УНИВЕРСИТЕТ ИТМО

Лабораторная работа №1 "Автоматическое распараллеливание программ"

Дисциплина

Параллельные вычисления

Автор

Дмитрий Рачковский

10 ноября 2020 г.

Содержание

1 Введение

1.1 Цель работы

В данной работе необходимо исследовать эффективность автоматического распараллеливания программ на языке C компилятором gcc. Для этого одна и та же программа, производящая достаточное количество вычислений, компилируется сначала линейно, а затем с использованием автоматического распараллеливания на разное количество потоков. Производительность полученных выполняемых файлов сравнивается. Также производится контроль результата вычислений, который не должен меняться (в пределах допустимой погрешности) при распараллеливании, гарантируя правильность выполнения.

1.2 Использованное оборудование

Для проведения экспериментов использовалась виртуальная машина с OS Debian. Виртуальная машина получила доступ к 6 ядрам (используется 64-битный процессор Intel Core i7-8750H @ 2.20 GHz с 6 физическими и 12 логическими ядрами) и 6 ГБ оперативной памяти (из 16, доступных в системе). Во время выполнения экспериментов система была подключена к источнику питания.

Версия использованного компилятора: gcc (Debian 6.3.0-18+deb9u1) 6.3.0 20170516.

2 Ход работы

2.1 Используемая программа

Для проведения экспериментов была написана программа, производящая следующие действия 50 раз:

- Создаются массивы $M1$ размерностью N и $M2$ размерностью $N/2$, которые заполняются случайными числами из определённых интервалов. Использование при заполнении массивов функции *rand_r* с начальным значением параметра *seed*, зависящим от номера итерации, обеспечивает одни и те же начальные значения массивов для соответствующих итераций при каждом запуске программы.
- К каждому элементу массива $M1$ применяется операция гиперболического синуса с последующим возведением в квадрат.
- Каждый элемент массива $M2$ складывается с предыдущим (к первому элементу ничего не добавляется), и затем заменяется модулем тангенса получившегося значения.
- Каждый элемент массива $M2$ заменяется соответствующим элементом массива $M1$, возведённым в степень этого элемента $M2$ ($M2[i] = M1[i]^{M2[i]}$)
- Массив $M2$ сортируется с использованием сортировки выбором
- Определяется минимальный ненулевой элемент массива $M2$
- Рассчитывается сумма элементов массива $M2$, которые при делении на минимальный ненулевой элемент массива, найденный на предыдущем шаге, дают чётное число (при проверке на чётность дробная часть игнорируется). Полученное при каждой итерации число складывается в итоговое число X. Оно служит результатом, который не должен изменяться при распараллеливании программы.
- На отдельных строчках выводится число N, время выполнения программы в миллисекундах и значение X.

Полный текст программы может быть найден в приложении 4.1.

2.2 Получение данных

Полученная программа компилируется без использования автоматического распараллеливания с использованием команды *gcc -O3 -Wall -Werror -lm -o lab1-seq lab1.c*. Затем эта же программа компилируется с использованием встроенного в gcc средства автоматического распараллеливания Graphite с использованием команды *gcc -O3 -Wall*

