Concurrency and Parallel Programming

CUDA assignment

Maarten de Jonge

December 12, 2012

Wave Equation

Implementation

The implementation was done similarly to the previous assignments; each thread gets a slice of the wave to simulate, although in this case the slices are markedly smaller than previously due to the enormous amount of threads. The amount of threads per block has been set at 512, while the number of blocks is taken as the number of simulated points divided by the number of threads (so ideally, every thread would only simulate a single point). Because the device can only run 65535 blocks in parallel, the number of blocks is limited to the interval $[1, 65535]$.

The time iterations are synchronized by only handling a single iteration in the kernel, leading to one kernel invocation per time step.

Results

Table 1 shows the results of the old experiments using pthreads and OpenMP. For the CUDA experiments, shown in table 2, each trial has been run 10 times and averaged to account for relatively large fluctuations in runtime.

The results are quite clear; compared to the previous implementations, CUDA is really really fast.

Table 3 shows the effect of the number of threads per block, from 8 up to the maximum of 1024. i_max and t_max are kept constant at 10000000 and 500, respectively. These results do not take fluctuations into account. There is definitely a constant increase in speed with higher numbers of threads, but there are clear diminishing returns. Because the number of blocks in 1 dimension is limited to 65535, a low number of threads will equal a larger chunk of the wave to be handled per thread, and thus less parallelism.

Reduction

The first version of the reduction algorithm uses a simple architecture with 1 block and 512 threads per block. Each thread reduces a chunk of the input array, reducing the array to one of 512 elements. Then 1 of threads further reduces that array to a single value. Sample output is as follows, using a randomly filled array of 10 million doubles:

i_max	t_max	num_threads	time pthreads	time OMP
1000	1000000	1	42.4294	2.7425
1000	1000000	2	31.5108	42.3953
1000	1000000	4	80.9137	20.493
1000	1000000	8	167.284	23.8034
1000	1000000	16	350.395	21.2966
10000	500000	1	69.3228	12.4777
10000	500000	2	44.066	85.0008
10000	500000	4	41.1647	47.4733
10000	500000	8	85.3845	36.9657
10000	500000	16	183.268	33.3651
1000000	5000	1	52.108	15.6997
1000000	5000	2	25.6232	83.3668
1000000	5000	4	13.1806	42.1536
1000000	5000	8	7.08639	27.3887
1000000	5000	16	7.06076	29.6989
10000000	500	1	51.4277	15.1485
10000000	500	2	25.8168	83.1574
10000000	500	4	13.0537	41.439
10000000	500	8	7.21982	27.4845
10000000	500	16	7.16489	36.9947

Table 1: The raw test data, where “i_max” is the number of simulated points on the wave and “t_max” is the amount of iterations.

i_max	t_max	time taken (s)
1000	1000000	4.2354
10000	500000	2.3655
1000000	5000	1.6818
10000000	500	1.9001

Table 2: The CUDA implementation at various parameter values.

threads per block	time taken (s)
8	15.0367
16	9.2053
32	5.15237
64	3.27603
128	2.457
256	2.28942
512	2.03195
1024	1.65433

Table 3: 10000000 wave points over 500 iterations, using various numbers of threads per block.

```
Parallel max: 999.999711
Time taken: 0.397627 seconds

Sequential max: 999.999711
Time taken: 0.017945 seconds
```

Although this version is correct, it's slower than doing it sequentially. Apparently, a higher degree of parallelism is required to be worth the overhead.