Obliczanie wartości π z wykorzystaniem wzoru Leibniza

Rafał Selewońko

9 grudnia 2013

1 Zadanie

Napisać program obliczający liczbę Pi do określonej precyzji (należy przetestować na komputerach jaka będzie rozsądna wartość - tak aby program wykonywał się kilka sekund). Następnie należy zrównoleglić go przy pomocy OpenMP (dla różnej liczby wątków). Dodatkowo należy policzyć i sporządzić wykresy przyśpieszenia i wydajności. Do oceny należy przedstawić raport z wykonania zadania zawierający wydruk programu + wykresy.

2 Rozwiązanie

pi.c

```c
// Copyright (c) 2013 Rafal Selewońko <rafal@selewonko.com>.
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <time.h>

double get_pi(int num_threads, int k_max);


int main(int argc, char *argv[]) {
 double pi;
 pi = get_pi(atoi(argv[1]), atoi(argv[2]) * 1000000.);

 //fprintf(stdout, "%s\n",
    "3.14159265358979323846264338327950288419716939937 5");
 //fprintf(stdout, "%1.48f\n", pi);
 return 0;
}

double get_pi(int num_threads, int k_max){
 double i;
 double pi;
 pi = 0.;
 for(i = 0; i <= k_max; i++){
  pi += 4 * (pow(-1, i)/((2 * i) + 1));
 }

 return pi;
}
```

main.c

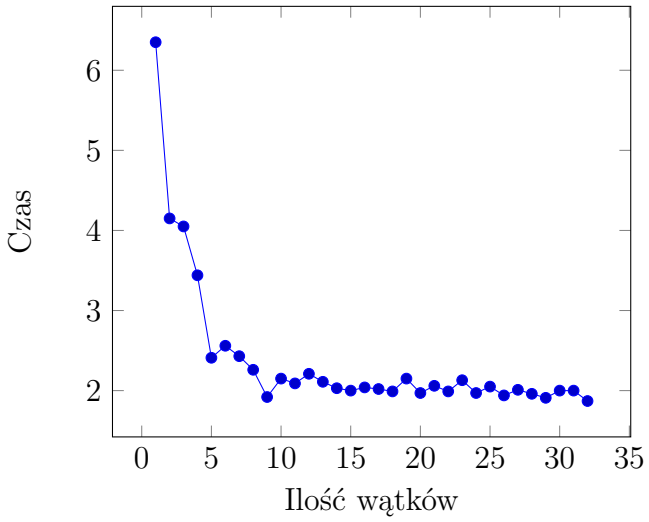
```c
// Copyright (c) 2013 Rafal Selewońko <rafal@selewonko.com>.
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <time.h>

double get_pi(int num_threads, int k_max);


int main(int argc, char *argv[]) {
 double pi;
 pi = get_pi(atoi(argv[1]), atoi(argv[2]) * 1000000.);

 return 0;
}

double get_pi(int num_threads, int k_max){
 double pi;
 double sum[num_threads];
 int i;
 int nthreads;

 omp_set_num_threads(num_threads);

 #pragma omp parallel
 {
  double i;
  int id = omp_get_thread_num();
  int nthrds = omp_get_num_threads();
  if (id == 0) {
   nthreads = nthrds;
  }
  sum[id] = 0.0;

  for(i = id; i <= k_max; i+=nthrds)
  {
   sum[id] += 4 * (pow(-1, i)/((2 * i) + 1));
  }
 }

 pi = 0.0;
 for(i = 0; i < nthreads; i++){
  pi += sum[i];
 }
 return pi;
}
```

test.sh

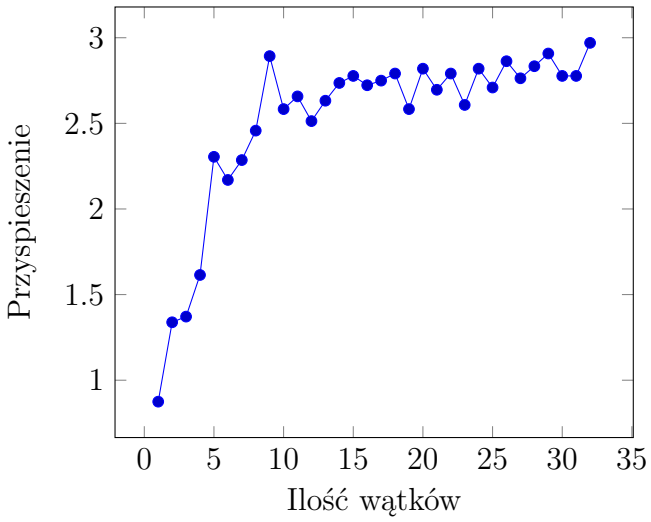
```
1 #!/bin/bash
2
3 for i in $(seq 1 64); do
4   t=`{ time -p ./pi $i 100 >/dev/null;} |& grep "real" | sed
        's/real //'`
5       echo "$i, $t"
6 done
```

