

Parallel Sorting on GPU Clusters

Thomas Kerwin

June 8, 2007

1 Introduction

It is becoming more common to install modern graphics cards on small to medium size commodity clusters. In addition to applications such as display walls and CAVE environments, graphics cards can be used as dedicated coprocessors that can run certain parallel algorithms very quickly.

Sorting has been long recognized as an important algorithm in terms of both mathematical analysis and a way to judge overall performance of computer systems[8]. I show in this report that combining two different sorting algorithms for a hierarchical parallel system gives a significant increase in performance for large numbers of keys. The parallel system used has two major parallel structures. The system is a commodity PC cluster running Linux and connected with an InfiniBand network. Each node in the cluster has a NVIDIA GeForce 7900 video card connected to the system board via PCI Express x16. For more information on the system see [1].

2 Using the GPU for sorting

The challenge of optimizing sorting algorithms for GPU architectures is still a topic of ongoing research. Most sorting algorithms used by implementations designed to run on GPUs are based on sorting networks, such as the odd-even sort and the bitonic sort. Odd-Even merge sort was used by Lutz Latta[6] in developing a large particle simulation implemented fully on the GPU. Recent implementations based on bitonic sort has been shown to outperform odd-even sort on GPUs. Kipfer et al.[5] in demonstrating their GPU particle system showed an improvement of the bitonic sort implemented by Purcell et al.[7] that was used for GPU based photon mapping. Naga Govindaraju et al. [4] have developed an bitonic sort implemented on GPUs for general numerical sorting. They have released their code as a C++ library

called GPUSort. I use this package for my local sorting function. Version 2 purports to have slightly better performance, but I had difficulty getting it to work correctly, so all tests are done with version 1.

2.1 Limitations

GPU sorting in general has several limitations. All math on current GPUs is single precision. This makes it impossible to perform sorting on 64-bit floating point numbers without double precision emulation on the sorting software running on the GPU. NVIDIA does have plans to release GPUs with native double precision floating point capabilities in “late 2007” [2].

Another limitation is the texture memory and texture bind size for the GPU. Current GPUs use a local memory storage, and all elements to be sorted must be uploaded to this local memory. The sort keys are stored in a RGBA 4-channel floating point texture, with one channel used for the sort keys and the other channels used for data associated with the key, such as a unique id to record stored elsewhere on the system. With a texture memory size of 256MB, this gives a maximum of 16M keys.

Higher numbers of keys could be sorted by a variant of merge sort that partitioned the data into the minimum number of subsets so that each subset of keys would fit in memory on the GPU. This would require a k -way merge on the CPU which would negatively impact the total performance of the sort.

3 Sample sort with MPI

Sample sort is a multi-stage sort that is easily applicable to a cluster parallel system first proposed by Frazier and McKellar [3]. It consists of four major steps:

1. Local sorting of $\frac{n}{p}$ elements on each node.
2. Determine partitions:
 - (a) Selection and transmission of k samples on each node to the root
 - (b) Based on the $k * p$ total samples, estimate partitions that will evenly distribute the data to the nodes.
 - (c) Broadcast p partition keys to each node.
3. Each node sends all elements in the i^{th} partition to node i .

Elements	Sample (qsort)	Sample (GPUSort)	sequential qsort
10,000	0.02009	0.1844	0.0027
100,000	0.01396	0.2121	0.0323
1,000,000	0.0886	0.2451	0.3923
10,000,000	0.8595	0.5362	4.6402
100,000,000	9.99	5.507	53.656

Table 1: Comparison between three sorting methods. Sample sort is using eight nodes. Times are in seconds.

4. Local sorting of received data by each node.

In my implementation, I chose to use 32 as the value of k , which gave me satisfactory load balancing. My elements to be sorted were generated from a uniform distribution; non-uniform distributions may require a higher sampling rate.

Some implementations include the distribution of data at the beginning to all the nodes before the first local sorting and the merging of data after the second local sorting in the definition of the sort. These steps are not necessary for all uses of sorting, especially if the results of the sort are to be used as input to a further parallel algorithm. For this reason, I do not include these steps in my implementation and they are not taken into account in the timing results.

4 Results

I compared the running time of a normal sequential sort using the GNU implementation of `qsort()` to the implementation of sample sort using `qsort` as the local sort and using the GPU as the local sort. See Table 1 for the results for eight processors.

For small element sizes, the overhead from the data transmission dominates the algorithm and so the sequential sort is the best choice. As the element size increases, the `qsort()`-based sample sort performs better than the GPU-based sample sort until the total element count passes 10,000,000. Before that point, the increased performance of the graphics card can not overcome the performance penalty incurred by transmission on the bus from main memory to texture memory and back. At 100,000,000 elements, the GPU-based sample sort is 1.8 times faster than the sample sort using a CPU local sort and 9.74 times faster than the sequential CPU sort.

Elements	2 Nodes	4 Nodes	8 Nodes	16 Nodes
10,000	0.1909	0.1918	0.1844	0.2129
100,000	0.1982	0.1947	0.2121	0.2297
1,000,000	0.3973	0.2722	0.2451	0.2352
10,000,000	1.5499	0.8504	0.5362	0.4836
100,000,000	N/A	10.903	5.507	3.2781

Table 2: Time for Sample GPUSort for different node sizes.

The GPU sample sort was tested on two to sixteen nodes to examine scaling. The results are seen in Table 2. A timing on two nodes could not be completed, since the maximum values able to be sorted in the current implementation of the GPU local sort is 2^{24} , which is less than the 50,000,000 required by the largest test. For total element size below 1,000,000, the overhead of communication makes adding more nodes a performance loss. For element sizes above that mark, adding more processors lowers the total time of the sort, but the increase in speed is sub-linear.

The ideal division of work is to send each node a number of elements that can be sorted entirely in the GPU’s memory. Given a element size of n and maximum texture memory usable for the sorting algorithm of M , the optimum number of nodes to use for most efficient use of the computing resources is $\lfloor \frac{n}{M} \rfloor$.

5 Conclusion

By leveraging consumer graphics hardware, performance on large scale parallel sorts on clusters can be improved. However, the overhead generated by the transmission of data to the GPUs makes this type of sort inefficient for small numbers of elements.

References

- [1] NSF research infrastructure project. http://www.cse.ohio-state.edu/nsf_ri/.
- [2] NVIDIA Corporation. Cuda toolkit 0.8 release notes. <http://developer.nvidia.com/object/cuda.html>.
- [3] W. D. Frazer and A. C. McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *J. ACM*, 17(3):496–507, 1970.

- [4] Naga K. Govindaraju, Dinesh Manocha, Nikunj Raghuvanshi, and David Tuft. Gpusort: High performance sorting using graphics processors. <http://gamma.cs.unc.edu/GPUSORT/>.
- [5] Peter Kipfer, Mark Segal, and Rüdiger Westermann. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM Press.
- [6] Lutz Latta. Building a million particle system. In *Proceedings of Graphics Hardware*, 2004.
- [7] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50. Eurographics Association, 2003.
- [8] Kurt Thearling. An evaluation of sorting as a supercomputer benchmark. Technical Report RNR-93-003, NASA Advanced Supercomputing Division, 1993.