Harmonizing Speculative and Non-Speculative Execution in Architectures for Ordered Parallelism

MARK C. JEFFREY, VICTOR A. YING, SUVINAY SUBRAMANIAN, HYUN RYONG LEE, JOEL EMER, DANIEL SANCHEZ

MICRO 2018
There is a (false) dichotomy in parallelization

<table>
<thead>
<tr>
<th>SPECULATIVE PARALLELIZATION</th>
<th>NON-SPECULATIVE PARALLELIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplifies parallel programming</td>
<td>Lower overheads</td>
</tr>
<tr>
<td>Uncovers abundant parallelism</td>
<td>Parallel irrevocable actions</td>
</tr>
</tbody>
</table>

Current systems offer all-or-nothing speculation
Goal: Bring non-speculative execution to systems that support ordered parallelism

**Espresso**
- Expressive task-based execution model
- Coordinates concurrent speculative and non-speculative ordered tasks
- 256-core speedups up to 2.5x vs. all-speculative

**Capsules**
- Let speculative tasks safely invoke software-managed speculation
- Enable important system services: e.g. memory allocator that improves performance up to 69x
Espresso in action

THE NEED FOR SPECULATIVE AND NON-SPECULATIVE PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

A ——— 3 ——— B ——— 1 ——— D
|      | 2   |      | 4   |      |
|      | 2   |      | 3   |      |

source

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Order = Distance from source node

Task graph

To be processed
First to visit vertex
Vertex already visited

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Order = Distance from source node

Task graph

Input graph

Example graph:

- A connected to B (weight: 3)
- A connected to C (weight: 2)
- B connected to C (weight: 2)
- B connected to D (weight: 4)
- C connected to D (weight: 3)
- C connected to E (weight: 3)
- D connected to E (weight: 3)

Order:
- A
- C
- B
- D
- E

A - C - B - D - E

A
B
C
D
E

0 1 2 3 4 5 6 7 8

Source node A

Speculative Execution in Architectures for Ordered Parallelism

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

![Input Graph Diagram]

Task graph

![Task Graph Diagram]

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

To be processed
First to visit vertex
Vertex already visited

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

**Input graph**

```
A --- B --- D
<p>| | | |
|    |    |    |</p>
<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B --- C --- E
|    |    |    |
| 2  | 4  | 1  |
|    |    |    |
|    |    |    |

source
```

**Task graph**

```
A --- C --- B --- E --- D
|    |    |    |    |    |
| 0  | 2  | 3  | 3  | 4  |
|    |    |    |    |    |

Order = Distance from source node
```

- **To be processed**
- **First to visit vertex**
- **Vertex already visited**
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

**Input graph**

```
A --- B --- D
  |   ^   |
  3   1   |
  |   v   |
C ----> E
  |   3   |
  2   4   |
  |       |
```

Source node: A

**Task graph**

```
A → C → B → E → D
```

Order = Distance from source node

- To be processed
- First to visit vertex
- Vertex already visited

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph:
- Source node A
- Vertices B, C, D, E
- Edges with weights 2, 3, 4, 3, 3, 4

Task graph:
- Task nodes A, B, C, D, E
- Order: Distance from source node
  - A (source) at 0
  - B at 3
  - C at 3
  - D at 4
  - E at 5

Legend:
- To be processed
- First to visit vertex
- Vertex already visited

Order = Distance from source node
Example: Dijkstra’s algorithm

Finds shortest path tree on a graph with weighted edges

Input graph

Source

Task graph

Order = Distance from source node

To be processed
First to visit vertex
Vertex already visited
Parallelism in Dijkstra’s algorithm?

Order = Distance from source node

Task graph

Order = Distance from source node
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

0 1 2 3 4 5 6 7 8
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Non-speculative

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Non-speculative
Parallelism in Dijkstra’s algorithm?

**Task graph**

Order = Distance from source node

0 1 2 3 4 5 6 7 8

**Dijkstra performance**

Dijkstra on USA-E

1 256 512

Speedup

Data dependences

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Parallelism in Dijkstra’s algorithm?

