TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

o0o

GIẢI BÀI TOÁN TRUYỀN NHIỆT BẰNG PHƯƠNG PHÁP SAI PHÂN HỮU HẠN VỚI OPEMMP

TÍNH TOÁN SONG SONG

Giáo viên hướng dẫn: TS Đoàn Duy Trung
Sinh viên thực hiện : Trần Đại Dương - 20195863
Lớp : Toán tin 02 - K64

Hà Nội - 2021

Mục lục

Chương 1

Giới thiệu bài toán truyền nhiệt

1.1 Giới thiệu chung

Trong thực tế, sự truyền nhiệt của vật chất phụ thuộc vào cả vị trí của từng phần tử cấu thành lên vật chất và thời gian. Bài toán truyền nhiệt ổn định (steady state heat transfer) là khi sự truyền nhiệt của vật chất không phụ thuộc vào thời gian. Bài tiểu luận này sẽ tập trung chủ yếu vào bài toán truyền nhiệt ổn định, cụ thể là khảo sát sự truyền nhiệt của một vật thể trong không gian 2 chiều bằng phương pháp sai phân hữu hạn (finite difference method).

1.2 Mô hình toán học và phương pháp sai phân hữu hạn

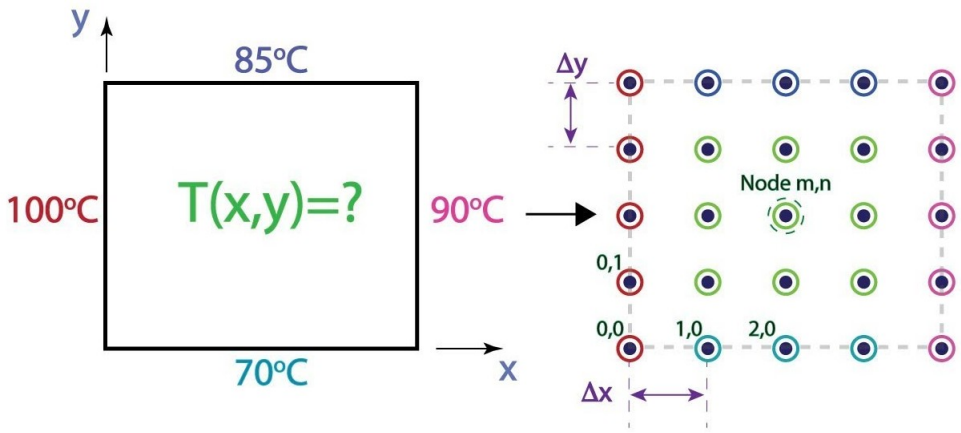
Giả sử có một đĩa phẳng đồng chất đang ở trạng thái ổn định và không chịu tác động của bất cứ nguồn nhiệt nào và dòng dịch chuyển nhiệt (heat flow) tỉ lệ thuận với độ chênh lệch nhiệt độ (tức nhiệt được truyền từ điểm có nhiệt độ cao sang các điểm có nhiệt độ thấp hơn).

Các điểm phía bên trái, bên phải, bên trên và bên dưới của đĩa phẳng có nhiệt độ cố định (gọi là điều kiện biên - boundary condition), như trong hình vẽ dưới lần lượt là 100°C, 90°C, 85°C, và 70°C.

Gắn hệ trục tọa độ Oxy như hình vẽ. Gọi $T_{x,y}$ là nhiệt độ tại điểm (x, y). Ví dụ $T_{0,1} = 70°$ và $T_{1,0} = 100°$. Mục tiêu của bài toán là tính toán sự phân bố nhiệt độ trong đĩa phẳng.

Sự truyền nhiệt trong đĩa phẳng ở trưởng hợp này được mô tả bởi phương trình sau:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$



Phương pháp sai phân hữu hạn:

- Chia trục Ox thành M khoảng bằng nhau, chia trục oy thành N khoảng bằng nhau. Khi đó đĩa phẳng được chia thành $M * N$ phần bằng nhau, gọi mỗi phần này là một nút. Có thể coi đĩa phẳng là một mạng lưới bao gồm các nút (m, n) với $m \in (0, M-1), m \in \mathbb{N}$ và $n \in (0, N-1), n \in \mathbb{N}$ với các điểm phía ngoài (các điểm biên) đã biết trước.

- Biến đổi phương trình nhiệt thành phương trình sai phân hữu hạn có dạng sau:

$$T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1} - 4T_{m,n} = 0$$

suy ra:

$$T_{m,n} = \frac{T_{m+1,n} + T_{m-1,n} + T_{m,n+1} + T_{m,n-1}}{4}$$

Nhận xét:

Việc tính toán tại một điểm lưới (m,n) cần thông tin tại những điểm lưới xung quanh: $(m-1, n), (m+1, j), (m, n-1), (m, n+1)$.

