Visualization of Data Movements and Accesses

Til Mohr

High Performance Computing
RWTH AACHEN UNIVERSITY

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

Matrix:

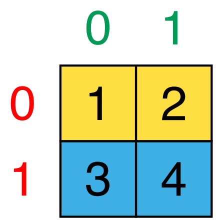
$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 2 \\ 1 & 3 & 4 \end{array}$$

Sum of Matrix Elements

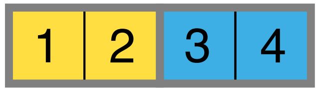
Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

Matrix:



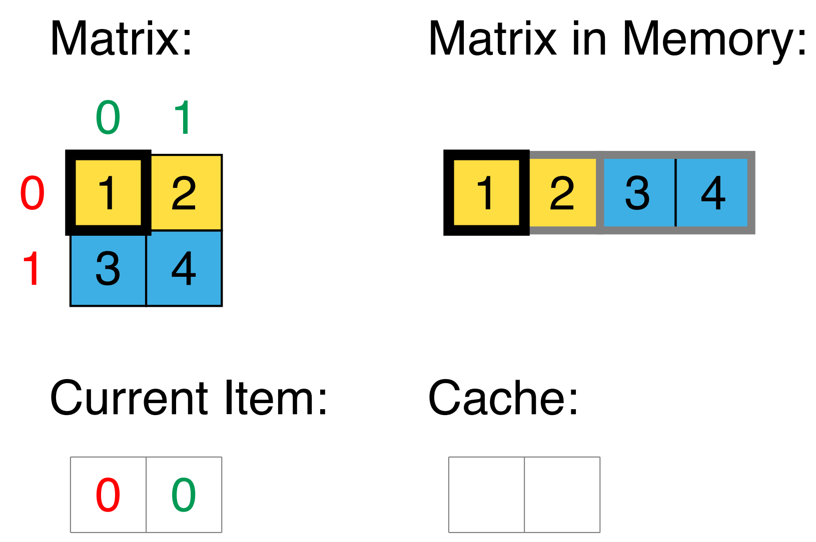
Matrix in Memory:



Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

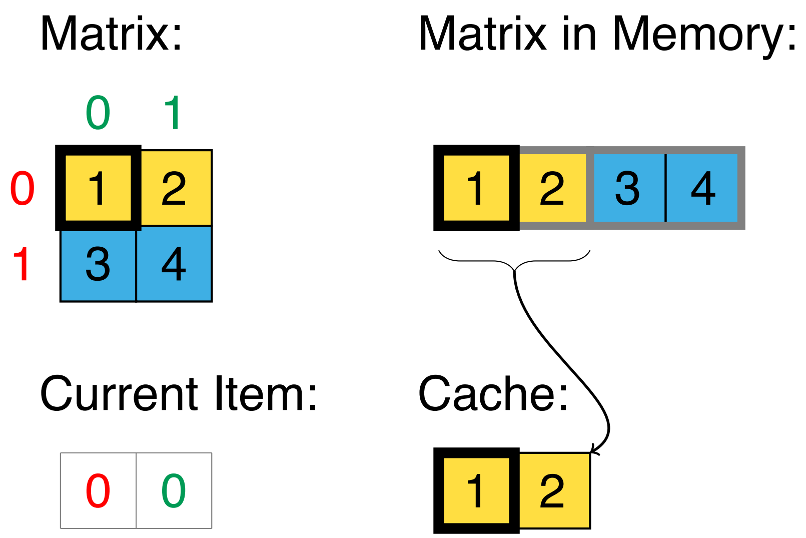


Number of cache misses: **0**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

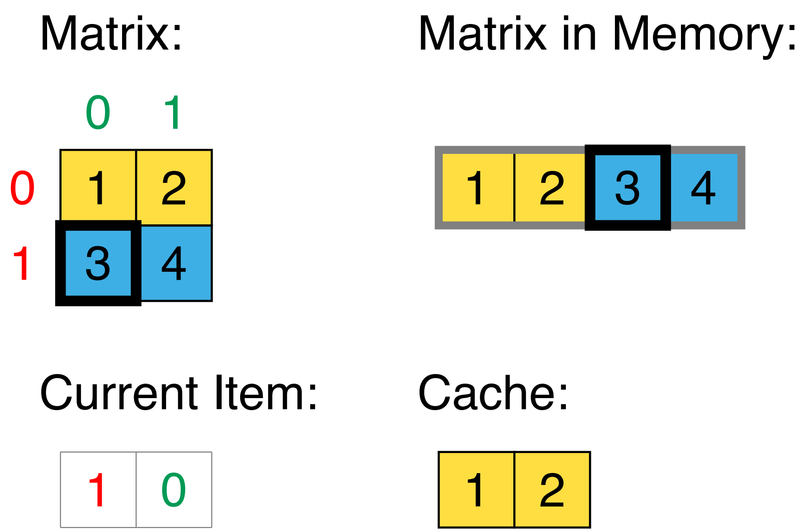


Number of cache misses: **1**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

