Fine-Grain Parallelization of Entropy Coding on GPGPUs

Ana Balevic

Abstract

Massively parallel GPUs are increasingly used for acceleration of data-parallel functions, such as image transforms and motion estimation. The last stage, i.e. entropy coding, is typically executed on the CPU due to inherent data dependencies in lossless compression algorithms. We propose a simple way of efficiently dealing with these dependencies, and present two novel parallel compression algorithms specifically designed for efficient execution on many-core architectures. By fine-grain parallelization of lossless compression algorithms we can significantly speed-up the entropy coding and enable completion of all encoding phases on the GPGPU.

Parallelization of Entropy Coding on GPGPUs

GPGPU Tesla Architecture

- Parallel array of streaming processors (SPs)
- Large number of lightweight threads
- Shared memory space for each block of threads (e.g. 256 threads)
- Fine-grain scheduling as a mechanism for memory latency hiding

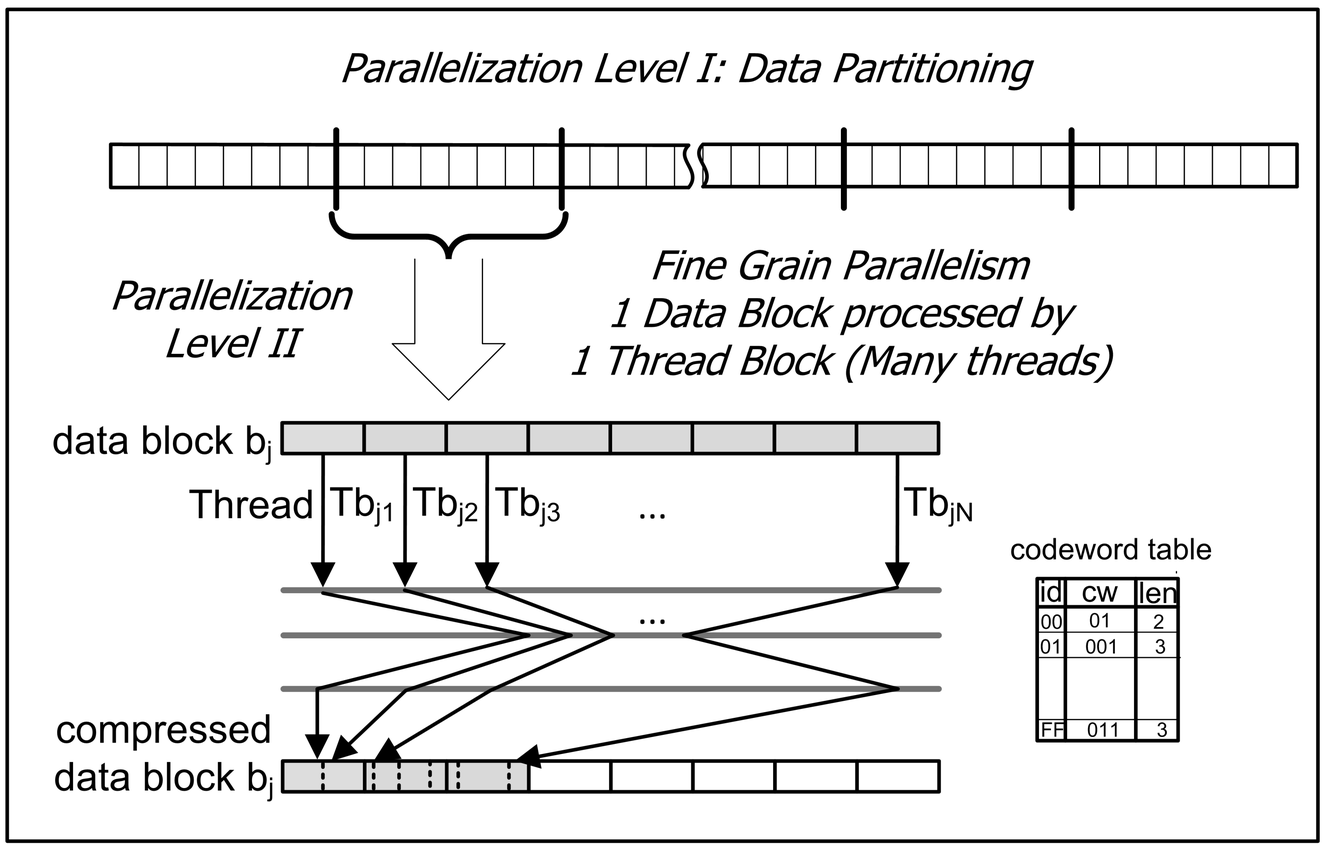
Algorithm Parallelization

Parallelization Levels

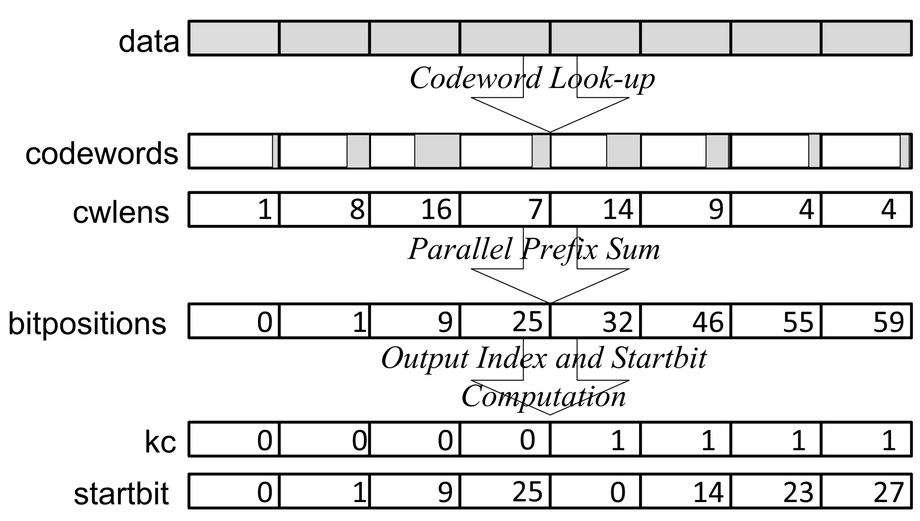
I. Data partitioning
II. Parallelization on the block level
   a) Coarse-grain (one thread processes one data block)
   b) Fine-grain (one thread block processes one data block)

Fine-Grain Parallel Algorithm Design

I. Assignment of codes to the input data
II. Computation of destination memory locations for the codes
III. Concurrent output of the codes to the compressed data array



Parallel Variable Length Encoding Algorithm (PAVLE)



PAVLE Algorithm Steps

I. Assignment of codewords to source data
   - Static codeword table (e.g. Huffman codewords as in JPEG standard)
   - Codewords can be also computed on the fly (e.g. Golomb codes)

II. Computation of output positions
   - Bit-accurate computation of the output position.
     The output position is determined by the memory address and the starting bit position.
   - Efficient parallel computation using prefix sum primitive