Chương 2

Chương trình tính toán tuần tự

2.1 Thuật toán giải tuần tự:

- **Bước 1:** Gọi W là một mảng hai chiều lưu trữ nhiệt độ của các nút trong mạng. Khởi tạo giá trị ban đầu cho các nút trong mạng với giá trị các nút ở biên được cho trước và các nút bên trong gán bằng giá trị trung bình của các nút ở biên.
- **Bước 2:** Khởi tại U cũng la mảng hai chiều có dung lượng như mảng W để lưu trữ giá trị tạm thời của mảng W.
- **Bước 3:** Bắt đầu đếm thời gian tính toán
- **Bước 4:** Lưu gía trị hiện tại của W vào U
- **Bước 5:** Với mỗi nút nằm phía trong, tính nhiệt độ theo công thức:

$$W(m,n) = (W(m+1,n) + W(m-1,n) + W(m,n+1) + W(m,n-1))/4$$

- **Bước 6:** Tìm sự sai khác giữa giá trị vừa tính W và giá trị cũ U, nếu hội tụ thì chuyển sang bước tiếp theo, nếu không thì quay lại **Bước 4**.
- **Bước 7**: Tính thời gian tính toán và hiển thị kết quả.

2.2 Chi tiết chương trình tính toán tuần tự:

- import các thư viện cần thiết và gán $M = 500$, $N = 500$. (lưới sẽ gồm 500 x 500 nút).

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 #define M 500
7 #define N 500
```

- Tạo hàm main và khai báo các biến

```
 9 int main()
10 {
11   double epsilon = 0.001;
12   double mean = 0.0;
13   double diff; // difference between new and old value of a node
14   double u[M][N]; // for storing the old value of the grid
15   double w[M][N]; // for storing the current value of the grid
16   double wtime;   // for estimate computed time
17   int i, j, iterations, iterations_print; // loop variables
18
19   printf("\n");
20   printf("HEATED_PLATE_OPENMP\n");
21   printf("  C/Serial version\n");
22   printf("  A program to solve for the steady state temperature distribution\n");
23   printf("  over a rectangular plate.\n");
24   printf("\n");
25   printf("  Spatial grid of %d by %d points.\n", M, N);
26   printf("  The iteration will be repeated until the change is <= %e\n", epsilon);
27
```

- Gán giá trị cho các điểm biên: phía trái, phải, trên dưới lần lượt là $100, 100, 100, 0$

```
28   // fix boundary conditions
29   for (i = 1; i < M - 1; i++)
30   {
31     w[i][0] = 100.0;
32     w[i][N - 1] = 100.0;
33   }
34   for (j = 0; j < N; j++)
35   {
36     w[0][j] = 100.0;
37     w[M - 1][j] = 0.0;
38   }
39
```

- Tính giá trị trung bình của các nút biên, lưu vào biến *mean* và gán giá trị đó cho các nút phía trong.

```
40
41   // average the boundary values
42   for (i = 1; i < M - 1; i++)
43   {
44     mean = mean + w[i][0] + w[i][N - 1];
45   }
46   for (j = 0; j < N; j++)
47   {
48     mean = mean + w[M - 1][j] + w[0][j];
49   }
50
51   mean = mean / (double)(2 * M + 2 * N - 4);
52   printf("\n");
53   printf("  MEAN = %f\n", mean);
54
55   // initialize temperature array
56   for (i = 1; i < M - 1; i++)
57   {
58     for (j = 1; j < N - 1; j++)
59     {
60       w[i][j] = mean;
61     }
62   }
63
```

- Gán biến *iteration=0* để đếm số vòng lặp. bắt đầu tình thời gian sử dụng hàm *omp_get_wtime*() của openMP và gán $diff = epsilon$

```
64  // main loop
65  iterations = 0;
66  iterations_print = 1;
67  printf("\n");
68  printf(" Iteration  Change\n");
69  printf("\n");
70  wtime = omp_get_wtime();
71  diff = epsilon;
72
```

- Tạo vòng lặp *while* lặp cho tới khi $epsilon > diff$ (đến khi hội tụ), bước đầu trong vòng lặp là lưu giá trị hiện tại của các nút trong W vào U.

```
73  while (epsilon <= diff)
74  {
75    // Save the old solution in U
76    for (i = 0; i < M; i++)
77    {
78      for (j = 0; j < N; j++)
79      {
80        u[i][j] = w[i][j];
81      }
82    }
83
```

- Tính giá trị của các nút trong dựa vào 4 nút xung quanh

```
84    // Determine the new estimate of the solution at the interior points.
85    for (i = 1; i < M - 1; i++)
86    {
87      for (j = 1; j < N - 1; j++)
88      {
89        w[i][j] = (u[i - 1][j] + u[i + 1][j] + u[i][j - 1] + u[i][j + 1]) / 4.0;
90      }
91    }
92
```

- đặt lại biến $diff = 0$, cập nhật lại chênh lớn nhất giữa kết quả cũ (trong U) và kết quả mới (trong W) vào biến $diff$.

