УНИВЕРСИТЕТ ИТМО

Лабораторная работа №4 "Метод доверительных интервалов при измерении времени выполнения паралелльной OpenMP-программы"

Дисциплина

Параллельные вычисления

Автор

Дмитрий Рачковский

27 декабря 2020 г.

Содержание

1 Введение **1**
1.1 Цель работы 1
1.2 Использованное оборудование 1

2 Ход работы **2**
2.1 Используемая программа 2
2.2 Получение данных 2
2.3 Анализ результатов 3
2.4 Доверительный интервал 4

3 Вывод **5**

4 Приложения **I**
4.1 Исходный код программы lab4.c I
4.2 Скрипт для компиляции программы lab4.c V
4.3 Скрипт для запуска программы lab4.c V

1 Введение

1.1 Цель работы

В данной работе необходимо исследовать эффективность распараллеливания программ на языке C с помощью технологии OpenMP. Для этого одна и та же программа, производящая достаточное количество вычислений с использованием канонических циклов for, компилируется сначала без использования OpenMP, а затем с использованием распараллеливания циклов for. Время выполенения полученных выполняемых файлов сравнивается с помощью вычисления доверительных интервалов. 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.

Версия OpenMP: 4.5 (201511)

2 Ход работы

2.1 Используемая программа

Для проведения экспериментов программа, использованная в лабораторной работе №3, была модифицирована следующим образом:

- Вызовы функции *gettimeofday* были заменены на *omp_get_wtime*.
- Вычисления на этапе Sort были распараллелены на k потоков, где k - количество ядер в системе.
- Была добавлена функция, выполняющаяся в отдельном потоке и раз в секунду выводящая на экран процент выполнения программы.
- Программа была проверена на прямую совместимость с компиляторами без поддержки OpenMP.

Полный текст программы может быть найден в приложении 4.1.

2.2 Получение данных

Полученная программа компилируется без подключения OpenMP для проверки прямой совместимости. Затем эта же программа распараллеливания с использованием флага -openmp. Команды для компиляции программы могут быть найдены в приложении 4.2.

Было найдено значение $N_x = 6700$, при котором накладные расходы на распараллеливание прекращают превышать выигрыш от распараллеливания. Полученная программа запускается со значениями $\frac{N_x}{2}, \frac{N_x}{2} + \Delta, \frac{N_x}{2} + 2\Delta, \frac{N_x}{2} + 3\Delta, ..., N_2$, где Δ была найдена в лабораторной работе №1, а $N_2 = 36650$.

2.3 Анализ результатов

По выполнению первого эксперимента было замечено, что этап Sort действительно был узким местом прошлой программы, которое сводило усилия по параллелизации на нет. После его распараллеливания параллельное ускорение линейно возрастает с увеличением значения N.

