Three Lectures on Parallel Programming

Keshav Pingali
The University of Texas at Austin
My background

- UT Austin
  - Professor in CS and ECE departments
  - Member of Institute for Computational Engineering and Science (ICES)
- Cornell University
  - Professor in CS and ECE departments
- MIT
  - ScD (Advisor: Arvind)
- IIT Kanpur, India
  - B.Tech.
Intelligent Software Systems group (ISS)

- Faculty
  - Keshav Pingali, CS/ECE/ICES

- Research associates
  - Swarnendu Biswas
  - Vishwesh Jatala

- PhD students
  - Roshan Dathathri
  - Gurbinder Gill
  - Michael He
  - Ian Hendrickson
  - Loc Hoang
  - Yi-Shan Lu
  - Sepideh Maliki
  - Francis Pei

- Visitors from France, India, Norway, Poland, Portugal

- Home page: http://iss.ices.utexas.edu

- Funding: DARPA, NSF, BAE, HP, NEC, NVIDIA...
Parallel computing is changing

**Old World**

- **Platforms**
  - Dedicated clusters *versus* cloud, mobile
- **People**
  - Small number of scientists and engineers *versus* large number of self-trained parallel programmers
- **Data**
  - Structured (vector, dense matrix) *versus* unstructured, sparse

**New World**
The Search for “Scalable” Parallel Programming Models

• Tension between productivity and performance
  – support large number of application programmers with small number of expert parallel programmers
  – performance comparable to hand-optimized codes

• Galois project
  – data-centric abstractions for parallelism and locality
  – operator formulation of algorithms
Some projects using Galois

- BAE Systems (RIPE DARPA program)
  - intrusion detection in computer networks
  - data-mining in hierarchical interaction graphs
- HP Enterprise
  - [ICPE 2016, MASCOTS 2016] workloads for designing enterprise systems
- FPGA tools
  - [DAC’14] “Parallel FPGA Routing based on the Operator Formulation”, Moctar and Brisk
  - [IWLS’18] “Parallel AIG Rewriting” Andre Reis et al. (UFRGS, Brazil), Alan Mishchenko (UCB), et al.
- Multi-frontal finite-elements for fracture problems
  - Maciej Paszynski, Krakow
- 2017 DARPA HIVE Graph Challenge Champion
Data-centric abstractions
Parallelism: Old world

• Functional languages
  – map \( f(e_1,e_2,\ldots,e_n) \)

• Imperative languages
  for \( i = 1, N \)
  
  \[ y[i] = a \times x[i] + y[i] \]

  for \( i = 1, N \)
  
  \[ y[i] = a \times x[i] + y[i-1] \]

  for \( i = 1, N \)
  
  \[ y[2 \times i] = a \times x[i] + y[2 \times i-1] \]

• Key idea
  – find parallelism by analyzing algorithm or program text
  – major success: auto-vectorization in compilers (Kuck, UIUC)
Parallelism: Old world (contd.)

- **Static analysis techniques**
  - points-to and shape analysis
- **Fail to find parallelism**
  - may be there is no parallelism in program?
  - may be we need better static analysis techniques?

Mesh m = /* read in mesh */
WorkList wl;
w.l.add(m.badTriangles());
while (true) {
  if (wl.empty()) break;
  Element e = wl.get();
  if (e no longer in mesh)
    continue;
  Cavity c = new Cavity();
  c.expand();
  c.retriangulate();
  m.update(c); // update mesh
  wl.add(c.badTriangles());
}
Parallelism: New world

- **Parallelism:**
  - Bad triangles whose cavities do not overlap can be processed in parallel
  - Parallelism must be found at runtime

- **Data-centric view of algorithm**
  - Active elements: bad triangles
  - Operator: local view
    - Find cavity of bad triangle (blue);
    - Remove triangles in cavity;
    - Retriangulate cavity and update mesh;
  - Schedule: global view
    - Processing order of active elements
  - Algorithm = Operator + Schedule

