Applications of Parallel Computers Final Project

A comparison between Floyd Warshall and Square Sum

Group members

- Abdelrahman Kamel (ak883)
- Hee Jung Ryu (hr99)

Hypothesis

In this project we would like to compare Floyd Warshall's path finding algorithm to the square sum implementation from our previous project. We expect the two implementations to differ only by a constant. Given n Nodes, Floyd Warshall has a time complexity of $O(n^3)$ while the Square Sum method preforms in $O(n^3 \log(n))$. We hypothesise that when we parallelize Floyd Warshall's algorithm cache efficiency will be similar and performance will only differ by at maximum, a constant of log(n).

Analysis

Timing Results from Square Sum

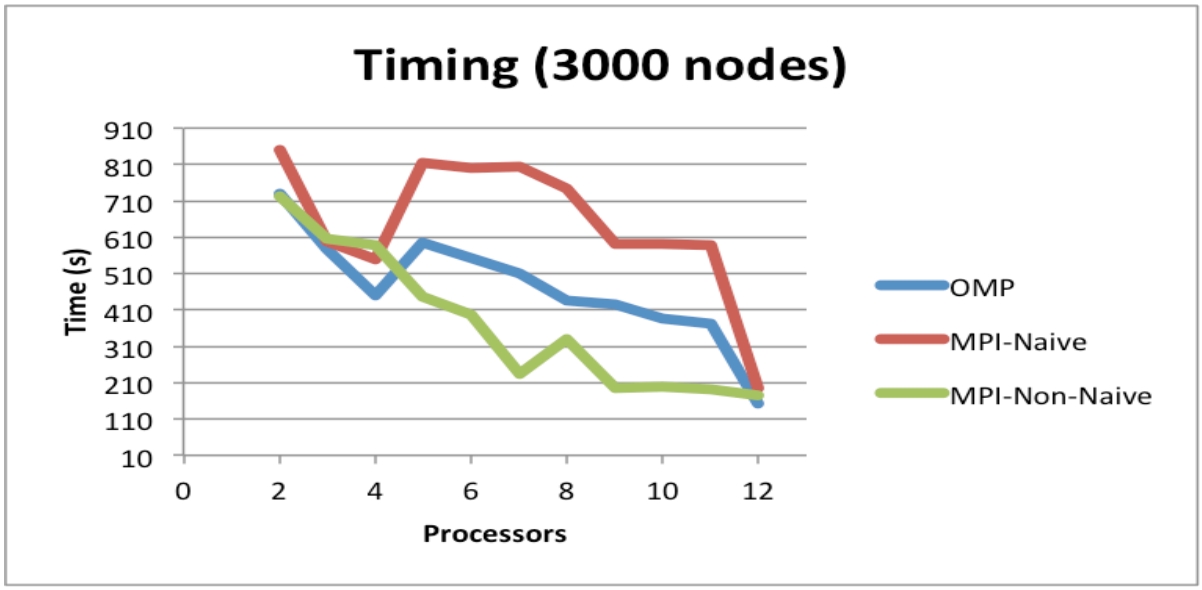

Figure1: Here we show results from our previous implementation. Note the time taken to solve three thousand nodes varies from ~900s to ~210s. We later compare this time with Floyd Warshall's algorithm.

