Проблема масштабируемости в OpenMP: неравномерная загрузка нитей – использование конечного параллелизма.

Подготовил Павел Никишкин
323 группа, СКИ ВМК

Московский государственный университет имени М. В. Ломоносова, факультет ВМК

Проблема

При разработке алгоритмов, использующих в том или ином виде конечный параллелизм, естественным образом возникает проблема ограниченности возможностей оптимизации. Мы не можем получить значительное ускорение для таких алгоритмов, так как параллельная структура алгоритма предусмотрена заранее и не поддается дополнительному распараллеливанию, используя, например, более производительные архитектуры. Поэтому, разрабатывая тот или иной параллельный алгоритм, необходимо учитывать возможность его исполнения произвольным числом нитей.

Пример – алгоритм Merge Sort (сортировка слиянием)

Так как при такой сортировке массив делится пополам, после чего выполняется сортировка каждой отдельной части и последующее слияние отсортированных массивов в один (перемещением указателей), естественным образом возникает желание распараллелить алгоритм на этапе слияния: первая нить организует слияние с начала массива и расставляет минимальные элементы в начале, вторая нить – с конца массива и расставляет максимальные элементы. Получаем ускорение алгоритма в 2 раза и не более.

Иллюстрация алгоритма Merge Sort

Пример – алгоритм Merge Sort (сортировка слиянием)

При разделении алгоритма на большее число нитей есть возможность получить ускорение в большем объеме

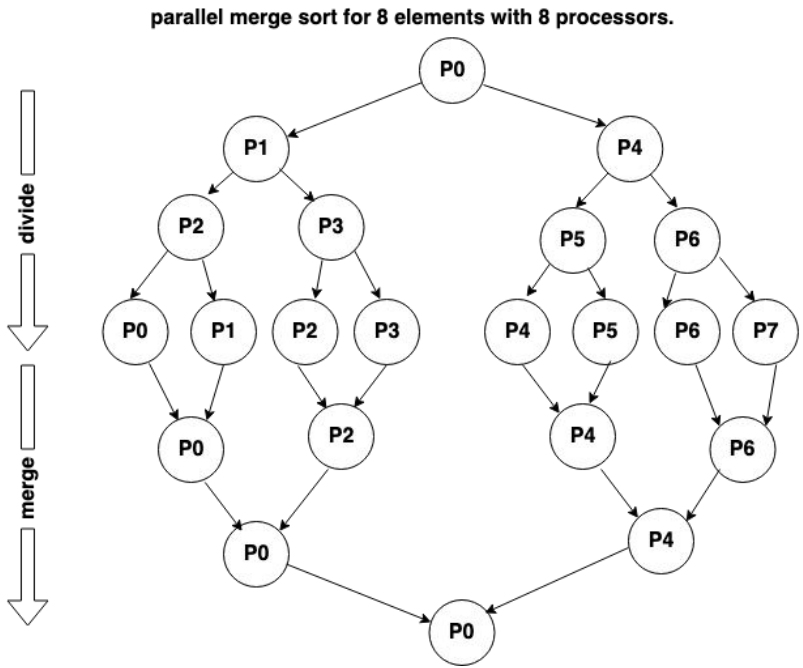

Иллюстрация алгоритма Merge Sort с распараллеливанием

Сравнение версий

Конечный параллелизм

```
void merge(std::vector<double>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    std::vector<double> L(n1), R(n2);
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            int i = 0, j = 0, k = l;
            while (i < n1 && j < n2) {
                if (L[i] <= R[j]) {
                    arr[k] = L[i];
                    i++;
                } else {
                    arr[k] = R[j];
                    j++;
                }
                k++;
            }
            while (i < n1) {
                arr[k] = L[i];
                i++;
                k++;
            }
        }
        #pragma omp section
        {
            int i = n1 - 1, j = n2 - 1, k = r;
            while (i >= 0 && j >= 0) {
                if (L[i] >= R[j]) {
                    arr[k] = L[i];
                    i--;
                } else {
                    arr[k] = R[j];
                    j--;
                }
                k--;
            }
            while (j >= 0) {
                arr[k] = R[j];
                j--;
                k--;
    } } } }

void mergeSort(std::vector<double>& arr, int l, int r) {
    if (l >= r)
        return;
    int m = l + (r - l) / 2;
    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);
    merge(arr, l, m, r);
}
```

