Lektion 4: Parallellisering av existerande kod

Målsättningen med denna laboration är att förstå de grundläggande premisserna för ”task-based parallelism” och Javas ramverk för att utföra parallella strukturerade beräkningar.

Som vanligt kommer denna uppgift att ta 15 minuter för vissa och andra kommer kanske inte att bli klara. Det beror på om man tänker rätt från början, hur van man är att läsa Javakod, etc.

Du kan redovisa mål L33 utifrån denna uppgift (på en labb). Du kan också utöka detta program och ta målen L34 och L35. Att profilera passar också bra i samband med detta.

Setup – uppvärmning

1. I filen `Quicksort.java` finns en relativt enkel implementation av sorteringsalgoritmen quicksort[1].

2. Mainmetoden anropar `qsortSequential(int[])` med en heltalsarray av slumptal.

3. Funktionen `qsortSequential(int[])` implementerar den sekventiella quicksortalgoritmen. Målet med denna övning är

4. Quicksort är en såkallad “divide and conquer”-algoritm, och just denna implementation är rekursiv. Det finns en punkt, när datat är tillräckligt litet, då det lönar sig att byta till en enklare algoritm. I filen `Quicksort.java` finns en metod `insertionSort(int[], int, int)`. Utöka `qsortSequential(int[], int, int)` så att `insertionSort` anropas, och slutför sorteringen, när datat är tillräckligt litet. Observera att med storleken på datat avses det intervall som utgörs av `start` och `end`. Du skall alltså vid något tillfälle när `end - start` är ett *tillräckligt litet tal* slutföra beräkningen genom att anropa `insertionSort` på detta intervall.

   Var (ungefär) ligger då denna punkt då det är fördelaktigt att byta till `insertionSort`? Det måste du ta reda på själv genom att *mäta körtiden.*

   Ta max 5 minuter till detta.

Lektionsppgiften

1. Skriv färdigt metoden `qsortParallel(int[])`, alltså en parallell version av Quicksort. Sorteringen beräknar ju inget värde utan utför sorteringen i den aktuella arrayen, så det är rimligt att använda sig av klassen `RecursiveAction`. Om du vill ha ett kodexempel kan du titta i `ParFrequencyCounter2.java`.

2. (Frivillig utökning) Precis som det i den sekvensiella versionen av quicksort finns en punkt då det lönar sig att byta till en enklare algoritm finns det i den parallella en punkt då konstanta faktorer (minnesallokering framförallt) gör att det lönar sig att byta till den sekvensiella versionen. Modifiera den parallella implementationen så att den sekvensiella anropas när datat är tillräckligt litet. Var (ungefär) ligger denna punkt då det lönar sig att byta till den sekvensiella versionen?

[1] `http://en.wikipedia.org/wiki/Quicksort`