Grupo 4 - Tiago Koji Castro Shibata (8988730)

Sistemas de Computação de Alto Desempenho

Aula 4

Relatório: OpenMP

- **Introdução**

Em aula, estudamos o uso de OpenMP para paralelismo. No laboratório, implementamos dois programas paralelos simples operando em matrizes 1000x1000.

- **Configuração do sistema**

Testes foram realizados em um processador Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, com 4 núcleos. O sistema utilizado é Linux e os processos foram executados com *niceness* mínimo para que outros processos do sistema interferissem menos nos resultados.

Todos os programas foram compilados com *`-O3 -DNDEBUG -Wall -Wextra -pedantic`*.

- **Programas**

No **exercício 1**, iniciamos uma matriz 1000x1000 utilizando a fórmula *`A[i][j] = (3 * i - j) / 2`* e realizamos a soma de seus elementos. Tanto a inicialização quanto a soma foram implementados em paralelo com OpenMP. O laço *for* externo foi paralelizado (ou seja, houve paralelismo entre as linhas da matriz).

Primeiramente implementei o programa utilizando dois laços paralelos:

```c
#include <stdio.h>
#include <sys/sysinfo.h>
#include <time.h>

#include <omp.h>

#define SIZE 1000
double A[SIZE][SIZE];

int main() {
    int nprocs = get_nprocs();
    printf("nprocs = %d\n", nprocs);
    omp_set_num_threads(nprocs);

    struct timespec start_time, end_time;
    clock_gettime(CLOCK_MONOTONIC, &start_time);
    #pragma omp parallel for
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = (3 * i - j) / 2;
```

```
        }
    }

    double sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            sum += A[i][j];
        }
    }
    clock_gettime(CLOCK_MONOTONIC, &end_time);
    printf("Sum: %f, time: %.5fms\n", sum,
        (end_time.tv_sec * 1000 + 1e-6 * end_time.tv_nsec) -
(start_time.tv_sec * 1000 + 1e-6 * start_time.tv_nsec));

    return 0;
}
```

Após 40 execuções, o melhor tempo registrado foi 1.71280ms.

Uma otimização possível é criar apenas uma região paralela, e criar dois loops com *#pragma omp for*. Dessa maneira, thread serão criadas apenas uma vez, e reutilizadas entre os laços. Como visto em http://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-sync.html#Implicitbarriers, por padrão o OpenMP insere uma barreira implícita entre *worksharing constructs* (como laços for), portanto não é necessário que adicionemos uma barreira entre os laços.

O laço foi modificado para:

```
double sum = 0;
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = (3 * i - j) / 2;
        }
    }

    #pragma omp for reduction(+:sum)
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            sum += A[i][j];
        }
    }
}
```

Após 40 execuções, o menor tempo registrado foi 1.67801ms. O ganho foi muito pequeno nesse exemplo, mas pode ser maior em sistemas com maior número de threads.

Implementei também uma versão serial do programa. Seus laços são:

```
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        A[i][j] = (3 * i - j) / 2;
    }
}

double sum = 0;
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        sum += A[i][j];
    }
}
```

Após 40 execuções, o menor tempo registrado foi 6.58002ms.

Serial	2 regiões paralelas	1 região paralela
6.58002ms	1.71280ms	1.67801ms

A título de curiosidade, para demonstrar a importância de pensar no cache ao escrever programas, mudei a versão de 1 região paralela, invertendo a ordem de iteração (variáveis *i* e *j*). Dessa maneira, os acessos das threads não serão sequenciais, pulando 4 * 1000 = 4KB por iteração do loop, e afetando muito negativamente o comportamento do cache. Após 40 execuções, o melhor tempo desse programa com acesso irregular da memória foi 8.97835ms (~5.3x mais lento). Podemos ver a importância de dividir o trabalho nas threads em linhas (já que em C as matrizes são *row-major order*).

No **exercício 2**, implementei busca de um elemento em matriz. A matriz foi iniciada pela fórmula A[i][j] = (2 * i - j) % 1000. Em seguida, um elemento foi lido do teclado e suas ocorrências na matriz contadas, com paralelismo entre linhas:

```
#include <stdio.h>
#include <sys/sysinfo.h>
#include <time.h>
```

```c
#include <omp.h>

#define SIZE 1000
double A[SIZE][SIZE];

int main() {
    int nprocs = get_nprocs();
    printf("nprocs = %d\n", nprocs);
    omp_set_num_threads(nprocs);

    #pragma omp parallel for
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = (2 * i - j) % 1000;
        }
    }

    for (;;) {
        float target;
        printf("Type target element:\n");
        scanf("%f", &target);

        struct timespec start_time, end_time;
        clock_gettime(CLOCK_MONOTONIC, &start_time);
        double occurences = 0;
        #pragma omp parallel for reduction(+:occurences)
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                occurences += A[i][j] == target;
            }
        }
        clock_gettime(CLOCK_MONOTONIC, &end_time);
        printf("Occurences: %f, time: %.5fms\n", occurences,
            (end_time.tv_sec * 1000 + 1e-6 * end_time.tv_nsec) -
(start_time.tv_sec * 1000 + 1e-6 * start_time.tv_nsec));
    }

    return 0;
}
```

Após 40 buscas, o menor tempo registrado foi 0.78713ms. Implementei também uma versão serial do programa, cujo laço principal é:

```c
double occurences = 0;
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        occurences += A[i][j] == target;
    }
}
```

Após 40 buscas, o menor tempo registrado foi 1.52199ms.

Serial	Paralelo
1.52199ms	0.78713ms