Data-Centric Execution of Speculative Parallel Programs

Mark Jeffrey, Suvinay Subramaniam, Maleen Abeydeera, Joel Emer, Daniel Sanchez

MICRO 2016
Executive summary

Many-cores must exploit cache locality to scale

Current speculative systems, e.g. TLS or TM, **do not exploit locality**

**Spatial Hints:** run tasks likely to access the same data in the same place
- A software-given *hint* denotes the data a new task is likely to access
- Hardware maps tasks with the same hint to the same place
- Hardware uses hints to perform locality-aware load balancing

Our techniques make **speculative parallelism practical** at large scale
- It is easy to modify programs to convey locality through hints
- Performance improves by 3.3x at 256 cores
- We reduce network traffic by 6.4x and wasted work by 3.5x
Prior speculative systems scale poorly

**TRANSACTIONAL MEMORY (TM) SCHEDULERS**

Reduce **wasted work** of coarse-grain txns

Limit concurrency: **When** to run a task?

**SPATIAL HINTS**

Make accesses **local** for fine-grain tasks

Less **data movement**: **Where** to run a task?

Spatially map tasks for improved **locality** and less **waste**
Prior non-speculative locality techniques do not work for speculation

**STATIC TASK MAPPING**

Data dependences known a priori
- Linear algebra, Anton 2 [ASPLOS ‘13]

Graph partitioning
- **Localizes** communication and scheduling
- **Slow preprocessing** step
- **Cannot adapt** to imbalance

**DYNAMIC TASK MAPPING**

Work stealing
- **Cheap, local** enqueues
- Steals to **adapt** to imbalance
- **Limited** application types
- Stealing **interferes** with speculation
Baseline Architecture: Swarm [MICRO ‘15]
Baseline Swarm execution model

Programs consist of timestamped tasks
- Tasks can create children tasks with >= timestamp
- Tasks appear to execute in timestamp order

```cpp
class swarm::enqueue(function_pointer, timestamp, arguments...);
```

General execution model supports ordered and unordered parallelism
Baseline Swarm architecture

Speculatively executes tasks out of order

Large hardware task queues
Scalable ordered speculation
Scalable ordered commits

Efficiently supports tiny speculative tasks
Spatial Hints in Action

COMBINING SPECULATION AND LOCALITY
Example: Discrete event simulation (DES)

\[ r \oplus s = t \]

**Tasks**
- \( r=1 \)
- \( A=1 \)
- \( C_0=0 \)
- \( B=1 \)
- \( D_1=0 \)
- \( E_1=0 \)
- \( t=0 \)
- \( D_0=1 \)
- \( E_1=1 \)
- \( t=1 \)

**Order = Simulated time (ns)**
- 0
- 1
- 2
- 3
- 4
- 5
- 6

**Tasks Diagram**
- Nodes: A, B, C, D, E
- Edges: A \(\rightarrow\) C, B \(\rightarrow\) D, D \(\rightarrow\) E, E \(\rightarrow\) t

**Data-Centric Execution of Speculative Parallel Programs**
Extracting parallelism in DES

Execute independent tasks out of order

Parallelism is plentiful despite data dependences

Order = Simulated time (ns)
Speculation scales poorly without locality

Swarm sends new tasks to random tiles
- Good for load balance
- Poor locality hurts scalability beyond 100 cores

Work stealing: a non-speculative scheduler
- Enqueue new tasks locally
- Steal from the most-loaded tile
- Not a good strategy for DES
Where is the locality?

