Matthew Crotty

CUDA cla-parallel.cu Performance Report

**Implementation of CLA adder functions as CUDA kernels:**

Compute\_gp: I unrolled the loop so that each thread reads from the binaries to calculate a single bit of the gi and pi, with the index that the thread works on calculated from the block Id, block Dim and the thread Id

Compute\_group\_gp, Compute\_section\_gp, Compute\_super\_section\_gp, and Compute\_super\_super\_section\_gp: All of these kernels have the exact same structure. Each thread starts at its index, calculated the same way as the last kernel, and does the gp calculations on a block\_size length slice of the previous level arrays, where block\_size is fixed at 32 at least for this assignment. Each thread sets 1 bit of the respective array.

Compute\_super\_super\_section\_carry: This one could not be parallelized I think, because it is just one 32 length for loop that depends on the previous iterations results to work.

Compute\_super\_section\_carry, Compute\_section\_carry, Compute\_group\_carry, Compute\_carry: Also all had the same structure. They set the first value of the respective array using 0 as the carry in. Then each thread takes 1 bit of the input carry, and iterates through a block\_size chunk in the next step array using the carry in bit from the last iteration.

Compute\_sum: Similar to compute\_gp, this is an unrolled loop with each thread calculating 1 bit of the output using the same 1 bit of the inputs.

**Clock cycles**:

RCA: 34,000,000

CLA-s: 330,000,000

CLA-c-32: 160,000,000

CLA-c-64: 160,000,000

CLA-c-128: 160,000,000

CLA-c-256: 160,000,000

CLA-c-512: 160,000,000

The CUDA block size that yields the best performance is