# CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

## Aula 10: Interface PAPI

**Celso L. Mendes, Stephan Stephany**

**LAC / INPE**

**Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br**

# PAPI: Performance API

- Objetivos:
  - Padronizar o acesso aos contadores de qualquer processador
  - Padronizar os tipos de eventos disponíveis, mantendo

  acesso a eventos específicos de cada processador se desejável
- Histórico:
  - Desenvolvido na Univ. Tennessee (grupo de J.Dongarra)
  - Mantido e atualizado por quase duas décadas
  - Suportado em máquinas de quase todos os fabricantes
  - Versões atuais premitem acesso a outros tipos de contadores
    - Ex: GPU, Rede de interconexão, Sistema de armazenamento, etc.

# PAPI - Documentação

- Locais com documentação:
  - Website geral: http://icl.cs.utk.edu/papi/
  - ➢ APIs: http://icl.cs.utk.edu/projects/papi/wiki/Counter_Interfaces
  - Funções das APIs e utilitários: http://icl.cs.utk.edu/papi/docs/
- Versões disponíveis:
  - v. 5.5.0: Suporte para medição de energia; procs. Power8, KNL
  - v. 5.4.3: Instalada no Santos Dumont
  - Código-fonte: http://icl.cs.utk.edu/papi/software/index.html

# PAPI - Interfaces Disponíveis

- Duas interfaces distintas: Alto Nível × Baixo Nível
- Interface de Alto Nível:
  - Funções básicas para configurar e acessar os contadores
  - Poucas funções (8), simples de aprender e de programar
  - Acesso a eventos pré-definidos (Ex: PAPI_TOT_INS, PAPI_... )
    - Cerca de 100 eventos mais comuns a vários processadores
    - Contudo, semântica pode mudar de um processador para outro!
  - Funcionalidade limitada, porém faz o essencial

# PAPI - Interfaces Disponíveis (cont.)

- Interface de Baixo Nível:
  - Oferece muito maior controle ao programador
  - Ampla variedade de funcionalidades disponíveis
  - Voltada para suportar ferramentas de desempenho
  - Permite acesso a qualquer evento *nativo*
    - Eventos específicos de um certo processador
  - Número bem mais alto de funções (~50), maior eficiência
  - Utilizada pelas funções de alto nível; uso conjunto permitido

# PAPI – Interface de Alto Nível

- Verificação do número de contadores existentes
  - *int PAPI_num_counters (void)*
- Escolha e ativação dos contadores
  - *int PAPI_start_counters (int *events, int array_len)*
    - Array *events* contém os eventos escolhidos para contagem
- Leitura dos contadores: vários modos possíveis
  - *int PAPI_read_counters(long long *values, int array_len)*
    - Copia valores atuais para o array *values* e zera (resseta) contadores
  - *int PAPI_accum_counters(long long *values, int array_len)*
    - Soma valores atuais ao array *values* e zera (resseta) contadores
  - *int PAPI_stop_counters(long long *values, int array_len)*
    - Copia valores atuais para o array *values* e paraliza os contadores

# Uso de PAPI no Santos Dumont

- Acesso aos utilitários, manpages e libraries de PAPI

  *module load bullxde*

  *module load papi*

- Construção de programas em C (similar para Fortran)

  *icc -o prog prog.c -I/opt/bullxde/perftools/papi-bull/5.4.3.0/include*
  *-L/opt/bullxde/perftools/papi-bull/5.4.3.0/lib64 -lpapi –lm*

  - Para programas em Fortran: usar *ifort* ao invés de *icc*
  - Para programas com MPI, supondo Intel-MPI:
    - C: usar *mpiicc* ao invés de *icc*
    - Fortran: usar *mpiifort* ao invés de *ifort*

# PAPI – Exemplo de Uso

```
#include <papi.h>
#include <stdio.h>
#include <math.h>
#define NUM_EVENTS 3
#define TIMES 1000000
```

```
main() {
  int i, Events[NUM_EVENTS] = {PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_FP_INS};
  long_long avalues[NUM_EVENTS],bvalues[NUM_EVENTS];
  double x=0.5;
  int nc = PAPI_num_counters();
  printf("Num.Counters=%d\n",nc);
  printf("PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_FP_INS\n");
/* Start counting events */
  if (PAPI_start_counters(Events, NUM_EVENTS) != PAPI_OK) exit(-1);
  for (i=0; i<TIMES/4; i++) x=sin(x);
/* Read the counters */
  if (PAPI_read_counters(avalues, NUM_EVENTS) != PAPI_OK) exit(-1);
  for (i=0; i<TIMES; i++) x=sin(x);
/* Read the counters again */
  if (PAPI_read_counters(bvalues, NUM_EVENTS) != PAPI_OK) exit(-1);
/* Print counters */
  printf("Apos primeira leitura dos counters: %lld, %lld, %lld, x=%lf\n",
          avalues[0],avalues[1],avalues[2],x);
  printf("Apos segunda leitura dos counters: %lld, %lld, %lld, x=%lf\n",
          bvalues[0],bvalues[1],bvalues[2],x);
  printf("All done\n");
}
```

