Mateusz Łąpieś (mateusz.lapies@student.pk.edu.pl)

## Sprawozdanie – Laboratorium 1

1. Pominięty w sprawozdaniu – załączony kod zawiera katalog roboczy.
2. Każda strategia wspomniana w definicji laboratoriów została zaimplementowana w postaci osobnej funkcji nazwanej według konwencji strategy\_(litera reprezentująca strategię)  
   Przykładowa implementacja procedury strategy\_a:

void strategy\_a(int iterations) {

int i;

#pragma omp parallel for schedule(static,3)

for(i=0;i<iterations;i++){ printf("thread %d, index %d\n",omp\_get\_thread\_num(),i); }

}

1. Poniżej listingi otrzymanych wpisów do konsoli dla poszczególnych przypadków:

thread 1, index 3

thread 1, index 4

thread 2, index 6

thread 2, index 7

thread 2, index 8

thread 0, index 0

thread 0, index 1

thread 0, index 2

thread 0, index 12

thread 0, index 13

thread 0, index 14

thread 3, index 9

thread 3, index 10

thread 3, index 11

thread 1, index 5



thread 3, index 12

thread 0, index 0

thread 0, index 1

thread 0, index 2

thread 0, index 3

thread 3, index 13

thread 3, index 14

thread 2, index 8

thread 2, index 9

thread 2, index 10

thread 2, index 11

thread 1, index 4

thread 1, index 5

thread 1, index 6

thread 1, index 7



thread 0, index 3

thread 0, index 4

thread 0, index 5

thread 0, index 9

thread 0, index 10

thread 0, index 11

thread 0, index 12

thread 0, index 13

thread 0, index 14

thread 2, index 6

thread 2, index 7

thread 2, index 8

thread 3, index 0

thread 3, index 1

thread 3, index 2



thread 0, index 0

thread 0, index 2

thread 0, index 3

thread 0, index 4

thread 0, index 5

thread 0, index 6

thread 0, index 7

thread 0, index 8

thread 0, index 9

thread 0, index 10

thread 0, index 11

thread 0, index 12

thread 0, index 13

thread 0, index 14

thread 2, index 1

|  |  |
| --- | --- |
| Strategia | Czas |
| a | 0,005454 |
| b | 0,007336 |
| c | 0,013642 |
| d | 0,007949 |

1. W tym podpunkcie zgodnie z poleceniem została wybrana operacja dodawania. Oczywiście w implementacji tuż przed uruchomieniem poszczególnych przypadków pobierany jest aktualny czas za pomocą funkcji omp\_get\_wtime(), natomiast po zakończeniu działania procedury tworzony jest wpis do konsoli zawierający różnicę zapisanej wartości czasu od aktualnie zwracanej wartości z funkcji omp\_get\_wtime().  
   Poniżej wykres zależności czasu działania zaimplementowanego algorytmu względem strategii dystrybucji zadań na poszczególne wątki.
2. Poniżej przedstawiona jest implementacja algorytmu dla obu przypadków (bez redukcji oraz z nią) oraz wykres zależności czasu pracy algorytmu od uruchomionego przypadku. Ilość iteracji (5000) jest przekazywane w postaci parametru dla obu procedur. Definicja ilości iteracji oraz mierzenie czasu pracy algorytmu są przeprowadzane w głównej procedurze (main).

void strategy\_non\_reduction(int iterations) {

int i; double sum = 0;

#pragma omp parallel for

for(i=0;i<iterations;i++){ sum += pow(2,2); }

}

void strategy\_reduction(int iterations) {

int i; double sum = 0;

#pragma omp parallel for reduction(+:sum)

for (i = 0; i < iterations; i++) { sum += pow(2, 2); }

}

|  |  |  |
| --- | --- | --- |
| Strategia | Czas | % zmiany |
| bez redukcji | 0,019711 |  |
| z redukcją | 0,005673 | 0,712191 |

1. Pominięte w sprawozdaniu – przykładowy program
2. Poniżej można znaleźć implementację algorytmu mnożenia dwóch macierzy o wymiarach NxM oraz MxP. Macierz wynikowa ma wymiary NxP.

for(i = 0; i < N; i++) {

for(j = 0; j < P; j++) {

for(k = 0; k < M; k++) {

output[i][j] += main[i][k] \* multi[k][j];

}

}

}

1. Zgodnie z treścią zadania zostały wybrane trzy zestawy rozmiarów macierzy
   1. N: 4, M: 3, P: 3 (czas: 0,000001)
   2. N: 16, M: 12, P: 12 (czas: 0,000014)
   3. N: 10, M: 8, P: 4 (czas: 0,000004)
2. Algorytm został zrównoleglony dla zewnętrznej i wewnętrznej pętli. Poniżej, pierwszy wykres przedstawia zależność czasu pracy algorytmu dla pętli zewnętrznej, natomiast następny pokazuje tą samą zależność dla pętli wewnętrznej. Dla każdego przypadku znajduje się również fragment kodu.

for(i = 0; i < N; i++) {

#pragma omp parallel for private(j) private(k) schedule(dynamic, 3)

for(j = 0; j < P; j++) {

int sum = 0;

for(k = 0; k < M; k++) {

sum += main[i][k] \* multi[k][j];

}

output2[i][j] = sum;

}

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Przypadek / Strategia | brak | a | b | c | d |
| N: 4, M: 3, P: 3 | 0,000001 | 0,000728 | 0,001706 | 0,000612 | 0,000921 |
| N: 16, M: 12, P: 12 | 0,000014 | 0,001807 | 0,001532 | 0,001775 | 0,000772 |
| N: 10, M: 8, P: 4 | 0,000004 | 0,000774 | 0,000736 | 0,000715 | 0,000754 |

for(i = 0; i < N; i++) {

#pragma omp parallel for private(j) private(k) schedule(dynamic, 3)

for(j = 0; j < P; j++) {

int sum = 0;

for(k = 0; k < M; k++) {

sum += main[i][k] \* multi[k][j];

}

output2[i][j] = sum;

}

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Przypadek / Strategia | brak | a | b | c | d |
| N: 4, M: 3, P: 3 | 0,000001 | 0,00077 | 0,000635 | 0,000682 | 0,000729 |
| N: 16, M: 12, P: 12 | 0,000014 | 0,001665 | 0,00191 | 0,001766 | 0,000716 |
| N: 10, M: 8, P: 4 | 0,000004 | 0,00078 | 0,000634 | 0,000671 | 0,00074 |

1. W oparciu o wykres z zadania można stwierdzić, że dla podanego przypadku strategia ‘static, rozmiar porcji = 3’ jest najszybsza.  
   Natomiast w zadaniu 5 redukcja działania dodawania pozwoliła na zmniejszenie czasu pracy algorytmu o ~71%.  
   Ostatnie zadania pokazały, że najkrótszy czas mnożenia dwóch macierzy osiąga się nie korzystając ze zrównoleglenia. Patrząc na strategie przy korzystaniu z równoległości dla obu rodzajów pętli, najbardziej uniwersalną strategią jest ‘dynamic’ z domyślną wielkością porcji.  
   Najlepszą strategią dla pętli zewnętrznej i dla stosunkowo małych macierzy jest ‘dynamic’, rozmiar porcji = 3’, dla pętli wewnętrznej w tych samych przypadkach jest nią ‘static’ z domyślną wielkością porcji. Zestaw największych wielkości macierzy najlepiej był obsługiwany przez przypadek ‘dynamic’.