Scan

Parallel Algorithm Design WS21/22

N. Kochendörfer, C. Alles, S. Proß

February 16, 2022

2022-02-16

Scan

Scan
Parallel Algorithm Design WS21/22
N. Kochendörfer, C. Alles, S. Proß
February 16, 2022

Table of Contents

1. Scan Theory
2. Implementation
3. Optimizations
4. Summary

└─Table of Contents

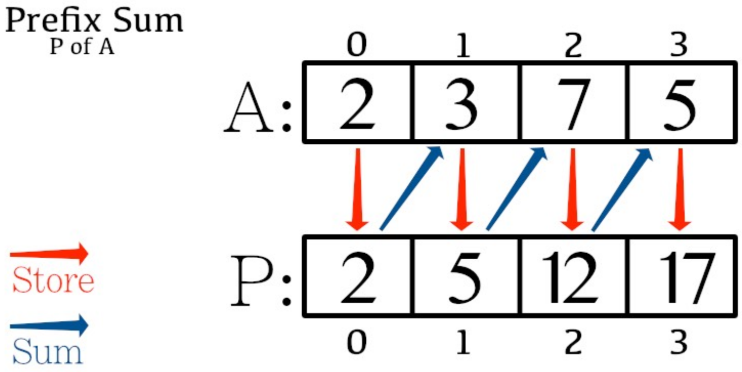
Table of Contents
1. Scan Theory
2. Implementation
3. Optimizations
4. Summary

