Making Sequential Consistency Practical in Titanium

Amir Kamil and Jimmy Su

Sequential Consistency

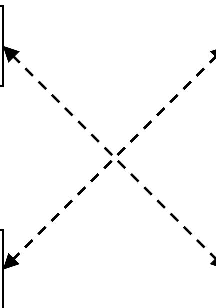
Definition: A parallel execution must behave as if it were an interleaving of the serial executions by individual threads, with each individual execution sequence preserving the program order.

Initially, `flag = data = 0`

T1

a [set `data = 1`]

x [set `flag = 1`]



T2

y [read `flag`]

b [read `data`]

Critical cycle

Legal execution: a x y b

Illegal execution: x y b a

Motivation

- Reduce the cost of sequential consistency in Titanium programs
  - Fences are inserted for memory accesses that can run concurrently to specify order
  - Inserted fences can prevent optimizations such as code motion and communication aggregation
- **In order to reduce the number of fences, precisely find all pairs of heap accesses to the same location that can run concurrently**

Titanium Features

- *Barrier*: the thread executing the barrier waits until all other threads have executed the same **textual** instance of the barrier call.
  - Example:

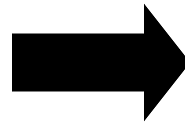
    ```
    work1(); Ti.barrier(); work2();
    ```
- A *single* value expression has the same value on all threads.
  - Example:

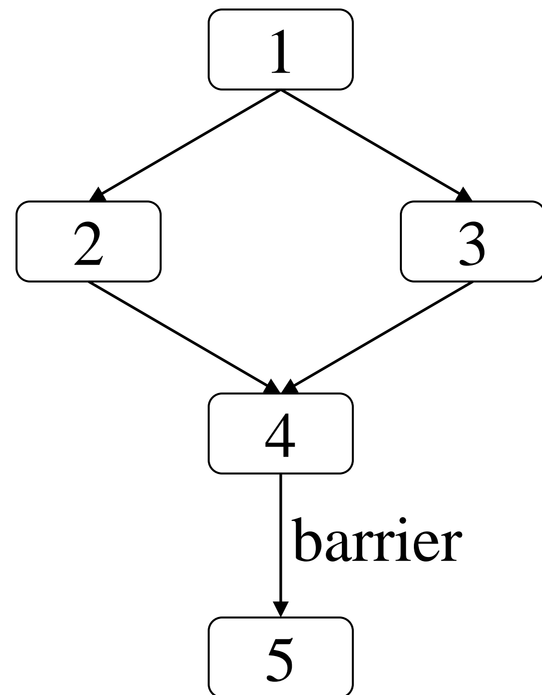
    ```
    Ti.numProcs() == 2
    ```
  - For a branch guarded by a single value expression, all threads are guaranteed to take the same branch.

Concurrency Analysis (I)

- Graph generated from program as follows:
  - Node added for each code segment between barriers and single conditionals
  - Edges added to represent control flow between segments

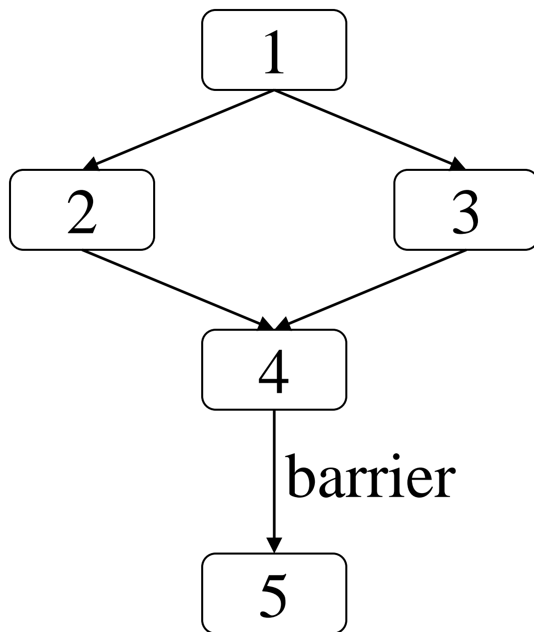
```
// code segment 1
if ([single])
  // code segment 2
else
  // code segment 3
// code segment 4
Ti.barrier()
// code segment 5
```

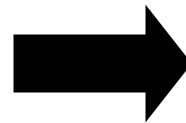




Concurrency Analysis (II)

