Chronos: Efficient Speculative Parallelism for Accelerators

MALEEN ABEYDEERA, DANIEL SANCHEZ
ASPLOS 2020
Current hardware accelerators are limited to easy parallelism

**Current Accelerators**

Target easy parallelism

Tasks and dependences known in advance

- e.g.: Deep learning, Genomics

**Chronos**

Targets hard parallelism

Require speculative execution

- e.g.: Graph analytics, simulation, transactional databases
Problem and Insight

Problem

Prior speculation mechanisms (Transactional Memory, Thread Level Speculation) require global conflict detection

Insight

Limit the data that each core can access
Divide work into tiny tasks and send them to data
Coordinate tasks through order constraints

Shared memory system → coherence protocol
Coherence poorly suited for accelerators

Local conflict detection → No coherence needed
Contributions

SLOT (Spatially Located Ordered Tasks): A new execution model that does not require coherence, but relies on task ordering and spatial task mapping to detect conflicts.

Chronos: An implementation of SLOT that provides a common framework for acceleration of applications with speculative parallelism.

https://chronos-arch.csail.mit.edu/
Speculative parallelism with single-object tasks

Discrete Event Simulation (DES) for Digital Circuits

If X6 is being speculatively executed
Prior techniques rely on global conflict detection

Why? No restriction on where a task can run

Relies on coherence protocol to find conflicts
Insight 1: Leveraging spatial task mapping for local conflict detection

Impose restrictions on where a task can run

Conflict detection is local to a core
Insight 2: Leveraging order to ensure atomicity

Banking application:
Each transaction decrements the balance of one account and increments another

<table>
<thead>
<tr>
<th>Account (object)</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>$100</td>
</tr>
<tr>
<td>X</td>
<td>$1500</td>
</tr>
<tr>
<td>Y</td>
<td>$200</td>
</tr>
<tr>
<td>Z</td>
<td>$400</td>
</tr>
</tbody>
</table>

Assign a disjoint timestamp range for each coarse transaction
Benefits of fine-grained tasks

- Increased data locality
- Reduced network traffic
- Increased parallelism

Brings data to compute

Sends compute to data

Transaction 1 (W, Y)
Transaction 2 (Z, Y)

Core 1
Core 2

Order constraints

- Low probability and impact of aborts
- Asynchronous communication

✓ Increased data locality
✓ Reduced network traffic
✓ Increased parallelism
SLOT (Spatially Located Ordered Tasks)

SLOT programs consist of tasks

Tasks can create children tasks through a simple API:

```
slot::enqueue(fn_ptr, timestamp, object-id, arguments...);
```

*Timestamp*: Specifies order. Tasks appear to execute in timestamp order

*Object-id*: Specifies dependences. Tasks with same object-id are treated as data-dependent

Tasks with different object-ids can only communicate through *arguments*
// Simulates an event arriving at a gate
void simToggle(Time time, GateInput input) {
    gate = input.gate;
    toggledOutput = updateState(gate, input);
    if (toggledOutput) {
        // create events for connected gates
        for (GateInput i : gate.connectedInputs()) {
            Time nextTime = time + gate.delay(input, i);
            eventQueue.enqueue(nextTime, i);
        }
    }
}

PriorityQueue<Time, GateInput> eventQueue;
enqueueInitialEvents();
// event loop. Sequentially execute in ts order
while (!eventQueue.empty()){
    (time, input) = eventQueue.dequeue();
    simToggle(time, input);
}

// Simulates an event arriving at a gate
void simToggle(Time time, GateInput input) {
    gate = input.gate;
    toggledOutput = updateState(gate, input);
    if (toggledOutput) {
        // create events for connected gates
        for (GateInput i : gate.connectedInputs()) {
            Time nextTime = time + gate.delay(input, i);
            slot::enqueue(
                simToggle, nextTime, i.gateID, i);
        }
    }
}
enqueueInitialTasks();
slot::run();
Chronos: An implementation of SLOT
Chronos overview

