Lecture 24: Parallel Processor Architecture & Algorithms

Processor Architecture

Computer architecture has evolved:

- Intel 8086 (1981): 5 MHz (used in first IBM PC)
- Intel 80486 (1989): 25 MHz (became i486 because of a court ruling that prohibits the trademarking of numbers)
- Pentium (1993): 66 MHz
- Pentium 4 (2000): 1.5 GHz (deep ≈ 30-stage pipeline)
- Pentium D (2005): 3.2 GHz (and then the clock speed stopped increasing)
- Quadcore Xeon (2008): 3 GHz (increasing number of cores on chip is key to performance scaling)

Processors need data to compute on:

Problem: SRAM cannot support more than ≈ 4 memory requests in parallel.
Most of the time program running on the processor accesses local or “cache” memory.

Every once in a while, it accesses remote memory:

Round-trip required to obtain data
Research Idea: Execution Migration

When program running on a processor needs to access cache memory of another processor, it migrates its “context” to the remote processor and executes there:

One-way trip for data access

\[
\text{Context} = \underbrace{\text{ProgramCounter} + \text{RegisterFile} + \ldots}_{\text{few Kbits}} \quad \text{(can be larger than data to be accessed)}
\]

Assume we know or can predict the access pattern of a program

\[
m_1, m_2, \ldots, m_N \quad \text{(memory addresses)}
\]
\[
p(m_1), p(m_2), \ldots p(m_N) \quad \text{(processor caches for each } m_i)\]

Example

\[
p_1 \ p_2 \ p_2 \ p_1 \ p_1 \ p_3 \ p_2
\]
\[
\text{cost}_{\text{mig}}(s, d) = \text{distance}(s, d) + L \quad \leftarrow \text{load latency } L \text{ is a function of context size}
\]
\[
\text{cost}_{\text{access}}(s, d) = 2 \ast \text{distance}(s, d)
\]
if \( s == d \), costs are defined to be 0
Problem

Decide when to migrate to minimize total memory cost of trace. For example:

Example:

\[
\begin{align*}
\text{start at } p_1 & \rightarrow \text{migrate to } p_2 \\
\text{migrate to } p_1 & \rightarrow \text{remote access} \\
\text{remote access} & \rightarrow \text{remote access}
\end{align*}
\]

What can we use to solve this problem?

Dynamic Programming!

Dynamic Programming Solution

Program at \( p \), initially, number of processors = \( Q \)

Subproblems?

Define \( DP(k, p_i) \) as cost of optimal solution for the prefix \( m_1, \ldots, m_k \) of memory accesses when program starts at \( p_1 \) and ends up at \( p_i \).

\[
DP(k + 1, p_j) = \begin{cases} 
DP(k, p_j) + \text{cost}_{\text{access}}(p_j, p(m_{k+1})) & \text{if } p_j \neq p(m_{k+1}) \\
\text{MIN}_{i=1}^{Q}(DP(k, p_i) + \text{cost}_{\text{mig}}(p_i, p_j)) & \text{if } p_j = p(m_{k+1})
\end{cases}
\]

Complexity?

\[
O\left( \frac{N \cdot Q}{\text{no.of subproblems}} \cdot \frac{Q}{\text{cost per subproblem}} \right) = O(NQ^2)
\]

My research group is building a 128-processor Execution Migration Machine that uses a migration predictor based on this analysis.
Lecture 24: Research Areas and Beyond 6.006

Erik’s Main Research Areas

- computational geometry [6.850]
  - geometric folding algorithms [6.849]
  - self-assembly
- data structures [6.851]
- graph algorithms [6.899]
- recreational algorithms [SP.268]
- algorithmic sculpture

Geometric Folding Algorithms: [6.849], Videos Online

Two aspects: design and foldability

- design: algorithms to fold any polyhedral surface from a square of paper [Demaine, Demaine, Mitchell (2000); Demaine & Tachi (2011)]
  - bicolor paper \(\implies\) can 2-color faces
  - OPEN: how to best optimize “scale-factor”
  - e.g. best \(n \times n\) checkerboard folding — recently improved from \(\approx n/2 \rightarrow \approx n/4\)

- foldability: given a crease pattern, can you fold it flat
  - NP-complete in general Bern & Hayes (1996)
  - OPEN: \(m \times n\) map with creases specified as mountain/valley [Edmonds (1997)]
  - just solved: \(2 \times n\) Demaine, Liu, Morgan (2011)
  - hyperbolic paraboloid [Bauhaus (1929)] doesn’t exist [Demaine, Demaine, Hart, Price, Tachi (2009)]
– understanding circular creases
– any straight-line graph can be made by folding flat & one straight cut [Demaine, Demaine, Lubiw (1998); Bern, Demaine, Eppstein, Hayes (1999)]

Self-Assembly

Geometric model of computation

- glue e.g. DNA strands, each pair has strength
- square tiles with glue on each side
- Brownian motion: tiles/constructions — stick together if \( \sum \) glue strengths \( \geq \) temperature