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

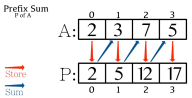


https://williamrjribeiro.com/?p=132

2022-02-16

Scan
└Scan Theory
└Inclusive Scan

Inclusive Scan


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

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

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

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

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

segmented variant

1	2	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

segmented variant

1	2	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;
}
```

$\Rightarrow$ Sequential to a fault!

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;
}
⇒ Sequential to a fault!

Implementation

Alternatives

Alternatives to STL:

- OpenMP: scan pragma
- TBB: parallel_scan function

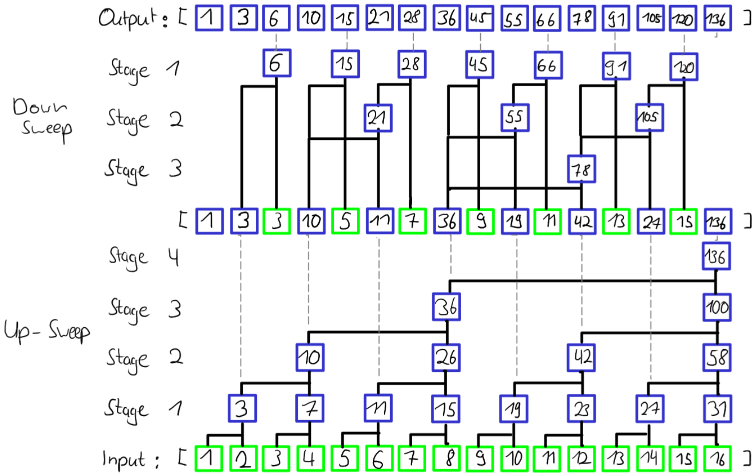
Alternative Algorithms:

- Up-Down Sweeping Scan
- Tiled Scan

Implementation
Alternatives
Alternatives to STL:
- OpenMP: scan pragma
- TBB: parallel_scan function
Alternative Algorithms:
- Up-Down Sweeping Scan
- Tiled Scan

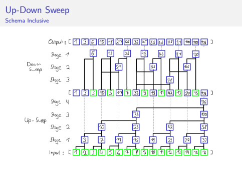
Up-Down Sweep

Schema Inclusive



2022-02-16

Scan
└Implementation
└Up-Down Sweep



Up-Down Sweep

Dependency:

- Only between stages

⇒ Lots of parallelism

Downsides:

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

Scan
└Implementation
└Up-Down Sweep
2022-02-16

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
  - $\Rightarrow$ Second pass over data.
- Workload: 2N

Our solution:

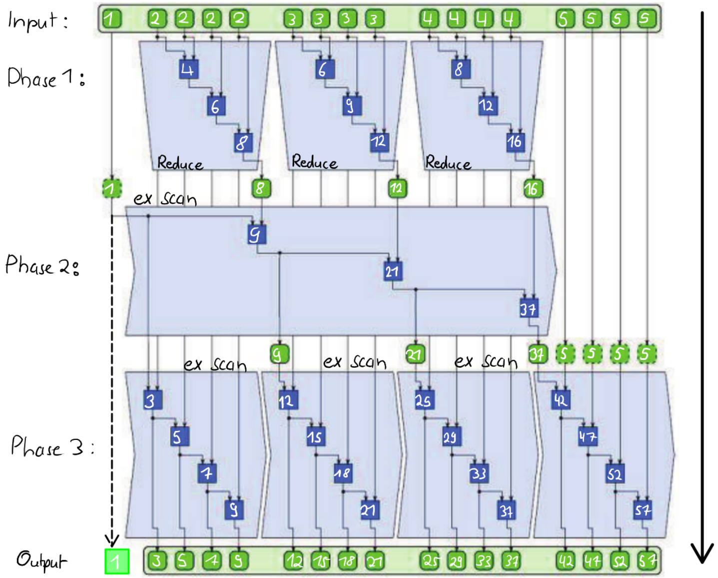
- Temporary vector for intermediate sums.
- Only one write to output.

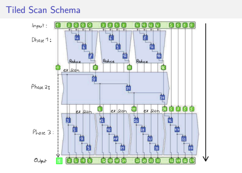
Tiled Scan

Idea: Process input in independent chunks.
- Each chunk misses previous results
  - ⇒ Second pass over data.
- Workload: 2N

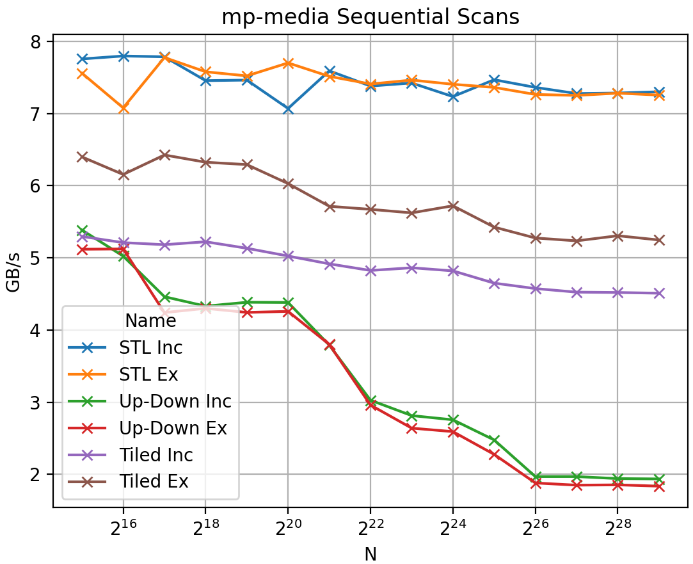
Our solution:
- Temporary vector for intermediate sums.
- Only one write to output.

Tiled Scan Schema



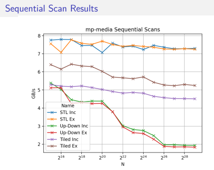


Sequential Scan Results



2022-02-16

Scan
└Implementation
└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;
});
```


