Constrained Data-Driven Parallelism

Virendra J. Marathe

with
Tim Harris
Yossi Lev
Victor Luchangco
Mark Moir

Scalable Synchronization Research Group
Oracle Labs
Data-Driven Parallelism

Parallel computation driven by data updates
Data-Driven Parallelism

*Parallel computation driven by data updates*
Data-Driven Parallelism

Parallel computation driven by data updates
Data-Driven Parallelism

Parallel computation driven by data updates
Data-Driven Parallelism

*Parallel computation driven by data updates*
Data-Driven Parallelism

Parallel computation driven by data updates
Data-Driven Parallelism

Parallel computation driven by data updates
Data-Driven Parallelism

Great, but not good enough!
Data-Driven Parallelism
Great, but not good enough!

Does not apply to lots of problem classes
Data-Driven Parallelism

Great, but not good enough!

Does not apply to lots of problem classes

Major performance problems
Illustrative Example: Single-Source Shortest Paths (SSSP)
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm

Now what?
A Data-Driven SSSP algorithm

Directed edge
Update
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm

Directed edge

Update
A Data-Driven SSSP algorithm

Directed edge

Update
A Data-Driven SSSP algorithm

Directed edge

Update
A Data-Driven SSSP algorithm

Directed edge

Update
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm
A Data-Driven SSSP algorithm

Directed edge

Update
A Data-Driven SSSP algorithm

Directed edge
Update
A Data-Driven SSSP algorithm

Too much wasted work
A Data-Driven SSSP algorithm

Directed edge
Update

Too much wasted work
Possible exponential work
Problem: Execution order of tasks

- Need to *constrain* the execution order
- But “balance” is important

Constraint spectrum

Excessive constraints
(little parallelism
 e.g. Dijkstra SSSP)

No constraints
(potential for lots of wasted work
 e.g. Naïve DD)
Problem: Execution order of tasks

- Need to *constrain* the execution order
- But “balance” is important

Excessive constraints (little parallelism e.g. Dijkstra SSSP)

Constraint spectrum

Looking for intermediate sweet spots

No constraints (potential for lots of wasted work e.g. Naïve DD)
Constrained Data-Driven Parallelism

Abstractions to constrain computation structure

- Task: Schedulable unit of computation
- Triggers: Associate data with triggered function

**Phased** Execution of Tasks:
- Partitions tasks into a sequence of *phases*
- Triggers annotated to trigger tasks “right now” or in the “next phase”
- All the “right now” tasks complete before the “next phase” are scheduled
Constrained Data-Driven Parallelism

Language Constructs

1: int * a;
2: *a triggers [deferred] foo();

3: ...
4: *a = ...; // triggers a task that runs foo()
5: ...

6: WaitForTasks(); // wait for all tasks to complete
Constrained Data-Driven Parallelism

Language Constructs

1: int * a;
2: *a triggers [deferred] foo();

3: ...
4: *a = ...; // triggers a task that runs foo()
5: ...

6: WaitForTasks(); // wait for all tasks to complete

Library based implementation at present
Data-Driven SSSP Algorithms
Computation on a Vertex

<table>
<thead>
<tr>
<th>Naïve Data-Driven</th>
<th>Data-Driven with Deferred Triggering</th>
</tr>
</thead>
<tbody>
<tr>
<td>RelaxNeighbors(Vertex&amp; v)</td>
<td>RelaxNeighbors(Vertex&amp; v)</td>
</tr>
<tr>
<td>for all n in v.neighbors do</td>
<td>for all n in v.neighbors do</td>
</tr>
<tr>
<td>int * dist = &amp;n.dist;</td>
<td>int * dist = &amp;n.dist;</td>
</tr>
<tr>
<td>*dist <strong>triggers</strong> RelaxNeighbors(n)</td>
<td>*dist <strong>triggers deferred</strong> RelaxNeighbors(n)</td>
</tr>
<tr>
<td>if (*dist &gt; v.dist + weight(v,n))</td>
<td>if (*dist &gt; v.dist + weight(v,n))</td>
</tr>
<tr>
<td>*dist = v.dist + weight(v,n)</td>
<td>*dist = v.dist + weight(v,n)</td>
</tr>
</tbody>
</table>
## Data-Driven SSSP Algorithms

### Computation on a Vertex

<table>
<thead>
<tr>
<th>Naïve Data-Driven</th>
<th>Data-Driven with Deferred Triggering</th>
</tr>
</thead>
</table>
| **RelaxNeighbors**\(\text{Vertex} \& \text{v}\) \[
\text{for all} \ n \ \text{in} \ v.\text{neighbors} \ \text{do} \\
\quad \text{int} \ * \ \text{dist} = &\text{n.dist}; \\
\quad *\text{dist} \ \text{triggers} \ \text{RelaxNeighbors}(n) \\
\ if \ (*\text{dist} > \text{v.dist} + \text{weight}(v,n)) \\
\quad *\text{dist} = \text{v.dist} + \text{weight}(v,n)
\] | **RelaxNeighbors**\(\text{Vertex} \& \text{v}\) \[
\text{for all} \ n \ \text{in} \ v.\text{neighbors} \ \text{do} \\
\quad \text{int} \ * \ \text{dist} = &\text{n.dist}; \\
\quad *\text{dist} \ \text{triggers deferred} \ \text{RelaxNeighbors}(n) \\
\ if \ (*\text{dist} > \text{v.dist} + \text{weight}(v,n)) \\
\quad *\text{dist} = \text{v.dist} + \text{weight}(v,n)
\] |
A Data-Driven SSSP algorithm (with deferred triggering)

