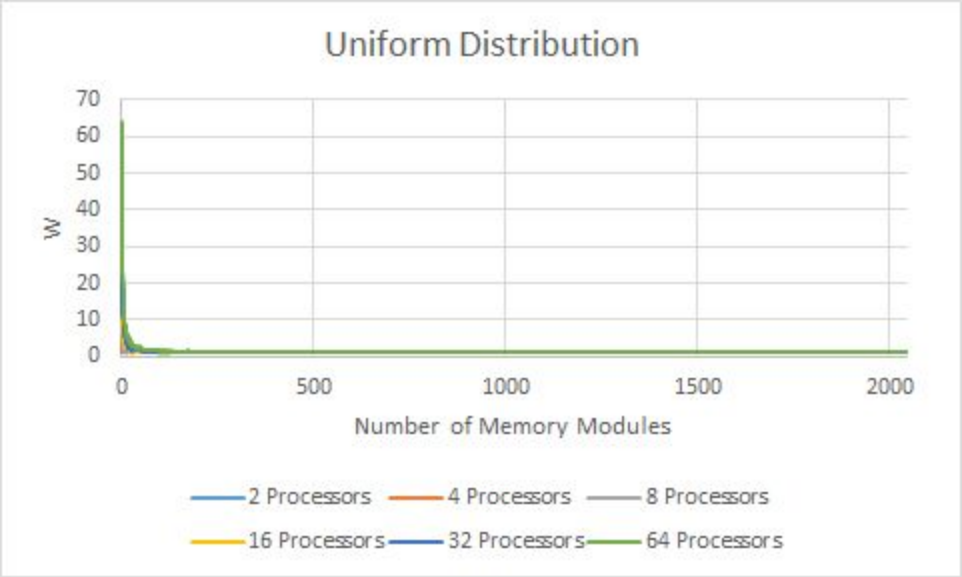
Homework Group- 34
Name: Parth Shah
UCINetId: parths4
Student Id: 54809514

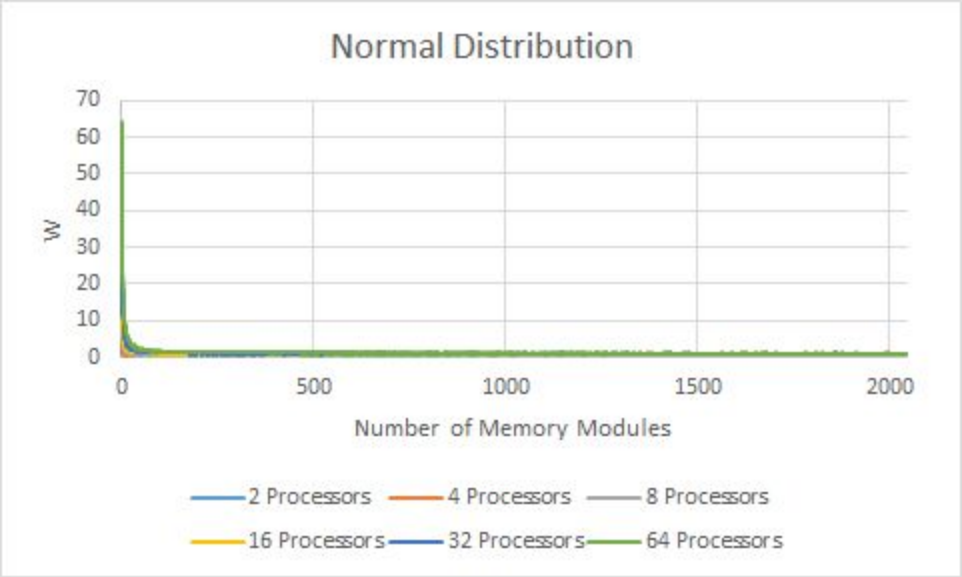
Name: Yasin Zuhuruddin
UCINetId: kzuhurud
Student Id: 11443054

Parallel and Distributed Systems

Homework 1 Report

1. Results for Uniform and Normal Distribution





Homework Group- 34
Name: Parth Shah
UCINetId: parths4
Student Id: 54809514

Name: Yasin Zuhuruddin
UCINetId: kzuhurud
Student Id: 11443054

2. Analysis of Results

From the simulation and the graphs, it is clear that we don't need many memory modules to have fairly high uptime across all processors. Depending on the performance required, purchasing memory modules linear to the number of processors generally gives a 1.5 cycle access time. If they're very expensive, going as low as #processors/2 still yields relatively fast results at 2.5 cycles. If performance is critical you'd want to triple or quadruple the number of memory modules to get 1.1-1.2 cycle access time. Both graphs follow similar patterns where each new memory module starts to provide diminishing returns quickly. This is normally at 2x-3x the number of processors in that simulation. However, even as the number of memory modules approaches 2048, you sometimes see processors waiting for memory modules, due to the scholastic nature of the memory module assignment.

Given the locality of the memory modules to processors, the simulation using the normal distribution probably better represents a real world application. In that simulation, it takes slightly more memory modules before it reaches a stable state as the chances for collisions are higher. While a uniform distribution, in theory, would provide better performance across the board, it does not account for the latency between a processor and it's requested memory module. In a real world application, it's much faster for a processor to access a memory module that is physically closer to it. Thus, the simulation using a normal distribution gives better results when trying to estimate the number of memory modules to purchase.

When you take the above scenario a bit further, given the random nature of our assignment of processors to memory modules, the chance that two processors choose the same module is low. This provides an optimistic look on the number of memory modules, maybe underestimating the true number needed for good performance. If, in a real system, the same processors are generally accessing the same subset of memory modules, it may be better to simulate a more distributed version, where subsets of processors are assigned to subsets of memory modules, and look at the access time then. As there are fewer and fewer modules to choose from, the chance of collision goes up, increasing access time. An even better approach would be to include some time delay if a processor accesses a memory module farther away. Either way, our simulation provides a lower bound on the number of processors needed.

To avoid starvation of processors we maintain a queue, starting with the first unassigned processor and storing all the unassigned processors in that queue. Now, in the next cycle all the unassigned processors will be tried to assign a memory module before the remaining processors and then the remaining processors will follow natural order. Any processor not assigned in the current cycle will have higher priority in the next cycle till they are assigned their memory module. They will wait no more than #processor cycles before guaranteeing access to the requested module.