## 2022-02-17 Scan

## 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

Stan Table of Contents

Stan Table of Contents

Table of Contents

## Scan Theory

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

Scan

Scan Theory

Scan Theory

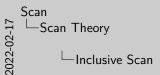
Scan Theory

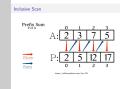
Synonyms: prefix sum, cumulative sum or scan
 inclusive and exclusive version
 further specialization: segmented scan

## Inclusive Scan

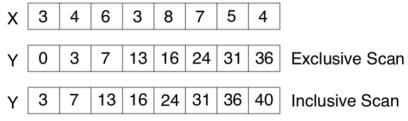
# $\underset{P \text{ of } A}{\text{Prefix}} \text{Sum}$ Store 0 3

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





## Inclusive vs. Exclusive scan



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

Scan
1-Scan Theory
Control
Con

Inclusive vs. Exclusive scan

X 3 4 6 3 8 7 5 4

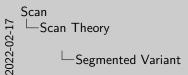
Y 0 3 7 13 16 24 31 36 Exclusive Scan

Y 3 7 13 16 24 31 36 40 Inclusive Scan

## Segmented Variant

1	<b>2</b>	3	4	5	6	input
1	0	0	1	0	1	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
15: Aquinos
55: Aquinos
55: Equinos
55: Equinos
65: Equinos
65:
```

→ 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 Absentives

Alternatives to STL:

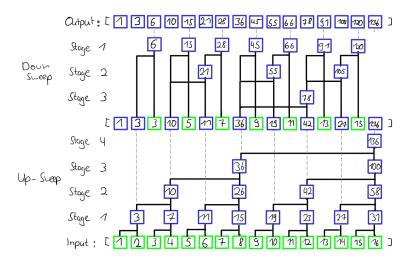
OpenMP: scan pragma

TBB: 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!



### 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.

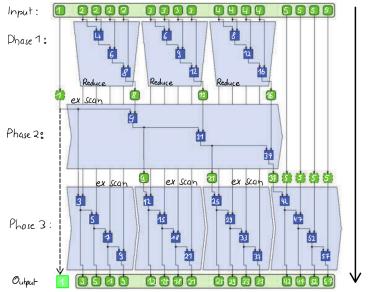
• Workload: 2N

Our solution:

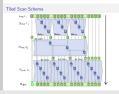
• Temporary vector for intermediate sums.

• Only one write to output.

## Tiled Scan Schema



Scan
Implementation
Tiled Scan Schema



## **Benchmarks**

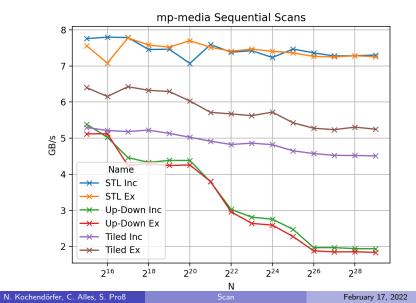
#### Parameters:

- In-Place
- Datatype: float
- Values: Linear Distribution. between 0-10.
- Benchmarking Suite: Catch2
- Number of input elements: N



- In-Place
   Datatype: float
- Values: Linear Distribution. between 0-10.
- Benchmarking Suite: Catch2
   Number of input elements: N

## Sequential Scan Results



Scan
Implementation
Sequential Scan Results

2022-02-17

14/34

## 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
Implementation

• Not prisent in STU

• Not prisent in STU

• No reference implementations...
Solution: Wapping the binny operated

[ blancay_og | Peul Type | felt, | Peul Type + right] {
    Pair Type = new_right = - right;
    If (ent - right. right);
}
```

return new right:

binary\_op(left.value, right.value);

## Segmented Scan

Solution

#### Works for:

- STL Scans
- Most inclusive scans

## Challenge: Exclusive Scan

- Exclusive Segmented is complex
- Custom solution for each variant

Scan
Implementation
Segmented Scan

2022-02-17

Segmented Scan
bilarias

Works for

V STL Scans

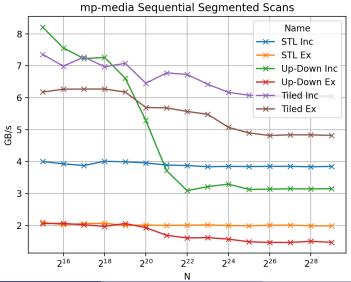
Most inclusive scans

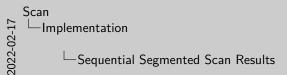
Challenge: Exclusive Scan

Exclusive Segmented is complex

Custom solution for each variation for each variation

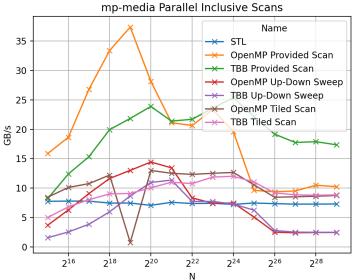
## Sequential Segmented Scan Results

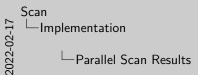


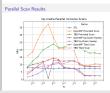




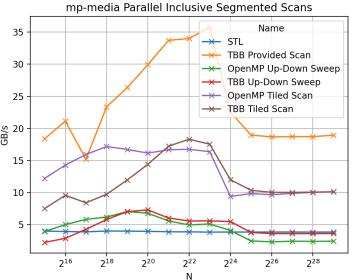
## Parallel Scan Results

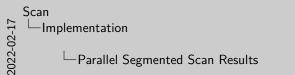


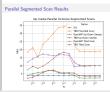




## Parallel Segmented Scan Results







### 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
- 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
1-0
Continuous
Co

Algorithmic Optimization

Initial Goal: functional correctness.