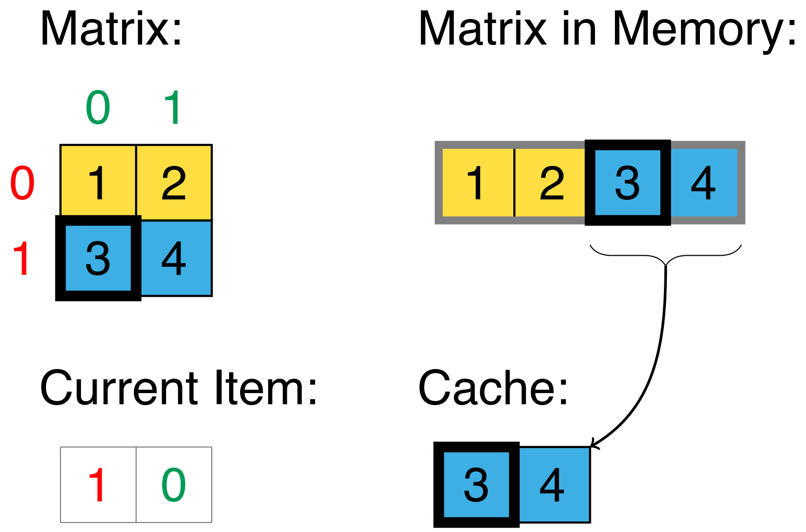


Number of cache misses: **1**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

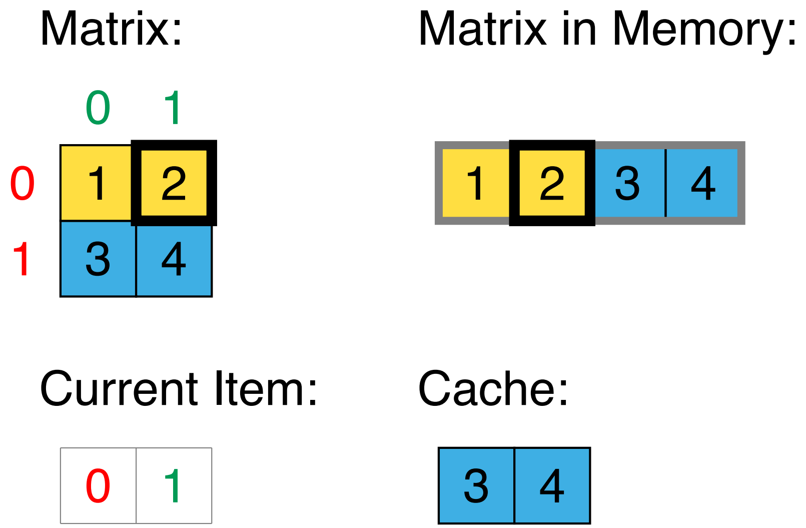


Number of cache misses: **2**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

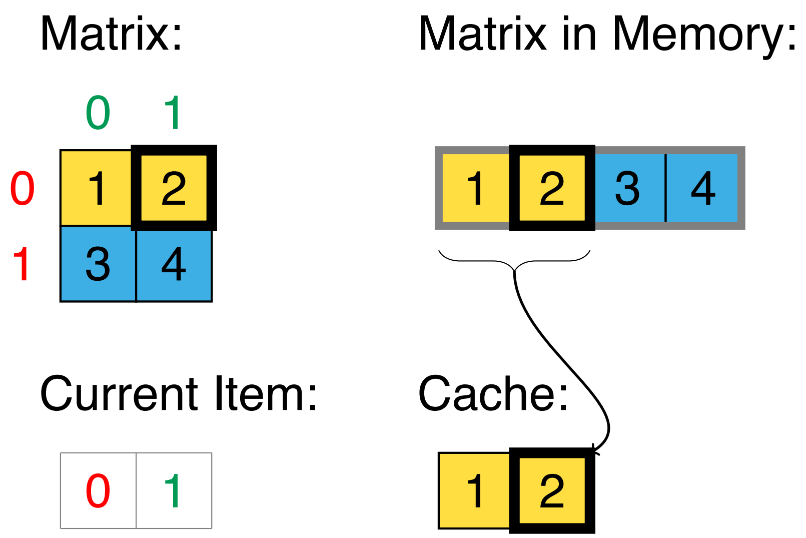


Number of cache misses: **2**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

