MELD: Merging Execution- and Language-level Determinism

Joe Devietti, Dan Grossman, Luis Ceze
University of Washington
CoreDet results

<table>
<thead>
<tr>
<th>Overhead normalized to nondet</th>
<th>barnes</th>
<th>blacksch</th>
<th>lu</th>
<th>radix</th>
<th>streamcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>800%</td>
<td>2 4 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600%</td>
<td>2 4 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400%</td>
<td>2 4 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200%</td>
<td>2 4 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0%</td>
<td>2 4 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bergan et al., ASPLOS ’10; Devietti et al. ASPLOS ’11
CoreDet results

overhead normalized to nondet

Bergan et al., ASPLOS ’10;
Devietti et al. ASPLOS ’11
determinism recipe

1. isolate threads’ updates
2. merge updates
   i. in a deterministic way
   ii. at deterministic times
determinism recipe

- **isolate** threads’ updates
- use store buffers to buffer updates

- **merge** updates
  - i. in a deterministic way
  - ii. at deterministic times
determinism recipe

- **isolate** threads’ updates
- **merge** updates
  - i. in a deterministic way
  - ii. at deterministic times
- use store buffers to buffer updates
- parallel merge algorithm
determinism recipe

**Isolate** threads' updates

- use store buffers to buffer updates
- parallel merge algorithm

**Merge** updates

i. in a deterministic way

- count fixed # of instructions

ii. at deterministic times
determinism via store buffers
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS '09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS ’09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS ’09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS '09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS '09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS ’09]
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS ’09]
commit mode: deterministically publish buffers
determinism via store buffers

parallel mode: buffer all stores, sync via [Olszewski et al, ASPLOS ’09]
commit mode: deterministically publish buffers
sources of overhead

parallel

commit

T_1

T_2

T_3

time →
sources of overhead

store buffer instrumentation
sources of overhead

parallel

commit

store buffer instrumentation

imbalance

T_1 

T_2 

T_3 

time →
sources of overhead

parallel commit

store buffer instrumentation

imbalance
# MELD

Merging Execution- and Language-level Determinism

<table>
<thead>
<tr>
<th>language-level</th>
<th>execution-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td>DMP, Kendo</td>
</tr>
<tr>
<td>Jade, DPJ</td>
<td></td>
</tr>
</tbody>
</table>
MELD
Merging Execution- and Language-level Determinism

<table>
<thead>
<tr>
<th>language-level</th>
<th>execution-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td>examples</td>
</tr>
<tr>
<td>Jade, DPJ</td>
<td>DMP, Kendo</td>
</tr>
<tr>
<td>runtime overhead?</td>
<td>none</td>
</tr>
</tbody>
</table>
## MELD

Merging Execution- and Language-level Determinism

<table>
<thead>
<tr>
<th></th>
<th>language-level</th>
<th>execution-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td>Jade, DPJ</td>
<td>DMP, Kendo</td>
</tr>
<tr>
<td>runtime overhead?</td>
<td>none</td>
<td>moderate-high</td>
</tr>
<tr>
<td>supports all code?</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
# MELD

Merging Execution- and Language-level Determinism

<table>
<thead>
<tr>
<th></th>
<th>language-level</th>
<th>execution-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td>Jade, DPJ</td>
<td>DMP, Kendo</td>
</tr>
<tr>
<td>runtime overhead?</td>
<td>none</td>
<td>moderate-high</td>
</tr>
<tr>
<td>supports all code?</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>sequential semantics?</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
# MELD

