Практикум по параллельному программированию
Задание [Системное-1]

Гаврилов Олег Алексеевич
МГУ-Саров

Описание проблемы. Одним из недостатков высокоуровневых языков программирования является потенциальное значительное снижение скорости работы программы. Поэтому компиляторы стараются автоматически оптимизировать код и сделать его действительно быстрым или компактным (или сразу вместе).

Оптимизация – это процесс преобразования фрагмента кода в другой фрагмент, который функционально эквивалентен исходному, с целью улучшения одной или нескольких его характеристик, из которых наиболее важными являются скорость и размер кода. Другие характеристики включают количество потребляемой энергии на выполнения кода и время компиляции. Некоторые способы оптимизации, применяемые компилятором:

- подстановка функций,
- свёртка констант,
- устранение указателя,
- устранение общих подвыражений,
- регистровые переменные,
- анализ времени жизни переменной,
- объединение одинаковых ветвей,
- устранение переходов,
- раскрутка цикла,
- векторизация,
- алгебраические преобразования.

Таким образом, при использовании параллельных программ отключённая оптимизация компилятора может являться причиной существенного снижения масштабируемости.

Проблемный код. Рассматривается пример с умножением квадратной матрицы порядка N на вектор в параллельном режиме с помощью OpenMP. Сравнивались два случая: с включённой оптимизацией компилятора и без неё. Ниже представлен полный код программы на C++.

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    int T = atoi(argv[1]);
    int N = atoi(argv[2]);
    double** a = new double*[N];
    double* b = new double[N];
    double* c = new double[N];
    omp_set_num_threads(T);

    //initialization
    #pragma omp parallel for shared(N, a, b)
    for (int i = 0; i < N; ++i) {
        a[i] = new double[N];
        for (int j = 0; j < N; ++j) {
            a[i][j] = (i + j)*0.1234;
        }
        b[i] = 3.64*i;
    }

    //matrix-vector product
    double start = omp_get_wtime();
    #pragma omp parallel for shared(N, a, b, c)
    for (int i = 0; i < N; ++i) {
        c[i] = 0.0;
        for (int j = 0; j < N; ++j) {
            c[i] += a[i][j] * b[j];
        }
    }
    double time = omp_get_wtime() - start;
    printf("N = %d. T = %d. Wall time(s): %f \n", N, T, time);

    for (int i = 0; i < N; ++i)
        delete[] a[i];
    delete[] a;
    delete[] b;
    delete[] c;
}
```

Компиляция. Запуск проводился на одном процессе на полигоне ЦХАБТ. Конфигурация: четыре 36-ядерных узла на базе Intel Xeon Gold 6140 @ 2.30 GHz, суммарно 144 физических ядра. Для параллельной работы создавались от 1 до 32 нитей OpenMP. Исходный код программы компилировался дважды: с использованием ключа `-O0` (отсутствие оптимизации) и с использованием ключа `-O3` (максимальная оптимизация).

Компиляция проводилась при помощи `gcc-7.4`, который используется для программ на C++. После компиляции запускается `batch`-файл с заранее написанными командами. Ниже приведён порядок компиляции и запуска программы.

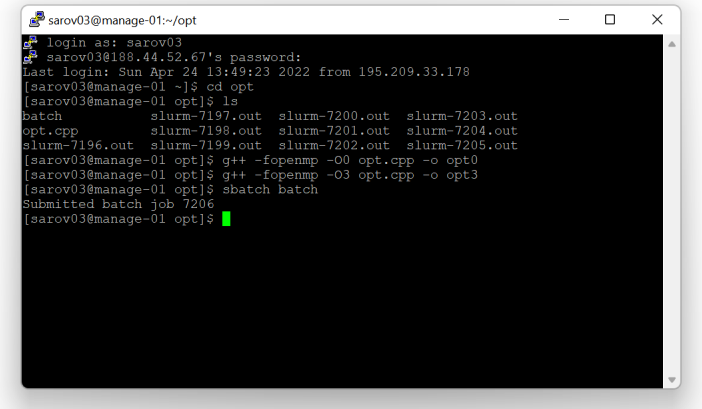

Рис. 1: Порядок компиляции и запуска на сервере.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --cpus-per-task=1

module load gcc/gcc-7.4

./opt0 1 84000
./opt0 2 84000
./opt0 4 84000
./opt0 6 84000
./opt0 8 84000
./opt0 10 84000
./opt0 12 84000
./opt0 14 84000
./opt0 16 84000
./opt0 18 84000
./opt0 20 84000
./opt0 22 84000
./opt0 24 84000
./opt0 26 84000
./opt0 28 84000
./opt0 30 84000
./opt0 32 84000

./opt3 1 84000
./opt3 2 84000
./opt3 4 84000
```

Рис. 2: Фрагмент содержимого `batch`-файла.