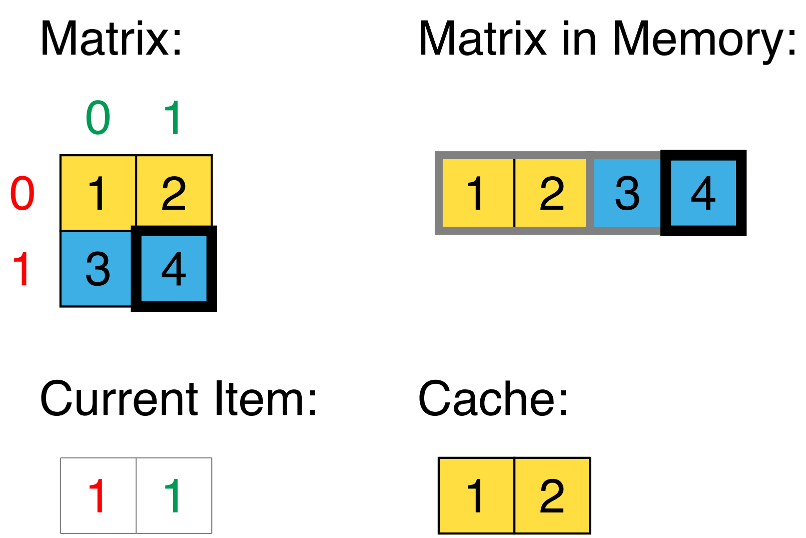


Number of cache misses: **3**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

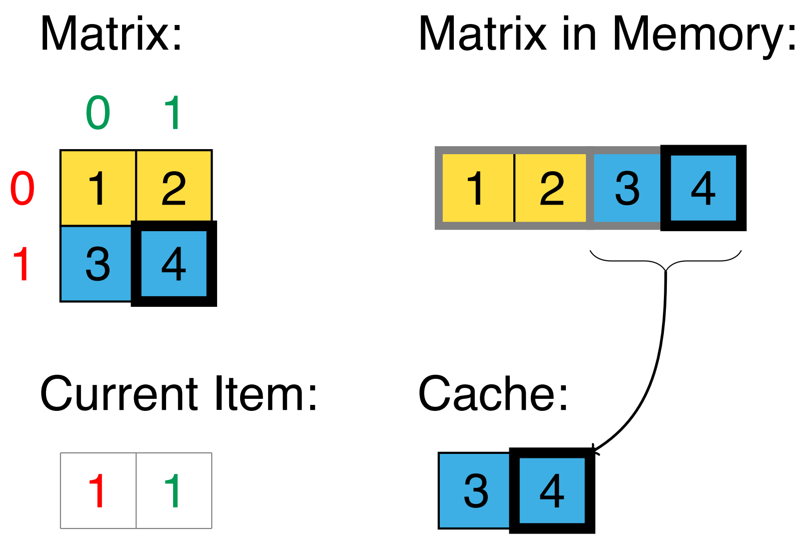


Number of cache misses: **3**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

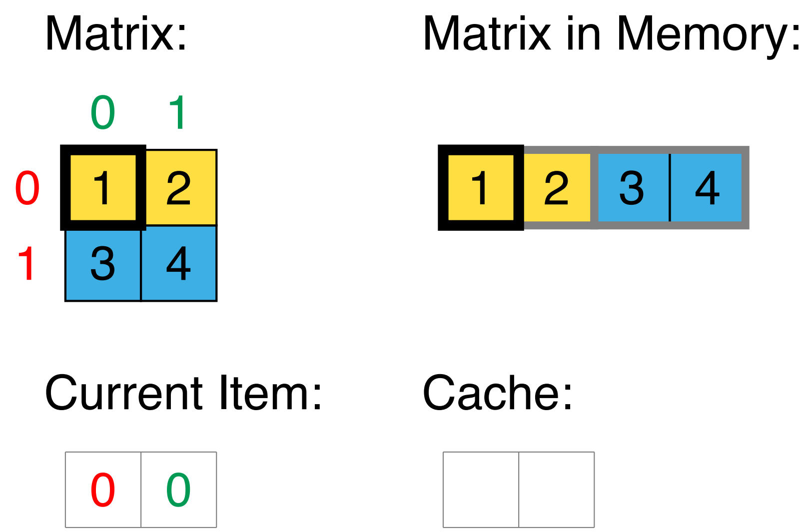


Number of cache misses: **4**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