- **Parallel data structures**
  - Graph
  - Worklist of bad triangles
**Operator formulation of algorithms**

- **Active node/edge:**
  - site where computation is needed

- **Operator:**
  - local view of algorithm
  - computation at active node/edge

- **Schedule:**
  - global view of algorithm
  - unordered algorithms:
    - active nodes can be processed in any order
    - all schedules produce the same answer but performance may vary
  - ordered algorithms:
    - problem-dependent order on active nodes
**Active nodes**
- Topology-driven algorithms
  - Algorithm is executed in rounds
  - In each round, all nodes/edges are initially active
  - Iterate till convergence
- Data-driven algorithms
  - Some nodes/edges initially active
  - Applying operator to active node may create new active nodes
  - Terminate when no more active nodes/edges in graph

**Operator**
- Morph: may change the graph structure by adding/removing nodes/edges
- Label computation: updates labels on nodes/edges w/o changing graph structure
- Reader: makes no modification to graph
Graph problem: SSSP

- Problem: single-source shortest-path (SSSP) computation
- Formulation:
  - Given an undirected graph with positive weights on edges, and a node called the source
  - Compute the shortest distance from source to every other node
- Variations:
  - Negative edge weights but no negative weight cycles
  - All-pairs shortest paths
  - Breadth-first search: all edge weights are 1
- Applications:
  - GPS devices for driving directions
  - Social network analyses: centrality metrics

Node A is the source
SSSP Problem

- Many algorithms
  - Dijkstra (1959)
  - Bellman-Ford (1957)
  - Chaotic relaxation (1969)
  - Delta-stepping (1998)
- In textbook presentations, they seem unrelated to each other
- Common structure:
  - Each node has a label $d$ that is updated repeatedly
    - initialized to 0 for source and $\infty$ for all other nodes
    - during algorithm: shortest known distance to that node from source
    - termination: shortest distance from source
  - All of them use the same operator
    - $\text{relax-edge}(u,v)$:
      - if $d[v] > d[u] + w(u,v)$
      - then $d[v] \leftarrow d[u] + w(u,v)$

  - $\text{relax-node}(u)$:
    - relax all edges connected to $u$
- Differences between algorithms: schedule
Chaotic relaxation (1969)

- **Active node**
  - node whose label has been updated
  - initially, only source is active

- **Schedule**
  - pick active node at random
  - use a (work)-set or multiset to track active nodes

- **TAO**: unordered, data-driven algorithm

- **Main inefficiency:**
  - number of node relaxations depends on the schedule
  - can be exponential in the size of graph

![Graph with labeled nodes and edges]
Dijkstra’s algorithm (1959)

- **Active nodes**
  - node whose label has been updated
  - initially, only source is active
- **Schedule for processing nodes**
  - ordered by increasing label
- **Implementation of work-set**
  - priority queue ordered by node label
- **Work-efficient ordered algorithm**
  - node is relaxed just once
  - $O(|E| \times \log(|V|))$
- **Main inefficiency:**
  - there is little parallelism for most graphs

![Graph with labels and edges]
Delta-stepping (1998)

• Controlled chaotic relaxation
  – Exploit the fact that SSSP is robust to priority inversions
  – “soft” priorities

• Implementation of work-set:
  – parameter: $\Delta$
  – sequence of sets
  – nodes whose current distance is between $n\Delta$ and $(n+1)\Delta$ are put in the $n^{th}$ set
  – nodes in set $n$ are completed before processing of nodes in set $(n+1)$ are started

• $\Delta = 1$: Dijkstra
• $\Delta = \infty$: Chaotic relaxation
• Picking an optimal $\Delta$:
  – depends on graph and machine
  – high-diameter graph $\rightarrow$ large $\Delta$
  – find experimentally
Bellman-Ford (1957)

