Universidade de Brasília

Programação Paralela

Relatório do Exercício 1

Autor:
Alexandre Lucchesi Alencar
09/0104471

Professor:
George Luiz Medeiros Teodoro

11 de setembro de 2014

1 Introdução

Este relatório tem como objetivo apresentar os resultados obtidos a partir da execução do primeiro exercício de programação paralela, que consiste na paralelização do algoritmo *count sort* utilizando a biblioteca OpenMP. Primeiramente, o algoritmo desenvolvido e a estratégia de paralelização utilizada é apresentada. Em seguida, é feita uma análise de desempenho comparando os tempos de execução do algoritmo em diversas configurações, isto é, variando-se a estratégia de escalonamento, o tamanho dos *chunks* e o número de *threads*. O código completo deste trabalho está publicamente disponível no GitHub [1].

2 O Algoritmo

O programa desenvolvido contempla três funcionalidades principais: geração de um arquivo de dados contendo um número arbitrário de valores de ponto-flutuante, processamento de um único arquivo de dados e a geração de um *benchmark*. Essas três funcionalidades são detalhadas nesta seção.

2.1 Geração do Arquivo de Dados

O algoritmo *count sort* é um algoritmo de ordenamento que foi utilizado neste trabalho para ordenar um vetor de números de ponto-flutuante. Dessa forma, foi necessária a criação de funções para a geração de quantidades configuráveis de números de ponto-flutante e que os gravasse em um formato apropriado para servir de entrada para o programa, possibilitando a fácil experimentação do *count sort* com diferentes entradas de dados.

A interface dessas funções é apresentada a seguir:

```
void generate_floats(FILE* out, int count);
int read_input(FILE* fp, float** vector, int* size);
float* parse_floats(char* values_str, int count);
```

A função `generate_floats` recebe um arquivo de destino e a quantidade de números que se deseja gerar. A função `read_input` recebe um arquivo (no formato gerado por `generate_floats`) e retorna em `vector` e `size` o vetor e seu tamanho, respectivamente. Por fim, a função `parse_floats` é utilizada internamente por `read_input` para realizar o *parsing* de *strings* para *floats*.

[1] `https://github.com/alexandrelucchesi/parallel-programming-ex01`

Para gerar um arquivo de dados, `vec.dat`, contendo, por exemplo, 1000 elementos, basta executar o binário passando-se a *flag* `-gen`, conforme descrito a seguir:

```
$ ./a.out -gen vec.dat 1000
```

2.2 Processamento

O processamento de um arquivo de dados foi encapsulado na função `process`, cuja assinatura é apresentada a seguir:

```
int process(float **vec, int *vec_size, double *time_sorting_only, double
        *time_with_input, const char *filename, omp_sched_t kind, int
        chunk_size);
```

Os quatro primeiros parâmetros: `vec`, `vec_size`, `time_sorting_only` e `time_with_input` são, na verdade, retornos da função `process`, e retornam respectivamente, o vetor de *floats* ordenado, o tamanho do vetor, o tempo transcorrido considerando apenas a execução do *count sort* e o tempo transcorrido desde a entrada de dados até o término da execução da função de ordenação. Os demais parâmetros são utilizados para parametrizar a execução da função *count_sort* e equivalem, respectivamente, ao nome do arquivo de dados, à política de escalonamento do OpenMP (*static*, *dynamic*, *guided* ou *auto*) e ao tamanho do *chunk*.

Encapsular a lógica de processamento na função *process* permitiu a automatização do *benchmarking* da aplicação, por meio do desenvolvimento da função `bench` (descrita a seguir), evitando trabalho manual.

2.3 *Benchmark*

3 Resultados

4 Conclusão

Por fim, é válido ressaltar que o programa está todo parametrizado via diretivas de pré-processamento (`#define`) e aceita alguns parâmetros em tempo de execução (como o tamanho do *chunk* e a política de escalonamento), possibilitando a experimentação com diferentes configurações. O *design* escolhido permite variar de forma fácil a política de escalonamento, o tamanho dos *chunks* e a quantidade de *threads* de forma não intrusiva e modular.

Referências