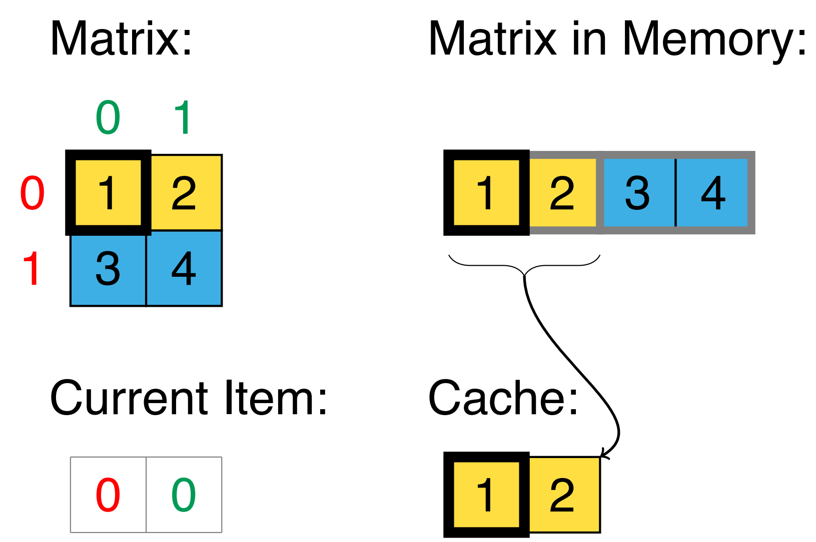


Number of cache misses: **0**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