3 Wyniki badań

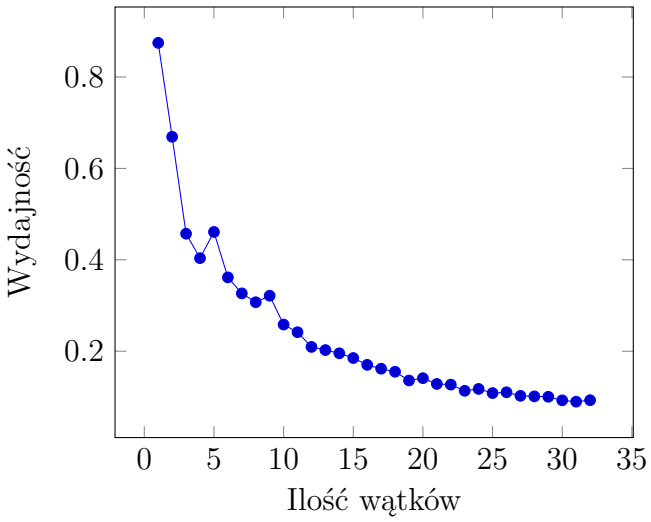
Czas wykonania implementacji jednowątkowej (plik pi.c) wyniósł 5,554 sekundy. Implementacja wielowątkowa (plik main.c) uruchomiona z jednym wątkiem wykonuje się 1.13 razy dłużej niż wersja wielowątkowa. Dla liczby wątków większej niż jeden widać bardzo duże przyspieszenie. Testy były wykonywane na maszynie 4 rdzeniowej z hyperthreadingiem dlatego przyspieszenie rośnie do 8 wątków a dla większej ilości utrzymuje się na podobnym poziomie. W tabeli przedstawiono czas wykonania implementacji wielowątkowej dla ilości wątków od 1 do 32.



Rysunek 1: Wykres czasu w zależności od liczby wątków.



Rysunek 2: Wykres przyspieszenia w zależności od liczby wątków.



Rysunek 3: Wykres wydajności w zależności od liczby wątków.

Wątki	Czas	Przyspieszenie	Wydajność
1	6.35	0.8746456693	0.8746456693
2	4.15	1.338313253	0.6691566265
3	4.05	1.3713580247	0.4571193416
4	3.44	1.6145348837	0.4036337209
5	2.41	2.3045643154	0.4609128631
6	2.56	2.16953125	0.3615885417
7	2.43	2.2855967078	0.3265138154
8	2.26	2.4575221239	0.3071902655
9	1.92	2.8927083333	0.321412037
10	2.15	2.583255814	0.2583255814
11	2.09	2.6574162679	0.2415832971
12	2.21	2.5131221719	0.2094268477
13	2.11	2.6322274882	0.2024790376
14	2.03	2.7359605911	0.1954257565
15	2.00	2.777	0.1851333333
16	2.04	2.7225490196	0.1701593137
17	2.02	2.7495049505	0.1617355853
18	1.99	2.7909547739	0.155053043
19	2.15	2.583255814	0.1359608323
20	1.97	2.8192893401	0.140964467
21	2.06	2.6961165049	0.1283865002
22	1.99	2.7909547739	0.1268615806
23	2.13	2.6075117371	0.1133700755
24	1.97	2.8192893401	0.1174703892
25	2.05	2.7092682927	0.1083707317
26	1.94	2.8628865979	0.110111023
27	2.01	2.7631840796	0.1023401511
28	1.96	2.8336734694	0.1012026239
29	1.91	2.9078534031	0.100270807
30	2.00	2.777	0.0925666667
31	2.00	2.777	0.0895806452
32	1.87	2.9700534759	0.0928141711

Tabela 1: Wyniki testu.