Single Source, Shortest Path

Kevin Nuss

CS 530 Parallel Computing

Statistics and Verification

Every run saves its runtime and message count

Hand check a few small solutions from sequential program

Save solutions from all sequential runs

Every parallel run checks itself against sequential solutions

A few mismatches

- **Too many messages?**
  - **Only occurred in 400k messages or more**
  - **Repeat runs did not have the mismatches**
- **Program bug?**

Moore's Algorithm

The problem involves finding the shortest path from a single, designated source to each of the other vertices in a directed graph that has weighted edges.

Begin with every vertex in a work queue. Try to find a shorter path to another vertex than a direct path from the source. If a shorter path is found, add that destination vertex to the work queue, if not already there. Continue processing vertices that are in the queue looking for shorter paths to other vertices than those already found. If any shorter ones are found, add those destination vertices to the work queue, and so on.

Dijkstra's algorithm uses a similar strategy, but takes advantage of a priority queue which allows each vertex's edges to be examined only once.

Termination Strategy

Dual Pass Ring Termination Algorithm

When finished with its work, a designated initiator passes a 'white' token to the next process. When that next process finishes its work, it passes the token onward. However, if a process sends work to another process that is before it in the ring, it changes the next token it receives to 'black.' A process can not know whether the work it sent was to a process that already sent on a 'white' token. Therefore, when a "white" token does arrive, it does not know whether it is still valid.

If the initiator receives a "black" token, it initiates a new white token for another pass.

If the initiator receives a white token, all work is complete because all processes passed on the token without change.

Methods of Optimization

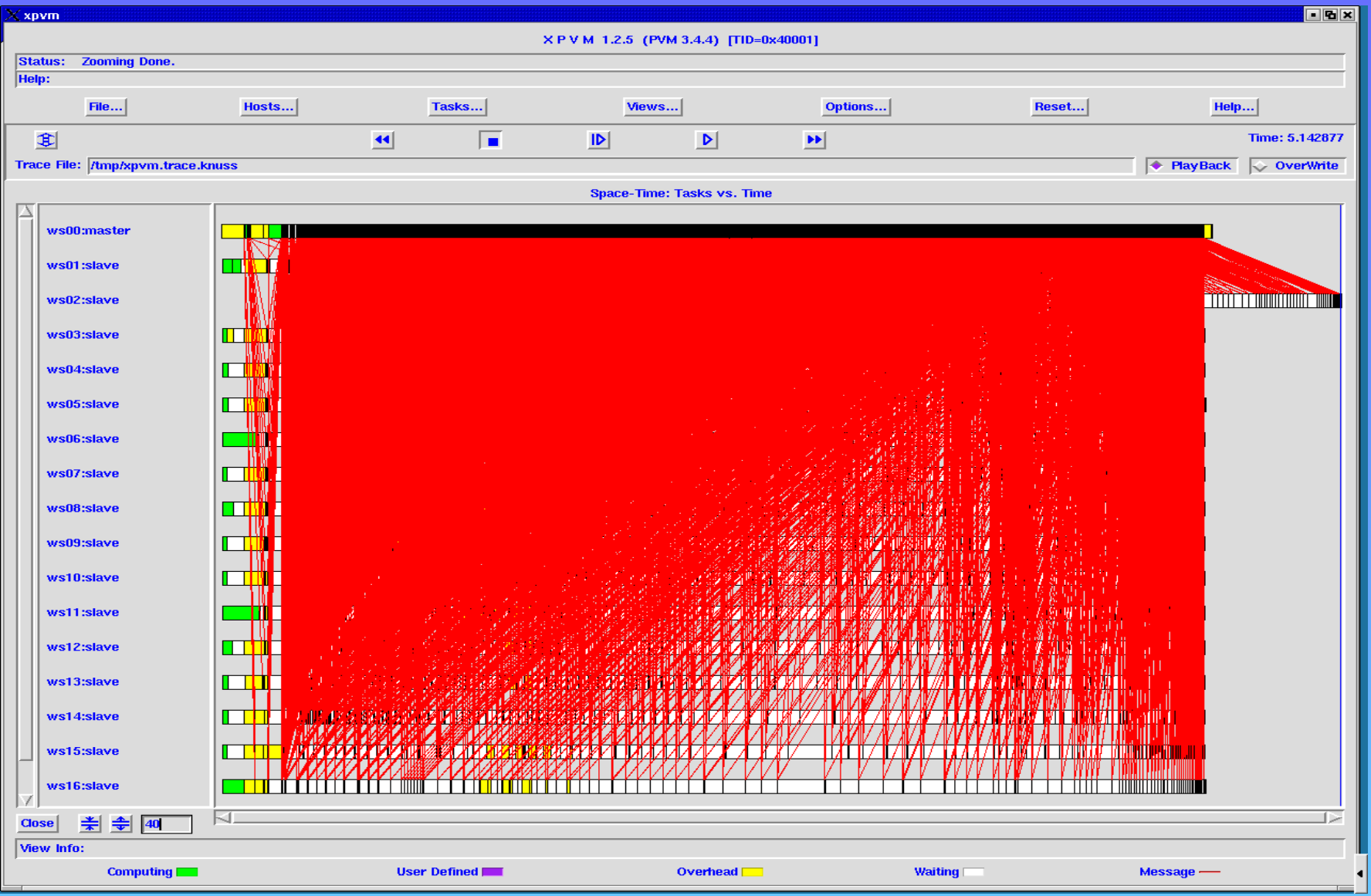
Consolidate outgoing messages to reduce communication bottlenecks.