III. Parallel bit-level output
   - Parallelized writing of variable-length codewords
   - Safe handling of race conditions via atomic operations

Parallel Run Length Encoding Algorithm (PARLE)

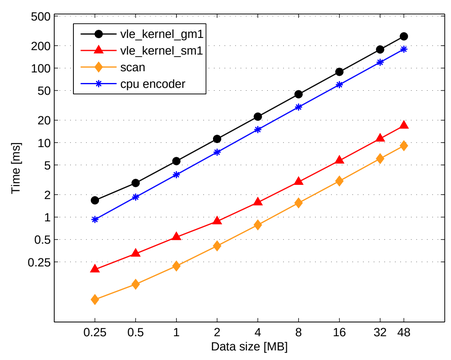
- Repetitions of a symbol are replaced by a (symbol, run length) pair
- Start and ends of symbol runs can be identified by parallel flag generation
- Output positions for RLE pairs also efficiently computed using prefix sum
- Run lengths accumulated using reduction or computed by differencing

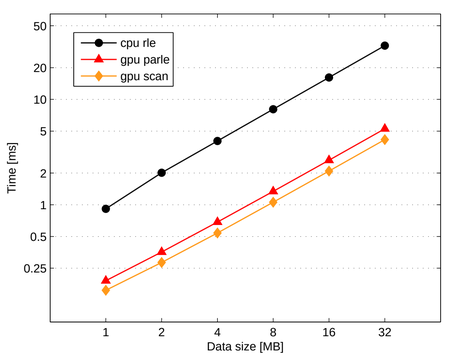
Performance Results

Algorithm complexity: Algorithm work complexity: $O(n)$, Parallel run time $O(\log n)$. Dominant component: Prefix sum primitive.

Test EnvironmentL Intel 2.66GHz QuadCore CPU, 2GB RAM, nVidia GeForce GTX280 GPU w. SM atomic instructions on 32-bit words. Serial encoder and scan kernel given as a reference.

Performance Results Huffman encoding using PAVLE achieves a significant speed-up on a GPU in comparison to a serial (CPU) based encoder. An optimized version of PARLE using shared memory atomics achieves 35x speed-up, PARLE achieves 6x speed-up.





Conclusion

The fine-grain parallel compression algorithms for fixed-length and variable-length encoding achieved up to 6x (PARLE) and 35x (PAVLE) speed-up on the nVidia GeForce GTX280 GPGPU supporting CUDA1.3. These two algorithms can be easily combined with readily available data-parallel transforms and motion-estimation algorithms. In this way, we provide attractive parallel algorithm building blocks for compression, which enable completion of all coding stages of GPU-accelerated codecs directly on a GPGPU. With this approach, not only do we significantly speed-up the entropy coding process (while relieving the CPU), but also reduce the data transfer time from the GPU to system memory. PAVLE and PARLE will be released as open source in the upcoming CUDA Compression library.

Ana Balevic, 2009. Contact: ana.balevic@gmail.com Source Code: http://svn2.xp-dev.com/svn/ana-b-pavle/