Timing Results from Floyd Warshall OMP

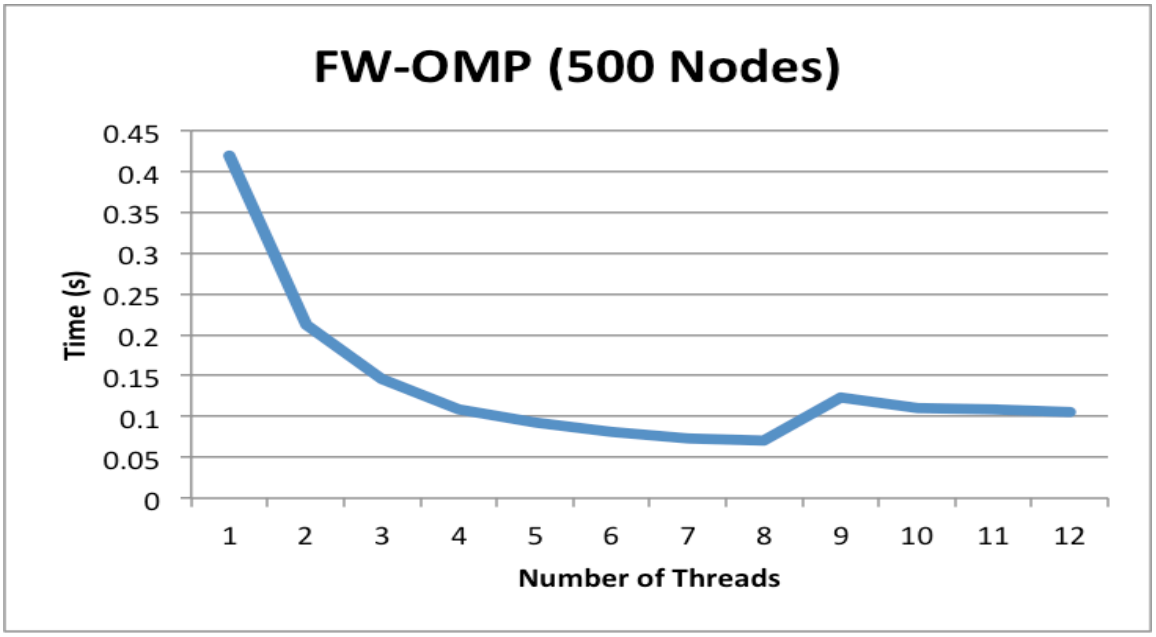

Figure 2: Above we show results for 500 nodes. In this plot we show the performance of Floyd Warshall's algorithm using OMP. The results where obtained by varying the number of threads.

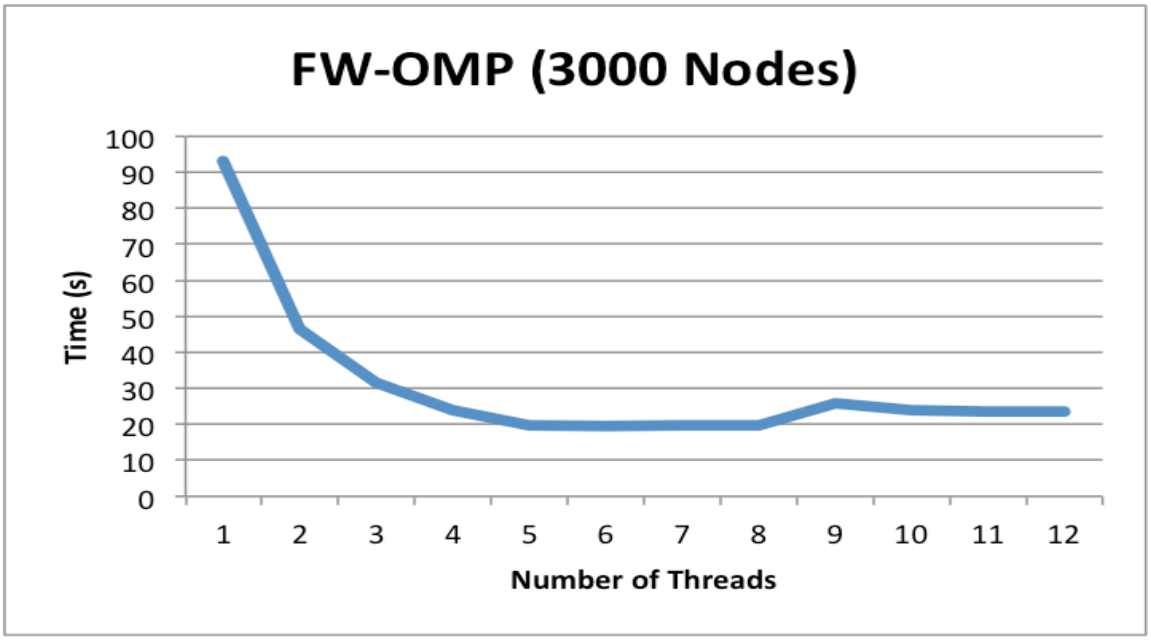

Figure 3: Here we show timing results for 3000 nodes. This plot also demonstrates the performance of Floyd Warshall's algorithm using OMP.

