|  |  |  |
| --- | --- | --- |
| ПрИб-181 | Лабораторный практикум СРС | Зачёт |
| Кащенко В. А. | СРС 01.11.2021 |  |

Цель работы: Выполнить задание m.volsu

Задание:

Напишите параллельную программу с использованием OpenMP, в которой N нитей параллельно вычисляют сумму чисел от 1 до N. Для сложения результатов вычисления нитей воспользуйтесь OpenMP-параметром reduction,  директивой atomic, директивой critical, и замками (lock).  Проведите серию экспериментов на персональном компьютере по исследованию влияния различных инструментов синхронизации на время работы программы.  Оформить подробный отчет в формате pdf.

Код программы:

#include <stdio.h>

#include <omp.h>

int main()

{

int k, N;

int sum = 0;

double start\_time, end\_time;

scanf("%d%d", &k, &N);

// reduction

start\_time = omp\_get\_wtime();

#pragma omp parallel num\_threads(k) reduction(+: sum)

{

#pragma omp for

for (int i = 1; i <= N; ++i)

sum += i;

}

end\_time = omp\_get\_wtime();

printf("Sum = %d\n", sum);

printf("Время работы reduction = %f\n\n", end\_time - start\_time);

sum = 0;

// atomic

start\_time = omp\_get\_wtime();

#pragma omp parallel num\_threads(k)

{

#pragma omp for

for (int i = 1; i <= N; ++i)

{

#pragma omp atomic

sum+=i;

}

}

end\_time = omp\_get\_wtime();

printf("Sum = %d\n", sum);

printf("Время работы atomic = %f\n\n", end\_time - start\_time);

sum = 0;

// critical

start\_time = omp\_get\_wtime();

#pragma omp parallel num\_threads(k)

{

#pragma omp for

for (int i = 1; i <= N; ++i)

{

#pragma omp critical

sum+=i;

}

}

end\_time = omp\_get\_wtime();

printf("Sum = %d\n", sum);

printf("Время работы critical = %f\n\n", end\_time - start\_time);

sum = 0;

// lock

start\_time = omp\_get\_wtime();

omp\_lock\_t lock;

omp\_init\_lock(&lock);

#pragma omp parallel num\_threads(k)

{

omp\_set\_lock(&lock);

sum += omp\_get\_thread\_num() + 1;

omp\_unset\_lock(&lock);

}

end\_time = omp\_get\_wtime();

omp\_destroy\_lock(&lock);

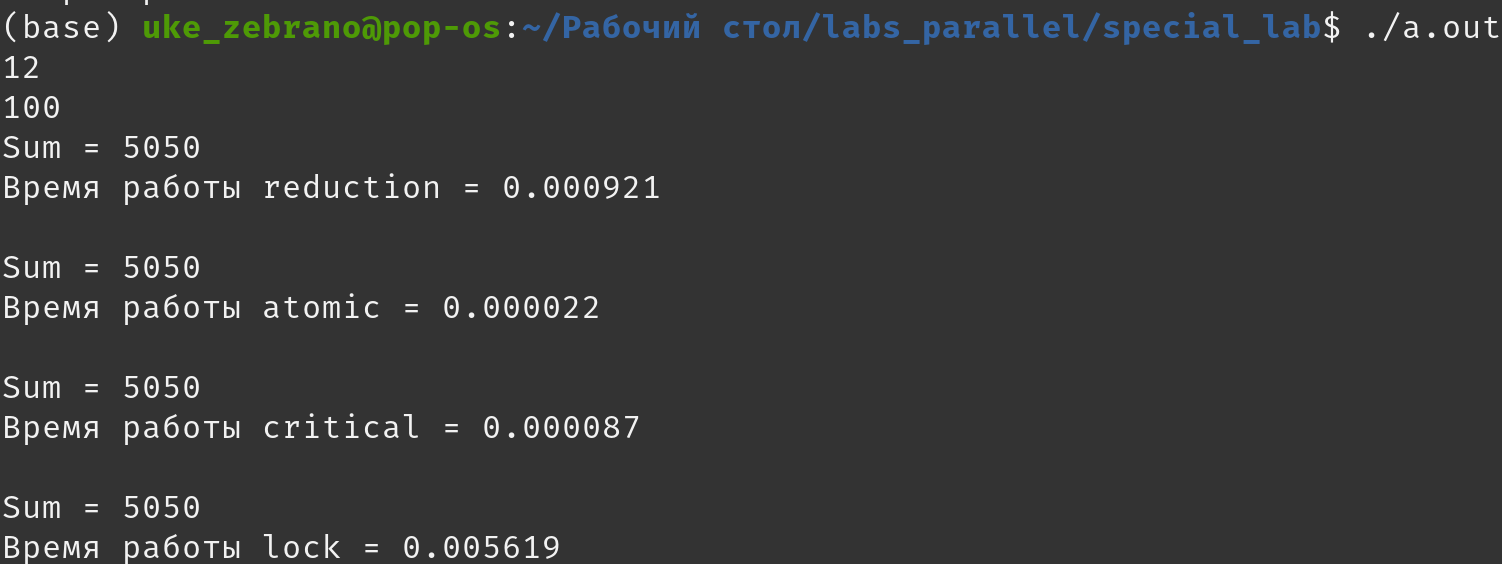
printf("Sum = %d\n", sum);

