Runtime Thread Creation for Improved Ray-Tracing Performance on Wide SIMT/SIMD Processors

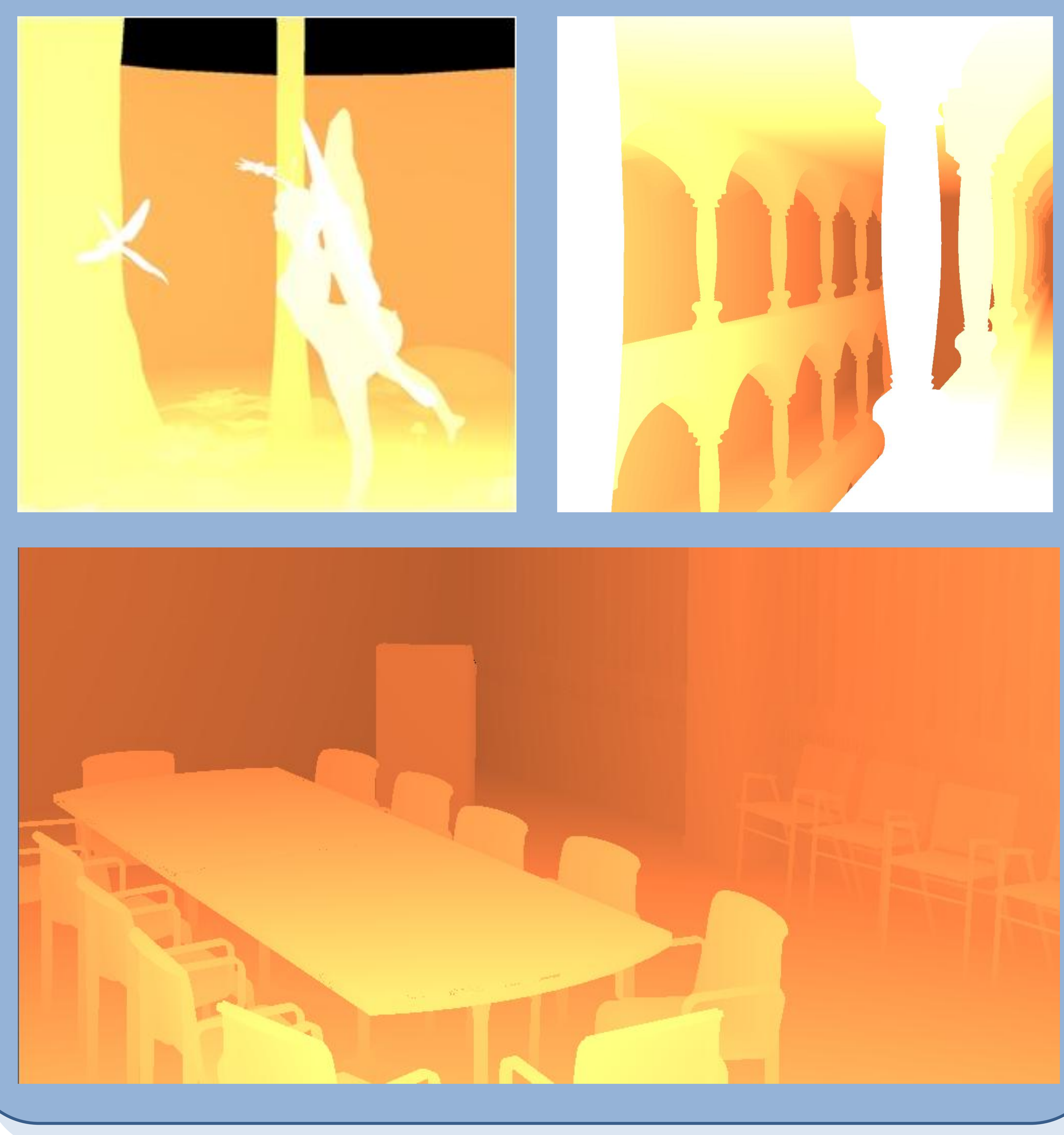
Michael Steffen Joseph Zambreno

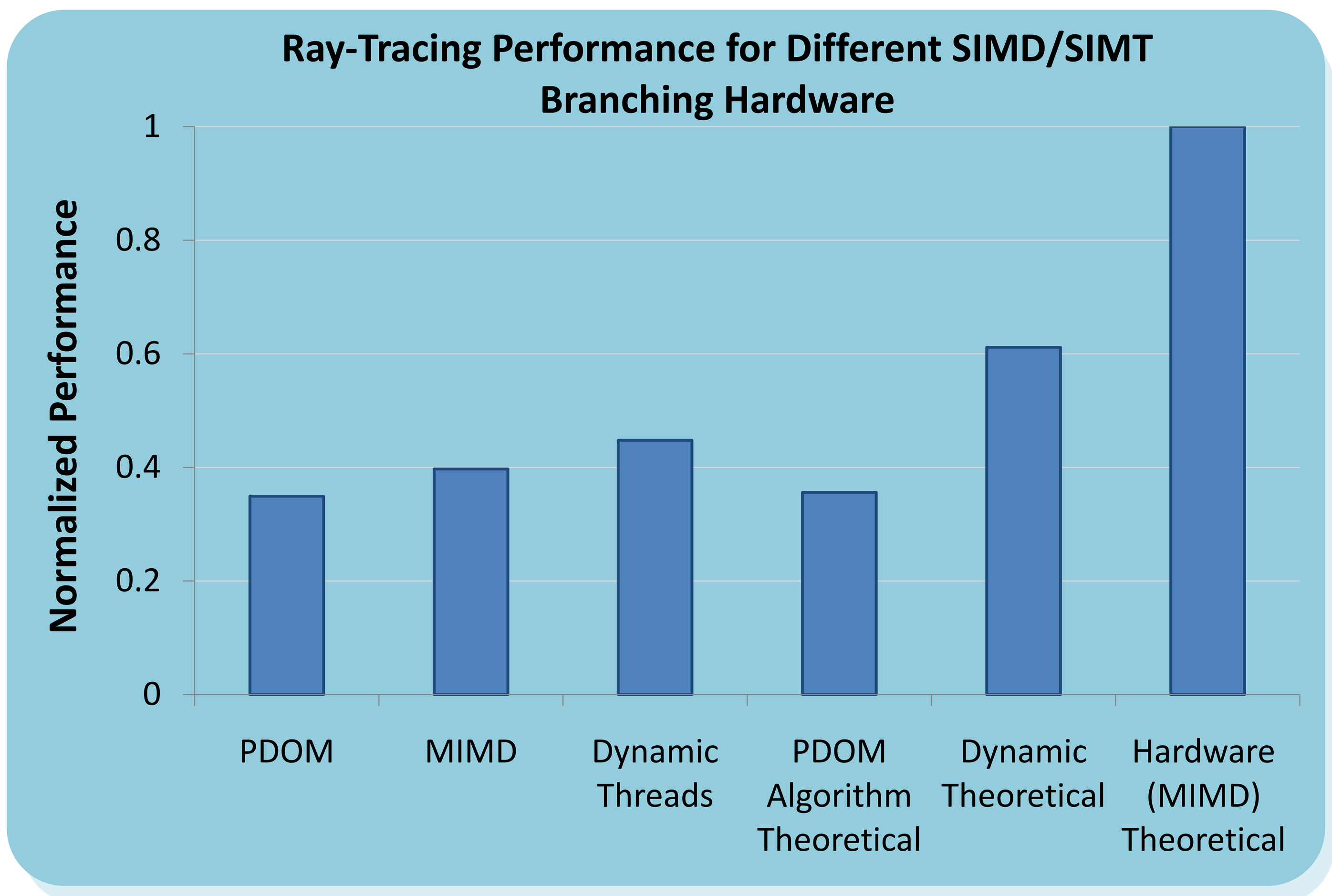
Introduction

Wide SIMD/SIMT machines often require a static allocation of a group of threads (called a Thread Warp) that are executed in lockstep through the entire application. Branching is supported within a thread warp by executing both control flow paths for all threads and disabling processors running threads requiring the opposite control path. Applications requiring complex control flow and heterogeneous thread runtimes often result in low processor efficiency. We introduce a hardware architecture designed to allow for threads to be created dynamically during runtime. Dynamically created threads can be grouped into new thread warps based on the threads control path, allowing for improved processor efficiency.

We use a ray-tracing application as our benchmark since it is an example of both complex control flow and heterogeneous thread runtimes. While ray-tracing applications support large amounts of parallel threads, performance is limited by complex control paths from three data dependent looping operations

