Blake Rieschick

Final Project Implementation Document

**Implementation Approach**

The approach I took in creating this software was to break the program up into 3 parts. The first part is the import where the files are converted into an array filled with strings broken by spaces and end lines within the input files. Each Processor is created as a struct with various properties and methods that work to run the program smoothly. The importing to the an array is a member variable of the struct, named tTable. The imported array also has states attached with each tag that have been initialized to ‘i’.

Now that the arrays is created, we start the second part which is starting the global counter and loop through each processor and push it another member variable, a vector, transTable if the time is less than the global clock. Now that transactions are starting to be populated in each processors section we start processing the transaction by popping a transaction from a transTable if one exists. That transaction is then processed as a read or write action, and ran.

When a transaction is ran, a tag and index is passed along with the processor that is calling for the read or write. The state on that tagline is the update according to the MOSEI protocol. As the update occurs the state that is being affected modifies struct values needed for the output (write backs, lines in state, etc). This is done while transactions exist in any one processors transTable. After all tables have been emptied the function returns.

The last part of the program, the output, is then collected from all the processors and printed to the screen for the user to see the results of the program on the imported files. The program will also display the number of transactions processed successfully, and the time taken for the entire program to execute in seconds.

**Implementation Results**

The results of the program varied depending on a couple factors. The largest factor that I saw play a role in producing bad data was a clock that executes to quickly. Originally the program counter was clock cycles but the time it took to execute the import took longer then the number of clock cycles it was being compared to. As a result all the transactions were being imported sequentially rather than by timestamp. The fix for this transitioning from clock cycles to seconds and then skewing the time to make it slower as needed. This accurately fixed the import process so that transaction were processed in an accurate way.

Another problem that was seen while producing this fighting the quick rising complexity of this program where in a given read or write the active processor is searching other processors for that tag and index, and then doing so for all the inputs in each of the files. Functions seemed to grow rather quickly greatly increasing the program execution time. Keeping functions simple and reducing continuous nested functions calls within the program all helped to reduce this process and keep the main duration of execution time on the importing process.