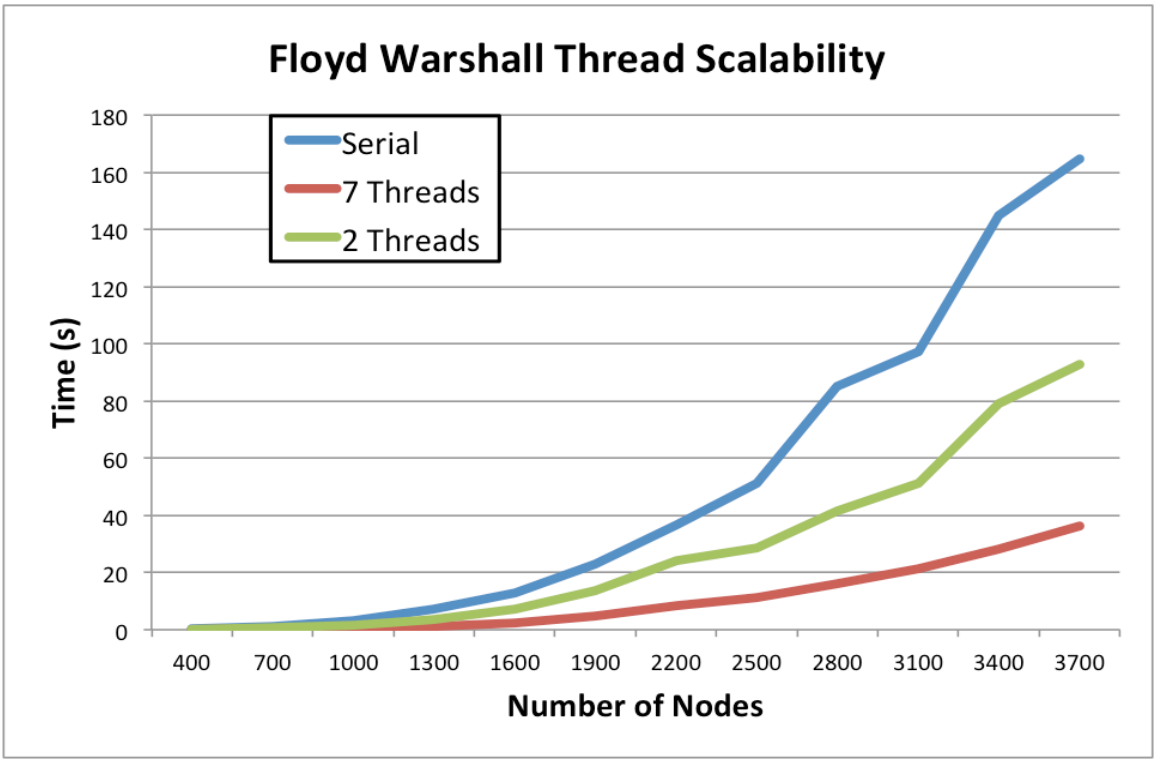

Figure 4: By varying the number of nodes we can see the performance of serial code compared to 7 threads. We can see at 3700 nodes the time varying by a factor of 8.

Timing Results Conclusion

From Figure 1 and Figure 2 we can conclude that no improvement is obtained after five threads. The reason for this is explained in the later section. However, we can conclude that we do see a major improvement in timing. In Figure 1 we show results from the square sum implementation. In comparison to Figure 3, we see that major improvement in timing.

When using 1 processor to solve a problem of 3000 nodes, we see that the square sum implementation takes approximately 800 seconds. Floyd Warshall's algorithm solves the same problem in approximately 100 seconds. While these results show a major improvement, they are not the timing results we expected.

