Empowered by Innovation NEC

Atelier
Profilage de codes de calcul

Introduction aux compteurs hardware avec PAPI

Laurent Gatineau
Support applicatif
NEC HPC Europe

Ecole Centrale de Paris

11 juin 2014

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Qu'est-ce que les compteurs hardware ?

Compteurs Hardware: Registres présents dans les processeurs permettant de compter des évènements liés à l'activité des processeurs.

Peut-être liés à un cœur ou à un socket.

Par exemple, compter:

- Le nombre d'instructions.
- Le nombre de cycles en attente ou à travailler.
- Les accès mémoire (cache L1,L2,L3, mémoire, prefetch, …)
- Les branchements, opérations vectorielles, flottantes, …
- Les accès aux caches / cœurs distants.
- Les accès aux liens QPIs.

Auteur: Laurent Gatineau

Qu'est-ce que les compteurs hardware ?

- Le nombre de compteurs dépend du processeur.

Processeur	Nb HC
AMD 6378 (Abu Dhabi)	6
Intel Xeon E5472 (Harpertown)	5
Intel Xeon X5650 (Westmere)	7
Intel Sandy Bridge E5-2670	11
Intel Ivy Bridge E5-2695v2	11

- Les types d'évènements qui peuvent être comptés dépendent du processeur.
- Référence Intel:
  - *Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System Programming Guide, Part 2*
  - Chapitre 18 (96 pages) et 19 (198 pages)

Auteur: Laurent Gatineau

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

PAPI: The Performance API

PAPI: Performance Application Programming Interface

- Harmonise l'accès aux compteurs hardware entre différents processeurs et OS.
- Accessible aux utilisateurs sans privilège particulier.
- Support C,C++ et Fortran.

- http://icl.cs.utk.edu/papi/
- Existe depuis plus de 10 ans…
- PAPI-C (Component PAPI): Extension de PAPI au système.

- Pour Linux:
  - Noyau < 2.6.31: nécessite un patch (perfctr).
  - Noyau >= 2.6.31: utilise l'interface perf_events du kernel.

Auteur: Laurent Gatineau

PAPI: The Performance API

- Deux niveaux d'API:
  - Haut niveau:
    - 8 fonctions.
    - Simplicité.
    - Accès aux évènements prédéfinis.
  - Bas niveau:
    - Accès aux évènement prédéfinis et natifs.
    - Flexibilité.
    - Support Multi-thread.
    - Support Multiplexing.
- Chronomètres PAPI: Temps passé exprimés en microsecondes ou en cycles d'horloge.
- `papi_avail`: programme donnant la liste des évènements disponibles.

Auteur: Laurent Gatineau

PAPI: The Performance API

API haut niveau

Fonction	Commentaire
`PAPI_num_counters`	Nombre de compteurs hardware disponibles
`PAPI_flips`	Mflips/s (Floating Point Instruction rate) + timers
`PAPI_flops`	Mflops/s (Floating Point Operation rate) + timers
`PAPI_ipc`	Instruction par cycle + timers
`PAPI_start_counters`	Commence à compter les évènements
`PAPI_read_counters`	Copie les valeurs des compteurs dans un tableau, et remet les compteurs à zéro
`PAPI_accum_counters`	Ajoute les valeurs des compteurs dans un tableau, et remet les compteurs à zéro
`PAPI_stop_counters`	Arrête de compter les évènements et copie les valeurs des compteurs dans un tableau

Auteur: Laurent Gatineau

PAPI: The Performance API

Chronomètres PAPI

- Temps total: Temps de restitution, inclus les I/Os, les communications, le temps passé dans le noyau, le temps en attente…
- Temps utilisateur: Temps passé en mode utilisateur.

Fonction	Commentaire
`PAPI_get_real_cyc`	Temps total en cycle CPU
`PAPI_get_real_usec`	Temps total en microsecondes
`PAPI_get_virt_cyc`	Temps utilisateur en cycle CPU
`PAPI_get_virt_usec`	Temps total en microsecondes

PAPI: The Performance API

papi_avail

```
gatineaul@service0:~> module load papi
gatineaul@service0:~> papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 5.3.0.0
[...]
Number Hardware Counters : 7
Max Multiplex Counters   : 64
--------------------------------------------------------------------------------

    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  Yes   No   Level 2 instruction cache misses
PAPI_L3_DCM  0x80000004  No    No   Level 3 data cache misses
PAPI_L3_ICM  0x80000005  No    No   Level 3 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes   Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes   No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  Yes   No   Level 3 cache misses
```

