

GEM 1501 Problem Solving With Computers

Lecture 10:

Parallelism and Concurrency

Martin Henz

Summary of Previous Lecture

- Turing machines are simple computational models
- All computers are the same, computability is robust
- All computers are the same, tractability is robust

Appendix to last lecture: Finite automata

- Finite automata are one-way Turing machines
- The “eye” only moves to the right (R)
- There is no use writing to the tape, because it will never be written
- Example: Automaton that detects words with an even number of a’s

Finite Automata Cannot Count!

- No finite automaton can decide whether its input sequence contains precisely the same number of a’s and b’s.
- Proof by contradiction.
- Assume that there is such a machine. Let us say it has N states.
- Consider an input sequence with $N + 1$ a’s followed by $N + 1$ b’s.

Parallelism

- Idea: Distribute the work over many workers, who work simultaneously
- Issues:
 - Independence
 - Resources
 - Overhead

Independence

- Some tasks are independent of each other and can easily be parallelized.
Example: $X \leftarrow 3; Y \leftarrow 4;$
- Other tasks depend on each other.
Example: $X \leftarrow 3; Y \leftarrow X;$

Resources

- Parallelism requires more hardware resources. There must be several processors available to do work in parallel.
- Resources incur cost.
- Only for large problems, this investment pays off.

Overhead

- Parallelism often incurs an overhead for combining the work.
- This overhead may be small or large depending on the problem.
- Sometimes the overhead is so large that parallel solutions end up slower than sequential ones.

Example: Summing Salaries

- Sum all salaries in a company
- Need $N/2$ processors
- Need $\log_2 N$ time

Trade-off of Cost and Time

- If we have more processors, we can do more work in parallel.
- If we have less processors, we need to sequentialize.
- The assumption of *expanding parallelism* allows to grow the number of processors with the size of the problems.

Example: Sorting

Subroutine *sort-L*

- if L consists of one element then it is sorted;
- otherwise do the following:
 - split L into two halves, L_1 and L_2
 - call *sort- L_1* ;
 - call *sort- L_2* ;
 - merge resulting lists into a single sorted list

Example: Sorting in Parallel

Subroutine *parallel-sort-L*

- if L consists of one element then it is sorted;
- otherwise do the following:
 - split L into two halves, L_1 and L_2
 - call *parallel-sort- L_1* and *parallel-sort- L_2* simultaneously;
 - merge resulting lists into a single sorted list

Product Complexity

- How can we compare algorithms that require different numbers of processors?
- Product complexity: Time \times Size
- Parallel merge sort has product complexity of $O(N \times N)$
- Sequential merge sort has product complexity of $O(N \times \log N)$
- There are sorting algorithms that require $O(N)$ processors and $O(\log N)$ time

Some Complexity Results

- NP-complete problems can be solved with parallelism in polynomial time.
- The number of processors required is exponential.
- Sequential-PSPACE = Parallel-PTIME

Nick's Class

- Nick's class contains problems that can be solved very fast (some power of the logarithm of N) with only $O(N)$ processors.
- Salary summation and sorting are examples.
- $NC \subseteq P \subseteq NP \subseteq PSPACE$

Mutual Exclusion

- Hotel shower problem
- Keep-trying solution: Starvation
- Blackboard solution: Deadlock
- Solution with two guests

Dining Philosophers

- Philosophers need two chopsticks to eat spaghetti
- Find a way to avoid deadlock and starvation
- Need a central synchronization mechanism

Semaphores

- Programming languages for concurrent programming have constructs for synchronization
- Example: Semaphore
 - S is shared integer variable
 - $request(S)$ decrements S if it is positive and waits otherwise
 - $release(S)$ increments S .
 - Both actions are atomic (cannot be interrupted).

Next Week

- Probabilistic algorithms
- Algorithmics and intelligence