DPC++
Анализ
Тестирование

Летняя международная молодежная Школа-конференция по параллельному программированию

Содержание доклада

01

ВВЕДЕНИЕ

Общая информация о проекте

02

ВОЛНОВОЕ УРАВНЕНИЕ

- Постановка задачи
- Реализация
- Анализ

03

ЗАКЛЮЧЕНИЕ

Выводы по проделанной работе

Архитектур **МНОГО**,

API один

-- принцип oneAPI --

01

OneAPI и DPC++

- Архитектур много , API (application programming interface) один.
- Задача oneApi – обеспечить простоту использования и эффективность работы на разных архитектурах (CPU, GPU, FPGA, …).
- DPC++ – новый язык программирования, дающий возможности написания производительного кода под разные архитектуры, имеющий для разработчиков привычную модель, включает в себя SYCL (модель программирования, основанная на OpenCL) и расширения от intel.
- Наша задача – изучить и проанализировать возможности DPC++ и сравнить с другими средствами распараллеливания программ.

01

Постановка задачи

- Изучить средство программирования DPC++
- Сравнить производительность DPC++ с другими средствами на тестовой задаче
- Провести анализ производительности

02

2D Волновое уравнение

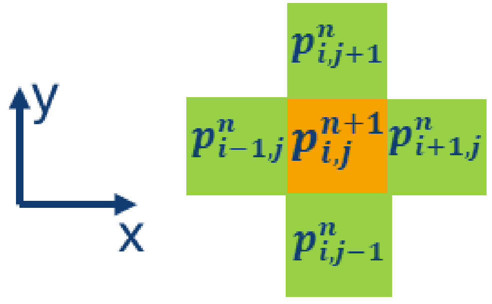
Решение уравнения , описывающего распространение волн в среде.

$$\frac{\partial^2 p}{\delta t^2} = v^2\left(\frac{\partial^2 p}{\delta x^2} + \frac{\partial^2 p}{\delta y^2}\right)$$

p – давление , v – скорость среды; x,y – декартовы координаты, t - время

Разностная схема

$$p_{i,j}^{n+1} = 2p_{i,j}^{n} - p_{i,j}^{n-1} + v^2\Delta t^2\left(\frac{p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}}{\Delta x^2} + v^2\frac{p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}}{\Delta y^2}\right)$$



02

Свойства задачи

- Вычисления в каждой точке пространства независимые на каждом шаге моделирования.
- Способ распараллеливания – пространственная декомпозиция.
- Можно использовать все виды параллелизма в оборудовании (по ядрам, векторизация).

02

DPC++ vs OpenMP

- Для сравнения было выбрано такое средство распараллеливания как OpenMP, имеющее схожую с DPC++ модель памяти.
- Были проведены тесты производительности на разных данных.
  - ❖DPC++ : компилятор на основе clang++ (dpcpp).
  - ❖OpenMP : компилятор icpc (Intel).
- Device: Intel® Xeon® CPU E5-2620 v4 @ 2.10Ghz (8-ми ядерный).

