CSCE 5160 Parallel Processing

Project Reports Due on the last day of classes: May 1, 2019
Final Exam: Monday May 6, 2019: 1:30-3:30, F280
Complete Course Evaluations using the new SPOT (Student Perception Of Teaching)

Review

GPU and CUDA programming
Defining grids, blocks and threads
finding a thread x and y (and possibly z) coordinates to be used to find i, j indexes

Logical organization of threads
(grids, blocks, threads) and physical architecture
threads per core (or CU) cores per SM

Memory organization
CPU Memory (main memory) is not accessible to GPU
GPU memory or Device Memory or Global Memory
accessible to both GPU and CPU
Shared memory \(\rightarrow\) accessible to cores (and threads inside cores) of a SM
Limited in size (32K or 64KBets)
similar to L1 cache

Constant and texture caches \(\rightarrow\) also per SM

Move data between CPU and GPU’s global memory
\texttt{cudaMemcpy(d\_A, h\_A, size, cudaMemcpyHostToDevice);}  
\texttt{cudaMemcpy(h\_C, d\_C, size, cudaMemcpyDeviceToHost);}  

Inside CUDA code, you can move the data from Global memory to Shared or Constant memory

Synchronizing threads in a SM or Grid
\texttt{__syncthreads();}  

Atomic instructions (AtomicAdd, AtomicSub…)

CSCE5160 April 22, 2019
CSCE 5160 Parallel Processing

Discrete Optimizations (or search solutions space) – Chapter 11

In general an optimization attempts to satisfy some constraints while minimizing or maximizing some objectives. We can describe the problem as

$$\text{Maximize (or minimize) } (c^T)^* x \text{ subject to } A x \geq b$$

Here A is a n*n matrix; x is n*1 vector of unknowns; b and c are n*1 vectors.

Consider the example on page 471. We are given the following problem

Minimize $2x_1 + x_2 - x_3 - 2x_4$ subject to

\[
\begin{align*}
5x_1 + 2x_2 + x_3 + 2x_4 &\geq 8 \\
x_1 - x_2 - x_3 + 2x_4 &\geq 2 \\
3x_1 + x_2 + x_3 + 3x_4 &\geq 5
\end{align*}
\]

We need to try to first solve for the unknowns; but we have only 3 equations with 4 unknowns. We need to try to find all possible values that meet the constraints; and then find one set of values that minimizes the cost function.

We can describe this problem as a graph algorithm also.

For now let us assume that each $x_i$ can only take values of 0 and 1.

Under this assumption we refer to problems of this type as "satisfiability"

Let us generate a tree with different values assigned to the various $x_i$ variables.

Two feasible solutions:
1) $x_1 = 1; x_2 = 0; x_3 = 1; x_4 = 1$ but the cost equation becomes 0
2) $x_1 = 1; x_2 = 1; x_3 = 0; x_4 = 1$; the cost equation equals 2

Cost Equation: $2x_1 + x_2 - x_3 - 2x_4$

So we use the second equation as desired solution.
Consider another example. See page 470 which describes the 8 puzzle problem.

Kinds of combinatorial search problem
- Decision problem
- Optimization problem

Examples of Combinatorial Search Problems
- Laying out circuits in VLSI
- Planning motion of robot arms
- Assigning crews to airline flights
- Proving theorems (or proving programs)
- Playing games
Most the problems in this class are NP hard (some NP complete). 
So we will use heuristics.

Normally we use some heuristic functions to decide which node to traverse next.
The estimated cost of a feasible solution from any node \( x \) is sometimes called an Heuristic function

Not all heuristic functions are acceptable.
If the functions gives a lower bound on real cost, then they are useful.

At any time, at node \( x \), we know the actual cost of the solution so far (say \( g(x) \))
If \( h(x) \) is the heuristic estimate from \( x \) to final solution, the expected cost of a feasible solution = \( g(x) + h(x) = f(x) \)

If this cost is more than an already known solution, stop the search

Most common search algorithms can be classified as Depth first and Best first (not Breadth First)

Examples of Depth First Algorithms are

- **Simple backtracking.** Each time an infeasible solution is found backtrack to previous node and try a different alternative.

- **Branch and Bound.** We generate all feasible solutions. 
  Even when a solution is found we continue our search.
  If a new better solution is found, update but continue

- **Iterative Deepening A\(^\ast\) (IDA\(^\ast\)).** When the solution trees are very deep, DFS may waste time by going deep in one part of the tree, where no solution exist

  So, here we perform DFS to only some depth
  If no solution is found, increase the acceptable depth and try again.

Suppose we know how to estimate the cost of a feasible solution from any node already in the tree, then we can decide whether to continue the depth first search or not.
CSCE 5160 Parallel Processing

Best First Search (not breadth first)

In this method we will try to identify the best potential node to traverse.

For example, suppose it is always better to move to the left in a 8 puzzle we then try to expand that node of the search tree first.

A* algorithm is the most commonly used best first algorithm
This is similar to IDA* (Iterative Deepening A*)

We maintain two lists. Open and Closed.

Open is the list of nodes that could be expanded leading to a better solution
Closed is the list of nodes already visited or nodes that will not lead to better solutions.