- **Algorithm:**
  - execute algorithm in rounds
  - in each round, iterate over all nodes and apply relaxation operator
  - terminate rounds when no node changes value in a round

- **Work-efficiency:**
  - $O(|E| \times |V|)$
  - in each round, we may visit many nodes where there is no work to do
  - however, we do not need a worklist, so there is one less problem for the implementation to worry about

- **TAO analysis:**
  - topology-driven
  - each round is unordered
Summary of SSSP Algorithms

• Chaotic relaxation
  – unordered, data-driven algorithm
    • use sets/multisets for work-set
  – amount of work depends on schedule: can be exponential in size of graph
• Dijkstra’s algorithm
  – ordered, data-driven algorithm
    • use priority queue for work-set
  – $O(|V| \log(|E|))$: work-efficient but little parallelism
• Delta-stepping
  – controlled chaotic relaxation: parameter $\Delta$
  – $\Delta$ permits trade-off between parallelism and work-efficiency
• Bellman-Ford algorithm
  – unordered, topology-driven algorithm
  – $O(|V||E|)$ time
• Operator formulation brings out commonality and differences
  – useful even if you do not care about parallelism
Stencil computation

- **Active nodes**
  - nodes in $A_{t+1}$
- **Operator**
  - five-point stencil
- **Different schedules have different locality**
- **Regular application**
  - grid structure and active nodes known statically
  - application can be parallelized using static analysis

```java
//Jacobi iteration with 5-point stencil
//initialize array A
for time = 1, nsteps
  for <i,j> in [2,n-1]x[2,n-1]
    temp(i,j) = 0.25*(A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))
  for <i,j> in [2,n-1]x[2,n-1]:
    A(i,j) = temp(i,j)
```
Machine learning

• Examples:
  – Page rank: used to rank webpages to answer Internet search queries
  – Recommender systems: used to make recommendations to users in Netflix, Amazon, Facebook etc.
Recommender system

• Problem
  – given a database of users, items, and ratings given by each user to some of the items
  – predict ratings that user might give to items he has not rated yet (usually, we are interested only in the top few items in this set)

• Netflix challenge
  – in 2006, Netflix released a subset of their database and offered $1 million prize to anyone who improved their algorithm by 10%
  – triggered a lot of interest in recommender systems
  – prize finally given to BellKor’s Pragmatic Chaos team in 2009
Data structure for database

- **Sparse matrix view:**
  - rows are users
  - columns are movies
  - $A(u,m) = v$ is user $u$ has given rating $v$ to movie $m$

- **Graph view:**
  - bipartite graph
  - two sets of nodes, one for users, one for movies
  - edge $(u,m)$ with label $v$

- **Recommendation problem:**
  - predict missing entries in sparse matrix
  - predict labels of missing edges in bipartite graph

[Diagram of Matrix View]

[Diagram of Graph View]
One approach: matrix completion

- **Optimization problem**
  - Find $m \times k$ matrix $W$ and $k \times n$ matrix $H$ ($k \ll \min(m,n)$) such that $A \approx WH$
  - Low-rank approximation
  - $H$ and $W$ are dense so all missing values are predicted

- **Graph view**
  - Label of user nodes $i$ is vector corresponding to row $W_{i*}$
  - Label of movie node $j$ is vector corresponding to column $H_{*j}$
  - If graph has edge $(u,m)$, inner product of labels on $u$ and $m$ must be approximately equal to label on edge
One algorithm: SGD

- **Stochastic gradient descent (SGD)**
- **Iterative algorithm:**
  - initialize all node labels to some arbitrary values
  - iterate until convergence
    - visit all edges \((u,m)\) in some order and update node labels at \(u\) and \(m\) based on the residual
- **TAO analysis:**
  - topology-driven, unordered
Summary of discussion of algorithms
von Neumann programming model

Program Execution

Program counter

State update: assignment statement (local view)

