МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М. В. ЛОМОНОСОВА
ФИЛИАЛ МОСКОВСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
ИМЕНИ М. В. ЛОМОНОСОВА В ГОРОДЕ САРОВЕ

Чернышов Михаил Михайлович
студент 1 курса магистратуры

sor_3d.c

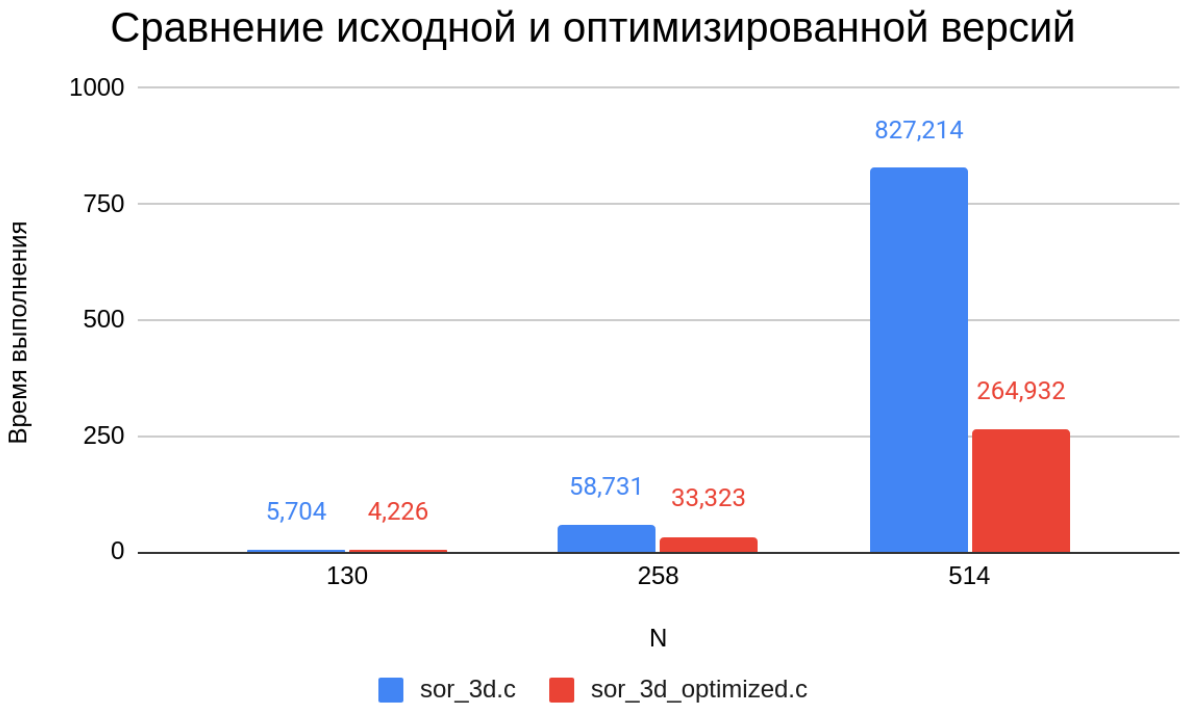
Отчет

Саров – 2022

ОГЛАВЛЕНИЕ

Оптимизация исходной программы

- Добавлен замер времени с помощью omp_get_wtime().
- В relax, init, verify изменен порядок циклов, чтобы обращение происходило к последовательным участкам памяти.
- Вынесено деление на константу из цикла в verify.



OpenMP

init и verify не имеют зависимости по данным, поэтому их можно легко распараллелить, используя прагмы.

```
void init()
{
        #pragma omp parallel for private(i, j, k)
        for(i=0; i<=N-1; i++)
        for(j=0; j<=N-1; j++)
        for(k=0; k<=N-1; k++)
        {
                if(i==0 || i==N-1 || j==0 || j==N-1 || k==0 || k==N-1)
                        A[i][j][k]= 0.;
                else
                        A[i][j][k] = (4 + i + j + k);
        }
}
```