Trabalho = O(TIMES/4)

Trabalho = O(TIMES)

# PAPI – Exemplo de Uso

- Execução no Santos Dumont:

**Num.Counters=11**

**PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_FP_INS**

**Apos primeira leitura dos counters: 14293918, 21254316, 7250097, x=0.001549**

**Apos segunda leitura dos counters: 57138777, 85000634, 29000187, x=0.001549**

**All done**

- Processador: */proc/cpuinfo*

model name : Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz

( Ivy Bridge )

# PAPI – Utilitários Principais

- papi_avail:
  - Revela quais eventos estão disponíveis no processador
  - Alguns eventos são derivados de outros eventos

Exemplo-1 - Execução no Santos Dumont:

| 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 – Utilitários Principais (cont.)

- papi_avail:

Exemplo-2 – Execução no Tupã/CPTEC:

```
PAPI Version                  : 4.1.0.0
Vendor string and code        : AuthenticAMD (2)
Model string and code         : AMD Opteron(tm) Processor 6172 (9)
Cores per Socket              : 12
NUMA Nodes                    : 4
CPU's per Node                : 6
Total CPU's                   : 24
Number Hardware Counters : 4

 Name            Code          Avail Deriv Description (Note)
PAPI_L1_DCM   0x80000000   Yes   No   Level 1 data cache misses
PAPI_L1_ICM   0x8000000 1  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002   Yes   No   Level 2 data cache misses
PAPI_L2_ICM  0x80000003   Yes   No   Level 2 instruction cache misses
. . .
```

# PAPI – Utilitários Principais (cont.)

- papi_command_line:
  - Revela se uma certa escolha de eventos é valida
  - Formato: *papi_command_line <ev-1> <ev-2> … <ev-N>*
    - Caso haja algum retorno nulo, a escolha não é válida

Exemplos (Santos Dumont):

*papi_command_line PAPI_FP_INS PAPI_LD_INS PAPI_SR_INS*
PAPI_FP_INS : 40607641
PAPI_LD_INS : 120067042
PAPI_SR_INS : 0

*papi_command_line PAPI_LD_INS PAPI_SR_INS PAPI_FP_INS*
PAPI_LD_INS : 120067711
PAPI_SR_INS : 40065631
PAPI_FP_INS : 43251183

# PAPI – Utilitários Principais (cont.)

- papi_cost: Avalia o custo das funções mais comuns

Exemplo: execução no Santos Dumont

*Performing start/stop test...*
*Total cost for PAPI_start/stop (2 counters) over 1000000 iterations*
*min cycles : 16312*
*max cycles : 91420*
*mean cycles : 17067.317592*
*std deviation: 542.593340*
*Performing read test...*
*Total cost for PAPI_read (2 counters) over 1000000 iterations*
*min cycles : 8172*
*max cycles : 65324*
*mean cycles : 8517.125784*
*std deviation: 348.123895*

# PAPI – Outros Comentários

- Contagens podem ser inexatas
  - Instruções efetivamente executadas podem cobrir execução especulativa, funções do Sistema Operacional, etc.
- Contadores podem ser incompatíveis entre si
  - Solução-1: múltiplas execuções
  - Solução-2: contadores são multiplexados (resultados imprecisos)
- Eventos específicos de alguns processadores
  - Podem ser utilizado apenas com a API de baixo nível
  - Necessário estudar em detalhe a documentação do processador
- Uso em programas com *Threads*
  - Permitido apenas com a API de baixo nível
  - Cada thread deve inicializar e terminar o acesso aos contadores

# PAPI – Uso em Aplicação Real

- Código: BRAMS, excutado com AMPI num Cray XT5 (ORNL)

  Várias execuções com P=64 e diferentes níveis de virtualização AMPI

  Ref: Rodrigues, E.R. *et al* – HiPC, 2010

| Nível de Virtualização | Tempo de Execução |
|---|---|
| 64 | 3363.4 s |
| 256 | 2998.8 s |
| 1024 | 2616.4 s |
| 2048 | 2677.7 s |

| Nível Virt. | L2 cache misses | L3 cache misses |
|---|---|---|
| 64 | 12.416 M | 8.448 M |
| 256 | 10.560 M | 4.416 M |
| 1024 | 9.408 M | 3.904 M |
| 2048 | 13.696 M | 5.056 M |