The Multicore Future

- “The power wall + the memory wall + the ILP wall = a brick wall for serial performance.”
  David Patterson
- “If you build it, they will come.”
  - 10, 100, 1000 cores
- There will be spare cycles.
- What do we do with them?
Redundant Computation

- Cheap computation changes the economics of exploiting parallelism.
  - Swap expensive communication with recomputation.
  - Parallelize short “nuggets” of code, such as invariants

Sequential Execution
Concurrent Execution

Communication cost = synchronization + sending

Communication cost
Traditional Parallelism

input available

result required

Narrow Window

Traditional techniques fail to parallelize code when overlap < 2 * comm. cost
Mithridates

overlap < 1 * comm. cost

input available

result required

Eliminate input
communication

result required

What about result communication?

- *Run ahead* to reduce the synchronization cost of result communication
  - Specialize via slicing
  - Schedule result calculation across n threads
- Small results
  - invariants → one bit
Approach

Transform a checked program into

- A worker
  - Core application logic, shorn of invariant checks
- Scouts
  - Minimum code necessary to check invariants assigned to them

Then execute in parallel

Architecture
Coordination

int a[10];
...
for(int i; i < 10; i++) {
    t = f(i);
    assert (t < 10);
    assert (t >= 0);
    sum += a[t];
}
...

Original

Worker

Scout

Scout Transformation

• Assign invariants to each scout
• Remove code not related to assigned invariants
  – Program slicing
• Scouts do less work, so they can run ahead
• Short-sighted oracles
Control Flow Graph

Environment

- Any data not computed by the program
  - I/O, embedded programs, entropy

... d = prompt user;
... d = prompt user;
... d = q.dequeue();
... sem.down();
... sem.up();

Original
Worker
Scout
Invariant Scheduling

```java
int a[10];
...
for(int i; i < 10; i++) {
    t = f(i);
    a: assert (t < 10 && t >= 0);
    sum += a[t];
}
...
```

Linked List

```java
class Employee {
    int ssn;
    String name;
    int salary;
    int manager;
    String department;
    String location;
    Employee next;
};

(a) Employee in $P$ and $P_{te}$

(b) Employee in $P_s$
```
Linked List Results

| $|S|$ | Time | Peak RSS |
|----|------|----------|
| 1  | 0.6157 s | 2.9 MB   |
| 2  | 0.386 s   | 2.9 MB   |
| 3  | 0.2935 s  | 3.1 MB   |
| 4  | 0.27 s    | 2.9 MB   |
| 5  | 0.285 s   | 3.0 MB   |
| 6  | 0.2715 s  | 2.9 MB   |
| 7  | 0.279 s   | 3.0 MB   |
| 8  | 0.2905 s  | 3.0 MB   |
| 9  | 0.3035 s  | 2.9 MB   |
| 10 | 0.3415 s  | 3.0 MB   |

(a) Baseline, linear invariant.

| $|S|$ | Time | Peak RSS |
|----|------|----------|
| 1  | 614.7 s | 12.7 MB |
| 2  | 308.6 s  | 13.1 MB |
| 3  | 206.4 s  | 13.3 MB |
| 4  | 155.3 s  | 13.6 MB |
| 5  | 124.2 s  | 14.4 MB |
| 6  | 103.8 s  | 14.0 MB |
| 7  | 88.91 s  | 14.4 MB |
| 8  | 92.90 s  | 14.5 MB |
| 9  | 88.62 s  | 14.6 MB |
| 10 | 86.13 s  | 15.0 MB |

(b) Baseline, quadratic invariant.

| $|S|$ | Time | Peak RSS |
|----|------|----------|
| 1  | 30.5 s  | 84 MB    |
| 2  | 124.8 s | 71 MB    |

(c) Parallelized checks using Mithridates, linear invariant.
(d) Parallelized checks using Mithridates, quadratic invariant.

Apache Lucene

| $|S|$ | Time | Peak RSS |
|----|------|----------|
| 1  | 125.9 s | 141 MB  |
| 2  | 74.7 s  | 182 MB  |
| 3  | 60.1 s  | 179 MB  |
| 4  | 52.7 s  | 189 MB  |
| 5  | 48.3 s  | 206 MB  |
| 6  | 45.9 s  | 224 MB  |
| 7  | 44.6 s  | 246 MB  |

(a) Baseline, single-threaded.

Dynamic Scheduling Only

| $|S|$ | Time | Peak RSS |
|----|------|----------|
| 1  | 118.0 s | 104 MB  |
| 2  | 72.0 s  | 110 MB  |
| 3  | 55.3 s  | 148 MB  |
| 4  | 48.2 s  | 163 MB  |
| 5  | 43.2 s  | 162 MB  |
| 6  | 39.5 s  | 165 MB  |
| 7  | 38.1 s  | 173 MB  |

(b) Parallelized checks using Mithridates.

Figure 19: Results of applying Mithridates to the Apache Lucene Indexer. Figures represent the mean of three runs.
Future Work

- Pre-compute expensive functions?
- Extend to multi-threaded code
- Automate the transformation
  - Javassist
  - Soot
  - WALA
- Share Memory

Memory Cost

- $O(n \times (|P| + e))$
  - $n =$ number of scouts + 1
  - $|P|$ is the high-water size of
    - Program
    - Stack
    - Heap
  - $e$ is
    - input queue
    - semaphores
    - code to check invariants
Memory Sharing

Worker

Questions?
Related Work

• Thread level speculation (TLS)
  – Specialized hardware
  – Rollback implies expected performance gain

• Mithridates: Language-level, source-to-source
  – Runs on commercially-available, commodity machines today
  – Predictable performance gain

Related Work

• Shadow processing
  – Main and Shadow
  – Shadow trails Main to produce debugging output

• Mithridates
  – Enforces safety properties (sound)
  – Formal transformation
  – Invariant scheduling
## Summary Static Costs

<table>
<thead>
<tr>
<th></th>
<th>Mithridates</th>
<th>TLS</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Handling</strong></td>
<td>Rewrite to synchronize environmental interactions</td>
<td>Identify guess points</td>
<td>Identify input available</td>
</tr>
<tr>
<td><strong>Result Handling</strong></td>
<td>Identify result required and rewrite to insert milestones</td>
<td>Add logic to detect and resolve conflict and identify result required</td>
<td>Identify result required</td>
</tr>
</tbody>
</table>

## Summary Runtime Costs

<table>
<thead>
<tr>
<th></th>
<th>Mithridates</th>
<th>TLS</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Handling</strong></td>
<td>Synchronized environmental interaction</td>
<td>Communication cost</td>
<td>Communication cost</td>
</tr>
<tr>
<td><strong>Result Handling</strong></td>
<td>Communication cost - mitigation (slicing &amp; invariant scheduling)</td>
<td>Communication cost + conflict resolution</td>
<td>Communication cost</td>
</tr>
</tbody>
</table>
Questions?

Issues – Handling Libraries

- $\frac{P_s}{P_w}$ is too large
- Libraries – not applications
- Few Concerns / High Cohesion
Assumptions

- Cores run at same speed
- Cores share main memory
- We do not model cache effects
- We have source code

Related Work: TLS