Receive all available information before calculating shortest paths to reduce wasted effort and wasted outgoing work messages.

Include locally known shortest paths with work messages and tokens to reduce wasted effort and wasted outgoing messages.

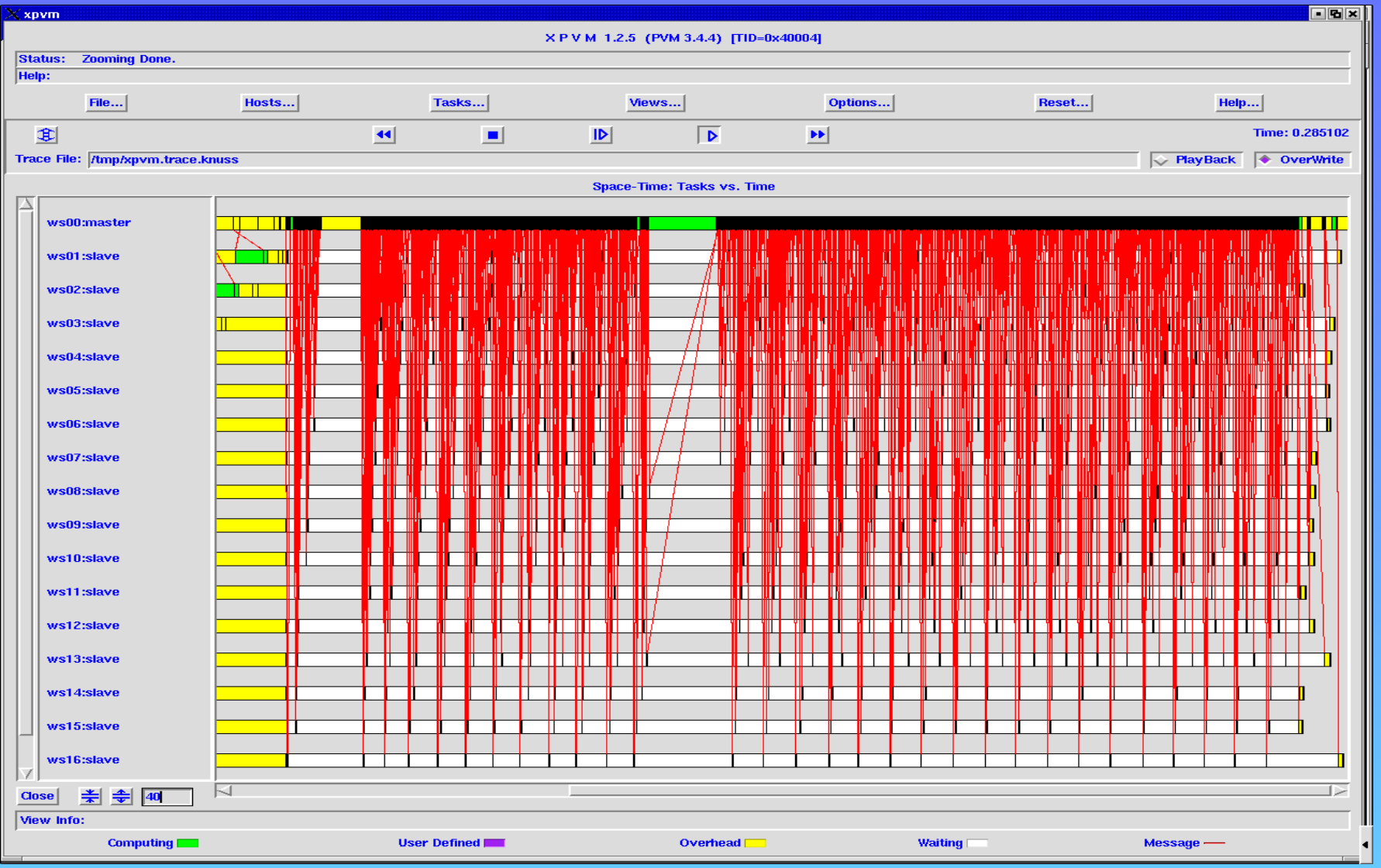
Use a priority queue within a process to improve useage of cpu.

