Эффективное программирование современных микропроцессоров и мультипроцессоров

Практическое задание 4

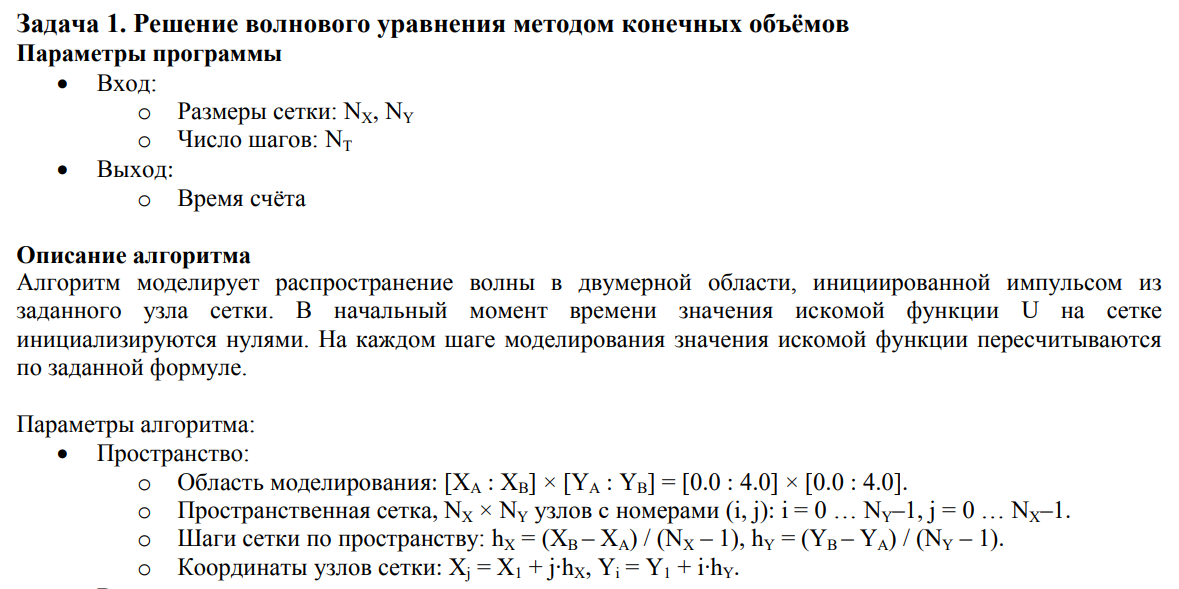
Выполнила:

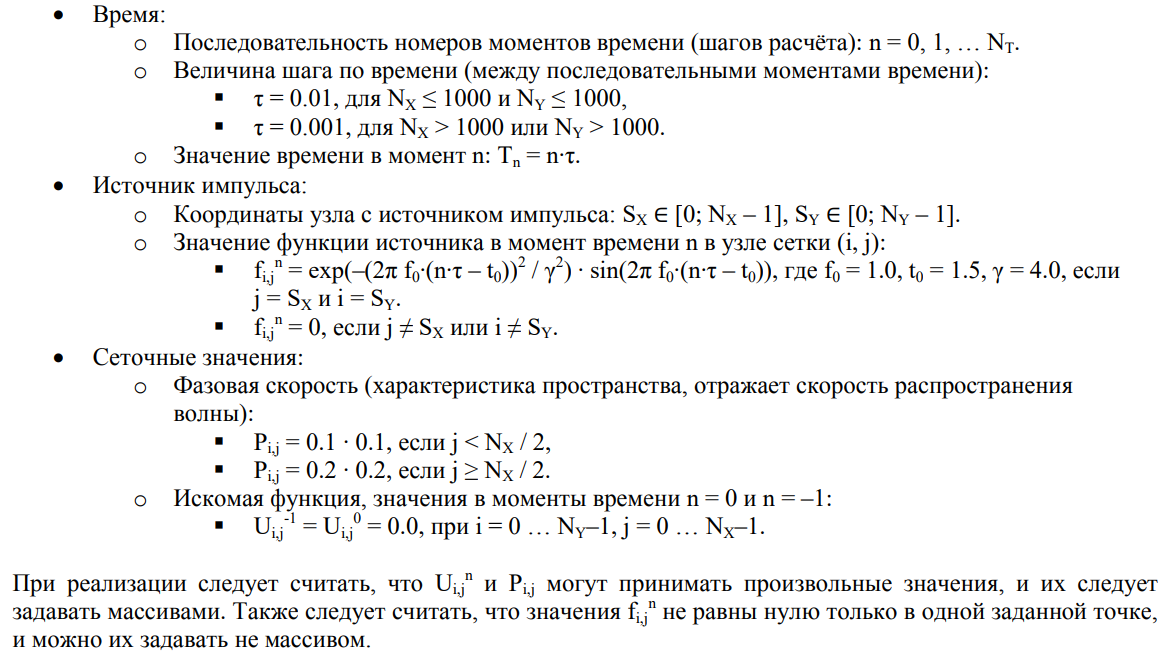
…

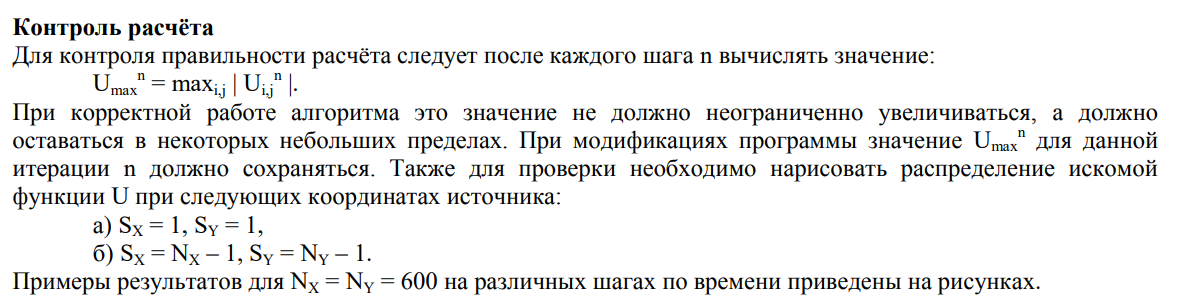
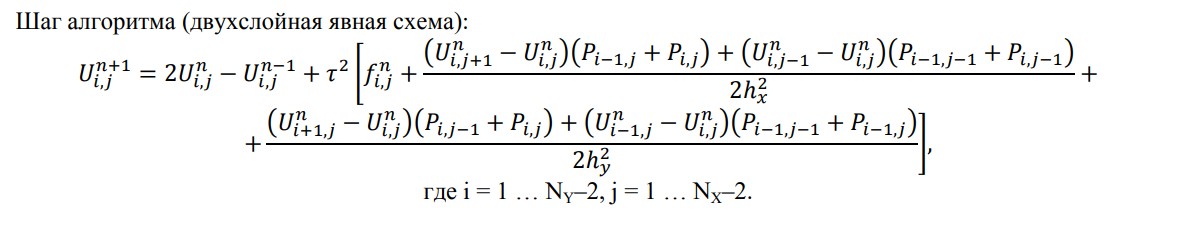
Группа …

Преподаватель:

…







**Результаты тестирования различных версий программы**

Nx = Ny = 8000, Nt = 100;

1. Изначальный вариант программы

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 204.2 | 47.2 | 37.5 | 22.9 |

1. Исключение повторяющихся вычислений

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 203.0 | 42.6 | 34.6 | 22.3 |

1. Замена всех операций деления на умножение

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 195.2 | 42.0 | 33.4 | 21.9 |

1. Заблаговременное обращение к массиву

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 139.7 | 38.7 | 32.9 | 22.4 |

1. Заблаговременное вычисление индекса элемента массива

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 102.8 | 31.5 | 24.5 | 23.7 |

1. Векторизация (n+=1)

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 101.2 | 18.2 | 15.9 | 15.4 |

1. n+=4

|  |  |  |  |
| --- | --- | --- | --- |
| Без оптимизации | -О1 | -О3 | -Ofast |
| 109.2 | 15.6 | 10.5 | 10.7 |

1. threads = 2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | -Оfast без барьеров | -Ofast с барьерами |
|  |  | 6.0 | 6.5 |