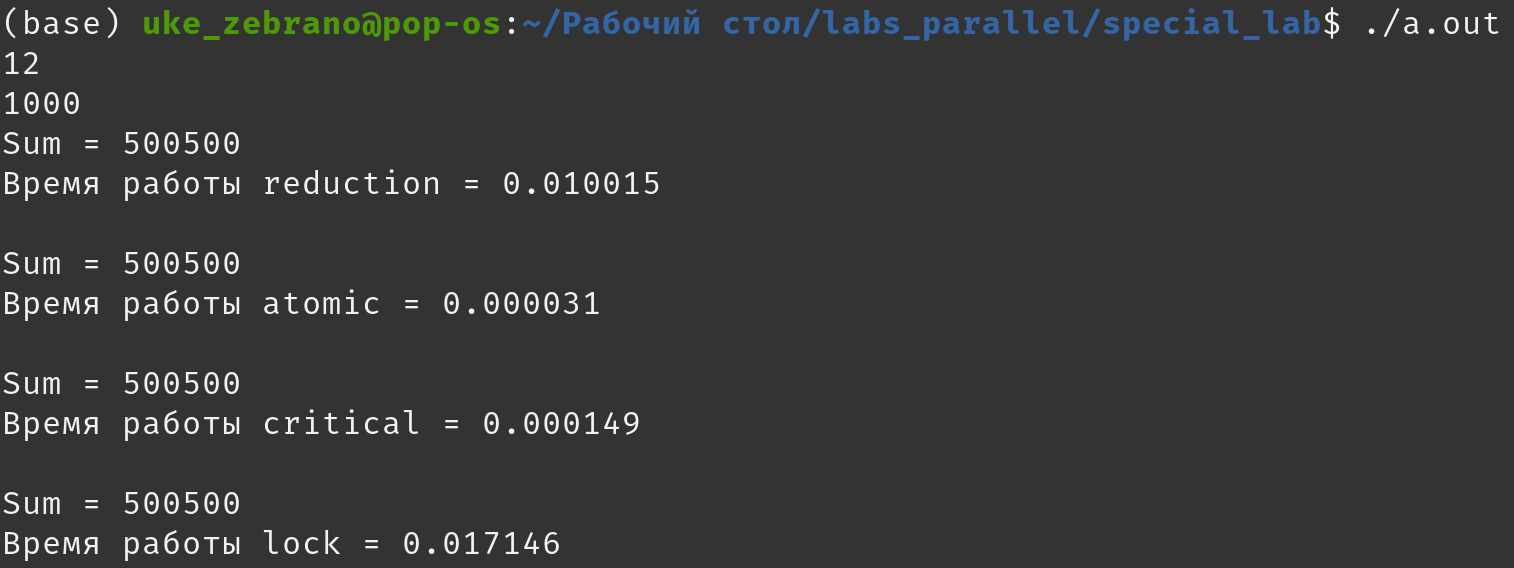
printf("Время работы lock = %f\n\n", end\_time - start\_time);

