Improving Memory Performance of Embedded Java Applications by Dynamic Layout Modifications

F. Li, P. Agrawal, G. Eberhardt, E. Manavoglu, S. Ugurel, M. Kandemir

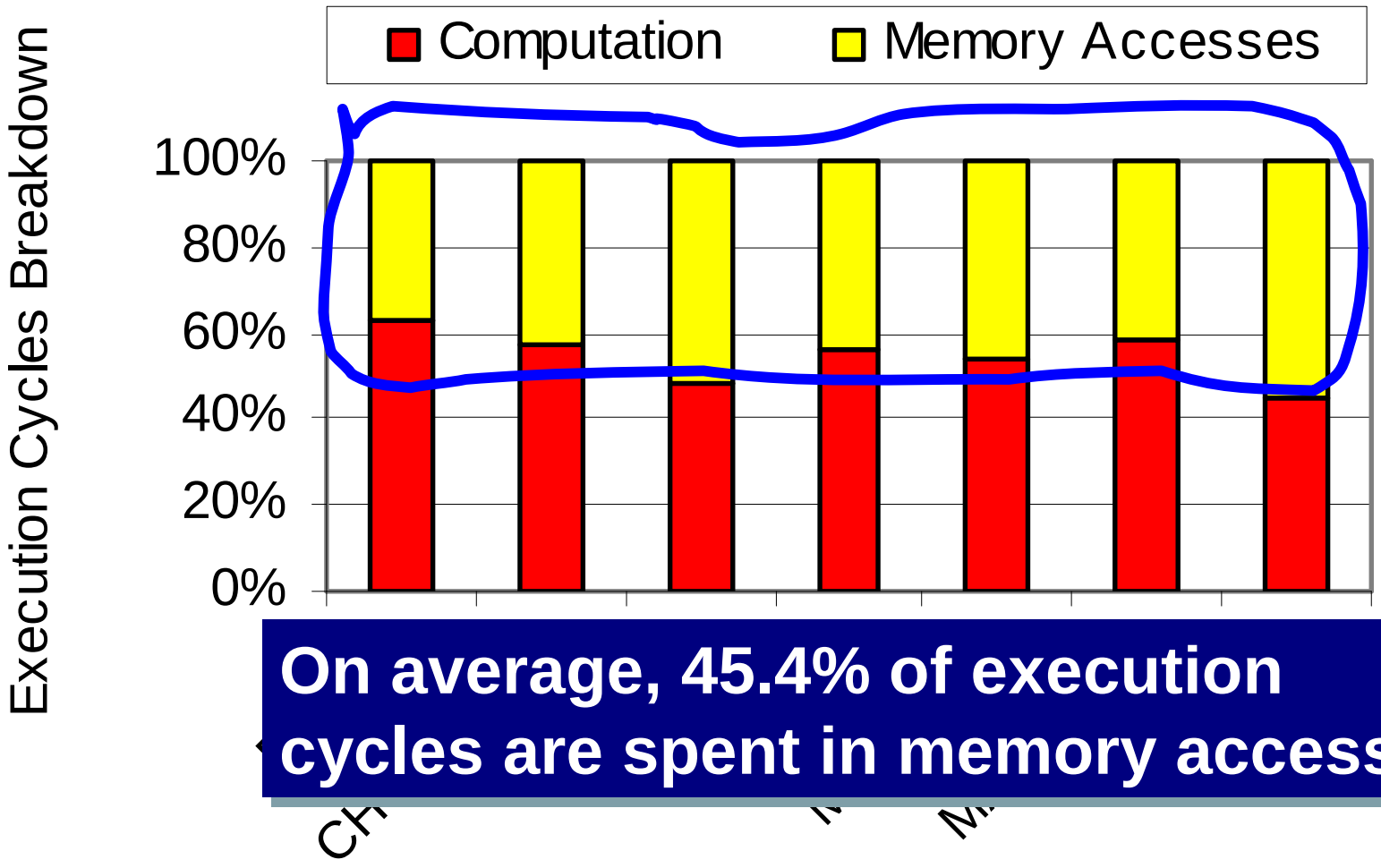
Department of Computer Science & Engineering

The Pennsylvania State University

Introduction and Motivation

- Java widely used for embedded platforms
- Java is slow for array-based applications
  - 130 times more slowly than C program for a matrix multiplication program
- Many embedded applications are array-based
  - Image, video…
- Optimization needed for array based embedded Java applications