2022-02-16

Scan
└Implementation
  └Segmented Scan

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;
});
```

Segmented Scan

Solution

Works for:

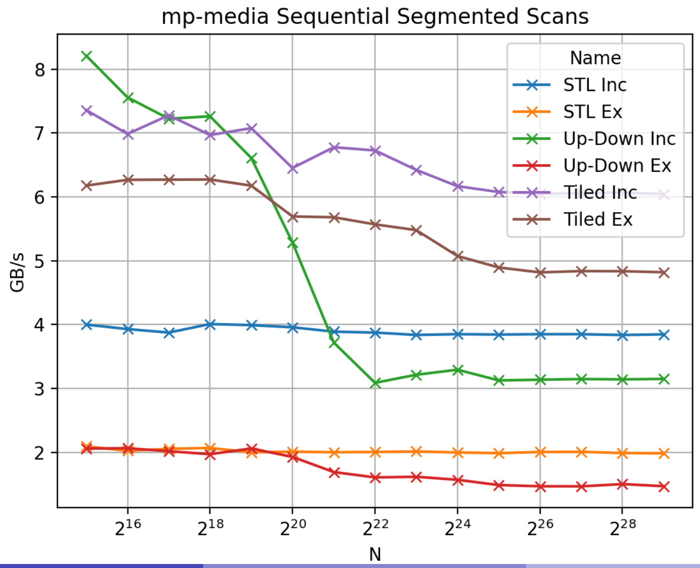
- STL Scans
- Most inclusive scans

Challenge: Exclusive Scan

- Exclusive Segmented is complex
- Custom solution for each variant

Segmented Scan
Solution
Works for:
- STL Scans
- Most inclusive scans
Challenge: Exclusive Scan
- Exclusive Segmented is complex
- Custom solution for each variant

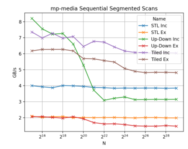
Sequential Segmented Scan Results



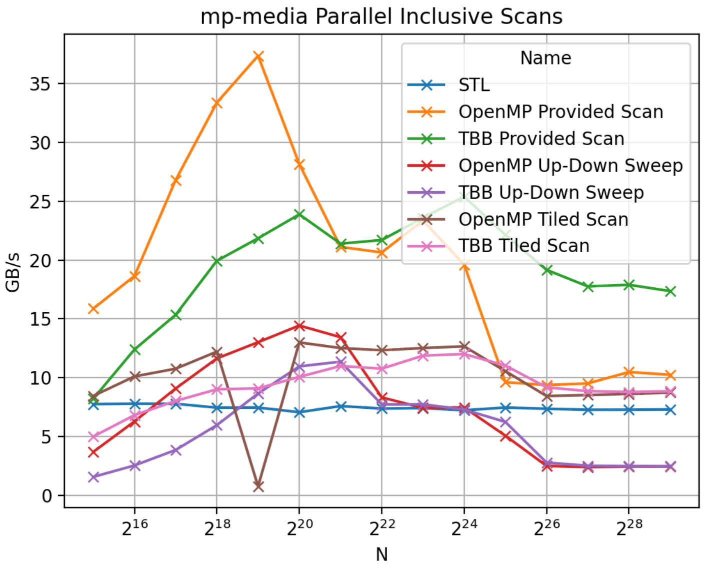
2022-02-16

Scan
└─Implementation
└─Sequential Segmented Scan Results

Sequential Segmented Scan Results

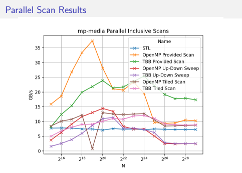


Parallel Scan Results

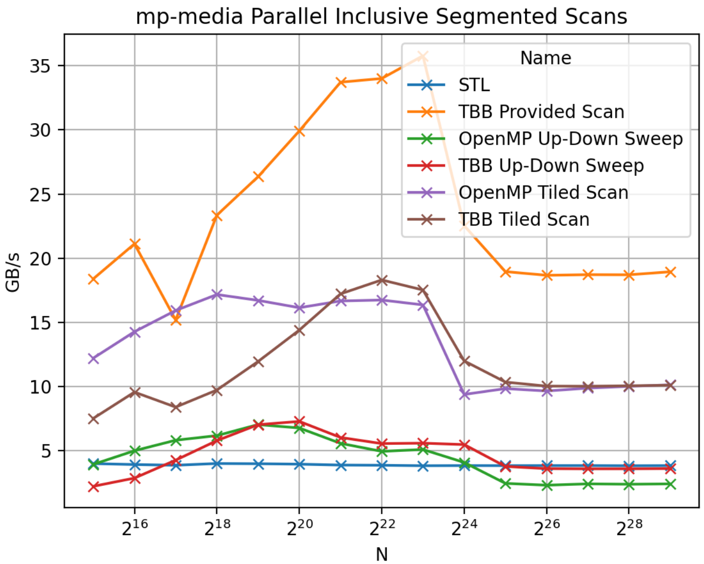


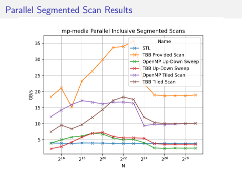
2022-02-16

Scan
└─Implementation
  └─Parallel Scan Results



Parallel Segmented Scan Results





Intermediate Results

Ranking:

1. Library provided implementations
2. Tiled Scan
3. Up-Down Sweeping Scan

Remarks:

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