Task graph

Order = Distance from source node

Dijkstra performance

Valid out-of-order schedule

Data dependences

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra as a Swarm program [MICRO’15]

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}
```
Dijkstra as a Swarm program [MICRO’15]

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}
```
Dijkstra as a Swarm program [MICRO’15]

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}
```
Dijkstra as a Swarm program [MICRO’15]

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}
```
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}

swarm::enqueue(dijkstraTask, 0, sourceVertex);
swarm::run();
Dijkstra as a Swarm program [MICRO’15]

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            Timestamp nDist = dist + weight(v, n);
            swarm::enqueue(dijkstraTask, nDist, n);
        }
    }
}

swarm::enqueue(dijkstraTask, 0, sourceVertex);
swarm::run();
```

Implicit Parallelism

No explicit synchronization

Conveys new work to hardware as soon as possible
Swarm microarchitecture [MICRO’15]

Swarm executes all tasks speculatively and out of order

Large hardware task queues

Scalable ordered speculation

Scalable ordered commits

Efficiently supports thousands of tiny speculative tasks
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Non-speculative
Dijkstra’s algorithm has speculative parallelism

Task graph

Dijkstra performance

Order = Distance from source node

Speedup

Dijkstra on USA-E

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra’s algorithm has speculative parallelism

Task graph

Dijkstra performance

Order = Distance from source node

Dijkstra on USA-E

Speedup

1c 128c 256c

Non-speculative

All-speculative

[MICRO’15]
Dijkstra’s algorithm has speculative parallelism

Dijkstra on USA

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Non-speculative
All-speculative

1c 128c 256c

[DIGITAL’15]

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra’s algorithm has speculative parallelism

Order = Distance from source node

Task graph

Dijkstra performance

Speedup vs. number of cores for Dijkstra's algorithm on USA and cage14.
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Dijkstra on cage14

Speedup

Non-speculative
All-speculative

[MICRO’15]
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Dijkstra on cage14

Non-speculative
All-speculative

[MICRO’15]
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Dijkstra on cage14

MICRO’15

Non-speculative
All-speculative

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Non-speculative
All-speculative

[MICRO’15]
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Dijkstra on USA-E

Dijkstra on cage14

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM

MICRO’15
Dijkstra’s algorithm has speculative parallelism

Task graph

Order = Distance from source node

Dijkstra performance

Speedup

Dijkstra on USA-E

Non-speculative
All-speculative [MICRO’15]

Dijkstra on cage14

Speedup

20%
Dijkstra’s algorithm has speculative parallelism

All-or-nothing speculation unduly burdens programmers

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Dijkstra’s algorithm needs a hybrid strategy
Dijkstra’s algorithm needs a hybrid strategy

**Task graph**
- To be processed
- Finished

**Order** = Distance from source node

0 1 2 3 4 5 6 7

**Order** = Distance from source node
Dijkstra’s algorithm needs a hybrid strategy

Task graph
- To be processed
- Finished

Order = Distance from source node
Dijkstra’s algorithm needs a hybrid strategy

**Task graph**
- □ To be processed
- ■ Finished
- ◼ Running non-speculatively

Run tasks **non-speculatively** when possible
Dijkstra’s algorithm needs a hybrid strategy

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism
Dijkstra’s algorithm needs a hybrid strategy

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism
Dijkstra’s algorithm needs a hybrid strategy

Task graph
- Grey: To be processed
- Diagonal: Finished
- Orange: Running non-speculatively
- Green: Running speculatively

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism
Dijkstra’s algorithm needs a hybrid strategy

- **Task graph**
  - Gray circle: To be processed
  - Black circle: Finished
  - Orange circle: Running non-speculatively
  - Green circle: Running speculatively

**Order** = Distance from source node

- Run tasks **non-speculatively** when possible
- Keep cores busy with **speculative** ordered parallelism
Dijkstra’s algorithm needs a hybrid strategy

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism
Dijkstra’s algorithm needs a hybrid strategy

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism

**Each task must be runnable in either mode**
Dijkstra’s algorithm needs a hybrid strategy

Task graph
- To be processed
- Finished
- Running non-speculatively
- Running speculatively

Run tasks **non-speculatively** when possible

Keep cores busy with **speculative** ordered parallelism

Each task must be runnable in either mode

Tasks in both modes must coordinate on shared data
Espresso reaps the benefits of non-speculative and speculative parallelism

Espresso avoids pathologies and scales best
Espresso

COORDINATING SPECULATIVE AND NON-SPECULATIVE PARALLELISM

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(
                Function -> dijkstraTask,
                pointer dist + weight(v, n),
                n->id,
            Arguments->n);
    }
}
```
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
  if (v->distance == UNSET) {
    v->distance = dist;
    for (Vertex* n : v->neighbors)
      espresso::create(
        dijkstraTask, 
        dist + weight(v, n),
        n->id,
      );
  }
}
```
**Espresso execution model**

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
  if (v->distance == UNSET) {
    v->distance = dist;
    for (Vertex* n : v->neighbors)
      espresso::create(
        dijkstraTask,
        dist + weight(v, n),
        n->id,
        n);
  }
}
```

