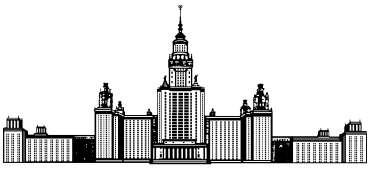

Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра Суперкомпьютеров и Квантовой Информатики

Исследование проблемы масштабируемости в OpenMP, вызванной неявной синхронизацией нитей

Выполнил:

студент 423 группы,

Имашев Владислав Родиславович

Москва, 2023

Содержание

1 Описание сути проблемы

В программе, написанной с использовании технологии openMP, в конце параллельных блоков/циклов/участков происходит неявная барьерная синхронизация параллельно работающих нитей: их дальнейшее выполнение происходит только тогда, когда все они достигнут данной точки. Если в подобной задержке нет необходимости, то опция nowait позволяет нитям уже дошедшим до конца параллельного участка продолжить выполнение без синхронизации с остальными. Если данная опция в явном виде и не указана, то в конце параллельного цикла синхронизация все равно будет выполнена.

2 Модельная задача

2.1 Постановка задачи

Необходимо посчитать выход линейного слоя нейронной сети и определить среднюю активацию нейронов на выходе:

$$output = sigmoid(W \cdot input + bias),$$

где $sigmoid(\cdot)$ – функция активации сигмоида,
$W \in R^{n \times m}$ – матрица весов линейного слоя, $bias$ – смещение,
$input \in R^m$ – входной вектор слоя,
$output \in R^n$ – выходной вектор (активация слоя).

Программная реализация представляет из себя параллельный алгоритм подсчета умножения матрицы весов на вектор (распараллеливается внутренний цикл), для корректной работы каждая нить записывает свой "вклад"в результат в локальную переменную, затем в критической секции происодит формирование результата умножения. Далее параллельно происходит подсчёт функции активации от результата умножения с поправкой на смещение. Следом в критической секции выполяется подсчет средней активации нейронов на выходе.

2.2 Неоптимизированный вариант программы

В неоптимизированном варианте описанные этапы выполняются "последовательно"из-за наличия барьерных синхронизаций после параллельных циклов.

```
#pragma omp parallel private(local_result)
{
        for (int i = 0; i < n; ++i)
        {
                local_result = 0.0;
                #pragma omp for
                for (int j = 0; j < m; ++j)
                {
                        local_result += *(W + i * m + j) * *(b + j);
                }
                #pragma omp critical
```

```
                {
                        *(c + i) += local_result;
                }
        }

        #pragma omp for
        for (int j = 0; j < n; ++j)
        {
                *(c + j) = sigmoid(*(c + j) + rand());
                #pragma omp critical
                {
                        mean_activation += *(c + j) / n;
                }
        }
}
```

2.3 Оптимизированный вариант программы

Оптимизированный вариант программы отличается от исходного добавлением опции nowait в директиву распараллеливания внутреннего цикла при подсчете умножения матрицы на входной вектор. При добавлении данной опции не будет происходить барьерной синхронизации нитей на каждой итерации,а поэтому первые освободившиеся нити смогут зайти сразу в критическую секцию, не дожидаясь остальных, и продолжить работу.

```
#pragma omp parallel private(local_result)
{
        for (int i = 0; i < n; ++i)
        {
                local_result = 0.0;
                #pragma omp for nowait
                for (int j = 0; j < m; ++j)
                {
                        local_result += *(W + i * m + j) * *(b + j);
                }
                #pragma omp critical
                {
```

```
                        *(c + i) += local_result;
                }
        }

        #pragma omp for
        for (int j = 0; j < n; ++j)
        {
                *(c + j) = sigmoid(*(c + j) + rand());
                #pragma omp critical
                {
                        mean_activation += *(c + j) / n;
                }
        }
}
```

3 Сравнение двух реализаций

3.1 Метод сравнения

Производилось сравнение двух реализаций с помощью замера времени работы параллельной секции. Для этого использовалась функция $omp\_get\_wtime()$.

Все запуски производились на вычислительной системе IBM Polus, на 4 узле. Характеристики узла:

- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков) всего
- 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек

Компиляция программ производилась с помощью компилятора gcc:

```
gcc program.c -std=gnu99 -o0 -fopenmp -lm -o program
```

Постановка задач на выполнения производилась с помощью планировщика LSF.