02

Реализация алгоритма

OpenMP

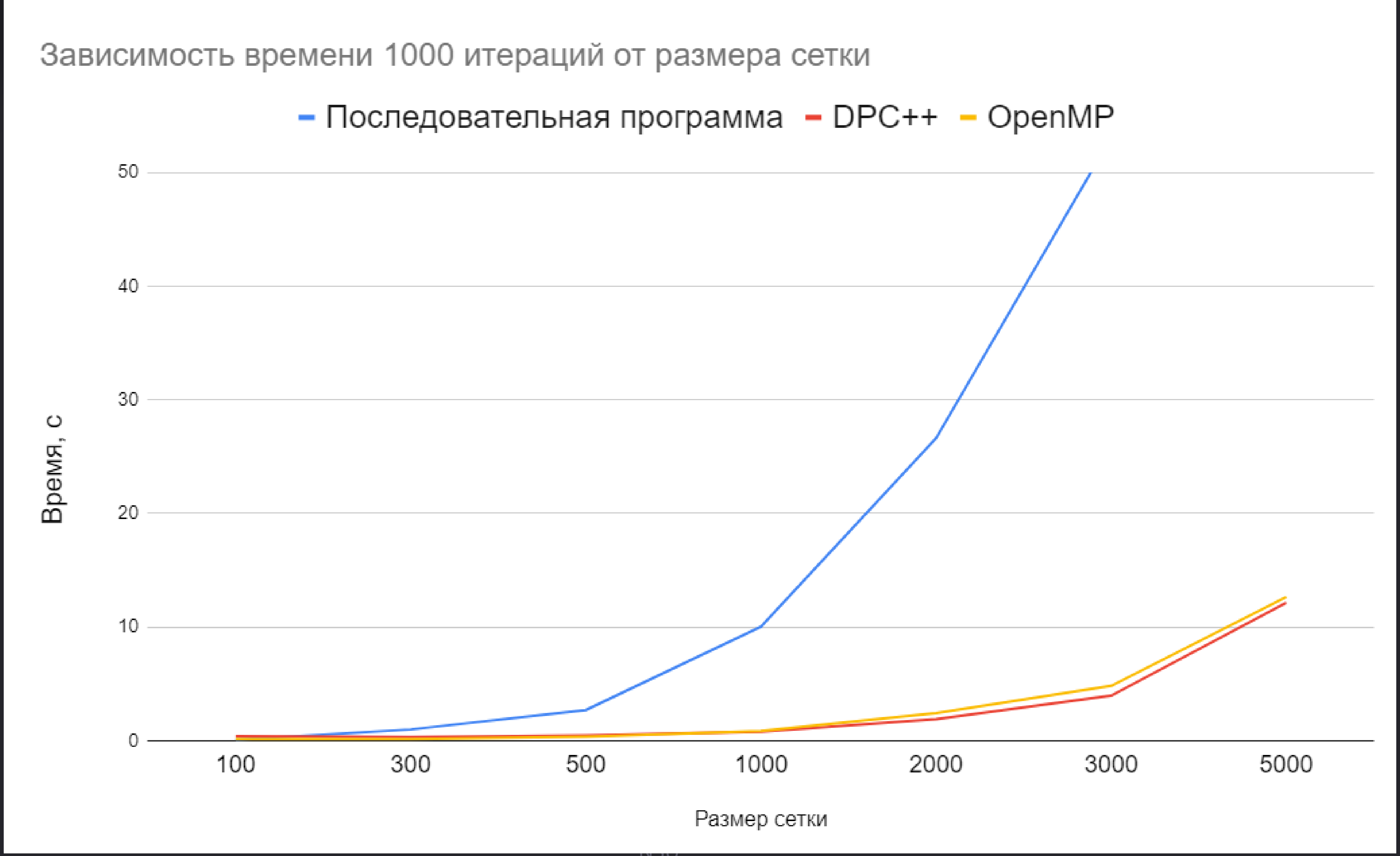
```
void iso_2dfd_iteration_omp(float *next, float *prev, float *vel,
                            const float dtDIVdxy, int nRows, int nCols,
                            int nIterations) {
    float *swap;
    for (unsigned int k = 0; k < nIterations; k += 1) {
#pragma omp parallel for
        for (unsigned int i = 1; i < nRows - HALF_LENGTH; i += 1) {
#pragma omp simd
            for (unsigned int j = 1; j < nCols - HALF_LENGTH; j += 1) {
                int gid = j + (i * nCols);
                float value = 0.0f;
                value += prev[gid + 1] - 2.0f * prev[gid] + prev[gid - 1];
                value += prev[gid + nCols] - 2.0f * prev[gid] + prev[gid - nCols];
                value *= dtDIVdxy * vel[gid];
                next[gid] = 2.0f * prev[gid] - next[gid] + value;
            }
        }
        swap = next;
        next = prev;
        prev = swap;
    }
}
```

DPC++

```
buffer<float, 1> b_next(next_base, range<1>{nsize});
buffer<float, 1> b_prev(prev_base, range<1>{nsize});
buffer<float, 1> b_vel(vel_base, range<1>{nsize});
        for (unsigned int k = 0; k < nIterations; k += 1) {
            q.submit([&](sycl::handler &h) {
                auto next = b_next.get_access<access::mode::read_write>(h);
                auto prev = b_prev.get_access<access::mode::read_write>(h);
                auto vel = b_vel.get_access<access::mode::read>(h);
                auto global_range = range<2>(nRows, nCols);
                if (k % 2 == 0)
                    h.parallel_for(global_range, [=](id<2> it) {
                    iso_2dfd_iteration_global(it, next.get_pointer(),
                        prev.get_pointer(), vel.get_pointer(),
                        dtDIVdxy, nRows, nCols);
                });
                else
                    h.parallel_for(global_range, [=](id<2> it) {
                    iso_2dfd_iteration_global(it, prev.get_pointer(),
                        next.get_pointer(), vel.get_pointer(),
                        dtDIVdxy, nRows, nCols);
                });
            });
        }
```