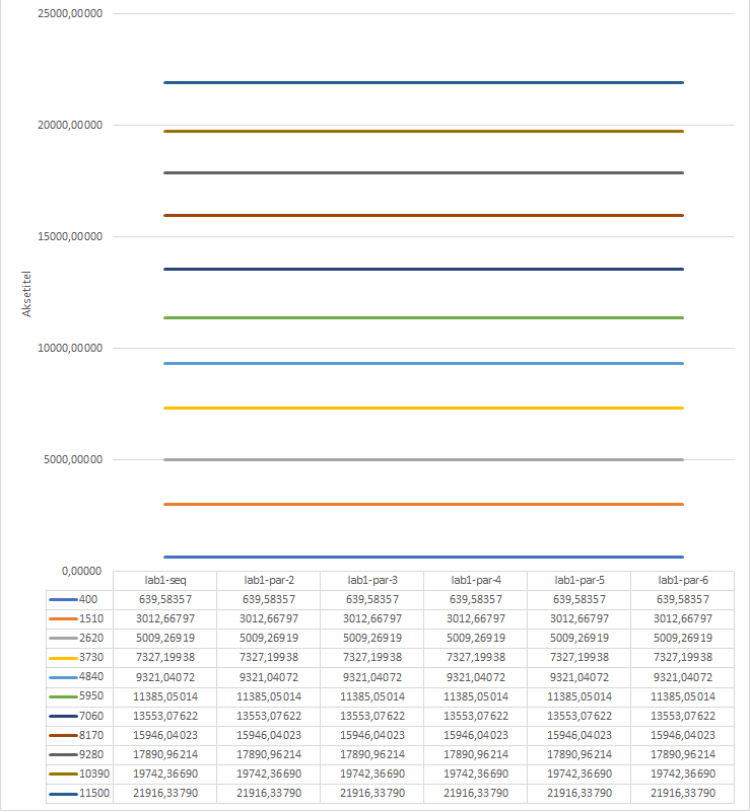
-Werror -lm -floopparallelize-all -ftree-parallelize-loops=K -o lab1-par-K lab1.c, где K поочерёдно принимает значения от 2 до 6. Таким образом программа поочерёдно распараллеливается на количество потоков от 2 до 6, достигая максимального количества ядер, доступного системе.

Были подобраны значения $N1 = 400$, при котором время выполнения программы *lab1-seq* превысило 10 мс, и $N2 = 11500$, при котором время выполнения программы превысило 2 с. Затем было найдено значение $\Delta = \frac{N2-N1}{10} = 1110$. Каждая из полученых программ запускается со значениями $N = N1, N1 + \Delta, N1 + 2\Delta, N1 + 3\Delta ... N2$.

2.3 Проверка результата

Перед тем, как анализировать полученные данные, нужно убедиться, что распараллеливание прошло правильно и результат не был искажен. Для этого нужно обратить внимание на значения X, которые были получены в ходе работы программ. Эти значения не должны отличаться для одних и тех же итераций при одинаковом значении N.

Действительно, при разных значениях K значения X не отличаются. На представленном графике видно, что при всех значениях N число X остаётся одним и тем же для всех значений K:

