :

```
...  
Convergence apres 913989 iterations en 172.729974 secs  
...  
real 2m54.224s  
user 22m32.978s  
sys 0m31.832s
```



# Resources

## ■ Binding :

```
$ time mpirun -report-bindings -np 16 ./poisson.mpi 100000
```

```
Convergence apres 100000 iterations en 4.249197 secs
```

```
$ time mpirun -bind-to-none -np 16 ./poisson.mpi 100000
```

```
Convergence apres 100000 iterations en 25.626133 secs
```

```
$ man mpirun / mpiexec ... for the required option
```

```
$ export OMP_NUM_THREADS=8
```

```
$ time mpirun -report-bindings -np 1 ./poisson.omp 100000
```

```
$ time mpirun -bind-to-socket -np 1 ./poisson.omp 100000
```

## ■ But not only :

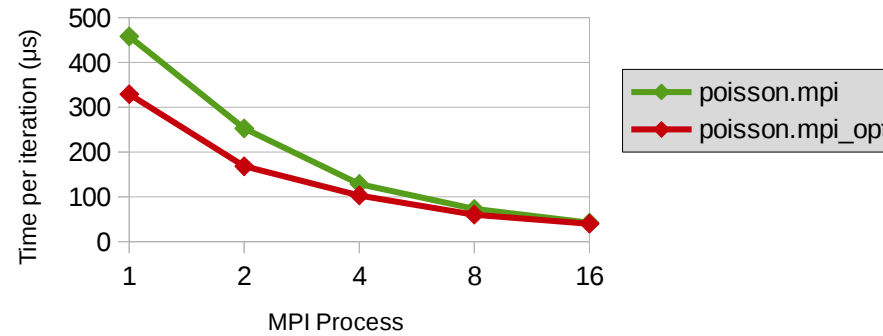
- Process/thread distribution
- Dedicated resources



# Scaling metrics & Optimisation

## Optimisation

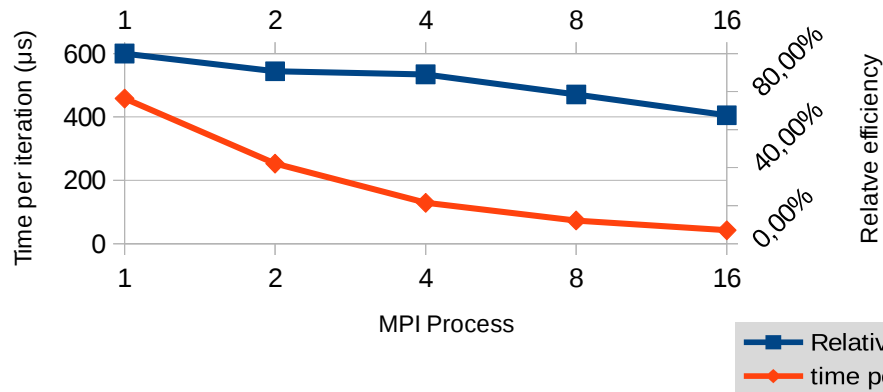
Grid size : 480 x 400



**Damaged scaling but ...  
Better restitution time**

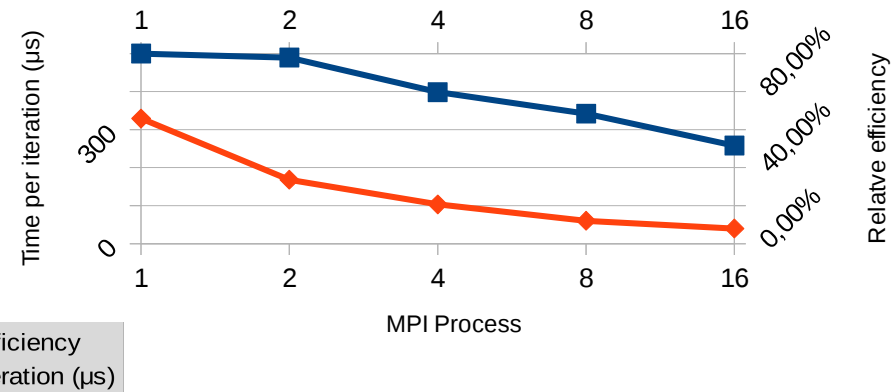
## MPI Scaling : Optimised

Grid Size : 480 x 400



## MPI Scaling : Additional Optim

Grid size : 480 x 400





# gprof : Basics

- Widely available :
  - GNU, Intel, PGI ...
- Regular code pattern, limit the number of iterations
  - Should be consolidated after optimisation
  - Measure reference on a limited number of iterations
- Edit `make_inc` to enable `-pg` option, then recompile
- **Command lines :**

```
$ mpirun -np 1 ./poisson 100000
Convergence apres 100000 iterations en 47.714439 secs
$ ls gmon.out
gmon.out
$ gprof poisson gmon.out
```



# gprof : Flat profile

Each sample counts as 0.01 seconds.

| %<br>time | cumulative<br>seconds | self<br>seconds | calls  | self<br>us/call | total<br>us/call | name           |
|-----------|-----------------------|-----------------|--------|-----------------|------------------|----------------|
| 74.87     | 35.66                 | 35.66           | 100000 | 356.60          | 356.60           | calcul         |
| 25.27     | 47.70                 | 12.04           | 100000 | 120.37          | 120.37           | erreur_globale |
| 0.00      | 47.70                 | 0.00            | 100000 | 0.00            | 0.00             | communication  |

**Consolidate application behavior using an external tool**

| index | % time | self  | children | called        | name                   |
|-------|--------|-------|----------|---------------|------------------------|
|       |        |       |          |               | <spontaneous>          |
| [1]   | 100.0  | 0.00  | 47.70    |               | main [1]               |
|       |        | 35.66 | 0.00     | 100000/100000 | calcul [2]             |
|       |        | 12.04 | 0.00     | 100000/100000 | erreur_globale [3]     |
|       |        | 0.00  | 0.00     | 100000/100000 | communication [4]      |
|       |        | 0.00  | 0.00     | 1/1           | creation_topologie [5] |
| ...   |        |       |          |               |                        |
|       |        | 35.66 | 0.00     | 100000/100000 | main [1]               |
| [2]   | 74.8   | 35.66 | 0.00     | 100000        | calcul [2]             |



# gprof : Call graph

Each sample counts as 0.01 seconds.

| %<br>time | cumulative<br>seconds | self<br>seconds | calls  | self<br>us/call | total<br>us/call | name           |
|-----------|-----------------------|-----------------|--------|-----------------|------------------|----------------|
| 74.87     | 35.66                 | 35.66           | 100000 | 356.60          | 356.60           | calcul         |
| 25.27     | 47.70                 | 12.04           | 100000 | 120.37          | 120.37           | erreur_globale |
| 0.00      | 47.70                 | 0.00            | 100000 | 0.00            | 0.00             | communication  |

**index % time self children called name**

<spontaneous>

|     |       |       |              |               |                        |
|-----|-------|-------|--------------|---------------|------------------------|
| [1] | 100.0 | 0.00  | <b>47.70</b> |               | main [1]               |
|     |       | 35.66 | 0.00         | 100000/100000 | calcul [2]             |
|     |       | 12.04 | 0.00         | 100000/100000 | erreur_globale [3]     |
|     |       | 0.00  | 0.00         | 100000/100000 | communication [4]      |
|     |       | 0.00  | 0.00         | 1/1           | creation_topologie [5] |

...

