Московский государственный университет
имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Практикум по учебному курсу «Суперкомпьютеры и параллельная обработка данных»

Задание

Разработка параллельной версии программы для вычисления определенного интеграла с использованием метода Симпсона

Отчет

о выполненном задании

студента 320 учебной группы факультета ВМК МГУ

Москаленко Андрея Викторовича

Москва, 2019 г.

Содержание

1. Постановка задачи

Ставится задача нахождения численного значения определенного интеграла функции $f(x)$ на отрезке $[a, b]$ с использованием метода Симпсона.

Требуется:

- Реализовать последовательную и параллельную версию программы
- Протестировать полученные программы на суперкомпьютерах Polus и Blue Gene
- Сравнить эффективность, проанализировать масштабируемость

2. Описание метода

Для удобства разобьём отрезок $[a, b]$ на $N = 2n$ элементарных отрезков. Тогда значение исходного интеграла является суммой результатов интегрирования на этих отрезках:

$$\int_a^b f(x) \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n-1} f\left(x_{2j}\right) + 4\sum_{j=1}^{n} f\left(x_{2j-1}\right) + f(x_N)\right]$$

где $h = \frac{b-a}{N}$, $x_j = a + jh$

Полученную формулу можно переписать в виде:

$$\int_a^b f(x) \approx \frac{h}{3}\sum_{k=1,2}^{N-1} [f(x_{k-1}) + 4f(x_k) + f(x_{k+1})]$$

где запись $k = 1,2$ означает, что k меняется от единицы с шагом 2

3. Реализация

Были реализованы три версии программы:

1) Наивная реализация последовательной программы (baseline)
2) Последовательная версия
3) Параллельная версия с использованием OpenMP

3.1 Последовательная версия (наивная реализация)

```
double integral(double (*f)(double), const double a, const double b,
                const long long n) {
    double h = (b - a) / (2 * n);
    double sum = 0.0;
    long long N = 2 * n;

    for (long long k = 1; k <= N - 1; k += 2) {
        double x_k = a + k * h;
        sum += f(x_k - h)  + 4 * f(x_k)+ f(x_k + h);
    }
    return sum * h / 3;
}
```

3.2 Последовательная и параллельная версии (отличия выделены цветом)

```
double integral(double (*f)(double), const double a, const double b,
                const long long n) {
    double h = (b - a) / (2 * n);
    double sum = 0.0;
    long long N = 2 * n;
    double sum1 = 0.0;
    double sum2 = 0.0;

    #pragma omp parallel for reduction(+:sum1) reduction(+:sum2)
    for (long long j = 1; j < N; j += 2) {
        sum1 += f(a + j * h);
        sum2 += f(a + (j + 1) * h);
    }
    #минус f(b) т.к мы её добавили дважды в sum2
    sum += f(a) - f(b) + 4 * sum1 + 2 * sum2;
    return sum * h / 3;
}
```

С полным кодом программы можно ознакомиться в репозитории – https://github.com/a-mos/skpod

4. Тестирование

Для теста были выбраны следующие функции:

$$f_1(x) = x^3$$

$$f_2(x) = \frac{1}{1 + e^{-x}}$$

$$f_3(x) = \sqrt{1 - \sin(x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7)}$$

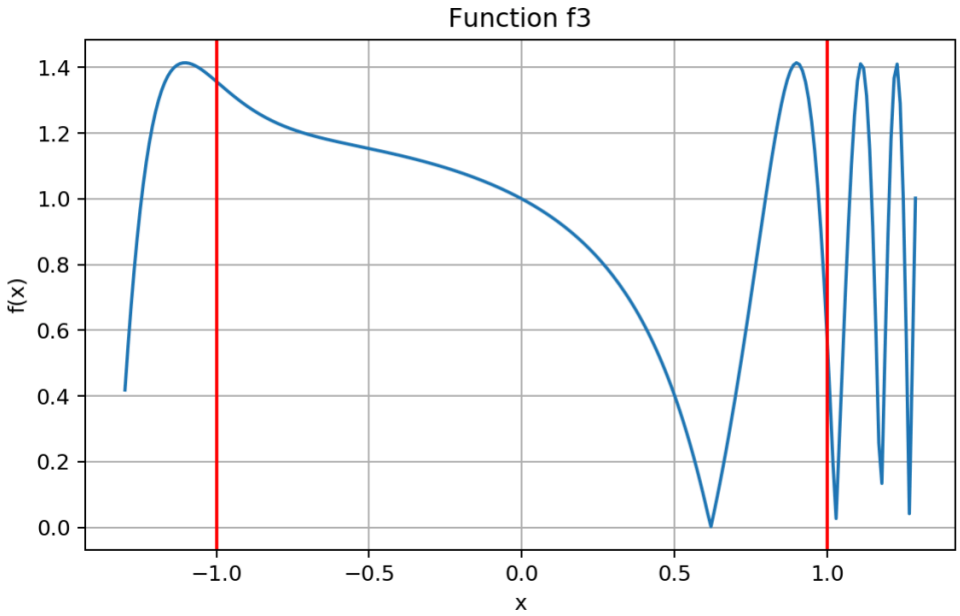
Компиляция производилась с включенной оптимизацией (-O3). Замер времени начинался с момента вызова функции и заканчивался в момент возврата. Тестирование проводилось трижды, затем результаты усреднялись.