- can build \( n \times n \) square using \( O\left(\frac{\log n}{\log \log n}\right) \) tiles [Rothemund & Winfree 2000] or using \( O(1) \) tiles & \( O(\log n) \) “stages” algorithmic steps by the bioengineer [Demaine, Demaine, Fekete, Ishaque, Rafalin, Schweller, Souvaine (2007)]

- can replicate \( \infty \) copies of given unknown shape using \( O(1) \) tiles and \( O(1) \) stages [Abel, Benbernou, Damian, Demaine, Flatland, Kominers, Schweller (2010)]

Data Structures: [6.851], Videos Next Semester

There are 2 main categories of data structures

- Integer data structures: store \( n \) integers in \( \{0, 1, \cdots u - 1\} \) subject to insert, delete, predecessor, successor (on word RAM)
  - hashing does exact search in \( O(1) \)
  - AVL trees do all in \( O(\log n) \)
  - \( O(\log \log u) \) op van Emde Boas
  - \( O\left(\frac{\log n}{\log \log n}\right)\) op fusion trees: Fredman & Willard
  - \( O\left(\sqrt{\frac{\log n}{\log \log n}}\right)\) op min of above

- Cache-efficient data structures
  - memory transfers happen in blocks (from cache to disk/main memory)
  - searching takes \( \Theta(\log_B N) \) transfers (vs. \( \log n \))
  - sorting takes \( \Theta\left(\frac{N}{B} \log_C \frac{N}{B}\right) \) transfers
  - possible even if you don’t know \( B \) & \( C \)!
(Almost) Planar Graphs: [6.889], Videos Online

- Dijkstra in $O(n)$ time [Henzinger, Klein, Rao, Subramanian (1997)]
- Bellman-Ford in $O\left(\frac{n \log^2 n}{\log \log n}\right)$ time [Mozes & Wolff-Nilson (2010)]
- Many problems NP-hard, even on planar graphs. But can find a solution within $1 + \varepsilon$ factor of optimal, for any $\varepsilon$ [Baker 1994 & Others]:

\[
\begin{align*}
\text{FAST} & \quad \text{CPU} \\
\text{SLOW} & \quad \text{MEMORY} \\
\text{CACHE} & \quad \text{DISK/MAIN} \\
\end{align*}
\]
– run BFS from any root vertex \( r \)
– delete every \( k \) layers
– for many problems, solution messed up by only \( 1 + \frac{1}{k} \) factor (\( \iff k = \frac{1}{\varepsilon} \))
– connected components of remaining graph have \( < k \) layers. Can solve via DP typically in \( \approx 2^k \cdot n \) time

Recreational Algorithms

• many algorithms and complexities of games \([\text{some in SP.268 and our book Games, Puzzles \& Computation (2009)}]\)

• \( n \times n \times n \) Rubik’s Cube diameter is \( \Theta \frac{n^2}{\lg n} \) \([\text{Demaine, Demaine, Eisenstat, Lubiw, Winslow (2011)}]\)

• Tetris is NP-complete \([\text{Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell (2004)}]\)

• balloon twisting any polyhedron \([\text{Demaine, Demaine, Hart (2008)}]\)

• algorithmic magic tricks
Algorithms Classes at MIT: (post 6.006)

- 6.046: Intermediate Algorithms (more advanced algorithms & analysis, less coding)
- 6.047: Computational Biology (genomes, phylogeny, etc.)
- 6.854: Advanced Algorithms (intense survey of whole field)
- 6.850: Geometric Computing (working with points, lines, polygons, meshes, …)
- 6.849: Geometric Folding Algorithms (origami, robot arms, protein folding, …)
- 6.851: Advanced Data Structures (sublogarithmic performance)
- 6.852: Distributed Algorithms (reaching consensus in a network with faults)
- 6.853: Algorithmic Game Theory (Nash equilibria, auction mechanism design, …)
- 6.855: Network Optimization (optimization in graph: beyond shortest paths)
- 6.856: Randomized Algorithms (how randomness makes algorithms simpler & faster)
- 6.857: Network and Computer Security (cryptography)

Other Theory Classes:

- 6.045: Automata, Computability, & Complexity
- 6.840: Theory of Computing
- 6.841: Advanced Complexity Theory
- 6.842: Randomness & Computation
- 6.845: Quantum Complexity Theory
- 6.440: Essential Coding Theory
- 6.441: Information Theory
Top 10 Uses of 6.006 Cushions

10. Sit on it: guaranteed inspiration in constant time (bring it to the final exam)
9. Frisbee (after cutting it into a circle)*
8. Sell as a limited-edition collectible on eBay (they’ll probably never be made again—at least $5)
7. Put two back-to-back to remove branding* (so no one will ever know you took this class)
6. Holiday conversation starter... and stopper (we don’t recommend re-gifting)
5. Asymptotically optimal acoustic paneling (for practicing piano & guitar fingering DP)
4. Target practice for your next LARP* (Live Action Role Playing)
3. Ten years from now, it might be all you'll remember about 6.006 (maybe also this top ten list)
2. Final exam cheat sheet*
1. *Three words: OkCupid profile picture