Centralized Work Queue, by the Book



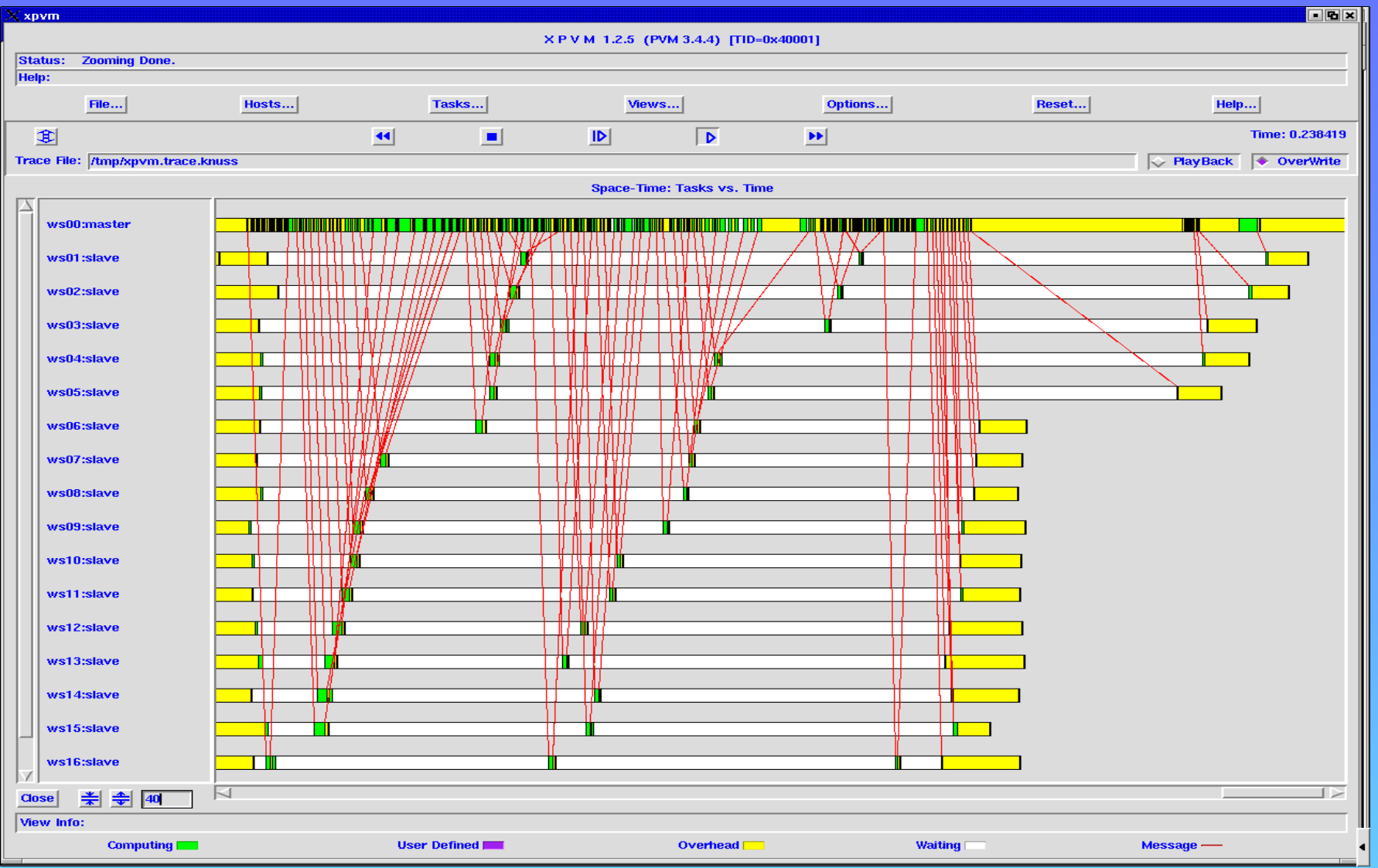
Centralized Work Queue