4.1 Сравнение используемых функций

	1 threads	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads	128 threads
f1	0.701202	0.370331	0.255652	0.157813	0.135874	0.112560	0.141225	0.210439
f2	3.082734	1.574076	0.848379	0.562202	0.423933	0.289902	0.289148	0.374779
f3	10.535222	5.295680	2.805154	1.520768	1.106129	0.808859	0.707183	0.779625

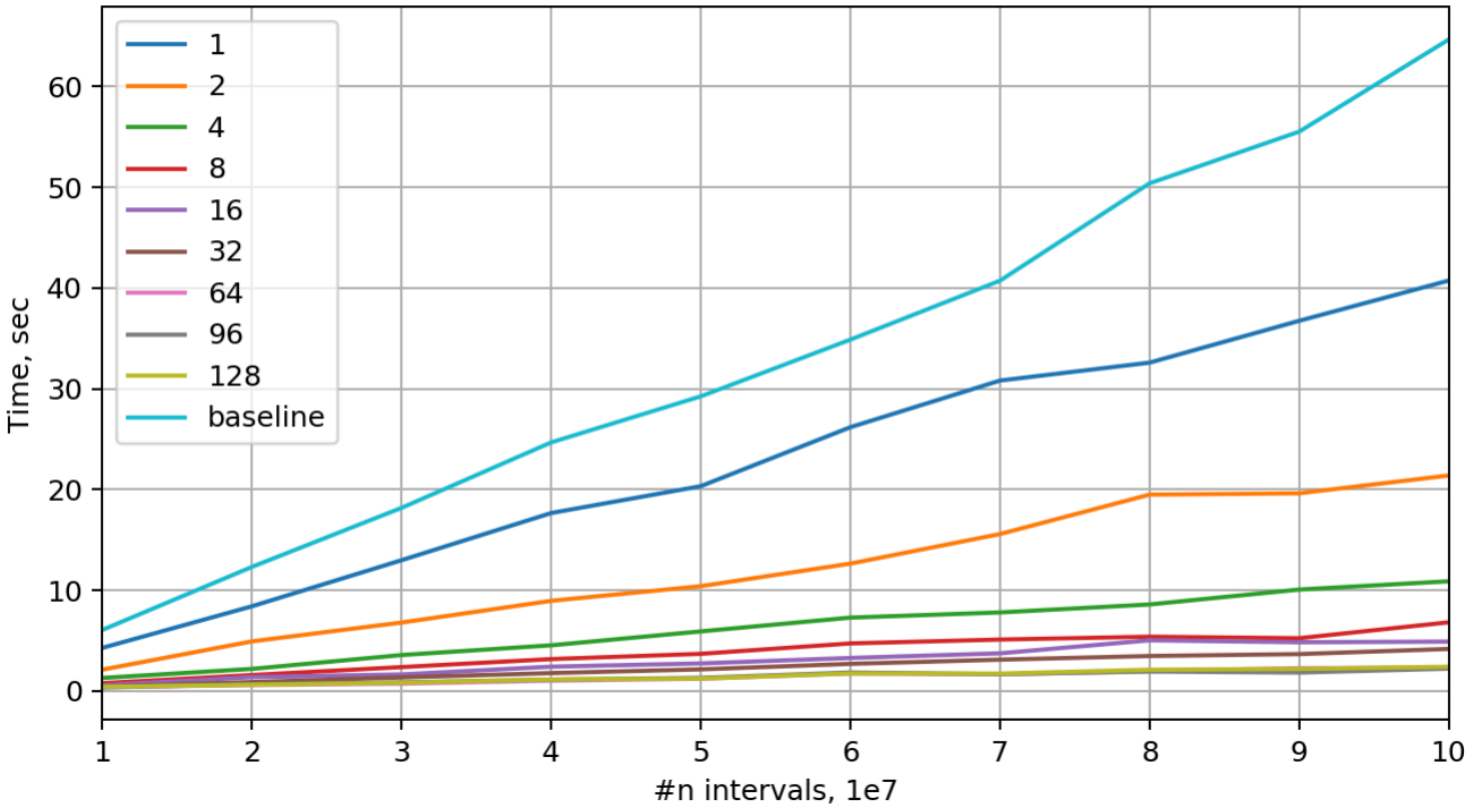
Время исполнения функций в зависимости от числа нитей