Algorithmic Optimizations:

Up-Down Sweep
 Exclusive Segmented Scan
 Limiting Memory Accesses

General clean up
 Performance gain ~ 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-

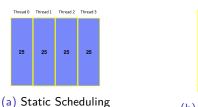
Data Locality

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





Dynamic Scheduling



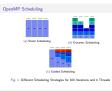
(c) Guided Scheduling

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

Scan **Optimizations** 

2022-02-17

-OpenMP Scheduling



Static: Round-robin

Dynamic: Constant Chunk Size

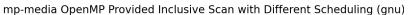
Guided: Variable Chunk Size

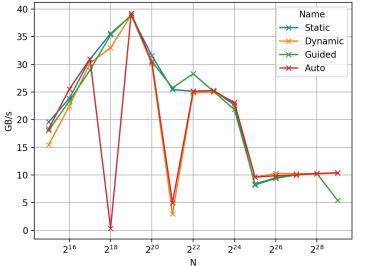
Auto: Compiler and/or Runtime System

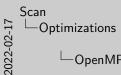
Cache Fusion if:

Export or during runtime

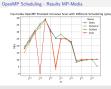
## OpenMP Scheduling - Results MP-Media







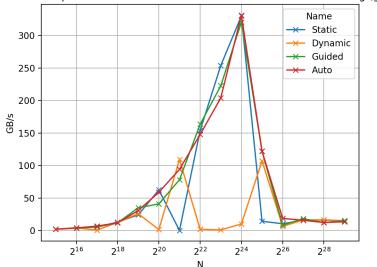
└─OpenMP Scheduling - Results MP-Media

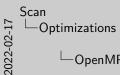


- Static and Guided perform best
- Expected due to overhead

## 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

Scan
—Optimizations
—TBB Partitioning

2022-02-17

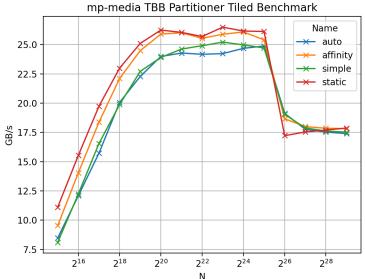
TBB Partitioning

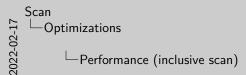
TBB parallel constructs used:

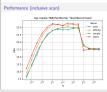
parallel\_for
 available partitioners

auto\_partitioner
 affinity\_partitioner
 simple\_partitioner
 static\_partitioner

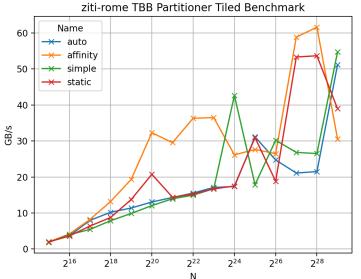
## Performance (inclusive scan)

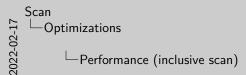


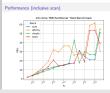




## Performance (inclusive scan)







Optimizations

### Vectorization

#### Requirements:

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

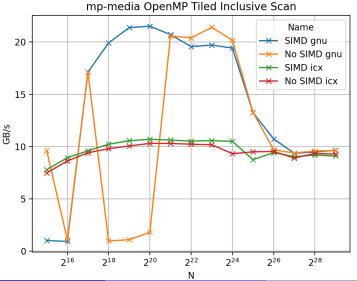
#### Realising it:

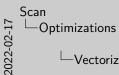
- #pragma omp simd
- Compiling with -O3
- Using Intel Icx Compiler
  - No OpenMP provided scan!

Scan
Optimizations

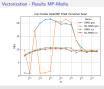
Regionments:
1 to loop carried dependency
1 loop boards
1 conference of the confer

## Vectorization - Results MP-Media



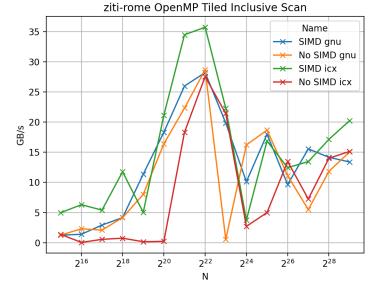


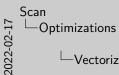
└─Vectorization - Results MP-Media



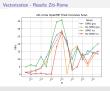
- Difference: Annotation of simd to loop
- Both compiled with -O3
- Possibly auto vectorization
- ICX much worse for MP-Media
- SIMD more stable

## 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 University Scan Scan Summary Scan Summary

Summary

Library Provided Scans are fastest

Summary

## Summary

#### Library Provided Scans are fastest

### Optimization done:

- Algorithmic
- Data Locality
- Scheduler & Partitioner
- Vectorization

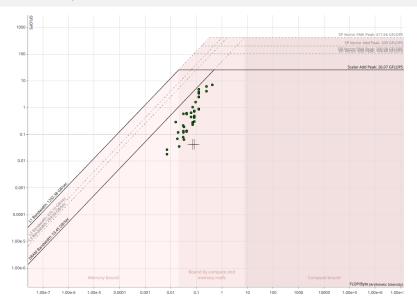
#### We have

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

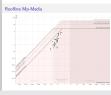
 $\Rightarrow$  36 Versions to keep track of!



## Roofline Mp-Media



Scan
L1-20-200
Roofline Mp-Media



## Roofline Ziti-Rome