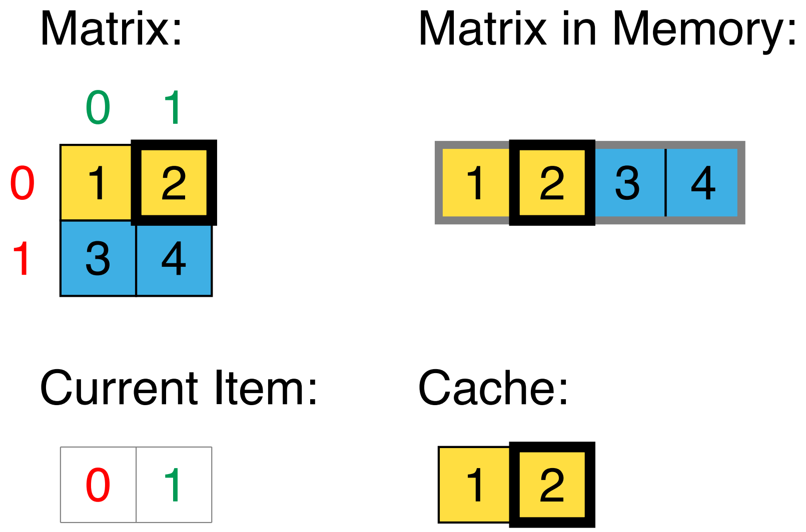


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

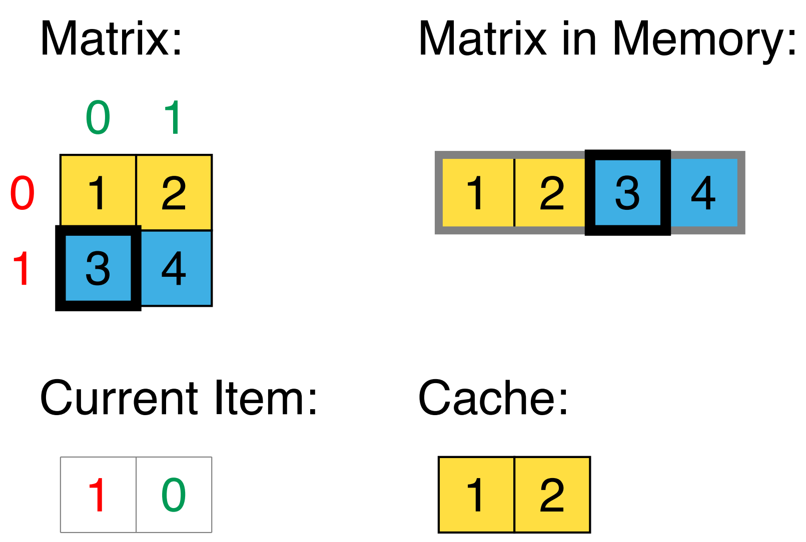


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

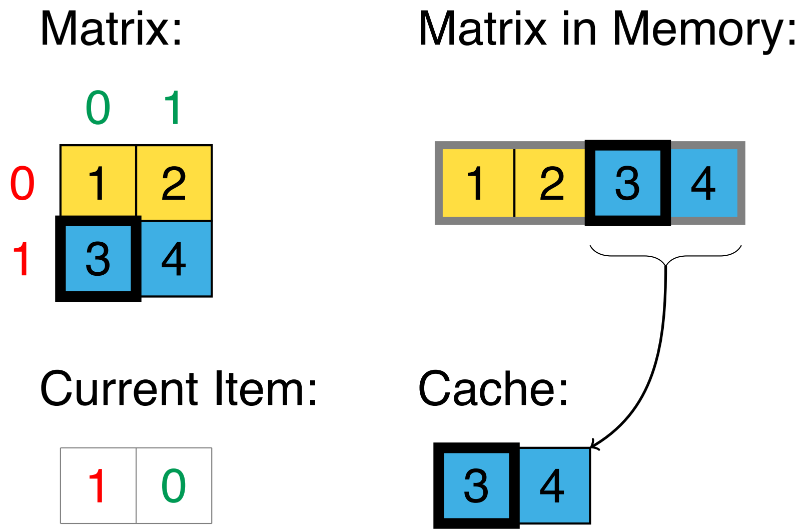


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

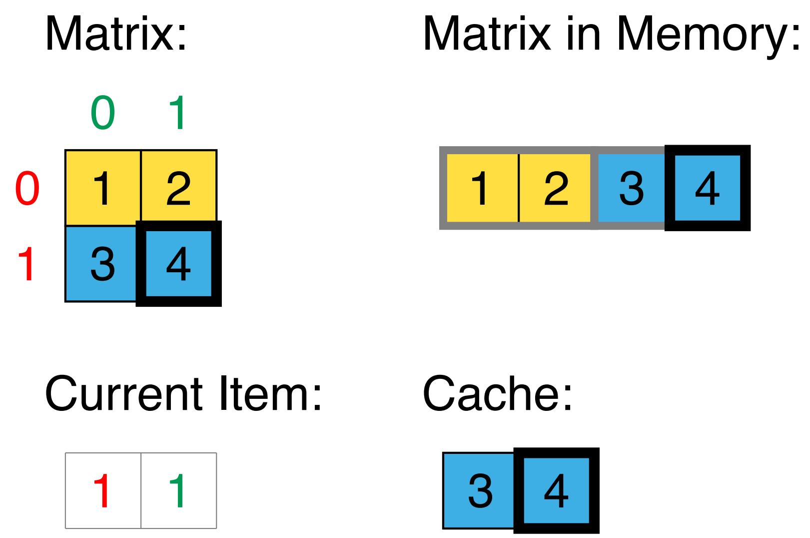


Number of cache misses: **2**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```



Number of cache misses: **2**

Outline

- Memory-Related Performance Problems
  - Data Locality
  - Processor-Memory Performance Gap
- Overview of the Optimization Workflow
- Data Gathering Approaches
- Visualization Techniques
- Specific Optimization Tool
- Conclusion

Memory-Related Performance Problems I Data Locality

$$t_{avg} = p \cdot t_c + (1 - p) \cdot t_m \tag{1}$$

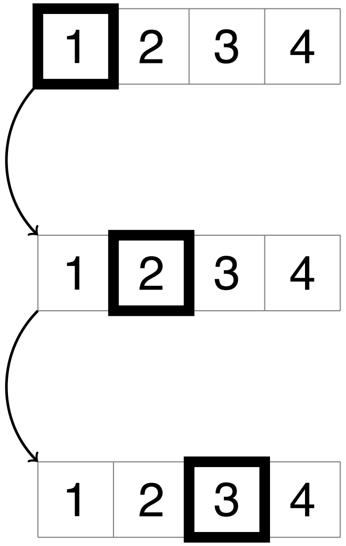
t_{avg} : average access time
p : cache hit percentage
t_c : cache access time
t_m : memory access time

$$t_c \ll t_m \tag{2}$$

Memory-Related Performance Problems I Data Locality

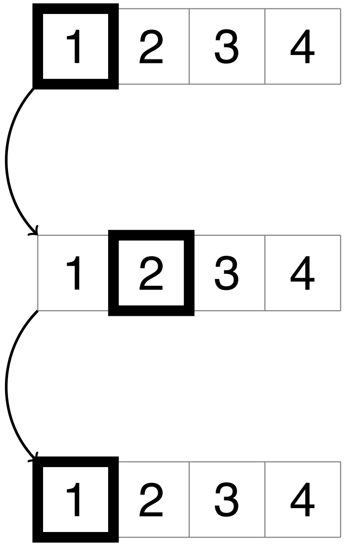
Spacial Locality

- Data that is referenced spatially close together is likely to be accessed in the near future