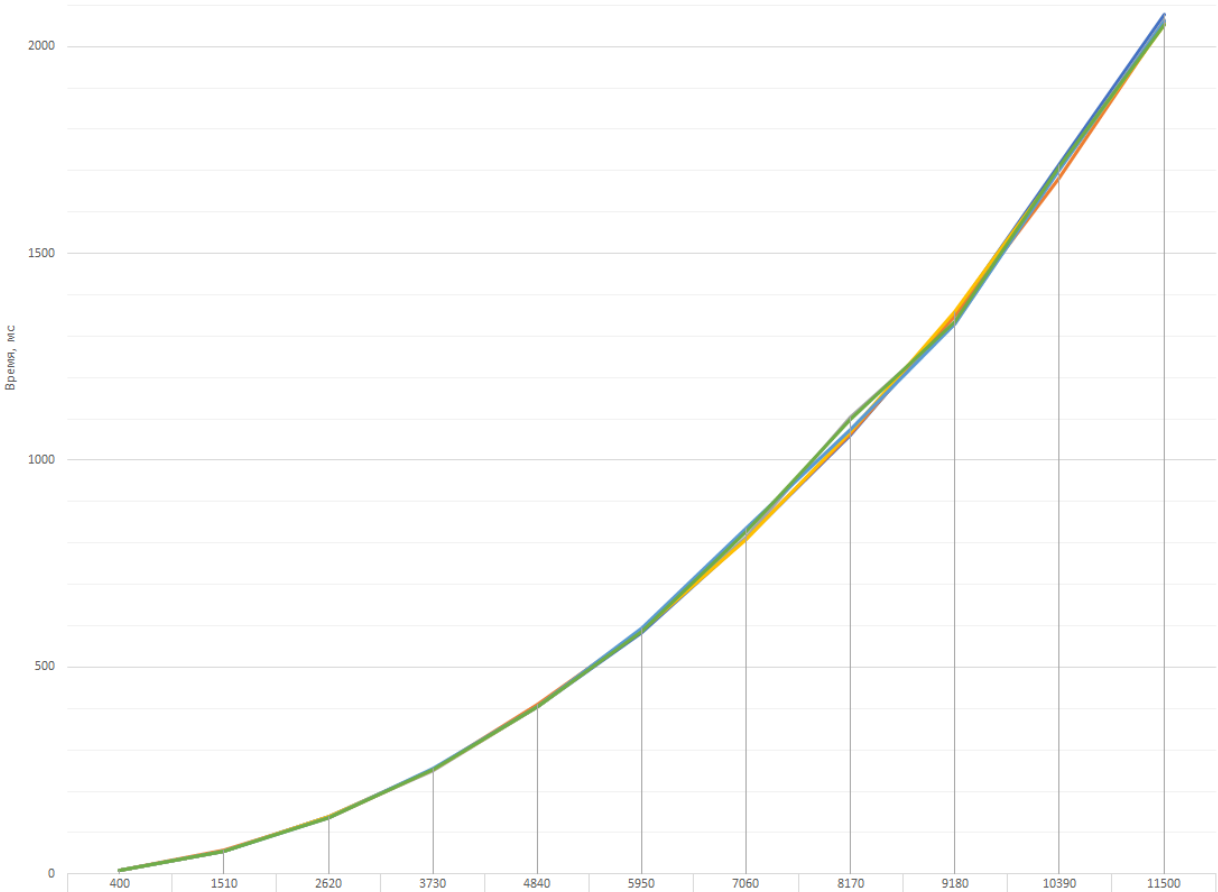


	lab1-seq	lab1-par-2	lab1-par-3	lab1-par-4	lab1-par-5	lab1-par-6
400	639,58357	639,58357	639,58357	639,58357	639,58357	639,58357
1510	3012,66797	3012,66797	3012,66797	3012,66797	3012,66797	3012,66797
2620	5009,26919	5009,26919	5009,26919	5009,26919	5009,26919	5009,26919
3730	7327,19938	7327,19938	7327,19938	7327,19938	7327,19938	7327,19938
4840	9321,04072	9321,04072	9321,04072	9321,04072	9321,04072	9321,04072
5950	11385,05014	11385,05014	11385,05014	11385,05014	11385,05014	11385,05014
7060	13553,07622	13553,07622	13553,07622	13553,07622	13553,07622	13553,07622
8170	15946,04023	15946,04023	15946,04023	15946,04023	15946,04023	15946,04023
9280	17890,96214	17890,96214	17890,96214	17890,96214	17890,96214	17890,96214
10390	19742,36690	19742,36690	19742,36690	19742,36690	19742,36690	19742,36690
11500	21916,33790	21916,33790	21916,33790	21916,33790	21916,33790	21916,33790

Рис. 1: Значения X при различных значениях N и K

2.4 Анализ результатов

По выполнению экспериментов я заметил, что время выполнения программ, автоматически распараллеленных компилятором, не отличается. Время не абсолютно идентично, но отличия не превышают нескольких десятков миллисекунд и являют собой погрешность в измерении. Точные результаты видны на следующем графике:



	400	1510	2620	3730	4840	5950	7060	8170	9180	10390	11500
seq	8	56	139	252	404	585	811	1061	1355	1715	2077
K=2	8	58	136	252	409	588	812	1064	1348	1683	2059
K=3	8	55	139	251	405	592	814	1105	1330	1704	2063
K=4	9	54	138	253	405	590	808	1070	1360	1703	2052
K=5	8	54	136	255	406	594	837	1074	1330	1702	2061
K=6	8	55	137	253	405	587	828	1098	1334	1709	2053

Рис. 2: Отличия производительности по-разному распараллеленых программ

Так как время выполнения не отличается, величина параллельного ускорения, найденная по формуле $S(p)_{w=const} = \frac{t(1)}{t(p)}$, приблизительно равна 1 для всех случаев.