Заметим, что изменение функции лишь пропорционально влияет на скорость выполнения программы, поэтому в последующих сравнениях остановимся на интегрировании функции $f_3(x)$ на интервале $[-1, 1]$:

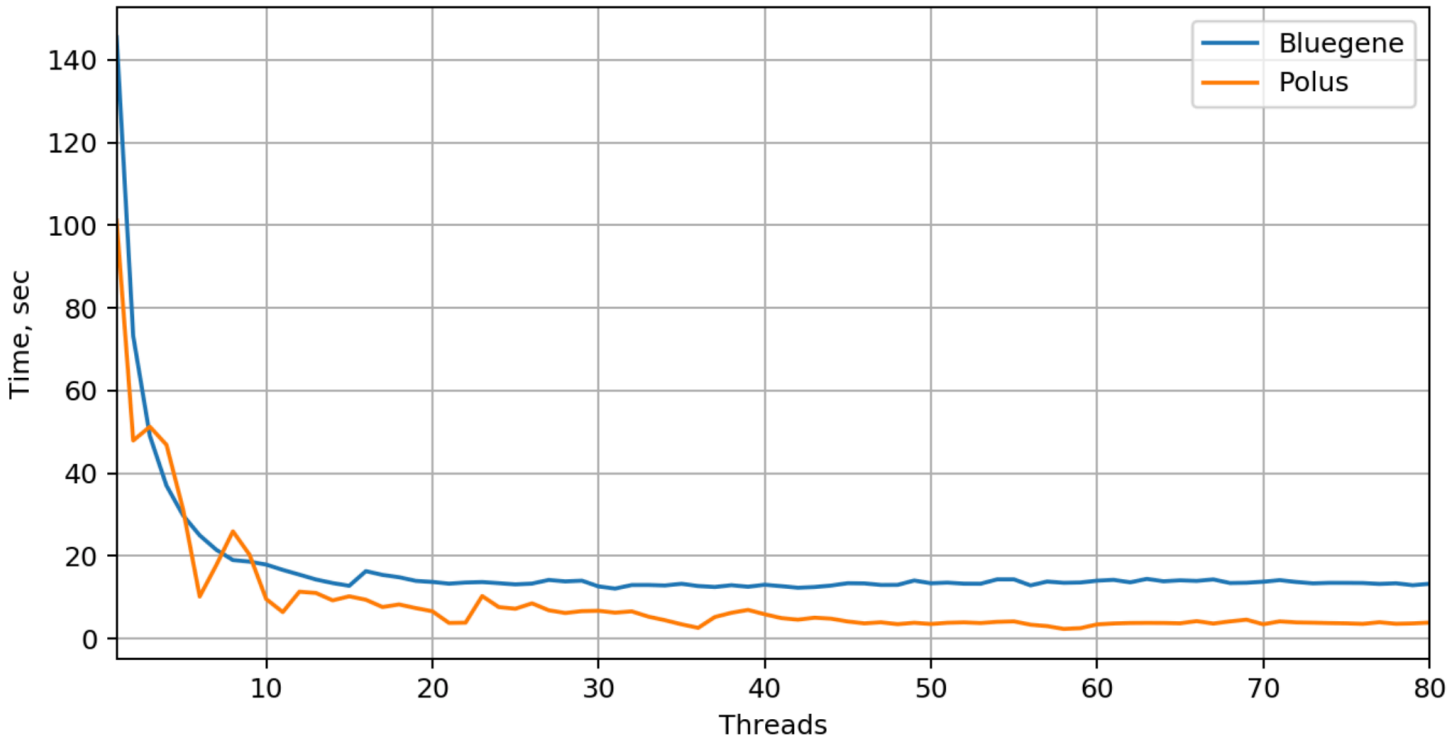


4.2 Визуализации

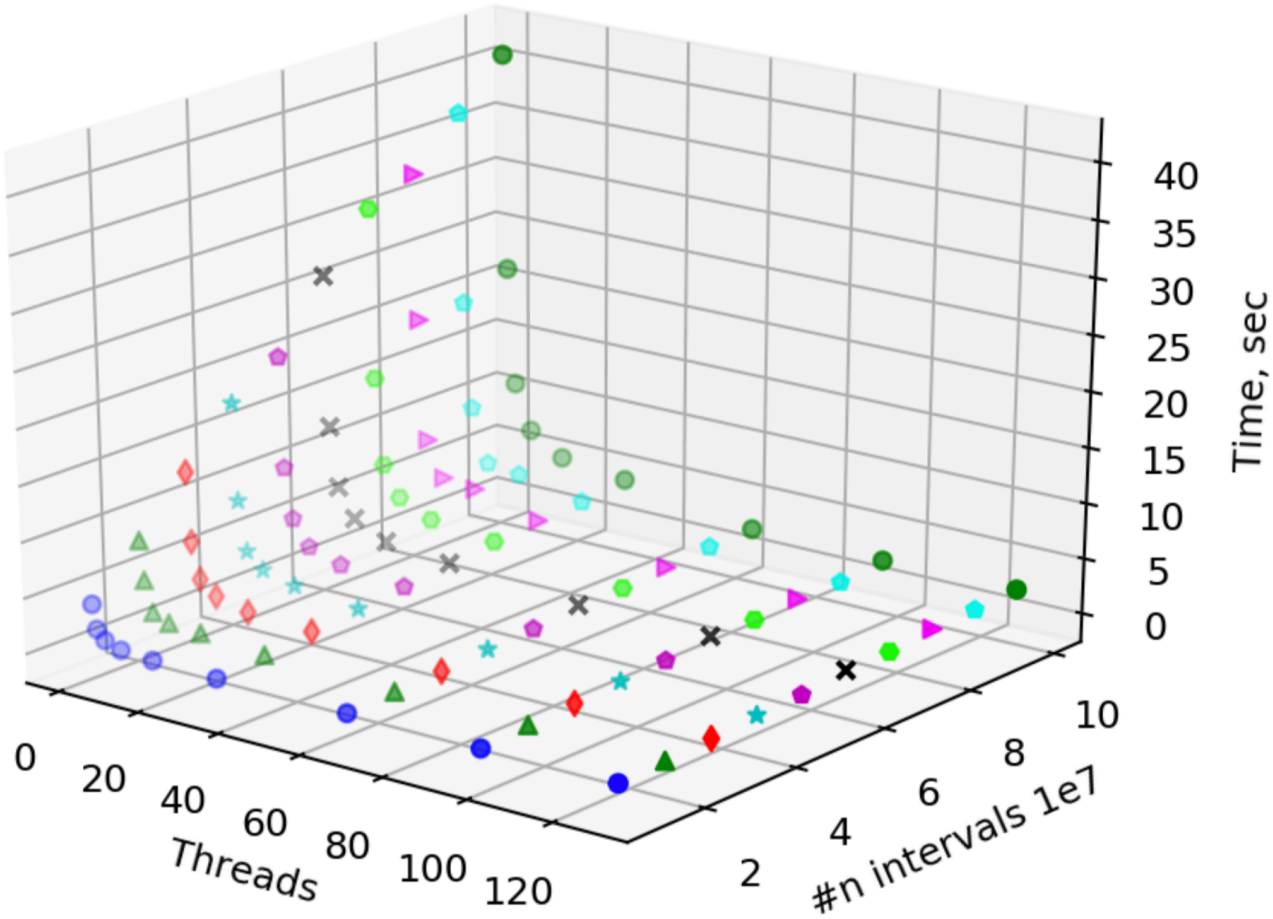
4.2.1 Сравнение скорости работы всех версий программы, включая наивную реализацию (baseline)

