An Optimized Reduction Design to Minimize Atomic Operations

Ettore Speziale Andrea Di Biagio Giovanni Agosta

Politecnico di Milano

May 20, 2011

Contents

Contents

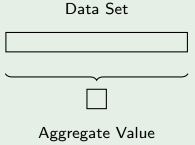
Motivating Scenario

Reduction Operations in Massively Parallel Programs

High-performance applications process huge amounts of data:

- Need to extract aggregate values

Aggregation



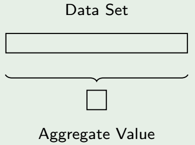
Motivating Scenario

Reduction Operations in Massively Parallel Programs

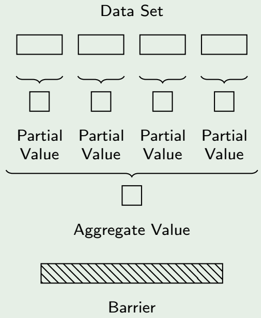
High-performance applications process huge amounts of data:

- Need to extract aggregate values

Aggregation



Parallelization



State of the Art Solutions

Partial indexes can be combined/reduced in different ways:

1. By master thread
   - Explicit bottleneck
2. By each thread using atomic Read-Modify-Write instructions
   - Stress cache coherency algorithm
3. Using fast barrier synchronization instructions
   - Need special purpose hardware
   - Need specially designed aggregation algorithm

Contents

Proposed Reduction Design

Merging Reduction and Barrier Synchronization

Key idea:

- Merge reduction and barrier synchronization
- Exploit unused bits in synchronization data to carry partial reduction values

Target barrier synchronization algorithm: tournament barrier

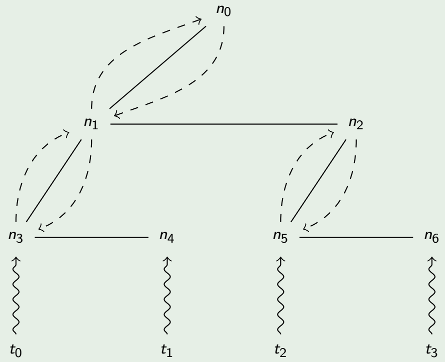
- Scalable: avoid RMW instructions by construction
- Synchronization tree can be exploited to implicitly parallelize reduction computation
- Synchronization tree nodes contain exploitable unused space: only 1 bit used

To achieve reasonable performance:

- Data aligned in memory for faster loads/stores
- Data padded to avoid false-sharing effects

Four threads want to achieve synchronization:

Tournament Barrier Example

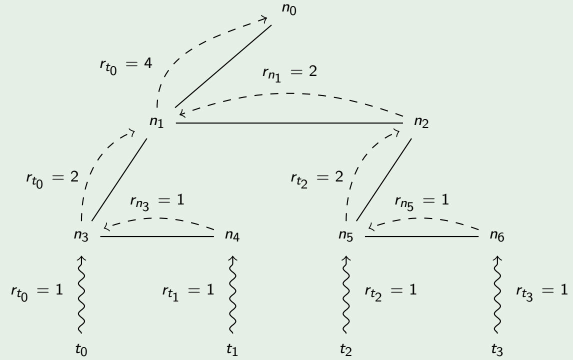


Proposed Reduction Design

Exploiting Tournament Barrier for Reductions

Partial reductions computed together with barrier execution:

Reduce and Synchronize



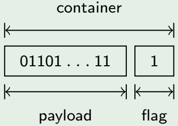
Global reduction available at n_0 node

Implementation Details
Node Layout

Each node is atomically accessible:

- A native integer
- Cache line aligned

Container Layout



Payload: reduction partial value

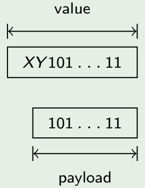
Flag: needed by barrier synchronization

Design Optimizations

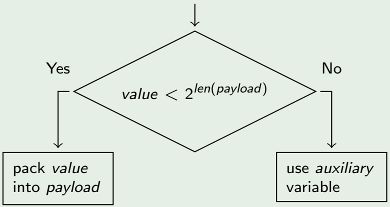
Packing More Data

What if value type size is greater than payload type size?

