2022-02-17 Scau

Scan

Parallel Algorithm Design WS21/22

N. Kochendörfer, C. Alles, S. Proß

February 17, 2022

Scan Parallel Algorithm Design WS21/22

N. Kochendörfer, C. Alles, S. Proß

February 17, 2022

Table of Contents

- Scan Theory
- 2 Implementation
- Optimizations
- 4 Summary

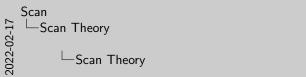
Scan

Table of Contents

Scan Thory
Implication
Optimization
Sammary

Scan Theory

- Synonyms: prefix sum, cumulative sum or scan
- inclusive and exclusive version
- further specialization: segmented scan



Scan Theory

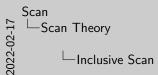
Synonyms: prefix sum, cumulative sum or scan
 inclusive and exclusive version
 further specialization: segmented scan

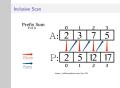
Inclusive Scan

$\underset{P \text{ of } A}{\text{Prefix}} \text{Sum}$ Store

0

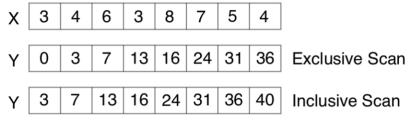
https://williamrjribeiro.com/?p=132





3

Inclusive vs. Exclusive scan



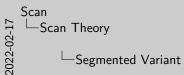
https://livebook.manning.com/book/parallel-and-high-performance-computing/chapter-5/v-11/

Scan
1-Scan Theory
Control
Con

Segmented Variant

						input
						flag bits
1	3	6	4	9	6	segmented scan $+$

https://en.wikipedia.org/wiki/Segmented_scan





Implementation

STL Algorithm

STL provides:

- std::inclusive_scan
- std::exclusive_scan

Essentially equivalent to:

```
float sum = 0;
for(size_t i =0; i<N; i++)
{
    sum += input[i];
    output[i] = sum;
}</pre>
```

 \Rightarrow Sequential to a fault!

```
Scan
Implementation
Implementation
```

2022-02-17

```
Implementation
371. Agents
371. Agents
a std includes, scan
a std studes, scan
a std studes, scan
Essentially equivant to

Float tom = 0

std state, = 1 = 0; clk; i++)

sum == inpa[[:];
sumpa[[:] = sum;
```

→ Sequential to a fault!

Implementation

Alternatives

Alternatives to STL:

- OpenMP: scan pragma
- TBB: parallel_scan function

Alternative Algorithms:

- Up-Down Sweeping Scan
- Tiled Scan

Scan
Implementation
Implementation

Implementation
Alternatives to STL:

OpenMP: scan pragma

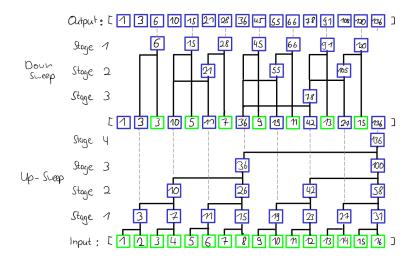
*TBE: parallel, scan function
Alternative Algorithms:

Up-Down Sweeping Scan

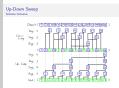
Tiled Scan

Up-Down Sweep

Schema Inclusive







Up-Down Sweep

Dependency:

- Only between stages
- ⇒ Lots of parallelism

Downsides:

- Workload of 2N
- Communication!
- Workload stage dependent!

Scan
Implementation

Description
Up-Down Sweep

Up-Down Sweep

Up-Down Sweep

Tiled Scan

Idea: Process input in independent chunks.

- Each chunk misses previous results
 - ⇒ Second pass over data.
- Workload: 2N

Our solution:

- Temporary vector for intermediate sums.
- Only one write to output.

Scan
—Implementation
—Tiled Scan

Tiled Scan

Idea: Process input in independent chunks.

• Each chunk misses previous results

• Second pass over data.

