Abstract.

One of the difficulties faced when using a general purpose graphing processing on memory intensive tasks, is the considerable amount of time taken to transfer data from a CPU. Such is the case when one tries to upload a projection index from a CPU onto a GPU. One way to minimize the amount of data that needs to be transferred is through the use of compression. In this paper a Run Length Encoding (RLE) compression scheme will be used to minimize the size of the data needed to be transferred.

The idea is to compress a projection index using the RLE scheme and then uncompress it within the GPU with a parallel prefix sum scan which will determine how to allocate and copy the uncompressed projection index within the GPU.

To conclude, a benchmark test will compare whether there's any improvement in performance by loading compressed and uncompressing, as opposed to loading an uncompressed index.

References (for now)

* Gosink, L., Kesheng Wu, E. Wes Bethel, John D. Owens, Kenneth I. Joy: Data Parallel Bin-Based Indexing for Answering Queries on Multi-core Architectures. SSDBM 2009: 110-129
* Gosink, L., E. Wes Bethel, John D. Owens, Kenneth I. Joy. Bin-Hash Indexing: A Parallel GPU-Based Method For Fast Query Processing. IDAV (2008)
* Wu, K., Otoo, E., Shoshani, A.: On the performance of bitmap indices for high cardinality attributes. In: Proc. of VLDB, pp. 24–35 (2004)
* O’Neil, P.E., Quass, D.: Improved query performance with variant indexes. In: Proc. of SIGMOD, pp. 38–49 (1997)
* Nvidia Programming Guide
* [Mark Harris, Parallel Prefix Sum (Scan) with CUDA](http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/scan/doc/scan.pdf)

Notes

Transfer a compressed projection index column and uncompress it in the GPU. The compression scheme used would be RLE (Run Length Encoding) and the algorithm for uncompressing in parallel within the GPU should be the Prefix Sum algorithm.

Example: A5B3A12 Prefix sum.

One of the difficulties faced when using a general purpose graphing processing on memory intensive tasks, is the considerable amount of time taken to transfer data from a CPU. Such is the case when one tries to upload a projection index from a CPU onto a GPU. One way to minimize the amount of data that needs to be transferred is through the use of compression. In this paper a Run Length Encoding (RLE) compression scheme will be used to minimize the size of the data needed to be transferred.

The idea is to compress a projection index using the RLE scheme and then uncompress it within the GPU with a parallel prefix sum scan which will determine how to allocate and copy the uncompressed projection index within the GPU.

To conclude, a benchmark test will compare whether there's any improvement in performance by loading compressed and uncompressing, as opposed to loading an uncompressed index.

I want to use the CUDPP libraries because they will simplify a lot of the work that needs to be done. The CUDPP libraries provide primitives that allow me to simplify a lot of the work. Interesting enough they have a common.mk file which must be modified from my make file, and is just simply included at the end of my makefile.

I just gave it some thought and I think that the apps folder is where I should put the application I am trying to build as to keep the other stuff nicely untouched, as to not mess anything up. Yeah.. Thinking about it I should have just checked out a copy of the cudpp library and then export it.. and then add it to my repository. After that I will probably want to make my own app folder and based on the simpleCUDPP.. as a duplicate, and then add it to the new repository, and then add the changes. It would be neat that way as a final deliverable for the bastard.

I have worked a bit more on the project, and right now I’m having problems with the memory getting filled too quickly. I also notice that when I try to run the GPUTimer, it doesn’t work well.

So I need to come up with a hypothetical example as for the algorithm to work: been thinking this one:

1… 32

A… symbol

528 is the size of the uncompressed array.

Final result should be:

ABBCCCDDDD… etc.