МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу

"Суперкомпьютеры и параллельная обработка данных"

**Задание №1:**

### Разработка параллельной версии программы для перемножения матриц с использованием алгоритма Кэннона с использованием технологии OpenMP.

Отчет

студента 325 группы

факультета ВМК МГУ

Дагриджан Артем Владимирович

2019 год

**Постановка задачи**

Разработать параллельную версию программы для перемножения матриц с использованием алгоритма Кэннона с использованием технологии OpenMP, а затем исследовать масштабируемость полученной программы, построить графики зависимости времени её выполнения от числа используемых ядер и объёма входных данных.

Код программы

1. *#include <stdio.h>*
2. *#include <stdlib.h>*
3. *#include <omp.h>*
4. *# include <time.h>*
5. *# include <sys/time.h>*
6. *void get\_walltime(double\* wcTime);*
7. *int data\_set(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);*
8. *int serial\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);*
9. *int mp\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N);*
10. *int main( int argv, char \*\*argc){*
11. *if(argv!=3){*
12. *printf("ERROR: \nPlease enter NUM\_THREADS and NODE\_NUMBERA: \n./run NUM\_THREADS NODE\_NUMBERS\n");*
13. *return- 1;*
14. *}*
15. *int NUM\_THREADS=atoi(argc[1]);*
16. *long long int N=atoi(argc[2]);*
17. *double \*a=(double\*)malloc(N\*N\*sizeof(double));*
18. *double \*b=(double\*)malloc(N\*N\*sizeof(double));*
19. *double \*c=(double\*)malloc(N\*N\*sizeof(double));*
20. *double d\_s, d\_e;*
21. *double d\_s\_mp, d\_e\_mp;*
22. *double d\_s\_mpi, d\_e\_mpi;*
23. *int i, j;*
24. *data\_set(a, b, c, NUM\_THREADS, N);*
25. *get\_walltime(&d\_s);*
26. *serial\_mult(a, b, c, NUM\_THREADS, N);*
27. *get\_walltime(&d\_e);*
28. *data\_set(a, b, c, NUM\_THREADS, N);*
29. *printf("openMP:\n");*
30. *get\_walltime(&d\_s\_mp);*
31. *mp\_mult(a, b, c, NUM\_THREADS, N);*
32. *get\_walltime(&d\_e\_mp);*
33. *printf("-----------------\nN=%lld, NUM\_THREADS=%d\n", N, NUM\_THREADS);*
34. *printf("serial time: %lf\n", d\_e-d\_s);*
35. *printf("openMP time: %lf\n-----------------\n\n\n", d\_e\_mp-d\_s\_mp);*
36. *free(a);*
37. *free(b);*
38. *free(c);*
39. *return 0;*
40. *}*
41. *int mp\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N){*
42. *int i, j, k;*
43. *omp\_set\_num\_threads(NUM\_THREADS);*
44. *#pragma omp parallel shared(a,b,c) private(i,j,k)*
45. *{*
46. *#pragma omp for schedule(dynamic)*
47. *for(i=0; i<N; i++){*
48. *for(j=0; j<N; j++){*
49. *for(k=0; k<N; k++){*
50. *c[i\*N+j]+=a[i\*N+k]\*b[k\*N+j];*
51. *}*
52. *}*
53. *}*
54. *}*
55. *return 0;*
56. *}*
57. *int serial\_mult(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N){*
58. *int i, j, k;*
59. *for(i=0; i<N; i++){*
60. *for(j=0; j<N; j++){*
61. *for(k=0; k<N; k++){*
62. *c[i\*N+j]+=a[i\*N+k]\*b[k\*N+j];*
63. *}*
64. *}*
65. *}*
66. *return 0;*
67. *}*
68. *int data\_set(double \*a, double \*b, double \*c, int NUM\_THREADS, long long int N){*
69. *int i, j;*
70. *for(i=0; i<N; i++){*
71. *for(j=0; j<N; j++){*
72. *a[i\*N+j]=2.0;*
73. *b[i\*N+j]=3.0;*
74. *c[i\*N+j]=0.0;*
75. *}*
76. *}*
77. *return 0;*
78. *}*
79. *void get\_walltime(double\* wcTime){*
80. *struct timeval tp;*
81. *gettimeofday(&tp, NULL);*
82. *\*wcTime = (double)(tp.tv\_sec + tp.tv\_usec/1000000.0);*
83. *}*

**Результаты замеров времени выполнения**

Работа задачи рассмотрена на суперкомпьютере Polus с различным числом нитей (от 1 до 64 ) и различными размерами матрицы (от 500 до 2000).