02

Замеры времени



02

Анализ производительности: векторизация (Intel® Advisor)

OpenMP

Program metrics

Elapsed Time	31.42s	GFLOPS	23.85
Vector Instruction Set	AVX	GFLOP Count	749.490
Number of CPU Threads	8	FP Arithmetic Intensity	0.312
		GINTOPS	1.51

Performance characteristics

Metrics	*Total*	
Total CPU time	242.21s	100%
Time in **2** vectorized loops	209.11s	86.3%
Time in scalar code	33.10s	13.7%

DPC++

Program metrics

Elapsed Time	61.04s	GFLOPS	14.75
Vector Instruction Set	AVX, SSE2, SSE	GFLOP Count	900.005
Number of CPU Threads	8	FP Arithmetic Intensity	0.313
		GINTOPS	8.62

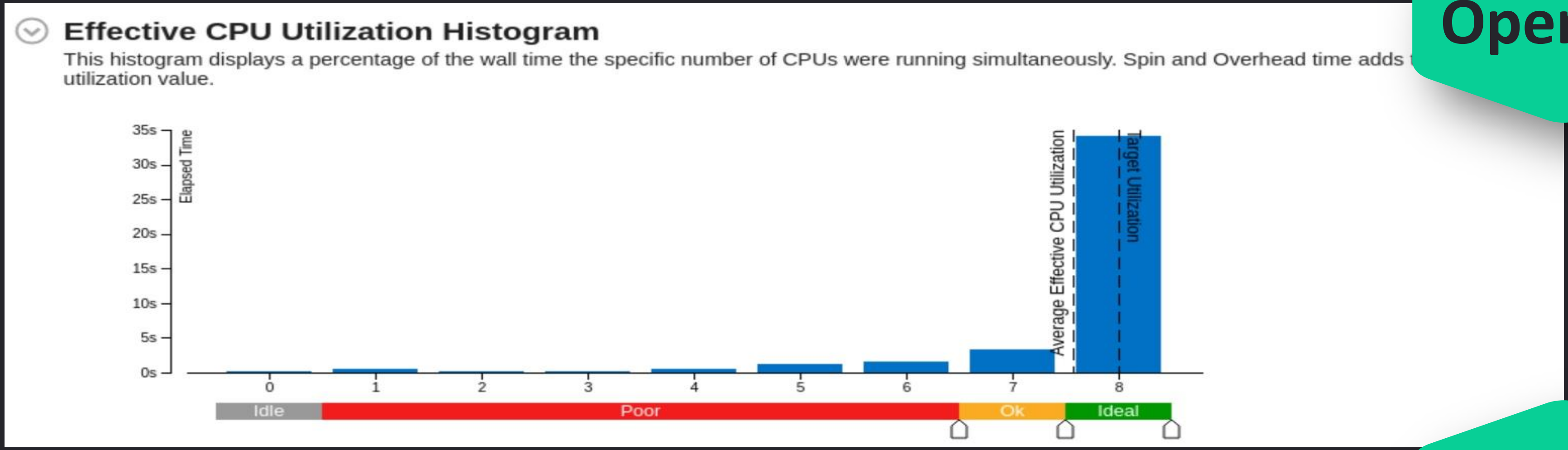
Performance characteristics

Metrics	*Total*	
Total CPU time	456.27s	100%
Time in **9** vectorized loops	433.67s	95%
Time in scalar code	22.60s	5%