-----

|     |      |       |      |               |            |
|-----|------|-------|------|---------------|------------|
|     |      | 35.66 | 0.00 | 100000/100000 | main [1]   |
| [2] | 74.8 | 35.66 | 0.00 | 100000        | calcul [2] |



## *Additional informations : gprof & MPI*

- A per process profile :
  - Setting environment variable : **GMON\_OUT\_PREFIX**

### ■ **Command lines :**

```
$ cat exec.sh
```

```
-----  
#!/bin/bash  
# "mpirun -np 1 env|grep RANK"  
export GMON_OUT_PREFIX='gmon.out-${OMPI_COMM_WORLD_RANK}'  
./poisson  
-----
```

```
$ mpirun -np 2 ./exec.sh  
$ ls gmon.out-*  
gmon.out-0.18003  
gmon.out-1.18004  
$ gprof poisson gmon.out-0.18003
```



# Vtune - Amplificator

---



## VTune : Start

Optimise the available sources :

```
$ mpirun -np 16 ./poisson
```

Convergence apres 913989 iterations en **1270.757420** secs

Reminder :

```
$ mpirun -np 16 ./poisson.mpi
```

Convergence apres 913989 iterations en **38.221655** secs

Reduce number of iterations to 10000 :

```
$ mpirun -np 1 ./poisson 10000
```

Convergence apres 10000 iterations en 38.011032 secs

```
$ mpirun -np 1 amplxe-cl -collect hotspots -r profil ./poisson
```

Convergence apres 10000 iterations en 38.109222 secs

```
$ amplxe-gui profil.0 &
```

<https://software.intel.com/en-us/qualify-for-free-software/student>



# VTune : Analysis

Welcome | profil.0 x

**Basic Hotspots** Hotspots by CPU Usage viewpoint (change) ? Intel VTune Amplifier XE 2015

Analysis Target | Analysis Type | Collection Log | **Summary** | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

### Elapsed Time: 39.350s

[Total Thread Count:](#) 2  
[Overhead Time:](#) 0s  
[Spin Time:](#) 0s  
[CPU Time:](#) 39.150s  
[Paused Time:](#) 0s

### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

| Function                       | CPU Time |
|--------------------------------|----------|
| <a href="#">calcul</a>         | 35.220s  |
| <a href="#">erreur_globale</a> | 2.770s   |
| <a href="#">__psm_ep_close</a> | 1.000s   |
| <a href="#">read</a>           | 0.070s   |
| <a href="#">PMPI_Init</a>      | 0.020s   |
| [Others]                       | 0.070s   |

### CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

Elapsed Time

Average

Target Concurrence

Simultaneously Utilized Logical CPUs: 18  
Elapsed Time: 0s

Idle Poor Ok Ideal

Simultaneously Utilized Logical CPUs

### Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: ./poisson  
Operating System: 2.6.32-431.el6.x86\_64 CentOS release 6.5 (Final)  
MPI Process Rank: 0  
Computer Name: poincare090-adm  
Result Size: 2 MB



# Vtune : Profile

Basic Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2015

Analysis Target Analysis Type Collection Log Summary Bottom-up Caller/Callee Top-down Tree Tasks and Frames

Grouping: Function / Call Stack

| Function / Call Stack | CPU Time                      |      |    |       |      | Module    | Function (Full) | Source File            | Start Address  |                  |
|-----------------------|-------------------------------|------|----|-------|------|-----------|-----------------|------------------------|----------------|------------------|
|                       | Effective Time by Utilization |      |    |       |      |           |                 |                        |                |                  |
|                       | Idle                          | Poor | Ok | Ideal | Over | Spin Time | Ov.. Time       |                        |                |                  |
| ▸ calcul              | 35.220s                       |      |    |       |      | 0s        | 0s              | poisson                | calcul         | 0x402140         |
| ▸ erreur_globale      | 2.770s                        |      |    |       |      | 0s        | 0s              | poisson                | erreur_globale | 0x401f60         |
| ▸ __psm_ep_close      | 1.000s                        |      |    |       |      | 0s        | 0s              | libpsm_infinipath.so.1 | __psm_ep_close | psm_ep.c 0x14a90 |
| ▸ read                | 0.070s                        |      |    |       |      | 0s        | 0s              | libc-2.3.4.so          | read           | 0xbad10          |
| ▸ PMPi_Init           | 0.020s                        |      |    |       |      | 0s        | 0s              | libmpi.so.1.0.6        | PMPi_Init      | pinit.c 0x7cc20  |
| ▸ strlen              | 0.020s                        |      |    |       |      | 0s        | 0s              | libc-2.3.4.so          | strlen         | 0x70100          |
| ▸ strcpy              | 0.010s                        |      |    |       |      | 0s        | 0s              | libc-2.3.4.so          | strcpy         | 0x6fb80          |
| ▸ strcmp              | 0.010s                        |      |    |       |      | 0s        | 0s              | libc-2.3.4.so          | strcmp         | 0x6fb00          |
| ▸ __GI_memset         | 0.010s                        |      |    |       |      | 0s        | 0s              | libc-2.12.so           | __GI_memset    | 0x83cc0          |
| ▸ _IO_vfscanf         | 0.010s                        |      |    |       |      | 0s        | 0s              | libc-2.3.4.so          | _IO_vfscanf    | 0x4e320          |
| ▸ __psm_ep_open       | 0.010s                        |      |    |       |      | 0s        | 0s              | libpsm_infinipath.so.1 | __psm_ep_open  | psm_ep.c 0x16690 |

Selected 1 row(s): 35.220s 0s 0s

Thread: poisson (TID: ips\_ptl\_pollint)

CPU Usage

Thread:  Running  CPU Time  Spin and ...  CPU Sample

CPU Usage:  CPU Time  Spin and ...

No filters are applied. Any Process Any Thread Any Module Any Utilization User functions + 1 Inline Mode: on Functions only



# VTune : Data filtering

The screenshot displays the Intel VTune Amplifier XE 2015 interface. The main window shows a table of CPU hotspots, with the 'calcul' function highlighted. A blue oval labeled 'Per function' is drawn around the 'calcul' row. Below the table is a timeline view for the 'poisson' thread, showing CPU usage over time. A blue oval labeled 'Timeline filtering' is drawn around the timeline. At the bottom, a filter bar shows 'Filter: 97.0% is shown' and 'Any Process', 'Any Thread', '[97.0%] poisson', 'Any Utilization', 'User functions + 1', 'Inline Mode: on', and 'Functions only'. A blue oval labeled 'Application/MPI/system' is drawn around the filter bar.

| Function / Call Stack | CPU Time                      |      | Spin Time | Ov... Time | Module | Function (Full)        | So.. File | Start Address |
|-----------------------|-------------------------------|------|-----------|------------|--------|------------------------|-----------|---------------|
|                       | Effective Time by Utilization |      |           |            |        |                        |           |               |
|                       | Idle                          | Poor | Ok        | Ideal      | Over   |                        |           |               |
| calcul                | 35.220s                       |      |           |            |        | poisson calcul         |           | 0x402140      |
| erreur_globale        | 2.770s                        |      |           |            |        | poisson erreur_globale |           | 0x401f60      |

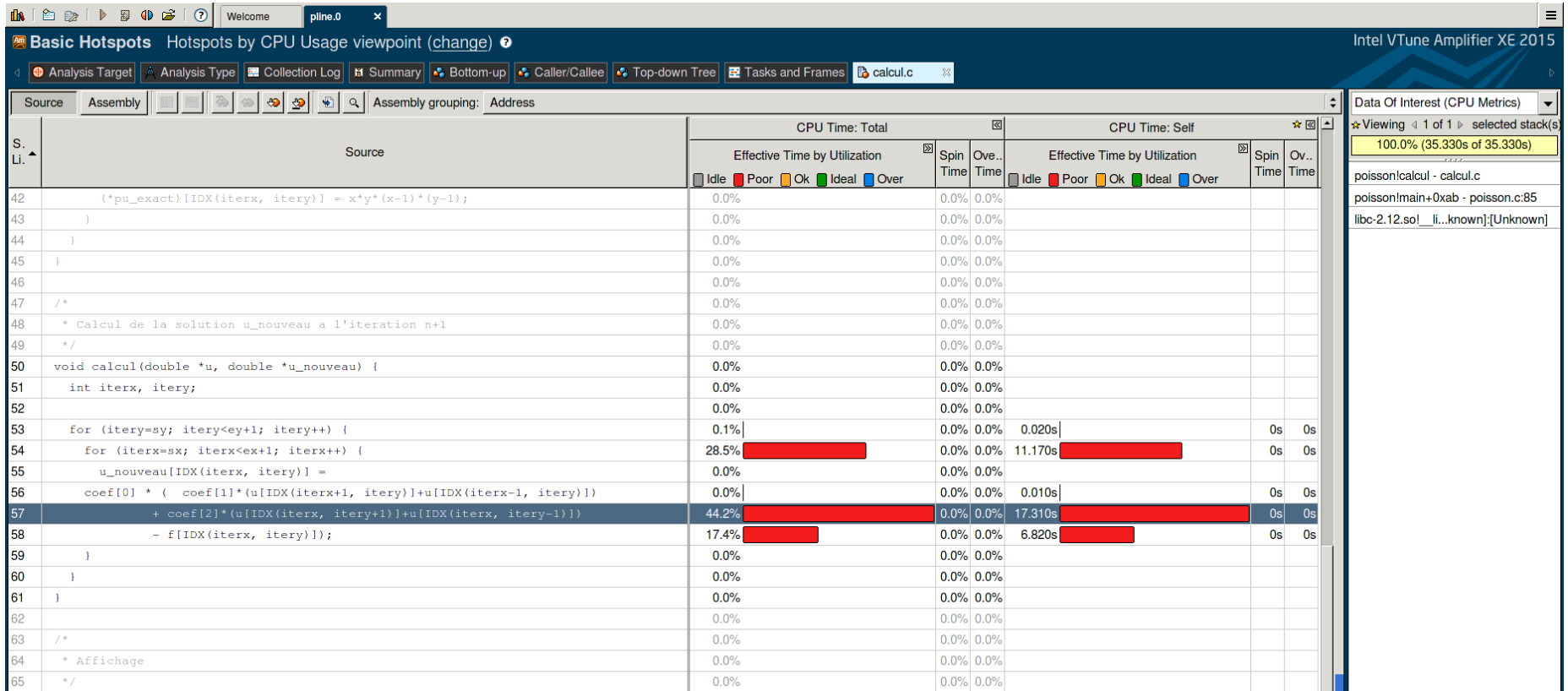
Selected 1 row(s): 35.220s 0s 0s

Filter: 97.0% is shown Any Process Any Thread [97.0%] poisson Any Utilization User functions + 1 Inline Mode: on Functions only

Application/MPI/system



# VTune : Per line profile



```
Edit make_inc to enable -g option, then recompile
$ mpirun -np 1 amplxe-cl -collect hotspots -r pline ./poisson
Convergence apres 10000 iterations en 38.109222 secs
$ amplxe-gui pline.0 &
```



# VTune : Line & Assembly

Intel VTune Amplifier XE 2015

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

Analysis Target: Analysis Type: Collection Log: Summary: Bottom-up: Caller/Callee: Top-down Tree: Tasks and Frames: calcul.c

| S. Li. | Source   | CPU Time: Total               |      |    |       |      | Address      | So. Line | Assembly                                 | Effective Time |  |  |
|--------|--|-------------------------------|------|----|-------|------|--------------|----------|--|----------------|--|--|
|        |  | Effective Time by Utilization |      |    |       |      |              |          |  | Effective Time |  |  |
|        |  | Idle                          | Poor | Ok | Ideal | Over | Idle Poor Ok |          |  |                |  |  |
| 53     | free( *pu_exact );                             | 0.0%                          |      |    |       |      | 0x4024c8     | 69       | lea (\$rbx,%r9,1), %r14d                 | 0.0%           |  |  |
| 54     |  | 0.0%                          |      |    |       |      | 0x4024cc     | 67       | movsxd %r9d, %r9                         | 0.0%           |  |  |
| 55     | free( f );                                     | 0.0%                          |      |    |       |      | 0x4024cf     | 67       | movl %ebx, -0x10(%rsp)                   | 0.0%           |  |  |
| 56     | }  | 0.0%                          |      |    |       |      | 0x4024d3     | 67       | movq %rdi, -0x18(%rsp)                   | 0.0%           |  |  |
| 57     |  | 0.0%                          |      |    |       |      | 0x4024d8     | 67       | movq -0x20(%rsp), %rbx                   | 0.0%           |  |  |
| 58     | /*   | 0.0%                          |      |    |       |      | 0x4024dd     | 67       | lea (%rdi,%r12,8), %r12                  | 0.0%           |  |  |
| 59     | * Calcul de la solution u_nouveau a l'iterati  | 0.0%                          |      |    |       |      | 0x4024e1     | 67       | movq -0x28(%rsp), %rbp                   | 0.0%           |  |  |
| 60     | */   | 0.0%                          |      |    |       |      | 0x4024e6     | 67       | lea (%rdi,%r9,8), %r11                   | 0.0%           |  |  |
| 61     | void calcul(double *u, double *u_nouveau) {    | 0.0%                          |      |    |       |      | 0x4024ea     | 67       | nopw %ax, (%rax,%rax,1)                  | 0.0%           |  |  |
| 62     | int iterx, itery;                              | 0.0%                          |      |    |       |      | 0x4024f0     |          | Block 5:                                 | 0.0%           |  |  |
| 63     |  | 0.0%                          |      |    |       |      | 0x4024f0     |          |  | 0.0%           |  |  |
| 64     | for (itery=sy; itery<ey+1; itery++) {          | 0.1%                          |      |    |       |      | 0x4024f0     | 67       | vmovsdq 0x38(%r12), %xmm3                | 0.0%           |  |  |
| 65     | for (iterx=sx; iterx<ex+1; iterx++) {          | 27.0%                         |      |    |       |      | 0x4024f7     | 68       | lea (%r14,%r13,1), %edi                  | 11.4%          |  |  |
| 66     | u_nouveau[IDX(iterx, itery)] =                 | 0.0%                          |      |    |       |      | 0x4024fb     | 69       |  |                |  |  |
| 67     | coef[0] * ( coef[1]*(u[IDX(iterx+1, itery      | 0.0%                          |      |    |       |      | 0x4024fe     | 65       |  |                |  |  |
| 68     | + coef[2]*(u[IDX(iterx, itery+1)])             | 43.8%                         |      |    |       |      | 0x402501     | 67       |  |                |  |  |
| 69     | - f[IDX(iterx, itery)]);                       | 17.1%                         |      |    |       |      | 0x402507     | 68       | vmovsdq 0x28(%rbx,%rdi,8), %xmm5         | 23.7%          |  |  |
| 70     | }  | 0.0%                          |      |    |       |      | 0x40250d     | 65       | lea (%r12,%rbp,8), %r12                  | 0.0%           |  |  |
| 71     | }  | 0.0%                          |      |    |       |      | 0x402511     | 67       | vmlsd %xmm4, %xmm1, %xmm7                | 0.0%           |  |  |
| 72     | }  | 0.0%                          |      |    |       |      | 0x402515     | 68       | vaddsdq 0x18(%rbx,%rdi,8), %xmm5, %xmm6  | 7.2%           |  |  |
| 73     |  | 0.0%                          |      |    |       |      | 0x40251b     | 68       | vmlsd %xmm6, %xmm0, %xmm8                | 0.1%           |  |  |
| 74     | /*   | 0.0%                          |      |    |       |      | 0x40251f     | 65       | lea (%r11,%rbp,8), %r11                  | 0.0%           |  |  |
| 75     | * Affichage                                    | 0.0%                          |      |    |       |      | 0x402523     | 68       | vaddsd %xmm8, %xmm7, %xmm9               | 1.4%           |  |  |
| 76     | */   | 0.0%                          |      |    |       |      | 0x402528     | 69       | vsubsdq 0x20(%rdx,%rdi,8), %xmm9, %xmm10 | 2.8%           |  |  |
| 77     | void sortie_resultats(double *u, double *u_exa | 0.0%                          |      |    |       |      | 0x40252e     | 69       | vmlsd %xmm10, %xmm2, %xmm11              | 13.8%          |  |  |
| 78     | int itery;                                     | 0.0%                          |      |    |       |      | 0x402533     | 65       | lea 0x3(%r13,%r10,1), %r13d              | 8.0%           |  |  |
| 79     |  | 0.0%                          |      |    |       |      | 0x402538     | 66       | vmovsdq %xmm11, 0x20(%rsi,%rdi,8)        | 0.0%           |  |  |
| 80     | printf("Solution exacte u_exact - Solution c   | 0.0%                          |      |    |       |      | 0x40253e     | 65       | cmp %eax, %r15d                          | 18.5%          |  |  |
| 81     | for(itery=sy; itery<ey+1; itery++)             | 0.0%                          |      |    |       |      | 0x402541     | 65       | jb 0x4024f0 <Block 5>                    | 0.0%           |  |  |
| 82     | printf("%12.5e - %12.5e\n", u_exact[IDX(1,     | 0.0%                          |      |    |       |      | 0x402543     |          | Block 6:                                 | 0.0%           |  |  |
| 83     | );   | 0.0%                          |      |    |       |      | 0x402543     | 64       | inc %r9d                                 | 0.0%           |  |  |
|        |  |                               |      |    |       |      | 0x402546     | 64       | movl -0x8(%rsp), %ebp                    | 0.0%           |  |  |
|        |  |                               |      |    |       |      | 0x40254a     | 64       | movl -0x10(%rsp), %ebx                   | 0.1%           |  |  |

