**#7**

#include <iostream>

#include <omp.h>

using namespace std;

int main(){

const int n = 12;

int a[n];

int b[n];

int c[n];

omp\_set\_num\_threads(3);

#pragma omp parallel

{

#pragma omp for schedule(static,6)

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

{

a[i] = rand() % n + 1;

b[i] = rand() % n + 1;

printf("Threads - %d\nNum of thread - %d\n", omp\_get\_max\_threads(), omp\_get\_thread\_num());

}

}

printf("\n");

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

printf("%d ", a[i]);

} printf("\n");

printf("\n");

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

printf("%d ", b[i]);

} printf("\n");

printf("\n");

omp\_set\_num\_threads(4);

#pragma omp parallel

{

#pragma omp for schedule(dynamic,3)

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

{

c[i] = a[i] + b[i];

printf("Threads - %d\nNum of thread - %d\n", omp\_get\_max\_threads(), omp\_get\_thread\_num());

}

}

printf("\n");

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

printf("%d ", c[i]);

} printf("\n");

return 0;

}

**#8**

#include <iostream>

#include <omp.h>

#include<ctime>

#include <cstdio>

using namespace std;

int\* consist(int\*\* a, int\* b, int n) {

int\* result = new int[n];

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

{

result[i] = 0;

for (int j = 0; j < n; j++)

result[i] += a[i][j] \* b[j];

}

return result;

}

int\* p\_static(int\*\* a, int\* b, int n) {

int\* result = new int[n];

#pragma omp parallel

{

#pragma omp for schedule(static,5)

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

{

result[i] = 0;

for (int j = 0; j < n; j++)

result[i] += a[i][j] \* b[j];

}

}

return result;

}

int\* p\_dynamic(int\*\* a, int\* b, int n) {

int\* result = new int[n];

#pragma omp parallel

{

#pragma omp for schedule(dynamic,5)

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

{

result[i] = 0;

for (int j = 0; j < n; j++)

result[i] += a[i][j] \* b[j];

}

}

return result;

}

int main(){

int r = 1000;

int c = 1000;

int\*\* a = new int\*[r];

int\* b = new int[r];

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

{

a[i] = new int[c];

}

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

for (int j = 0; j < c; j++)

a[i][j] = rand() % 9 + 1;

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

b[i] = rand() % 9 + 1;

//SHOW MASSIVE AND VECTOR

/\*

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

{

for (int j = 0; j < c; ++j)

{

printf("%d ", a[i][j]);

}

printf("\n");

} printf("\n");

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

{

printf("%d ", b[i]);

} printf("\n");

printf("\n");

\*/

clock\_t start1 = clock();

int\* result1 = p\_dynamic(a, b, r);

clock\_t end1 = clock();

double seconds1 = (double)(end1 - start1) / CLOCKS\_PER\_SEC;

printf("P\_DYNAMIC: %f\n", seconds1);

//RESULT OF MULT

/\*

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

{

printf("%d ", result1[i]);

} printf("\n");

\*/

clock\_t start2 = clock();

int\* result2 = p\_static(a, b, r);

clock\_t end2 = clock();

double seconds2 = (double)(end2 - start2) / CLOCKS\_PER\_SEC;

printf("P\_STATIC: %f\n", seconds2);

//RESULT OF MULT

/\*

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

{

printf("%d ", result2[i]);

} printf("\n");

\*/

clock\_t start3 = clock();

int\* result3 = consist(a, b, r);

clock\_t end3 = clock();

double seconds3 = (double)(end3 - start3) / CLOCKS\_PER\_SEC;

printf("CONSIST: %f\n", seconds3);

//RESULT OF MULT

/\*

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

{

printf("%d ", result3[i]);

} printf("\n");

\*/

delete[]result1;

delete[]result2;

delete[]result3;

delete[]b;

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

delete[] a[i];

delete[]a;

return 0;

}

Результат работы программы:

