Sprawozdanie z zadań 2-4

Programowanie Współbieżne 2014-11-22

Łukasz Ochmański 183566

Marcel Wieczorek 173526

Zadanie 2, 3

Dla aplikacji z ćwiczenia laboratoryjnego numer 2 wyznaczyliśmy następujące dane:

ω(n) = 4000000 operacji obliczeniowych ponieważ należy dodać tyle razy liczbę 1 do tablicy histogramu.

h(n,p) = 8000000 operacji we/wy ponieważ należy wykonać 4000000 odczytów z i 4000000 zapisów do pamięci operacyjnej.

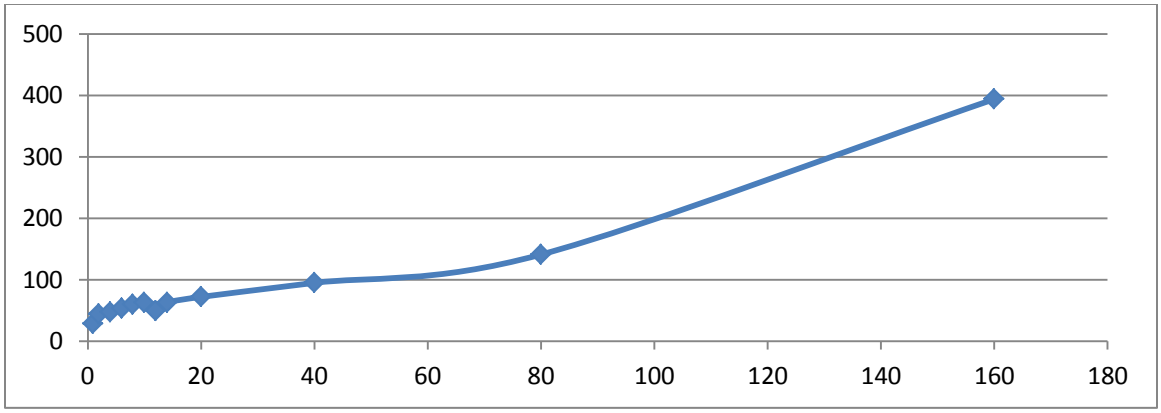
Zatem:

$$\eta(n,p) = \frac{\omega(n)}{\omega(n) + h(n,p)} = \frac{4000000}{4000000 + 8000000} = \frac{4}{12} = 33{,}33\%$$

sprawność programu równoległego realizującego to zadanie na maszynie z p procesorami wynosi zawsze 33,33%. W tym wypadku liczba procesorów nie ma znaczenia, gdyż liczba operacji wejścia/wyjścia jest zawsze stała jeśli korzystamy z jednego źródła danych. Powoduje to, że procesory muszą czekać w kolejce, aż uzyskają dostęp do pamięci. Ten problem jest przykładem zadania, które nie może być rozwiązane równolegle, ponieważ tablica bajtów oraz histogram są współdzielonym zasobem i muszą być stale odczytywane. Komputer osobisty może odczytywać tylko jedną komórkę pamięci na raz. Procesor wykonuje obliczenia kilkaset razy szybciej i 99% czasu czeka na wykonanie operacji przez inny procesor. Niewielki przyrost szybkości można uzyskać poprzez użycie pamięci podręcznej procesora. Jednak programista nie ma bezpośreniej możliwości manipulacji tą warstwą.

Dodatkowo do sprawozdania załączam rezultaty działania programu z ćwiczenia laboratoryjnego nr 2.

Liczba wątków	1	2	4	6	8	10	12	14	20	40	80	160
Średni czas w ms	27.78	44.32	46.52	52.8	59.45	62.3	48.57	63.04	72.23	95.05	141.11	393.81



Obliczenia sprawności:

liczba procesorów	**średni czas wykonania**	**wsp. przyspieszenia**	**wzg. wsp. przyspieszenia**
p	**T(n,p)**	**S(n,p)**	**S(n,p)/p**
1	27.78	1.00	100.00%
2	44.32	0.63	31.34%
4	46.52	0.60	14.93%
6	52.8	0.53	8.77%
8	59.45	0.47	5.84%
10	62.3	0.45	4.46%
12	48.57	0.57	4.77%
14	63.04	0.44	3.15%
20	72.23	0.38	1.92%
40	95.05	0.29	0.73%
80	141.11	0.20	0.25%
160	393.81	0.07	0.04%

Zadanie 4

Program 4 jest ulepszoną wersją programu 2 który zawiera 3 klasy:

-Zadanie04.java (klasa odpowiedzialna za uruchomienie programu, zawierająca main())
-MainThread.java (klasa odpowiedzialna za przetrzymywanie danych całego programu)
-Wątek.java (klasa, której instancje będą konstruowane w zależności od potrzeb od 1 do 160)

Aby uzyskać pełną liczbę punktów za zadanie 2 wystarczyło napisać program, który korzysta w niekontrolowany sposób z 3 zasobów: tablica bajtów, histogram, countDownLatch counter. Przypadkowo zblokowałem elementy 1 i 3, ale zapomniałem zsynchronizować histogram i wywołanie rokzazu histogram[i]++ przez kilka wątków powodowało ginięcie dodań. Zatem po zsumowaniu wszystkich liczb w histogramie zazwyczaj brakowało około 10% wartości z 4000000.

Okazało się, że instrokcja i++ jest tłumaczona na trzy atomowe rozkazy assemblerowe:

```
LOAD    @i, r0    ;load the value of 'i' into a register from memory
ADD     r0, 1     ;increment the value in the register
STORE   r0, @i    ;write the updated value back to memory
```

Rozwiązaniem tego problemu było zsynchronizowanie zmiennej histogram[], tak aby rozkazy się nie nakładały.

```
/**
 * Główna metoda służąca do podbicia podanego licznika w histogramie.
 * Przyjmuje numer indexu histogramu, który będzie podbity.
 * @param bajt
 * @return zwraca dotychczasową liczbę wystąpień podanego bajtu.
 */
public synchronized int increment(int bajt) {
      return ++this.histogram[bajt];
}
```

Metoda ochrony zasobów użyta w moich programach to mechanizm synchronizacji języka Java, który jest odpowiednikiem zwykłego zamka.

Dodatkowo synchronizacja zapewnia, że kompilator nie zamieni kolejności wykonywania instrukcji.