We implemented following basic algorithms for our project:

1) O(n) and O(nlogn) prefix sum (first one is better)
2) O(nlogn) Range Minimum Query
3) O(nlogn) All nearest smaller value
4) O(n) and O(nlogn) list jumping
5) O(n) parallel merge.

We implemented our initial version of LZ77 algorithm which uses some of the above algorithms, we use open source parallel suffix array implementation, which does not scale well on compile using openMP, mostly because the parallel merge and parallel sort in it has nested parallel which is not good in openMP, so we improve it by using our non-recursive parallel merge.

We implemented test framework including parallel IO operations, text generator.

We implemented the first version of parallel LZW algorithm. Each processor works on a independent chunk of data and compresses it. We did experiments on how different implementation of dictionary would effect the performance, we compare the balanced binary tree Hash table for dictionary. Then we also compared how the hash table size and the number of entries in the hash table would effect the performance.

Forthcoming work:

Fuyao:
The recursive parallel sort in original suffix array construction is not scale well, we plan to implement parallel merge sort and parallel sort by random sampling, and see if there is any improvement.

The RMQ seems not scale well according to the result, we probably need to improve it if it become a bottleneck after suffix array improvement.

Nagarathnam:
Implemented tree dependent version of LZW, which will suppose to improve the compression ratio with slight reduce of scalability.

Implement the tree dependent version of LZ78 algorithm, which provides increased compression ratio with reduced overhead of maintaining dictionary.

All:
We need to run parallel LZ77 and LZW on following settings and do more detail compression ratio and scalability:
Alphabet size: 4, 16, 64, 256, 1024, 4096
Text size: 2^24 random generated text, English text, binary files.
Cores: 1, 2, 4, 8, 16, 32, 64 (more if possible)
We are going to make a lot of graphs on them.

We will list our initial comparison in following table and graph. The result columns shows the the size of array after compression, noted in LZ77, the actual size would be larger that the value since we need extra storage for match length, one can assume 1.5 factor large than it.

	LZ77 Time	LZ77 Speedup	LZ77 Result	LZW Time	LZW Speedup	LZW Result
Sequential	5.53		788646			
1 Cores	6.63	1	788646	2.42479	1	1488314
2 Cores	3.37	1.97	788646	1.54193	1.57	1489591
4 Cores	1.82	3.64	788646	0.740329	3.28	1491012
8 Cores	1.09	6.08	788646	0.389586	6.22	1493253
16 Cores	0.858	7.73	788646	0.856146	2.83	1497917