Schedule: control-flow constructs (global view)

von Neumann bottleneck [Backus 79]
Data-centric programming model

von Neumann bottleneck [Backus 79]
Connections

• Functional languages: $\lambda$-calculus
  – operator: $\beta$-reduction
  – schedule: applicative order, normal order,…

• Unity (Chandy and Misra)
  – atomic state updates
  – fair-scheduling for unordered algorithms

• Transactional memory (Herlihy and Moss)
  – operators have transactional semantics

• Stencil programs (Steele), Halide (Amarasinghe)
  – finite-differences, image processing
  – do not handle irregular graph algorithms

• Vertex programs (Pregel, GraphLab, Ligra)
  – neighborhood restricted to immediate neighbors of active node: not adequate for pointer-jumping algorithms
  – graph structure cannot be modified
Questions

• How do we implement this model?
  – Shared-memory machines
  – GPUs
  – Distributed-memory machines

• What structure can we exploit for efficiency?
  – (e.g.) Why can we find parallelism statically in finite-differences but not in Delaunay mesh-refinement?
  – Locality
Graph Algorithms
Overview

• **Graph: abstract data type**
  – $G = (V,E)$ where $V$ is set of nodes, $E$ is set of edges $\subseteq V \times V$

• **Structural properties of graphs**
  – Power-law graphs, uniform-degree graphs

• **Graph representations: concrete data type**
  – Compressed-row/column, coordinate, adjacency list

• **Graph algorithms**
  – Operator formulation: abstraction for algorithms
  – Algorithms for single-source shortest-path (SSSP) problem

• **Machine learning algorithms**
  – Page-rank
  – Matrix-completion for recommendation systems
Structural properties of graphs
Graph-matrix duality

- **Graph (V,E) as a matrix**
  - Choose an ordering of vertices
  - Number them sequentially
  - Fill in \(|V| \times |V|\) matrix
    - \(A(i,j)\) is \(w\) if graph has edge from node \(i\) to node \(j\) with label \(w\)
  - Called *adjacency matrix* of graph
  - Edge \((u \rightarrow v)\):
    - \(v\) is *out-neighbor* of \(u\)
    - \(u\) is *in-neighbor* of \(v\)

- **Observations**:
  - Diagonal entries: weights on self-loops
  - Symmetric matrix \(\leftrightarrow\) undirected graph
  - Lower triangular matrix \(\leftrightarrow\) no edges from lower numbered nodes to higher numbered nodes
  - Dense matrix \(\leftrightarrow\) clique (edge between every pair of nodes)
Sparse graphs

• Terminology:
  – Degree of node: number of edges connected to it
  – (Average) diameter of graph: average number of hops between two nodes

• Power-law graphs
  – small number of very high degree nodes (see next slide for example)
  – low diameter
    • “six degrees of separation” (Karinthy 1929, Milgram 1967), on Facebook, it is 4.74
  – typical of social network graphs like the Internet graph or the Facebook graph

• Uniform-degree graphs
  – nodes have roughly same degree
  – high diameter
  – road networks, IC circuits, finite-element meshes

• Random (Erdös-Rényi) graphs
  – constructed by random insertion of edges
  – mathematically interesting but few real-life examples
Airline route map: power-law graph
Road map: uniform-degree graph
Graph representations: how to store graphs in memory
Three storage formats: CSR, CSC, COO

**Coordinate storage**

Labels on nodes are stored in a separate vector (not shown)
Adjacency list representation

Permits you to add and remove edges from graph
Deleting edges: often it is more efficient to just to mark an edge as deleted rather than delete it physically from the list

From: https://www.thecrazyprogrammer.com
Graph algorithms
Overview

• Algorithms: usually specified by pseudocode
• We take a different approach:
  – operator formulation of algorithms
  – data-centric abstraction in which data structures play central role
• Advantages of operator formulation abstraction:
  – Connections between seemingly unrelated algorithms
  – Sources of parallelism and locality become evident
  – Suggests common set of mechanisms for exploiting parallelism and locality for all algorithms
Web search

• When you type a set of keywords to do an Internet search, which web-pages should be returned and in what order?