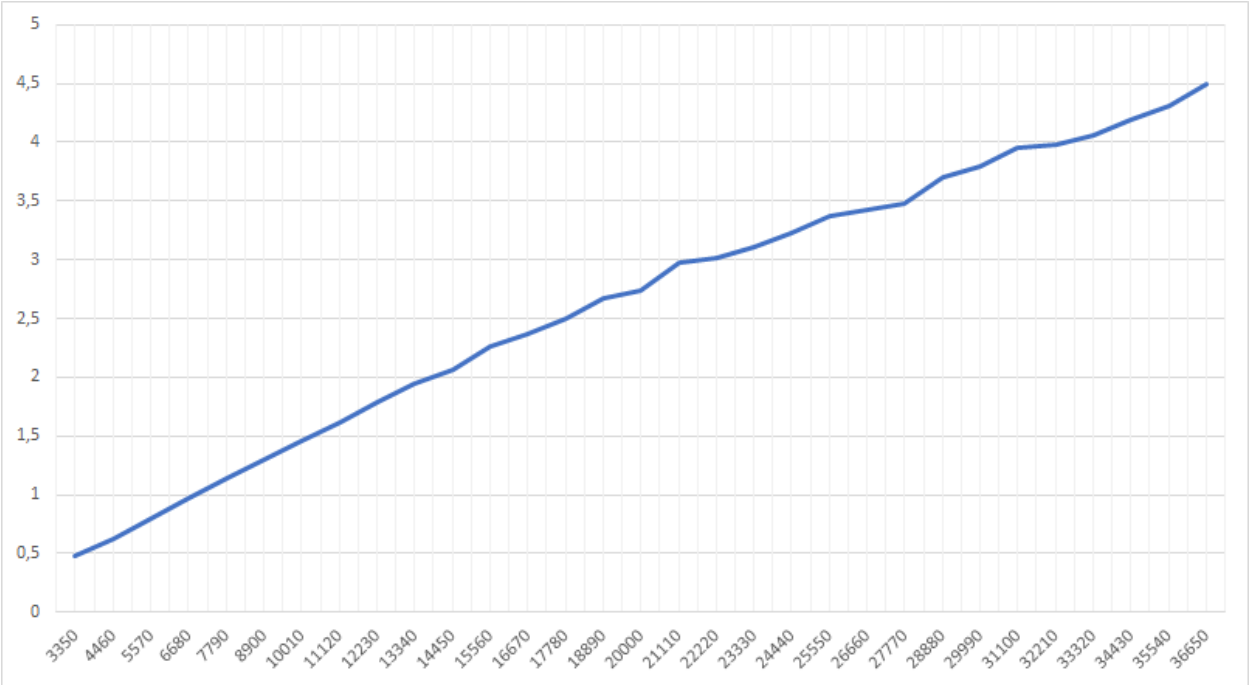

Рис. 1: Параллельное ускорение распараллеленой программы

Нужно, однако, заметить, что значение N_x, при котором накладные расходы на распараллеливание прекращают превышать выигрыш от распараллеливания, значительно выросло. При $N_x = 100$ в прошлой программе, сейчас $N_x = 6700$.

2.4 Доверительный интервал

Следующим экспериментом я уменьшил количество итераций основного цикла до 10, записывая время выполнения каждого из них. Затем я попробовал вычислить параллельное ускорение, беря наименьшее значение из 10, а также вычисляя верхнюю и нижнюю границу доверительного интервала с уровнем доверия 95%.