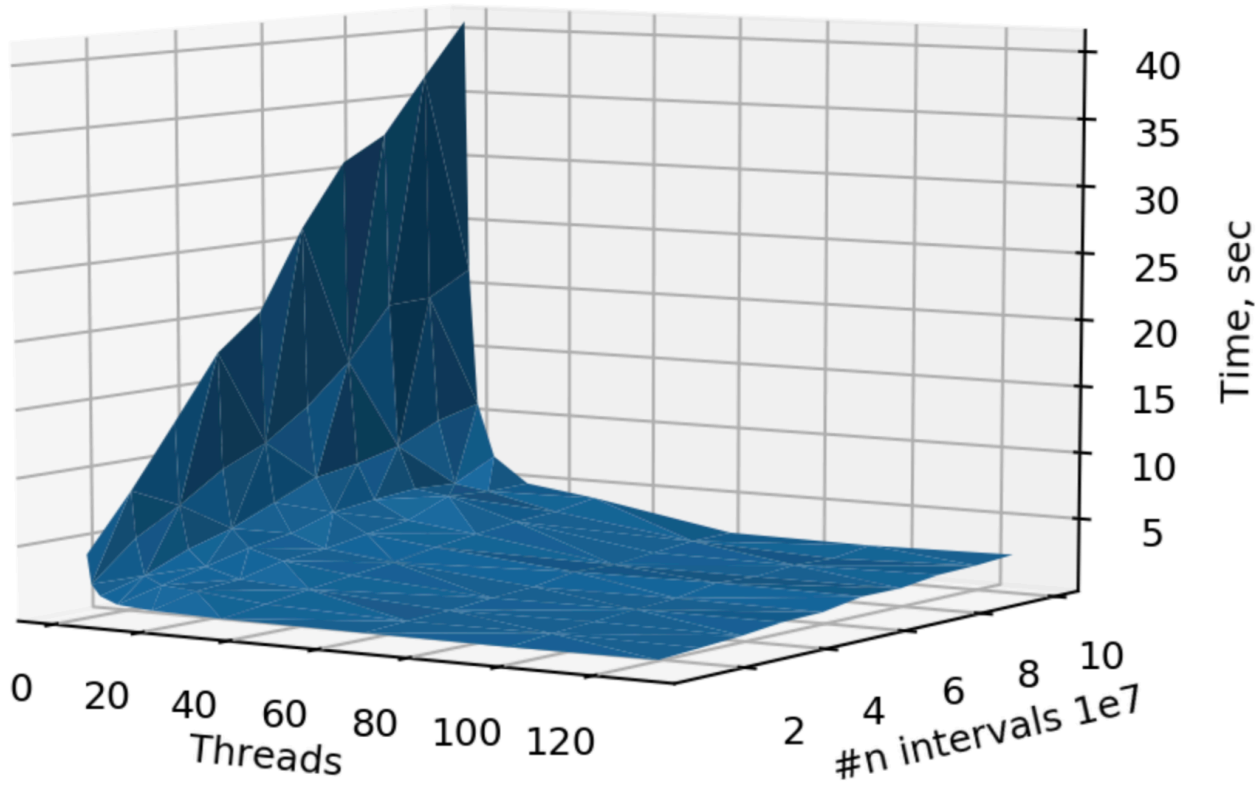


4.2.2 Время работы на BlueGene и Polus ($n = 10^8$)



4.2.3 Пространственная визуализация





5. Анализ результатов

Задачу удалось значительно ускорить с применением OpenMP. В результате были сделаны наблюдения:

- Существует предел количества нитей (в эксперименте с числом интервалов 1e8 – 96), когда время перестает значительно меняться. Связано это с увеличением накладных расходов.
- Одновременный подсчет суммы на четных и нечетных позициях дает прирост производительности, т.к. уменьшается число витков цикла и число дополнительных операций проверки четности.
- С целью уменьшения накладных расходов была сделана попытка имплементировать реализацию функции в место её вызова, однако прироста производительности это не дало – компилятор с уровнем оптимизации (-O3) самостоятельно выполняет встраивание простых функций в место вызова.
- При сравнении суперкомпьютеров Polus и BlueGene выявлено, что первый работает быстрее. Возможно, дело в тактовой частоте – у Polus (Power8) – 2.5GHz, у BlueGene – 850MHz.

6. Выводы

Выполнена разработка программы для подсчета значения интеграла методом Симпсона. Были реализованы последовательная и параллельная версии с использованием технологии OpenMP, проведено тестирование на суперкомпьютерах Polus и Blue Gene. Исследована масштабируемость параллельной версии программы, выполнен анализ результатов.