Can we do better?

Intermediate Results

Ranking:
1. Library provided implementations
2. Tiled Scan
3. Up-Down Sweeping Scan

Remarks:
- 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!

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!

Data Locality

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

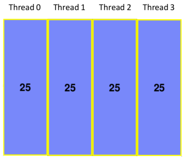
Data Locality

Ensure that the data generated is local to the node:

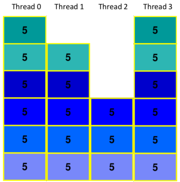
```
std::vector<float> data(N);
#pragma omp parallel for schedule(static)
for (size_t i = 0; i < data.size(); i++)
{
data[i] = rand();
}
```

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

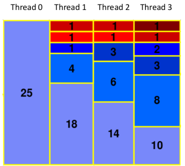
OpenMP Scheduling



(a) Static Scheduling

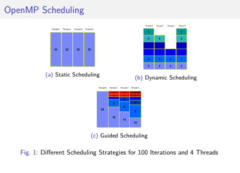


(b) Dynamic Scheduling



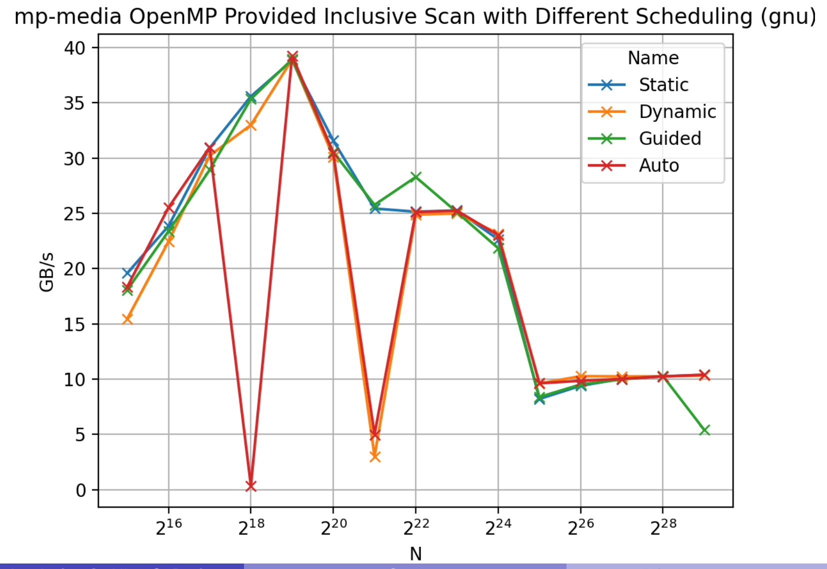
(c) Guided Scheduling

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



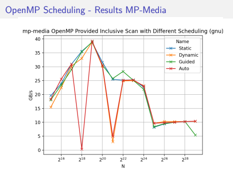
- Static: Round-robin
- Dynamic: Constant Chunk Size
- Guided: Variable Chunk Size
- Auto: Compiler and/or Runtime System
- Cache Fusion if:
  1. single parallel region executes all of the maps to be fused.
  2. The loop for each map has the same bounds and chunk size.
  3. Each loop uses the static scheduling mode, either as implied by the environment or explicitly specified.
- Export or during runtime

OpenMP Scheduling - Results MP-Media



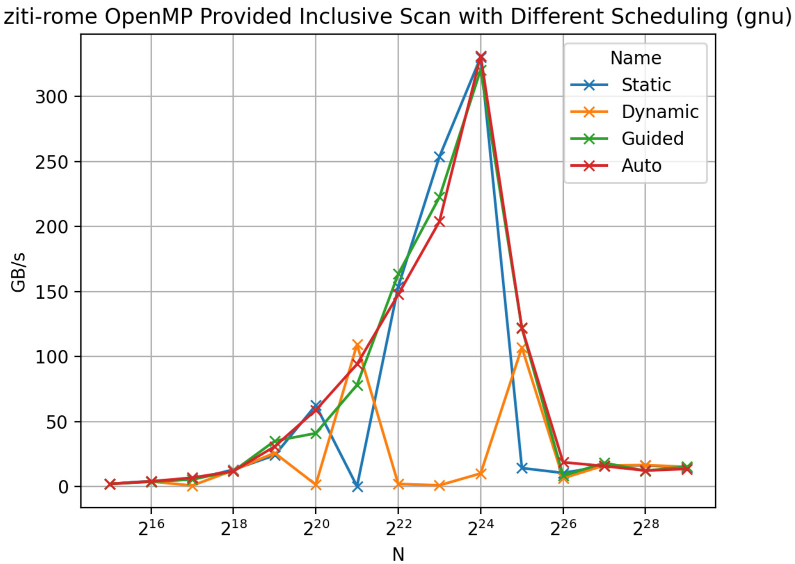
2022-02-16

Scan
└Optimizations
└OpenMP Scheduling - Results MP-Media



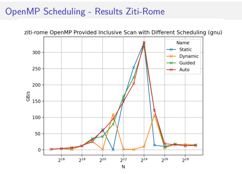
- Static and Guided perform best
- Expected due to overhead

OpenMP Scheduling - Results Ziti-Rome



2022-02-16

Scan
└Optimizations
└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:

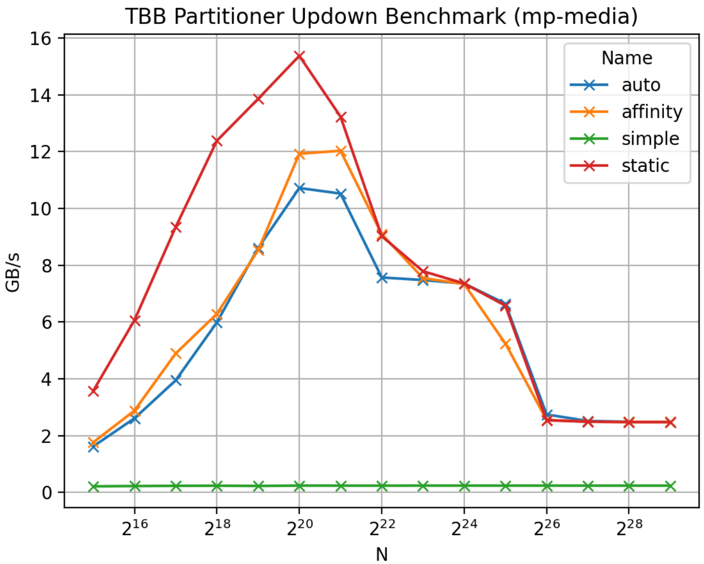
- parallel_scan
- parallel_for

available partitioners:

- auto_partitioner
- affinity_partitioner
- simple_partitioner
- static_partitioner

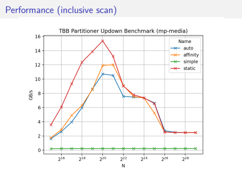
TBB Partitioning
TBB parallel constructs used:
parallel_scan
parallel_for
available partitioners:
auto_partitioner
affinity_partitioner
simple_partitioner
static_partitioner

Performance (inclusive scan)

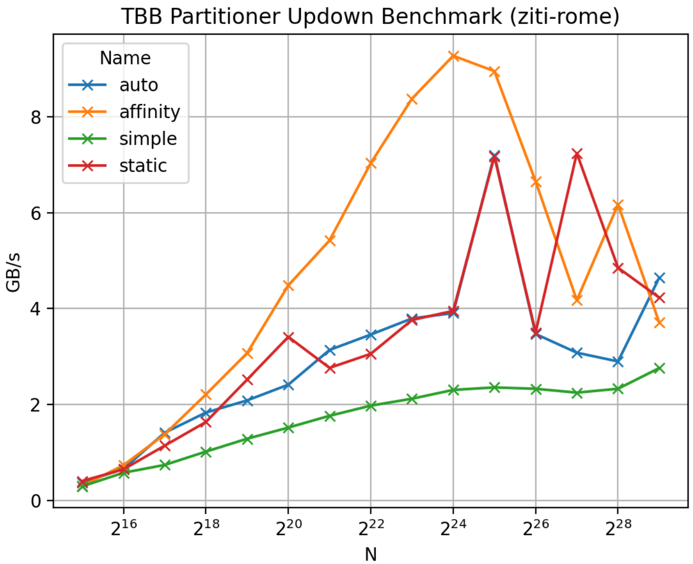


