There are two main types of memory systems that are used when considering parallelized programs, namely shared memory and distributed memory. In a shared memory system, all of the processors share a common main memory; in other words, each processor can see each other processor's reads and writes and each processor has equal latency when accessing the memory. (As a side note, newer computers use more complicated hardware than simply allowing each processor direct access to the same main memory, however the concept of shared memory still works the same from a programming level). In a distributed memory system, on the other hand, multiple processors utilize their own separate memories and is only able to transmit data between the processors using explicit messages to do so.

Most libraries that focus on parallelization are primarily focused on optimizing for either shared memory systems or distributed ones. For example, the first library we found and used as an initial translation, Armadillo, has a parallelization option that utilizes OpenMP, which is optimized for shared memory systems. The other two libraries we considered, Eigen and Scalapack, are similarly suited for shared memory and distributed memory respectively. Thus, one of the major deciding factors for which library we would use was which memory system we would have access to for running the final program.

Originally, Dr. Zare was offering us access to his computer system on campus which, we believe, uses a shared memory system. Because of this, we decided to go with the Eigen. Even with access to LoneStar6, which would have been able to support a distributed memory set up if we wanted, we decided to stick with Eigen since LS6 can also be used as a shared memory system. Our main reason for this was that the Eigen library was more easily readable and the documentation was easier to understand, so we determined that this would give us the best chance of finishing the project on time since it would be quicker to learn; however, further research has revealed that this might be the correct direction to go for the CCAMA algorithm regardless of which library would be quicker to learn.

Because a distributed memory system has to use outside processes to send messages between the processors to transmit data from one to the other, the overhead becomes very massive for problems that require memory to be accessible to multiple threads at once; in these cases, a shared memory system would be more efficient. Conversely, distributed memory can be even more efficient than shared memory when the processors do not need to communicate with each other as much. When it comes to matrices, operations between sparse matrices require much less communication between processors than operations between dense matrices do. Thus, a shared memory system would be more efficient than a distributed memory system for problems requiring calculations done with dense matrices, such as the CCAMA algorithm.

For all of these reasons, we have decided to use—and had some preliminary success with—the Armadillo and Eigen libraries for parallelizing linear algebra calculations in C++.