1. threads = 3

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | -Оfast | -Ofast |
|  |  | 5.5 | 5.9 |

1. threads = 4

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | -Оfast | -Ofast |
|  |  | 5.7 | 6.0 |

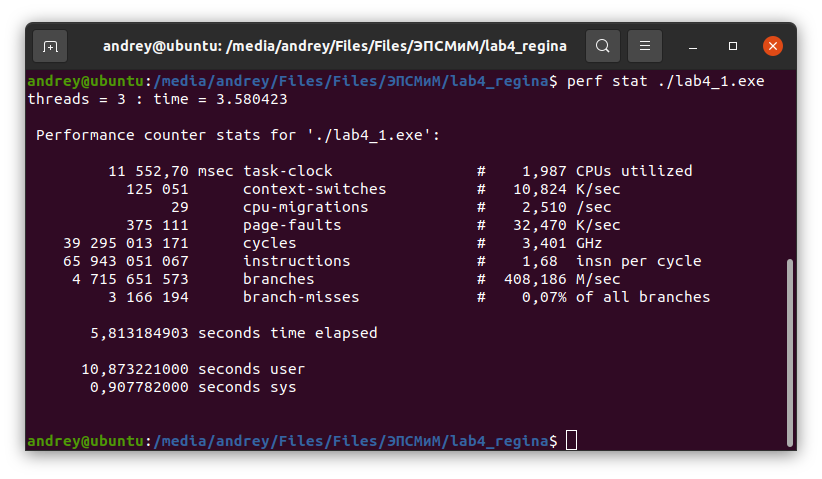
1. threads = 5

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | -Оfast | -Ofast |
|  |  | 6.4 | 6.8 |

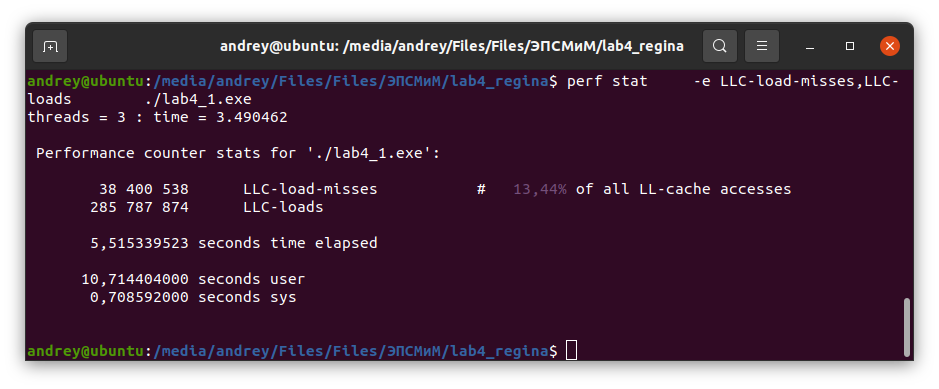
**Вывод**

Распараллеливание в потоках ускорило выполнение программы в 1.6 раз. Из приведённых выше результатов видно, что при барьерной синхронизации итоговое время получается хуже, чем при попарной синхронизации между потоками с помощью флаговых переменных и атомарных операций с ними, ведь каждый поток теперь должен ждать завершение вычислений не всех остальных, а только двух своих соседей. Также видно, что несмотря на четырехъядерность системы, лучшее время достигается на трёх потоках. Это связано с ограниченностью пропускной способностью памяти, то есть потоки хотят получать больше данных, чем память способна предоставить единовременно, и вынуждены ждать.