2022-02-16

Scan
└─Optimizations
└─Performance (inclusive scan)

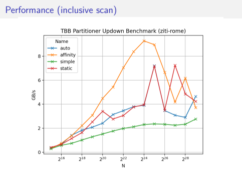


Performance (inclusive scan)



2022-02-16

Scan
└Optimizations
└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

Vectorization

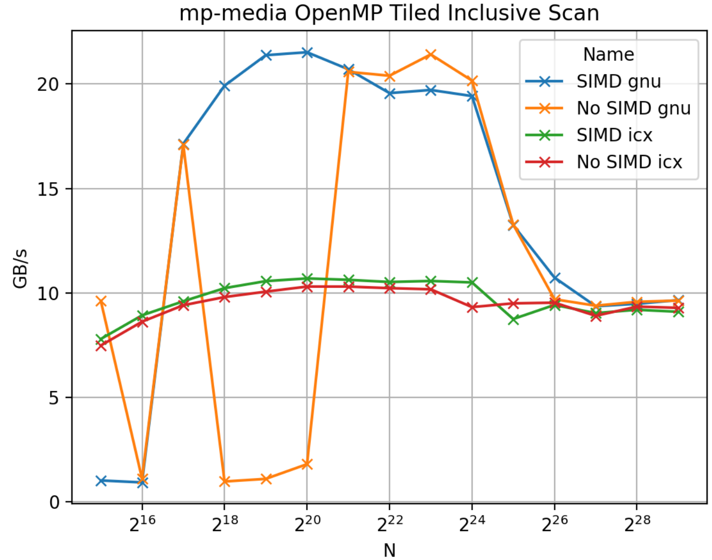
Requirements:

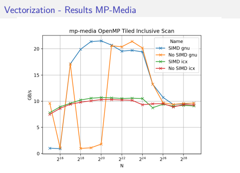
- No loop carried dependency
- Loop bounds
- No jumps in code

Realising it:

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

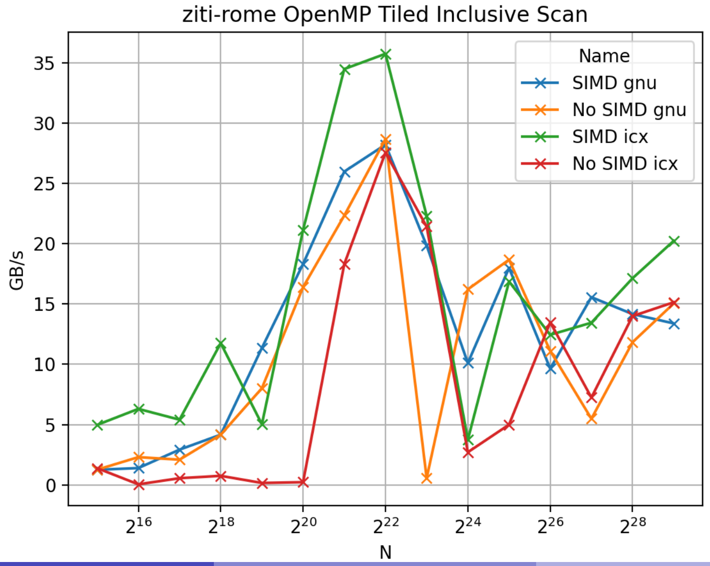
Vectorization - Results MP-Media

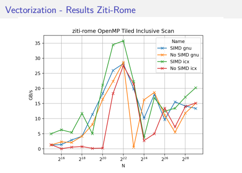




- Difference: Annotation of simd to loop
- Both compiled with -O3
- Possibly auto vectorization
- ICX much worse for MP-Media
- SIMD more stable

Vectorization - Results Ziti-Rome





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

Summary

Library Provided Scans are fastest

Scan
└Summary
└Summary
2022-02-16

Summary
Library Provided Scans are fastest

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

⇒ 36 Versions to keep track of!

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

⇒ 36 Versions to keep track of!