

# Dynamic Partition Algorithm for Distributed Mur $\varphi$

Rahul Kumar, Michael D. Jones, Eric G. Mercer and John Lesuer

June 2, 2004

This technical report will present the dynamic partition algorithm and results for the algorithm implemented within distributed Mur $\varphi$ .

To further explore the effects of the partition function, a dynamic hash function can be created which will take into account the irregular state space of the model being verified, and in theory try to create a perfect partition of the states in the hash table as well as evenly balanced queues across all processors. The primary goal of the dynamic partition function is to verify models in a manner where memory and computational resources are used only if necessary, making the algorithm time and time efficient. To do so, verification is started on a single processor. The *memory threshold* is defined as the hash occupancy percentage at which a processor allocates more memory by including another processor. Once the memory threshold is reached on the first processor, a second processor is requested with the same amount of memory allocated for the hash table as the parent processor. States from the hash table as well as the queue are transferred to the child processor via messages on the underlying interconnect. Before the states have been transferred, the partition function is updated to accommodate the new processor and to create an even partition of states. The new partition function is communicated to the other processors to maintain correctness of the verification and the partition. Splitting and state generation continues until the model has been verified completely or there are no more processors to split to. The current algorithm uses the state vector as the only input to calculate the new hash function. The memory threshold parameter is predefined by the user during the time of compilation. For the results presented in this section, the memory threshold was set at 70%.

Figure 1 shows a splitting graph where each processor represents a processor. Initially there is only

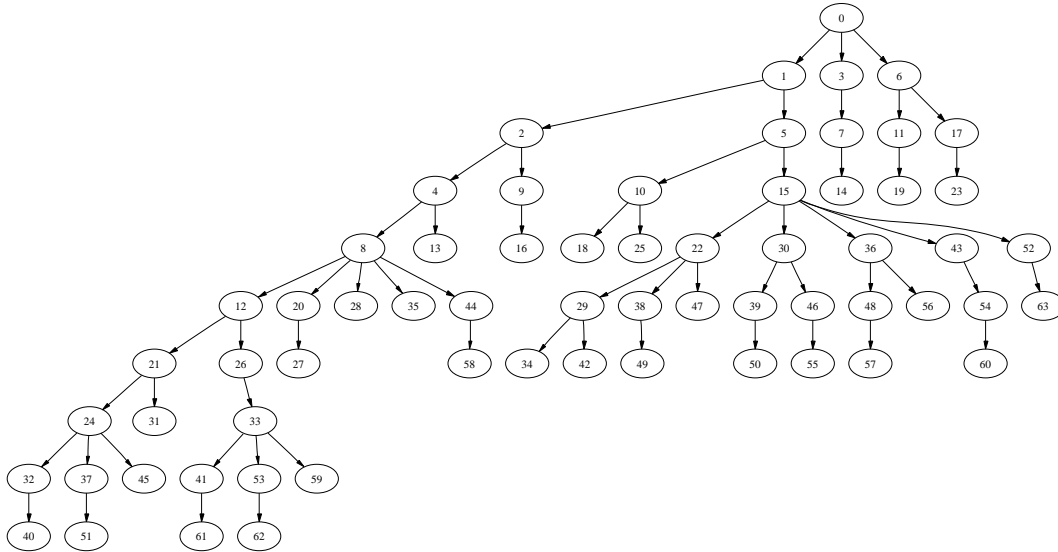


Figure 1: Splitting pattern of processors in dynamic partition algorithm using 64 processors on NOW

