Data-Centric Execution of Speculative Parallel Programs

Mark C. Jeffrey, Suvinay Subramanian, Maleen Abeydeera, Joel Emer, Daniel Sanchez
{mcj, suvinay, maleen, emer, sanchez}@csail.mit.edu


1. Motivation

Multicore systems must exploit locality to scale
- Locality-aware systems are non-speculative
- Speculative systems remain locality-oblivious

To scale, the system must map speculative tasks to cores to minimize data movement
Our solution: spatial hints
- A SW-given hint denotes a task’s likely accesses
- HW maps equal-hint tasks to the same tile
- HW uses hints for locality-aware load balancing

Locality-aware speculation is 3.3x faster at 256 cores
- localizes accesses of most data to one tile
- reduces network traffic
- reduces cycles wasted to mispeculation

2. Baseline: Swarm [MICRO’15]

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

swarm::enqueue(fptr, ts, hint, args...)

Supports unordered (TM) and ordered (TLS) parallelism

3. Spatial Task Mapping with Hints

A spatial hint is an integer value
- given at task creation time
- denotes the data likely to be accessed by the task
We extend the Swarm API with one field, hint:
swarm::enqueue(fptr, ts, hint, args...)

Hardware Mechanisms

1. Map equal-hint tasks to the same tile
2. Serialize equal-hint tasks

Often localizes data accesses within one tile
Avoids running conflicting tasks in parallel

4. Adding Hints to Benchmarks is Easy

- Requires changing a few lines of code per task
- We observed a few common patterns emerge

```c
void sssp(Timestamp dist, Vertex* v)
{
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n: v->neighbors)
            sssp(n);
        //TS=*v/dist + length(v,n),
        //Hint= cacheline(n,n);
    }
}
```

5. Load-Balancing Hints

- Statically mapping hints to tiles can cause hotspots
- Load is hard to measure
- Hint-to-tile mapping using buckets
- Balance committed cycles per unit time across tiles through periodic reconfiguration

6. Evaluation

At 256 cores:
- Random speedups: 5—180x (gmean 58x)
- LBHints speedups: 63—561x (gmean 193x)

---

### Aggregate core cycles

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>L3Hits</th>
<th>Hints</th>
<th>Random</th>
<th>Stealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs</td>
<td>1.8</td>
<td>5.5</td>
<td>4.3</td>
<td>1.5</td>
</tr>
<tr>
<td>sssp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>astar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>des</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>genome</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Mem accs, Aborts, Spill, Stall, Empty

- Mem accs: 1.6x lower
- Aborts: 6.4x lower
- Spill: 3.5x lower
- Stall: 2.0x lower
- Empty: 3.5x lower