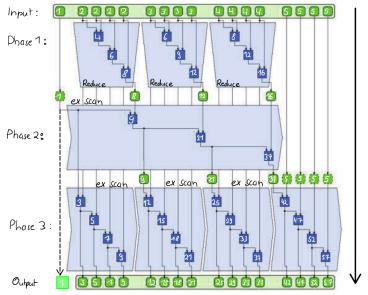
• Workload: 2N

Our solution:

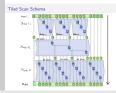
• Temporary vector for intermediate sums.

• Only one write to output.

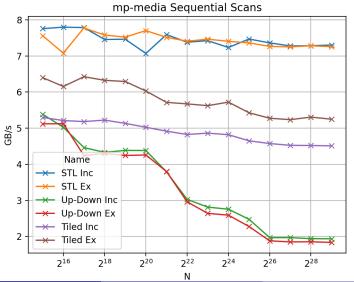
Tiled Scan Schema



Scan
Implementation
Tiled Scan Schema



Sequential Scan Results





Segmented Scan

Implementation

- Not present in STL!
- No reference implementations...

Solution: Wrapping the binary operation!

```
[binary_op](PairType left, PairType right){
    PairType new_right = right;
    if (not right.flag)
        new_right.value =
              binary_op(left.value, right.value);
    return new_right;
});
```

```
Scan
Implementation
Segmented Scan
```

2022-02-17

```
Segmented Scan
Important STU

• Not present in STU

• Not reference implementations.

Salos reference implementations.

[Disorty_map([FastType Left, FastType right)]

If (mat right flag [fastType left, refer [fast Venlar])

If (mat right flag [fast])
```

return new right:

Segmented Scan

Solution

Works for:

- STL Scans
- Most inclusive scans

Challenge: Exclusive Scan

- Exclusive Segmented is complex
- Custom solution for each variant

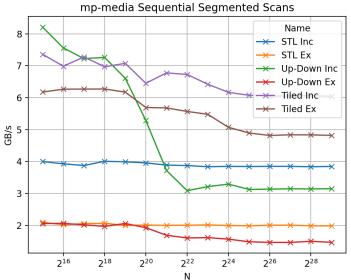
 Segmented Scan
Solution

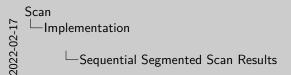
Works for:

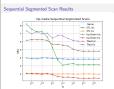
y STL Scans
w Most inclusive scans
Challenge: Evaluaive Scan
a Evaluaive Segmented in complex

· Custom solution for each variant

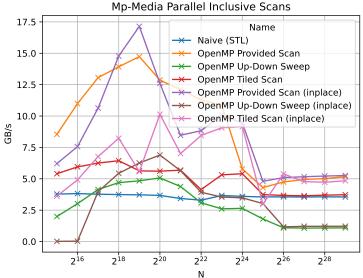
Sequential Segmented Scan Results



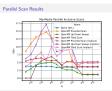




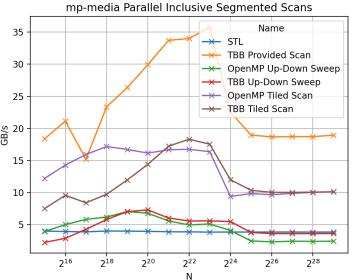
Parallel Scan Results

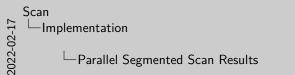


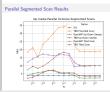
Scan
Implementation
Parallel Scan Results



Parallel Segmented Scan Results







Intermediate Results

Ranking:

- Library provided implementations
- 2 Tiled Scan
- Up-Down Sweeping Scan

Remarks:

- OpenMP >= TBB (if available)
- Up-Down Sweep is slow

Can we do better?

Scan
—Optimizations

_Intermediate Results

Intermediate Results

Ranking:

- Library provided implementations
 Tiled Scan
- Tiled Scan
 Up-Down Sweeping Scan
- OpenMP >= TBB (if available)
 Up-Down Sweep is slow

Can we do better?