First Try: Combine Messages from Slaves

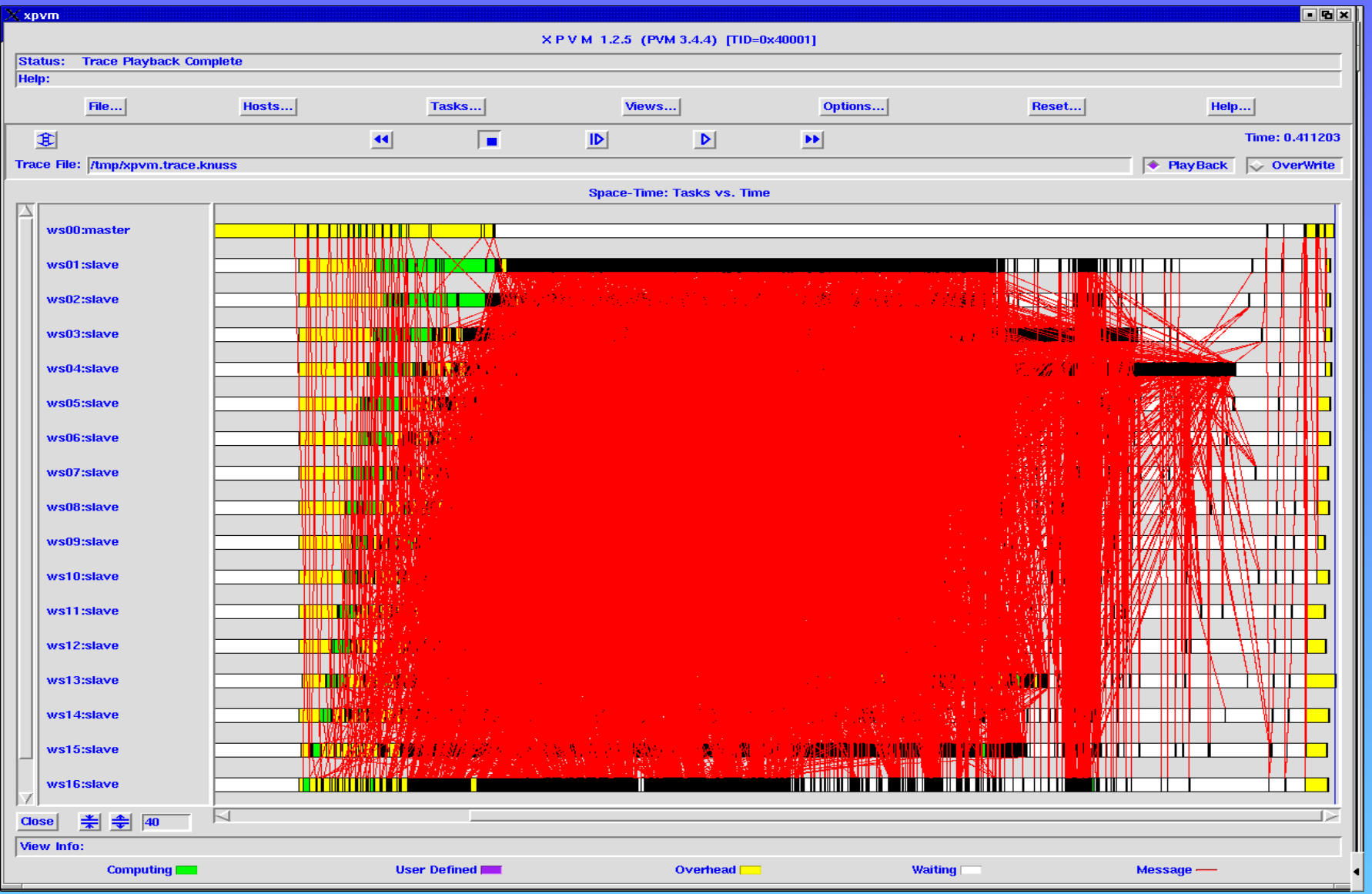


Centralized Work Queue