В verify необходимо также использовать редукцию суммы по s.

```
void verify()
{
        double s = 0;

        #pragma omp parallel for private(i, j, k) reduction(+ : s)
        for(i=0; i<=N-1; i++)
        for(j=0; j<=N-1; j++)
        for(k=0; k<=N-1; k++)
        {
                s += A[i][j][k]*(i+1)*(j+1)*(k+1);
        }

        s /= (N*N*N);

        printf("  S = %f\n",s);
}
```

С relax немного сложнее. Тройной цикл имеет прямые и обратные зависимости по данным по каждой из осей матрицы.

sor_3d_openmp_diag_2d.c

Можно рассматривать этот трехмерный цикл, как выполнение процедуры для $N-2$ двумерных матриц. Таким образом, можно параллельно обходить двумерную матрицу по диагонали. Это не даст большого прироста производительности из-за того, что кэш будет использоваться неэффективно. Поэтому следует ходить по диагонали с удлинением по оси хранения данных в памяти.

Нужно выбирать удлинение таким образом, чтобы каждая нить выполняла как можно большую работу и все нити были загружены. Поэтому удлинение можно выбрать так

$$STEP = \frac{N-2}{threads\_num},$$

$STEP$ должно получаться целочисленным.

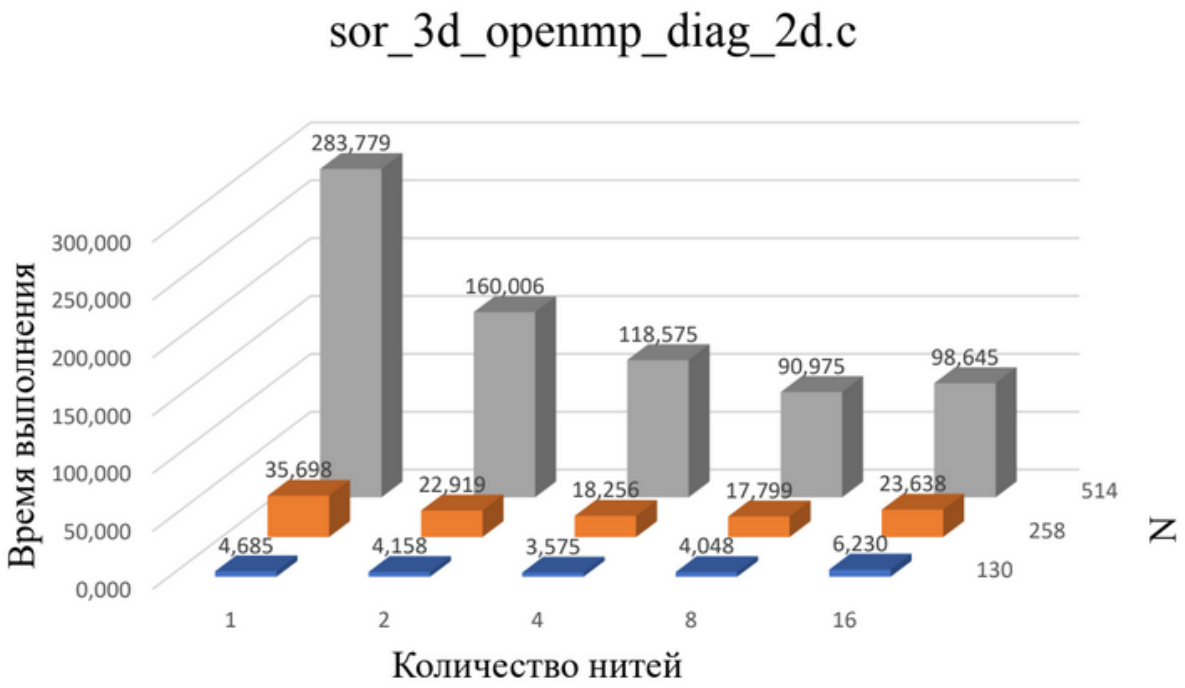
На операции, выполняющиеся последовательно, благодаря удлинению, целесообразно применить векторизацию с помощью