PAPI: The Performance API

papi_avail -d (détail)

```
PAPI_L1_DCM     0x80000000      1       |L1D cache misses|
 |Level 1 data cache misses|
 ||
 |NOT_DERIVED|
 ||
 Native Code[0]: 0x40000006 |L1D:REPL|

PAPI_L2_DCM     0x80000002      2       |L2D cache misses|
 |Level 2 data cache misses|
 ||
 |DERIVED_ADD|
 ||
 Native Code[0]: 0x40000007 |L2_RQSTS:LD_MISS|
 Native Code[1]: 0x40000008 |L2_RQSTS:RFO_MISS|

PAPI_VEC_SP     0x80000069      2       |SP Vector/SIMD instr|
 |Single precision vector/SIMD instructions|
 ||
 |DERIVED_POSTFIX|
 |N0|4|*|N1|8|*|+||
 Native Code[0]: 0x40000018 |FP_COMP_OPS_EXE:SSE_PACKED_SINGLE|
 Native Code[1]: 0x40000019 |SIMD_FP_256:PACKED_SINGLE|
```


Auteur: Laurent Gatineau

PAPI: The Performance API

- Premier programme: afficher le nombre de compteurs hardware.

- Notes:
  - Si `number < PAPI_OK` alors erreur…
  - Les fichiers d'en-têtes sont là: `/opt/san/profiling/papi/5.3.0/include`
  - Les librairies sont là (`-lpapi`): `/opt/san/profiling/papi/5.3.0/lib`

Interface C	Interface Fortran
`#include <papi.h>`	`include "f90papi.h"`
`int PAPI_num_counters(void);`	`PAPIF_num_counters(C_INT number)`
`void PAPI_perror(char *s);`	`PAPIF_perror(C_STRING message)`

Auteur: Laurent Gatineau

PAPI: The Performance API

```c
#include <stdio.h>
#include <papi.h>

int main(int argc, char *argv[])
{
    int nb_hwc;

    nb_hwc = PAPI_num_counters();
    if (nb_hwc < PAPI_OK) {
        PAPI_perror("PAPI_num_counters");
        return 1;
    }

    printf("%d available hardware"
           " counter(s).\n", nb_hwc);

    return 0;
}
```

```fortran
PROGRAM TP1_F

  USE iso_c_binding
  IMPLICIT NONE

  include "f90papi.h"

  INTEGER(C_INT) :: nb_hwc

  CALL PAPIF_num_counters(nb_hwc)

  IF (nb_hwc.lt.PAPI_OK) THEN
    CALL PAPIF_perror('PAPIF_num_counters')
    STOP
  END IF

  WRITE(6, '(I0,A)') nb_hwc, &
    " available hardware counter(s)."

END PROGRAM TP1_F
```

Sources dans: /home/gatineaul/TP_PAPI/TP1

Auteur: Laurent Gatineau

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Nombre de cycles et nombre d'instructions

- Connaitre le nombre de cycles et le nombre d'instructions permet d'avoir le `CPI`: *Cycle Per Instruction*.

- Sur les processeurs actuels:
  - 1 cycle par instructions c'est déjà bien…
  - Vectorisation `SIMD`: < 1 cycle par instruction (si la bande passante mémoire le permet…).
  - Petits noyaux de calcul: 0.3 – 0.5 cycle par instruction.

- Fonction de haut niveau `PAPI_ipc`…
- Mais aussi, les évènements:
  - `PAPI_TOT_INS`
  - `PAPI_TOT_CYC`

Auteur: Laurent Gatineau

Nombre de cycles et nombre d'instructions

Deuxième programme: calculer le CPI !

- `PAPI_TOT_INS`
- `PAPI_TOT_CYC`
- `events`: Tableau contenant la liste des évènements.
- `values`: Tableau qui contiendra la valeur des compteurs.
- Code de retour < `PAPI_OK` indique une erreur.