One suspicion is the cluster system we are running our experiments is returning in consistent times. We have verified that our timing is consistent on our own machines. By running several other trial runs, we did notice inconsistent timing results. However, we did obtain enough data to make a definitive conclusion that we do not notice any improvement after five threads. We also conclude that there is a major improvement in timing results, however we are unsure if the improvement differs by at maximum constant of log(n). From Figure 4 we can also conclude that a speedup exists by varying

the number of threads. We can see that 7 threads preform much better than 2 threads or the serial implementation.

Speedup

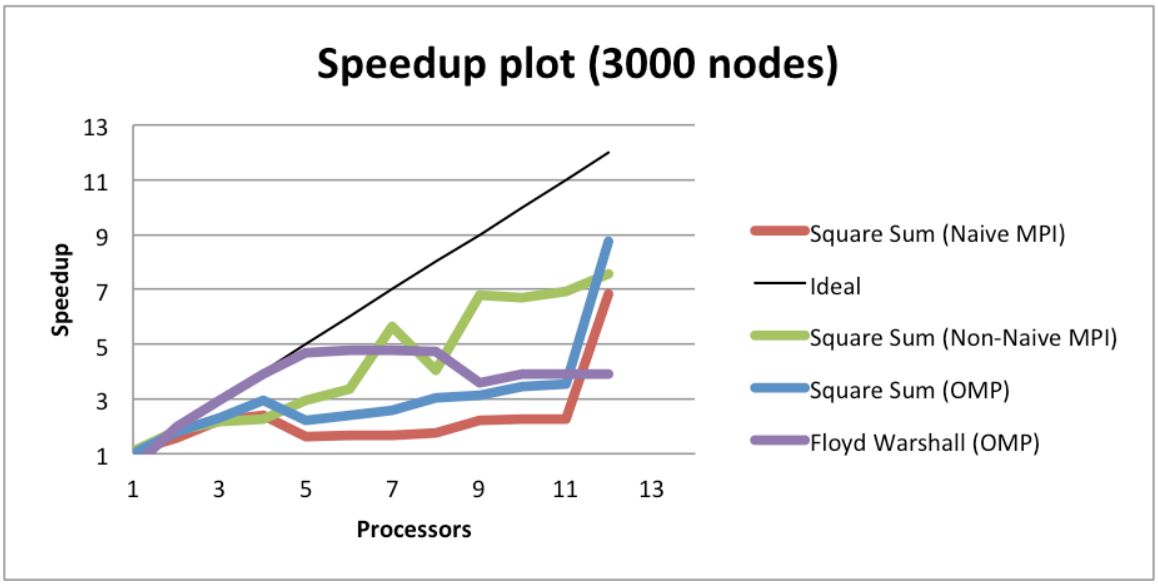

Figure 5: In this speedup plot we compare the speedup of Floyd Warshall's algorithm with the Square sum implementation from project 3 by varying the number of processors.