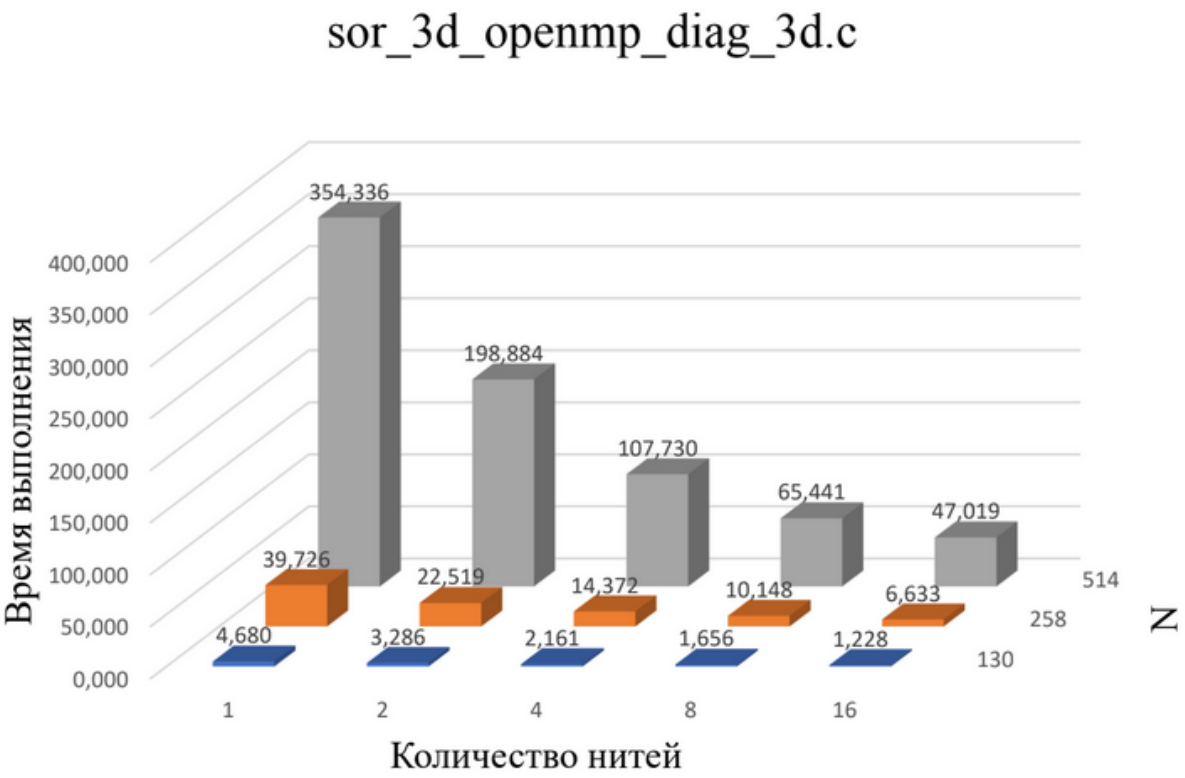
#pragma omp simd

sor_3d_openmp_diag_3d.c

Трехмерную матрицу можно обходить по трехмерной диагонали. Для большей эффективности следует обходить с удлинением по оси хранения данных. Это довольно сложно изобразить, но идея аналогична предыдущему пункту. $STEP$ выбирается таким же образом.

