Inleveropgave 3: Array Sum

Leerling: Sietse Neve

Leerlingnummer: 1810364

Github link: https://github.com/DrZisnotavailable/HPP.git

1. Inleiding

In dit rapport worden de benchmarkresultaten gepresenteerd van het parallelliseren van een array-sommetelling met behulp van OpenMP. Het originele programma (`arraySum.cc`) is sequentieel uitgevoerd, en vervolgens is een parallelle versie (`ompArraySum.cpp`) gemaakt met gebruik van een `parallel for` en `reduction`-patroon.

De metingen zijn uitgevoerd op vier datasets (`10k.txt`, `100k.txt`, `1m.txt`, `10m.txt`) en met 1, 2, 4 en 8 threads. De code is uitgevoerd via een aangepaste Makefile zonder gebruik van extra scripts. Zowel de sequentiële als de parallelle versies zijn handmatig uitgevoerd via `mingw32-make run_seq` en `mingw32-make run_omp`.

2. Makefile-aanpassing voor OpenMP

De relevante aanpassing in de Makefile voor de parallelle uitvoering met meerdere threads zag er als volgt uit:

```
run_omp: ompArraySum 10k.txt 100k.txt 1m.txt 10m.txt
    @echo "== Parallelle versie met verschillende aantallen threads =="
    set OMP_NUM_THREADS=1 && ompArraySum 10k.txt
    set OMP_NUM_THREADS=2 && ompArraySum 10k.txt
    set OMP_NUM_THREADS=4 && ompArraySum 10k.txt
    set OMP_NUM_THREADS=8 && ompArraySum 10k.txt
    ... (herhaald voor alle bestanden)
```

3. Benchmarkresultaten

Bestand	Threads	Tijd (s)
10k.txt	1	0.000000
10k.txt	2	0.000000
10k.txt	4	0.001000
10k.txt	8	0.002000
100k.txt	1	0.000000
100k.txt	2	0.000999

Bestand	Threads	Tijd (s)
100k.txt	4	0.000999
100k.txt	8	0.000999
1m.txt	1	0.001000
1m.txt	2	0.000999
1m.txt	4	0.002000
1m.txt	8	0.002000
10m.txt	1	0.000999
10m.txt	2	0.002000
10m.txt	4	0.001999
10m.txt	8	0.002000

4. Uitleg van `ompArraySum.cpp`

De parallelle versie van het programma gebruikt OpenMP om de som van een array over meerdere threads te verdelen. De kern van de aanpassing zit in de functie `sumArray_parallel()` in `ompArraySum.cpp`. Daar wordt het volgende gedaan:

1. `#pragma omp parallel for reduction(+:result)`
   Deze directive vertelt OpenMP om een `for`-loop parallel uit te voeren. Elke thread krijgt een deel van de array.
   De `reduction(+:result)` zorgt ervoor dat iedere thread een lokale `result` gebruikt, en deze na afloop worden opgeteld.

2. `a->at(i)` wordt gebruikt om waarden op te halen uit de gedeelde array (`const std::vector<double>`).

Relevante codefragment:

```
double sumArray_parallel(input_array a) {
    double result = 0.0;
    size_t n = a->size();

    #pragma omp parallel for reduction(+:result)
    for (size_t i = 0; i < n; i++) {
        result += a->at(i);
    }

    return result;
}
```

De rest van het programma leest het inputbestand in via `readArray()` (ongewijzigd t.o.v. de sequentiële versie) en meet de tijd met `omp_get_wtime()`.