Также был проведен результат для более большой матрицы, размерами (10^4 )при количестве нитей ( от 10 до 100).

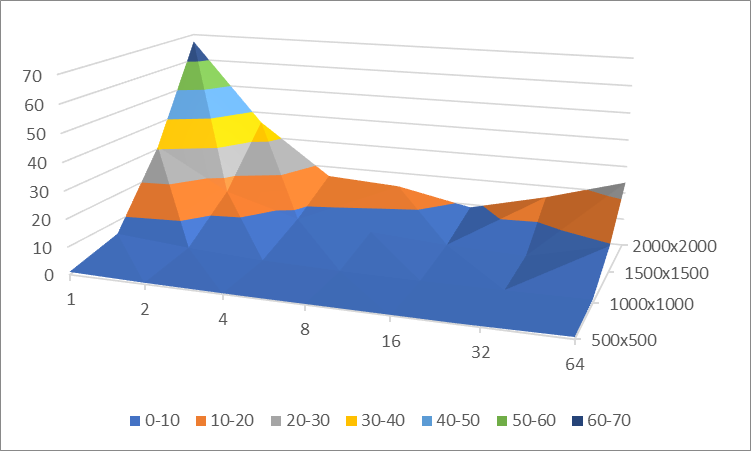
Каждое измерение проводилось 3 раза. В таблице и на графиках записаны усредненные результаты времени выполнения.

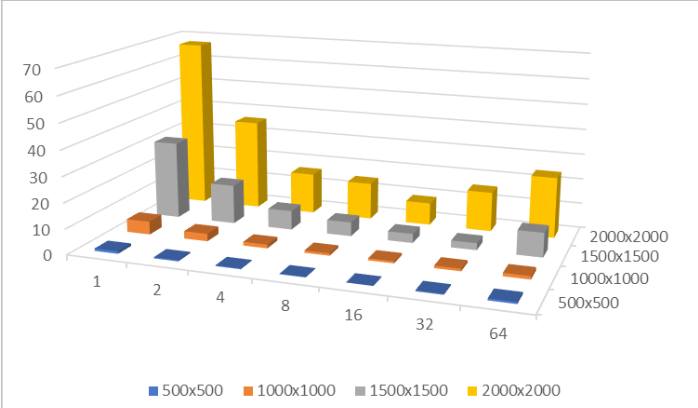
**Таблица с результатами**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Нити/размер матрицы | 500 | 1000 | 1500 | 2000 |
| 1 | 0.979765 | 5.220071 | 30.845354 | 67.194614 |
| 2 | 0.497535 | 2.993845 | 15.575311 | 36.031578 |
| 4 | 0.325390 | 1.489782 | 7.602025 | 16.217851 |
| 8 | 0.152676 | 0.943819 | 5.553080 | 14.617279 |
| 16 | 0.092484 | 0.795361 | 3.636858 | 12.876796 |
| 32 | 0.353654 | 0.874929 | 2.743467 | 8.67863 |
| 64 | 0.721731 | 1.298607 | 9.545367 | 23.97678 |

|  |  |
| --- | --- |
| Нити/размер матрицы | 10^4 |
| 10 | 986.635844 |
| 20 | 712.362738 |
| 30 | 482.23092 |
| 40 | 613.456213 |
| 50 | 610.856213 |
| 60 | 764.263356 |
| 80 | 723.492590 |
| 100 | 732.252536 |

**Графики: время выполнения программы в зависимости от размера матрицы и количества потоков**





**Вывод:**

Согласно результатам и диаграмме графика OpenMP, мы можем знать, что время выполнения увеличивается, когда размерность матрицы увеличивается. Затем время выполнения уменьшается, когда нити увеличивается.

Но когда нить >=40, то время работы увеличивается или устойчиво от времени работы нить = 30.

Также протестировал нити от 30 до 40, и получается, что нить-32 является лучшим дизайном для дизайна версии OpenMP. А нити от 32 до 100, что не очень полезны для версии OpenMP в данном алгоритме.