Parallel Programming in the Age of Ubiquitous Parallelism

Keshav Pingali
The University of Texas at Austin
Parallelism is everywhere

Texas Advanced Computing Center

Laptops

Cell-phones
Parallel programming?

- 40-50 years of work on parallel programming in HPC domain
- Focused mostly on “regular” dense matrix/vector algorithms
  - Stencil computations, FFT, etc.
  - Mature theory and robust tools
- Not useful for “irregular” algorithms that use graphs, sparse matrices, and other complex data structures
  - Most algorithms are irregular 😞
- Galois project:
  - General framework for parallelism and locality
  - Galois system for multicores and GPUs
What we have learned

• **Algorithms**
  – Yesterday: regular/irregular, sequential/parallel algorithms
  – Today: some algorithms have more structure/parallelism than others

• **Abstractions for parallelism**
  – Yesterday: computation-centric abstractions
    • Loops or procedure calls that can be executed in parallel
  – Today: data-centric abstractions
    • Operator formulation of algorithms

• **Parallelization strategies**
  – Yesterday: static parallelization is the norm
    • Inspector-executor, optimistic parallelization etc. needed only when you lack information about algorithm or data structure
  – Today: optimistic parallelization is the baseline
    • Inspector-executor, static parallelization etc. are possible only when algorithm has enough structure

• **Applications**
  – Yesterday: programs are monoliths, whole-program analysis is essential
  – Today: programs must be layered. Data abstraction is essential not just for software engineering but for parallelism.
Parallelism: Yesterday

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());
}

• What does program do?
  – Who knows
• Where is parallelism in program?
  – Loop: do static analysis to find dependence graph
• Static analysis fails to find parallelism.
  – May be there is no parallelism in program?
  – It is irregular.
• Thread-level speculation
  – Misspeculation and overheads limit performance
  – Misspeculation costs power and energy
Parallelism: Today

- **Data-centric view of algorithm**
  - Bad triangles are **active elements**
  - Computation: **operator** applied to bad triangle:
    - Find cavity of bad triangle (blue);
    - Remove triangles in cavity;
    - Retriangulate cavity and update mesh;

- **Algorithm**
  - Operator: what?
  - Active element: where?
  - Schedule: when?

- **Parallelism**:
  - Bad triangles whose cavities do not overlap can be processed in parallel
  - Cannot find by compiler analysis
  - Different schedules have different parallelism and locality

---

Delaunay mesh refinement

*Red Triangle: badly shaped triangle*
*Blue triangles: cavity of bad triangle*
Example: Graph analytics

- Single-source shortest-path problem
- Many algorithms
  - Dijkstra (1959)
  - Bellman-Ford (1957)
  - Chaotic relaxation (1969)
  - Delta-stepping (1998)

- Common structure:
  - Each node has distance label $d$
  - Operator:
    
    $$\text{relax-edge}(u,v):$$
    
    if $d[v] > d[u] + \text{length}(u,v)$
    
    then $d[v] \leftarrow d[u] + \text{length}(u,v)$

  - Active node: unprocessed node whose distance field has been lowered
  - Different algorithms use different schedules
  - Schedules differ in parallelism, locality, work efficiency
Example: Stencil computation

- **Finite-difference computation**
- **Algorithm:**
  - 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 at compile-time


//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)
Operator formulation of algorithms

- **Active element**
  - Node/edge where computation is needed

- **Operator**
  - Computation at active element
  - Activity: application of operator to active element

- **Neighborhood**
  - Set of nodes/edges read/written by activity
  - Distinct usually from neighbors in graph

- **Ordering**: scheduling constraints on execution order of activities
  - Unordered algorithms: no semantic constraints but performance may depend on schedule
  - Ordered algorithms: problem-dependent order

- **Amorphous data-parallelism**
  - Multiple active nodes can be processed in parallel subject to neighborhood and ordering constraints

Parallel program = Operator + Schedule + Parallel data structure
Nested ADP

- **Two levels of parallelism**
  - Activities can be performed in parallel if neighborhoods are disjoint
    - Inter-operator parallelism
  - Activities may also have internal parallelism
    - May update many nodes and edges in neighborhood
    - Intra-operator parallelism

- **Densely connected graphs (clique)**
  - Single neighborhood may cover entire graph
  - Little inter-operator parallelism, lots of intra-operator parallelism
  - Dominant parallelism in dense matrix algorithms

- **Sparse matrix factorization**
  - Lot of inter-operator parallelism initially
  - Towards the end, graph becomes dense so need to switch to exploiting intra-operator parallelism
Locality

• **Temporal locality:**
  – Activities with overlapping neighborhoods should be scheduled close together in time
  – Example: activities $i_1$ and $i_2$

• **Spatial locality:**
  – Abstract view of graph can be misleading
  – Depends on the concrete representation of the data structure

• **Inter-package locality:**
  – Partition graph between packages and partition concrete data structure correspondingly
  – Active node is processed by package that owns that node

<table>
<thead>
<tr>
<th>src</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>val</td>
<td>3.4</td>
<td>3.6</td>
<td>0.9</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Abstract data structure

Concrete representation: coordinate storage
Dijkstra SSSP: general graph, data-driven, ordered, local computation
Chaotic relaxation SSSP: general graph, data-driven, unordered, local computation
Delaunay mesh refinement: general graph, data-driven, unordered, morph
Jacobi: grid, topology-driven, unordered, local computation
Parallelization strategies: Binding Time

When do you know the active nodes and neighborhoods?