Algorithmic Optimization

Initial Goal: functional correctness.

Algorithmic Optimizations:

- Loop-Fusion:
 - Up-Down Sweep
 - Exclusive Segmented Scan
- Limiting Memory Accesses
- General clean up

Performance gain \sim 1-5 GB/s!

Scan
Optimizations
Algorithmic Optimization

Algorithmic Optimization

Initial Goal: functional correctness.

Algorithmic Optimizations:

Loop-Fusion:

Up-Down Sweep

Exclusive Segmented Scan
 Limiting Memory Accesses
 General clean up

Performance gain \sim 1-5 GB/sl

Ensure that the data generated is local to the node:

```
std::vector<float> data(N);
#pragma omp parallel for schedule(static)
for (size_t i = 0; i < data.size(); i++)
data[i] = rand();
```

• The performance gain by using data local structures is likely to be small due to the warmup of Catch2

Scan **Optimizations** ☐ Data Locality

2022-02-

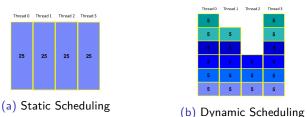
Data Locality

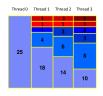
Ensure that the data generated is local to the node:

#pragma omp parallel for schedule(static for (size_t i = 0; i < data.size(); i++)

. The performance gain by using data local structures is likely to be small due to the warmup of Catch2

OpenMP Scheduling





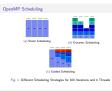
(c) Guided Scheduling

Fig. 1: Different Scheduling Strategies for 100 Iterations and 4 Threads

 ${\displaystyle \stackrel{\textstyle \subseteq}{ }}{} {\displaystyle \stackrel{\textstyle \subseteq}{ }}{} {\displaystyle \stackrel{\textstyle \cap}{ }} {\displaystyle \stackrel{\textstyle }{ }} {\displaystyle \stackrel{\textstyle \cap}{ }} {\textstyle \stackrel{\textstyle \cap}{$

2022-02-17

└─OpenMP Scheduling



Static: Round-robin

Dynamic: Constant Chunk Size

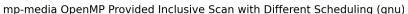
• Guided: Variable Chunk Size

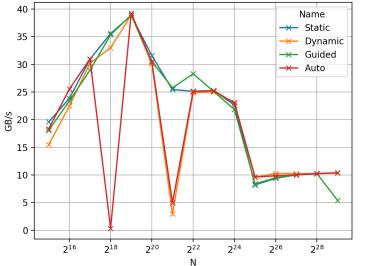
Auto: Compiler and/or Runtime System

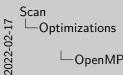
Cache Fusion if:

Export or during runtime

OpenMP Scheduling - Results MP-Media







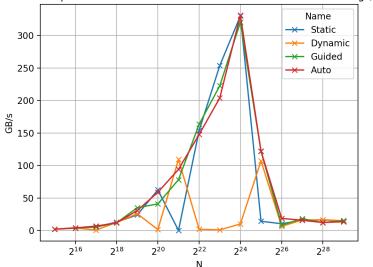
OpenMP Scheduling - Results MP-Media

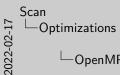


- Static and Guided perform best
- Expected due to overhead

OpenMP Scheduling - Results Ziti-Rome

ziti-rome OpenMP Provided Inclusive Scan with Different Scheduling (gnu)





OpenMP Scheduling - Results Ziti-Rome



- More severe difference
- Static least overhead
- Static Bound size and Distribution of Work
- => Static best

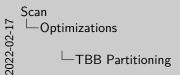
TBB Partitioning

TBB parallel constructs used:

- parallel_scan
- parallel_for

available partitioners:

- auto_partitioner
- affinity_partitioner
- simple_partitioner
- static_partitioner



