ALTER: Exploiting Breakable Dependences for Parallelization

Kaushik Rajan
Abhisheek Udupa
William Thies

Rigorous Software Engineering
Microsoft Research, India
Parallelization Reconsidered

Are there dependences between loop iterations?

- No → DOALL Parallelism
- Yes → Sequential program
Parallelization Reconsidered

Are there dependences between loop iterations?

- Yes: Sequential program
- No: DOALL Parallelism

Speculation

- Dependences are Imprecise
  - No Speedup

Commutativity Analysis

- Dependences can be Reordered
  - No Speedup

Break Dependences!

- Dependences can be Broken
  - Our Technique: 2.0x speedup on four cores
Parallelization Reconsidered

Are there dependences between loop iterations?

- No: DOALL Parallelism
- Yes: Sequential program

Our Technique: ALTER

- Speculation
- Commutativity Analysis
- Break dependences!

Dependences are imprecise
Dependences can be reordered
Dependences can be broken

- Agglomerative Clustering
- SG3D
- Gauss Seidel
- Floyd-Warshall
- K-Means

No Speedup
Outline

- Breakable Dependences: Stale Reads
- Deterministic Runtime System
- Assisted Parallelization
- Results

*other details in the paper*
while(!converged) {
    for i = 1 to n {
        refine(soln[i])
    }
}

Examples:
- Floyd Warshall algorithm
- Monotonic data-flow analyses
- Linear algebra solvers
- Stencil computations

**sequential**

DO WHILE

I(n)

...  

I(2)

I(1)

**ALTER: stale reads**

DO WHILE

I(n)

...  

I(2)

I(1)

merge

**privatized**

DO WHILE

I(n)

...  

I(2)

I(1)

shared memory
Stale Reads Execution Model

- Execution valid under staleReads model iff
  - Commit order is some serial order of iterations (can be different from sequential order)
  - Each iteration reads a stale but consistent snapshot
  - Staleness is bounded: no intersecting writes by intervening iterations

Akin to Snapshot Isolation for databases
Stale Reads with Reduction

\[ (W_1 \setminus W_1^R) \cap (W_2 \setminus W_2^R) = \{ \} \]

\textit{reduction} \ R := (var, O) \ where

1. Every access to \( var \) is an update using operation \( O \)
2. Operator \( O \) is commutative and associative
Deterministic Runtime System

StaleReads Commit(i):
\[ \forall j \text{ st. } j < i \text{ writes}(i) \cap \text{writes}(j) = \{\} \]
while(error < EPSILON) { //convergence loop
    error = 0.0;
    for(uint32_t i = 1; i < grid->xmax - 1; ++i) {
        for(uint32_t j = 1; j < grid->ymax - 1; ++j) {
            for(uint32_t k = 1; k < grid->zmax - 1; ++k) {
                oldValue = grid[i][j][k]
                grid[i][j][k] = a * grid[i][j][k] + b * AddDirectNbr(grid)
                                 + c * AddSquareNbr(grid) + d * AddCubeNbr(grid);
                error = max(error, (OldValue,GridPtr[i][j][k])));
            }
        }
    }
}
Test Driven Parallelism Inference

Exhaustive parallelization engine

- For each annotation run all test cases, record outcome
- Outcome of a single run $\text{success, failure} \in \{\text{crash, timeout, high contention, output mismatch}\}$
  - Output mismatch: assertion failures or floating point difference < 0.01%

Diagram:
- Sequential program
- Test suite
- Exhaustive parallelization engine
- Candidate Parallel program
- User validation
Assisted Parallelism

Prior art
Automatic parallelism

- Sequential program
- Conservative Compiler analysis
- Parallel program

Preserve program dependences

ALTER
Assisted parallelism

- Sequential program
- Test suite
- Exhaustive parallelization engine
- Candidate Parallel program
- Auto tune for perf

