Parallel Data-processing on GPGPU

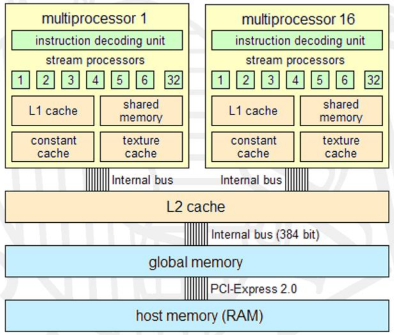
Radim Vansa, Charles University in Prague, Faculty of Mathematics and Physics

Motivation

Modern graphic cards are no longer limited to image rendering and geometric calculations but also allow parallel processing of non-graphical data.
In practice, these general-purpose GPUs can be used in database management systems as co-processors, accelerating certain time-consuming tasks.

Challenges

The execution model used on GPUs called SIMT is substantially different from the common one used on multi-core systems.
In order to achieve best performance, the programmer must effectively use the system of caches and adhere optimal memory access patterns.


multiprocessor 1
instruction decoding unit
stream processors
1 2 3 4 5 6 32
L1 cache
shared memory
constant cache
texture cache
multiprocessor 16
instruction decoding unit
stream processors
1 2 3 4 5 6 32
L1 cache
shared memory
constant cache
texture cache
Internal bus
Internal bus
L2 cache
Internal bus (384 bit)
global memory
PCI-Express 2.0
host memory (RAM)

Scheme of GPGPU memory hierarchy

Implemented Algorithms

- Merge-join
- Binary search
- Interpolation search
- Generalized quadratic search
- Linear hashing
- Cuckoo hashing
- Universe reduction (bucketting)
- Set reduction using Bloom filters
- Quicksort
- Bitonicsort
- Mergesort

Objectives

We have studied two problems often solved in database systems:

- sorting, used for index creation, duplicities removal or grouping
- set intersection, basically the table join, with either sorted or unordered sets

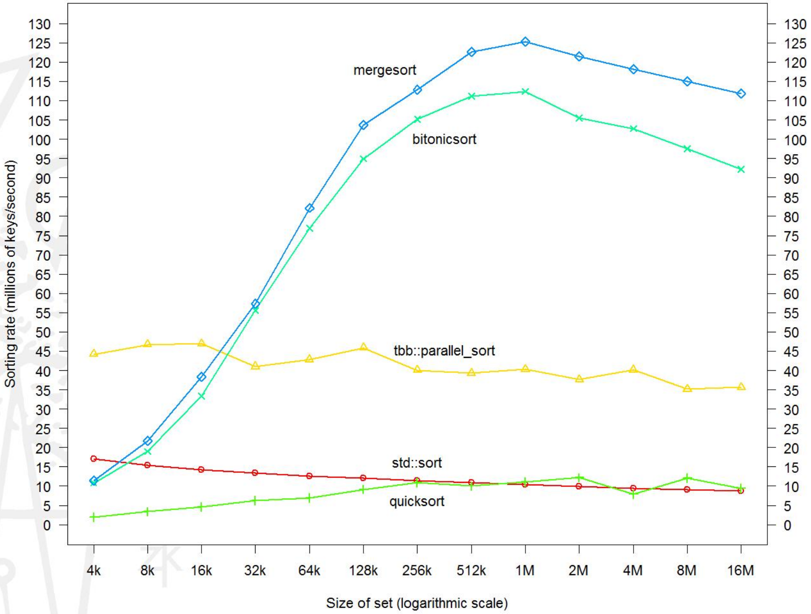
OpenCL

OpenCL framework was used as an open standard for parallel programming of heterogeneous systems.
This allowed us to write GPGPU programs portable across graphic cards from different vendors.

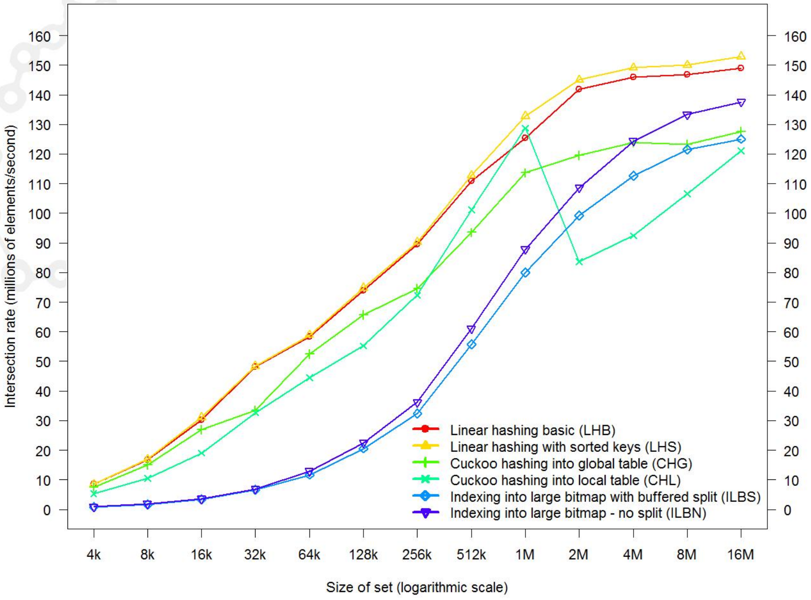
Results and Conslusion

We have implemented and benchmarked parallel versions of many algorithms, bringing an extensive comparison of various approaches to our problems.
Significant speedup was achieved compared to CPU-based solutions - for example our mergesort implementation was up to 12.4x faster than std::sort and up to 3.1x faster than tbb::parallel_sort.
Our GPU set intersection algorithm was also more than twice faster than optimized parallel CPU algorithm.

Comparison of GPGPU and CPU algorithms for sorting



GPGPU algorithms for unordered set intersection


Intersection rate (millions of elements/second)
Linear hashing basic (LHB)
Linear hashing with sorted keys (LHS)
Cuckoo hashing into global table (CHG)
Cuckoo hashing into local table (CHL)
Indexing into large bitmap with buffered split (ILBS)
Indexing into large bitmap - no split (ILBN)
4k 8k 16k 32k 64k 128k 256k 512k 1M 2M 4M 8M 16M
Size of set (logarithmic scale)