- SAXPY: `y(i) = a * x(i) + y(i)`

Interface C

```
int PAPI_start_counters(int *events, int array_len);
int PAPI_stop_counters(long_long *values, int array_len);
```

Interface Fortran

```
PAPIF_start_counters(C_INT(*) events, C_INT array_len, C_INT check)
PAPIF_stop_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT check)
```


Auteur: Laurent Gatineau

Nombre de cycles et nombre d'instructions

```
[…]
#define NUM_EVENTS 2

int main(int argc, char *argv[])
{
    int i, nb_hwc, retval;
    int events[NUM_EVENTS] = {PAPI_TOT_INS,
                              PAPI_TOT_CYC};
    char event_name[PAPI_MAX_STR_LEN];
    long_long values[NUM_EVENTS];

    retval = PAPI_start_counters(events,
             NUM_EVENTS);

    […]

    retval = PAPI_stop_counters(values,
             NUM_EVENTS);

    for (i = 0; i < NUM_EVENTS; i++) {
        PAPI_event_code_to_name(events[i],
                                event_name);
        printf("%s: %lld\n", event_name,
               values[i]);
    }
    printf("CPI: %.2f\n", ((double) values[1])
           / ((double) values[0]));
```

```
[…]
 INTEGER, PARAMETER :: NUM_EVENTS = 2

 INTEGER(C_INT) :: nb_hwc, retval
 INTEGER(C_INT), DIMENSION(NUM_EVENTS) :: &
      events = (/ PAPI_TOT_INS, PAPI_TOT_CYC/)
 INTEGER(C_LONG_LONG), DIMENSION(NUM_EVENTS) &
      :: values
 CHARACTER(LEN=PAPI_MAX_STR_LEN) :: event_name

 CALL PAPIF_start_counters(events, &
      NUM_EVENTS, retval)

 […]

 CALL PAPIF_stop_counters(values, &
      NUM_EVENTS, retval)

 DO i = 1, NUM_EVENTS
   CALL PAPIF_event_code_to_name(events(i), &
        event_name, retval)
   WRITE(6, '(A,A,I0)') TRIM(event_name), &
        ": ", values(i)
 END DO

 WRITE(6, '(A,F4.2)') 'CPI: ', &
       REAL(values(2)) / REAL(values(1))
```

Sources dans: /home/gatineaul/TP_PAPI/TP2

Auteur: Laurent Gatineau

Nombre de cycles et nombre d'instructions

- Compiler le code avec les options suivantes:
  `-ftree-vectorize -msse`

- Que ce passe-t-il ?

```
gcc tp2_c.c -o tp2_c -O2 …
11 available hardware counter(s).
PAPI_TOT_INS: 70000195612
PAPI_TOT_CYC: 20950391439
CPI: 0.30

real    0m6.391s
user    0m6.383s
sys     0m0.004s
```

```
gcc tp2_c.c -o tp2_c -O2 -ftree-vectorize -msse …
11 available hardware counter(s).
PAPI_TOT_INS: 17500058470
PAPI_TOT_CYC: 10664180752
CPI: 0.61

real    0m3.279s
user    0m3.266s
sys     0m0.007s
```

- 70000195612 / 17500058470 = 3.99999
- SSE: 128 bits = 16 octets = 4 réels simple précision
- CPI = 10664180752/(17500058470*4)=0.15

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Compter les FLOPS

- Troisième programme: compter les MFLOPS !
  - `PAPI_FP_OPS`
  - `PAPI_TOT_CYC`
  - SAXPY: `y(i) = a * x(i) + y(i)`
- Vérifier que le nombre d'opérations flottantes est cohérent.

Interface C

```
long_long PAPI_get_real_usec(void);
```

Interface Fortran

```
PAPIF_get_real_usec(C_LONG_LONG)
```


Auteur: Laurent Gatineau

Compter les FLOPS

```
int events[NUM_EVENTS] = {PAPI_FP_OPS,
                          PAPI_TOT_CYC};
long_long start_usec, end_usec;
double elapse;


retval = PAPI_start_counters(events,
         NUM_EVENTS);
start_usec = PAPI_get_real_usec();

[…]

end_usec = PAPI_get_real_usec();
retval = PAPI_stop_counters(values,
         NUM_EVENTS);


elapse = ((double) (end_usec - start_usec))
         / 1000000.0;
printf("Wall clock-time = %.2fs\n", elapse);
printf("MFLOPS = %8.2f\n",
  ((double) values[0]) / (elapse * 1000000));
```

```
INTEGER(C_INT), DIMENSION(NUM_EVENTS) :: &
       events = (/ PAPI_FP_OPS, PAPI_TOT_CYC/)
INTEGER(C_LONG_LONG) :: start_usec, end_usec
REAL(KIND=8) :: elapse


CALL PAPIF_start_counters(events, &
     NUM_EVENTS, retval)
CALL PAPIF_get_real_usec(start_usec)

[…]

CALL PAPIF_get_real_usec(end_usec)
CALL PAPIF_stop_counters(values, &
     NUM_EVENTS, retval)

elapse = REAL(end_usec - start_usec, 8) &
       / 1000000.0
WRITE(6, '(A,F5.2,A)') "Wall clock-time = ",&
       elapse, "s"
WRITE(6, '(A,F8.2)') "MFLOPS = ",&
       REAL(values(1), 8) / (elapse * 1000000)
```