Preserve functionality

User validation

Assisted Parallelism

Auto tune for perf
## Benchmarks

<table>
<thead>
<tr>
<th>BENCHMARK</th>
<th>ALGORITHM TYPE</th>
<th>PARALLELISM</th>
<th>LOOP WGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggloClust</td>
<td>Branch &amp; bound</td>
<td>STALE READS</td>
<td>89%</td>
</tr>
<tr>
<td>GSdense</td>
<td>Dense algebra</td>
<td>STALE READS</td>
<td>100%</td>
</tr>
<tr>
<td>GSsparse</td>
<td>Sparse algebra</td>
<td>STALE READS</td>
<td>100%</td>
</tr>
<tr>
<td>FloydWarshall</td>
<td>Dynamic programming</td>
<td>STALE READS</td>
<td>100%</td>
</tr>
<tr>
<td>SG3D</td>
<td>Structured grids</td>
<td>STALE READS, (error, max)</td>
<td>96%</td>
</tr>
<tr>
<td>BarnesHut</td>
<td>N-body methods</td>
<td>DOALL</td>
<td>99.6%</td>
</tr>
<tr>
<td>FFT</td>
<td>Spectral methods</td>
<td>DOALL</td>
<td>100%</td>
</tr>
<tr>
<td>HMM</td>
<td>Graphical models</td>
<td>DOALL</td>
<td>100%</td>
</tr>
<tr>
<td>Genome</td>
<td>Bioinformatics</td>
<td>STALE READS</td>
<td>89%</td>
</tr>
<tr>
<td>SSCA2</td>
<td>Scientific</td>
<td>STALE READS</td>
<td>76%</td>
</tr>
<tr>
<td>K-means</td>
<td>Data mining</td>
<td>STALE READS, (delta, +)</td>
<td>89%</td>
</tr>
<tr>
<td>Labyrinth</td>
<td>Engineering</td>
<td>_</td>
<td>99%</td>
</tr>
</tbody>
</table>
Experimental Setup

- Experiments on a 2 x quad core Xeon processor
- Alter transformations in Microsoft Phoenix compiler framework
- Comparison with dependence speculation and manual parallelization of 2 applications
Results: Baseline

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Speculate</th>
<th>DOALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggloClust</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GSdense</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GSsparse</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Floyd</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SG3D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>barnesHut</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HMM</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Genome</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SSCA2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kmeans</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

No scope for dependence speculation.
Results : Alter

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>staleReads</th>
<th>OutOfOrder</th>
<th>speculate</th>
<th>DOALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggloClust</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GSdense</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>GSsparse</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Floyd</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SG3D</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>barnesHut</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HMM</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Genome</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSCA2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kmeans</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results: Manual Parallelization

- Good speedup with fine grain locking
- Comparable performance

Bar chart showing performance of different algorithms:
- AggloClust
- GSdense
- GSparses
- Floyd
- SG3D
- barnesHut
- HMM
- Genome
- SSCA2
- Kmeans

- manual
- staleReads
- OutOfOrder
- speculate
- DOALL
In the Paper...

- ALTER multi-process memory allocator
- ALTER collections
- Usage scenario’s for ALTER
- Profiling and instrumentation overhead
- DOALL parallelism and speculation within ALTER
Related Work

• Test-driven parallelization
  – QuickStep: similar testing methods for non-deterministic programs, offers accuracy bounds [Rinard 2010]

• Assisted parallelization [Taylor 2011] [Tournavitis 2009]
  – Paralax: annotations improve precision of analysis, but dependences respected [Vandierendonck 2010]

• Implicit parallelization [Burckhardt 2010]
  – Commutative annotation for reordering [August 2007, 11]
  – Optimistic execution of irregular programs [Pingali 2008]
  – As far as we know, stale reads execution model is new
Conclusions

- **Breakable dependences** must be exploited in order to parallelize certain classes of programs.

- We propose a new execution model, **StaleReads**, that violates dependences in a principled way.

- Adopt database notion of **Snapshot Isolation** for loop parallelization.

- **ALTER** is a compiler and deterministic runtime system that discovers new parallelism in programs.

- We believe tools for **assisted parallelism** can help to overcome the limits of automatic parallelization.