Benchmark Scenes



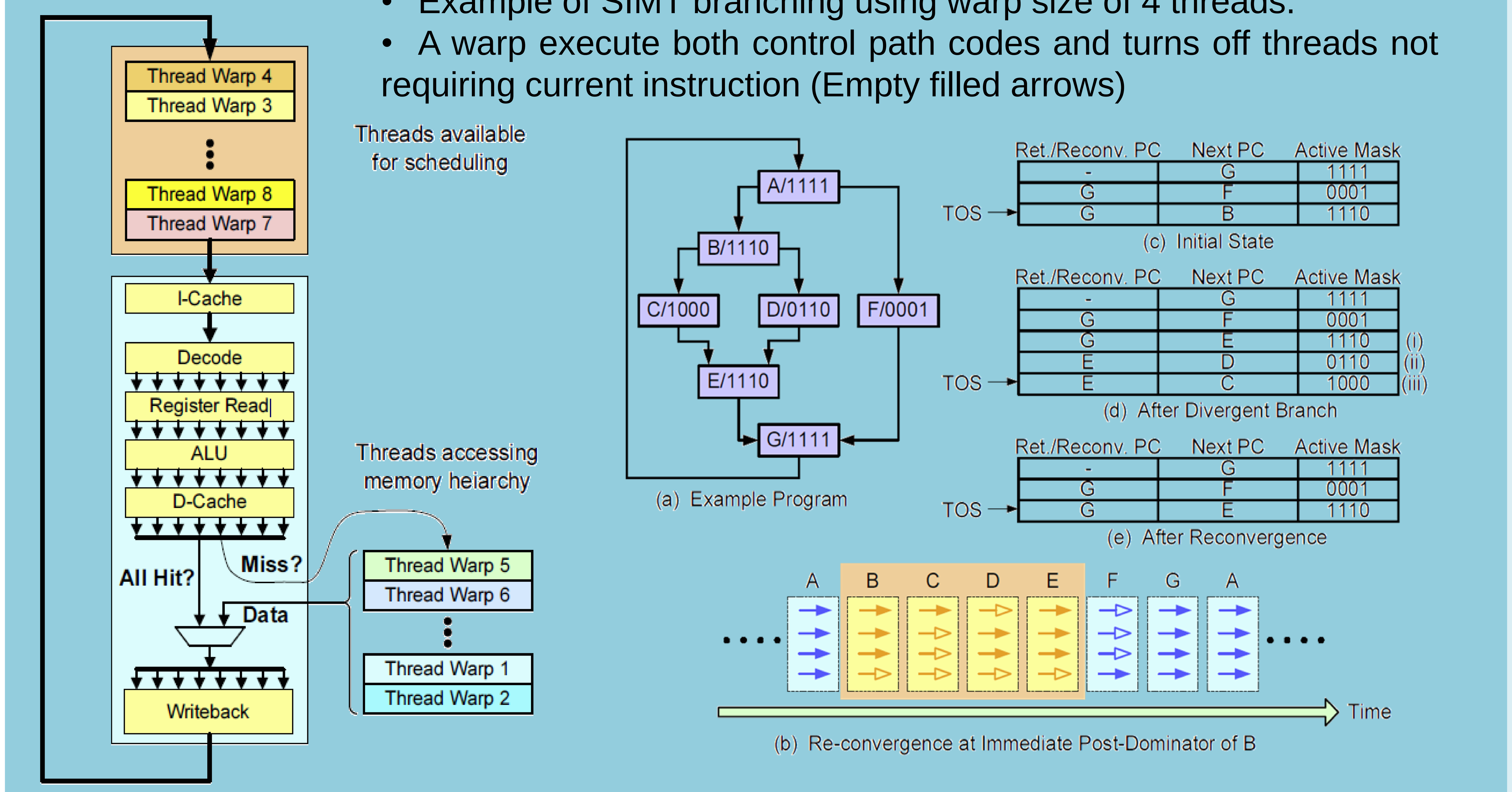


Ray Processing Algorithm

while ray not terminated
while node is not a leaf node
traverse to the next node
while node contains untested objects
perform ray-object intersection test