Compile-time
- Static parallelization (stencil codes, FFT, dense linear algebra)

After input is given
- Inspector-executor (Bellman-Ford)

During program execution
- Interference graph (DMR, chaotic SSSP)

After program is finished
- Optimistic Parallelization (Time-warp)

“The TAO of parallelism in algorithms” Pingali et al, PLDI 2011
Galois system

Parallel program = Operator + Schedule + Parallel data structures

• Ubiquitous parallelism:
  – small number of expert programmers (Stephanies) must support large number of application programmers (Joes)
  – cf. SQL

• Galois system:
  – Library of concurrent data structures and runtime system written by expert programmers (Stephanies)
  – Application programmers (Joe) code in sequential C++
    • All concurrency control is in data structure library and runtime system
  – Wide variety of scheduling policies supported
    • deterministic schedules also
Galois: Performance on SGI Ultraviolet

![Graph showing scaling with threads]

<table>
<thead>
<tr>
<th>App</th>
<th>Implementation</th>
<th>Threads</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmr</td>
<td>triangle</td>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>1</td>
<td>155.7</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>512</td>
<td>0.37</td>
</tr>
<tr>
<td>dt</td>
<td>triangle</td>
<td>1</td>
<td>1185</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>1</td>
<td>56.6</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>512</td>
<td>0.18</td>
</tr>
<tr>
<td>bh</td>
<td>splash2</td>
<td>1</td>
<td>&gt;6000</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>1</td>
<td>1386</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>512</td>
<td>3.55</td>
</tr>
<tr>
<td>bc</td>
<td>HPCS SSCA</td>
<td>1</td>
<td>6720</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>1</td>
<td>5394</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>512</td>
<td>21.6</td>
</tr>
<tr>
<td>tri</td>
<td>graphlab</td>
<td>2</td>
<td>531</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>1</td>
<td>7.03</td>
</tr>
<tr>
<td></td>
<td>Galois</td>
<td>512</td>
<td>0.028</td>
</tr>
</tbody>
</table>

Table 2: Serial runtime comparisons to other implementations rounded to the nearest second. Included are runtimes for Galois algorithms at 512 threads. The splash2 implementation of bh timed out after 100 minutes.
Galois: Parallel Metis
GPU implementation

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>SSSP: 23M nodes, 57M edges</th>
<th>SP: 1M literals, 4.2M clauses</th>
<th>DMR: 10M triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BH: 5M stars</td>
<td>PTA: 1.5M variables, 0.4M constraints</td>
<td></td>
</tr>
</tbody>
</table>

|                      | Multicore: 24 core Xeon     | GPU: NVIDIA Tesla             |                  |

- Single Source Shortest Path
- Survey Propagation
- Delaunay Mesh Refinement
- Barnes Hut
- Points-to Analysis
Galois: Graph analytics

- Galois lets you code more effective algorithms for graph analytics than DSLs like PowerGraph (left figure)
- Easy to implement APIs for graph DSLs on top on Galois and exploit better infrastructure (few hundred lines of code for PowerGraph and Ligra) (right figure)
- “A lightweight infrastructure for graph analytics” Nguyen, Lenharth, Pingali (SOSP 2013)
Elixir: DSL for graph algorithms

```
1  Graph [ nodes(node : Node, dist : int) 
2         edges(src : Node, dst : Node, wt : int) ]
3
4  source : Node
5
6  initDist = [ nodes(node a, dist d) ] ➞ 
   [ d = if (a == source) 0 else ∞ ]
7
8  relaxEdge = [ nodes(node a, dist ad) 
9                nodes(node b, dist bd) 
10               edges(src a, dst b, wt w) 
11               ad + w < bd ] ➞ 
12               [ bd = ad + w ]
13
14  init = foreach initDist
15  sssp = iterate relaxEdge ≫ sched
16  main = init ; sssp
```

---

**Algorithm** | **Schedule specification**
---|---
Dijkstra | sched = metric ad ≫ group b
Label-correcting | sched = group b ≫ approx metric ad ≫ unroll 2
Δ-stepping-style | DELTA : unsigned int
                 | sched = metric (ad + w) / DELTA
Bellman-Ford | sssp = for i=1..(NUM_NODES - 1)
            | step
            | step = foreach relaxEdge
SSSP: synthesized vs handwritten

• Input graph: Florida road network, 1M nodes, 2.7M edges
Relation to other parallel programming models

- **Galois:**
  - Parallel program = Operator + Schedule + Parallel data structure
  - Operator can be expressed as a graph rewrite rule on data structure
- **Functional languages:**
  - Semantics specified in terms of rewrite rules like $\beta$-reduction
  - But rules rewrite program, not data structures
- **Logic programming:**
  - (Kowalski) Parallel algorithm = Logic + Control
  - Control $\sim$ Schedule
- **Transactions:**
  - Activity in Galois has transactional semantics (atomicity, consistency, isolation)
  - But transactions are synchronization constructs for explicitly parallel languages whereas Joe programming model in Galois is sequential
Intelligent Software Systems group (ISS)

• **Members**
  – Faculty
    • Keshav Pingali
  – Research associates
    • Andrew Lenharth
    • Rupesh Nasre
  – PhD students
    • Amber Hassaan
    • Rashid Kaleem
    • Donald Nguyen
    • Dimitris Prountzos
    • Xin Sui
    • Gurbinder Singh
• **Visitors from China, France, India, Italy, Portugal**
• **Home page:** [http://iss.ices.utexas.edu](http://iss.ices.utexas.edu)
• **Funding:** NSF, DOE, Qualcomm, Intel, NEC, NVIDIA…
Conclusions

• Yesterday:
  – Computation-centric view of parallelism
• Today:
  – Data-centric view of parallelism
  – Operator formulation of algorithms
  – Permits a unified view of parallelism and locality in algorithms
  – Joe/Stephanie programming model
  – Galois system is an implementation
• Tomorrow:
  – DSLs for different applications
  – Layer on top of Galois

Parallel program = Operator + Schedule + Parallel data structure