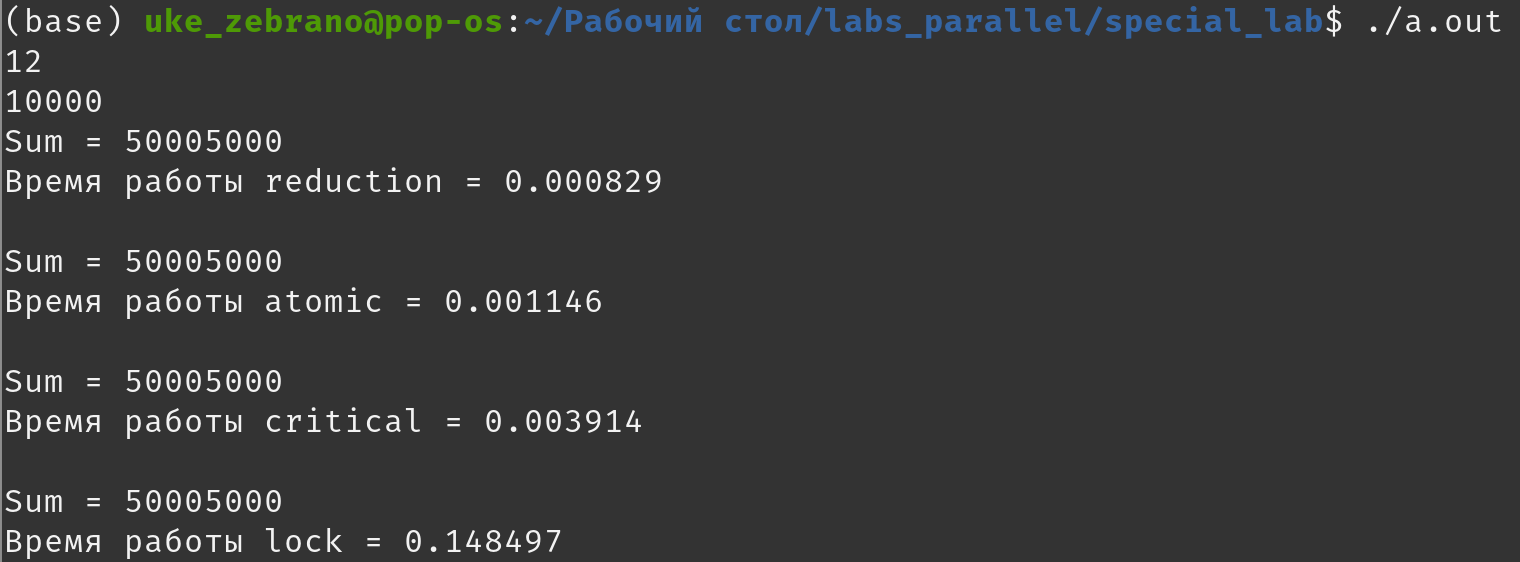
}

Результат замеров:

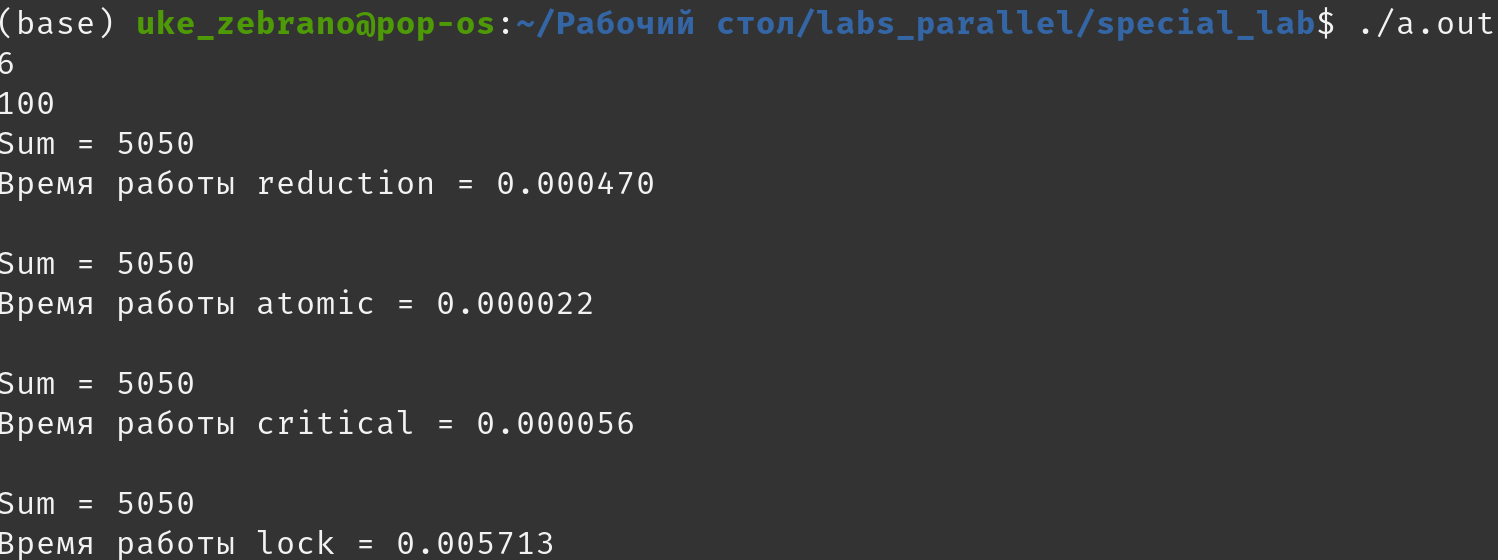
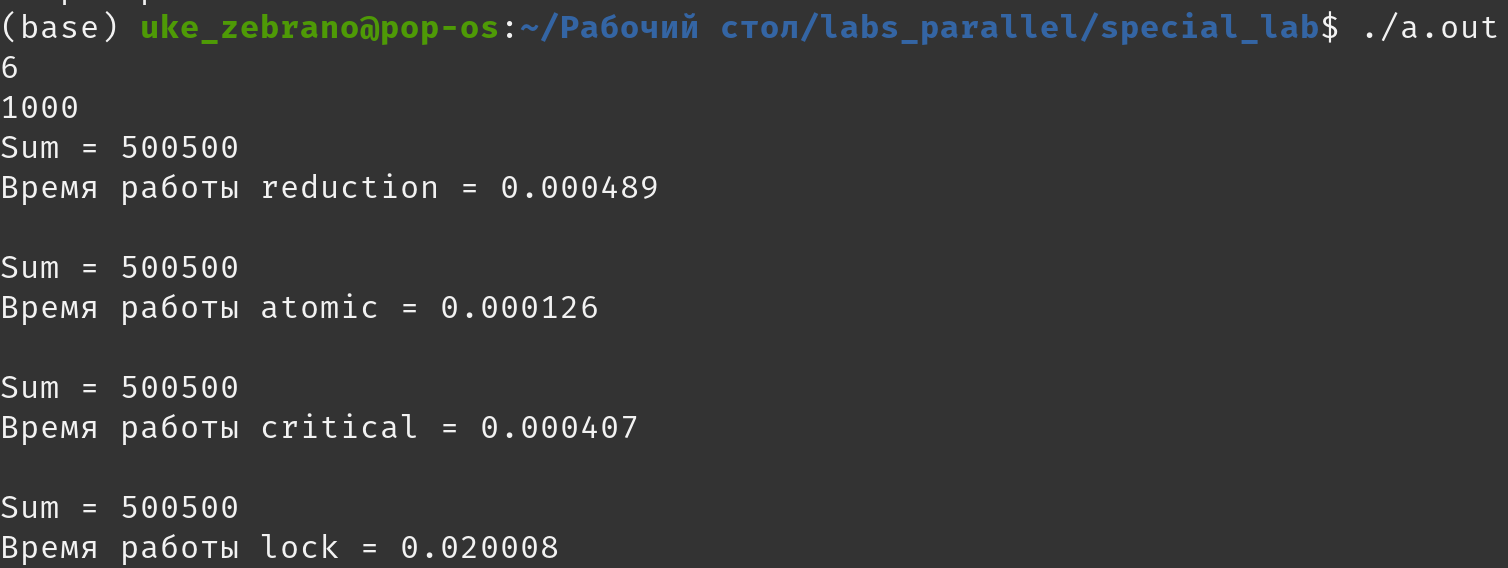
12 потоков, N = 100