Sources dans: /home/gatineaul/TP_PAPI/TP3

Compter les FLOPS

- Essayez en compilant avec
  `-ftree-vectorize -msse`
- Que se passe-t-il ?

- Testez:
  - `PAPI_FP_OPS`
  - `PAPI_VEC_SP / PAPI_SP_OPS`
  - `PAPI_VEC_DP / PAPI_DP_OPS`

- Différence entre l'Ice et l'IceX ?

- Différence entre GCC et Intel avec AVX sur l'IceX ?

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Défaut de cache et TLB

- Défaut de cache: la donnée est accédée depuis la mémoire ou un niveau de cache supérieur.

- Défaut de TLB (Translation Lookaside Buffer): la donnée n'est pas dans une page mémoire physique connue du système, le système doit faire une conversion pour trouver la page mémoire.

- Evènements PAPI:
  - `PAPI_L1_DCM / PAPI_L1_TCM`
  - `PAPI_L2_DCM / PAPI_L2_TCM`
  - `PAPI_L3_DCM / PAPI_L3_TCM`
  - `PAPI_TLB_DM`

Auteur: Laurent Gatineau

Défaut de cache et TLB

- Connaître les limites / caractéristiques du cache:
  - Taille et nombre de niveau de cache.
  - Type de cache (Instructions, Données, les deux).
  - Partage entre cœurs / processeurs.
  - Ligne de cache, associativité.

```
$ cat /sys/devices/system/cpu/cpu0/cache/index*/type | xargs echo
Data Instruction Unified Unified
$ cat /sys/devices/system/cpu/cpu0/cache/index{0,2,3}/level | xargs echo
1 2 3
$ cat /sys/devices/system/cpu/cpu0/cache/index{0,2,3}/size | xargs echo
32K 256K 20480K
$ cat /sys/devices/system/cpu/cpu0/cache/index{0,2,3}/coherency_line_size | xargs echo
64 64 64
$ cat /sys/devices/system/cpu/cpu0/cache/index{0,2,3}/ways_of_associativity | xargs echo
8 8 20
$ cat /sys/devices/system/cpu/cpu0/cache/index{0,2,3}/number_of_sets | xargs echo
64 512 16384
$
```

Défaut de cache et TLB

- Quatrième programme: compter les défauts de cache !
  - `PAPI_L1_TCM / PAPI_L2_TCM / PAPI_L3_TCM`
  - Copier un tableau 2D dans un autre…

Non optimal (C)	Optimal (C)

`for (j = 0; j < N; j++)`
`  for (i = 0; i < N; i++)`
`  dst[i][j] = src[i][j]`
	`for (i = 0; i < N; i++)`
	`  for (j = 0; j < N; j++)`
	`  dst[i][j] = src[i][j]`

  - Choix de N pour sortir du cache à chaque itération L2:
    `N = (taille du cache) / (ligne de cache) * (taille float) * (nb tableaux)`

- Ratio entre les défauts de cache (L2/L3) et le nombre d'instructions (`PAPI_TOT_INS`), le nombre de cycles (`PAPI_TOT_CYC`) ou le nombre de load (`PAPI_LD_INS`)

- Comparer entre l'Ice et l'IceX.

Auteur: Laurent Gatineau

Défaut de cache et TLB

```
$ time ./tp4_c
11 available hardware counter(s).
PAPI_L1_TCM: 33379693
PAPI_L2_TCM: 33336030
PAPI_L3_TCM: 32280748
PAPI_TOT_INS: 117480198

real    0m0.474s
user    0m0.433s
sys     0m0.034s
$ time ./tp4_opt_c
11 available hardware counter(s).
PAPI_L1_TCM: 2084187
PAPI_L2_TCM: 1216145
PAPI_L3_TCM: 159159
PAPI_TOT_INS: 83913037

real    0m0.064s
user    0m0.035s
sys     0m0.027s
[lgatineau@sabi62 TP4]$
```

Cache	Non optimal	Optimal
L1	28.41%	2.48%
L2	28.38%	1.45%
L3	27.48%	0.19%

- Différence PAPI_TOT_INS: Calcul d'indices !
- Gain: x 7.4 !

