Antoni Pawlak

401480

Laboratorium 7: JAVA

Cel ćwiczenia:

- Przetwarzanie równoległe i rozproszone.
- Nabycie umiejętności pisania programów w języku Java z wykorzystaniem puli wątków
- Wykorzystanie wielowątkowych procedur języka Java

Wykonanie

Przygotowania

1. Utworzenie katalogu i pobranie plików ze stronh
2. Uruchomienie kodu i sprawdzenie poprawności działania na znanym przedziale funkcji podcałkowej

Wyliczanie całki

3. Utworzenie nowej puli wątków

```
ExecutorService executor = Executors.newFixedThreadPool(NTHREADS);
```

4. Utworzenie listy obiektów Future do przechowywania wyników

```
List<Future<Double>> list = new ArrayList<>();
```

5. Ustalenie parametrów algorytmu

```
int N=20;
double pt= Math.PI/N;
double start=0;
double koniec=Math.PI;
```

6. Dekompozycja problemu, gdzie mamy więcej zadań niż wątków

```
for(int i=0; i<N; i++){
 start=pt*i;
 koniec=pt*(i+1);

 if(koniec>Math.PI) koniec=Math.PI;

 Callable<Double> callable = new Calka_callable(start, koniec, dok);

 Future<Double> future = executor.submit(callable);
 list.add(future);
}
```

7. Oczekiwanie na wykonanie obliczeń

```
for(Future<Double> future_calka:list){
 try {
 wynik_calka_parallel += future_calka.get();
 } catch (InterruptedException | ExecutionException e) {
 }
}
```

8. Zabicie executora

```
executor.shutdown();
```

9. Sprawdzenie poprawności wyników

```
System.out.println("Finished all threads");
System.out.println("Calka rownolegle " + wynik_calka_parallel);
System.out.println("Calka sekwencyjnie: " + wynik_calka);
```

Sortowanie tablicy

10. Ustalenie parametrów algorytmu
```
int n = 100;
int[] result = new int[n];
```

11. Wylosowanie tablicy
```
Randomizer r = new Randomizer();
int[] randomTab = r.generateRandomArray(n,0,10);
```

12. Utworzenie ForkJoinPool
```
ForkJoinPool forkJoinPool = new ForkJoinPool();
```

13. Utworzenie nowego Divide Task
```
DivideTask task = new DivideTask(randomTab);
```

    1. Podzielenie tablicy na dwie
```
DivideTask task1 = new DivideTask(getSliceOfArray(arrayToDivide,0,middleIndex));
DivideTask task2 = new DivideTask(getSliceOfArray(arrayToDivide,middleIndex, n));
```

    2. Wywołanie fork()
```
 task1.fork();
 task2.fork();
```

    3. Oczekiwanie na wyniki
```
int[] tab1 = task1.join();
int[] tab2 = task2.join();
```

    4. Scalenie wyników
```
scal_tab(tab1, tab2, scalona);
```

14. Uruchomienie Divide task za pomocą invoke()
```
result = forkJoinPool.invoke(task);
```

15. Porównanie wyników
```
System.out.println(Arrays.toString(randomTab));
System.out.println(Arrays.toString(result));
```

Wnioski

Pula wątków

Mechanizm służacy do implementacji programów równoległych w środowisku Java. Mając do dyspozycji mniej wątków niż zadań, stajemy przed problemem odpowiedniego rozdysponowania zadań. Java udostępnia nam zautomatyzowany proces będący pulą wątku, gdzie nie odwołujemy się bezpośrednio do danego wątku, a bardziej do puli jako całości i zlecamy puli wykonanie zadania, nie zwracając szczególnej uwagi na to który wątek i kiedy to zadanie wykona.

Takie rozwiązanie niesie ze soba dużo zalet, takich jak wygoda programisty czy bezpieczeństwo pamięci. Niestety odejmuje pewnej elastyczności czy sprawia, że program jest mniej przewidywalny.

Zatem, wykorzystanie puli wątku pozwala na sprawne rozwiązanie problemu zrównoleglania większej ilości zadań niż wątków.

ForkJoin

Realizuje strategię work stealing, będącego realizacją wzorca recursive splitting.

Struktura fork-join pozwala przerwać pewne zadanie na kilku pracownikach, a następnie poczekać, aż wynik je połączy. W dużym stopniu wykorzystuje pojemność maszyny wieloprocesorowej. Poniżej przedstawiono podstawowe pojęcia i obiekty używane w frameworku fork-join.

Future

Specjalna klasa w środowisku Java pozwalająca na przypisanie do obszaru pamięci wyniku który zostanie obliczony dopiero w przyszłości, stąd jego angielska nazwa. Mając do dyspozycji takie struktury mamy możliwość projektowania programów równoległych, gdzie zmienne przechowujące wynik mogą oczekiwać na wykonanie procedury w wątku. Daje nam tą dużą swobodę w implementacji oraz pozwala na bardziej zdefiniowane zarządzanie procesem przebiegu wykonania kodu źródłowego.

Zatem, obiekt Future pozwala na synchronizację obliczeń równoległych