<table>
<thead>
<tr>
<th>Function pointer</th>
<th>Arguments</th>
<th>Non-Spec.</th>
<th>Spec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>espresso::create</td>
<td>n</td>
<td>barrier</td>
<td>ordered commits</td>
</tr>
<tr>
<td>dijkstraTask</td>
<td>dist, v, n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(
                dijkstraTask, dist + weight(v, n),
                n->id, n);
    }
}
```
Espresso execution model

Programs consist of tasks that run *speculatively* or *non-speculatively*

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(
                dijkstraTask,      // Function pointer
                dist + weight(v, n), // Arguments
                n->id,               // Arguments
            );
    }
}
```

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locale</td>
<td>Timestamp</td>
<td>Non-Spec. Spec.</td>
</tr>
<tr>
<td>barrier</td>
<td>mutex</td>
<td>ordered commits reduce conflicts</td>
</tr>
</tbody>
</table>
**Espresso execution model**

Programs consist of tasks that run *speculatively* or *non-speculatively*

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create(dijkstraTask, dist + weight(v, n), n->id, n);
    }
}
```

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Function pointer</th>
<th>Non-Spec.</th>
<th>Spec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locale</td>
<td></td>
<td>barrier</td>
<td>ordered commits</td>
</tr>
<tr>
<td>Timestamp</td>
<td>espresso::create</td>
<td>mutex</td>
<td>reduce conflicts</td>
</tr>
<tr>
<td>Tasks in either mode can coordinate access to shared data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Espresso task dispatch

Espresso supports three task types that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create< type >(dijkstraTask,
                                     dist + weight(v, n),
                                     n->id,
                                     n);
    }
}
```
Espresso supports three task types that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors) {
            espresso::create< type >(dijkstraTask,
                                      dist + weight(v, n),
                                      n->id,
                                      n);
        }
    }
}
```
Espresso supports three task types that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create< SPEC >(
                dijkstraTask, 
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

Tile

<table>
<thead>
<tr>
<th>Dispatch Candidates</th>
<th>7</th>
<th>SPEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>SPEC</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>SPEC</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Core

Core
Espresso task dispatch

**Espresso** supports three task *types* that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<
                SPEC>
                (dijkstraTask,
                 dist + weight(v, n),
                 n->id,
                 n);
    }
}
```

---

**Tile**

<table>
<thead>
<tr>
<th>Dispatch Candidates</th>
<th>7</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPEC</td>
<td>SPEC</td>
<td>SPEC</td>
</tr>
</tbody>
</table>

---

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso task dispatch

**Espresso** supports three task *types* that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create< espresso::task < SPEC >>(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

---

**Tile**

<table>
<thead>
<tr>
<th>Dispatch Candidates</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SPEC</td>
</tr>
<tr>
<td>9</td>
<td>SPEC</td>
</tr>
<tr>
<td>10</td>
<td>SPEC</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

---

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso task dispatch

**Espresso** supports three task *types* that control speculation

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create< NONSPEC >(dijkstraTask, dist + weight(v, n), n->id, n);
    }
}
```

---

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso supports three task types that control speculation

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
  if (v->distance == UNSET) {
    v->distance = dist;
    for (Vertex* n : v->neighbors)
      espresso::create<NONSPEC>(
        dijkstraTask,
        dist + weight(v, n),
        n->id,
        n);
  }
}
```

---

**Dispatch Candidates**

<table>
<thead>
<tr>
<th>Tile</th>
<th>Dispatch Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NONSPEC</td>
</tr>
<tr>
<td>9</td>
<td>SPEC</td>
</tr>
<tr>
<td>10</td>
<td>NONSPEC</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

---

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso task dispatch

**Espresso** supports three task *types* that control speculation.

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<NONSPEC>(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

---

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
**Espresso** task dispatch

**Espresso** supports three task *types* that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbours)
            espresso::create<
                MAYSPEC>()(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

![Dispatch Candidates Table]

<table>
<thead>
<tr>
<th>Tile</th>
<th>Dispatch Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
</tr>
</tbody>
</table>

**Tile**

<table>
<thead>
<tr>
<th>Core</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MAYSPEC</td>
</tr>
<tr>
<td>9</td>
<td>SPEC</td>
</tr>
<tr>
<td>10</td>
<td>NONSPEC</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso task dispatch

Espresso supports three task types that control speculation

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<
                \_\_\_\_\_MAYSPEC\_\_\_\_\>
                (dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso supports three task types that control speculation

```c++
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<type>(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

Dispatch Candidates

<table>
<thead>
<tr>
<th>Tile</th>
<th>Dispatch Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MAYSPEC</td>
</tr>
<tr>
<td>9</td>
<td>SPEC</td>
</tr>
<tr>
<td>10</td>
<td>NONSPEC</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso supports three task types that control speculation