Selected 1 row(s): 43.8%

Highlighted 10 row(s):

Data Of Interest (CPU Metrics)

★ Viewing 1 of 1 selected stack(s)

100.0% (35.240s of 35.240s)

poisson/calcul - calcul.c

poisson/main+0xd3 - poisson.c:110

libc-2.12.so!\_ll...known-[Unknown]

50% function calcul in a mov op.



## *Addtionnal informations : Command line profile*

```
$ amplxe-cl -report hotspots -r profil.0
```

| Function       | Module                 | CPU Time:Self |
|----------------|------------------------|---------------|
| calcul         | poisson                | 35.220        |
| erreur_globale | poisson                | 2.770         |
| __psm_ep_close | libpsm_infinipath.so.1 | 1.000         |
| read           | libc-2.3.4.so          | 0.070         |
| PMPI_Init      | libmpi.so.1.0.6        | 0.020         |
| strlen         | libc-2.3.4.so          | 0.020         |
| strcpy         | libc-2.3.4.so          | 0.010         |
| __GI_memset    | libc-2.12.so           | 0.010         |
| __IO_vfscanf   | libc-2.3.4.so          | 0.010         |
| __psm_ep_open  | libpsm_infinipath.so.1 | 0.010         |



# VTune : Advanced Metrics

The screenshot shows the Intel VTune Amplifier XE 2015 interface. The main window displays performance metrics for a 'General Exploration' session. The 'Elapsed Time' is 40.740s. Key metrics include:

- Cycle Per Instruction (CPI):** 2.284 (highlighted in red). A blue circle is drawn around this value, with a link to <https://software.intel.com/en-us/node/544398>.
- Back-end Bound:** 0.849 (highlighted in red). A blue circle is drawn around this value, with a link to <https://software.intel.com/en-us/node/544419>.
- DTLB Overhead:** 0.482 (highlighted in red). A blue circle is drawn around this value, with a link to <https://software.intel.com/en-us/node/544419>.

Other metrics shown include Clockticks (128,512,192,768), Instructions Retired (56,270,084,405), MUX Reliability (1.000), Paused Time (0s), Filled Pipeline Slots (Retiring: 0.130, Bad Speculation: 0.002), Unfilled Pipeline Slots (Stalls: Back-end Bound: 0.849, DIV Active: 0.003, Flags Merge Stalls: 0.000, Slow LEA Stalls: 0.023), Memory Latency (LLC Load Misses Serviced By Remote DRAM: 0.000, LLC Miss: 0.000, LLC Hit: 0.942, Contested Accesses: 0.000, Data Sharing: 0.000), and Memory Replacements (L1D Replacement Percentage: 1.000, L2 Replacement Percentage: 1.000).

```
$ mpirun -np 1 amplxe-cl -collect general-exploration -r ge ./poisson
Convergence apres 10000 iterations en 38.109222 secs
$ amplxe-gui ge.0 &
```



# Advanced Metrics : Back-End Bound



Search document...



## Intel® VTune™ Amplifier XE 2016 and Intel® VTune™ Amplifier 2016 for Systems Help (Linux\*, OS X\*)

Home

Legal Information

> Introducing the Intel® VTune™ Amplifier

> Getting Started with Intel® VTune™ Amplifier

> Tuning Methodology

> Performance Analysis User's Guide

> Energy Analysis User's Guide

> Troubleshooting

## Back-End Bound

### Metric Description

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound issues.

Parent topic: [CPU Metrics Reference](#)

- [Memory Bandwidth](#)
- [Port Utilization](#)

### See Also

[Reference for Performance Metrics](#)

[Interpreting General Exploration Data](#)



Prev

[Average Loop Trip Count](#)

Next

[Memory Bandwidth](#)



For more complete information about compiler optimizations, see our [Optimization Notice](#).



# Advanced Metrics : DTLB



Search document...



## DTLB Overhead

### Metric Description

In x86 architectures, mappings between virtual and physical memory are facilitated by a page table, which is kept in memory. To minimize references to this table, recently-used portions of the page table are cached in a hierarchy of 'translation look-aside buffers', or TLBs, which are consulted on every virtual address translation. As with data caches, the farther a request has to go to be satisfied, the worse the performance impact. This metric estimates the performance penalty paid for missing the first-level data TLB (DTLB) that includes hitting in the second-level data TLB (STLB) as well as performing a hardware page walk on an STLB miss.

### Possible Issues

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large amounts of frequently-used data.

Parent topic: [L1 Bound](#)

### See Also

Reference for Performance Metrics



[Prev](#)  
4K Aliasing

[Next](#)  
FB Full



Intel® VTune™ Amplifier XE 2016 and Intel® VTune™ Amplifier 2016 for Systems Help (Linux®, OS X®)

[Home](#)

[Legal Information](#)

[Introducing the Intel® VTune™ Amplifier](#)

[Getting Started with Intel® VTune™ Amplifier](#)

[Tuning Methodology](#)

[Performance Analysis User's Guide](#)

[Energy Analysis User's Guide](#)

[Troubleshooting](#)





# Advanced Metrics : Per line profile

| S. Li. | Source   | Clockticks      | Instructions Retired | CPI Rate | MUX Reliability | Filled Pipeline Slots |               | Unfilled       |          |         |               |             |             |             |       |  |
|--------|--|-----------------|----------------------|----------|-----------------|-----------------------|---------------|----------------|----------|---------|---------------|-------------|-------------|-------------|-------|--|
|        |  |                 |                      |          |                 | Retiring              | Bad Specul... | Memory Latency |          |         |               | Memory Re   |             | Back-end    |       |  |
|        |  |                 |                      |          |                 |                       |               | LLC Loa...     | LLC Miss | LLC Hit | DTLB Overhead | Conteste... | Data Sha... | L1D Repl... | L2 Re |  |
| 43     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 44     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 45     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 46     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 47     | /*   |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 48     | * Calcul de la solution u_nouveau a l'iteration n+1        |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 49     | */   |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 50     | void calcul(double *u, double *u_nouveau) {                | 0               | 2,000,003            | 0.000    |                 | 0.000                 | 0.000         |                |          |         |               |             |             |             |       |  |
| 51     | int iterx, itery;  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 52     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 53     | for (itery=sy; itery<ey+1; itery++) {                      | 94,000,141      | 28,000,042           | 3.357    |                 | 0.080                 | 0.000         |                |          |         |               |             |             |             |       |  |
| 54     | for (iterx=sx; iterx<ex+1; iterx++) {                      | 36,254,054,3... | 9,784,014,676        | 3.705    |                 | 0.100                 | 0.000         |                |          |         |               |             |             |             |       |  |
| 55     | u_nouveau[IDX(iterx, itery)] =                             |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 56     | coef[0] * ( coef[1]*(u[IDX(iterx+1, itery)]+u[IDX(iterx-1, | 4,000,006       | 2,000,003            | 2.000    |                 | 0.000                 | 0.000         |                |          |         |               |             |             |             |       |  |
| 57     | + coef[2]*(u[IDX(iterx, itery+1)]+u[IDX(iterx, iter        | 57,680,086,5... | 14,964,022,...       | 3.855    |                 | 0.073                 | 0.008         |                |          |         |               |             |             |             |       |  |
| 58     | - f[IDX(iterx, itery)]);                                   | 22,088,033,1... | 9,898,014,847        | 2.232    |                 | 0.081                 | 0.014         |                |          |         |               |             |             |             |       |  |
| 59     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 60     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 61     | }  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 62     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 63     | /*   |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 64     | * Affichage  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 65     | */   |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 66     | void sortie_resultats(double *u, double *u_exact) {        |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 67     | int itery;   |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 68     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 69     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 70     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 71     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |
| 72     |  |                 |                      |          |                 |                       |               |                |          |         |               |             |             |             |       |  |

```
$ mpirun -np 1 amplxe-cl -collect general-exploration -r ge ./poisson  
Convergence apres 10000 iterations en 38.109222 secs  
$ amplxe-gui ge.0 &
```



