Parallel Radix Sort on OpenCL

Hennadiy Yatskov
Nico Mürdter

Karlsruhe Institute of Technology, Karlsruhe, Germany
hennadiy.yatskov@student.kit.edu
nico.muerdter@student.kit.edu

Abstract. TODO

1 Algorithm

The algorithm is based on an open source implementation of P. Helluy [1]. It is seperated into three different phases, which are executed consecutively.

Before we come to the detailed explanation of the algorithm, we first need to define several values. $K(j)$ for $j = 0 \dots N-1$, represents the non-sorted input values which consist only of integers. Each of these integers $K(j)$ is between 0 and $2^b - 1$ and will be called a *key* in the further explanation. Furthermore is the Radix represented by $R = 2^r$ where r is the number of bits necessary representing R. We suppose thath b is divisible by r and so the number of passes is denoted by $p = b/r$. This assumption can be satisfied by an appropriate definition of r, or the extension of the input list by adding keys which represent the biggest possible values. This extension is done with maximum values, so that the sorting is only influenced in the first phase of the algorithm, which will be explained in the next sections.

Each pass $q = 0 \dots p-1$ of the algorithm consists in sorting the list according to the q^{th} digit (in base R) $K(j)_q$. It is important to say, that each pass sorts the corresponding elements in a stable manner. So the work done in each pass is not corrupted by previous passes and will not corrupt following passes.

1.1 Histograming

This first phase of the algorithm is in charge of calculating the so called *Histograms*. A histogram represents the number of occuring Radix in the given list of elements K. To do this fully parallel, the porcessing units are seperated in Groups G with Items I, where an Item represents a Processor. So based on a GPU Architecture, the total amount of available Processing Units is GI. As said earlier, for the explanation we can suppose, thath $N = GI$. If this is (most likely) not the case in a real scenario, the data will be extended by keys which represent the biggest possible values, so that it fits the assertion.

1.2 Scanning

1.3 Reordering

2 Implementation Details

2.1 Main Routine

2.2 Kernel

3 Experimental Results

Your hardware.

What do you benchmark.

Running time 1 and speedup plots 2 (for each generator, 64-bit integer and 32-bit floating point (not for non-comparative integer sorting algorithms).

Interpretation.

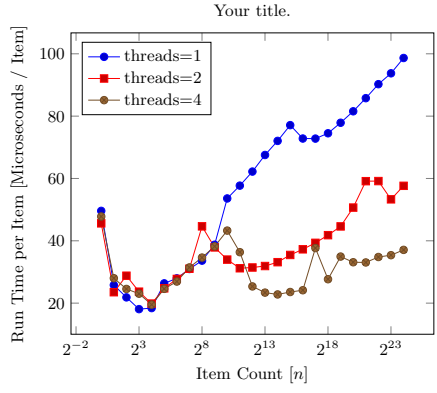

Fig. 1. Running times of `std::sort` with uniform input. Mean of 5 iterations.

References

1. Helluy, P.: A portable implementation of the radix sort algorithm in opencl (2011), https://github.com/phelluy/ocl-radix-sort

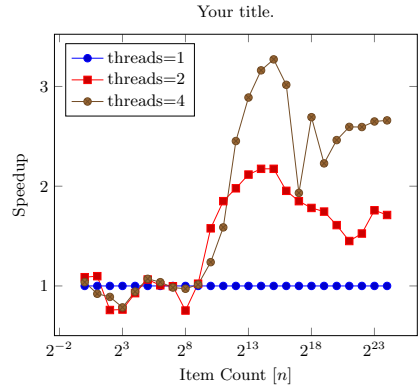

Fig. 2. Speedup of `std::sort` with uniform input. Mean of 5 iterations.

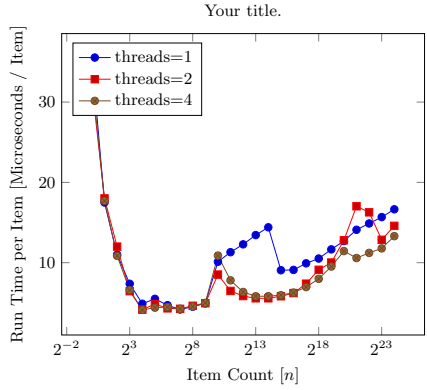

Fig. 3. Running times of `std::sort` with zero input. Mean of 5 iterations.

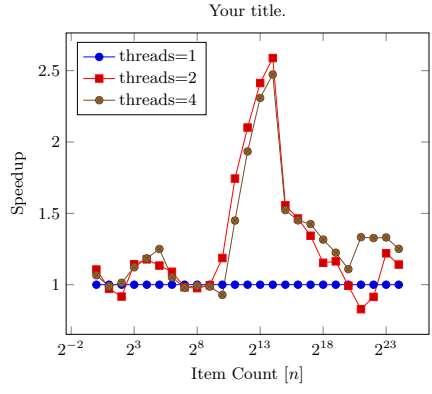

Fig. 4. Speedup of `std::sort` with zero input. Mean of 5 iterations.