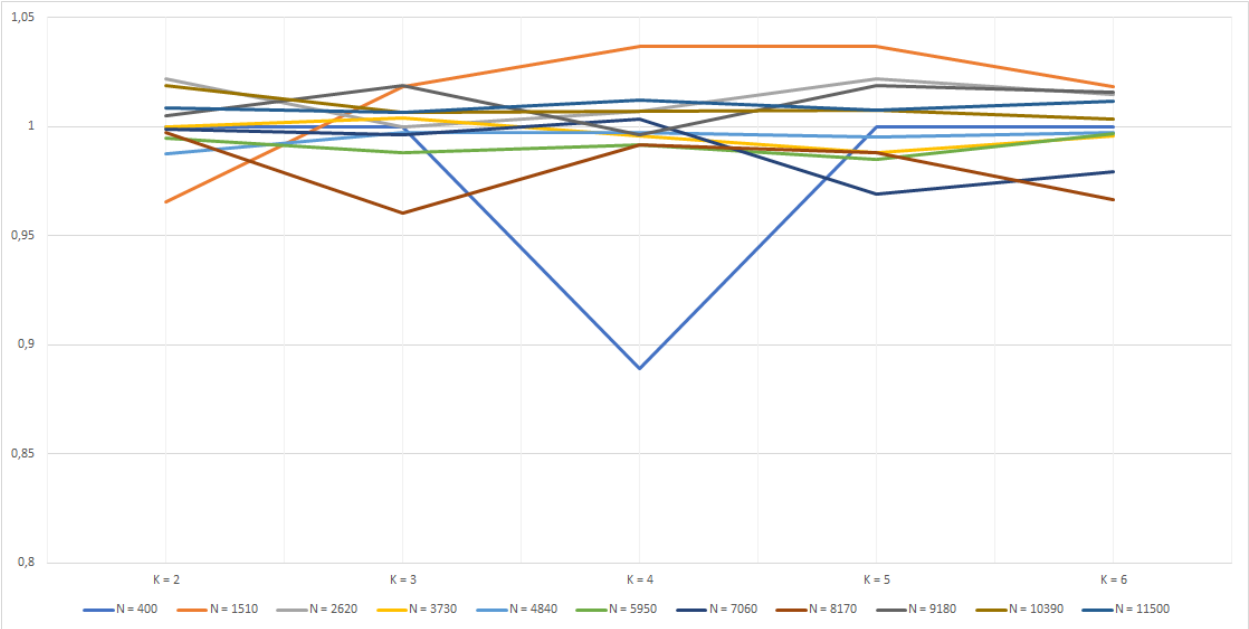


Рис. 3: Параллельное ускорение распараллеленых программ

Параллельная эффективность таким образом обратно пропорциональна числу K, использованному при компиляции и больше ни от чего не зависит.