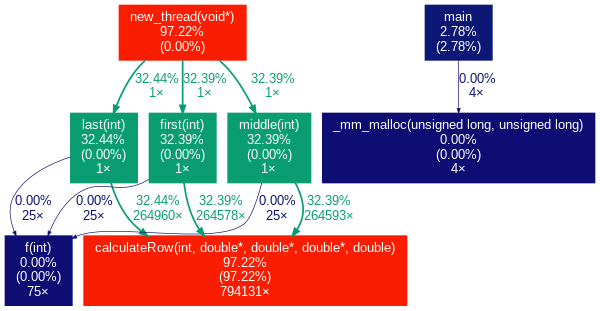
**Профилирование perf**



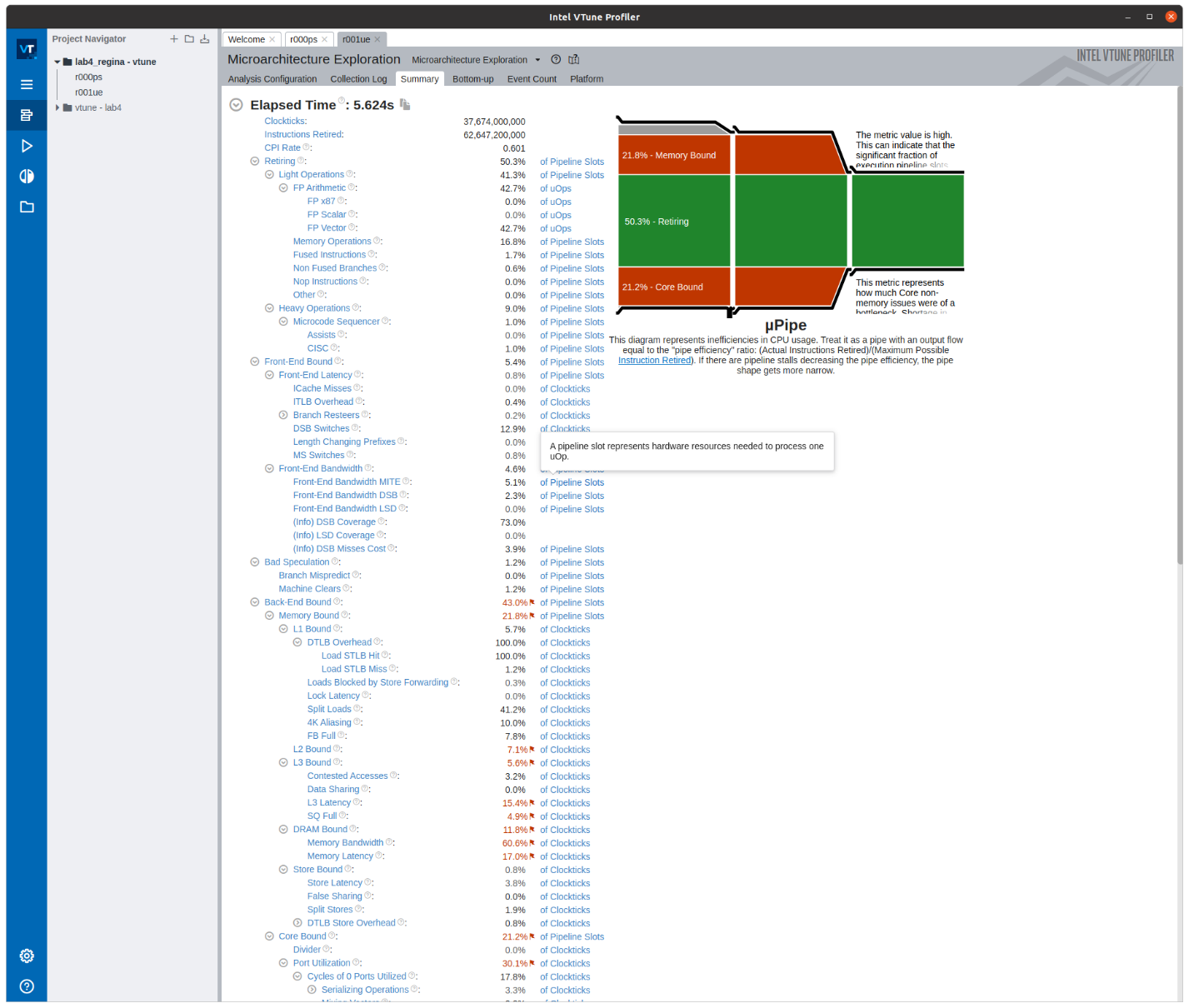




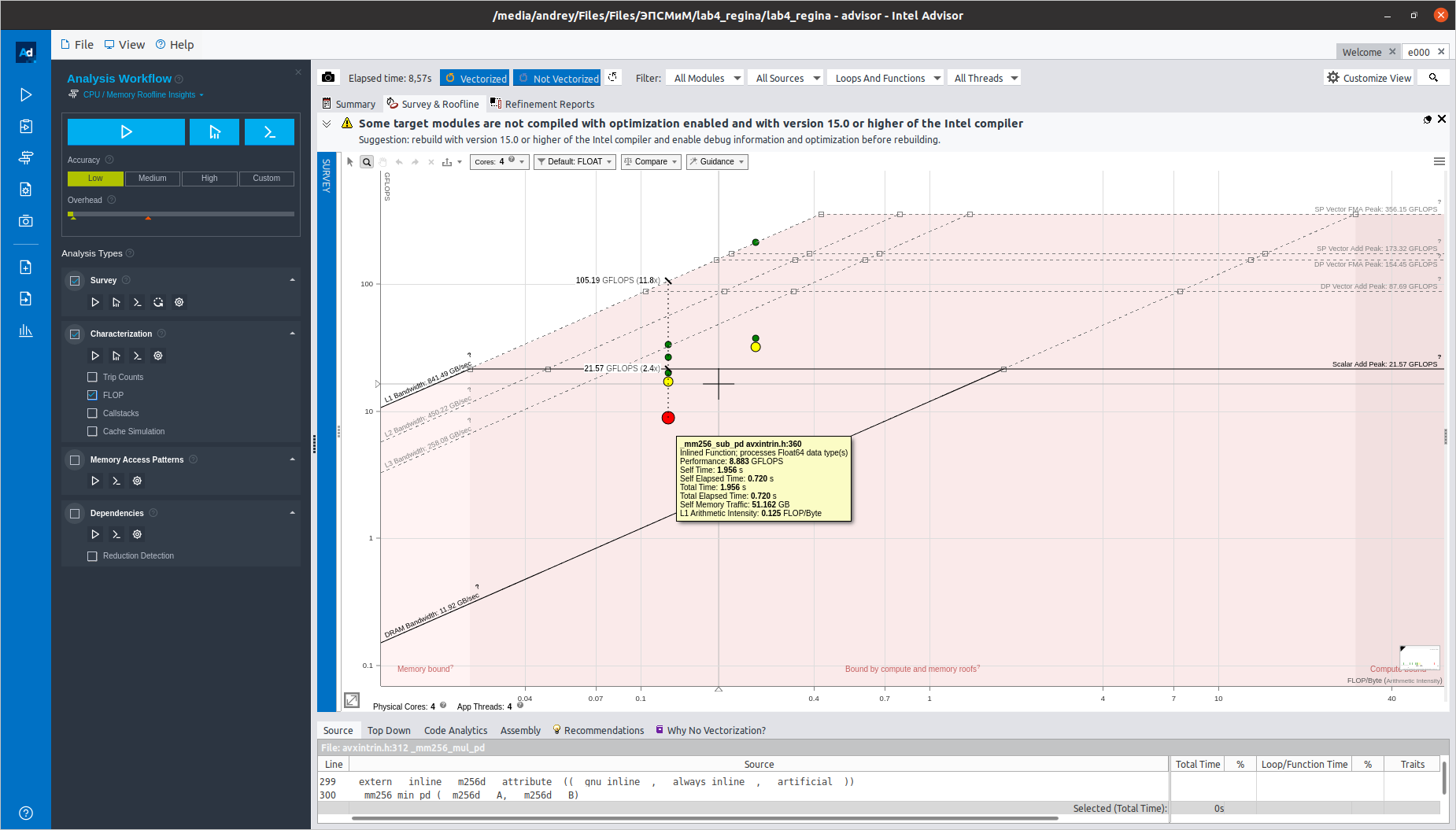
**Граф вызовов**



**Профилирование Vtune**



**Roofline-модель**



#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <math.h>

#include <time.h>

#include <immintrin.h>

#include <pthread.h>

const int r = 4;

const int threads = 3;

pthread\_t\* thread;

double\*\* U;

double\* P;

const int Nx = 8000, Ny = 8000, Nt = 100;