Сравнение параллельных реализаций



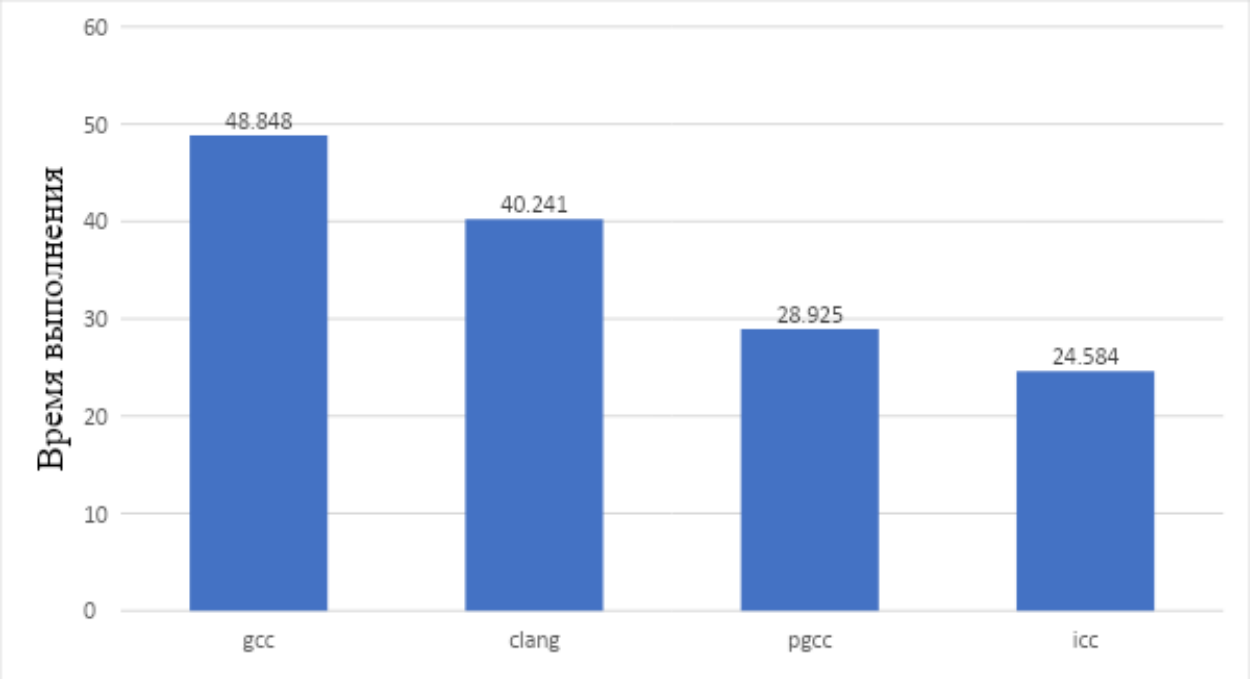


sor_3d_openmp_diag_2d.c имеет спад эффективности при увеличении количества нитей из-за накладных расходов на синхронизацию нитей. Во-первых, барьерные синхронизации стоят после обхода каждой диагонали. Во-вторых, такие диагонали надо обходить $N - 2$ раза (количество двумерных матриц). Огромное количество барьеров приводит к спаду производительности.

sor_3d_openmp_diag_3d.c реализована довольно неплохо. Обходятся трехмерные диагонали с удлинением по оси хранения данных. Барьеров сильно меньше, чем в sor_3d_openmp_diag_2d.c.

