Allie Wolf

Brian Steele

Cole Walsh

**Project 3: Matrix Multiplication Report**

The Matrix Multiplication program was designed and programmed in group or paired programming. These group meetings occurred three to five times per week.

The initial design document was made in group meetings in collaboration and updated only after the code was changed or updated to accommodate improved implementation.

Group coding sessions were attended by all group members 3-5 times a week.

Thanks to Lab 3 from our class work, each of us had preliminary Producer-Consumer programs to compare and use as a jumping off point. Cole gathered resources and wrote a preliminary draft of the SharedBuffer class, followed by code review, testing and debugging of the overall program. New classes were written by Allie and group coding sessions were held to add methods and test, debug and improve program utilization. Allie deserves the bulk of the credit for writing the initial version of the program. Brian spear-headed the Java documentation (along with contributions from Allie and Cole on various versions before code was frozen) and a JFileChooser for choosing scenario files. He also wrote drafts of the Producer class. The report was curated by Cole.

During these group coding sessions all team members contributed to the continuous testing and debugging of the program.

We observed that because our program is compute intensive, the threading has not saved much time. The additional time to set up the threads adds to the total simulation time. Multiprocessing may have been more efficient as opposed to multithreading.

There seems to be a correlation between the times the buffer is full and the overall simulation time. Additionally, implementing a low producer and consumer sleep time caused deadlock issues.

We attempted to create an unusual test case by implementing a 100 by 100 matrix, and it was only then that we ran into occasional issues with deadlock. However, we created this test case specifically to break the program, so were not surprised by the frequency of issues it caused. Our inference was that using locks or thread pools would alleviate these issues.