Execution Cycles Breakdown



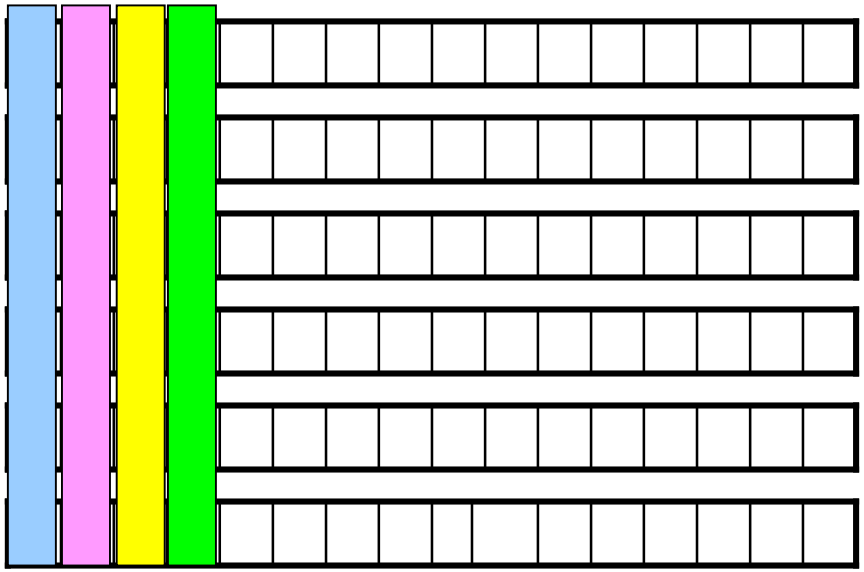
On average, 45.4% of execution cycles are spent in memory access

Objective

- Improve the memory performance of embedded Java applications

Cause of Cache Misses

- Mismatch between array access pattern and array data layout



Row-major data layout

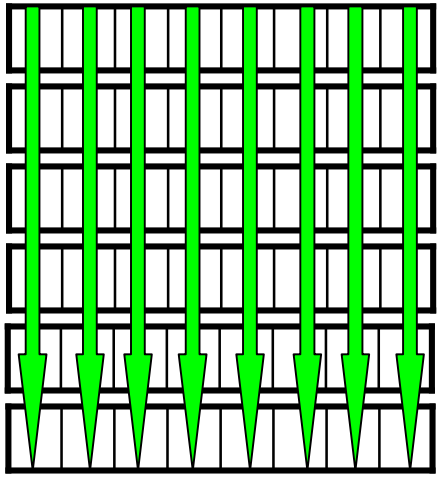
Access columns in the innermost loop

Classical Ways to Improve Memory Performance

- Loop transformation
  - Change access pattern
  - Restricted by inherent data dependence
- Array layout transformation
  - Change the way that array data are stored
  - Global effect for accesses to this array
  - Not restricted by data dependence

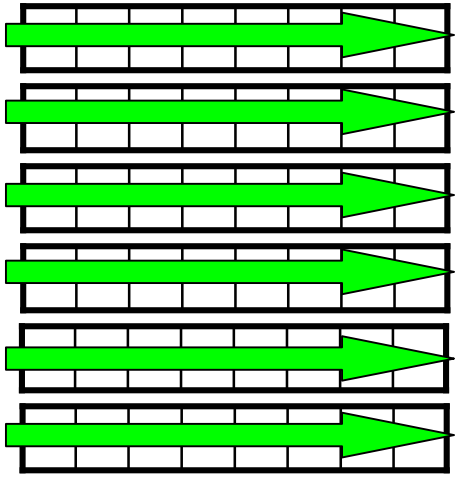
Transformations

Original



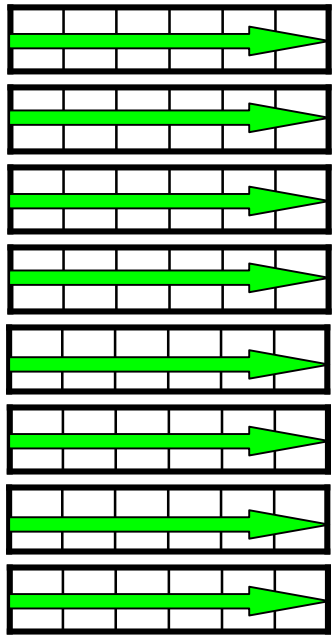
```
int A[M][N];
for(i=0;i<N;i++)
  for(j=0;j<M;j++)
  … = A[j][i]
```