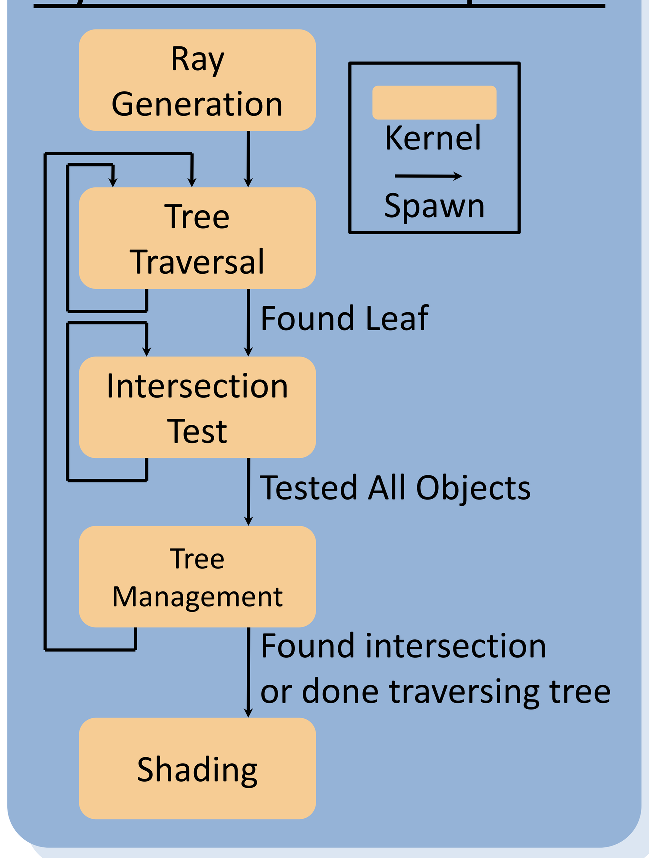
SIMT Pipeline and Control Flow

- Example of SIMT branching using warp size of 4 threads.
- A warp execute both control path codes and turns off threads not requiring current instruction (Empty filled arrows)