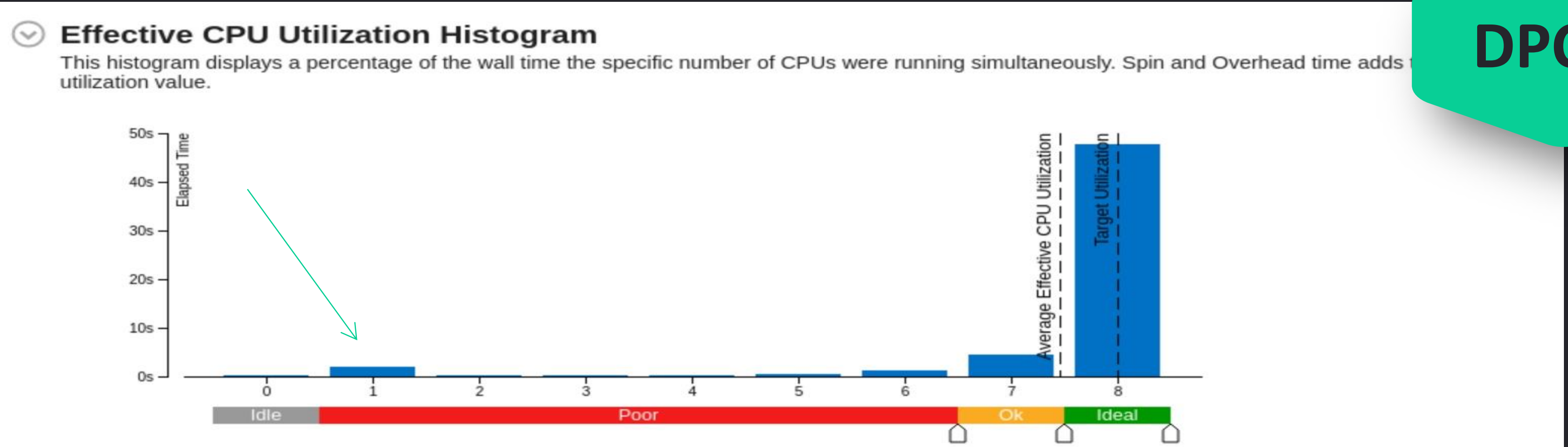
02

Анализ производительности: Параллелизм по ядрам(Intel® VTune)