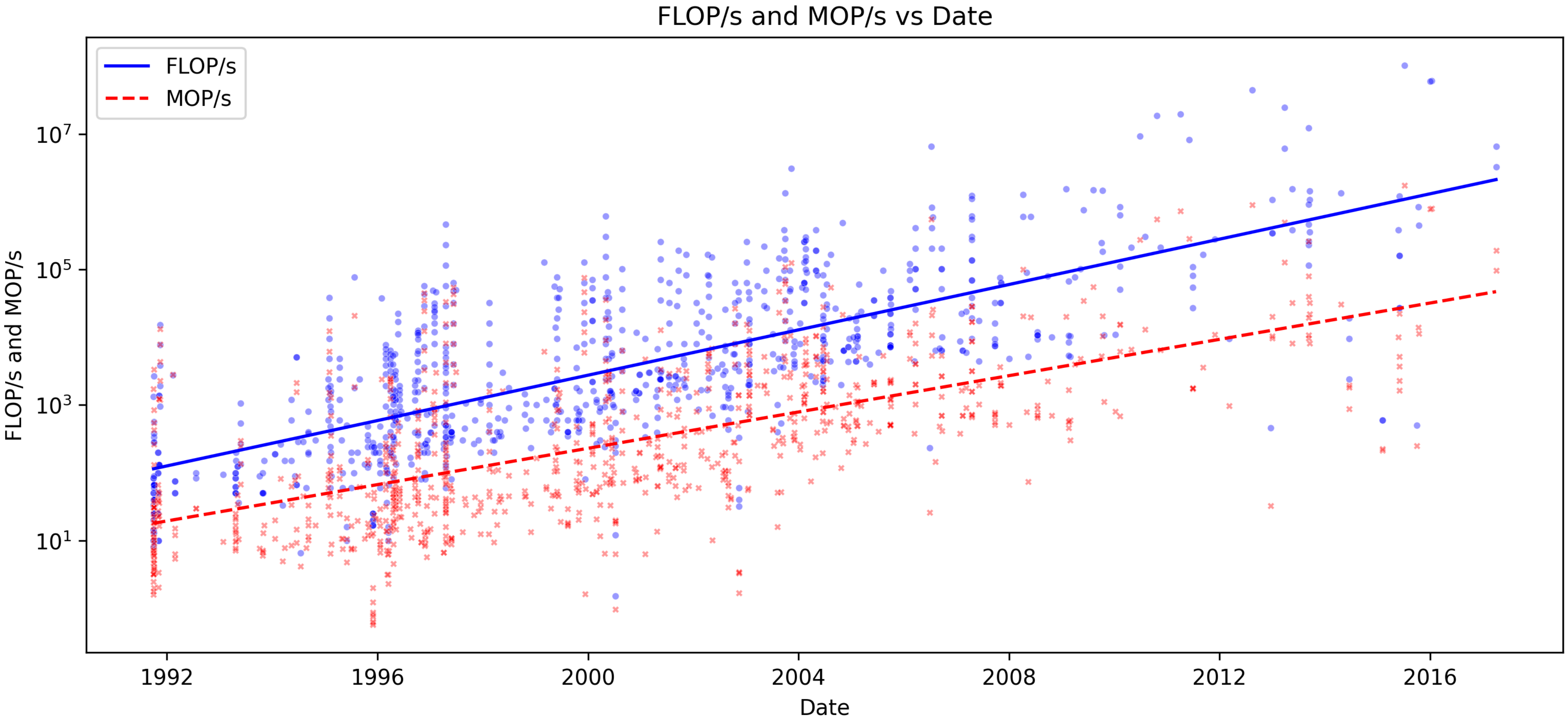


Temporal Locality

- Data that is referenced in the near past is likely to be accessed in the near future