const int Sx = 1, Sy = 1;

const double Xa = 0.0, Xb = 4.0, Ya = 0.0, Yb = 4.0;

double hx, hy;

double t;

const int ALIGN\_INDENT = 64;

const int VECTOR\_SIZE = 4;

double t2\_arr[VECTOR\_SIZE];

double d\_x\_arr[VECTOR\_SIZE];

double d\_y\_arr[VECTOR\_SIZE];

const double two\_arr[VECTOR\_SIZE] = {2.0, 2.0, 2.0, 2.0};

int\* green;

int\* red;

int\* blue;

volatile int green\_wait\_blue[16 \* 32];

volatile int blue\_wait\_green[16 \* 32];

double f (int n) {

double f0 = 1.0, t0 = 1.5, c = 4.0;

double arg = 2\*3.14\*f0\*(n\*t - t0);

return exp(-0.0625 \* (arg\*arg)) \* sin(arg);

}

void calculateRow(int i, double \*U\_prv, double \*U\_new, double \*P, double f\_value) {

double f\_arr[VECTOR\_SIZE];

int row = i \* Nx, row\_ = (i - 1) \* Nx, row1 = (i + 1) \* Nx;

for (int j = 1; j < Ny - 3; j += VECTOR\_SIZE) {

int rowcol = row + j, rowcol\_ = row + j - 1, rowcol1 = row + j + 1,

row\_col = row\_ + j, row1col = row1 + j, row\_col\_ = row\_col - 1;

if (i == Sx && (Sy - j) < VECTOR\_SIZE && (Sy - j) >=0)

f\_arr[(Sy - j)] = f\_value;

else

memset(f\_arr, 0.0, 4 \* sizeof(double));

\_\_m256d first = \_mm256\_loadu\_pd(U\_prv+rowcol1);

\_\_m256d second = \_mm256\_loadu\_pd(U\_prv+rowcol);

\_\_m256d res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col);

second = \_mm256\_loadu\_pd(P+rowcol);

\_\_m256d res2 = \_mm256\_add\_pd(first, second);

\_\_m256d firstPath = \_mm256\_mul\_pd(res1, res2);

first = \_mm256\_loadu\_pd(U\_prv+rowcol\_);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col\_);

second = \_mm256\_loadu\_pd(P+rowcol\_);

res2 = \_mm256\_add\_pd(first, second);

\_\_m256d secondPath = \_mm256\_mul\_pd(res1, res2);

\_\_m256d lineDividend = \_mm256\_add\_pd(firstPath, secondPath);

first = \_mm256\_loadu\_pd(d\_x\_arr);

\_\_m256d firstLineResult = \_mm256\_mul\_pd(lineDividend, first);

first = \_mm256\_loadu\_pd(U\_prv+row1col);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+rowcol\_);

second = \_mm256\_loadu\_pd(P+rowcol);

res2 = \_mm256\_add\_pd(first, second);

firstPath = \_mm256\_mul\_pd(res1, res2);

first = \_mm256\_loadu\_pd(U\_prv+row\_col);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

res1 = \_mm256\_sub\_pd(first, second);

first = \_mm256\_loadu\_pd(P+row\_col\_);

second = \_mm256\_loadu\_pd(P+row\_col);

res2 = \_mm256\_add\_pd(first, second);

secondPath = \_mm256\_mul\_pd(res1, res2);

lineDividend = \_mm256\_add\_pd(firstPath, secondPath);

first = \_mm256\_loadu\_pd(d\_y\_arr);

\_\_m256d secondLineResult = \_mm256\_mul\_pd(lineDividend, first);

\_\_m256d sumOfTwoLines = \_mm256\_add\_pd(firstLineResult, secondLineResult);

\_\_m256d impulseVector = \_mm256\_loadu\_pd(f\_arr);

\_\_m256d bracketResult = \_mm256\_add\_pd(sumOfTwoLines, impulseVector);

first = \_mm256\_loadu\_pd(two\_arr);

second = \_mm256\_loadu\_pd(U\_prv+rowcol);

\_\_m256d totalResult = \_mm256\_mul\_pd(first, second);

second = \_mm256\_loadu\_pd(U\_new+rowcol);

totalResult = \_mm256\_sub\_pd(totalResult, second);