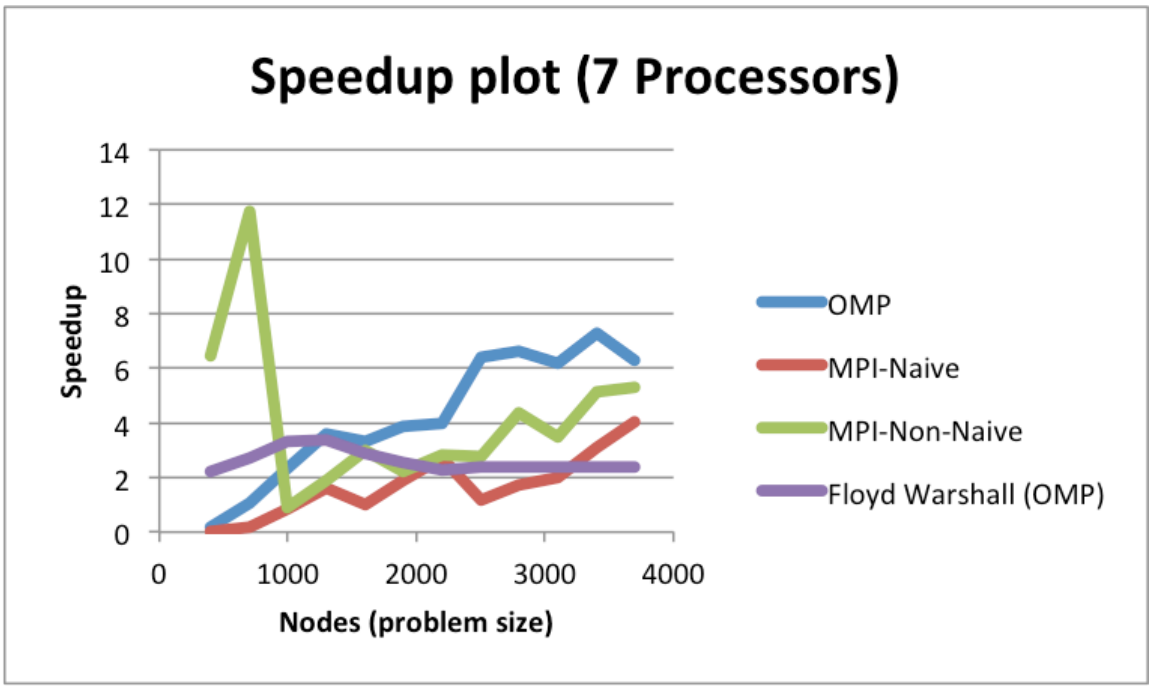

Figure 6: Here we also compare the algorithms by varying the number nodes.

Speedup Conclusion

In Figure 5 we see how Floyd Warshall's algorithm follows the ideal speedup until five threads. While the algorithm performs much better than square sum OMP and MPI implementations, we did not observe any speedup after five threads due to the way we are parallelizing the algorithm:

Floyd Warshall Algorithm:

```
for(int k=0; k<n; k++){
        #pragma omp parallel shared(l,k)
        {
                #pragma omp for schedule(static)
                for(int i=0; i<n; i++){
                        int ik = i * n + k;
                        for (int j=0; j<n; j++){
                                int ij = i * n + j;
                                int kj = k * n + j;
                                if(i == j ) l[ij] = 0;
                                if(l[ik]+l[kj]< l[ij])
                                        l[ij] = l[ik]+l[kj];
                        }
                }
        }
}
```

Code Snippet #1: Floyd Warshall OMP Implementation

We are parallelizing the two inner most loops of the algorithm. Due to this type of parallelization, we only expect to see ⅔ of the total speedup. Since the clustering environment only supports eight processors; we only expect to see a speedup to ⅔ of 8 which results with a speedup of five. This conclusion is consistent with our results from the speedup plot.

However in Figure 6, we notice a linear speedup. There are several reasons that can explain this behavior. The serial implementation of Floyd Warshall preforms very well in comparison to the parallel version. Another explanation is that the timing results from the Crocus cluster were inconsistent. As we have stated before, by experimentation and various other trials we noticed inconsistent timings from the cluster.

