Trabalho de Arquitetura de Computadores I

Comparativo de desempenho de multiplicação de matrizes

Aluno: Raul Sena Ferreira
Matrícula: 2010780254

Introdução

Este pequeno trabalho fala sobre desempenho em multiplicação de matrizes de 2 dimensões.

Foram analisados em cada matriz:

O tempo de execução, número de cache misses (L1 e L2) e o total de ciclos.
O programa foi desenvolvido na linguagem C e faz uso da biblioteca PAPI.
Foi necessário rodar o programa dentro da pasta “meu_diretorio/papi-5.1.1/src/examples” para funcionar.

A matriz é quadrada (2x2) de tamanho 800x800, o programa foi executado 5 vezes consecutivas e os valores variaram muito pouco e mantiveram praticamente a mesma proporção para os resultados apresentados abaixo.

Os cálculos foram feitos em uma máquina com processador Intel Core i3 com 4GB de RAM.

Resultados

O seguinte rank(do mais rápido ao mais lento) no que diz respeito a performance na execução é:

Ordem dos Índices	Cache Misses (L1)	Cache Misses (L2)	Total de Ciclos	Tempo Total(seg)
IKJ	103728	11188	7338665184	3.240000
KIJ	31789787	1300507	7403013583	3.280000
JIK	1144698	122689	8797769470	3.890000
IJK	50342	8550	8824967812	3.920000
JKI	542712257	961031	18328789932	8.110000
KJI	554239169	33367624	19490865483	8.620000

Com os resultados podemos perceber que utilizando-se os índices IKJ:

- Tornamos a execução da multiplicação de matrizes mais rápida do que a tradicional ordem de índices (IJK);
- A ordem de índice IKJ teve a segunda melhor média de cache misses, tanto em L1 quanto em L2, ou seja, a quantidade de vezes que o processador precisou de um dado e não o achou no cache;
- Gastou menos ciclos do que as demais para realizar a tarefa.

Código do programa

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "papi.h"

#define NUM_EVENTS 3
```

```
#define CONST 800
#define ERROR_RETURN(retval) { fprintf(stderr, "Error %d %s:line %d: \n",
retval,__FILE__,__LINE__);  exit(retval); }

int Events[3] = {PAPI_L1_TCM, PAPI_L2_TCM, PAPI_TOT_CYC};
int retval;
char errstring[PAPI_MAX_STR_LEN];
long long values[NUM_EVENTS];

void mult_matriz_ijk();
void mult_matriz_jik();
void mult_matriz_ikj();
void mult_matriz_jki();
void mult_matriz_kij();
void mult_matriz_kji();

int main(){
  //system("cls");//limpa terinal no windows
  system("clear");//limpa terminal no unix

  printf("\nMatriz ijk\n");
  mult_matriz_ijk();
  printf("\n\n\nMatriz jik\n");
  mult_matriz_jik();
  printf("\n\n\nMatriz ikj\n");
  mult_matriz_ikj();
  printf("\n\n\nMatriz jki\n");
  mult_matriz_jki();
  printf("\n\n\nMatriz kij\n");
  mult_matriz_kij();
  printf("\n\n\nMatriz kji\n");
  mult_matriz_kji();

  exit(0);
}

void mult_matriz_ijk(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST],inicio, fim;


  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);
```

```
  inicio = clock();

  for(i=0;i<CONST;i++){
    for(j=0;j<CONST;j++){
      for(k=0;k<CONST;k++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1): %lld \n", values[0] );
  printf("Total de cache misses (L2): %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}

void mult_matriz_jik(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST], inicio, fim;

  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  inicio = clock();

  for(j=0;j<CONST;j++){
    for(i=0;i<CONST;i++){
      for(k=0;k<CONST;k++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1) %lld \n", values[0] );
```

```
  printf("Total de cache misses (L2) %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}

void mult_matriz_ikj(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST], inicio, fim;

  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  inicio = clock();

  for(i=0;i<CONST;i++){
    for(k=0;k<CONST;k++){
      for(j=0;j<CONST;j++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1): %lld \n", values[0] );
  printf("Total de cache misses (L2): %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}

void mult_matriz_jki(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST], inicio, fim;

  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);
```

```
  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  inicio = clock();

  for(j=0;j<CONST;j++){
    for(k=0;k<CONST;k++){
      for(i=0;i<CONST;i++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1): %lld \n", values[0] );
  printf("Total de cache misses (L2): %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}

void mult_matriz_kij(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST], inicio, fim;

  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  inicio = clock();

  for(k=0;k<CONST;k++){
    for(i=0;i<CONST;i++){
      for(j=0;j<CONST;j++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
```

```
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1): %lld \n", values[0] );
  printf("Total de cache misses (L2): %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}

void mult_matriz_kji(){
  int i, j, k;
  float m[CONST][CONST], m1[CONST][CONST], maux[CONST][CONST], inicio, fim;

  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ){
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  inicio = clock();

  for(k=0;k<CONST;k++){
    for(j=0;j<CONST;j++){
      for(i=0;i<CONST;i++){
        maux[i][j] = maux[i][j] + (m[i][k] * m1[k][j]);
      }
    }
  }
  fim = clock();

  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("Total de cache misses (L1): %lld \n", values[0] );
  printf("Total de cache misses (L2): %lld \n", values[1] );
  printf("Total de ciclos: %lld \n", values[2] );
  printf("Total de tempo gasto: %f \n", (fim-inicio) / (float)CLOCKS_PER_SEC );
}
```

Bibliografia

http://icl.cs.utk.edu/papi/
https://en.wikipedia.org/wiki/CPU_cache