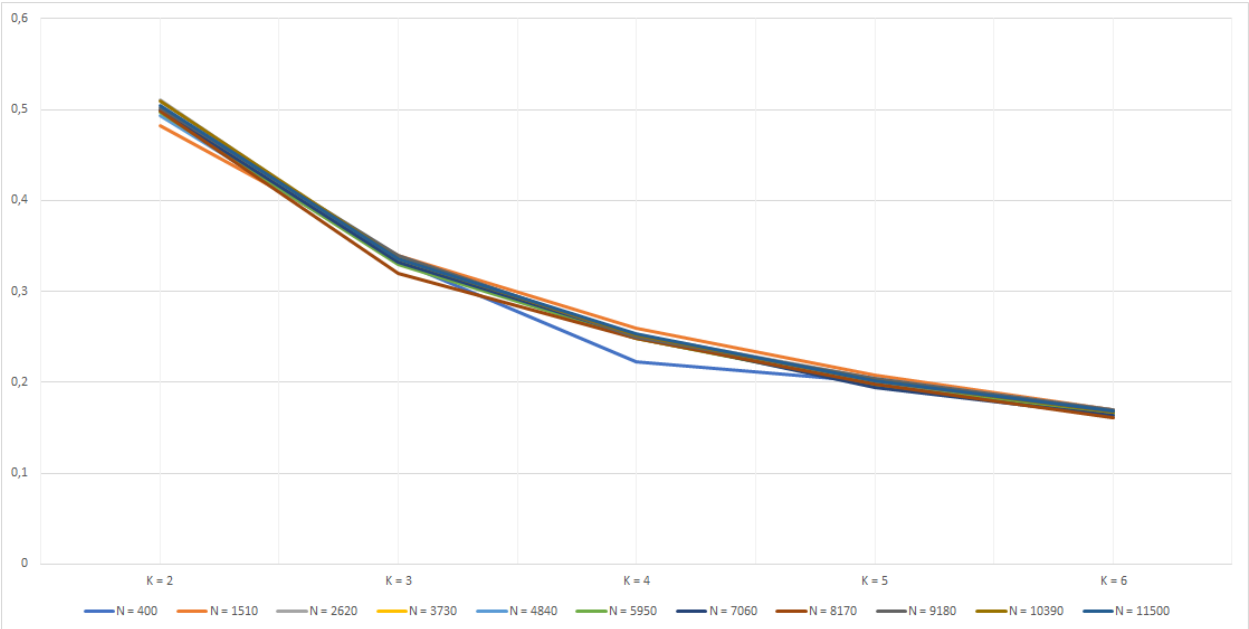


Рис. 4: Параллельная эффективность распараллеленых программ

Я провёл подобный эксперимент для бóльших значений N, где время выполнения составляло от 10 до 60 секунд, но также получил не отличающиеся результаты и величину параллельного ускорения равную 1.

3 Вывод

На основании данного эксперимента можно сделать вывод, что автоматическое распараллеливание средствами компилятора gcc не дало для моей программы абсолютно никакого эффекта. Несмотря на полное отсутствие зависимостей между 50 итерациями, которые проводились в каждой программе, положительного эффекта от параллелизации не было замечено. Причин этому может быть несколько:

- **Накладные расходы на параллелизацию** Наличие накладных расходов могло бы нивелировать преимущество во времени, которое дает параллелизация. Однако, в этом случае время работы могло бы значительно ухудшиться, особенно для небольших значений N. Так как время работы для по-разному распараллеленных программ практически одинаковое, мне не кажется, что накладные расходы явились причиной отсутствия прироста скорости.
- **Невозможность параллелизации** Несмотря на отсутствие зависимостей между циклами (как минимум между 50 различными экспериментами не было никаких зависимостей), gcc не смог распараллелить программу и поэтому время работы не изменилось.
- **Отсутствие параллелизации** Некоторые источники утверждают, что с определенной версии компилятор gcc перестал осуществлять автоматическое распараллеливание, не упоминая этого в документации. Информация не достоверная, но это также может быть причиной отсутствия какого-либо увеличения производительности распараллеленых программ.

4 Приложения

4.1 Исходный код программы lab1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>

#define A 700 //
#define NUMBER_OF_ITERATIONS 50

long double random_on_interval(long double min, long double max, unsigned int *seed) {
    return (long double) ((rand_r(seed) % (int)(max + 1 - min)) + min);
}

int main(int argc, char* argv[]) {
    struct timeval T1, T2;
    int N = atoi(argv[1]);

    long double M1[N];
    long double M2[N/2];
    long double M2_shifted_copy[N/2];

    double X = 0;

    gettimeofday(&T1, NULL);

    int i, j;
    for (i=0; i<NUMBER_OF_ITERATIONS; i++) {
        /******************* GENERATE ********************/
        unsigned int seed = i;
        for (j = 0; j < N; j++) {
            M1[j] = random_on_interval(0, A, &seed);
        }
        for (j = 0; j < N/2; j++) {
            M2[j] = random_on_interval(A, A*10, &seed);
            M2_shifted_copy[j] = j == 0 ? 0 : M2[j-1]; // shifted copy of M2 needed on the next step
        }