Сравнение при использовании различных компиляторов

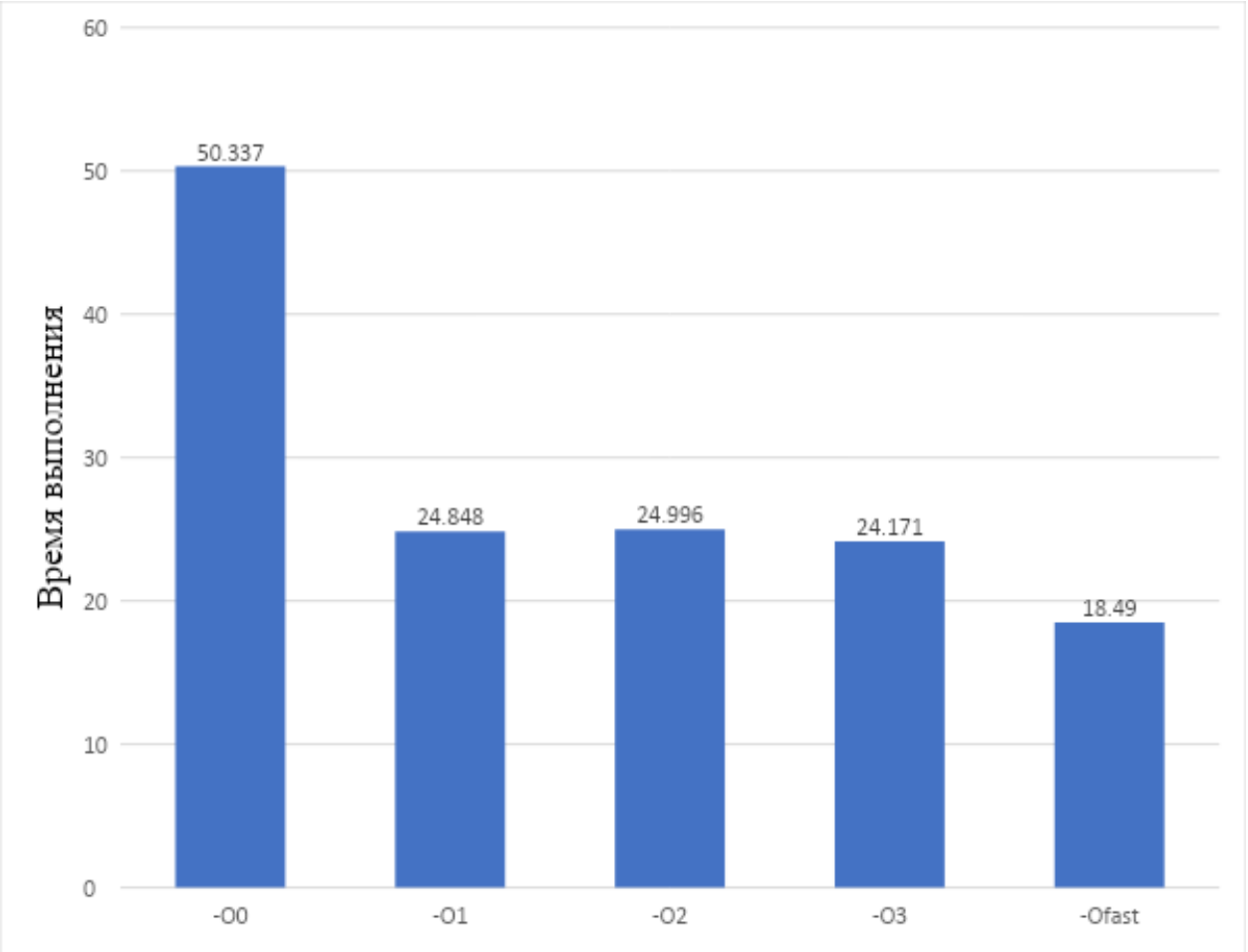
Рассмотрим sor_3d_openmp_diag_3d.c с $N = 514$.



Исследование опций оптимизации компилятора gcc

- ➢-O0 – сокращает время компиляции и заставляет отладку давать ожидаемые результаты. Это значение по умолчанию.
- ➢-O или -O1 – оптимизирует программу. Компиляции занимает несколько больше времени и намного больше памяти.
- ➢-O2 – оптимизирует еще больше. GCC выполняет почти все поддерживаемые оптимизации.
- ➢-O3 – оптимизирует еще больше.
- ➢-Ofast – включает в себя оптимизации, которые не разрешены стандартом.

Рассмотрим sor_3d_openmp_diag_3d.c с $N = 514$.