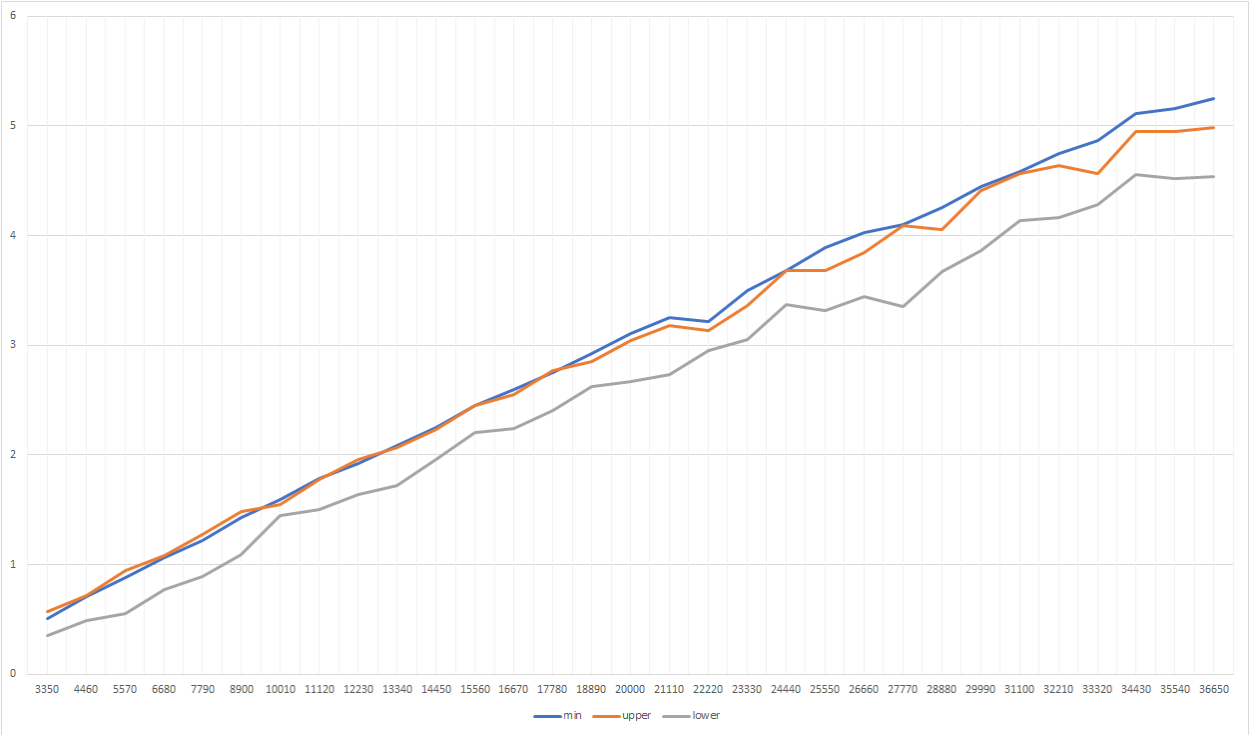

Рис. 2: Параллельное ускорение вычисленное методом доверительных интервалов

Параллельное ускорение, вычисленное путём выбора наименьшего времени выполнения, практечески всегда не входит в доверительный интервал, превышая его.

3 Вывод

На основании данного эксперимента можно сделать несколько выводов:

- Параллелизация процесса сортировки позволила получать ощутимый выигрыш при параллелизации и линейно растущее параллелиное ускорение с возрастанием значения N. Однако, минимальное значение, при котором накладные расходы на распараллеливание не превышают выигрыш от распараллеливания, возрасло в 6.7 раз.
- При вычислении параллельного ускорения использование минимального из полученных замеров даёт значение, превышающее верхнюю границу метода доверительных интервалов.

4 Приложения

4.1 Исходный код программы lab4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <math.h>

#define A 700 //
#define NUMBER_OF_ITERATIONS 50

typedef enum stage_ {
    STAGE_UNDEFINED = 0,
    STAGE_GENERATE = 0,
    STAGE_MAP,
    STAGE_MERGE,
    STAGE_SORT,
    STAGE_REDUCE,
    STAGE_NUM_OF,
} stage_t;

int current_iteration = 0;
stage_t current_stage = STAGE_UNDEFINED;

void monitor_execution_percent();

#ifdef _OPENMP
    #include "omp.h"
    void monitor_execution_percent() {
        while (1) {
            sleep(1);
            double execution_percent = (100. / NUMBER_OF_ITERATIONS) * current_iteration + (100. / NUMBER_OF_ITERATIONS) * ((double)current_stage / STAGE_NUM_OF);
            if (execution_percent < 100) {
                printf("Current progress: %.2f%%\n", execution_percent);
            } else {
                break;
            }
        };
    }
#else
    #include <sys/time.h>
    double omp_get_wtime() { struct timeval T; gettimeofday(&T, NULL); return T.tv_sec + T.tv_usec / 1000000.; }
    int omp_get_num_procs() { return 1; }
    int omp_get_thread_num() { return 0; }
    void omp_set_nested(int n) {}
    void monitor_execution_percent() {}
#endif