Cache Efficiency

Flop Rate

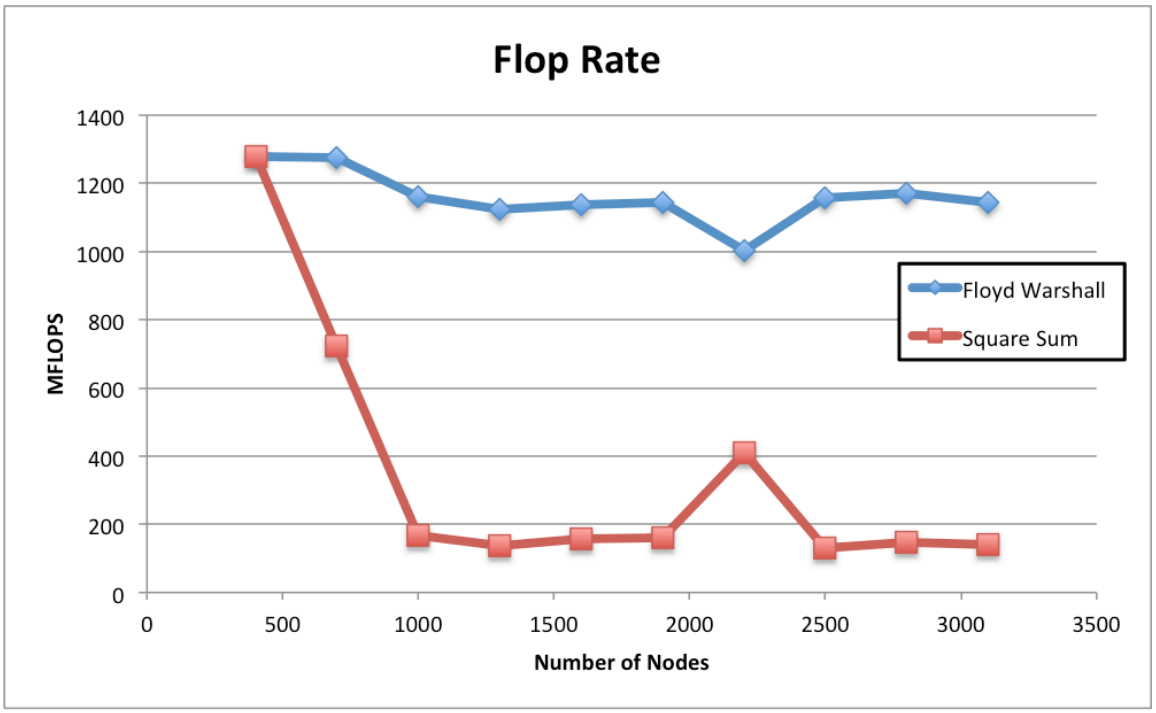

Figure 7: This plot shows how many floating point operations per second for both algorithms Floyd Warshall and Square Sum.

In contrast with the square sum implementation, Floyd Warshall's algorithm preforms drastically better than the square sum method. The main reason for this is cache efficiency.

Square Sum Algorithm:

```
for (int j = 0; j < n; ++j) {
    for (int i = 0; i < n; ++i) {
        int lij = lnew[j*n+i];
        for (int k = 0; k < n; ++k) {
            int lik = l[k*n+i];
            int lkj = l[j*n+k];
            if (lik + lkj < lij) {
                lij = lik+lkj;
                done = 0;
            }
        }
        lnew[j*n+i] = lij;
```

```
    }
}
```

Code Snippet #2: Square Sum Serial Implementation

Square sum algorithm has many random access to the array for the part highlighted in yellow in the code snippet above. Furthermore, square sum implementation does not utilize blocking to improve cache efficiency. So the results we are seeing for square sum are expected.

However, the results we obtained for Floyd Warshall's algorithm were beyond our expectation. When we implemented the algorithm we expected to see the similar type of cache inefficiencies we saw in square sum. The square sum implementation makes random accesses in the second outer loop, the code line highlighted with yellow in the Code Snippet #2. This is not the case, however, we see that the cache utilization is very optimal for approximately four thousand nodes. This behavior can partially be explained by two points: (1) the assignments being made in the algorithm and (2) the consecutive accesses to the array in the inner-most loop. Assignments are only made when a shorter distance has been found. Unlike in Square Sum, Floyd Warshall accesses the array consecutively only in the inner most loop. To further analyze this unexpected behavior we decided to use Cachegrind. A tool provided in the Valgrind toolkit.

