Instituto Tecnológico de Aeronáutica

CE-265: Processamento Paralelo

Exercício 10: OpenMP para GPU

Bruno Benjamim Bertucci - Turma 23.2

Programa original

Para fins de comparação, o código original, não paralelizado de forma a utilizar somente a CPU, foi executado. A saída da execução, que inclui o tempo de computação, está mostrado abaixo.

```
 MMS heat equation

--------------------
Problem input

 Grid size: 1000 x 1000
 Cell width: 9.990010E-01
 Grid length: 1000.000000 x 1000.000000

 Alpha: 1.000000E-01

 Steps: 10
 Total time: 5.000000E-01
 Time step: 5.000000E-02
--------------------
Stability

 r value: 0.005010
--------------------
Results

Error (L2norm): 3.808796E-10
Solve time (s): 0.045860
Total time (s): 0.170220
--------------------
```

Tarefa 1

Para a primeira tarefa, uma versão mais simples da paralelização OpenMP na GPU foi feita, com uma diretiva para alocar os vetores necessários na GPU e outra para a execução da cooperação de trabalho. Assim, o código da função `solve` foi adaptado para a seguinte forma

```
// Compute the next timestep, given the current timestep
void solve(const int n, const double alpha, const double dx,
const double dt, const double * restrict u, double * restrict u_tmp) {

  // Finite difference constant multiplier
  const double r = alpha * dt / (dx * dx);
  const double r2 = 1.0 - 4.0*r;
```

```
  // Loop over the nxn grid
  #pragma omp target map(to: u[:n*n]) map(from: u_tmp[:n*n])
  #pragma omp teams distribute parallel for simd collapse(2)
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {

      // Update the 5-point stencil, using boundary
      // conditions on the edges of the domain.
      // Boundaries are zero because the MMS solution is zero there.
      u_tmp[i+j*n] =  r2 * u[i+j*n] +
      r * ((i < n-1) ? u[i+1+j*n] : 0.0) +
      r * ((i > 0)   ? u[i-1+j*n] : 0.0) +
      r * ((j < n-1) ? u[i+(j+1)*n] : 0.0) +
      r * ((j > 0)   ? u[i+(j-1)*n] : 0.0);
    }
  }
}
```

Esse novo código foi executado, e a saída está anexada abaixo.

```
 MMS heat equation

--------------------
Problem input

 Grid size: 1000 x 1000
 Cell width: 9.990010E-01
 Grid length: 1000.000000 x 1000.000000

 Alpha: 1.000000E-01

 Steps: 10
 Total time: 5.000000E-01
 Time step: 5.000000E-02
--------------------
Stability

 r value: 0.005010
--------------------
Results

Error (L2norm): 3.808795E-10
Solve time (s): 7.005487
Total time (s): 7.124564
--------------------
```

O tempo de execução é significativamente maior do que a execução não paralelizada, o que pode ser explicado pelo gargalo imposto pela alocação de memória na GPU.

Tarefa 2

Em um segundo experimento, as trocas de dados com a GPU foram separadas em comandos `enter data` e `exit data`, englobando entre eles os laços da computação principal. Esse novo código está conforme ilustrado abaixo.

```
// Compute the next timestep, given the current timestep
void solve(const int n, const double alpha, const double dx,
const double dt, const double * restrict u, double * restrict u_tmp) {

  // Finite difference constant multiplier
  const double r = alpha * dt / (dx * dx);
  const double r2 = 1.0 - 4.0*r;

  // Loop over the nxn grid
  #pragma omp target enter data map(to: u[:n*n]) map(alloc: u_tmp[:n*n])
  #pragma omp target teams distribute parallel for simd collapse(2)
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {

      // Update the 5-point stencil, using boundary
      // conditions on the edges of the domain.
      // Boundaries are zero because the MMS solution is zero there.
      u_tmp[i+j*n] =  r2 * u[i+j*n] +
      r * ((i < n-1) ? u[i+1+j*n] : 0.0) +
      r * ((i > 0)   ? u[i-1+j*n] : 0.0) +
      r * ((j < n-1) ? u[i+(j+1)*n] : 0.0) +
      r * ((j > 0)   ? u[i+(j-1)*n] : 0.0);
    }
  }
  #pragma omp target update from(u_tmp[:n*n])
  #pragma omp target exit data map(delete: u_tmp[:n*n]) map(delete: u[:n*n])
}
```