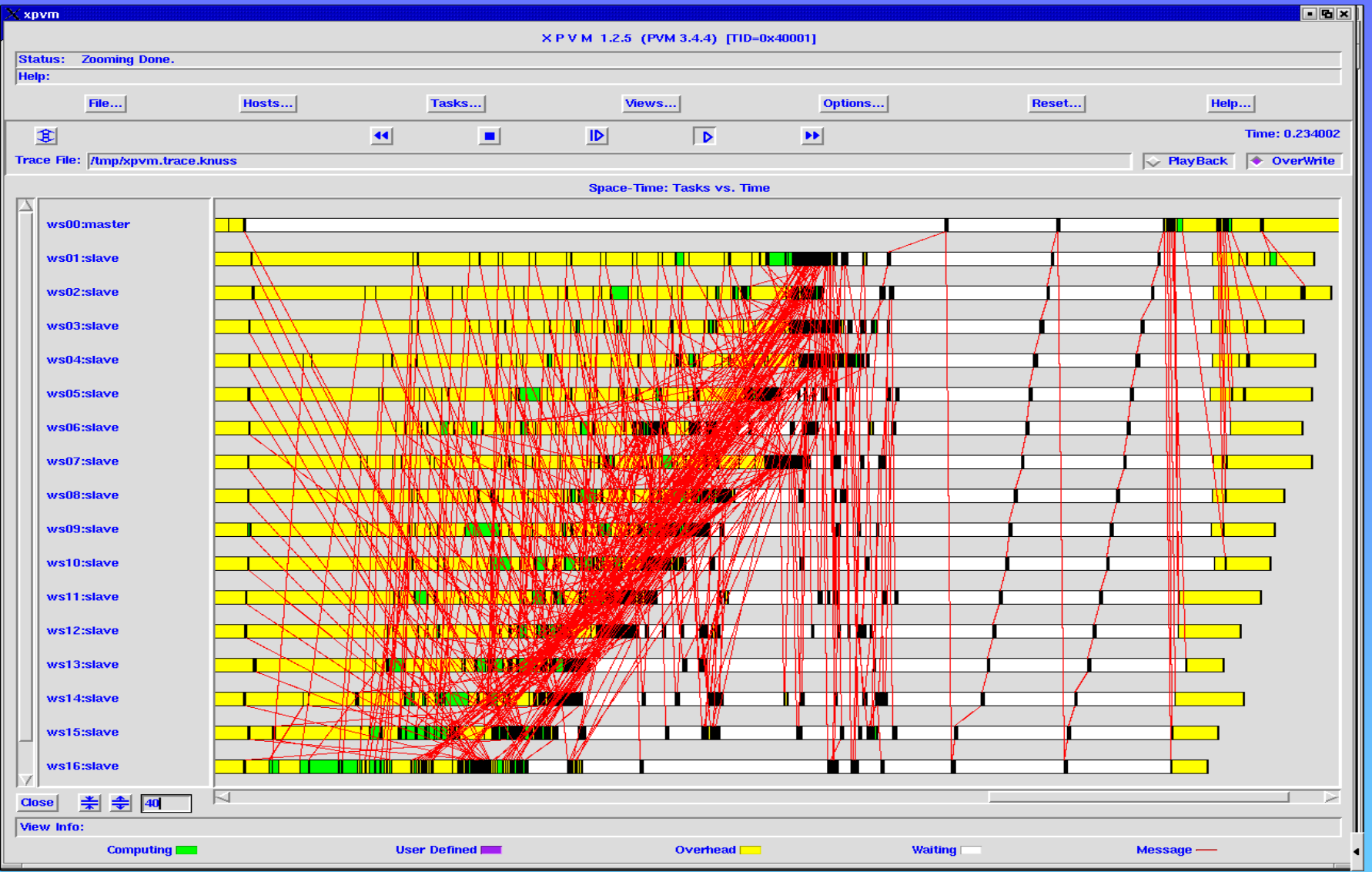
Second Try: Priority Queue & Multiple Vertices Sent to Slave in Each Message



