Przetwarzanie współbieżne

Laboratorium 5

Jan Bartula

# Zadanie 1

## Uruchomienie

Thread 0 a=7

Thread 0 a=8

Thread 0 a=9

Thread 3 a=7

Thread 2 a=7

Thread 2 a=8

Thread 2 a=9

Thread 1 a=7

Thread 1 a=8

Thread 1 a=9

Wyniki są rozsynchronizowane, każdy wątek bierze i dodaje do nieznanej wartości, ponieważ nie ma żadnej blokady.

# Zadanie 2

## Uruchomienie

|  |  |  |
| --- | --- | --- |
| Private:  Thread 0 a=0  Thread 0 a=1  Thread 1 a=0  Thread 1 a=1  Thread 2 a=0  Thread 2 a=1  Thread 5 a=0  Thread 4 a=0  Thread 3 a=0  Thread 6 a=0 | First Private:  Thread 0 a=7  Thread 1 a=7  Thread 0 a=8  Thread 1 a=8  Thread 6 a=7  Thread 5 a=7  Thread 3 a=7  Thread 2 a=7  Thread 2 a=8  Thread 4 a=7 | Shared:  Thread 0 a=7  Thread 6 a=7  Thread 4 a=7  Thread 2 a=7  Thread 2 a=11  Thread 3 a=7  Thread 0 a=8  Thread 1 a=7  Thread 1 a=15  Thread 5 a=7 |

Wielokrotne uruchamianie daje podobne rezultaty.

W przypadku private, zmienna jest zadeklarowana (jako 0) w każdym wątku z osobna i tam jest incrementowana.

W przypadku first private, zmienna jest zadeklarowana w każdym wątku z osobna (kopiując jej wartość) i tam jest incrementowana.

W przypadku shared, zmienna jest dzielona między wątkami (aczkolwiek dostęp do niej jest niekontrolowany)

# Zadanie 3

## Uruchomienie

static3 | 1 2 1 1 3 3 3 0 4 4 4 0 0 2 2

static | 3 4 6 0 1 5 1 6 3 5 0 0 4 2 2

dynamic3 | 0 2 0 0 5 2 2 5 5 3 3 3 6 6 6

dynamic | 5 2 1 4 4 4 4 4 4 4 4 4 6 0 3

W przypadku ustalenia rozmiaru porcji, każdy wątek wykonuje dokładną liczbę operacji równą jej rozmiarowi.

Podział pracy:

* Static, podział na zbiory o rozmiarze porcji, kolejno przydzielane poszczególnym wątkom
* Dynamic, wątki na bieżąco otrzymują kolejne porcje do wykonania aż do ich wyczerpania

Strategia dynamic wydaje się być bardziej losowa, w przypadku strategii static, często zdarza się kolejno przypisywanie zadań kolejnym wątkom.

W przypadku sztywnego ustawienia rozmiaru porcji, dokładnie kontrolujemy, ile operacji otrzyma każdy z wątków.

W przypadku domyślnej wartości rozmiaru porcji wybierana jest optymalna liczba operacji dla wszystkich wątków. (W zadaniu 4 widzimy, że czasami default działa lepiej, aczkolwiek zdarza się, że odgórne ustawienie porcji jest efektywniejsze)

# Zadanie 4

## Threads = 2

Static 3: 0.299320s

Static Default: 0.243750s

Dynamic 3: 0.861399s

Dynamic Default: 1.649487s

## Threads = 4

Static 3: 0.293241s

Static Default: 0.306228s

Dynamic 3: 0.892864s

Dynamic Default: 2.344841s

## Threads = 12

Static 3: 0.274165s

Static Default: 0.236188s

Dynamic 3: 0.749475s

Dynamic Default: 2.031401s

Jak możemy zauważyć strategia przydzielania wielkości porcji najczęściej przynosi dobre rezultaty (lecz nie zawsze), czasami sztywne ustawienie jest szybsze.

Możemy też zauważyć, że strategia STATIC jest szybsza od DYNAMIC.

Analizując czasy można uznać, że im więcej dajemy do decyzji OpenMP tym bardziej nieregularne mamy wyniki. Wynika z tego, że dla każdego z operacji dobrać odpowiednią strategie z przemyślaną porcją, jeżeli chcemy mieć równe, szybkie wykonywanie danej operacji. Zostawianie wyborów na łaskę domyślnych ustawień nie zawsze może przynieść oczekiwane rezultaty

## Zadanie 5

Funkcja licząca sumę kwadratów dowolnej liczby

long sum = 0;  
 int n = 1234;  
  
#pragma omp parallel for schedule(static) num\_threads(NO\_THREADS)  
 for (int i = 0; i < 500; i++) {  
 sum += n \* n;  
 }

## Wynik:

Sum: 11300

# Zadanie 6

## Threads = 1

Reduction: 15227560000000

Time: 0.013656s

Atomic: 15227560000000

Time: 0.061485s

Critical: 15227560000000

Time: 0.118466s

Lock: 15227560000000

Time: 0.122143s

## Threads = 2

Reduction: 15227560000000

Time: 0.006633s

Atomic: 15227560000000

Time: 0.266230s

Critical: 15227560000000

Time: 0.536720s

Lock: 15227560000000

Time: 0.800981s

## Threads = 4

Reduction: 15227560000000

Time: 0.004291s

Atomic: 15227560000000

Time: 0.222526s

Critical: 15227560000000

Time: 0.888884s

Lock: 15227560000000

Time: 1.230584s

## Threads = 8

Reduction: 15227560000000

Time: 0.002751s

Atomic: 15227560000000

Time: 0.192573s

Critical: 15227560000000

Time: 1.569812s

Lock: 15227560000000

Time: 1.119736s

## Threads = 12

Reduction: 15227560000000

Time: 0.003595s

Atomic: 15227560000000

Time: 0.239993s

Critical: 15227560000000

Time: 1.818721s

Lock: 15227560000000

Time: 2.333137s

## Podsumowanie

Analizując otrzymane wyniki, mogę stwierdzić, że najszybsze jest wykorzystanie strategia zabezpieczenia zmiennej jako reduction, niewiele gorsza jest z użyciem ATOMIC, CRITICAL i LOCK są najwolniejszymi z możliwych.

Wykonując wielokrotne uruchomienia wyznaczyłem, że najszybciej działa program przy użyciu 8 threads. Co ciekawe mam 6 rdzeniowy procesor dzielący się na 12 wątków, ale to nie w tej konfiguracji działa to najszybciej. (Aczkolwiek jest to zbyt prosty program, aby móc to traktować jako rzetelny test)