- Two accesses can run concurrently if:
  - They are in the same node, or
  - One access's node is reachable from the other access's node without hitting a barrier
- Algorithm: remove barrier edges, do DFS





Conflicts					
	1	2	3	4	5
1	X	X	X	X	
2	X	X		X	
3	X		X	X	
4	X	X	X	X	
5					X

Thread-Aware Alias Analysis

- Two types of abstract locations: local and remote
- Remote locations created on demand when necessary
  - points-to set of remote location is remote analog of points-to set of corresponding local location
- Two locations `A` and `B` may *alias across threads* if:

  $\exists$ `x` $\in$ pointsTo(`A`). R(`x`) $\in$ pointsTo(`B`),

  (where R(`x`) is the remote counterpart of `x`)

Thread-Aware AA Example

```
class phase20 {
    public static void main(String[] args) {
L1:     phase20 a = new phase20();
        phase20 b = broadcast a from 0;
L2:     a.z = new Object();
L3:     b.z = new Object();
    }
L4: Object z = new Object();
}
```

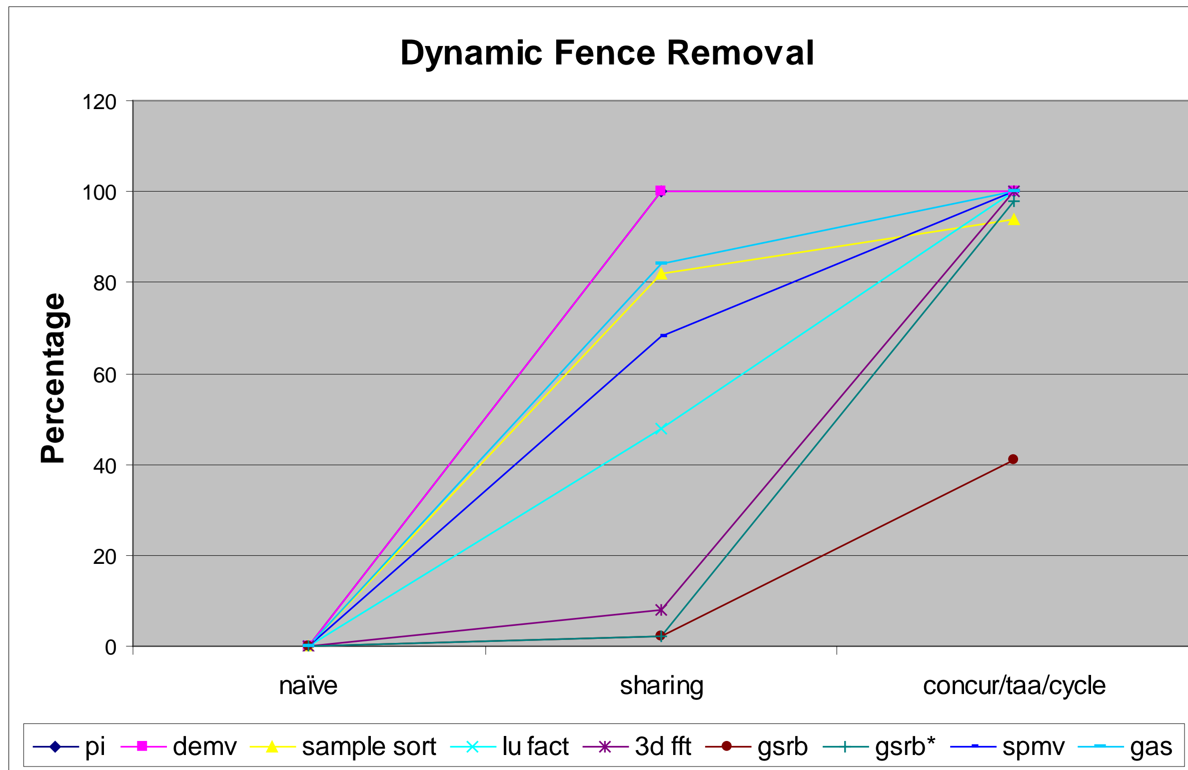
Points-to Sets
a → {1}
b → $\{1, 1_r\}$
1.z → $\{4, 2, 3, 3_r\}$
1_r.z → $\{4_r, 2_r, 3_r, 3\}$