Loop Transformation



```
int A[M][N];
for(j=0;j<M;j++)
  for(i=0;i<N;i++)
  … = A[j][i]
```

Data Transformation



```
int A[N][M];
for(i=0;i<N;i++)
  for(j=0;j<M;j++)
  … = A[i][j]
```

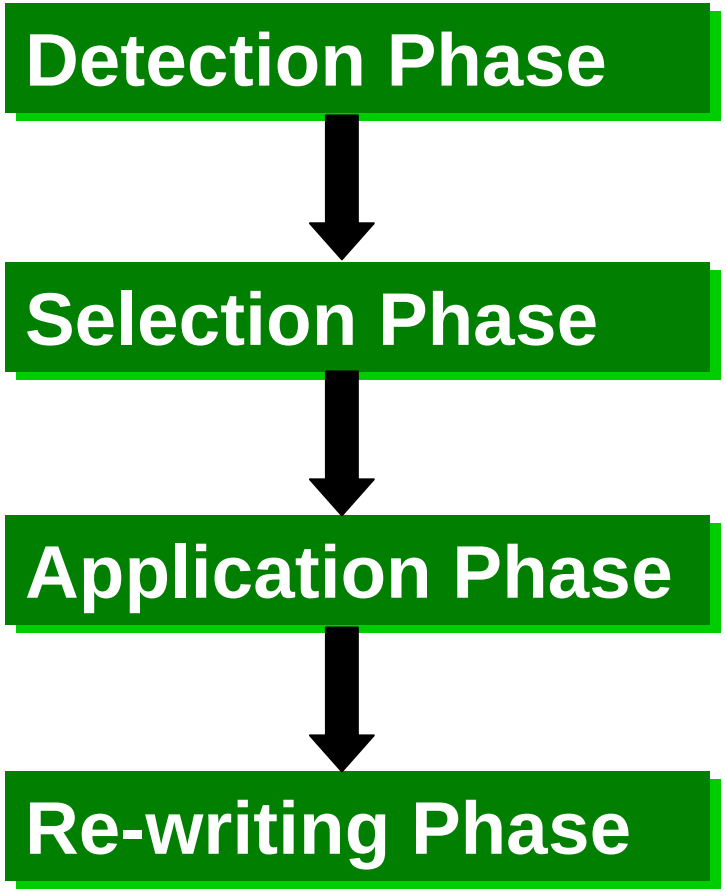
Transformation for Java Programs

- Loop transformation is not suitable for Java programs
  - Violate the precise exception rule
  - Java programs are distributed as bytecodes, not as Java source
- Array layout transformation
  - Can be implemented in JVM
  - Dynamic layout transformation