# Sequential optimisations

---

■ Hotspot #1 :

■ Hotspot #2 :

■ Can we go further ?

- Hotspot #3

- And further ?



# Sequential optimisations

- Hotspot #1 :
  - Stencil : DTLB
    - Invert loops in **calcul**
  
- Hotspot #2 :
  - Convergence criterion : Vectorisable
    - delete **#pragma -novector**
  
- Can we go further ?
  - Hotspot #3 : Back on stencil
    - using **-no-vec** compiler option : no impact on **calcul**
    - stencil vectorisable
      - add **#pragma simd** on the internal loop
  
  - Does it worth to call **erreur\_globale** for each iteration ?



## *Additional informations : MPI Context*

Hotspots :

```
$ mpirun -np 2 amplxe-cl -collect hotspots -r pmpi ./poisson
$ ls pmpi.*/*.amplxe
pmpi.0/pmpi.0.amplxe
pmpi.1/pmpi.1.amplxe
```

**Per MPI processus profile**

Advanced metrics through a dedicated driver :

```
$ mpirun -np 2 amplxe-cl -collect general-exploration -r gempi ./poisson
amplxe: Error: PMU resource(s) currently being used by another profiling
tool or process.
amplxe: Collection failed.
amplxe: Internal Error
```

MPMD like mode :

```
$ mpirun -np 1 amplxe-cl -collect general-exploration -r gempi ./poisson
: -np 1 ./poisson
```



# Additionnal informations : Thread concurrency

Welcome omp8\_tc\_1... x

Concurrency Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2015

Analysis Target Analysis Type Collection Log Summary Bottom-up Caller/Callee Top-down Tree Tasks and Frames

### Elapsed Time: 22.293s

Total Thread Count: 9  
Overhead Time: 0.406s  
Spin Time: 114.959s  
CPU Time: 170.498s  
Paused Time: 0s

A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait parameters, changing the lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.

### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

| Function                                   | CPU Time |
|--|----------|
| <a href="#">_kmp_launch_thread</a>         | 114.569s |
| <a href="#">calculomp\$parallel_for@53</a> | 38.340s  |
| <a href="#">erreur_globale</a>             | 15.593s  |
| <a href="#">_psm_ep_close</a>              | 1.000s   |
| <a href="#">_kmp_join_call</a>             | 0.596s   |
| [Others]                                   | 0.400s   |

### CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.

| Simultaneously Utilized Logical CPUs | Elapsed Time (s) |
|--------------------------------------|------------------|
| 0                                    | 0.0              |
| 1                                    | 4.5              |
| 2                                    | 6.5              |
| 3                                    | 5.5              |
| 4                                    | 3.5              |
| 5                                    | 1.5              |
| 6                                    | 0.5              |
| 7                                    | 0.0              |
| 8                                    | 0.0              |
| 9                                    | 0.0              |
| 10                                   | 0.0              |
| 11                                   | 0.0              |
| 12                                   | 0.0              |
| 13                                   | 0.0              |
| 14                                   | 0.0              |
| 15                                   | 0.0              |
| 16                                   | 0.0              |
| 17                                   | 0.0              |
| 18                                   | 0.0              |

### Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: /noisson

```
$ export OMP_NUM_THREADS=8
```

```
$ mpirun -bind-to-socket -np 1 amplxe-cl -collect concurrency -r omp ./poisson
```



# Addtionnal informations : Memory counters

**Elapsed Time:** 47.620s

CPU Time: 47.251s  
Paused Time: 0s

**Hardware Events**

| Hardware Event Type                                     | Hardware Event Count | Hardware Event Sample Count | Events Per Sample |
|---|----------------------|-----------------------------|-------------------|
| <a href="#">CPU_CLK_UNHALTED.REF_TSC</a>                | 122,852,184,278      | 61,426                      | 2000003           |
| <a href="#">MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT</a>  | 0                    | 0                           | 20011             |
| <a href="#">MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM</a> | 0                    | 0                           | 20011             |
| <a href="#">MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS</a> | 0                    | 0                           | 20011             |
| <a href="#">MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE</a> | 334,810,044          | 1,116                       | 100003            |
| <a href="#">MEM_LOAD_UOPS_RETIREDD.HIT_LFB_PS</a>       | 16,260,187,791       | 54,199                      | 100003            |
| <a href="#">MEM_LOAD_UOPS_RETIREDD.L1_HIT_PS</a>        | 109,656,164,484      | 18,276                      | 2000003           |
| <a href="#">MEM_LOAD_UOPS_RETIREDD.L2_HIT_PS</a>        | 257,707,731          | 859                         | 100003            |
| <a href="#">MEM_LOAD_UOPS_RETIREDD.LLC_MISS</a>         | 0                    | 0                           | 100007            |
| <a href="#">MEM_UOPS_RETIREDD.ALL_LOADS_PS</a>          | 127,062,190,593      | 21,177                      | 2000003           |
| <a href="#">MEM_UOPS_RETIREDD.ALL_STORES_PS</a>         | 19,668,029,502       | 3,278                       | 2000003           |

**Collection and Platform Info**

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: ./poisson  
User Name: jderouillat  
Operating System: 2.6.32-431.el6.x86\_64 CentOS release 6.5 (Final)  
MPI Process Rank: 0  
Computer Name: poincare005-adm  
Result Size: 28 MB  
Collection start time: 14:17:01 09/06/2016 UTC  
Collection stop time: 14:17:48 09/06/2016 UTC

**CPU**

Name: Intel(R) Xeon(R) E5 processor  
Frequency: 2,6 GHz  
Logical CPU Count: 16

```
$ mpirun -bind-to-socket -np 1 amplxe-cl -collect snb-memory-access -r mem ./poisson
```



# And more ...

**Choose Analysis Type**

Analysis Type

- Algorithm Analysis
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks and Waits
- Microarchitecture Analysis
  - General Exploration**
  - Bandwidth
- CPU Specific Analysis
  - Intel Core 2 Processor Analysis
  - Nehalem / Westmere Analysis
  - Sandy Bridge Analysis
    - Access Contention
    - Branch Analysis
    - Client Analysis
    - Core Port Saturation
    - Cycles and uOps
    - Memory Access
    - Port Saturation
  - Haswell Analysis
- Knights Corner Platform Analysis
- Custom Analysis

**General Exploration** Copy

Analyze general issues affecting the performance of your application. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

Collect stacks

Analyze memory bandwidth

**Details**

Events configured for CPU: Intel(R) Xeon(R) E5 processor

NOTE: For analysis purposes, Intel VTune Amplifier XE 2015 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the Project Properties dialog.

