A New Java Runtime for a Parallel World

Christoph Reichenbach, Yannis Smaragdakis
University of Massachusetts, Amherst
PARALLELIZABILITY ACCORDING TO COMPLEXITY THEORY

Serial Code

... :

FOL + transitive closure

FOL + simple reduce (count, majority...)

FOL (First-Order Logic): parallel map
OUR GOAL: ULTIMATE PARALLEL LANGUAGE

• Declarative language to express
  – “everything” efficiently parallelizable
  – only programs that are efficiently parallelizable

• Seamlessly integrated in Java
**OUR GOAL: ULTIMATE PARALLEL LANGUAGE**

- Declarative language to express
  - “everything” efficiently parallelizable
  - only programs that are efficiently parallelizable
- Seamlessly integrated in Java

```java
List<Person> ps = forall Area a.
    find p: p in a.people && p.name == "Waldo";

int AB[i][j] = reduce(+) (forall k. A[i][k] * B[k][j]);
```
OUR GOAL: ULTIMATE PARALLEL LANGUAGE

- Declarative language to express
  - “everything” efficiently parallelizable
  - only programs that are efficiently parallelizable
- Seamlessly integrated in Java

```java
List<Person> ps = forall Area a.
    find p: p in a.people && p.name == "Waldo";

int AB[i][j] = reduce(+) (forall k. A[i][k] * B[k][j]);
```

- Program consists of parallel and sequential “phases”
  - over same data
FOR RANDOM-ACCESS STRUCTURES, PARALLELIZING IS EASY

ArrayList<...> people;
... = ... find p: p in people ...

```
people
```

```
Core #1
Core #0
Core #2
Core #3
```
FOR RANDOM-ACCESS STRUCTURES, PARALLELIZING IS EASY

```java
ArrayList<...> people;
... = ... find p: p in people ...
```
**CHALLENGE: PARALLELIZE PROCESSING OF ALL JAVA DATA**

Support *any* user-defined data structure

- Arbitrary object references
- “Sequential” library structures (e.g., linked lists)

![Diagram showing parallel processing and core allocation]
IDEA: FROM OO HEAP TO RELATIONAL HEAP

class E {
    String name;
    E boss;
    E partner;
}
IDEA: FROM OO HEAP TO RELATIONAL HEAP

class E {
    String name;
    E boss;
    E partner;
}

NEW JAVA RUNTIME FOR PARALLEL WORLD  C. REICHENBACH, Y. SMARAGDAKIS (UMASS)
IDEA: FROM OO HEAP TO RELATIONAL HEAP

class E {
    String name;
    E boss;
    E partner;
}

name

<table>
<thead>
<tr>
<th>e0</th>
<th>&quot;Waldo&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>&quot;Waldina&quot;</td>
</tr>
<tr>
<td>e2</td>
<td>&quot;Wilma&quot;</td>
</tr>
<tr>
<td>e3</td>
<td>&quot;Werner&quot;</td>
</tr>
</tbody>
</table>

boss

<table>
<thead>
<tr>
<th>e0</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>e3</td>
</tr>
<tr>
<td>e2</td>
<td>e1</td>
</tr>
<tr>
<td>e3</td>
<td>—</td>
</tr>
</tbody>
</table>

partner

<table>
<thead>
<tr>
<th>e0</th>
<th>e1</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>e0</td>
</tr>
<tr>
<td>e2</td>
<td>e5</td>
</tr>
<tr>
<td>e3</td>
<td>e17</td>
</tr>
</tbody>
</table>
**Idea: From OO Heap to Relational Heap**

```java
class E {
    String name;
    E boss;
    E partner;
}
```

Diagram showing relationships between objects:

- **name**:
  - e0: "Waldo"
  - e1: "Waldina"
  - e2: "Wilma"
  - e3: "Werner"

- **boss**:
  - e0: e3
  - e1: e3
  - e2: e1
  - e3: —

- **partner**:
  - e0: e1
  - e1: e0
  - e2: e5
  - e3: e17

NEW JAVA RUNTIME FOR PARALLEL WORLD  C. REICHENBACH, Y. SMARAGDAKIS (UMASS)
CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
**CHALLENGE: REPRESENTATION MIGRATION**

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
CHALLENGE: REPRESENTATION MIGRATION

Ideas:

• Migrate dynamically between representations at sequential-parallel switch
  – cost comparable to GC per switch

• Maintain relational heap throughout execution
**CHALLENGE: REPRESENTATION MIGRATION**

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $>10\times$ overhead for sequential execution
**CHALLENGE: REPRESENTATION MIGRATION**

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $\geq 10\times$ overhead for sequential execution

... 

- Hybrid representation (relational+cache)
Challenge: Representation Migration

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $> 10 \times$ overhead for sequential execution

... 

- Hybrid representation (relational+cache)
- Hybrid + incremental migration (write barrier)
CHALLENGE: REPRESENTATION MIGRATION

Ideas:

- Migrate dynamically between representations at sequential-parallel switch
  - cost comparable to GC per switch
- Maintain relational heap throughout execution
  - incurs $>10\times$ overhead for sequential execution

...  

- Hybrid representation (relational+cache)
- Hybrid + incremental migration (write barrier)

Ask us for our preliminary numbers!
CONCLUSION

- The relational heap is a great basis for parallelism
- Relational and traditional heaps can coexist in many ways