Merging Execution- and Language-level Determinism

<table>
<thead>
<tr>
<th>language-level</th>
<th>execution-level</th>
<th>hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jade, DPJ</td>
<td>DMP, Kendo</td>
</tr>
<tr>
<td>runtime overhead?</td>
<td>none</td>
<td>moderate-high</td>
</tr>
<tr>
<td>supports all code?</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>sequential semantics?</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
90% of execution time is spent in 10% of the code
90% of execution time is spent in 10% of the code, often data-parallel.
program composition
program composition

locks
queues
pointers
privatization

condition
variables
flags
program composition

locks  condition
variables

queues  flags

pointers

privatization

regular data
parallel computation
program composition

locks  condition  variables
queues  flags
pointers
privatization

regular data
parallel computation
program composition

locks

condition
variables

queues

flags

pointers

privatization

execution-level
determinism

regular data
parallel
computation
program composition

locks
condition
variables
flags
queues
pointers
privatization

execution-level
determinism

language-level
determinism

regular data
parallel
computation
what could possibly go wrong?

mergesort(int* array) {
    // verified by det lang
}


what could possibly go wrong?

what other threads call

`mergesort` concurrently?

`mergesort(int* array) {
  // verified by det lang
}

what aliases `array`?

can other threads concurrently access `array`?
what could possibly go wrong?

what other threads call `mergesort` concurrently?

```c
mergesort(int* array) {
    // verified by det lang
}
```

what aliases `array`?

can other threads concurrently access `array`?
compilation flow
compilation flow

lightweight type qualifier system
compilation flow

lightweight type qualifier system

store buffer instrumentation
compilation flow

- lightweight type qualifier system
- store buffer instrumentation
- verifier

(done manually)
compilation flow

lightweight type qualifier system

store buffer instrumentation

verifier

insn counting instrumentation

(done manually)
example: radix

```c
int *dest = ...; // implicitly "exdet"
```
example: radix

int *dest = ...; // implicitly “exdet”

langdet int langdet *_source = cast(source);
example: radix

int *dest = ...; // implicitly "exdet"

langdet int langdet *_source = cast(source);

BARRIER();
for (int i = ...) {
    dest[COMPLICATED] = _source[i];
}
BARRIER();
experimental setup

- 8-core 2.4GHz Intel Nehalem, 10GB RAM
- C benchmarks from SPLASH2, PARSEC
- CoreDet compiler with consistency optimizations from RC-DC [Devietti et al., ASPLOS '11]
MELD results

overhead normalized to nondet

| 800% | 600% | 400% | 200% | 0% |
|----------------|
| 2 4 8 | 2 4 8 | 2 4 8 | 2 4 8 | 2 4 8 |
| barnes | blacksch | lu | radix | streamcl |

- CoreDet
- MELD
MELD results

- overhead normalized to nondet

- Barnes:
  - CoreDet: 200%
  - MELD: 0%

- Blacksch:
  - CoreDet: 400%
  - MELD: 0%

- Lu:
  - CoreDet: 800%

- Radix:
  - CoreDet: 200%
  - MELD: 0%

- Streamcl:
  - CoreDet: 200%
  - MELD: 0%
MELD results

Overhead normalized to nondet

- barnes
- blacksch
- lu
- radix
- streamcl

CoreDet
MELD
# Characterization

<table>
<thead>
<tr>
<th>Workload</th>
<th>LOC</th>
<th>Explicit Sync Ops (Static)</th>
</tr>
</thead>
<tbody>
<tr>
<td>barnes</td>
<td>2964</td>
<td>6</td>
</tr>
<tr>
<td>blackscholes</td>
<td>420</td>
<td>0</td>
</tr>
<tr>
<td>lu</td>
<td>993</td>
<td>1</td>
</tr>
<tr>
<td>radix</td>
<td>878</td>
<td>3</td>
</tr>
<tr>
<td>streamcluster</td>
<td>2347</td>
<td>4</td>
</tr>
</tbody>
</table>
### Usability

<table>
<thead>
<tr>
<th>workload</th>
<th>annotations</th>
<th>casts</th>
</tr>
</thead>
<tbody>
<tr>
<td>barnes</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>blackscholes</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>lu</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>radix</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>streamcluster</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
future work

• build fully integrated system

• supporting nondeterminism via information flow tracking type system

• find gainful employment
Questions?