Fitting the Payload



Choosing the Path



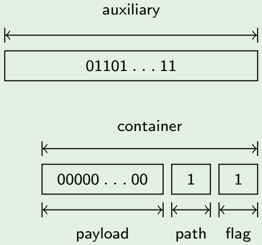
Bet on value assumed by data at run-time:

- **fast path** ignore two MSB, if they are 0
- **slow path** use a slower algorithm, otherwise

At runtime, partial reduction value does not fit payload size:

- An auxiliary variable is needed for each node
- Coherency forced via memory fences

Container Extended Layout



Consider the case of multiple aggregate values to compute:

Multiple Reductions in OpenMP

```
#pragma omp for reduce(+:a,b)
for(i = lw; i < up; ++i)
  ...
```

Design Optimizations

Nowait Reductions

Consider the case of multiple aggregate values to compute:

Multiple Reductions in OpenMP

```
#pragma omp for reduce(+:a,b)
for(i = lw; i < up; ++i)
  ...
```

Compiled

```
...
lock();
a += a_priv;
b += b_priv;
unlock();
barrier();
```

Design Optimizations

Nowait Reductions

Consider the case of multiple aggregate values to compute:

Multiple Reductions in OpenMP

```
#pragma omp for reduce(+:a,b)
for(i = lw; i < up; ++i)
  ...
```

Compiled

```
...
lock();
a += a_priv;
b += b_priv;
unlock();
barrier();
```

Optimized

```
...
nowait_barrier_reduce(&a, a_priv);
barrier_reduce(&b, b_priv);
```

Design Optimizations

Nowait Reductions

Consider the case of multiple aggregate values to compute:

Multiple Reductions in OpenMP

```
#pragma omp for reduce(+:a,b)
for(i = lw; i < up; ++i)
  ...
```

Compiled

```
...
lock();
a += a_priv;
b += b_priv;
unlock();
barrier();
```

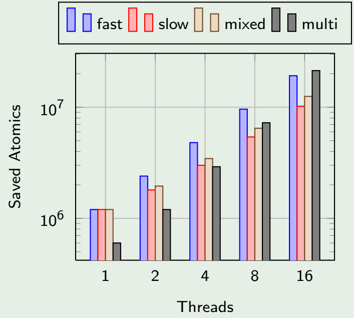
Optimized

```
...
nowait_barrier_reduce(&a, a_priv);
barrier_reduce(&b, b_priv);
```

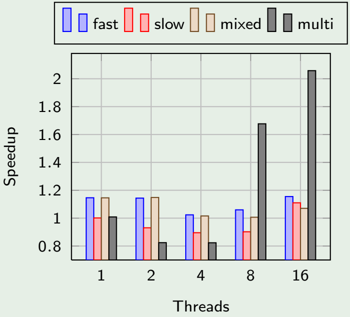
- No need for synchronization except for last aggregate

Baseline is GCC 4.6 *libgomp* [1], reductions via RMW

Saved RMW Instructions



Speedups

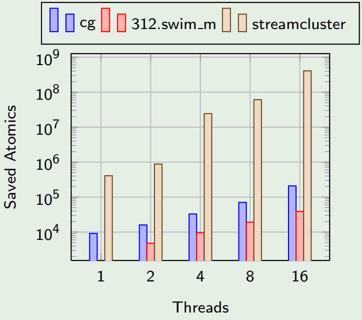


[1] Central counter barrier

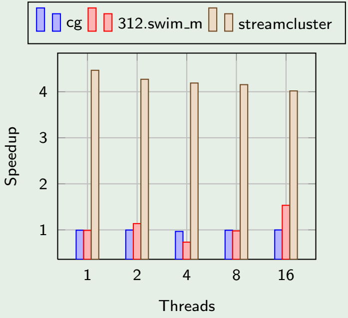
Reduction Benchmarks
From NAS, SpecOMP, Parsec

The `streamcluster` benchmark employs master reduce

Saved RMW Instructions



Speedups



Contents

Final Remarks

An optimized reduction design:

- combine reduction with barrier synchronization
- merging the two operations exposes more optimization opportunities
- optimization viable on shared memory multiprocessors with considerable number (≥ 16) of independent processors

Future directions:

- Topology aware synchronization/reduction tree
- Adaptive data-compaction method

That's All, Folks!

Questions are welcome