Figures from Fung et al. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow

Dynamic Thread Pipeline



Dynamic Thread Algorithm

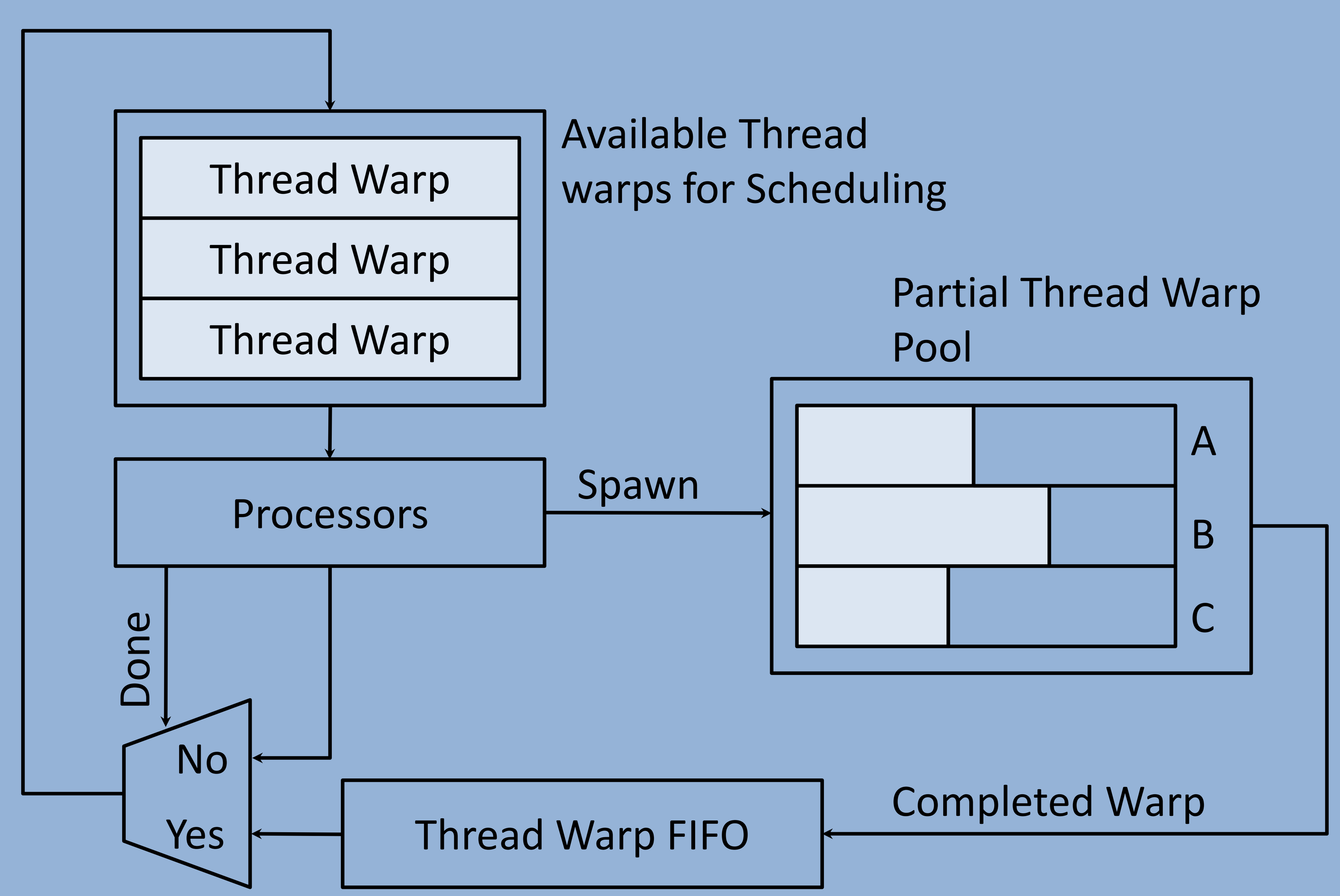
```
1:  restore state from memory
2:  traverse tree to next node
3:  store state to memory
4:  if node is not a leaf then
5:      spawn to line 1
6:  end if
7:  spawn to line 9
8:
9:  restore state from memory
10: ray-object intersection test
11: store state to memory
12: if untested objects remain then
13:     spawn to line 9
14: end if
15: spawn to line 17
16:
17: restore state from memory
18: if ray is not finished then
19:     spawn to line 1
20: end if
21: spawn to shading algorithm
```

Lines 1–7: Tree Traversal; lines 9–15: Intersection Test; lines 17–21: Tree Management