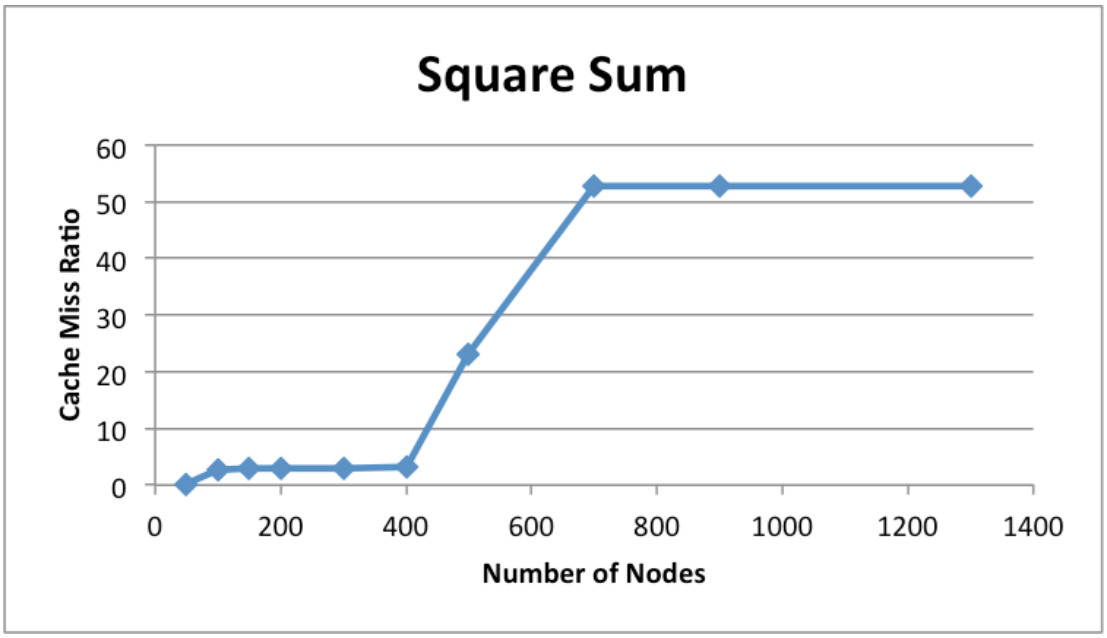

Figure 8: Cache utilization for square sum algorithm. We would like to note that we got similar results after one thousand nodes. Hence, we truncated the plot to show only fourteen hundred nodes. Values after one thousand were close to 53%.