| Event Name                       | Sample After | LBR Filter |     |
|----------------------------------|--------------|------------|-----|
| ARITH.FPU_DIV_ACTIVE             | 2000003      | None       | Cyc |
| BR_MISP_RETIRED.ALL_BRANCHES_PS  | 400009       | None       | Mis |
| CPU_CLK_UNHALTED.REF_TSC         | 2000003      |            | Ref |
| CPU_CLK_UNHALTED.THREAD          | 2000003      |            | Cor |
| CPU_CLK_UNHALTED.THREAD_P        | 2000003      | None       | Thr |
| DSB2MITE_SWITCHES.PENALTY_CYCLES | 2000003      | None       | Dec |

Analyze GPU usage (Intel HD Graphics only)

Collect stacks

Stack size, in pages:

Estimate call counts

Chipset events:

Linux Ftrace events:

Analyze memory bandwidth

Analyze user tasks

Event mode:

Analyze active power consumption

Analyze idle power consumption

**Start**

**Start Paused**

Project Properties

**Command Line...**

Explore the GUI :

```
$ amplxe-gui &
```

Extract command lines for  
custom analysis



**Scorep, Scalasca, Vampir**

---



# Scorep : instrumentation, profiling and tracing

Scorep : Scalable Performance Measurement Infrastructure for Parallel Codes

## Objectives

- Automatic instrumentation at compilation
- Generation of call-path profiles and event traces
- Recording time, number of visits, exchanged data volumes, hardware counters, ...
- A wide range of associated softwares for results analyse and visualization (Scalasca, Cube, Vampir, TAU, ...)
- Supports MPI, OpenMP, basic CUDA.

## Pros & Cons

- + Open Source, in continuous development, well supported
- + Scalable, portable on most HPC platforms
- + Compatible with the main compilers
- Introduces an overhead in time and memory, especially for the trace collection. (classic tradeoff of the instrumentation)
- Some open issues and remaining case-specific bugs (but refer to advantage

1)



# Scorep exercise : instrumentation and first profile

Environment - on Poincaré :

```
module use /gpfslocal/pub/vihps/UNITE/local
module load UNITE
module load scorep/2.0.2-intel13-openmpi163 scalasca/2.3.1-intel13-
openmpi163
```

Instrumentation : by prefixing the compilation and linking commands

```
Ex: mpif90 -> scorep mpif90
gcc -> scorep gcc
```

Profile generation : automatic by running a normal execution

→ Generates a folder `scorep-yyyymmdd_hhmm_####` containing a `profile.cubex` file

First analysis : using `scorep-score` command :

```
scorep-score scorep-#####/profile.cubex
```

Gives a summary analysis with the main informations

Second analysis : usins Scalasca analysis and Cube interface

```
scalasca --examine scorep-#####/
```

opens detailed results in Cube graphical interface



# Scorep-score : Summary Analysis Result Scoring

```
-bash-4.1$ scorep-score scorep-20160613_1739_7420448901032141/profile.cubex
```

```
Estimated aggregate size of event trace:          6GB
Estimated requirements for largest trace buffer (max_buf): 376MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 378MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=378MB to avoid intermediate flushes
or reduce requirements using USR regions filters.)
```

| flt | type | max_buf[B]  | visits      | time[s] | time[%] | time/visit[us] | region |
|-----|------|-------------|-------------|---------|---------|----------------|--------|
|     | ALL  | 393,966,035 | 116,461,281 | 1968.70 | 100.0   | 16.90          | ALL    |
|     | MPI  | 328,456,523 | 72,788,288  | 1129.39 | 57.4    | 15.52          | MPI    |
|     | COM  | 43,673,016  | 29,115,344  | 271.79  | 13.8    | 9.34           | COM    |
|     | USR  | 21,836,496  | 14,557,649  | 567.51  | 28.8    | 38.98          | USR    |

Region/callpath classification :

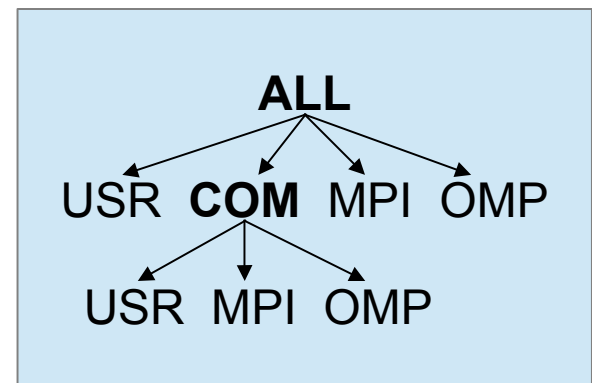
MPI : pure MPI library functions

OMP : pure OpenMP functions/refions

USR : user-level source local computation

COM : 'combined' USR + OMP/MPI

ANY/ALL : aggregation of all regions





# Cube interface (through scalasca --examine)

The screenshot displays the Cube-4.3.4 interface with the following components:

- Top Panel:** Title bar "Cube-4.3.4: scorep-20160613\_1739\_7420448901032141/summary.cubex" and menu items "File", "Display", "Plugins", "Help".
- Left Panel (Metric tree):** Shows absolute values for various metrics:
  - 1968.70 Time (sec)
  - 1.16e8 Visits (occ)
  - 8.06e10 Bytes transferred (bytes)
  - 64 MPI file operations (occ)
  - 424.71 Computational imbalance (sec)
  - 0.00 Minimum Inclusive Time (sec)
  - 123.33 Maximum Inclusive Time (sec)
  - 0 ALLOCATION\_SIZE (bytes)
  - 0 DEALLOCATION\_SIZE (bytes)
  - 0 bytes\_leaked (bytes)
  - 0.00 maximum\_heap\_memory\_allocated (bytes)
- Middle Panel (Call tree):** Shows a hierarchical view of the call stack:
  - 92.31 MAIN\_
  - 5.62 parallel.initialisation\_mpi\_
  - 2.14 parallel.creation\_topologie\_
  - 0.56 parallel.domaine\_
  - 0.07 calcul\_poisson.initialisation\_
  - 0.05 parallel.voisinage\_
  - 0.10 parallel.type\_derive\_
  - 13.89 parallel.communication\_
  - 177.88 MPI\_Sendrecv
  - 567.34 calcul\_poisson.calcul\_
  - 163.03 parallel.erreur\_globale\_
  - 943.52 MPI\_Allreduce
  - 0.00 calcul\_poisson.sortie\_resultats\_
  - 2.17 parallel.ecrire\_mpi\_
  - 0.02 parallel.finalisation\_mpi\_
- Right Panel (System tree):** Shows a tree view of the system components:
  - machine Linux
  - node poincare058-adm
  - 4.10 MPI Rank 0
  - 4.04 MPI Rank 1
  - 4.16 MPI Rank 2
  - 10.79 MPI Rank 3
  - 4.14 MPI Rank 4
  - 4.11 MPI Rank 5
  - 4.05 MPI Rank 6
  - 10.74 MPI Rank 7
  - 4.16 MPI Rank 8
  - 4.05 MPI Rank 9
  - 4.09 MPI Rank 10
  - 10.85 MPI Rank 11
  - 4.10 MPI Rank 12
  - 4.02 MPI Rank 13
  - 4.12 MPI Rank 14
  - 10.81 MPI Rank 15
- Bottom Panel:** Progress bars and a color scale. The "Time" metric is selected, showing a range from 0.00 to 1968.70 (100.00%).