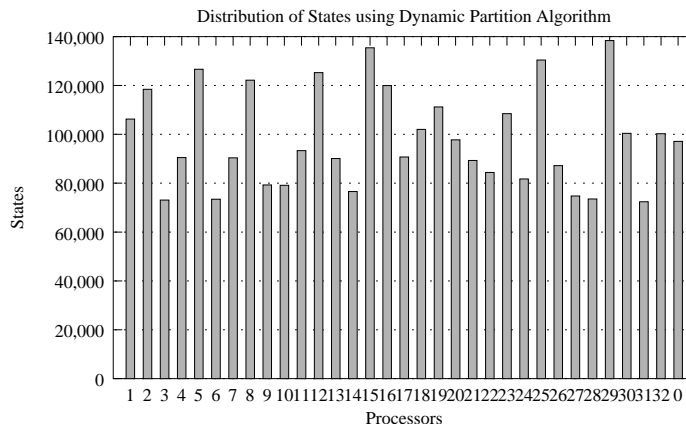


Figure 2: State distribution of hash table using Dynamic Partition Function on IBM 1350 Cluster

processor 0, which grows into 64 processors by the time the model has been completely verified. From the figure we can see that splitting does not occur in a balanced or sequential manner. Sequential here would imply that after processor 0 splits into two processors [0,1], and each processor 0 and 1 should be equally loaded, which then split into [0,2] and [1,3]. This is not the case though. Processor 0 splits into [0,1] and then [0,3] followed by a final split of [0,6] while 1 splits into [1,2] and [1,5]. This behavior occurs due to the presence of an imbalance in the queues and the hash table; thus, causing certain processors to process and save more states from their queue, causing them to split earlier than the other under worked processors.

The performance achieved by the algorithm is not comparable to the static partition algorithm discussed

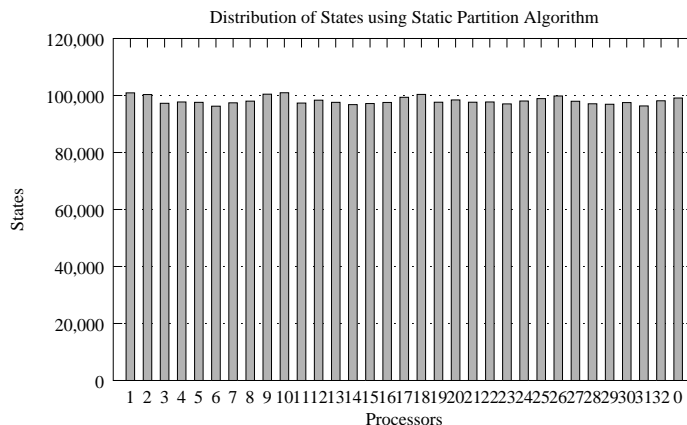


Figure 3: State distribution of hash table using Static Partition Function on IBM 1350 Cluster

above. The state distribution in the hash table achieved is very uneven and imbalanced. Figure 2 shows the distribution of states in the hash table on all processors using the dynamic partition algorithm on the IBM 1350 Cluster. The algorithm required 33 processors to completely verify the model. To compare the state distribution of the dynamic partition algorithm to the static partition algorithm, the static partition algorithm was run using 33 processors and the same amount of memory allocated for the hash table on each processor as the dynamic partition algorithm. Figure 3 shows the distribution of the states in the hash table on each processor achieved using the static partition algorithm using the IBM 1350 Cluster. As can be seen from the figures, the distribution is more even for the static partition algorithm compared to the dynamic partition algorithm. An important difference is the maximum number of states saved on a single processor, which can determine the fate of the verification. Using the dynamic partition algorithm, the maximum number of states is at most 1.5 times greater than the maximum number of states on a single processor using the static partitioning algorithm. Apart from the state distribution, the speedup achieved by the static partition is significant compared to the dynamic partition algorithm. This is because the dynamic partition algorithm, does not utilize all 33 processor throughout the verification, instead adding them to the current set of processors as needed by the memory requirements of the model.

Apart from the performance issues mentioned above, the dynamic partition algorithm does not lend itself very well to the super computing paradigms listed above. The greatest issue being the fact that new processors as well as memory have to be requested on the fly, which is difficult to do with current MPI

implementations.