Each task operates on a single gate

The gate is known when the task is created

With fine-grain tasks, most data accessed is known at creation time
Data-centric speculation scales well

**Hints**: map each gate to a statically-chosen tile

Send new tasks for a gate to its corresponding tile

1. Less data movement
2. Conflicts are local, cheap, and less frequent

But we can do better!
Load-balanced speculation scales best

Static gate-to-tile mapping may cause hotspots
- E.g. some gates toggle more frequently

Dynamically remap gates (Hints) across tiles

Programmer knows most of the data accessed

Spatial Hints convey program-level knowledge to exploit locality
Spatial Hints
Implementation
Hint mechanisms are straightforward

SOFTWARE

A **Spatial Hint** is an integer value
- Given at task creation time
- Denotes data likely to be accessed by the task
- E.g. the gate ID in DES

<table>
<thead>
<tr>
<th>Localize most data accesses within a tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialize tasks likely to conflict</td>
</tr>
</tbody>
</table>

HARDWARE

Hashes each new task’s **Hint** to a tile ID
Serializes same-**Hint** tasks

Diagram: 7, 4, 1, 1

DATA-CENTRIC EXECUTION OF SPECULATIVE PARALLEL PROGRAMS

16
Load balance with a level of indirection

Static hint-to-tile mapping may cause *imbalance*

Instead, periodically **remap hints** across tiles to equalize load
“Load” is different for speculation

Non-speculative systems use \# queued tasks as a proxy for load

When imbalanced, speculative systems often
- Don’t run out of work
- Abort more work or strain speculation resources

Remap hints to tiles to balance \# of committed cycles per tile
Adding hints to applications is easy

```c
void desTask(Timestamp ts, GateInput* input) {
    Gate* g = input->gate();
    bool toggledOutput = g.simulateToggle(input);
    if (toggledOutput) {
        // Toggle all inputs connected to this gate
        for (GateInput* i : g->connectedInputs())
            swarm::enqueue(desTask,
                            /*Timestamp*/ ts + delay(g, i),
                            /*Hint*/ i->gate()->id, i);
    }
}
```

One line of code to express the Gate ID as a Hint
Adding hints to applications is easy

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Hint</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>des</td>
<td>Gate ID</td>
<td>Map tasks for same gate to same tile</td>
</tr>
<tr>
<td>nocsim</td>
<td>Router ID</td>
<td>Frequent intra-router communication</td>
</tr>
<tr>
<td>bfs, sssp,</td>
<td>Cache-line address</td>
<td>Several vertices reside on the same line</td>
</tr>
<tr>
<td>astar, color</td>
<td>(Table ID, primary key)</td>
<td>Each task accesses one database tuple</td>
</tr>
<tr>
<td>silo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>genome, kmeans</td>
<td>Multiple</td>
<td></td>
</tr>
</tbody>
</table>
See the paper for more details!

Load balance reconfiguration algorithm

Choice of application hints

Relationship between task size and hint effectiveness
Evaluation
Methodology

Event-driven, Pin-based simulator
Scalability experiments from 1–256 cores
  - Scaled-down systems have fewer tiles

Target system: 256-core, 64-tile chip

- 64 MB shared L3 (1MB/tile)
- 256 KB per-tile L2s
- 16 KB per-core L1s
  - In-order, single-issue, scoreboarded
- 16K task queue entries (64/core)
- 4K commit queue entries (16/core)
Hints make speculation practical on large-scale systems

Load-Balanced Hints 3.3x faster than Random (193x gmean vs 58x)

Load-Balanced Hints
17% – 27% faster than Hints

Stealing is inconsistent across benchmarks
Hints make speculation more efficient

Reduce wasted work by 6.4x
Reduce network traffic by 3.5x
Conclusion

Speculative architectures must exploit locality to scale to 100s of cores

- Important to simplify parallel programming

Spatial Hints convey app-level knowledge to exploit cache locality

Hardware leverages hints by:

- Sending tasks likely to access the same data to the same tile
- **Serializing** tasks likely to conflict
- **Balancing work** in a locality-aware and speculation-friendly way

Our techniques make speculation practical on large-scale systems
Thank you! Questions?

Speculative architectures must exploit locality to scale to 100s of cores
- Important to simplify parallel programming

Spatial Hints convey app-level knowledge to exploit cache locality

Hardware leverages hints by:
- Sending tasks likely to access the same data to the same tile
- Serializing tasks likely to conflict
- Balancing work in a locality-aware and speculation-friendly way

Our techniques make speculation practical on large-scale systems