\_\_m256d tauSquareVector = \_mm256\_loadu\_pd(t2\_arr);

\_\_m256d mulResult = \_mm256\_mul\_pd(tauSquareVector, bracketResult);

totalResult = \_mm256\_add\_pd(totalResult, mulResult);

\_mm256\_storeu\_pd(U\_new+rowcol, totalResult);

}

}

void one\_thread(int thread\_id){

for (int n = 1; n < Nt; n += 4) {

double\* U\_new = U[n%2];

double\* U\_prv = U[(n+1)%2];

double f\_value = f(n);

calculateRow(1, U\_prv, U\_new, P, f\_value);

calculateRow(2, U\_prv, U\_new, P, f\_value);

calculateRow(1, U\_new, U\_prv, P, f\_value);

calculateRow(3, U\_prv, U\_new, P, f\_value);

calculateRow(2, U\_new, U\_prv, P, f\_value);

calculateRow(1, U\_prv, U\_new, P, f\_value);

for (int i = 4; i < Ny - 1; i++) {

calculateRow(i, U\_prv, U\_new, P, f\_value);

calculateRow(i-1, U\_new, U\_prv, P, f\_value);

calculateRow(i-2, U\_prv, U\_new, P, f\_value);

calculateRow(i-3, U\_new, U\_prv, P, f\_value);

}

calculateRow(Ny-2, U\_new, U\_prv, P, f\_value);

calculateRow(Ny-3, U\_prv, U\_new, P, f\_value);

calculateRow(Ny-2, U\_prv, U\_new, P, f\_value);

calculateRow(Ny-4, U\_new, U\_prv, P, f\_value);

calculateRow(Ny-3, U\_new, U\_prv, P, f\_value);

calculateRow(Ny-2, U\_new, U\_prv, P, f\_value);

}

}

void first(int thread\_id){

for (int it = 1; it < Nt; it += r){

int f\_value = f(it);

for (int k = 1; k < r; k++)

{

for (int p = green[thread\_id \* 16] + (k - 1), l = 1; l <= k; p--, l++)

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

for (int p = red[thread\_id \* 16]; p < green[(thread\_id + 1) \* 16]; p++)

{

for (int l = 1; l <= r; l++)

calculateRow(p - (l - 1), U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

while(blue\_wait\_green[(thread\_id + 1) \* 16] < it);

for (int k = 2; k <= r; k++)

{

for (int p = blue[thread\_id \* 16] + (k - 2), l = k; l <= r; p--, l++) {

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

calculateRow(p + 1, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

}

green\_wait\_blue[thread\_id \* 16] = it;

}

}

void middle(int thread\_id){

for (int it = 1; it < Nt; it += r){

int f\_value = f(it);

while (green\_wait\_blue[(thread\_id - 1) \* 16] < it - r);

for (int k = 1; k < r; k++)

{

for (int p = green[thread\_id \* 16] + (k - 1) \* 2, l = 1; l <= k; p--, l++) {

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

calculateRow(p+1, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

}

blue\_wait\_green[thread\_id \* 16] = it;

for (int p = red[thread\_id \* 16]; p < green[(thread\_id + 1) \* 16]; p++)

{

for (int l = 1; l <= r; l++)

calculateRow(p - (l - 1), U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

while(blue\_wait\_green[(thread\_id + 1) \* 16] < it);

for (int k = 2; k <= r; k++)

{

for (int p = blue[thread\_id \* 16] + (k - 2), l = k; l <= r; p--, l++) {

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

calculateRow(p + 1, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

}

green\_wait\_blue[thread\_id \* 16] = it;

}

}

void last(int thread\_id){

for (int it = 1; it < Nt; it += r){

int f\_value = f(it);

while (green\_wait\_blue[(thread\_id - 1) \* 16] < it - r);

for (int k = 1; k < r; k++)

{

for (int p = green[thread\_id \* 16] + (k - 1) \* 2, l = 1; l <= k; p--, l++) {

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

calculateRow(p + 1, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

}

blue\_wait\_green[thread\_id \* 16] = it;

for (int p = red[thread\_id \* 16]; p <= Nx - 2; p++)