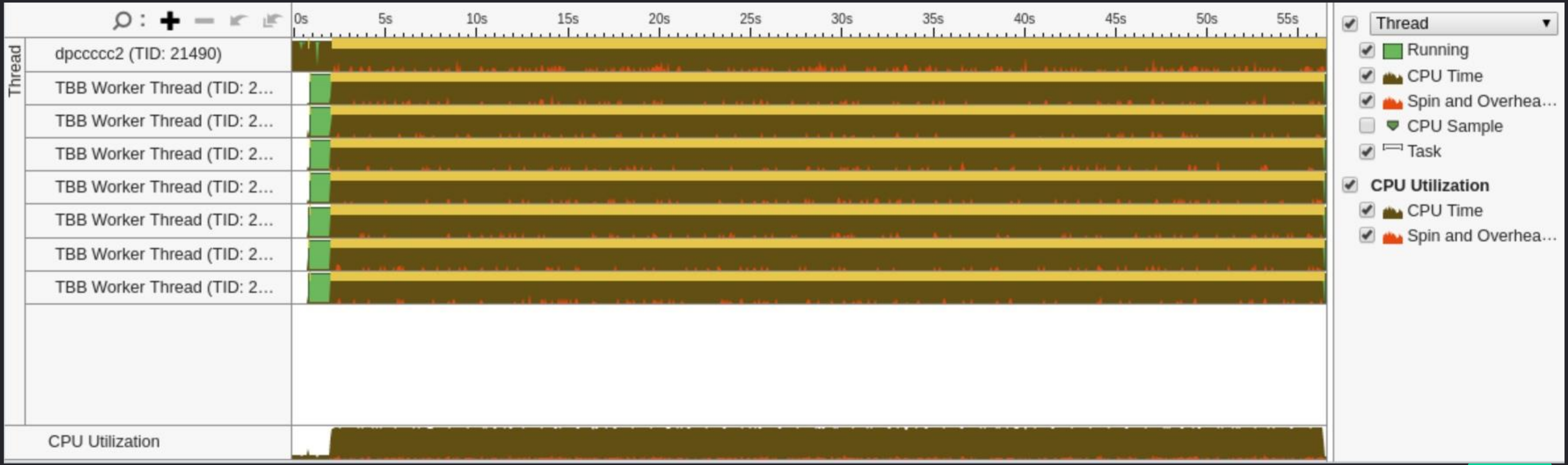
OpenMP



DPC++

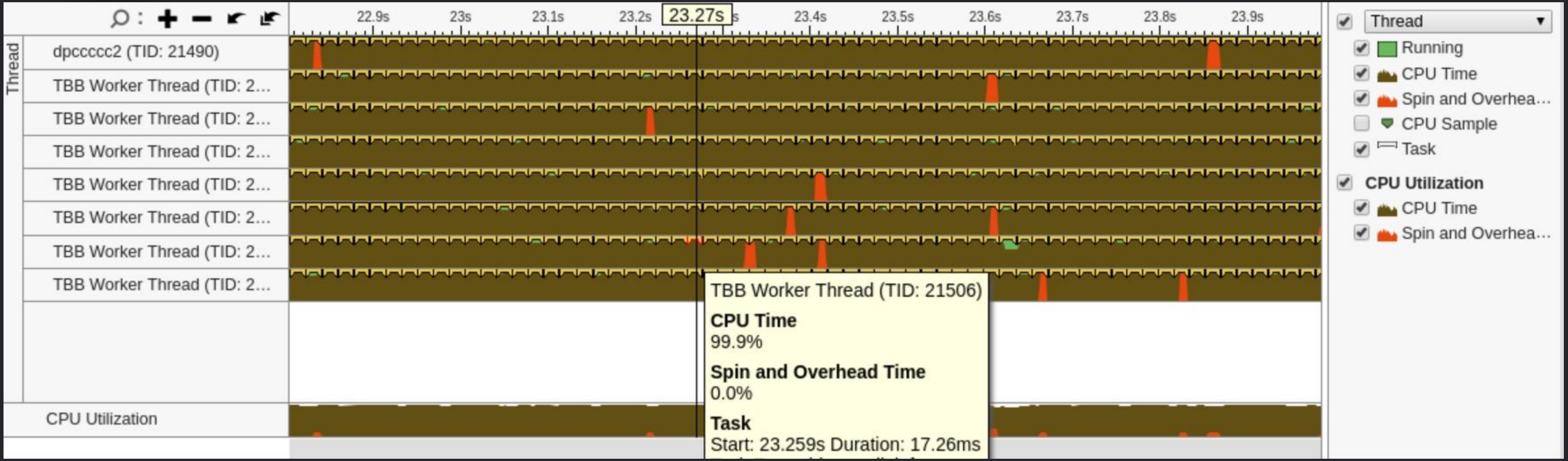


02 Профилирование DPC++



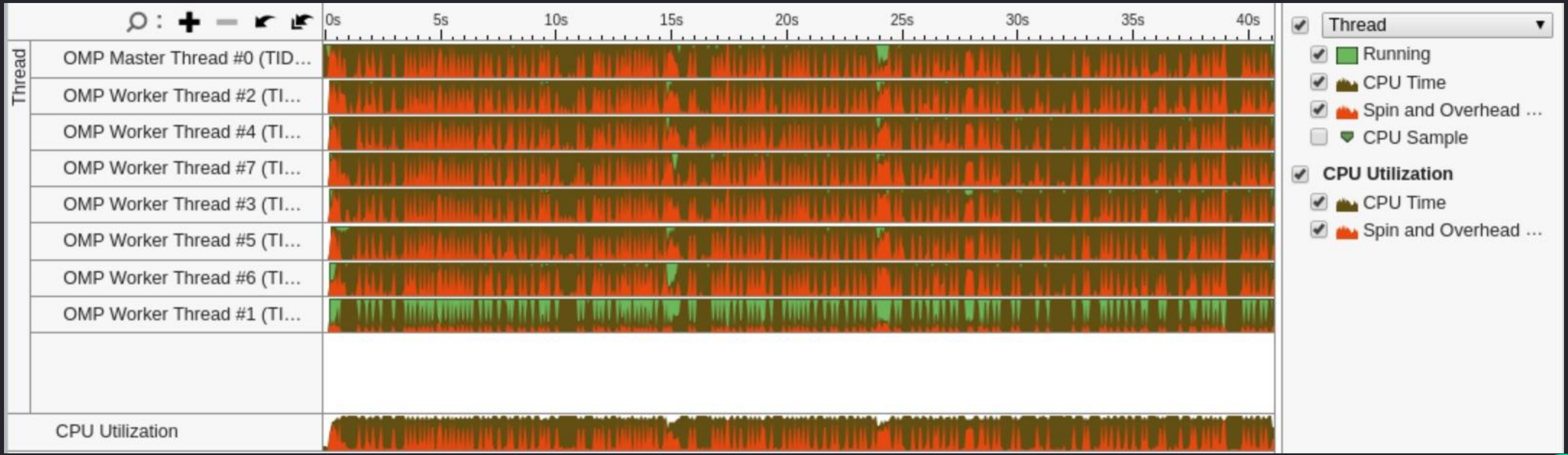
02

Профилирование DPC++



03

Профилирование OpenMP



03

Заключение

Что сделано?

- Познакомился со средством параллельного программирования DPC++.
- Проведено сравнение производительности DPC++ и OpenMP на задаче решения 2D волнового уравнения.
- Выполнен анализ производительности с помощью средств Intel.

Какие выводы?

- В тестовой программе на DPC++ без дополнительных усилий удалось добиться использования всех видов параллелизм а в CPU.

Спасибо за внимание

Вопросы?

BARANOV ILYA

STUDENT

i.baranov@g.nsu.ru

https://vk.com/awesomexdx