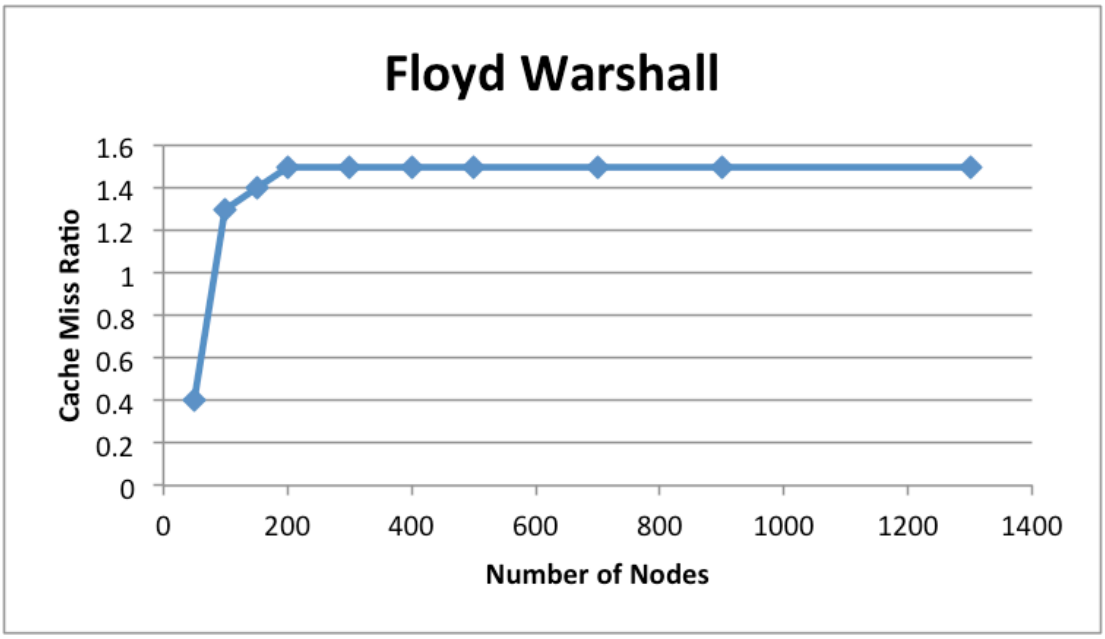

Figure 9: Cache utilization for Floyd Warshall. Results for both plots were obtained from Cachegrind. After three hundred nodes, we noticed similar results of 1.5% on the graph.

Cache Efficiency Conclusion

The results obtained from cachegrind show a major difference in cache utilization between the two algorithms. Square Sum implementation demonstrates a cache miss ratio of 53% for a large problem set while Floyd Warshall only has a cache miss ratio of 1.5%. We can safely assume that Floyd Warshall utilizes the cache more efficiently than Square Sum. This conclusion is unexpected from what we originally hypothesized.

As stated before, the two main contributions to this result is (1) the amount of assignments the algorithm makes and (2) the consecutive access to the array in the inner-most loop. Because of (1), a memory access is only made when a smaller distance is found in the matrix.

Compiler Optimization Flags

We also used some compiler optimization flags to help improve cache efficiency. The most noticeable flags that boosted performance where -O3 and -funroll-all-loops. We experimented with various flags such as -funroll-loops which unrolled predictable branches only rather than all branches, but the results were very similar.

Current and Future plans

If time permits, we would like to implement a blocked version of Square Sum implementation and compare the results. By doing so, we expect the algorithms to preform relatively the same in terms of time and cache. If the algorithm utilizes the cache more efficiently, then we expect to see a decrease in the total time taken to solve a problem of size n.

We would also like to implement a MPI version of the algorithm. Doing so would allow us to compare OMP with MPI to decide which library is the better for parallelism. In terms of expectations, we expect MPI to outpreform OMP for large problem sets since it can scale better.

Another improvement that we can add to the two implementations is the use of SSE instructions. The SSE instruction set contains a min operation that can potentially result in good speedup.

Conclusion

In conclusion, Floyd Warshall's algorithm preforms much better than repeated Square Sum. However, the algorithm does not lend itself to parallelisation while repeated Square Sum does. We see good speedup from Square Sum implementation using MPI and poor speedup after five threads from Floyd Warshall.

By comparing cache utilization, we see that Floyd Warshall performs much better than Square Sum in terms of cache. These results were mostly unexpected but explainable. We also saw a reflection of poor cache utilization of the Square Sum implementation in terms of time.

Due to various timing results from the Crocus cluster, we can not conclude that the two implementations vary by at maximum a constant of log(n). We show results that vary by a much greater constant than log(n) partially due to an unstable cluster environment.

Acknowledgement

We would like to thank Professor Bindel for providing the ground work for our project and the countless hours of help.

References

1. "Floyd-Warshall All-Pairs Shortest Pairs Algorithm." . N.p., n.d. Web. 25 Nov 2011. <http://www.pms.ifi.lmu.de/lehre/compgeometry/Gosper/shortest_path/shortest_path.html>

2. "Using the GNU Compiler (GCC)." . Free Software Foundation, n.d. Web. 25 Nov 2011. <http://gcc.gnu.org/onlinedocs/gcc-3.3.6/gcc/>

3. Badrinath, R. "Parallel Programming and MP." . Free Software Foundation, 2008. Web. 25 Nov 2011 <http://www.scribd.com/doc/23340655/11/Floyd's-sequential-algorithm>

4. Ian, Foster. "Case Study: Shortest-Path Algorithms." . Argonne National Laboratory, 1995. Web. 25 Nov 2011. <http://www.mcs.anl.gov/~itf/dbpp/text/node35.html>

5. " Cachegrind: a cache and branch-prediction profiler." . Valgrind, 2011. Web. 25 Nov 2011. <http://valgrind.org/docs/manual/cg-manual.html>