Para essa implementação, a saída da execução é conforme o anexo abaixo.

```
 MMS heat equation

--------------------
Problem input

 Grid size: 1000 x 1000
 Cell width: 9.990010E-01
 Grid length: 1000.000000 x 1000.000000

 Alpha: 1.000000E-01

 Steps: 10
 Total time: 5.000000E-01
 Time step: 5.000000E-02
--------------------
Stability

 r value: 0.005010
--------------------
Results

Error (L2norm): 3.808795E-10
Solve time (s): 6.322370
Total time (s): 6.442227
--------------------
```

Observa-se que a separação dos comandos de envio e de recebimento de dados entre o *host* e a GPU resultou em uma melhoria no tempo de execução.

Tarefa 3

Finalmente, o código do item anterior foi modificado de forma a otimizar o uso dos caches. Mais especificamente, a ordem na qual os elementos do vetor *u_tmp* eram percorridos foi invertida, de forma que um conjunto de n elementos eram acessados consecutivamente. Espera-se que isso melhore o reaproveitamento de endereços de memória trazidos aos caches, o que por sua vez deve melhorar o tempo de execução. O código associado a essa alteração está ilustrado abaixo.

```
// Compute the next timestep, given the current timestep
void solve(const int n, const double alpha, const double dx,
const double dt, const double * restrict u, double * restrict u_tmp) {

  // Finite difference constant multiplier
  const double r = alpha * dt / (dx * dx);
  const double r2 = 1.0 - 4.0*r;

  // Loop over the nxn grid
  #pragma omp target enter data map(to: u[:n*n]) map(alloc: u_tmp[:n*n])
  #pragma omp target teams distribute parallel for simd collapse(2)
  for (int j = 0; j < n; ++j) {
    for (int i = 0; i < n; ++i) {

      // Update the 5-point stencil, using boundary
      // conditions on the edges of the domain.
      // Boundaries are zero because the MMS solution is zero there.
      u_tmp[i+j*n] =  r2 * u[i+j*n] +
      r * ((i < n-1) ? u[i+1+j*n] : 0.0) +
      r * ((i > 0)   ? u[i-1+j*n] : 0.0) +
      r * ((j < n-1) ? u[i+(j+1)*n] : 0.0) +
      r * ((j > 0)   ? u[i+(j-1)*n] : 0.0);
    }
  }
  #pragma omp target update from(u_tmp[:n*n])
  #pragma omp target exit data map(delete: u_tmp[:n*n]) map(delete: u[:n*n])
}
```

Para essa versão do código, a saída da execução está anexa abaixo.

```
 MMS heat equation

--------------------
Problem input

 Grid size: 1000 x 1000
 Cell width: 9.990010E-01
 Grid length: 1000.000000 x 1000.000000

 Alpha: 1.000000E-01

 Steps: 10
 Total time: 5.000000E-01
 Time step: 5.000000E-02
--------------------
Stability

 r value: 0.005010
--------------------
Results

Error (L2norm): 3.808795E-10
Solve time (s): 6.250987
Total time (s): 6.370607
--------------------
```

Observa-se que houve uma melhora de desempenho com relação ao código anterior, o qual não utilizava os caches de forma eficiente. Apesar disso, o ganho de desempenho foi pequeno, o que pode ser explicado pelo fato de os acessos aos elementos do vetor u ainda serem feitos de forma não-consecutiva, apresentando mais ineficiências no uso dos caches.