Distributed Queue, by the Book

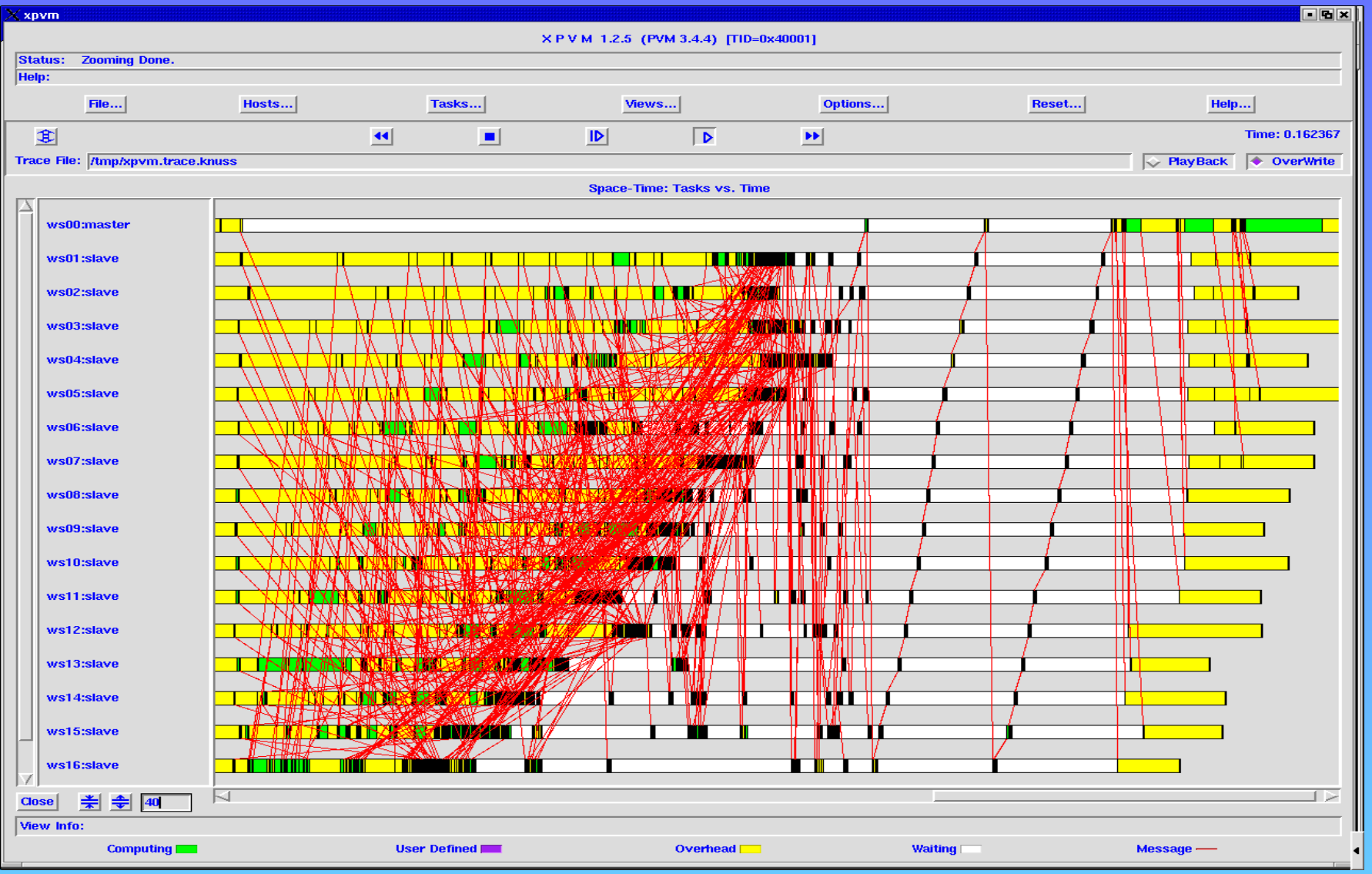


