Project 1
Odd-Even Sorting
Advanced multi core programming, DV2575

Filip Pentikäinen

22 november 2017

Innehåll

Implementation

Iteration 1: generating a random sequence of integers

Creating a function generating random numbers is quite easy in C using the rand() function. Using the defined constant *DATA_AMOUNT* the data[] array is filled as seen in Figure 1.

```
void FillArray(int data[]) {
    srand(time(NULL));
    for (int i = 0; i < DATA_AMOUNT; i++)
       data[i] = (rand()%DATA_AMOUNT) + 1;
}
```

Figur 1: Function responsible for filling array with random integers

Iteration 2: single thread based odd-even sorting

Creating an outer loop that runs as many times as there are elements and creating an inner loop that loops for every other element in the array will create an algorithm with $O(n^2)$ complexity. This will ensure that every odd/even element is compared to the element to its right and swapped accordingly.

```
int* SortCPU(int* data) {
   int* dataSorted = data;
   for (int i = 0; i < DATA_AMOUNT; ++i)
      for (int j = 0; j < DATA_AMOUNT-1; j +=2){
         pos = j + i%2;
         if(data[pos] > data[pos+1] && (pos+1) < DATA_AMOUNT) {
            int temp = data[pos];
            data[pos] = data[pos + 1];
            data[pos + 1] = temp;
         }
      }
   return dataSorted;
}
```

Figur 2: Algorithm for single threaded sorting

Iteration 3: parallel odd-even sorting by using CUDA

Performance

```
__global__
void OddEvenSort(int* data_d, int iterAmount) {
   int id = threadIdx.x + blockDim.x * blockIdx.x;

   if ((id *= iterAmount) < DATA_AMOUNT) {
      int mod = 0;

      for (int i = 0; i < DATA_AMOUNT; ++i) {
         __syncthreads();
         for (int j = 0; j < iterAmount; j += 2) {
            int pos = id + mod + j;
            if ((pos + 1) < DATA_AMOUNT) {
               int reg1 = data_d[pos];
               int reg2 = data_d[pos + 1];
               if (reg1 > reg2) {
                  data_d[pos] = reg2;
                  data_d[pos + 1] = reg1;
               }
            }
         }
         if (mod == 0) mod = 1;
         else mod = 0;
      }
   }
}
```

Figur 3: Algorithm for multi threaded sorting using CUDA