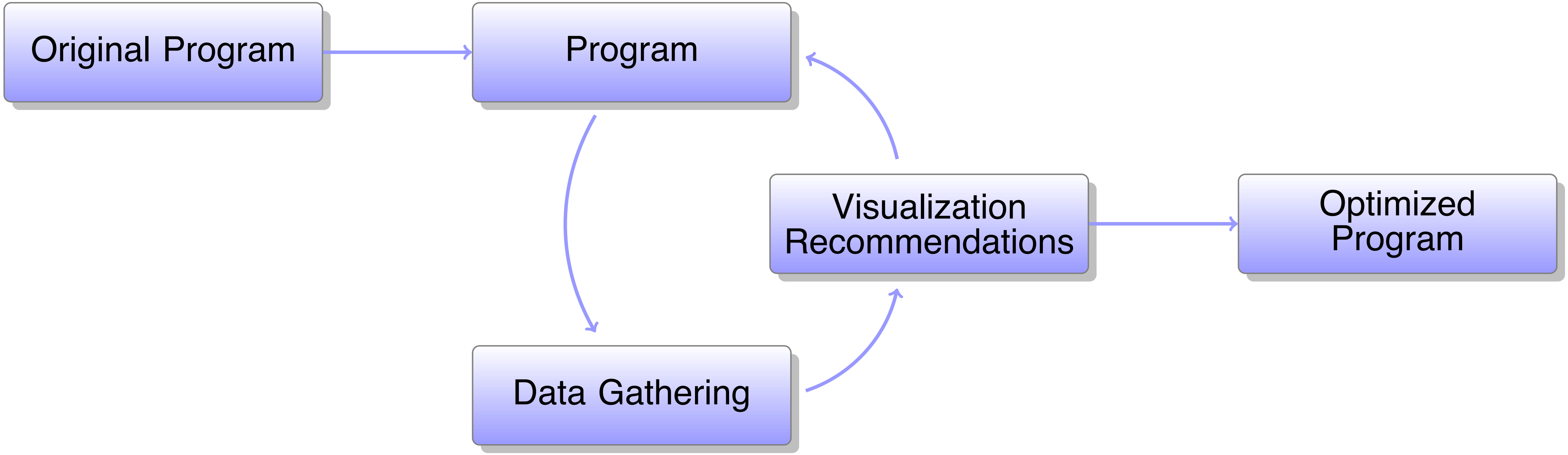


Memory-Related Performance Problems I Processor-Memory Performance Gap



Overview of the Optimization Workflow

Visualization-Guided Optimization



Data Gathering Approaches

Goal: Acquire Memory-Related Data for Visualization

- Data Accesses
  - Memory locations / variables
  - Frequencies
- Data access patterns
  - Nested loops
- Cache performance
  - Hit/miss rates
  - Utilization
  - Amount of data transfer in between different cache levels and main memory

Run Program and Capture Memory-Related Information

- Hardware counters
  - Counts cache hits/misses
- Tracing / profiling
- Store source code references alongside memory-related information

☺ Very accurate
  - Real program data
  - Actual physical hardware

☹ Can be very slow

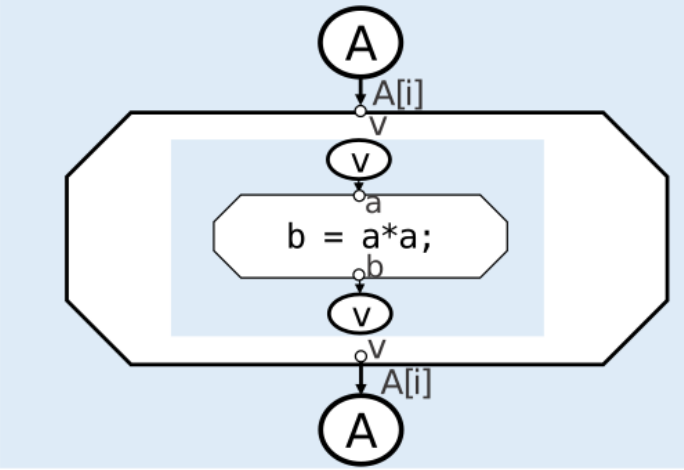
☹ Possible large overhead for very granular data

☹ Cannot easily analyze just parts of the program

Analyze the Programs Source Code for Data Accesses

- Extract any data access information purely from the source code
- Compile the program into a Data-Flow Oriented IR
- Statistics gathered by analyzing the IR
  - Algorithmic intensity
  - Volume of data circulating in the program

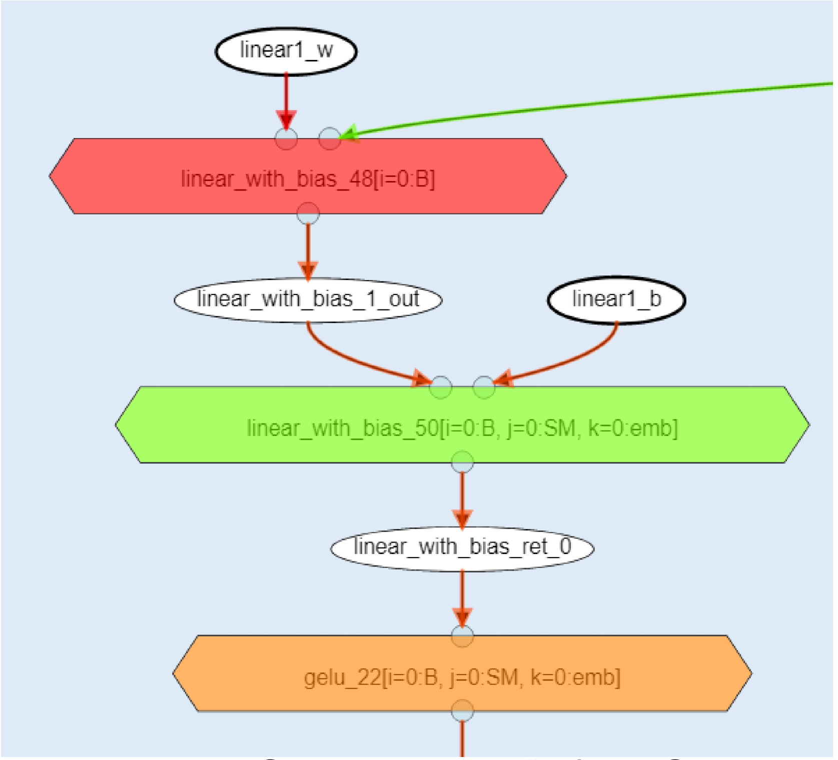
```
void square(double &v) {
      v = v * v;
}

// ...

square(A[i]);
```



