1. Abstract

Current multicore architectures are not designed to efficiently support fine-grained tasks, leading to poor performance for many applications.

Swarm: A Scalable Architecture for Ordered Parallelism

Mark C. Jeffrey, Suvinay Subramanian, Cong Yan, Joel Emer, Daniel Sanchez
{mcj, suvinay, congy, emer, sanchez}@csail.mit.edu

Tasks are enqueued in a priority queue:

```
prioQueue.enqueue(child, d)
```

where:

```
d = dist + distance(node, child)
```

We target ordered irregular applications:

- Common, but taxing to parallelize
- Comprise tasks that must follow some order
- Tasks dynamically created

We target ordered irregular applications for:

- A new execution model and microarchitecture
- Enables efficient management of speculative tasks

Parent-child dependences

Data dependences

Valid schedule

Tasks

2. Example: Parallelism in Dijkstra

Swarm extracts orders-of-magnitude of parallelism:

- A new execution model and microarchitecture
- Enables efficient management of speculative tasks

Each task:

- Operates on one node
- Is ordered by distance

3. Parallelism Limit Study

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. parallelism</th>
<th>Instrs. per task</th>
<th>Parallelism Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs (graph)</td>
<td>3440x</td>
<td>22</td>
<td>58x</td>
</tr>
<tr>
<td>sssp</td>
<td>793x</td>
<td>32</td>
<td>26x</td>
</tr>
<tr>
<td>astar</td>
<td>419x</td>
<td>195</td>
<td>16x</td>
</tr>
<tr>
<td>msf</td>
<td>158x</td>
<td>40</td>
<td>49x</td>
</tr>
<tr>
<td>des (sim.)</td>
<td>1440x</td>
<td>296</td>
<td>32x</td>
</tr>
<tr>
<td>silo (DB)</td>
<td>318x</td>
<td>1969</td>
<td>17x</td>
</tr>
</tbody>
</table>

1. With perfect speculation, parallelism is plentiful
2. Tasks are tiny
3. Independent tasks are far away in program order

4. Swarm: Execution Model

- Programs comprise timestamped tasks
- Tasks can create children with >= timestamp (T)
- Tasks appear to execute in timestamp order

```
swarm::enqueue(fptr, ts, args...);
```

Conveys new work as soon as possible

5. Swarm: Architecture

- Programs comprise timestamped tasks
- Tasks can create children with >= timestamp (T)
- Tasks appear to execute in timestamp order

```
swarm::enqueue(fptr, ts, args...);
```

Conveys new work as soon as possible

6. Selective Aborts

- Rollback reuses conflict-detection mechanism
- Dependencies found via rollback’s corrective writes

7. Distributed Commits

- Jefferson’s 1985 “Virtual Time” used to retire tasks
- Commit costs are amortized over many tasks

8. Evaluation

- Serial-relative speedups: 43x – 117x
- Outperforms state-of-the-art parallel software by 3x – 18x

Large queues enable a huge speculation window

Order = Distance from source node

2x parallelism