# Scorep : Trace Collection (1)

Tracing give more insight into the code behavior

It often generates too much collected data, resulting in huge time and memory overheads

First step : estimate this cost

- The scorep scoring gives an estimation of the needed memory buffer and trace files size.
- -r option shows all regions

External example : code X running on Curie on 288 processes

```
me@curie90 $ scorep-score -r scorep-20150701_1639_1289782540682648/profile.cube
Estimated aggregate size of event trace:          15TB
Estimated requirements for largest trace buffer (max_buf): 64GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 64GB
(hint: When tracing set SCOREP_TOTAL_MEMORY=64GB to avoid intermediate flushes
or reduce requirements using USR regions filters.)
```

Tracing requires 15TB on disk  
64GB buffer on each process

| flt | type  | max_buf[B]     | visits          | time[s]   | time[%] | time/visit[us] | region               |
|-----|-------|----------------|-----------------|-----------|---------|----------------|----------------------|
|     | ALL   | 68,445,407,406 | 684,452,054,099 | 308648.07 | 100.0   | 0.45           | ALL                  |
|     | USR   | 68,436,752,472 | 684,392,045,176 | 118937.03 | 38.5    | 0.17           | USR                  |
|     | MPI   | 8,701,920      | 59,381,659      | 76915.79  | 24.9    | 1295.28        | MPI                  |
|     | COM   | 52,272         | 627,264         | 112795.25 | 36.5    | 179821.02      | COM                  |
|     | USR   | 68,344,181,664 | 683,280,328,752 | 117688.27 | 38.1    | 0.17           | mod_kernel.kernel_   |
|     | USR   | 92,665,344     | 1,111,120,000   | 230.54    | 0.1     | 0.21           | mod_intgr.binsearch_ |
|     | MPI   | 2,463,231      | 8,195,263       | 26.98     | 0.0     | 3.29           | MPI_Isend            |
|     | MPI   | 2,325,231      | 8,195,268       | 18.44     | 0.0     | 2.25           | MPI_Irecv            |
|     | MPI   | 2,197,704      | 26,271,152      | 4.55      | 0.0     | 0.17           | MPI_Cart_rank        |
|     | MPI   | 1,452,744      | 14,908,466      | 40192.02  | 13.0    | 2695.92        | MPI_Waitall          |
|     | MPI   | 407,360        | 1,543,680       | 30548.99  | 9.9     | 19789.72       | MPI_Sendrecv         |
|     | MPI   | 71,750         | 1,435           | 0.22      | 0.0     | 152.44         | MPI_Ssend            |
|     | USR   | 46,128         | 430,000         | 0.21      | 0.0     | 0.48           | mod_intgr.sumrk3_    |
|     | (...) |                |                 |           |         |                |                      |



# Scorep : Trace Collection (2)

Observations on the overhead :

- Some USR regions, like kernel functions, called too many times, are problematic for scorep, generating too much overhead due to their instrumentation
- Filtering them out of the collection can improve the quality of the profile collection, and is necessary for the trace collection
- Other ways exist to reduce requirements. Ex :reducing the number of iteration

Filtering : by using a filter file

- Scorep scoring with `-f` option allows a new estimation without re-running

Example : code X

```
me@curie90 $ cat scorep.filt
SCOREP_REGION_NAMES_BEGIN
EXCLUDE
  mod_kernel.kernel*
  mod_intgr.binsear*
SCOREP_REGION_NAMES_END

me@curie90 $ scorep-score -f scorep.filt scorep-20150701_1639_12897825.../profile.cubex

Estimated aggregate size of event trace:                2176MB
Estimated requirements for largest trace buffer (max_buf): 9MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):    11MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=11MB to avoid intermediate flushes
or reduce requirements using USR regions filters.)
(...)
```

Trace files reduced to 2 GB

Buffer needed on each process to avoid intermediate flushes < 15M



# Scorep : Trace Collection (3) - Exercise

■ Run the trace collection : you need to

- Define the following variables :
  - SCAN\_ANALYZE\_OPTS=--time-correct
  - SCOREP\_FILTERING\_FILE=#the name of the filter file if needed#
  - SCOREP\_TOTAL\_MEMORY=#the required memory estimated by scorep-score#
- Prefix the execution line with 'scalasca --analyse -t'

■ First analysis : on Cube interface (scalasca --examine scalasca\_trace\_folder)

-> additional metrics such as Late Sender / Late Receiver cases quantification

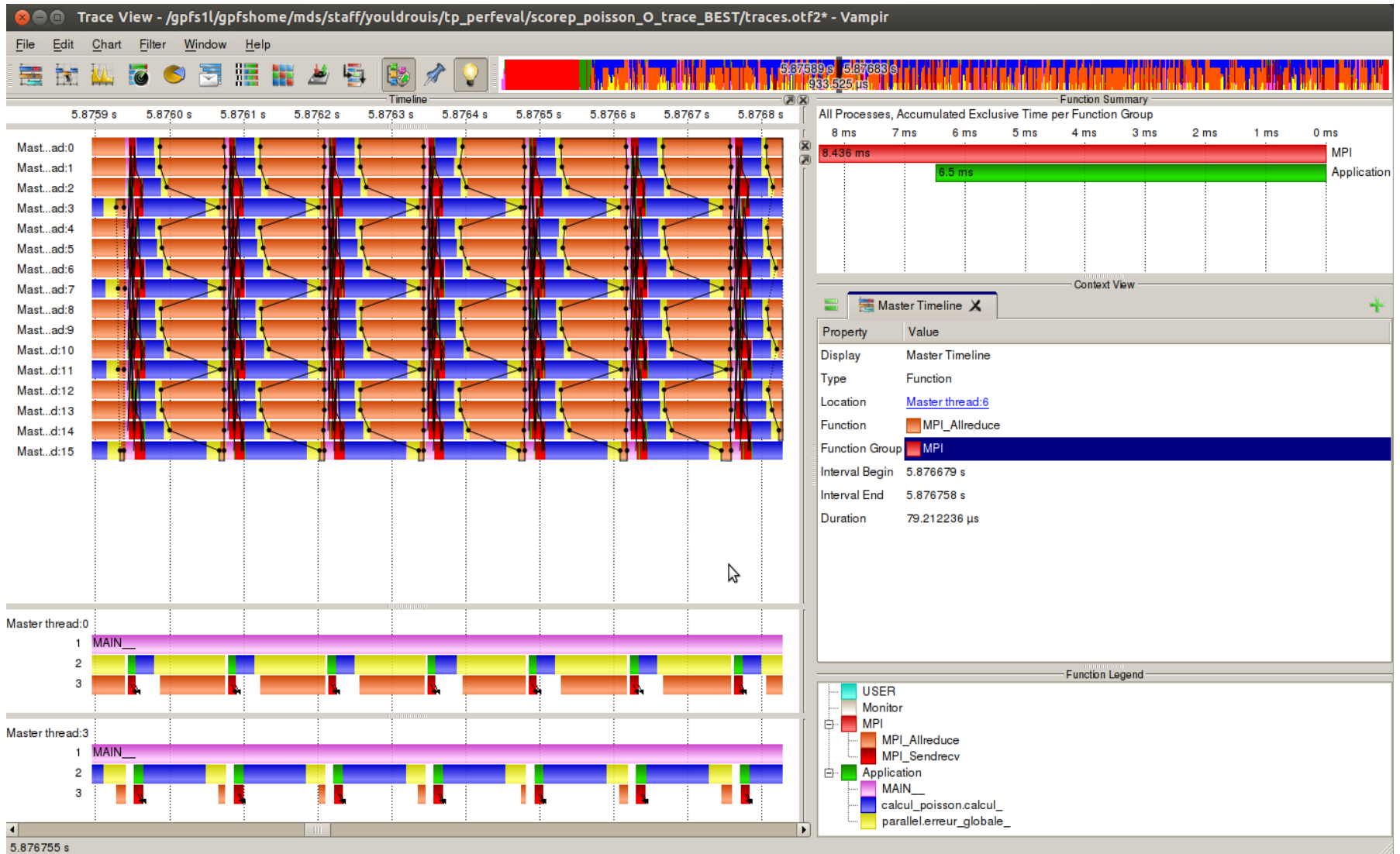
■ Second analysis : visualization with Vampir

- Load vampir module
- Open traces.otf2 file in the Vampir interface
  - » vampir scorep\_poisson\_0\_trace/traces.otf2 &
- Zoom in on the timeline, explore
- Color different regions for better visualization
- Check the 'Process timeline', the 'Communication matrix', and other charts available in the charts menu

— Got some idea about the imbalance reasons?