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);
}

unsigned int make_seed(int i, int j) {
    return 9572 + 234*i + 456*j;
}
```

```
54
55  int main(int argc, char* argv[]) {
56      int N = atoi(argv[1]);
57
58      omp_set_nested(1);
59
60      double T1, T2;
61      long double M1[N];
62      long double M2[N/2];
63      long double M2_copy[N/2];
64
65      double X = 0;
66
67      int j = 0;
68      #pragma omp parallel sections
69      {
70          #pragma omp section
71          {
72              monitor_execution_percent();
73          }
74          #pragma omp section
75          {
76              T1 = omp_get_wtime();
77              for (current_iteration=0; current_iteration<
                    NUMBER_OF_ITERATIONS; current_iteration++) {
78                  /******************** GENERATE ********************/
79                  current_stage = STAGE_GENERATE;
80                  unsigned int seed;
81
82                  #pragma omp parallel for default(none) private(j, seed)
                        shared(M1, N, current_iteration)
83                  for (j = 0; j < N; j++) {
84                      seed = make_seed(current_iteration, j);
85                      M1[j] = random_on_interval(0, A, &seed);
86                  }
87
88                  #pragma omp parallel for default(none) private(j, seed)
                        shared(M2, N, current_iteration)
89                  for (j = 0; j < N/2; j++) {
90                      seed = make_seed(current_iteration, j);
91                      M2[j] = random_on_interval(A, A*10, &seed);
92                  }
93                  // make a copy, shifting M2 one element to the right
94                  M2_copy[0] = 0;
95                  memcpy(&M2_copy[1], M2, sizeof(long double) * ((N/2)-1));
96
97                  /******************** MAP ************************/
98                  current_stage = STAGE_MAP;
99                  #pragma omp parallel for default(none) private(j) shared(
                        M1, N)
100                 for (j = 0; j < N; j++) {
101                     // operation #1, remember to convert to radians
102                     M1[j] = pow(sinhl((M1[j] * M_PI) / 180.0), 2);
103                 }
104
105                 #pragma omp parallel for default(none) private(j) shared(
                        M2, M2_copy, N)
106                 for (j = 0; j < N/2; j++) {
107                     // operation #3
108                     M2[j] = fabs(tanl(M2[j] + M2_copy[j]));
109                 }
110
111                 /******************** MERGE ***********************/
```

```
112                 current_stage = STAGE_MERGE;
113                 #pragma omp parallel for default(none) private(j) shared(
                        M1, M2, N)
114                 for (j = 0; j < N/2; j++) {
115                     // operation #1
116                     M2[j] = pow(M1[j], M2[j]);
117                 }
118
119                 /******************** SORT ************************/
120                 current_stage = STAGE_SORT;
121                 int elem_per_part = ((N/2) / omp_get_num_procs()) + 1;
122                 int start_locations[omp_get_num_procs()];
123                 #pragma omp parallel
124                 {
125                     int start_inc = elem_per_part * omp_get_thread_num();
126                     int finish_non_inc = (start_inc + elem_per_part) < N/2
                            ? start_inc + elem_per_part : N/2;
127                     start_locations[omp_get_thread_num()] = start_inc;
128                     for (int k = start_inc; k < finish_non_inc-1; k++)
129                     {
130                         int min_k = k;
131                         for (int l = k+1; l < finish_non_inc; l++)  {
132                             if (M2[l] < M2[min_k]) {
133                                 min_k = l;
134                             }
135                         }
136                         if (min_k != k) {
137                             long double temp = M2[k];
138                             M2[k] = M2[min_k];
139                             M2[min_k] = temp;
140                         }
141                     }
142                 }
143                 // we'll be reusing this variable
144                 memcpy(M2_copy, M2, sizeof(long double) * (N/2));
145                 for (j = 0; j < N/2; j++) {
146                     int min_interval = 0;
147                     for (int m = 1; m < omp_get_num_procs(); m++) {
148                         if ((start_locations[m] < N/2) && (M2_copy[
                                start_locations[m]] < M2_copy[start_locations[
                                min_interval]])) {
149                             min_interval = m;
150                         }
151                     }
152                     M2[j] = M2_copy[start_locations[min_interval]];
153                     M2_copy[start_locations[min_interval]] = INFINITY;
154                     start_locations[min_interval]++;
155                 }
156
157                 /******************** REDUCE **********************/
158                 current_stage = STAGE_REDUCE;
159                 int min_index;
160                 for (min_index = 0; M2[min_index] <= 0; min_index++) {}
161                 long double min = M2[min_index];
162
163                 #pragma omp parallel for default(none) private(j) shared(
                        M2, min, N, current_iteration) reduction(+:X)
164                 for (j = 0; j < N/2; j++) {
165                     if (isfinite(M2[j]) && (int)(M2[j] / min) % 2 == 0) {
166                         // remember to convert to radians
167                         X += sinl((M2[j] * M_PI) / 180.0);
168                     }
169                 }
```