{

for (int l = 1; l <= r; l++)

calculateRow(p - (l - 1), U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

for (int k = 2; k <= r; k++)

{

for (int p = blue[thread\_id \* 16], l = k; l <= r; p--, l++)

calculateRow(p, U[(it + l) % 2], U[(it + l - 1) % 2], P, f\_value);

}

}

}

void\* new\_thread(void\* param) {

int thread\_id = \*((int\*)param);

free((int\*)param);

if (threads == 1)

one\_thread(thread\_id);

else

if (thread\_id == 0)

first(thread\_id);

else

if (thread\_id == threads - 1)

last(thread\_id);

else

middle(thread\_id);

pthread\_exit(NULL);

}

int main(){

hx = (Xb - Xa) / (Nx - 1), hy = (Yb - Ya) / (Ny - 1);

if (Nx <= 1000 && Ny <= 1000) t = 0.01;

else t = 0.001;

P = (double\*)\_mm\_malloc(Nx \* Ny \* sizeof(double), ALIGN\_INDENT);

for (int i = 0; i < Ny; i++)

for (int j = 0; j < Nx; j++) {

if (j < Nx/2) P[i\*Nx + j] = 0.1\*0.1;

else P[i\*Nx + j] = 0.2\*0.2;

}

U = (double\*\*)\_mm\_malloc(2 \* sizeof(double\*), ALIGN\_INDENT);

U[0] = (double\*)\_mm\_malloc(Nx \* Ny \* sizeof(double), ALIGN\_INDENT);

U[1] = (double\*)\_mm\_malloc(Nx \* Ny \* sizeof(double), ALIGN\_INDENT);

for (int i = 0; i < Ny; i++)

for (int j = 0; j < Nx; j++) {

U[0][i\*Nx + j] = 0.0;

U[1][i\*Nx + j] = 0.0;

}

double d\_x = 0.5 / (hx \* hx), d\_y = 0.5 / (hy \* hy), t2 = t \* t;

for (int i = 0; i < VECTOR\_SIZE; i++) {

t2\_arr[i] = t2;

d\_x\_arr[i] = d\_x;

d\_y\_arr[i] = d\_y;

}

green = (int\*)calloc(threads \* 16, sizeof(int));

red = (int\*)calloc(threads \* 16, sizeof(int));

blue = (int\*)calloc(threads \* 16, sizeof(int));

memset(green, 0, threads \* 16\* sizeof(int));

memset(red, 0, threads \* 16 \* sizeof(int));

memset(blue, 0, threads \* 16 \* sizeof(int));

for (int i = 0; i < threads; i++) {

green[i \* 16] = i \* (Ny / threads) - (r - 1);

red[i \* 16] = i \* (Ny / threads) + (r - 1);

blue[i \* 16] = (i + 1) \* (Ny / threads) - r;

}

green[0] = 1;

red[0]++;

blue[(threads - 1) \* 16] = Ny - r;

for (int i = 0; i < 16 \* 32; i++) {

green\_wait\_blue[i] = 0;

blue\_wait\_green[i] = 0;

}

struct timespec start, finish;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

thread = (pthread\_t\*)calloc(threads, sizeof(pthread\_t));

for (int i = 0; i < threads; i++) {

int\* thread\_ID = (int\*)malloc(sizeof(int));

(\*thread\_ID) = i;

int error\_code = pthread\_create(&thread[i], NULL, new\_thread, thread\_ID);

if (error\_code != 0)

printf("[MAIN | Waring] Error in create thread.");

}

for (int i = 0; i < threads; i++)

pthread\_join(thread[i], NULL);

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &finish);

double time = finish.tv\_sec - start.tv\_sec + 0.000000001 \* (finish.tv\_nsec - start.tv\_nsec);

printf("threads = %d : time = %lf\n", threads, time);

FILE\* file = fopen("file.dat", "wb");

fwrite(U[1], sizeof(double), Nx \* Ny, file);

fclose(file);

free(U[0]);

free(U[1]);

free(P);

free(green);

free(red);

free(blue);

free(thread);

return 0;

}