3.2 Результаты сравнения

Ниже представлены результаты работы программ для разных размеров входных данных (для матрицы весов 20000x20000 и 40000x40000):

Число нитей	1	2	4	8	16	32	64
Время неоптимиз. (сек)	3.292	1.892	0.904	0.524	0.539	0.750	1.257
Время оптимиз. (сек)	3.183	1.605	0.814	0.431	0.316	0.462	1.073
Ускорение неоптимиз.	1.000	1.739	3.641	6.276	6.102	4.386	2.617
Ускорение оптимиз.	1.000	1.983	3.910	7.383	10.001	6.876	2.965
Разница (%)	3.310	15.175	9.959	17.806	40.993	38.316	14.660

Таблица 1: Матрица весов 20000x20000

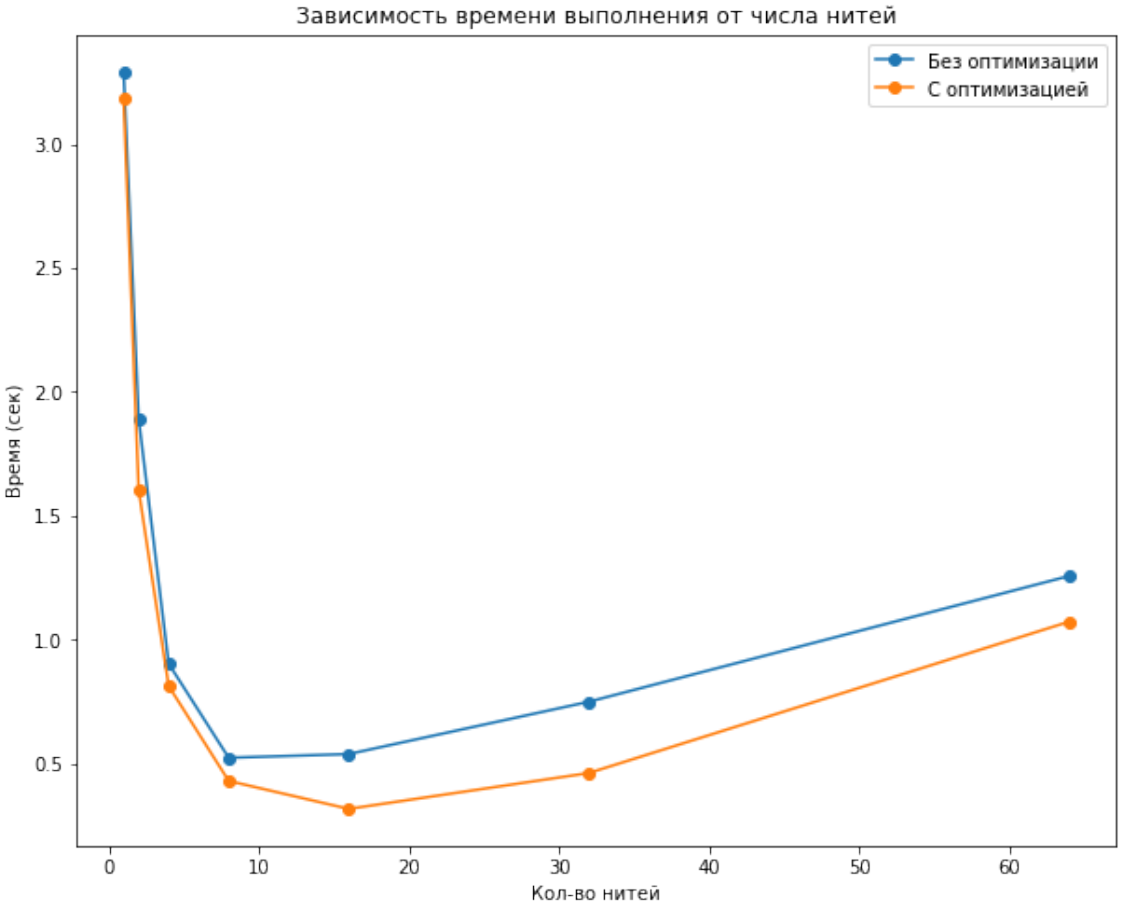

Рис. 1: Матрица весов 20000x20000

Число нитей	1	2	4	8	16	32	64
Время неоптимиз. (сек)	13.655	9.021	3.417	1.955	1.716	1.907	3.242
Время оптимиз. (сек)	13.882	6.598	3.269	1.697	1.047	0.823	2.470
Ускорение неоптимиз.	1.000	1.514	3.996	6.985	7.956	7.162	4.212
Ускорение оптимиз.	1.000	2.104	4.247	8.178	13.261	16.860	5.620
Разница (%)	1.657	26.856	4.340	13.179	39.012	56.814	23.808

Таблица 2: Матрица весов 40000x40000

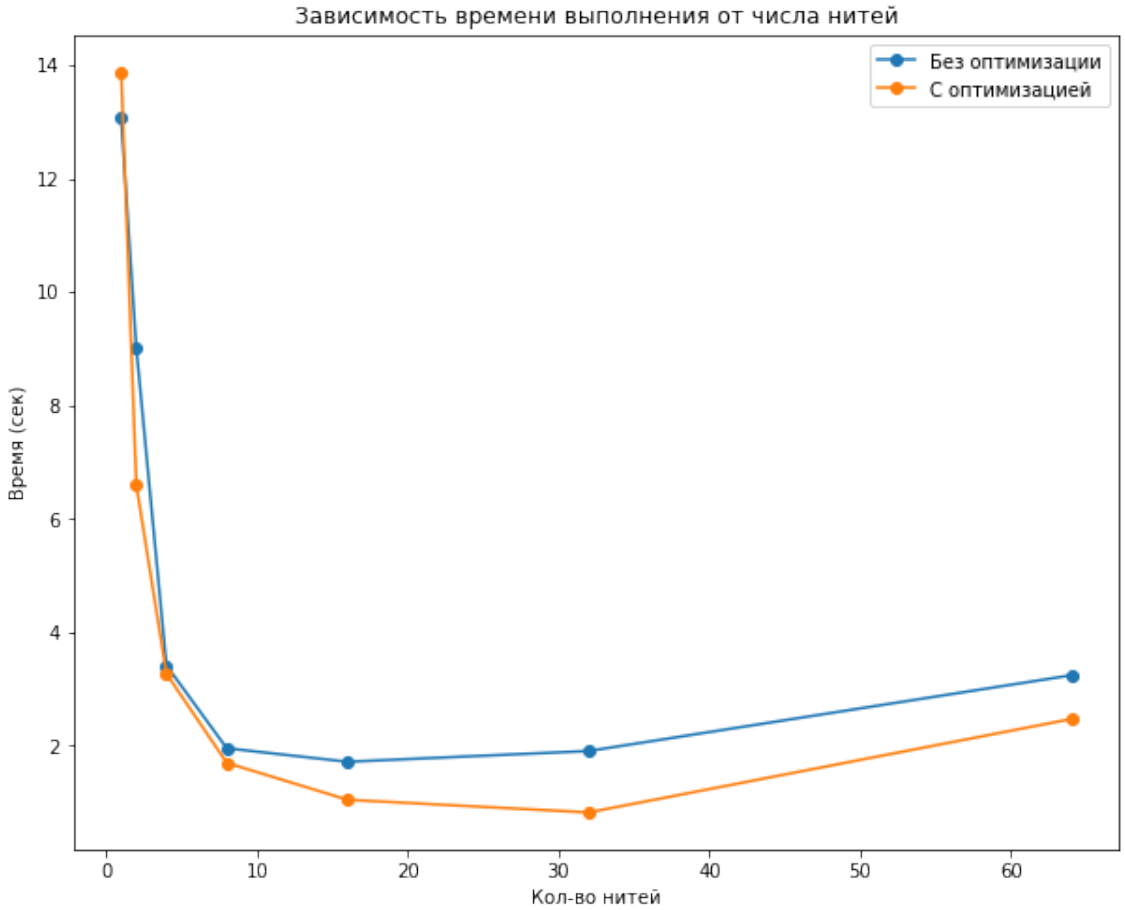

Рис. 2: Матрица весов 40000x40000