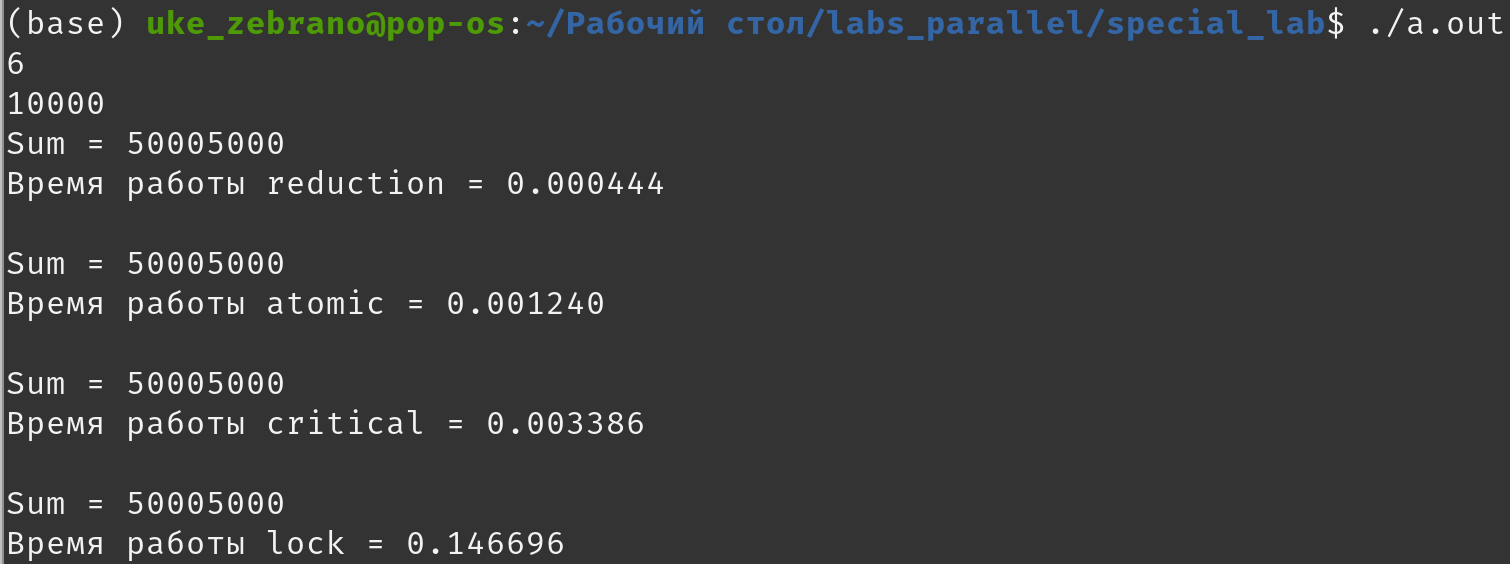
  
12 потоков, N = 1000

  
12 потоков, N = 10000



6 потоков, N = 100

  
6 потоков, N = 1000  
  
 6 потоков, N = 10000

  
По результатам замеров можно сказать, самым лучшим в плане скорости инструментом синхронизации является директива atomic и reduction.

Поэтому её использование можно ставить в приоритет.

(atomic блокирует лишь одну переменную, а не все возможные как critical или lock)

При маленьких значениях лучше использовать atomic, а при больших reduction.  
  
Вывод: Выполнено задание m.volsu.