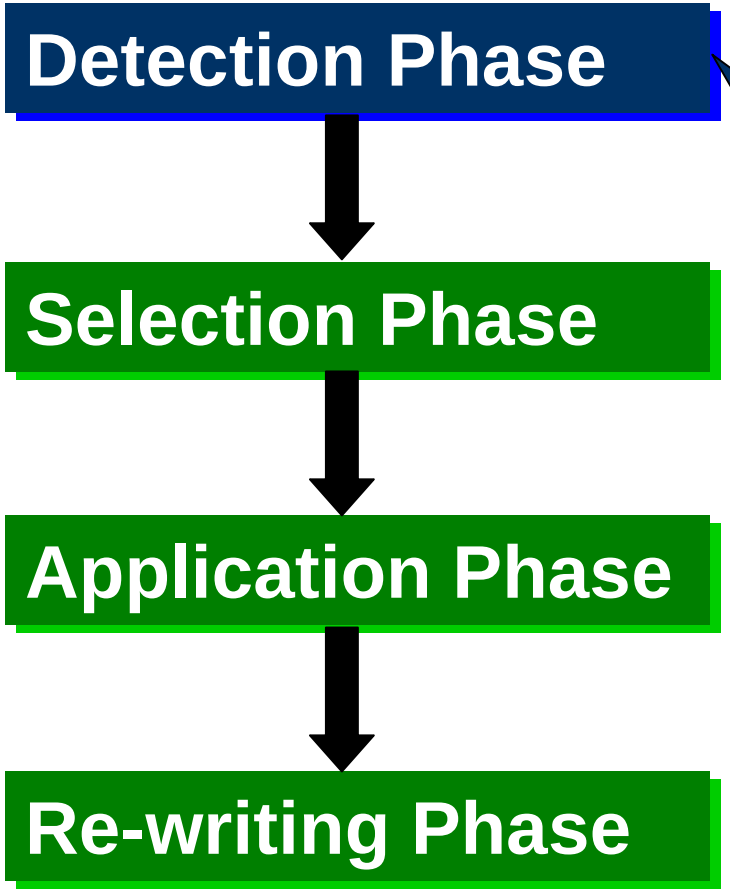
Dynamic Layout Transformation

- Transform array data layout while JVM is running
- Incur performance overhead
  - Apply with care
- Transform an array only if it is really necessary (very bad cache behavior)
  - Accessed frequently enough
  - Shows bad cache behavior

Our Approach

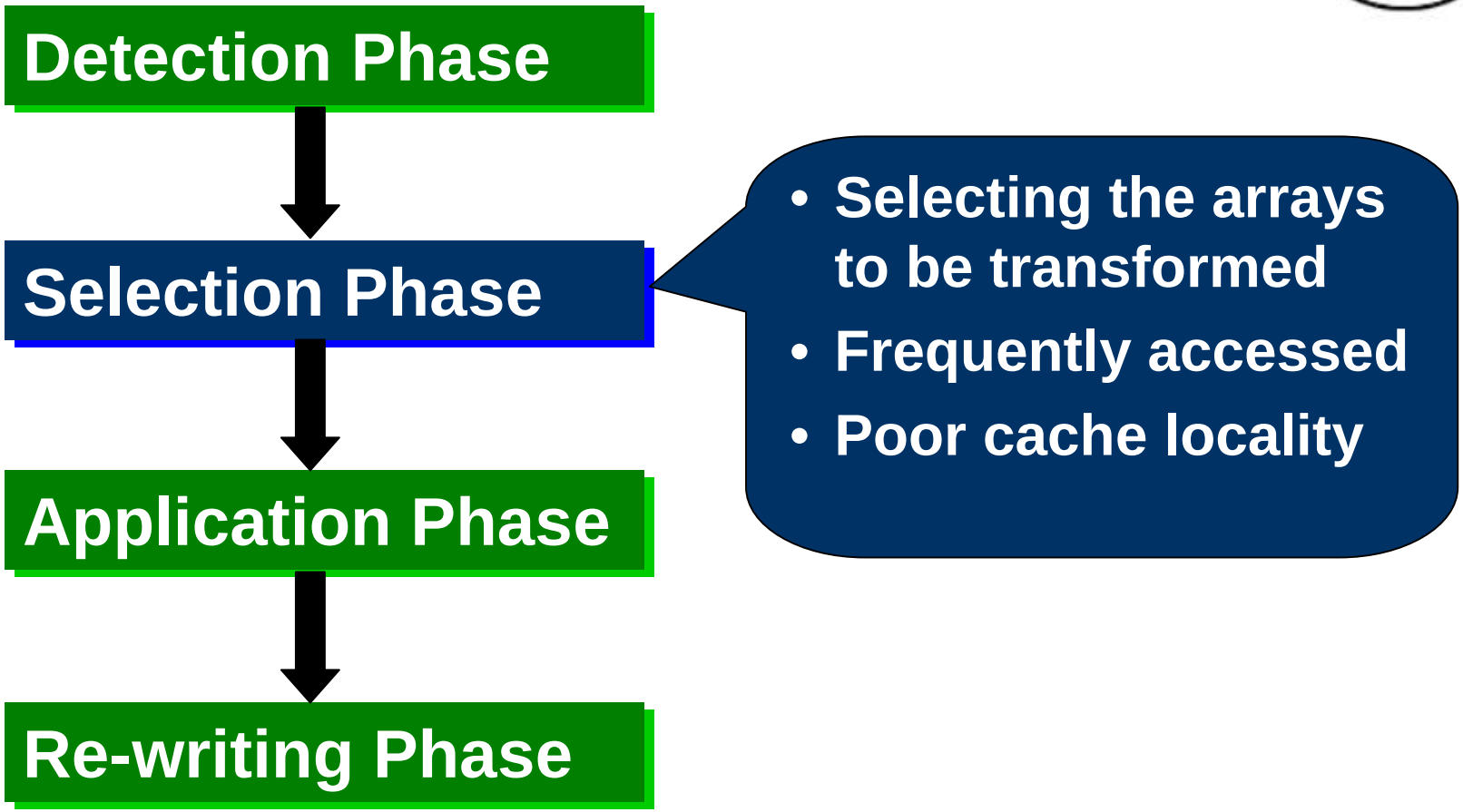


Our Approach

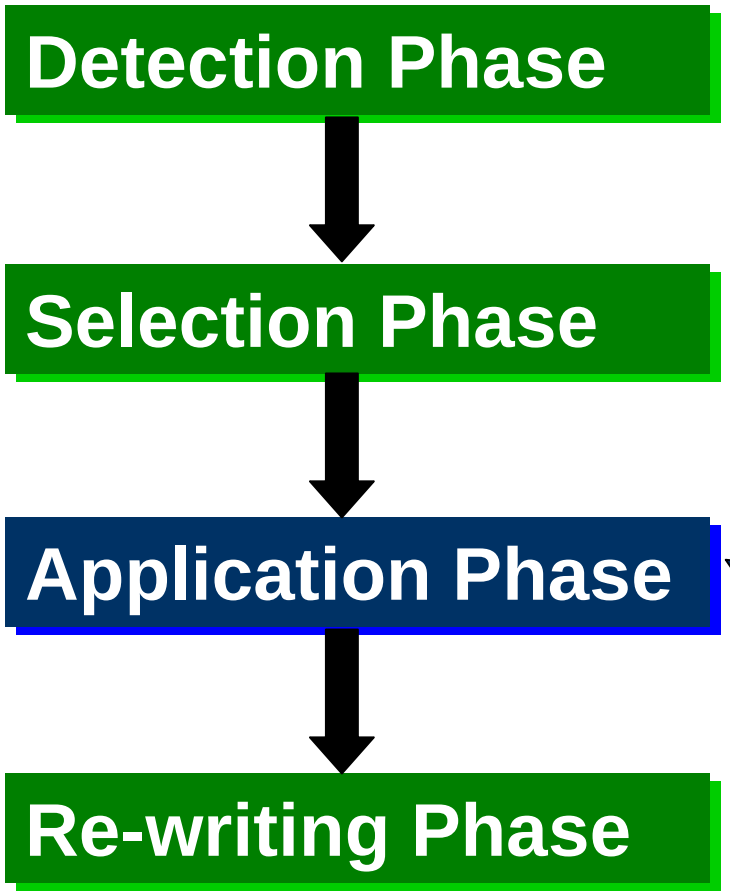


- Detecting the degradation in the cache performance
- Trigger data transformation only if cache miss rate is high enough

Our Approach

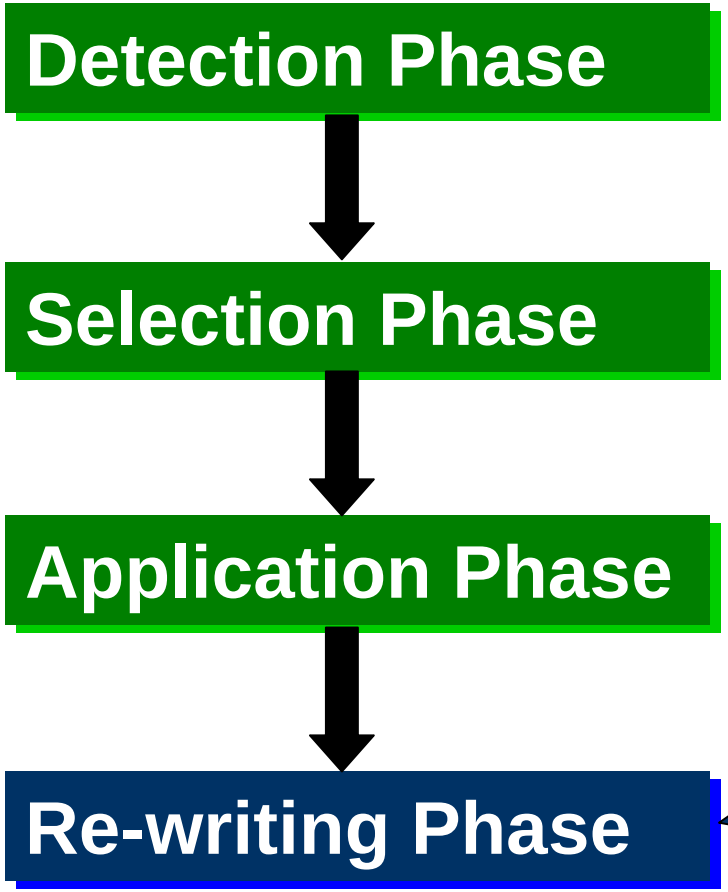


Our Approach



- Deciding how to transform the selected arrays
- Apply the transformations
- Bring an outer dimension to the innermost

Our Approach



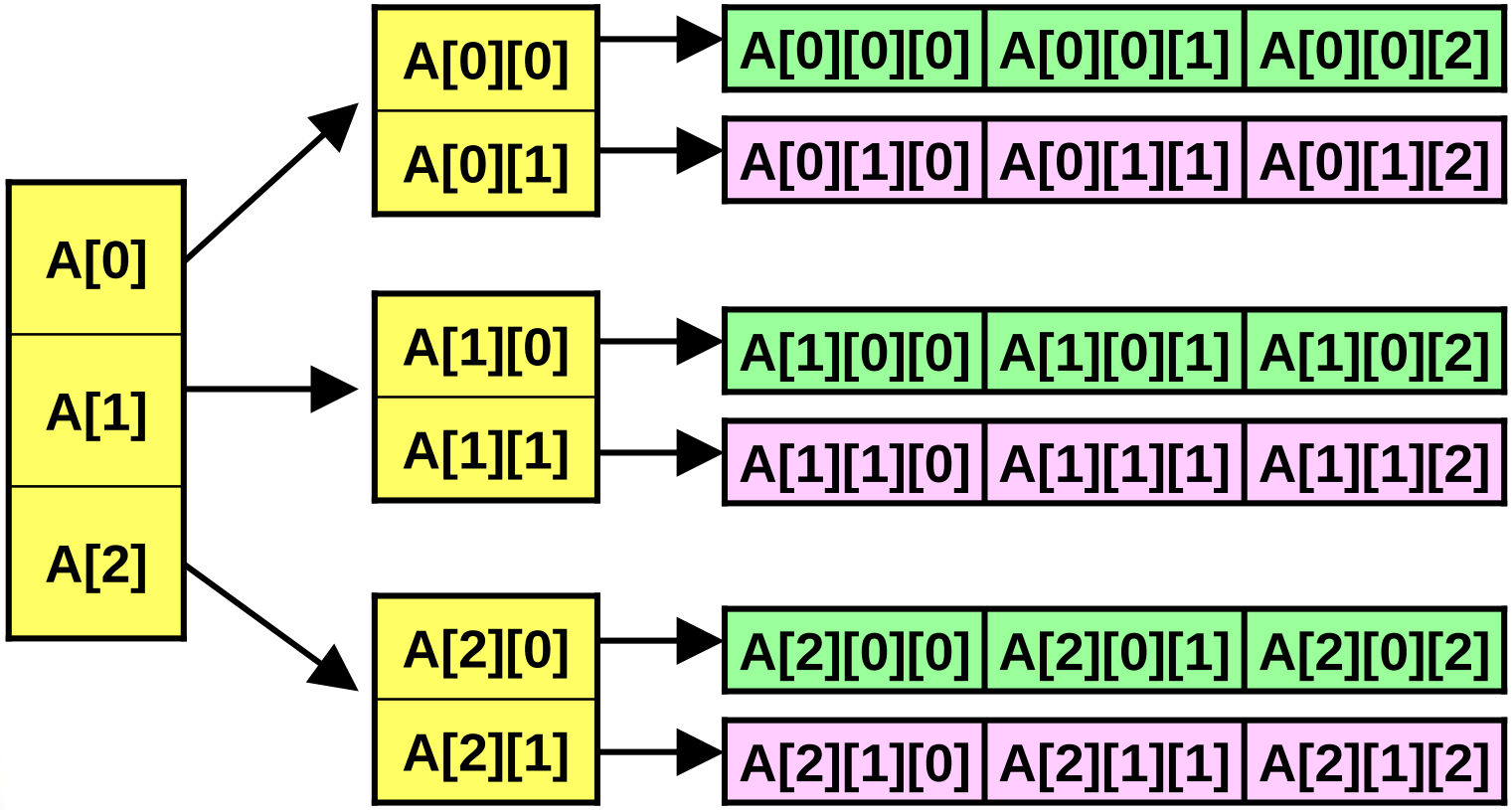
- Change the array references in the code to reflect the new memory layout
- Parse the bytecode and modify it