Phase 1

Directed edge

Update
A Data-Driven SSSP algorithm (with deferred triggering)

Directed edge
Update

Phase 1

Phase 2

0 1 1 1 (deferred)

T1

T2

T3

S

Directed edge
Update
A Data-Driven SSSP algorithm (with deferred triggering)

Phase 1

Phase 2

Directed edge

Update

S

1

0

1

1

10

1

1

1

∞

1

∞

1

∞

1

∞

∞
A Data-Driven SSSP algorithm (with deferred triggering)
A Data-Driven SSSP algorithm (with deferred triggering)
A Data-Driven SSSP algorithm (with deferred triggering)
A Data-Driven SSSP algorithm (with deferred triggering)
A Data-Driven SSSP algorithm (with deferred triggering)

Phase 1

Phase 2

Phase 3

Directed edge

Update
A Data-Driven SSSP algorithm (with deferred triggering)

Directed edge

Update

S

1

0

1

1

2

1

1

1

1

1

∞

∞

∞

∞

Phase 1

Phase 2

Phase 3

T5

Little Wasted Work

Directed edge

Update
Implementation

- C++ Library Implementation
- Work-stealing based Task Scheduler
SSSP Scalability Results

Execution time (msecs) vs. # of Threads

- Naïve DD
- DD-Wild

Input graph: ca-HepPh
#vertices: 12008
#edges: 237042

Architecture:
128-thread 2-socket SPARC T2+ system
SSSP Scalability Results

Execution time (msecs)

# of Threads

Architecture:
128-thread 2-socket SPARC
T2+ system

Input graph: ca-HepPh
#vertices: 12008
#edges: 237042

Bellman-Ford

Naïve DD

Dijkstra

Dijkstra
BF-OMP
DD-Wild

Input graph:
ca-HepPh
#vertices: 12008
#edges: 237042
SSSP Scalability Results

Architecture:
128-thread 2-socket SPARC T2+ system

Input graph: ca-HepPh
#vertices: 12008
#edges: 237042

Execution time (msecs) vs. # of Threads

- Dijkstra
- BF-OMP
- DD-Wild
- DD-Phased
- Naïve DD
- Bellman-Ford
- Phased DD

Input graph:
ca-HepPh
#vertices: 12008
#edges: 237042

Architecture:
128-thread 2-socket SPARC T2+ system
SSSP Work Chart

# of Nodes processed relative to Dijkstra

# of Threads

- Naïve DD
- Phased DD
- Bellman-Ford

BF-OMP
DD-Wild
DD-Phased

Dijkstra: Optimal Work
See the paper for…

- Task Groups
  - Phased Task Groups
- More detailed evaluation
  - SSSP
  - Communities
  - Betweenness Centrality
Takeaways...
Takeaways…

- Task ordering matters a lot for many data-driven computations
Takeaways…

- Task ordering matters a lot for many data-driven computations
- Need the ability to constrain task order
Takeaways…

- Task ordering matters a lot for many data-driven computations
- Need the ability to constrain task order
- *Phased* execution of tasks hits a sweet-spot
Takeaways…

- Task ordering matters a lot for many data-driven computations
- Need the ability to constrain task order
- *Phased* execution of tasks hits a sweet-spot
- Performance boost of several orders of magnitude
Takeaways…

- Task ordering matters a lot for many data-driven computations
- Need the ability to constrain task order
- *Phased* execution of tasks hits a sweet-spot
- Performance boost of several orders of magnitude

More to do
- Explore more workload classes – may lead to newer and better abstractions
- Distributed implementation
Hardware and Software

Engineered to Work Together
Data-Driven Computations

- Computation driven by dynamic data dependencies
- Not new
  - Dataflow machines (since 1970s)
  - Event-driven programming – around forever (e.g. interrupt handlers, GUI, sensor networks, SEDA, etc.)
  - Database triggers
- More recently
  - Data triggered threads
  - Data-driven tasks in Habanero
  - Incremental or self-adjusting computation frameworks
Phased Execution of Tasks

Phase 1

Phase 2

Phase 3 (current phase)

Phase 4 (next phase)

Time

Triggers

Completed task

Active task

Next phase task
Takeaways…

- Explore more abstractions
  - Go beyond having the current and next phase
  - DAG
  - Provide primitives to express ordering between task groups (like Dryad, Hive, etc.)

- Go distributed