```
93    // Compute the maximun diff
94    diff = 0.0;
95    for (i = 1; i < M - 1; i++)
96    {
97      for (j = 1; j < N - 1; j++)
98      {
99        if (diff < fabs(w[i][j] - u[i][j]))
100       {
101         diff = fabs(w[i][j] - u[i][j]);
102       }
103     }
104   }
105
```

- cập nhật lại biến đếm vòng lặp, hiển thị biến đếm và giá trị $diff$ nếu biến *iteration_print* là số mũ của 2 và kết thúc vòng lặp.

```
106     iterations++;
107     if (iterations == iterations_print)
108     {
109       printf("  %8d  %f\n", iterations, diff);
110       iterations_print = 2 * iterations_print;
111     }
112   }
113
```

- Quá trình tính toán hoàn thất nếu chương trình thực hiện xong vòng lặp. Tiến hành tính toán thời gian thực hiện và hiển thị kết quả.

```
114   wtime = omp_get_wtime() - wtime;
115
116   printf("\n");
117   printf("  %8d  %f\n", iterations, diff);
118   printf("\n");
119   printf("  Error tolerance achieved.\n");
120   printf("  Wallclock time = %f\n", wtime);
121   /*
122   Terminate.
123 */
124   printf("\n");
125   printf("HEATED_PLATE_SERIAL:\n");
126   printf("  Normal end of execution.\n");
127 }
```

Kết quả chạy chương trình:

```
(base) titlehunter@titlehunter:~/Desktop/ttss_cuoi_ky$ gcc -fopenmp heatTransferSerial.c -lm
(base) titlehunter@titlehunter:~/Desktop/ttss_cuoi_ky$ ./a.out

HEATED PLATE OPENMP
  C/Serial version
  A program to solve for the steady state temperature distribution
  over a rectangular plate.

  Spatial grid of 500 by 500 points.
  The iteration will be repeated until the change is <= 1.000000e-03

  MEAN = 74.949900

 Iteration  Change

        1  18.737475
        2  9.368737
        4  4.098823
        8  2.289577
       16  1.136604
       32  0.568201
       64  0.282805
      128  0.141777
      256  0.070808
      512  0.035427
     1024  0.017707
     2048  0.008856
     4096  0.004428
     8192  0.002210
    16384  0.001043

    16955  0.001000

  Error tolerance achieved.
  Wallclock time = 43.220377

HEATED PLATE SERIAL:
  Normal end of execution.
```

Chương 3

Chương trình tính toán song song

Có thể cải tiến chương trình tuần tự bằng cách song song hóa một số bước tính toán như sau:

- Tạo vùng song song để gán giá trị biên và tính giá trị trung bình của các giá trị biên đó với biến chia sẻ là W và biến riêng biệt là các biến đếm i, j.

```
36   // fix boundary condition
37 #pragma omp for
38     for (i = 1; i < M - 1; i++)
39     {
40       w[i][0] = 100.0;
41     }
42 #pragma omp for
43     for (i = 1; i < M - 1; i++)
44     {
45       w[i][N - 1] = 100.0;
46     }
47 #pragma omp for
48     for (j = 0; j < N; j++)
49     {
50       w[M - 1][j] = 100.0;
51     }
52 #pragma omp for
53     for (j = 0; j < N; j++)
54     {
55       w[0][j] = 0.0;
56     }
57
```

  Kết hợp khối song song *for* với mệnh đề *reduction* để tính *mean*. Các khối song song sẽ tính toán theo lệnh trong khối, kết quả sẽ được cập nhật bởi toán tử + và lưu kết quả cuối cùng vào biến *mean*.

```
58     //  average boundary values
59 #pragma omp for reduction(+ : mean)
60     for (i = 1; i < M - 1; i++)
61     {
62       mean = mean + w[i][0] + w[i][N - 1];
63     }
64 #pragma omp for reduction(+ : mean)
65     for (j = 0; j < N; j++)
66     {
67       mean = mean + w[M - 1][j] + w[0][j];
68     }
69   }
70
71   mean = mean / (double)(2 * M + 2 * N - 4);
72   printf("\n");
73   printf("  MEAN = %f\n", mean);
74
```

- Tạo vùng song song, gán giá trị cho các nút trong bằng giá trị của $mean$. Với biến chia sẻ là $W, mean$, biến riêng biệt là i, j

```
75     // initialize temperature array
76 #pragma omp parallel shared(mean, w) private(i, j)
77   {
78 #pragma omp for
79     for (i = 1; i < M - 1; i++)
80     {
81       for (j = 1; j < N - 1; j++)
82       {
83         w[i][j] = mean;
84       }
85     }
86   }
87
```

- Trong vòng lặp $while$, tạo một vùng song song để lưu giá trị hiện tại từ W vào U và tính toán giá trị mới cho W. Các biến chia sẻ là w, u và biến riêng là i, j.