Runtime Thread Creation

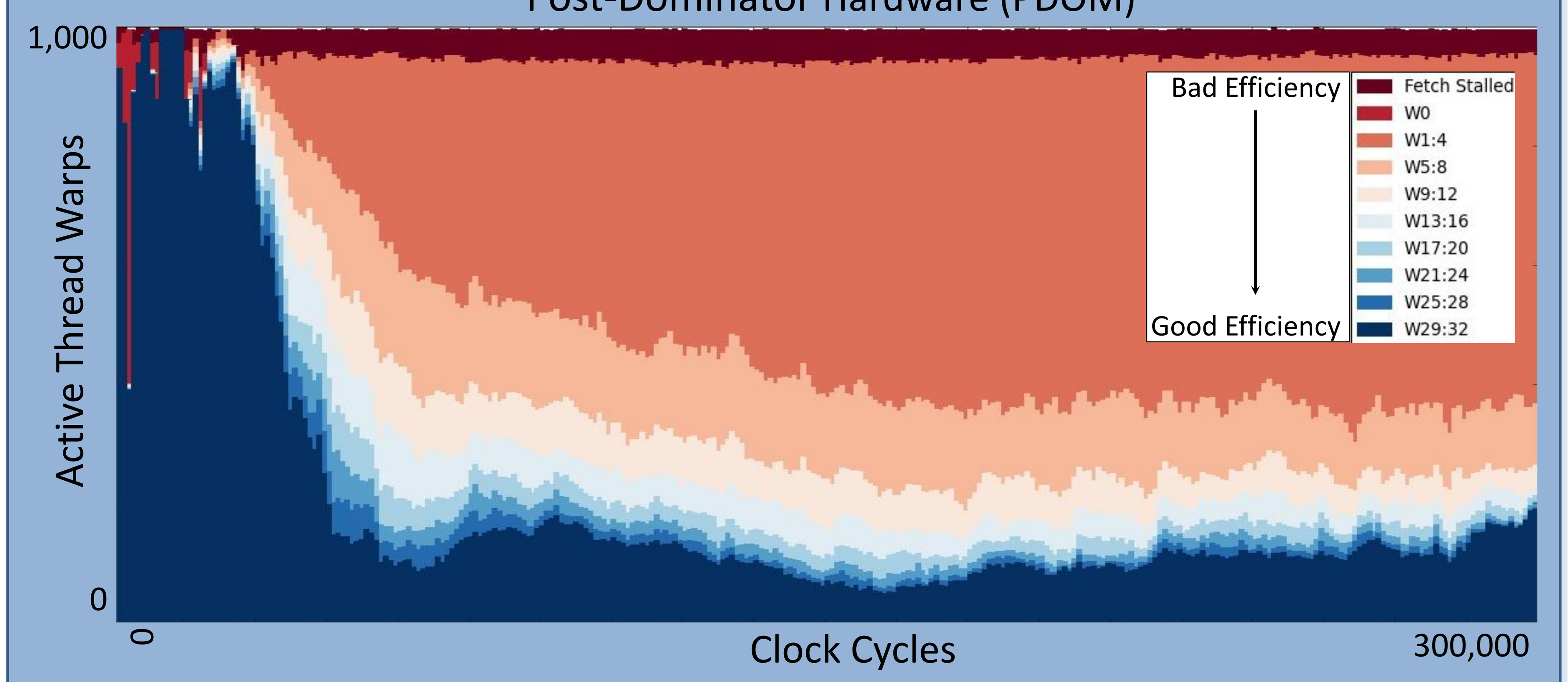
- Replace branching statements that cause low processor efficiency with our new spawn instruction that creates a new threads that begins execution at the original branching PC.
- New threads are combined into new warps based on the starting PC.
- When there are enough threads to create a warp, the warp is then scheduled back onto the processor.

Thread Creation Hardware



Experimental Results

Post-Dominator Hardware (PDOM)



Runtime Thread Creation Hardware