```cpp
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<speculative_selection_t::MAYSPEC>(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso task dispatch

Espresso supports three task types that control speculation

```c
void dijkstraTask(Timestamp dist, Vertex* v) {
    if (v->distance == UNSET) {
        v->distance = dist;
        for (Vertex* n : v->neighbors)
            espresso::create<type>(
                dijkstraTask,
                dist + weight(v, n),
                n->id,
                n);
    }
}
```

MAYSPEC lets the system decide whether to speculate
Espresso improves efficiency and programmability

MAYSPEC allows programmers to exploit the best of speculative and non-speculative parallelism

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Espresso improves efficiency and programmability

MAYSPEC allows programmers to exploit the best of speculative and non-speculative parallelism
Espresso improves efficiency and programmability

MAYSPEC allows programmers to exploit the best of speculative and non-speculative parallelism

MAYSPEC: 198x
Swarm: 162x \{ 22\% \} 6.9x
NONSPEC: 29x gmean
Microarchitectural details

Interactions between speculative and non-speculative tasks:
- How are conflicts detected and resolved?
- How do timestamps-as-barriers affect the ordered commit protocol?

**Espresso** exception model

Additional results analysis
Capsules

ENABLING SOFTWARE-MANAGED SPECULATION WITH ORDERED PARALLELISM
Some actions should bypass HW speculation

Discrete event simulation (DES) needs speculation to scale

DES also allocates memory within tasks
Some actions should bypass HW speculation

Discrete event simulation (DES) needs speculation to scale

DES also allocates memory within tasks
Some actions should bypass HW speculation

Discrete event simulation (DES) needs speculation to scale

DES also allocates memory within tasks
Some actions should bypass HW speculation

Discrete event simulation (DES) needs speculation to scale

DES also allocates memory within tasks
Some actions should bypass HW speculation

Discrete event simulation (DES) needs speculation to scale

DES also allocates memory within tasks
Some actions should bypass HW speculation

Dependences on allocator metadata cause aborts among otherwise independent tasks
Some actions should bypass HW speculation

Dependences on allocator metadata cause aborts among otherwise independent tasks

Memory

Free list

Core

Read & Write

Core

A

B

C

D

A

D

1

128

256

Speedup

1c

128c

256c

DES

Ideal allocator

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Some actions should bypass HW speculation

Dependences on allocator metadata cause aborts among otherwise independent tasks
Some actions should bypass HW speculation

Dependencies on allocator metadata cause aborts among otherwise independent tasks

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Disable hardware speculation [Moravan, ASPLOS’06]?

Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges.

Speculative tasks can access data written by earlier, uncommitted tasks. Critical for ordered parallelism.

![Chart showing speedup with and without data forwarding for DES](chart.png)

- With forwarding: Speedup increases significantly.
- No forwarding: Speedup is low and stable.

For DES, the speedup increases by 5x with forwarding compared to no forwarding.
Disable hardware speculation? Speculative data forwarding creates challenges.

Speculative tasks can access data written by earlier, uncommitted tasks.

Speculative tasks can access data written by earlier, uncommitted tasks.

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

![Graph showing speedup with and without data forwarding. The graph demonstrates a 5x speedup with forwarding compared to no forwarding.]
Disable hardware speculation [Moravan, ASPLOS’06]?

Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

**Graph:**
- Speedup
- DES

**Diagram:**
- Memory
- Core A
- Core D
- No forwarding vs. With forwarding

**Table:**
- No forwarding
- With forwarding
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

Speculative tasks can access data written by earlier, uncommitted tasks.

**Diagram:**
- A diagram illustrating the impact of speculative data forwarding on memory and free list management.
- The graph shows the speedup with and without forwarding, highlighting a 5x improvement.

**Legend:**
- DES
- Memory
- Core
- Free list
- No forwarding
- With forwarding

**Graph:**
- Y-axis: Speedup
- X-axis: Core count (1c, 128c, 256c)

**Notes:**
- Critical for ordered parallelism
- Can cause tasks to lose integrity

**Impact:**
- Speedup increases significantly with forwarding compared to no forwarding.
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges.

Speculative tasks can access data written by earlier, uncommitted tasks.

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

![Diagram showing speedup with and without forwarding]

![Diagram illustrating memory access and integrity issues]

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

![Graph showing speedup with and without data forwarding](image)

Memory

Core A

Core D

Free list

A

B

C

D

DES

With forwarding

No forwarding

5x
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks.