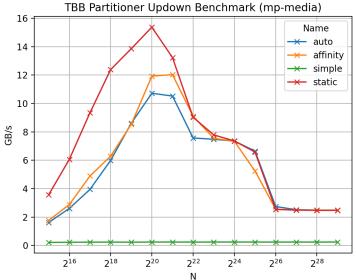
TBB Partitioning

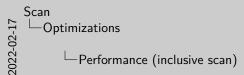
TBB parallel constructs used: • parallel_scan

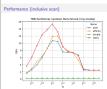
parallel_for
 available partitioners

auto_partitioner
 affinity_partitioner
 simple_partitioner
 static_partitioner

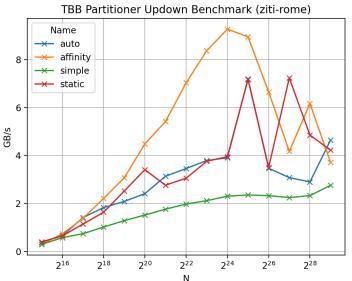
Performance (inclusive scan)



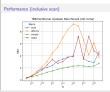




Performance (inclusive scan)







Vectorization

Requirements:

- No loop carried dependency
- Loop bounds
- No jumps in code

Realising it:

- #pragma omp simd
- Compiling with -O3
- Using Intel Icx Compiler

Scan

Optimizations

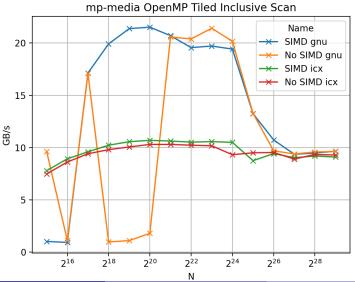
Respirements.

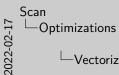
• The loop carried dependency
• Loop houses

Realize

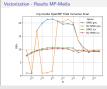
Perspirements
• * pragram one pind
• • (empire with -0)
• Using lated to Campilar

Vectorization - Results MP-Media



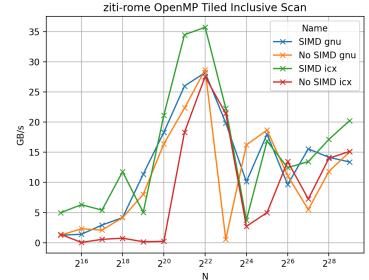


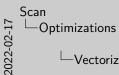
└─Vectorization - Results MP-Media



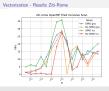
- Difference: Annotation of simd to loop
- Both compiled with -O3
- Possibly auto vectorization
- ICX much worse for MP-Media
- SIMD more stable

Vectorization - Results Ziti-Rome





└─Vectorization - Results Ziti-Rome



- Possibly auto vectorization
- ICX much better for ziti-rome
- SIMD better performance
- => SIMD

Library Provided Scans are fastest

Scan
—Summary
—Summary
—Summary

Summary

Library Provided Scans are fastest

Summary

Summary

Library Provided Scans are fastest

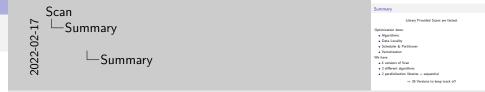
Optimization done:

- Algorithmic
- Data Locality
- Scheduler & Partitioner
- Vectorization

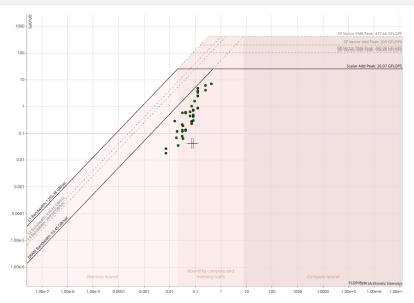
We have

- 4 versions of Scan
- 3 different algorithms
- 2 parallelization libraries + sequential

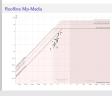
 \Rightarrow 36 Versions to keep track of!



Roofline Mp-Media



Scan
L1-20-200
Roofline Mp-Media



Roofline Ziti-Rome