DVM

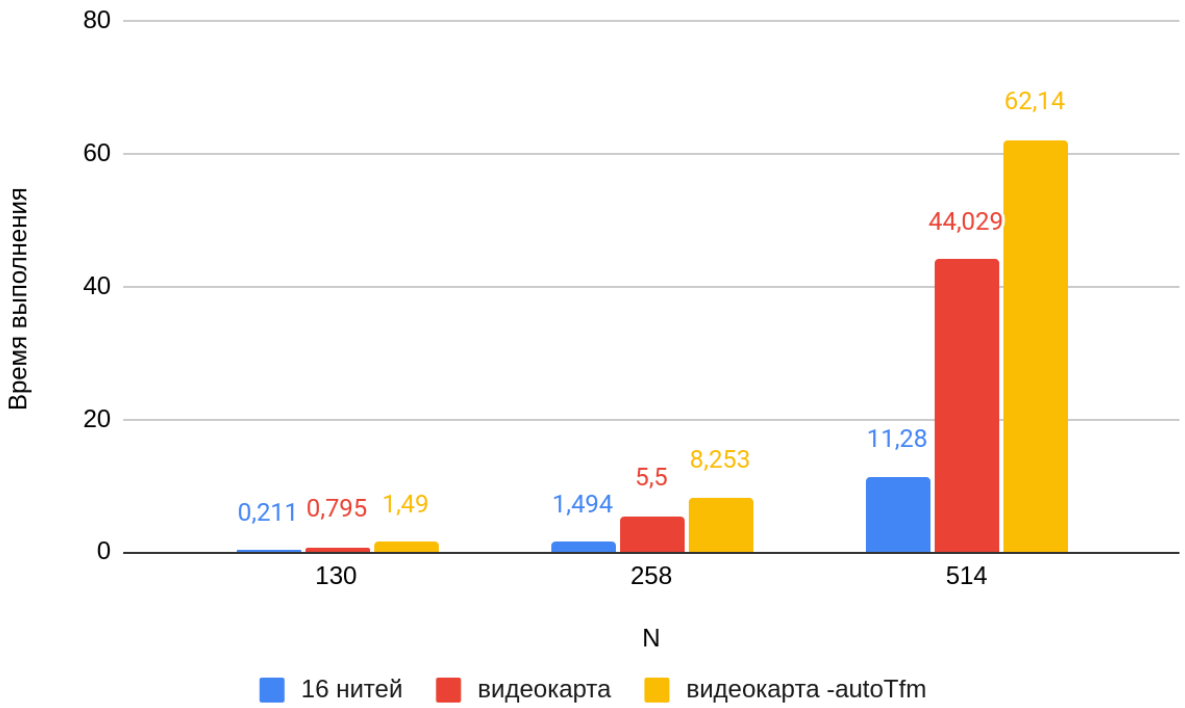
Для распараллеливания применяются следующие DVMH-директивы:

```
void relax()
{
	#pragma dvm region
	{
		#pragma dvm parallel ([i][j][k]) tie(A[i][j][k]) across(A[1:1][1:1][1:1]) reduction(max(eps))
		for(i=1; i<=N-2; i++)
		for(j=1; j<=N-2; j++)
		for(k=1; k<=N-2; k++)
		{
			double e = A[i][j][k];

			A[i][j][k]=(A[i-1][j][k]+A[i+1][j][k]+A[i][j-1][k]+A[i][j+1][k]+A[i][j][k-1]+A[i][j][k+1])/6.;

			eps=Max(eps, fabs(e-A[i][j][k]));
		}
	}
}
```

Также необходимо получать с видеокарты значение эпсилон на каждой итерации для контроля сходимости метода. Для проверки корректности распараллеливания (verify) матрицу необходимо получить с видеокарты.

```
int main(int an, char **as)
{
	int it;

	init();

	double start_time = dvtime_();

	for(it=1; it<=itmax; it++)
	{
		eps = 0.;
		#pragma dvm actual(eps)

		relax();

		#pragma dvm get_actual(eps)
		printf( "it=%4i   eps=%f\n", it,eps);

		if (eps < maxeps) break;
	}

	printf("Elapsed time = %3lf\n", dvtime_() - start_time);

	#pragma dvm get_actual(A)

	verify();

	return 0;
}
```

Сравнение



Вычисления на видеокарте занимают больше времени из-за частых обращений к памяти. На многих GPU обращение к памяти выполняется заметно медленнее, чем выполнение одной арифметической операции.

Приложение

Программы доступны в github-репозитории:

- https://github.com/Disfavour/OpenMP_2022

- sor_3d.c – начальная версия программы.
- sor_3d_optimized.c – оптимизированная последовательная версия.
- sor_3d_openmp_diag_2d.c – параллельная версия по двумерным диагоналям.
- sor_3d_openmp_diag_3d.c – параллельная версия по трехмерным диагоналям.
- sor_3d_dvm.c – параллельная версия с использованием dvm.