OpenMP – podstawy

Maciej A. Czyzewski
inf136698

Poznan University of Technology
Poland

1 Dane techniczne

Processor Name	Intel Core i5
Processor Speed	2.6 GHz
Number of Processors	1
Total Number of Cores	2
Logic Cores (max_threads)	4
L2 Cache (per Core)	256 KB
L3 Cache	3 MB
Memory	8 GB
Kernel Version	Darwin 17.7.0
Compiler	g++ 9.3.0

2 Rezultaty

Niektore wyjasnienia/spostrzezenia znaduja sie w implementacjach.

Task	`LOGIC`	`PHYSICAL`	`HALF`	average	ratio
PI1	0.496406	0.496406	0.496406	0.496406	1
PI2	0.720502	0.433100	0.518239	0.557280	0.89
PI3	6.735107	4.300983	1.378666	4.138252	0.11
PI4	**0.240313**	0.259092	**0.481288**	**0.326897**	**1.51**
PI5	0.244805	**0.250948**	0.488539	0.328097	**1.51**
PI6	0.531926	0.282501	0.505163	0.439863	1.12

- **PI1**: nasz referencyjny kod sekwencyjny, ze wzgledu na to ze u mnie zmienna `THREADS_HALF = 2 / 2 = 1`. Mozemy w trzeciej kolumnie porownac jak rozne implementacje rownolegle “dodaja” czas przetwarzania dla pojedynczego thread-u.
- **PI2**: wspoldzielenie wszystkich wartosci mocno spowalnia kod (zmienne globalne maja wolniejsze zapisy/odczyty), dodatkowo kod jest nie prawidlowy bo rozne watki uczestnicza w wyscigu (zly wynik Π).

- **PI3**: jest znaczaco wolniejsza poniewaz wieloktornie w petli wywolywana jest “atomic” ktory jest bardzo kosztowny.
- **PI4**: przesuniecie “atomic”-a z petli pozwala wywolania go tylko tyle razy ile mamy thread-ow (a wiec znaczace przyspieszenie).
- **PI5**: “reduction” dziala podobnie jak “PI4“ nie trzeba tylko przechowywac sum czesciowych wlasnej implementacji.
- **PI6**: tutaj mamy “false sharing” dlatego bedzie wolniej, jednak gdy sie znajdzie odpowiedni padding (uzywajac PI7), uzyskujemy takie same czasy jak w PI4/PI5.

3 PI7: obliczanie rozmiaru “cache line”

Postanowilem ze zaimplementuje troche inny ale rownowazny eksperyment. Program bedzie zmienial wartosc `memshift` ktora odpowiada za padding. Dodatkowo ustawiamy `schedule(static, 1) nowait` aby miec podzial pracy statyczny cykliczny oraz tylko 2 watki. W ten sposob mamy pewnosc ze beda na przemian pobierac swoje linie pamieci.

Napoczatku kiedy `memshift=0` wszystkie wartosci sa obok siebie w wektorze `vsum`, a to oznacza ze watki beda pobieraly ten sam wiersz (false sharing). Czas wykonania powinnien byc dluzszy niz w PI4. Teraz iterujac po coraz wiekszych paddingach probojemy znalesc pierwsza ktora dala znaczacy spadek czasu wykonania. To oznacza ze drugi watek zaczal korzystac z kolejnej linii. Na bazie tej odleglosci mozemy obliczyc dlugosc linii.

Z eksperymentu wynika ze `memshift=7` (bo czasy sa takie jak w PI4) a wiec dlugosc linii wynosi $8 \cdot 8$ bajtow $= 64$ na dane (co jest zgodne z moja architektura komputera).

```
#define THREADS_POLICY 2
#define NUM_STEPS 1000000000
#define MEMSHIFT 16
volatile double vsum[THREADS_POLICY * (MEMSHIFT + 1)] = {0};
// . . .
// iterate for best memshift (minimize time)
// . . .
#pragma omp for schedule(static, 1) nowait
      for (int i = 0; i < NUM_STEPS; i++)
        vsum[idx + idx * memshift] +=
            4.0 / (1. + (i + .5) * (i + .5) * (step2));
    }
```

3.1 Analiza memshift

```
threads_num=2

MEMSHIFT=0
  0 -> vsum[ 0]   1 -> vsum[ 1]
<time.h> time=8.232575
 <omp.h> time=4.170651

MEMSHIFT=1
  0 -> vsum[ 0]   1 -> vsum[ 2]
<time.h> time=7.778722
 <omp.h> time=3.921388

MEMSHIFT=2
  0 -> vsum[ 0]   1 -> vsum[ 3]
<time.h> time=7.801917
 <omp.h> time=3.932336

MEMSHIFT=3
  0 -> vsum[ 0]   1 -> vsum[ 4]
<time.h> time=7.810248
 <omp.h> time=3.949122

MEMSHIFT=4
  0 -> vsum[ 0]   1 -> vsum[ 5]
<time.h> time=8.097199
 <omp.h> time=4.086339

MEMSHIFT=5
  0 -> vsum[ 0]   1 -> vsum[ 6]
<time.h> time=7.827552
 <omp.h> time=3.954192

MEMSHIFT=6
  0 -> vsum[ 0]   1 -> vsum[ 7]
<time.h> time=7.848424
 <omp.h> time=3.952066

MEMSHIFT=7 (!)
  0 -> vsum[ 0]   1 -> vsum[ 8]
<time.h> time=5.211482
 <omp.h> time=2.619698

MEMSHIFT=8
  0 -> vsum[ 0]   1 -> vsum[ 9]
<time.h> time=5.438142
 <omp.h> time=2.752816

MEMSHIFT=9
  0 -> vsum[ 0]   1 -> vsum[10]
<time.h> time=5.214670
 <omp.h> time=2.624130

MEMSHIFT=10
  0 -> vsum[ 0]   1 -> vsum[11]
<time.h> time=5.413838
 <omp.h> time=2.740313

MEMSHIFT=11
  0 -> vsum[ 0]   1 -> vsum[12]
<time.h> time=5.261158
 <omp.h> time=2.648972

MEMSHIFT=12
  0 -> vsum[ 0]   1 -> vsum[13]
<time.h> time=5.504936
 <omp.h> time=2.802603

MEMSHIFT=13
  0 -> vsum[ 0]   1 -> vsum[14]
<time.h> time=5.456885
 <omp.h> time=2.780098

MEMSHIFT=14
  0 -> vsum[ 0]   1 -> vsum[15]
<time.h> time=5.604465
 <omp.h> time=2.878687

MEMSHIFT=15
  0 -> vsum[ 0]   1 -> vsum[16]
<time.h> time=5.297931
 <omp.h> time=2.668634
```