Sprawozdanie

Programowanie równoległe i rozproszone

Ćwiczenie 1 – Sygnały z pamięcią współdzieloną

12.03.2024

Cezary Skorupski, Adrian Nowosielski

1. Porównanie czasu działania programu bez pamięci współdzielonej z programem z pamięcią współdzielnością.

Uwaga. W implementacji sumowania wektorów została dodana sztuczna pętla, która wykonuje sumowanie 1000000 razy w celu zobaczenia różnicy w czasach wykonaniach – zwykłe sumowanie wektora o 1000 elementach jest zbyt proste i nakład potrzebny na ogarnięcie współdzielności pamięci i zwielowątkowienie programu jest zbyt duży w porównaniu do samych obliczeń, co przekładało się na to że czas wykonania naszego programu z współdzielnością pamięci wzrastał wraz ze zwiększaniem mu liczby procesów.

Oczywiście taka sama pętla została dodana również w programie bez współdzielności pamięci i wielowątkowości.

N – liczba procesów	Czas wykonania programu równoległego [s]	Czas wykonania programu nierównoległego [s]
1	1.892675	1.874052
2	0.947762	-
4	0.579507	-
6	0.476169	-
8	0.514962	-
16	0.410543	-

2. Uzasadnienie wyników

Jak widzimy nasze rezultaty są “praktycznie” zgodne z teorią. Widizmy, że dla N=1 czas jest minimalnie większy dla programu równoległego - wynika to z tego, że trochę czasu programowi zajmuje na skonfigurowanie pamięci współdzilonej i innych dodatkowych rzeczy wykorzystywanych w programie. Wraz ze wzrostem liczby procesów czas wykonania programu zmniejsza się. Jedynym wyjątkiem jest tutaj przypadek dla N=8, gdzie czas wykonania jest dłuższy niż w przypadku N=6. Możliwości dlatego tak jest może być wiele, najprawdopodobniej jest to związane z konkurencją o zasoby systemowe lub nakład tworzenia procesów był większy niż w przypadku N=6, co doprowadziło do wydłużenia czasu wykonania programu.

3. Opis implementacji

Uwaga: Mieliśmy problem z synchronizacją zakończenia się konfigurwania wątku tak, żeby proces macierzysty nie wysyłał sygnałów do procesów potomnych bez uprzednej konfiguracji. Na komputerze z systemem MacOs działało podejście ze zmienną n, którą inkrementowaliśmy wysyłając sygnał z potomka do rodzica i potem czekaliśmy w pętli aż wszystkie wątki skończą się konfigurować. Na innym komputerze niestety proces wpadał w nieskończoną pętle, więc zostaliśmy przy dodaniu metody sleep(maly czas), która działa na obu komputerach. Zostawiliśmy jednak implementację wraz z pętlą w kodzie.

1. Na początku programu pobieramy oczywiście potrzebne argumenty(liczba_procesów oraz ścieżka do pliku) i następnie sprawdzamy poprawność przekazanych argumentów.

2. Następnie czytamy wektor z podanego pliku i wczytujemy do zmiennej globalnej *vector wartości w wektorze, natomiast do zmiennej n - rozmiar wektora.

3. Następnie inicjalizowany jest zegar, który jest wykorzystywany do liczenia czasu wykonania sumowania oraz konfigurowany jest sygnał SIGUSR2. Sygnał SIGUSR2 jest wykorzystywany do wysyłania do rodzica informacji o tym, że sygnał został dobrze skonfigurowany. W obsłudze tego sygnału mamy inkrementowaną wartość zmiennej, która mówi nam ile procesów się skonfigurowało.

4. Następnie tworzone są procesy w pętli. W bloku procesu potomnego konfigurowany jest określony sygnał do obsługiwania sygnału SIGUSR1(sygnał jest wykorzystywany do wybudzenia funkcji, która liczy sumę w określonych indeksach w wektorze) oraz na koniec wysyłany jest sygnał do rodzica o tym, że dziecko zostało skonfigurowane. Została również dodana funkcja pause(), która czeka na otrzymanie sygnału od procesu macierzystego. W bloku procesu macierzystego ustawiamy numery_procesów, które zostały stworzone.

5. Tworzone są następnie dwie pamięci współdzielone - po pierwsze tworzona jest pamięć współdzielona dla range_indexow i przyłączamy pamięć do procesu głównego. Następnie wypełniamy tablicę indeksami, dzięki którym poszczególne procesy będą mogły obliczyć sumę. Analogicznie dołączamy do procesu głównego tablicę results oraz ustawiamy jej rozmiar na liczbę procesów.

6. Czekamy w pętli aż wszystkie procesy potomne się skonfigurują i następnie wysyłamy sygnał SIGUSR1 do wszystkich procesów potomnych do policzenia sumy dla poszczególnych indeksów.

7. W obsłudze sygnału SIGUSR1 oczywiście również podłączamy proces do wcześniej stworzonych pamięci współdzielonych na podstawie klucza, który generujemy za pomocą ftok i funkcji shmget i shmat. Po dołączeniu do procesu obu tablic obliczamy sumę dla określonego range_indeksów w tablicy i zapisujemy do tablicy results i na końcu odłączamy pamięci współdzielone od procesu.

8. Wracamy następnie do procesu macierzystego w którym oczywiście czekamy aż wszystkie procesy potomne skończą działanie i następnie zostaje nam zliczyć sumę

z tablicy results ze wszystkich policzonych w procesach sum. Wynik wypisujemy na konsolę.

9. Na koniec odłączamy pamięci współdzielone od procesu głównego oraz usuwamy stworzone pamięci współdzielone.