Результаты серии испытаний. Пусть $N = 84000$ (порядок матрицы и вектора). Число нитей T меняется от 1 до 32 (сначала 1, затем 2 и далее с шагом 2). В каждом случае – с оптимизацией и без – измеряется время выполнения программы (инициализация данных + непосредственно умножение матрицы на вектор).

```
N = 84000. T = 1. Wall time(s): 36.409197
N = 84000. T = 2. Wall time(s): 18.538189
N = 84000. T = 4. Wall time(s): 10.255362
N = 84000. T = 6. Wall time(s): 6.501349
N = 84000. T = 8. Wall time(s): 5.937970
N = 84000. T = 10. Wall time(s): 5.701944
N = 84000. T = 12. Wall time(s): 5.115629
N = 84000. T = 14. Wall time(s): 5.962367
N = 84000. T = 16. Wall time(s): 5.655157
N = 84000. T = 18. Wall time(s): 5.140663
N = 84000. T = 20. Wall time(s): 4.362861
N = 84000. T = 22. Wall time(s): 3.791639
N = 84000. T = 24. Wall time(s): 4.510370
N = 84000. T = 26. Wall time(s): 4.364644
N = 84000. T = 28. Wall time(s): 3.700607
N = 84000. T = 30. Wall time(s): 3.506943
N = 84000. T = 32. Wall time(s): 4.446835

N = 84000. T = 1. Wall time(s): 23.906773
N = 84000. T = 2. Wall time(s): 15.817432
N = 84000. T = 4. Wall time(s): 4.706036
N = 84000. T = 6. Wall time(s): 2.935820
N = 84000. T = 8. Wall time(s): 2.557739
N = 84000. T = 10. Wall time(s): 1.942370
N = 84000. T = 12. Wall time(s): 3.159858
N = 84000. T = 14. Wall time(s): 2.035423
N = 84000. T = 16. Wall time(s): 1.800621
N = 84000. T = 18. Wall time(s): 1.727391
```

Рис. 3: Вывод программы (первые 17 строк – случай без оптимизации).

T	Время без опт., сек	Время с опт., сек
1	36.409197	23.906773
2	18.538189	15.817432
4	10.255362	4.706036
6	6.501349	2.93582
8	5.93797	2.557739
10	5.701944	1.94237
12	5.115629	3.159858
14	5.962367	2.035423
16	5.655157	1.800621
18	5.140663	1.727391
20	4.362861	2.647768
22	3.791639	2.466685
24	4.51037	2.246717
26	4.364644	2.816147
28	3.700607	3.030796
30	3.506943	2.22006
32	4.446835	2.983185

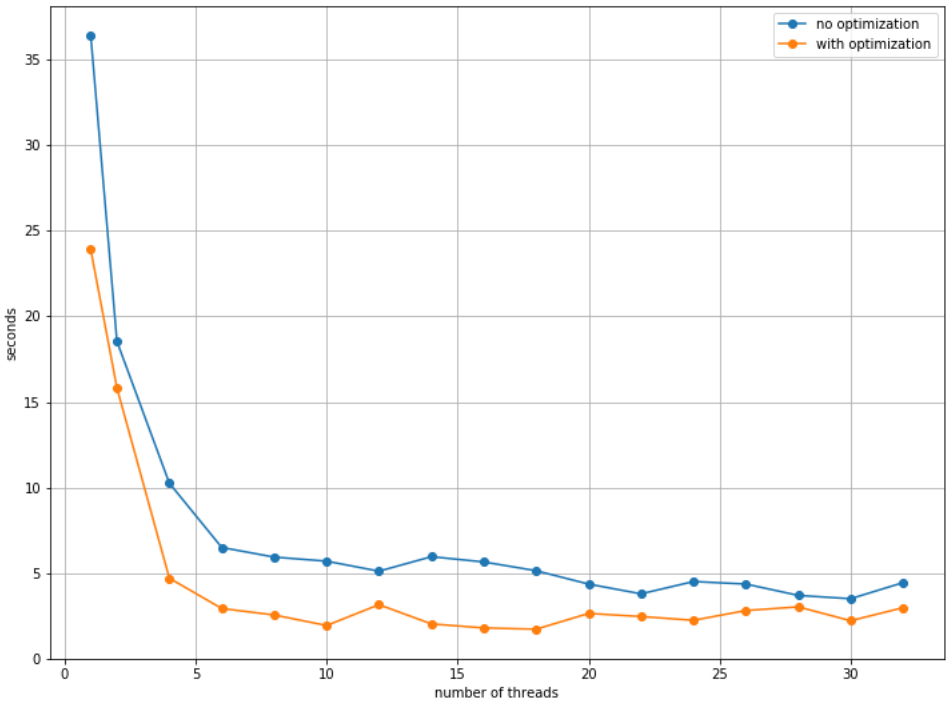

Рис. 4: Зависимость времени от числа нитей для обоих случаев.