We will use heuristics for deciding on what is a good Open node to expand
The Open list is sorted, so we always expand the top most node.

CSCE 5160 Parallel Processing

Search Tree Nodes
• Each node represents a problem or sub-problem
• Root of tree: initial problem to be solved
• Children of a node created by adding constraints
• AND node: to find solution, must solve problems represented by all children nodes
• OR node: to find solution, solve any of problems represented by children nodes

AND Trees – only AND nodes
  • Divide and conquer algorithms
OR Trees – Only OR nodes
  • Backtracking and branch and bound algorithms
AND/OR Trees --- contains both AND and OR trees
  • Common with games

• Divide-and-conquer methodology
  • Partition a problem into sub-problems
  • Solve sub-problems
  • Combine solutions to sub-problems
• Recursive: sub-problems may be solved using the divide-and-conquer methodology
If we have shared memory systems
Unsolved subprograms are stored in shared memory (as a pool or stack)
Idle processes/threads access the shared memory to acquire new work
Processors with extra work can store work in shared memory
Good way of load balancing
Shared memory (pool or stack) can be a bottleneck

For message passing processors
Subprograms are stored at one processor
    all processes request work from the single processor (Master/Worker approach)
Work is broadcast to all processors

We can also allow idle processors to request work from neighbors
Or heavily loaded processors to send work to neighbors

We will comeback to load balancing in message passing processors

---

Backtrack algorithms
Use depth first to consider alternate solutions
Backtrack when
    a node has no children (dead end)
    all node’s children have been explored

Example: Crossword puzzle creation
Given a blank crossword puzzle and a dictionary
    Assign letters to blank spaces so that all puzzles words are in the dictionary

A Search Strategy
Identify longest incomplete word in puzzle (break ties arbitrarily)
Look for a word of that length
If we cannot find such a word, backtrack
Otherwise, find longest incomplete word that has at least one letter assigned (break ties arbitrarily)
Look for a word of that length
If cannot find such a word, backtrack
Recurse until a solution is found or all possibilities have been attempted
State Space Tree

- Root of tree is initial, blank puzzle.
- Choices for word 1
- Choices for word 2
- Word 3 choices
Cannot find word. Must backtrack.
Time and Space Complexity

- Suppose average branching factor in state space tree is $b$
- Searching a tree of depth $k$ requires examining

$$1 + b + b^2 + \cdots + b^k = \frac{b^{k+1} - b}{b - 1} + 1 = \theta(b^k)$$

nodes in the worst case (exponential time)
- Amount of memory required is $\Theta(k)$

How to parallelize?
Give each processor a sub-tree?
Suppose $p = b^k$

A process searches all nodes to depth $k$
It then explores only one of sub-trees rooted at level $k$
If $d$ (depth of search) > $2k$, time required by each process to traverse first $k$ levels of state space tree inconsequential

Disadvantages of allocating one sub-tree per process

- In most cases state space tree is not balanced
- Example: in crossword puzzle problem, some word choices lead to dead ends quicker than others
- Alternative: make sequential search go deeper, so that each process handles many sub-trees (cyclic allocation)
Variant of backtrack
Need a way of estimating the optimality of a partial solution
prune the tree by avoiding sub-optimal branches

Consider the 8-puzzle
From initial state to final state
Define Manhattan distance
Other methods like Min-Max are used for game programming (for example, Chess).

Alternates between your moves and opponents moves on your move, optimize profit on opponent’s move minimize loss to you.

One observation. Parallel Search Overhead

Parallel implementations may do more work because they may search more nodes in the tree.

If $W_s$ is the work done by one processor, we can equate the work to the number of tree nodes searched $T_s$.

Likewise $W_p$ is the work done by $p$ processor and we an equate this to $T_p$ as the number of nodes searched.

If $T_p / T_s$ is less than 1; we get super linear speedup.

Otherwise we get less $p$ times speedup.

Note this does not include other overheads such as communication and synchronization.
CSCE 5160 Parallel Processing

Other techniques

Branch and Bound
A* and IDA*

We can use IDA* for min-max

Remember these technique use Depth First Search, but limit the depth

Shared memory implementations of A* and IDA*
Blackboard or Tuple Space
Single Queue vs multiple queues

Load balancing in distributed systems
Either a master node distributes work or workers request work from neighbors
Consider the example of 8 puzzle using depth first search

Each processor grows its tree, and maintains a stack with open (unexplored) nodes

Initial only node A is on the list. Then we will add B and C to the list
When a processor expands B, we will add D, E, F to the list (either ahead of C or after).

How should the list be organized? Queue or Stack?
Two ways of representing the tree
(b). Push nodes on to a stack
  Parent is not kept (placed on closed)

(c). Keep children with parent nodes on together

Nodes are added to Open only if the node is not already in the Open or Closed lists

Once a node is expanded, it is moved to closed list

Node 2 and 6 are explored and removed (or moved to closed)

Next node 3 is being explored

We can either distribute work from the top of the stack or the bottom of the stack
  with a cut off depth or half of the tree

Bottom of stack: bigger unexplored tree
At cut off, works if search tree is uniform
Also, distribute half of the nodes

Work Stealing/distribution techniques
Asynchronous Round Robin
Global Round Robin
Random Polling