```
 97   while (epsilon <= diff)
 98   {
 99 #pragma omp parallel shared(u, w) private(i, j)
100     {
101
102   // Save the old solution in U.
103 #pragma omp for
104       for (i = 0; i < M; i++)
105       {
106         for (j = 0; j < N; j++)
107         {
108           u[i][j] = w[i][j];
109         }
110       }
111
112   // Determine the new estimate of the solution at the interior points.
113 #pragma omp for
114       for (i = 1; i < M - 1; i++)
115       {
116         for (j = 1; j < N - 1; j++)
117         {
118           w[i][j] = (u[i - 1][j] + u[i + 1][j] + u[i][j - 1] + u[i][j + 1]) / 4.0;
119         }
120       }
121     }
122
```

- Tạo vùng song song để tính biến $diff$, khai báo biến $diffInThread$ là biến riêng trong mỗi thread. các biến $diffInThread$ sẽ được tính trong mỗi thread thực hiện một phần công việc trong khối for và kết quả sẽ lần lượt được cập nhật vào biến $diff$ trong khối $critical$. Các biến chia sẻ là $diff, u, w$ và các biến riêng là $diffInThread, i, j$.

```
123     // Compute the maximun diff
124     diff = 0.0;
125 #pragma omp parallel shared(diff, u, w) private(i, j, diffInThread)
126     {
127       diffInThread = 0.0;
128 #pragma omp for
129       for (i = 1; i < M - 1; i++)
130       {
131         for (j = 1; j < N - 1; j++)
132         {
133           if (diffInThread < fabs(w[i][j] - u[i][j]))
134           {
135             diffInThread = fabs(w[i][j] - u[i][j]);
136           }
137         }
138       }
139 #pragma omp critical
140       {
141         if (diff < diffInThread)
142         {
143           diff = diffInThread;
144         }
145       }
146     }
147
```

Kết quả chạy chương trình:

```
(base) titlehunter@titlehunter:~/Desktop/ttss_cuoi_ky$ gcc -fopenmp heatTransferParallel.c -lm
(base) titlehunter@titlehunter:~/Desktop/ttss_cuoi_ky$ ./a.out

HEATED PLATE OPENMP
  C/OpenMP version
  A program to solve for the steady state temperature distribution
  over a rectangular plate.

  Spatial grid of 500 by 500 points.
  The iteration will be repeated until the change is <= 1.000000e-03
  Number of processors available = 4
  Number of threads =              4

  MEAN = 74.949900

 Iteration Change

        1  18.737475
        2  9.368737
        4  4.098823
        8  2.289577
       16  1.136604
       32  0.568201
       64  0.282805
      128  0.141777
      256  0.070808
      512  0.035427
     1024  0.017707
     2048  0.008856
     4096  0.004428
     8192  0.002210
    16384  0.001043

    16955  0.001000

  Error tolerance achieved.
  Wallclock time = 23.274192

HEATED PLATE OPENMP:
  Normal end of execution.
```

Chương 4

Tổng kết

Dựa vào kết quả tính toán trong chương 2 và chương 3, có thể thấy rõ rằng nếu áp dụng tính toán song song vào bài toán truyền nhiệt nói chung hay các bài toán có khối lượng tính toán lớn có thể tận dụng tối đa sức mạnh của máy tính hiện đại để giảm thời gian tính toán.

Cụ thể đối với bài toán truyền nhiệt ở trên, việc áp dụng tính toán song song với openMP giúp tăng tốc độ tính toán gấp gần hai lần - 43.220377 với tính toán tuần tự và 23.274192 đói với tính toán song song.

Hay đối với bài toán truyền nhiệt trong vật thể 3D, khi khối lượng tính toán còn lớn hơn, thì việc áp dụng tính toán song song cũng cho ra kết quả nhanh hơn so với việc tính toán tuần tự.

Chi tiết có thể tham khảo trong **table1, trang 7,** [**1**] so sánh thời gian thực hiện tuần tự với thời gian thực hiện song song sử dụng CUDA, MPI và OpenMP.

Tài liệu tham khảo

[1] Designing a parallel algorithm for Heat conduction using MPI, OpenMP and CUDA
Vinaya Sivanandan, Vikas Kumar, Srisai Meher

[2] Finite difference method - Wikipedia
`https://en.wikipedia.org/wiki/Finite_difference_method`