```
170                 current_stage = STAGE_UNDEFINED;
171             }
172             T2 = omp_get_wtime();
173         }
174     }
175
176     long delta_ms = (T2 - T1) * 1000;
177     printf("%d\n", N);
178     printf("%ld\n", delta_ms);
179     printf("%.5f\n", X);
180
181     return 0;
182 }
```

4.2 Скрипт для компиляции программы lab4.c

```
gcc -O3 -Wall -lm -o lab4-seq lab4.c
gcc -O3 -Wall -Werror -fopenmp -lm -o lab4 lab4.c
```

4.3 Скрипт для запуска программы lab4.c

```
#!/bin/bash
set -eu

declare -a N_set=("3350" "4460" "5570" "6680" "7790" "8900" "10010" "11120" "12230" "13340" "14450" "15560" "16670" "17780" "18890" "20000" "21110" "22220" "23330" "24440" "25550" "26660" "27770" "28880" "29990" "31100" "32210" "33320" "34430" "35540" "36650")
TEMPLATE="lab4_template.csv"
OUTPUT="lab4_output.csv"
N_LINE=1
TIME_LINE_NUM_START=2
X_LINE_NUM_START=5
EXECUTABLE_SEQ="lab4-seq"
EXECUTABLE_OMP="lab4"

NUM_OF_TRIES=3

function add_to_line() {
  FILE_NAME="${1}"
  LINE_NUM="${2}"
  TEXT="${3}"
  sed -e "${LINE_NUM}s/\$/${TEXT};/" -i "${FILE_NAME}"
}

function execute() {
    EXECUTABLE="${1}"
    N="${2}"
    TIME_LINE_NUM="${3}"
    X_LINE_NUM="${4}"

    X=$(./${EXECUTABLE} ${N})
    y=(${X//$'\n'/ })
    min_time=${y[-2]}
    for (( meh=2; meh<=${NUM_OF_TRIES}; meh++ ))
    do
      X=$(./${EXECUTABLE} ${N})
      y=(${X//$'\n'/ })
      if [ "${y[-2]}" -lt "${min_time}" ]
      then
        min_time=${y[-2]}
      fi
    done
    add_to_line ${OUTPUT} ${time_line_num} ${min_time}
    add_to_line ${OUTPUT} ${x_line_num} $(sed 's/\./,/g' <<< ${y[-1]})
}

cp "./${TEMPLATE}" "./${OUTPUT}"

for N in "${N_set[@]}"
do
    add_to_line ${OUTPUT} ${N_LINE} ${N}
    time_line_num=${TIME_LINE_NUM_START}
    x_line_num=${X_LINE_NUM_START}
```

```
51
52      execute ${EXECUTABLE_SEQ} ${N} ${time_line_num} ${x_line_num}
53      ((time_line_num=time_line_num+1))
54      ((x_line_num=x_line_num+1))
55
56      execute ${EXECUTABLE_OMP} ${N} ${time_line_num} ${x_line_num}
57      ((time_line_num=time_line_num+1))
58      ((x_line_num=x_line_num+1))
59  done
60
61  exit 0
```