Analise da eficiência de cache na Multiplicação de Matrizes

Programa MM.c alterado

```
#include <stdio.h>
#include <stdlib.h>
#include <papi.h>
#define ind2d(i,j) i*tam+j

double wall_time(void);

// reportError: imprime o codigo de erro do PAPI e aborta
//              a computacao


void reportError(int code, char *name){
  printf("**(ERR)** PAPI returned error %d at %s\n", PAPI_strerror(code), name);
  exit(-1);
}


void MMijk(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (i=0; i<tam; i++)
    for (j=0; j<tam; j++)
      for (k=0; k<tam; k++)
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}


void MMikj(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (i=0; i<tam; i++)
    for (k=0; k<tam; k++)
      for (j=0; j<tam; j++)
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}

void MMkij(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (k=0; k<tam; k++)
    for (i=0; i<tam; i++)
      for (j=0; j<tam; j++)
```

```
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}

void MMkji(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (k=0; k<tam; k++)
    for (j=0; j<tam; j++)
      for (i=0; i<tam; i++)
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}

void MMjik(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (j=0; j<tam; j++)
    for (i=0; i<tam; i++)
      for (k=0; k<tam; k++)
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}

void MMjki(int tam, float* matA, float* matB, float* matC) {
  int i, j, k;
  for (j=0; j<tam; j++)
    for (k=0; k<tam; k++)
      for (i=0; i<tam; i++)
    matC[ind2d(i,j)] = matC[ind2d(i,j)] + matA[ind2d(i,k)]*matB[ind2d(k,j)];
}


int main(int argc, char *argv[]) {
#define NUM_EVENTS 3
  int EventCode[NUM_EVENTS] = {PAPI_LD_INS, PAPI_L1_DCM, PAPI_L2_DCM};
  char EventName[NUM_EVENTS][PAPI_MAX_STR_LEN];
  long long EventCount[NUM_EVENTS];
  int retValue;

  int i;
  int tam;
  float *matA, *matB, *matC;
  double t0, t1;

  if (argc != 2) {
    printf(" uso: <exec> <tamanho das matrizes> \n");
    exit(-1);
  }
  tam = atoi(argv[1]);
```

```
// inicializa PAPI e seus contadores

if ((retValue = PAPI_start_counters(EventCode, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "PAPI_start_counters");

// impime quantos contadores disponiveis e quais eventos solicitados

printf("This processor has %d counters\n", PAPI_num_counters());
printf("Selected PAPI events:");
for (i=0; i<NUM_EVENTS; i++) {
  if ((retValue=PAPI_event_code_to_name(EventCode[i], EventName[i])) != PAPI_OK)
    reportError(retValue, "PAPI_event_code_to_name");
  printf(" %s;", EventName[i]);
}
printf("\n");

// aloca e inicializa matrizes

matA  = (float *) malloc (tam*tam*sizeof(float));
matB  = (float *) malloc (tam*tam*sizeof(float));
matC  = (float *) malloc (tam*tam*sizeof(float));
for (i=0; i<tam*tam; i++) {
  matA[i]=1.0; matB[i]=1.0;
}

printf("tam=%d; tempos: \n", tam);

for (i=0; i<tam*tam; i++) matC[i]=0.0;
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMijk(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("ijk=%f \n", t1);
// fim do trecho a medir; le e reporta contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

for (i=0; i<tam*tam; i++) matC[i]=0.0;
```

```
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMikj(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("ikj=%f \n", t1);
// fim do trecho a medir; le e reporta contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

for (i=0; i<tam*tam; i++) matC[i]=0.0;
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMkij(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("kij=%f \n", t1);
// fim do trecho a medir; le e reporta contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

for (i=0; i<tam*tam; i++) matC[i]=0.0;
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMkji(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("kji=%f \n", t1);
// fim do trecho a medir; le e reporta contadores
```

```
if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

for (i=0; i<tam*tam; i++) matC[i]=0.0;
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMjik(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("jik=%f \n", t1);
// fim do trecho a medir; le e reporta contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

for (i=0; i<tam*tam; i++) matC[i]=0.0;
// inicio do trecho a medir; le e zera contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "first PAPI_read_counters");
t0 = wall_time();
MMjki(tam, matA, matB, matC);
t1 = wall_time()-t0;
printf("jki=%f \n", t1);
// fim do trecho a medir; le e reporta contadores

if ((retValue = PAPI_read_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
  reportError(retValue, "second PAPI_read_counters");
for (i=0; i<NUM_EVENTS; i++) {
  printf(" %s=%lld;", EventName[i], EventCount[i]);
}
printf("\n");

// para os contadores

if ((retValue = PAPI_stop_counters(EventCount, NUM_EVENTS)) != PAPI_OK)
```

```
    reportError(retValue, "PAPI_stop_counters");

  free(matA);
  free(matB);
  free(matC);
  exit(0);
}
```

Resultados

Multiplicação	Tempo	L1 Cache Miss %	L2 Cache Miss %
ijk	15.17	0.0458600576468	0.9848822845479
ikj	6.526	0.0028509738170	0.0471878040445
kij	6.523	0.0029461632761	0.0907508258928
kji	20.57	0.0914619106324	0.9943810082548
jik	15.18	0.0487397815127	0.9864749536038
jki	20.58	0.0915510690868	0.9942817499171

Análise

Observando os resultados obtidos, percebe-se a diferença que o bom planejamento da arquitetura de memória de um programa pode causar, com um aumento no tempo de mais de três vezes do programa mais eficiente até o menos eficiente.

Percebe-se tambem que existe correlação entre os acertos na cache L1 e na cache L2, aumentando ainda mais a eficiencia dos programas que utilizam a cache de forma planejada.