Chronos provides a framework to build accelerators for applications with speculative parallelism

The developer specifies the tasks and how they are implemented
- Either software routines on soft cores, or specialized Processing Elements (PE)

Framework takes care of task management and speculative execution
Task life cycle

- **Create**
  - **Idle**
  - **Dispatch**
  - **Running**
  - **Finish**
  - **Commit**

- **Parent aborted?**
  - **N**
  - **Y**

- **Actions**
  - **Abort**
  - **Discard**
  - **Requeue**
Chronos internal dataflow

- Task creation/dispatch
- Speculative state of finished tasks
- Abort messages
- Requeue task

IDLE (I)
RUNNING (R)
FINISHED (F)
Versioning and commit protocol

**Eager versioning**

- Updates speculative values in place
- Store old values in an undo log

**Main Memory / Cache**

**Core**

**Undo Log**

**Key benefits**
- Makes the common case (commits) fast
- Makes speculative data available before commit

**Commit Protocol (GVT – Global Virtual Time)**

- Tile 0
- Tile 1
- Tile N

**GVT Arbiter**

\[ GVT = \min\{LVT_0, \ldots, LVT_N\} \]

- LVT (Earliest unfinished ts in the tile)
- GVT (Earliest unfinished ts in the system)

**Key benefits**
- Achieves fast and parallel commits
Chronos FPGA implementation

Developed an FPGA implementation of Chronos – up to 16 tiles

Running at 125 MHz

High task throughput – can enqueue, dequeue, execute and commit 8 tasks per cycle on a 16-tile system
Experimental methodology

Four accelerators built using Chronos framework running on AWS FPGAs
• Discrete Event Simulation (DES)
• Maxflow
• Single Source Shortest Paths (SSSP)
• Astar Search

Custom PEs per application: 32-way multithreaded PE, single PE/tile
Baseline: Highly optimized software parallel implementations running on a 40-threaded Xeon AWS instance

<table>
<thead>
<tr>
<th>Platform</th>
<th>AWS Instance</th>
<th>Price ($/hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline CPU</td>
<td>M4.10xlarge</td>
<td>2.00</td>
</tr>
<tr>
<td>FPGA</td>
<td>F1.2xlarge</td>
<td>1.65</td>
</tr>
</tbody>
</table>
Chronos performance vs. 40-threaded Xeon

<table>
<thead>
<tr>
<th>App</th>
<th>Concurrent Max. Tasks</th>
<th>FPGA 1t/ CPU 1t</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>des</td>
<td>256</td>
<td>2.45×</td>
<td>15.3×</td>
</tr>
<tr>
<td>maxflow</td>
<td>192</td>
<td>0.11×</td>
<td>4.3×</td>
</tr>
<tr>
<td>sssp</td>
<td>512</td>
<td>0.24×</td>
<td>3.6×</td>
</tr>
<tr>
<td>astar</td>
<td>192</td>
<td>0.58×</td>
<td>3.5×</td>
</tr>
</tbody>
</table>

Runs many more tasks in parallel

Specialization helps to run a single task efficiently (narrowing the 19× frequency gap with CPU)
Observation:
Most work is ultimately useful
(only 11% of cycles result in wasted work)
See the paper for more

Non-speculative applications

Non-rollback applications

Chronos with RISC-V cores

Projected performance on ASIC Chronos

Chronos resource utilization
Conclusion

Prior speculative parallel systems have relied on cache coherence to detect conflicts, precluding their use in accelerators.

SLOT (Spatially Located Ordered Tasks): A new execution model that does not require coherence, but relies on task ordering and spatial task mapping to detect conflicts.

Chronos: An implementation of SLOT that provides a common framework for acceleration of applications with speculative parallelism.

- Use Chronos to build FPGA accelerators for four challenging applications providing up to 15x speedup over a multicore baseline.

https://chronos-arch.csail.mit.edu/