Source: *Alexandru Calotoiu et al. “Lifting C semantics for dataflow optimization”. In: Proceedings of the 36th ACM International Conference on Supercomputing. 2022, pp. 1-13.*

Data Gathering Approaches I Static Analyis

Analyze the Programs Source Code for Data Accesses



Source: *Alexandru Calotoiu et al. “Lifting C semantics for dataflow optimization”. In: Proceedings of the 36th ACM International Conference on Supercomputing. 2022, pp. 1-13.*

☺ Very fast

☺ Provides holistic view of the program and its performance

☹ Very abstract analysis

- Memory layout of data is not considered
- Hardware architecture unknown
- No information about real-world cache performance

Data Gathering Approaches I Cache Simulation

Imitate the Programs Memory Accesses on a Simulated Cache Hierarchy

- Replicate actual hardware through software
  - Cache hierarchy (size, associativity, etc.)
  - Cache replacement policies
  - Cache coherence protocols
- Simulate the programs memory-wise on the simulated hardware
  - Memory (de-)allocations
  - Data accesses

☺ Very detailed
- Insights about the memory-layout of data
- Enables step-by-step analysis of the caches

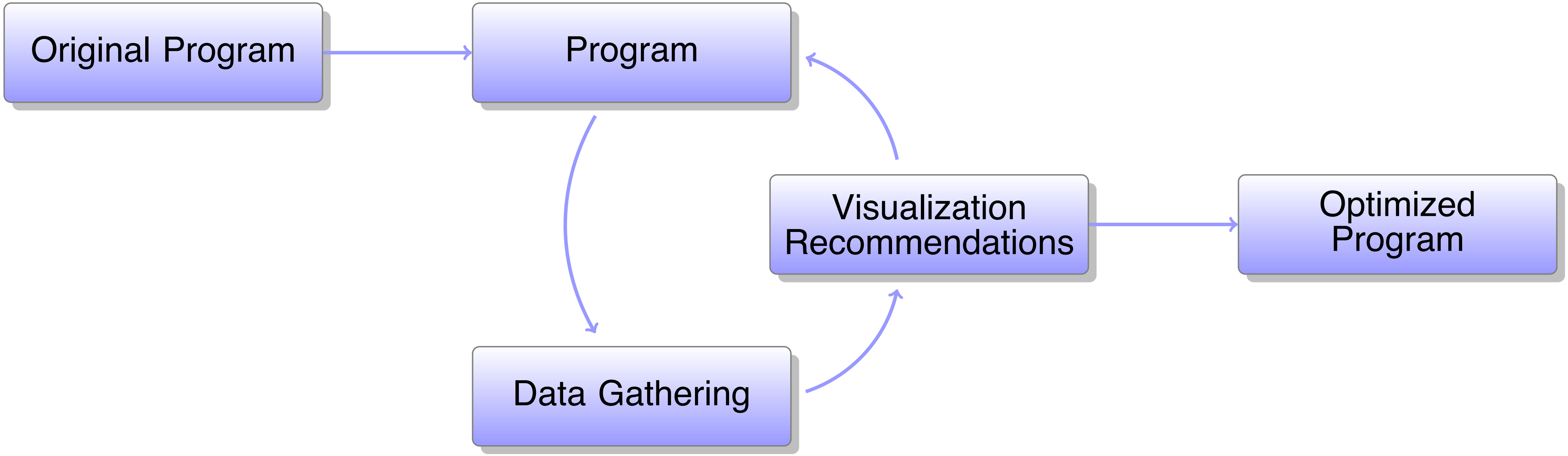
☺ Allows to analyze only parts of the program

☹ Requires precise parameterization

Visualization Techniques

Goal: Display Bottlenecks (and explain them)

Visualization-Guided Optimization



References I

- Alexandru Calotoiu et al. “Lifting C semantics for dataflow optimization”. In: *Proceedings of the 36th ACM International Conference on Supercomputing*. 2022, pp. 1-13.

i12 High Performance Computing

RWTH AACHEN UNIVERSITY