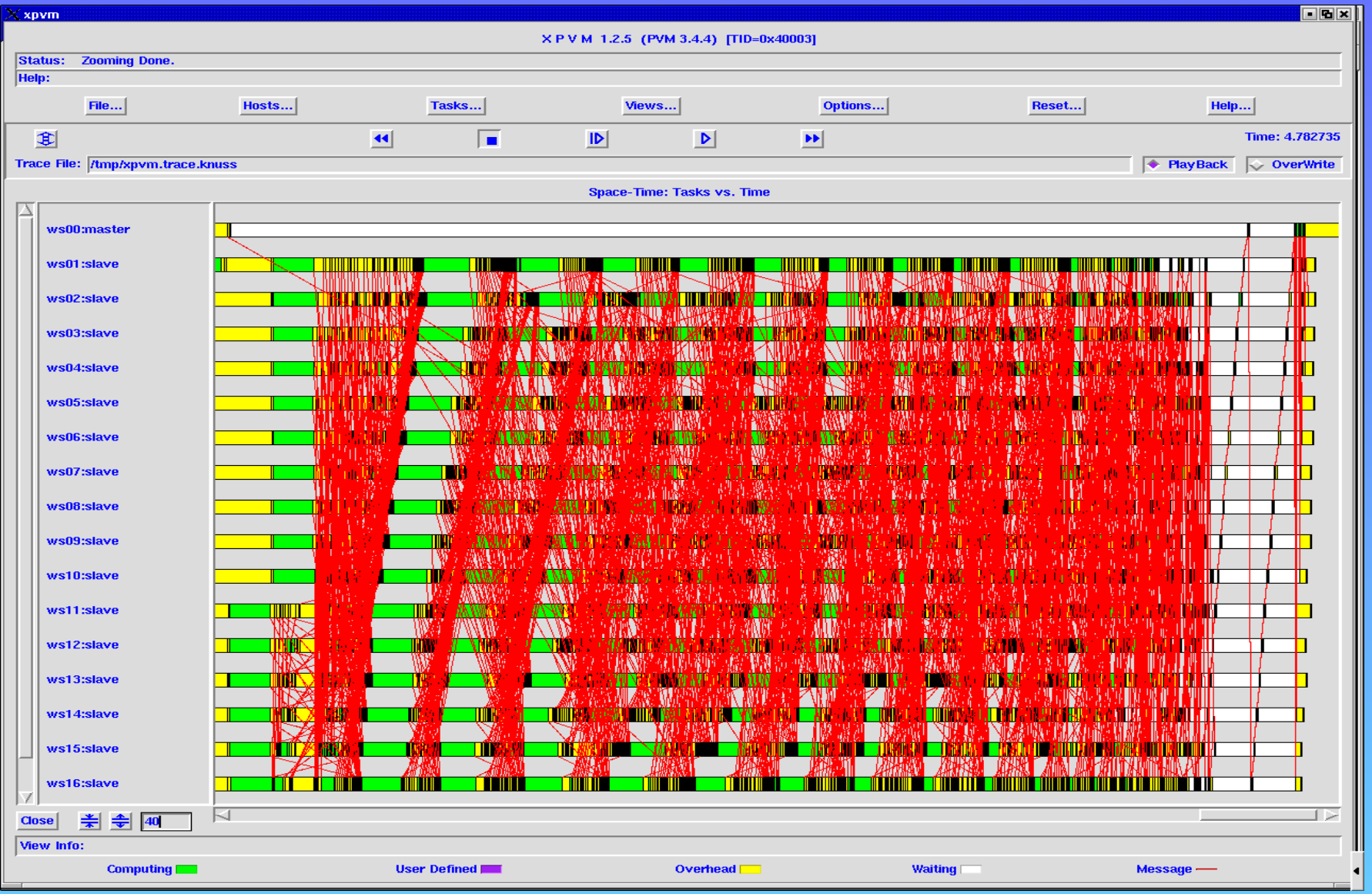
Distributed Queue
First Try: Combine Messages and Send Known Minimums