Sources dans: /home/gatineaul/TP_PAPI/TP4

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Introduction au multithread

- La librairie PAPI supporte les applications multithread (OpenMP, pthread, …).
- Depuis PAPI 3, l'API de haut niveau est *thread safe*:
  - Nécessite une initialisation particulière.
  - Appels des fonctions `PAPI_start_counters,…` pour chaque thread.
- Les évènements sont propres à chaque thread.
- Chaque thread peut compter des évènements différents, mais dans la limite du nombre de compteurs supportés.

Auteur: Laurent Gatineau

Introduction au multithread

- Cinquième programme: un peu d'OpenMP !
- Initialisation via le thread principal uniquement:
  - Initialisation de PAPI via `PAPI_library_init` ou via l'appel d'une fonction de haut niveau.
    - Prend en argument la version de PAPI (`PAPI_VER_CURRENT`)
  - Appel de la fonction `PAPI_thread_init`.
    - Prend en argument une fonction qui renvoie un identifiant unique par thread (`omp_get_thread_num`, `pthread_self`).

Interface C

```
int PAPI_library_init(int version);
int PAPI_thread_init(unsigned long int (* handle )());
```

Interface Fortran

```
PAPIF_library_init(C_INT check)
PAPIF_thread_init(C_INT FUNCTION handle, C_INT check)
```


Auteur: Laurent Gatineau

Introduction au multithread

```
retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT) {
    fprintf(stderr, "This program was "
        "compiled with PAPI %d.%d\n",
        PAPI_VERSION_MAJOR(PAPI_VER_CURRENT),
        PAPI_VERSION_MINOR(PAPI_VER_CURRENT));
    return 1;
}

retval = PAPI_thread_init(pthread_self);
if (retval != PAPI_OK) {
    perror("PAPI_thread_init");
    return 1;
}

#pragma omp parallel private(retval)
{
    retval = PAPI_start_counters(events,
             NUM_EVENTS);
    if (retval != PAPI_OK) {
        PAPI_perror("PAPI_start_counters");
        exit(EXIT_FAILURE);
    }
}
```

```
retval = PAPI_VER_CURRENT
CALL PAPIF_library_init(retval)
IF (retval.NE.PAPI_VER_CURRENT) THEN
  WRITE(6,*) "This program was compiled for "&
             "another PAPI version"
  STOP
END IF

CALL PAPIF_thread_init(omp_get_thread_num,
     retval);
IF (retval.lt.PAPI_OK) THEN
  CALL PAPIF_perror('PAPIF_thread_init')
  STOP
END IF

!$OMP PARALLEL PRIVATE(retval)
  CALL PAPIF_start_counters(events,
       NUM_EVENTS, retval)
  IF (retval.ne.PAPI_OK) THEN
    CALL PAPIF_perror('PAPIF_start_counters')
    STOP
  END IF
!$OMP END PARALLEL
```

Sources dans: /home/gatineaul/TP_PAPI/TP5

Auteur: Laurent Gatineau

Plan

- Qu’est-ce que les compteurs hardware ?
- PAPI: The Performance API
- Nombre de cycles et nombre d’instructions
- Compter les FLOPS
- Défaut de cache et TLB
- Introduction au multithread
- Utiliser les évènements natifs

Utiliser les évènements natifs

- La librairie PAPI dispose d'évènements prédéfinis (`PAPI_???`) générique à toutes les architectures.