# Trace visualisation with Vampir

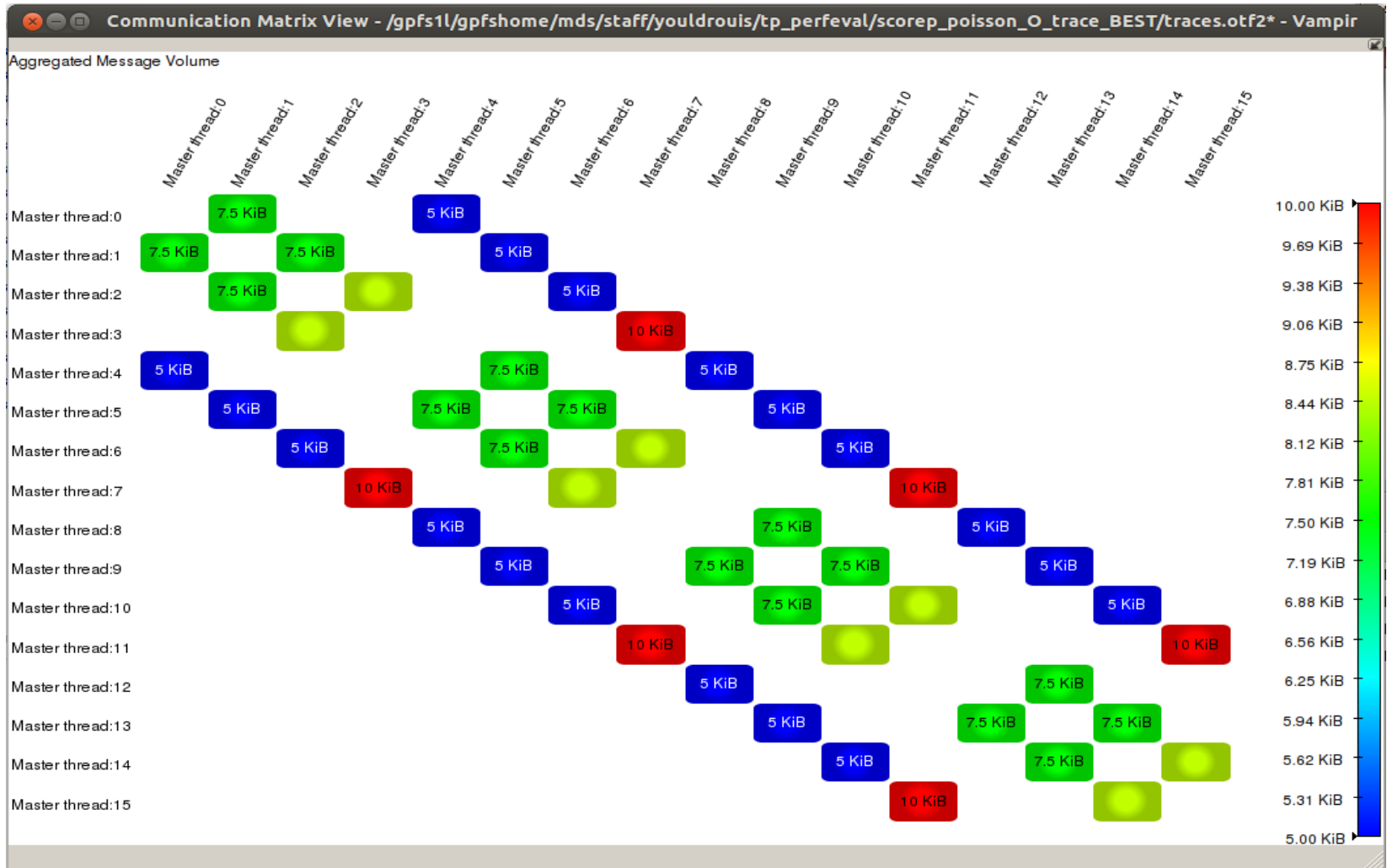


Running the trace collection needs :

- Prefixing the execution command with « scalasca -t »



# Trace visualisation with Vampir (2)



Running the trace collection needs :

- Prefixing the execution command with « scalasca -t »



PAPI

---



# PAPI : Performance Application Programming Interface

Processors are equipped with a big number of counters

PAPI is a portable library that gives easy access to these counters

PAPI can be used as :

- libraries for advanced use with C, Fortran, ...
- through end-user perf tools providing easy collection and visualization

Can provide insight into :

- Memory, L1, L2, L3 Cache and tlb behaviors
- Cycles, Instructions, Pipeline stalls, Branch behaviors
- Floating point operations, ...

Downsides :

- Only 4 to 6 counters can be accessed at the same time
- Simultaneous access to 2 available counters can be problematic (check with `papi_command_line`)
- The availability of each counter depends on the hardware (`papi_avail`)
- Some hardware counters are erroneous (ex : FP ops on Sandy Bridge)
- Increases size and buffer requirement of the profile or trace collection



# Importance of data access performance



## Data access

Number of cycles per data access depending on its location

| Location | Access                   |
|----------|--------------------------|
| L1       | ~4 cycles                |
| L2       | ~10 cycles               |
| L3       | ~40-65 cycles            |
| memory   | 60ns<br>(150-200 cycles) |

Source <http://software.intel.com/en-us/forums/topic/287236>



Bad data access has a huge effect on the performance



Causes instruction pipelines stalls, thus reduction of completed instructions per cycle.



# PAPI counters collection with Scorep - exercise

Load papi module `papi/5.3.0_intel13`

Choose set of PAPI counters to collect

- Browse available preset counters with `papi_avail`
- Check compatibility of a chosen set with `papi_command_line`
  - Ex : `papi_command_line PAPI_L1_DCM PAPI_L2_DCM PAPI_TOT_CYC`

Run profile collection :

- Define the `SCOREP_METRIC_PAPI` variable prior to the execution line:
  - Ex : `export SCOREP_METRIC_PAPI=list of counters separated with commas`

Open the profile in Cube (`scalasca --examine`)

Run trace collection :

- Check first the requirements with `scorep-score -c number_of_papi_metrics`
- Adapt your run to the requirements
- Define `SCOREP_METRIC_PAPI`

Open the trace in Vampir. Visualize the perf counters in the 'Counter Data Timeline' and the 'Performance Radar' charts





# PAPI metrics in Scalasca trace – Visualization with Vampir





# Memory

---



# Memory instrumentation

■ Ecosystem light :

- Initiative @ :
  - a memory trace included in EZTrace
  - Exascale Lab : MALT (a MAlloc Tracker)
  - MdLS : LIBMTM (Modeling and Tracing Memory Library)
- *Memory bandwidth estimation in Alinea Perf Reports*

■ Linux function : getrusage

```
#include <sys/resource.h>
{
    struct rusage r_usage;
    getrusage(RUSAGE_SELF, &r_usage);
    // r_usage.ru_maxrss Max value only
}
```

■ Read /proc/pid/status

```
$ grep VmRss /proc/pid/status
```



# Memory instrumentation

- Edit `make_inc` to enable `-D_INST_MEM` option, then recompile

```
$ mpirun -np 1 ./poisson 1
```

```
...
```

```
Taille du domaine : ntx=480 nty=400
```

```
...
```

```
Start --- Rang 0 - Memory usage: 24448 Kbytes
```

```
Alloc --- Rang 0 - Memory usage: 30512 Kbytes
```

```
...
```

```
Convergence apres 1 iterations en 0.003954 secs
```

```
Clean --- Rang 0 - Memory usage: 24736 Kbytes
```

- Grid footprint :

- $480 \times 400 \times 8 = 1500$  Kbytes / grid
- x 4 grids



# Memory instrumentation

- Edit `make_inc` to enable `-D_INST_MEM` option, then recompile

```
$ mpirun -np 1 ./poisson 1
```

```
...
```

```
Taille du domaine : ntx=480 nty=400
```

```
...
```

```
Start --- Rang 0 - Memory usage: 24448 Kbytes
```

```
Alloc --- Rang 0 - Memory usage: 30512 Kbytes
```

```
...
```

```
Convergence apres 1 iterations en 0.003954 secs
```

```
Clean --- Rang 0 - Memory usage: 24736 Kbytes
```

- **Memory leaks**

- **MPI ? Load imbalance**