Distributed Queue

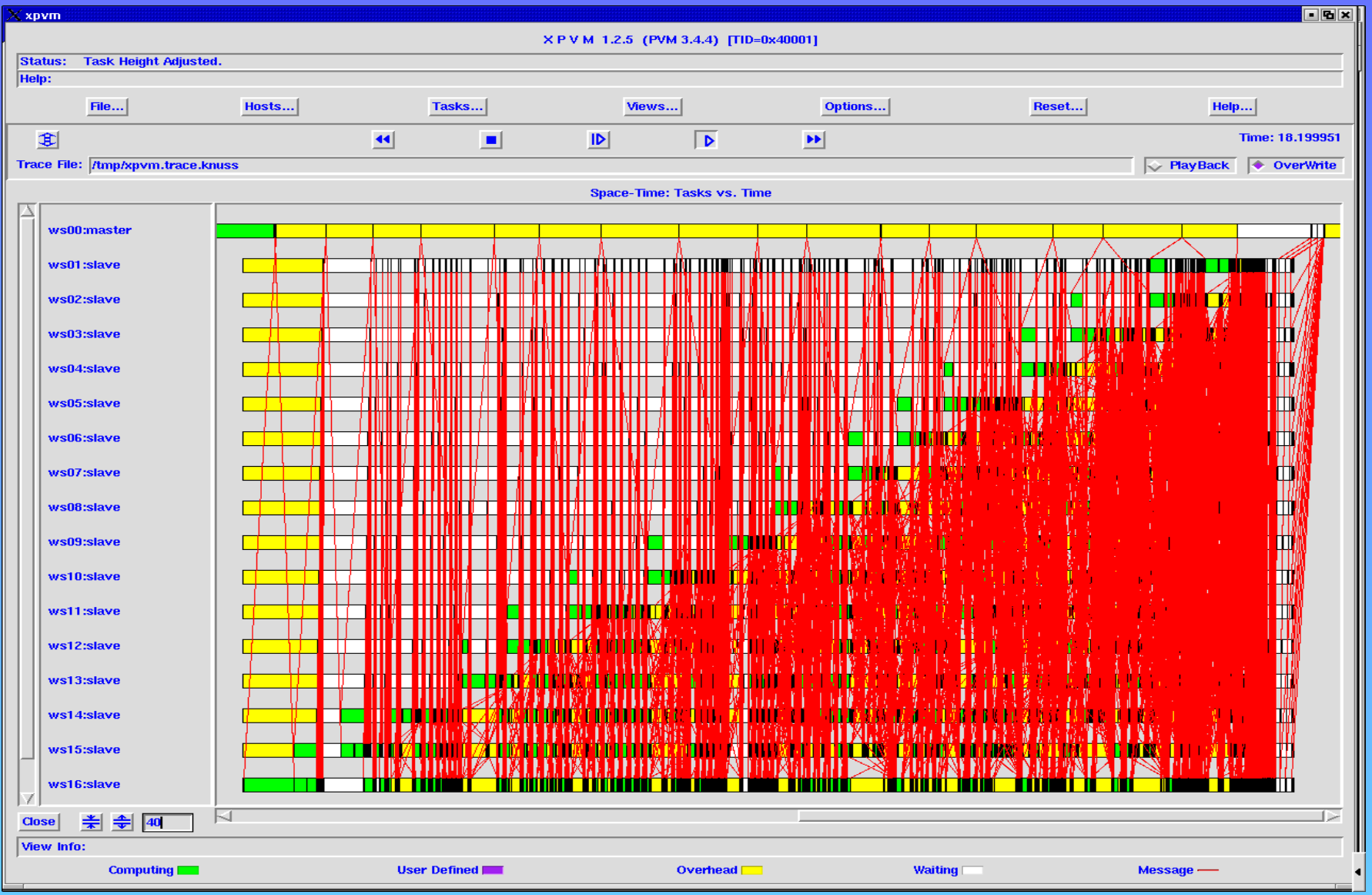
Second Try: Add Priority Queue




xpvm
XPVM 1.2.5 (PVM 3.4.4) [TID=0x40003]
Status: Zooming Done.
Help:
File...
Hosts...
Tasks...
Views...
Options...
Reset...
Help...
Time: 4.782735
Trace File: /tmp/xpvm.trace.knuss
PlayBack
OverWrite
Space-Time: Tasks vs. Time
ws00:master
ws01:slave
ws02:slave
ws03:slave
ws04:slave
ws05:slave
ws06:slave
ws07:slave
ws08:slave
ws09:slave
ws10:slave
ws11:slave
ws12:slave
ws13:slave
ws14:slave
ws15:slave
ws16:slave
Close
40
View Info:
Computing
User Defined
Overhead
Waiting
Message

Distributed Queue, Second Try - 10k Vertices

Processing of Vertices Commences Before All Processes Have Their Data



All Versions of Single Source, Shortest Path

Explanation of Previous Graph

"Sequential Moore's" is the sequential version of Moore's algorithm. It does not use a priority queue.
"Central by the book" is a naive implementation of the parallel pseudocode provided by the textbook.
"Central, First Try" is an initial attempt to improve the textbook's algorithm by combining messages from the slaves to the master.
"Central, Second Try" adds a priority queue to the master and allows it to send more than one vertex to a slave at a time.
"Central with Data Time" includes the time required to distribute the edge data to the slave processes. Each slave gets all edge data.
"Distrib with Data Time" is my best implementation of the distributed work queue but includes the time to distribute the data to the slaves. Work by a slave begins as soon as that slave has its partial set of edge data.
"Distrib, Second Try" is my best implementation of the distributed work queue version of the program. It includes a priority queue within each process.
"Distrib, First Try" does not have a priority queue. Like the second try, each process receives a subset of the edge data and when it discovers a new shorter path, it sends a message to the appropriate process that has that subset. The master process creates the data, distributes it, manages the termination strategy, records runtime, and checks solution.
"Distrib by the Book" is a naive approach to implementing the parallel pseudocode provided by the textbook.
"Sequential" is a sequential implementation of Dijkstra's Algorithm for finding single source, shortest path. It is the best known method.

Possible Improvements

MPI Versions of Fastest Algorithms

Better Queue Handling

Improve Memory Management

- **Release message buffers immediately**
- **Send large data in smaller pieces so message does not compete for memory.**

Improve Centralized Strategy

- **Only send part of edge data to slaves to reduce setup time and increase capacity before virtual memory is needed. Perhaps track which processes already have which edges and manage work queue accordingly.**