Critical for ordered parallelism

Can cause tasks to lose integrity 😞

With forwarding

No forwarding

5x

Speedup

1c 128c 256c

DES

Memory

Free list

Core

Core 😈
Disable hardware speculation [Moravan, ASPLOS’06]?
Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism
Can cause tasks to lose integrity 😞

![Graph showing speedup with and without forwarding]

- With forwarding: 5x speedup
- No forwarding: lower speedup

![Diagram showing memory and core connections]

- Memory: A, B, C, D
- Core: A, Core with a devil emoji
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges.

Speculative tasks can access data written by earlier, uncommitted tasks.

Critical for ordered parallelism

Can cause tasks to lose integrity 😞

![Graph showing speedup with and without data forwarding.](attachment:image.png)

![Diagram showing memory access with and without a free list.](attachment:diagram.png)
Disable hardware speculation [Moravan, ASPLOS’06]?

Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

![Graph showing speedup with and without data forwarding](image)

![Diagram illustrating speculative tasks accessing earlier data](image)
Disable hardware speculation [Moravan, ASPLOS’06]? Speculative data forwarding creates challenges

Speculative tasks can access data written by earlier, uncommitted tasks

Critical for ordered parallelism

Can cause tasks to lose integrity 😈

Simply disabling hardware speculation is unsafe with speculative forwarding

HARMONIZING SPECULATIVE AND NON-SPECULATIVE EXECUTION IN ARCHITECTURES FOR ORDERED PARALLELISM
Capsules ensure safety through OS-like protections

Untracked memory: protected from tasks that lose integrity
Capsules ensure safety through OS-like protections

Untracked memory: protected from tasks that lose integrity
Capsules ensure safety through OS-like protections

Untracked memory: protected from tasks that lose integrity
Capsules ensure safety through OS-like protections

Untracked memory: protected from tasks that lose integrity
Capsules ensure safety through OS-like protections

Untracked memory: protected from tasks that lose integrity

Tracked memory

A

B

C

D

Untracked memory

Free list

Unchecked

Core

A

D

Unversioned, no conflict checks

Only accessible by
- non-speculative tasks
- speculative capsules
Capsules ensure safety through OS-like protections

**Untracked memory**: protected from tasks that lose integrity

- **Unversioned, no conflict checks**
- Only accessible by
  - non-speculative tasks
  - speculative capsules

Tracked memory

![Diagram of tracked memory with nodes A, B, C, D and core A connected with lines indicating connections and movements.](image)

Untracked memory

![Diagram of untracked memory with free list and node A connected to core with arrows indicating movement.](image)
Capsules ensure safety through OS-like protections

**Untracked memory**: protected from tasks that lose integrity

- **Unversioned, no conflict checks**
  - Only accessible by:
    - non-speculative tasks
    - speculative capsules
Capsules ensure safety through OS-like protections

**Untracked memory**: protected from tasks that lose integrity

Unversioned, no conflict checks
Only accessible by
- non-speculative tasks
- speculative capsules
Capsules ensure safety through OS-like protections

**Untracked memory**: protected from tasks that lose integrity

**Vectored call interface**: guarantees control-flow integrity in a capsule

- **Tracked memory**
  - A
  - B
  - C
  - D

- **Untracked memory**
  - Free list
  - Only accessible by
    - non-speculative tasks
    - speculative capsules

- **Unversioned, no conflict checks**
  - Only accessible by
    - non-speculative tasks
    - speculative capsules
Capsules ensure safety through OS-like protections

**Untracked memory:** protected from tasks that lose integrity

**Vectored call interface:** guarantees control-flow integrity in a capsule

Unversioned, no conflict checks
Only accessible by
- non-speculative tasks
- speculative capsules

Holds the capsule call vector
Capsules enable important system services

Capsule-based allocator

malloc, etc. are capsule functions
metadata resides in untracked memory
Only gmean 30% slower than ideal
Capsules enable important system services

Capsule-based allocator

`malloc`, etc. are capsule functions

metadata resides in untracked memory

Only `gmean` 30% slower than ideal

**capalloc** retains the scalability of an ideal allocator
Conclusion

Speculative systems should support non-speculative execution to improve efficiency, ease programmability, and enable new capabilities.

**Espresso**: an execution model for speculative and non-speculative tasks
- Provides shared synchronization mechanisms to all tasks
- Lets the system adaptively run tasks speculatively or non-speculatively

**Capsules**: speculative tasks safely invoke software-managed speculation
- Enable important speculation-friendly services like scalable memory allocation