        /******************* MAP ************************/
        for (j = 0; j < N; j++) {
            // operation #1, remember to convert to radians
            M1[j] = pow(sinhl((M1[j] * M_PI) / 180.0), 2);
        }


        for (j = 0; j < N/2; j++) {
            // operation #3
            M2[j] = fabs(tanl(M2[j] + M2_shifted_copy[j]));
        }

        /******************* MERGE ***********************/
        for (j = 0; j < N/2; j++) {
            // operation #1
            M2[j] = pow(M1[j], M2[j]);
```

```
54          }
55
56          /******************* SORT ***********************/
57          int k, l, min_k;
58          for (k = 0; k < (N/2)-1; k++)
59          {
60              min_k = k;
61              for (l = k+1; l < N/2; l++)  {
62                  if (M2[l] < M2[min_k]) {
63                      min_k = l;
64                  }
65              }
66              if (min_k != k) {
67                  long double temp = M2[min_k];
68                  M2[min_k] = M2[k];
69                  M2[k] = temp;
70              }
71          }
72
73          /******************* REDUCE *********************/
74          int min_index;
75          for (min_index = 0; M2[min_index] <= 0; min_index++) {}
76          long double min = M2[min_index];
77
78          for (j = 0; j < N/2; j++) {
79              if (isfinite(M2[j]) && (int)(M2[j] / min) % 2 == 0) {
80                  // remember to convert to radians
81                  X += sinl((M2[j] * M_PI) / 180.0);
82              }
83          }
84      }
85
86      gettimeofday(&T2, NULL);
87      long delta_ms = 1000 * (T2.tv_sec - T1.tv_sec) + (T2.tv_usec - T1.tv_usec)
            / 1000;
88      printf("%d\n", N);
89      printf("%ld\n", delta_ms);
90      printf("%.5f\n", X);
91
92      return 0;
93  }
```

4.2 Скрипт для компиляции программы lab1.c

```
1 gcc -O3 -Wall -Werror -lm -o lab1-seq lab1.c
2 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=1 -o
      lab1-par-1 lab1.c
3 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=2 -o
      lab1-par-2 lab1.c
4 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=3 -o
      lab1-par-3 lab1.c
5 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=4 -o
      lab1-par-4 lab1.c
6 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=5 -o
      lab1-par-5 lab1.c
7 gcc -O3 -Wall -Werror -lm -floop-parallelize-all -ftree-parallelize-loops=6 -o
      lab1-par-6 lab1.c
```

4.3 Скрипт для запуска программы lab1.c

```
#!/bin/bash

declare -a executables=("lab1-seq" "lab1-par-2" "lab1-par-3" "lab1-par-4" "lab1-par-5" "lab1-par-6")
declare -a N_set=("400" "1510" "2620" "3730" "4840" "5950" "7060" "8170" "9180" "10390" "11500")
FILENAME="N.csv"

> N.csv
echo -e ";" >> ${FILENAME}
echo -e ";" >> ${FILENAME}
echo -e ";" >> ${FILENAME}

for N in "${N_set[@]}"
do
    for exec in "${executables[@]}"
    do
        X=$(./${exec} ${N})
        y=(${X//$'\n'/ })
        line_num=1
        for num in ${X}; do
            sed -e "${line_num}s/\$/$(sed 's/\./,/g' <<< ${num});/" -i ${FILENAME}
            ((line_num=line_num+1))
        done
    done
    sed -e "s/\$/;/" -i ${FILENAME}
done

exit 0
```