• Basic idea:
  – offline:
    • crawl the web and gather webpages into data center
    • build an index from keywords to webpages
  – online:
    • when user types keywords, use index to find all pages containing the keywords

• key problem:
  • usually you end up with tens of thousands of pages
  • how do you rank these pages for the user?
Ranking pages

- **Manual ranking**
  - Yahoo did something like this initially, but this solution does not scale

- **Word counts**
  - order webpages by how many times keywords occur in webpages
  - problem: easy to mess with ranking by having lots of meaningless occurrences of keyword

- **Citations**
  - analogy with citations to articles
  - if lots of webpages point to a webpage, rank it higher
  - problem: easy to mess with ranking by creating lots of useless pages that point to your webpage

- **PageRank**
  - extension of citations idea
  - weight link from webpage A to webpage B by “importance” of A
  - if A has few links to it, its links are not very “valuable”
  - how do we make this into an algorithm?
Web graph

- Directed graph: nodes represent webpages, edges represent links
  - edge from u to v represents a link in page u to page v
- Size of graph: commoncrawl.org (2012)
  - 3.5 billion nodes
  - 128 billion links
- Intuitive idea of pageRank algorithm:
  - each node in graph has a weight (pageRank) that represents its importance
  - assume all edges out of a node are equally important
  - importance of edge is scaled by the pageRank of source node
PageRank (simple version)

Graph $G = (V, E)$
$|V| = N$

- Iterative algorithm:
  - compute a series $PR_0, PR_1, PR_2, \ldots$ of node labels
- Iterative formula:
  - $\forall v \in V. \quad PR_0(v) = 1/N$
  - $\forall v \in V. \quad PR_{i+1}(v) = \sum_{u \in \text{in-neighbors}(v)} \frac{PR_i(u)}{\text{out-degree}(u)}$
- Implement with two fields $PR_{\text{current}}$ and $PR_{\text{next}}$ in each node
Page Rank (contd.)

- Small twist needed to handle nodes with no outgoing edges
- Damping factor: $d$
  - small constant: 0.85
  - assume each node may also contribute its pageRank to a randomly selected node with probability $(1-d)$
- Iterative formula
  - $\forall v \in V. \ PR_0(v) = \frac{1}{N}$
  - $\forall v \in V. \ PR_{i+1}(v) = \frac{1-d}{N} + d \sum_{u \in \text{in-neighbors}(v)} \frac{PR_i(u)}{\text{out-degree}(u)}$
- Convergence
  - $\forall v \in V. \ PR(v) = \frac{1-d}{N} + d \sum_{u \in \text{in-neighbors}(v)} \frac{PR(u)}{\text{out-degree}(u)}$
PageRank example

• Nice example from Wikipedia

• Note
  – B and E have many in-edges but pageRank of B is much greater
  – C has only one in-edge but high pageRank because its in-edge is very valuable

• Caveat:
  – search engines use many criteria in addition to pageRank to rank webpages
PageRank discussion

• TAO:
  – Topology: unstructured graph
  – Active nodes
    • Topology-driven
    • Unordered
  – Operator
    • Label computation operator
    • Pull-style

• Interesting application of TAO
  – Can you think of a data-driven version of pageRank?
What we have learned

- **Operator formulation:**
  - data-centric view of algorithms
- **TAO classification**
- **Location of active nodes**
  - Topology-driven algorithms
  - Data-driven algorithms
  - Data-driven algorithm may be more work-efficient than topology-driven one
- **Ordering of active nodes**
  - Unordered algorithms
  - Ordered algorithms
- **Some problems**
  - have both ordered and unordered algorithms (e.g. SSSP)
  - have both topology-driven and data-driven algorithms (e.g. SSSP, pageRank)