Benchmarks

Benchmark	Lines[1]	Description
`pi`	56	Monte Carlo integration
`demv`	122	Dense matrix-vector multiply
`sample-sort`	321	Parallel sort
`lu-fact`	420	Dense linear algebra
`3d-fft`	614	Fourier transform
`gsrb`	1090	Computational fluid dynamics kernel
`gsrb*`	1099	Slightly modified version of `gsrb`
`spmv`	1493	Sparse matrix-vector multiply
`gas`	8841	Hyperbolic solver for gas dynamics

[1] Line counts do not include the reachable portion of the 37,000 line Titanium/Java 1.0 libraries

Fence Counts



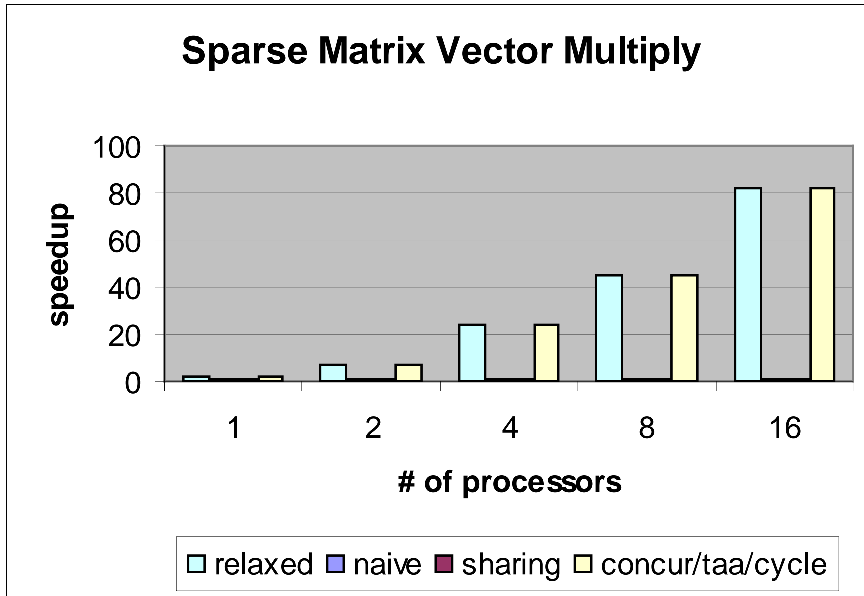
Percentages are for number of dynamic fences reduced over naive

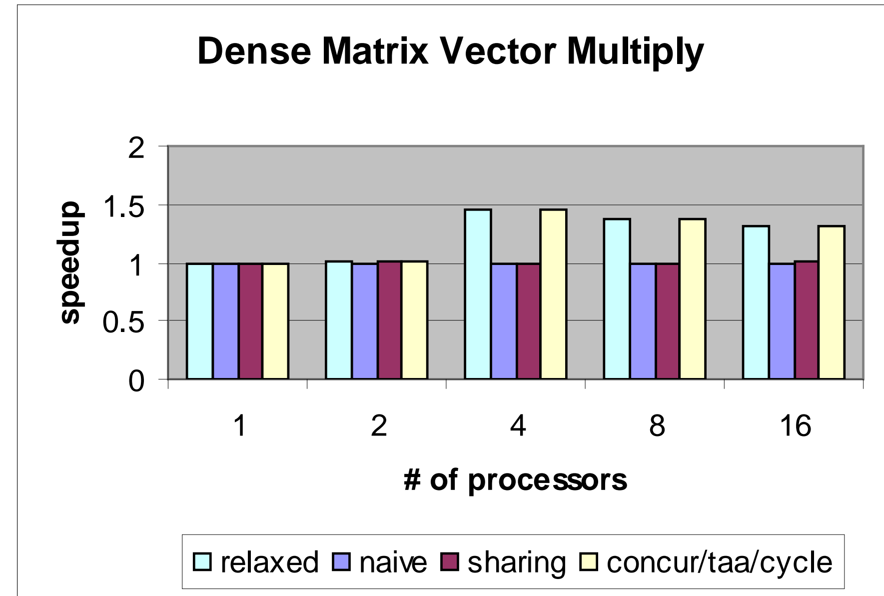
`naïve`	All heap accesses
`sharing`	All shared accesses
`concur/taa/cycle`	Concurrency analysis + thread-aware AA + cycle detection

Optimizations

- Overlap bulk memory copies
- Communication aggregation for irregular array accesses (ie `a[b[i]]`)
- Both optimizations reorder accesses, so sequential consistency can prevent them

Performance Results





Linux cluster with Itanium/Myrinet

Conclusion: sequential consistency can be provided with little or no performance cost