Parallel Project – Critical Level Detection

Eran Zimbler

Classes:

PointXY – a simple tuple for represting a point in a two dimensional space..

ControlPoint – inherits from PointXY contains representation of time as int in seconds (simplistic not real epoch).

Cell – representing one Cell on the board by a top left point and size.

ProjectSpace – the 2D space for the task has a collection of cells that have been used, also the definitions of the total space.

Plane – the moving object. Has awareness of the ProjectSpace/Cell and his own location/speed

ViewPath – simple class to hold a vector of cell pointers used for simple optimizations.

Basic idea –

1. Using MPI the first (master) computer loads the data file into memory.
2. The data is being broadcast to all other computers (slaves)
3. Each computer asks for a time slice from the master computer
4. After receiving a time slice (start and end time) each computer starts the main loop for intervals
5. A second loop moves the planes in the time slice (using openMP to parallelize the main movement loop which is possible only because the STL containers are thread-safe and because each plane is its own object)
6. After all planes (that should have moved) have moved another loop runs to generate unordered pairs
7. Now between each pair a ViewPath is being calculated (a list of all cells between top points using a basic algorithm of finding the line then by direction checking the relevant sides of a cell)
8. After having a path checking if each pair can see one another (no other objects in the middle)
9. Using MPI again the slave computer create update message for each plane and sends them to the master computer
10. After day ends the master computer calculates the most hidden plane and prints it to the screen

Master Procces

Load Header of file

Load Planes from file

Broadcast ProjectSpace array

Broadcast Planes

Loop until all slaves have finished

Wait for receive

If reqwork and not day end

Send timeshare

Advance start time

If reqwork and day end

Send endwork

Report slave finish

Else

Wait for update

Receive update messages for planes

Update planes

End Loop

Loop on Planes

Find Plane with max Critical Degree

End Loop

Print Plane

Slave Proccess

Get Project space data (broadcast)

Get Plane data (broadcast)

Report for time share

Loop until endwork

Loop from starttime to end time in interval

Loop on all planes – Time O(n) – n = number of planes

Move plane

Enter into moved planes

End loop

Loop on all moved planes - Time O(m(m-1)) – m is the number of moved planes

Generate unordered pairs

End loop

Loop on all unordered pairs – Time O(t) t =m\*(m-1)/2

Find path between planes in pair

Check if path is clear

Update planes

End loop

Report updates to master

Request more work

End Loop

**Adding CUDA**

While it might improve matters simply to use a strong GPU and replace the openMP calls with CUDA directives , it seems like it might not be a good match for the way I used to calculate path between planes, and this path calculation should be replaced by one with less branching while the best one is supposed to be sibbons algorithm but I couldn't a good explanation ( one that I could understand)

The way to use cuda would be to remove the critical points by allowing real multithreading on the second scale and actually using a thread for each second. In this way while doing much more calculations for the paths and consuming a bit more memory it can utilize the fact that the GPU is much suited to thousands of threads

In short

1. Prebuild all cells in the board
2. Load all objects on slave machines into GPU memory
3. Split to thread for each second
4. Change the path finding algorithm to run by copying instead of pointers
5. Collect all the updates to the plane for a slave and sending them out.