- Il est possible d'utiliser les évènements natifs.

```
gatineaul@service0:~> module load papi
gatineaul@service0:~> papi_native_avail
[…]
--------------------------------------------------------------------------------
| MEM_UNCORE_RETIRED                                                           |
|            Load instructions retired (Precise Event)                         |
|     :LOCAL_HITM                                                              |
|            Load instructions retired that HIT modified data in sibling core (|
|            Precise Event)                                                    |
|     :LOCAL_DRAM_AND_REMOTE_CACHE_HIT                                         |
|            Load instructions retired local dram and remote cache HIT data sou|
|            rces (Precise Event)                                              |
[…]
```


Auteur: Laurent Gatineau

Utiliser les évènements natifs

- Dans la documentation Intel l'évènement `MEM_UNCORE_RETIRED.OTHER_CORE_L2_HITM` est intéressant: *Load instructions retired that HIT modified data in sibling core (Precise Event)*.
  - `MEM_UNCORE_RETIRED.LOCAL_HITM` dans PAPI !!
- Permet de connaître le nombre d'accès mémoire qui demande un accès au cache L2 d'un autre cœur.
  Détecter les cas de *False Sharing*.
  Ratio cache miss / cache hit >= 1

- `EXT_SNOOP.ALL_AGENTS.HITM` sur Intel Harpertown.
- `MEM_UNCORE_RETIRED.LOCAL_HITM` sur Intel Nehalem&Westmere (Ice).
- Sur SandyBridge / IvyBridge: ???

Utiliser les évènements natifs

Principe du false sharing:

- Plusieurs cœurs accèdent à des données différentes mais contigües en mémoire, provoquant l'invalidation de lignes de cache.
- Exemple: `array[num_thread] = x;`

Notes:

- `EventName`: Nom de l'évènement au format PAPI, ex: `MEM_UNCORE_RETIRED:LOCAL_HITM`
- `EventCode`: code de l'évènement (valeur de retour).

Interface C

```
int PAPI_event_name_to_code(char *EventName, int *EventCode);
```

Interface Fortran

```
PAPIF_event_name_to_code(C_STRING EventName, C_INT EventCode, C_INT check)
```


Auteur: Laurent Gatineau

Utiliser les évènements natifs

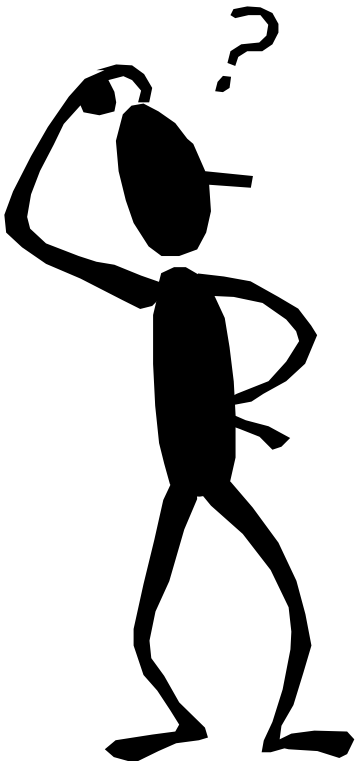
```
 float x[N], y[N], sum;
 float sum_local[MAX_THREADS];

 retval = PAPI_event_name_to_code
("MEM_UNCORE_RETIRED:LOCAL_HITM", &events[2]);
 if (retval != PAPI_OK) {
   perror("PAPI_event_name_to_code");
   return 1;
 }

#pragma omp parallel
{
  int my_thread_num = omp_get_thread_num();

  sum_local[my_thread_num] = 0.0;

#pragma omp for private(i)
  for (j = 0; j < N_ITER; j++) {
    for (i = 0; i < N; i++) {
      sum_local[my_thread_num] += x[i] * y[i];
    }
  }

#pragma omp atomic
  sum += sum_local[my_thread_num];
}
```

```
REAL, DIMENSION(N) :: x, y
REAL :: sum
REAL, DIMENSION(MAX_THREADS) :: sum_local

CALL PAPIF_event_name_to_code( &
 'MEM_UNCORE_RETIRED:LOCAL_HITM', events(3), &
  retval)
IF (retval.ne.PAPI_OK) THEN
  CALL PAPIF_perror('…')
  STOP
END IF

!$OMP PARALLEL PRIVATE(my_thread_num, nn)
  my_thread_num = omp_get_thread_num() + 1
  sum_local(my_thread_num) = 0.0
!$OMP DO private(i)
  DO j = 1, N_ITER
    DO i = 1, N
      sum_local(my_thread_num) = &
        sum_local(my_thread_num) &
        + x(i) * y(i);
    END DO
  END DO
!$OMP END DO
!$OMP ATOMIC
    sum = sum + sum_local(my_thread_num);
!$OMP END PARALLEL
```

Sources dans: /home/gatineaul/TP_PAPI/TP6

Auteur: Laurent Gatineau

QUESTIONS ?



Auteur: Laurent Gatineau