Turning the PageRank on PDN

Lexie Barthelemess
CS Senior

Mark Cunningham
CS Senior

Quentin Windsor
CS Senior

***Abstract*—The PageRank algorithm was developed to rank web pages for search engines. A page is considered ”important” if other important pages link to it. Each page is represented as a node on a graph. This paper examines storing these nodes in different sparse data formats as well as different parallel constructs for iteration to convergence.**

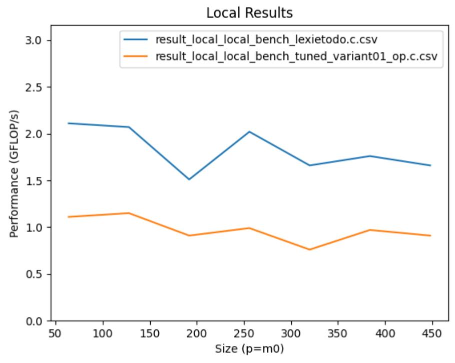
I. COO to CSR

The CSR memory format is more space-efficient than COO since we are able to consolidate references to the row index. Thus, we need to store less values for row in comparison to COO, which stores values 1-for-1-for-1.

In the transformation function that we wrote to take a COO matrix and produce a CSR, we first allocate appropriate memory for the CSR matrix and copy the metadata over. Then, it allocates buffers for row, column, and values. These are filled with information from the COO matrix.

The CSR format is beneficial for performance since it optimizes memory usage through contiguous memory access; since the non-zero elements are stored consecutively, we can avoid cache misses and improve efficiency for operations such as this matrix-vector multiplication.

Below is a graph showing the difference between using the baseline (COO) format and using the CSR format.



II. Parallelism in Isolation

A. SIMD

While analyzing the memory access pattern in the Coordinate Storage Format, we observed repeated instances of consecutive row indices. To leverage this pattern, we concurrently processed eight rows at a time where possible. The values in the row index array were often greater than eight, but were almost never a multiple of eight. To eliminate bounds checking during matrix multiplication, we divided the row index array into two groups: SIMD and regular. Each group is represented as an array containing (start, end) indices. In the SIMD version, pairs strictly spanned eight elements, while the regular version encompassed all other pairs. As the size and number of non zero entries grew, the percentage of non-zero entries covered by the SIMD operation increased. About 68% of entries are performed by SIMD at size 128, and about 89% at size 1024. The percent performed by SIMD levels off at around 92%.

The matrix vector multiplication was split into two for-loops: One that spans the size of the SIMD entries and the other spanning the size of the regular entries. In each iteration of the SIMD loop, eight column indices are loaded from the column-index-array, which are used to gather the corresponding values from the x-array. Then, eight values are loaded from the values-array and are multiplied by the values from the x-array. The resulting vector is summed into a single float value which is added to the value at the current index of the y-array. The SIMD variant improved doubled the performance across all sizes when compared to the baseline, but did not see further improvements as the size increased.

B. openMP

To parallelize the matrix vector multiplication, we used openMP to split the work among threads in a parallel region. Since there were many occasions where it was possible for two threads to be writing to the same element in the y-array concurrently, we made sure that each thread has its own private copy of the y-array. We used the reduction clause to avoid race conditions without sacrificing too much performance. After the parallel region has been completed, the private copies are copied back to the y-array.

The performance improvement over baseline for openMP increased linearly as the size of the array increased. However, the performance was worse before the size hit around 250, likely because the cost of allocating threads was not yet worth it.

C. 2 Forms Together

We chose to combine SIMD and openMP for our 2 Form variant. It was a relatively simple combination of the two that did not require any extra coding. The performance gain was slightly better than openMP by itself, increased by about the same amount that SIMD was over baseline after the size reached approximately 500.