Оптимизированная версия параллелизма

```
void merge(std::vector<double>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    std::vector<double> L(n1);
    std::vector<double> R(n2);
    #pragma omp parallel for
    for (int i = 0; i < n1; i++) {
        L[i] = arr[l + i];
    }
    #pragma omp parallel for
    for (int j = 0; j < n2; j++) {
        R[j] = arr[m + 1 + j];
    }
    int i = 0;
    int j = 0;
    int k = l;
    while (i < n1 && j < n2) {rr[k] = L[i];
            i+
        if (L[i] <= R[j]) {
            a+;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}


void mergeSort(std::vector<double>& arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        #pragma omp task
        mergeSort(arr, l, m);
        #pragma omp task
        mergeSort(arr, m + 1, r);
        #pragma omp taskwait
        merge(arr, l, m, r);
    }
}
```

Параметры тестирования

1. Массив double из 1000000 элементов.
2. Параметры системы:

   Polus - параллельная вычислительная система, 5 вычислительных узлов.

   Основные характеристики каждого 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

Результаты тестирования

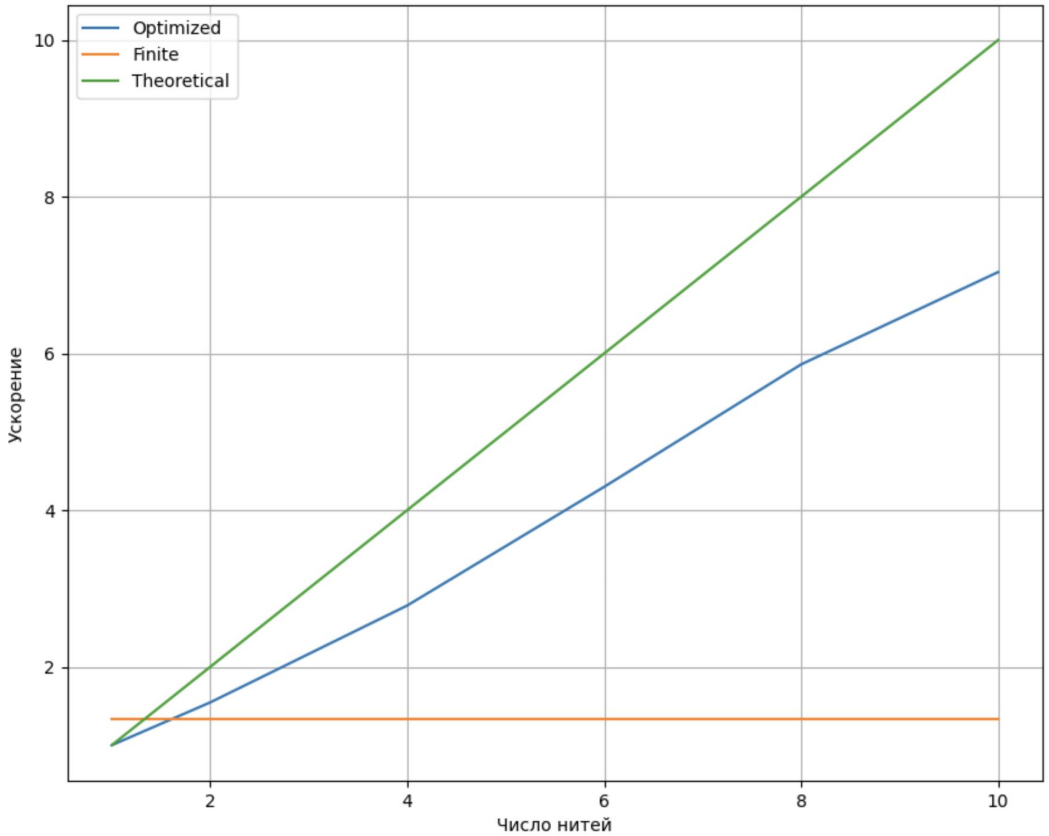

График зависимости ускорения от числа нитей