Orbay Degirmencioglu

MAZE SOLVER

Overview:

My approach for solving maze originally using two threads that one start from beginning to end and second thread start from end to beginning and if there is any intersection that means we found a solution so solver backtrack maze to get solution. I choose to use depth first search algorithm because it's easy to use and its more direct than breath first search. Also its DFS faster than BFS for mazes given to us. Another reason to not choose BFS is my earlier prototype that using BFS are much more slower than single thread ones. There is to may consistency in main queue. Even it became faster after design changed to one at time to many at a time for pushing and popping to queue result was still worse than single threaded solution.

I decided not use any shared queue so I choose DFS algorithm for my solver. Every thread has its own queue so there is no need for data protection. So Mutex and atomics are not used in my design. Another addition to solver is creating table for stamping sell and another table for set parent cell of the current maze cell. While DFS going forward it stamps the cell to tell I pass this cell so other thread can look at if current cell is stamped or not. If its stamped that means solver found the solution. Also while we going forward we set parent node of the current node. Because there is no solution list like in the single threaded solution and all thread list my differ and its hard to combine it. Just two thread that goes opposite direction there will be no problem and solution list still viable but if there is more than one thread goes same direction which is in my case then there will be a problem and it will hard to recollect solution data .After solution found than solver start from end to beginning to backtrack and create a solution list.

After two threads working correctly. To fulfil requirement for this assignment another thread that help to forward thread added. Because there are two thread that doing same job these two threads fill each other queues if certain condition is fulfilled. My design if local queue of the main forward has thousand item in it and helper threads queue is empty it splice half of its data to helper thread. Helper

thread also check this condition because main thread may finishes all of chooses it has that means all of them lead to dead-end so it just do not any work. This design solely for the requirement and it's slower than two threaded model. Because splicing list to two cost can be observable.

Adding another helper for backward may speed up but it need to change parent- child design. Also adding another forward helpers may increase speed but all these implementation make design more complex. For adding for threads more than two for each ways means we need thread safe request queue and all threads that reached certain condition pop pointer of local queue and push half of its data to it in the for- loop.

```
MTMazeStudentSolver_2::worker_thread_forward(std::list<Choice> *pChoiceStack)
{

	Choice ch;
	int count = 0;
	pChoiceStack->push_front( firstChoice( pMaze->getStart() ) ) ;
	while (!done)
	{


		if(!(pChoiceStack->size() == 0) )
		     {
				count++;

				ch = pChoiceStack->front();

				if(ch.isDeadend())
				{


					pChoiceStack->pop_front();
				       continue;
				}
				Direction dir =pChoiceStack->front().pChoices.pop_front();
				pChoiceStack->push_front( follow_front( ch.at,dir) );

			      if(pChoiceStack->size()>1000)
			      {
					if(pChoiceStack_forward_helper->empty())
					{

				  pChoiceStack_forward_helper->splice(
				  pChoiceStack_forward_helper->begin(),
				 *pChoiceStack,
				 pChoiceStack->begin(),

		std::next( pChoiceStack->begin(), pChoiceStack->size() / 2 ) );




	printf("(%llu)I Finished <Front>\n",GetCurrentID());
	finished_thread_count++;}
```

This is same single threaded depth first search algorithm with little nuances. First of all I get rid of all try catch block. This function start with finding first node until local stack empty in while loop it pop direction and push that data to follow function that return choose object to inserted to same local queue. As I talked before if stack size reaches thousand number of element it spice to list using splice method. I use list because of this feature. Using vector may be faster but too many popping and pushing in data structure and ability to adding front and back leads to use of list. Popping out choices from choice Stack is required because in sing threaded model it goes until it hits dead-end then add that dead-end to queue, pop again and remove the first choice because that's what we choose but because we need to split our list I need to remove that choose because otherwise helper queue go same way as main forward thread go and this gives no benefit at all. This way helper thread have a list that main thread didn't choose to go. For backward thread this is not needed because there is only one queue at all if it also have helper thread than this approach should be used.

```
Choice MTMazeStudentSolver_2::follow_front( Position at, Direction dir )    {
		ListDirection Choices;
		Direction go_to = dir;
		Direction came_from = reverseDir(dir);

		//pMaze->setMazeCellInfo(at);
		Position Parentpos = at;
		at = at.move(go_to);
		pMaze->setParentInfo(at,Parentpos,go_to);
		do
		{
			Parentpos = at;
			if( at == (pMaze->getEnd()) )
			{
				SolutionFound(at,go_to);

			}

			Choices = pMaze->getMoves(at);
			Choices.remove(came_from);

			if(Choices.size() == 1)
			{
				go_to = Choices.begin();
				at = at.move(go_to);
				came_from = reverseDir(go_to);
				pMaze->setParentInfo(at,Parentpos,go_to);
			}

		} while( Choices.size() == 1 );

		CellCheck(at,go_to,MazeThread::FORWARD);
		Choice pRet( at, came_from, Choices ) ;

		return pRet;
	}
```

Only difference from single threaded model is I am setting(setParentInfo) parent position and stamp and checking cell stamp. For forward positon (at) where came from become cell parent. Also direction recorded for backtrack. Its opposite for backward thread and backward have its similar version of this method. Current positon become parent of old position. I am setting parent data for all cells not for only choices. CellCheck method is checking data of the cell and if its stamped it says solution found and notifies waiting thread to backtrack and collect path. It uses condition variable. If not found it stamps that cell.After found solution it sets atomic value to true so all threads are break from loop and exit.

```
struct Parent
{
	Position parentPos;
	Direction path;
	Parent():parentPos(-1,-1),path(Direction::Uninitialized){}
};
struct MazeInfo
{
	bool isPassed;
	MazeThread passedThread;
	MazeInfo()
	{
		isPassed= false;
		passedThread =MazeThread:: UNINITILAZED;

	}
};
```

These are the structures that I am using for set parent and info for maze. For maze info that using for stamp cell I am not using atomics because all data changed to same value and not changed back also even if thread is miss one intersection(two of them see old values and set true at same time) at some point one thread will be see the flag and notify waiting thread to backtrack. These structures created in maze load but because of they are separate objects it does not effect other single threaded solutions except delay maze creation little bit.

Note: if Unexpected things happens like please try 158759(revision number) that has only two threads.Need to change final value to 1