Summary of Our Approach

- Adaptively select the arrays that need to be transformed
- Make the innermost loop traverse the array along the fastest changing subscript position
  - May take several transformations until the right one is found, and cache behavior is improved
- Rewrite bytecode to avoid using transformation table
- Reduce capacity misses

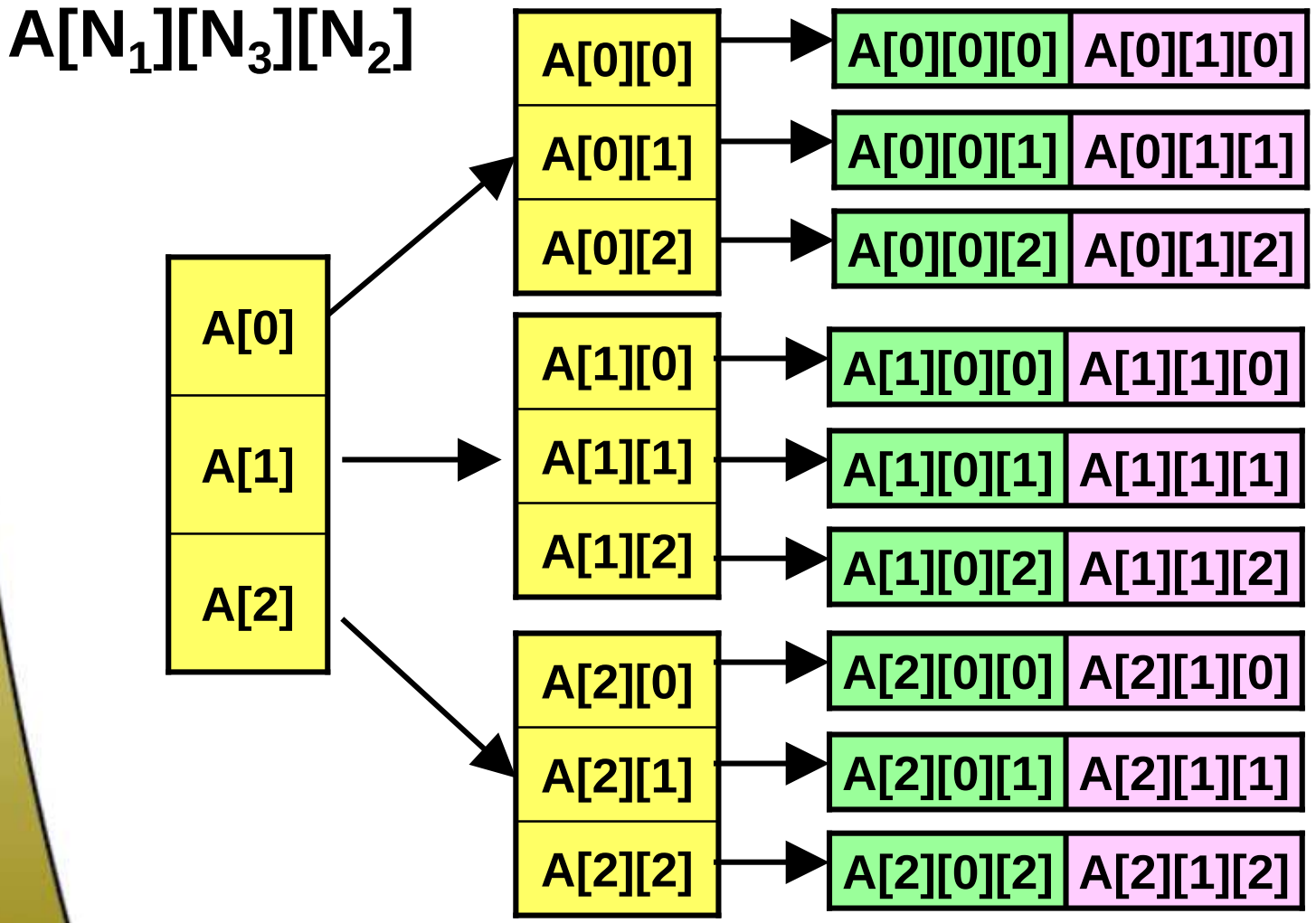
Java Array Layout

$A[N_1][N_2][N_3],\ N_1=3,\ N_2=2,\ N_3=3$



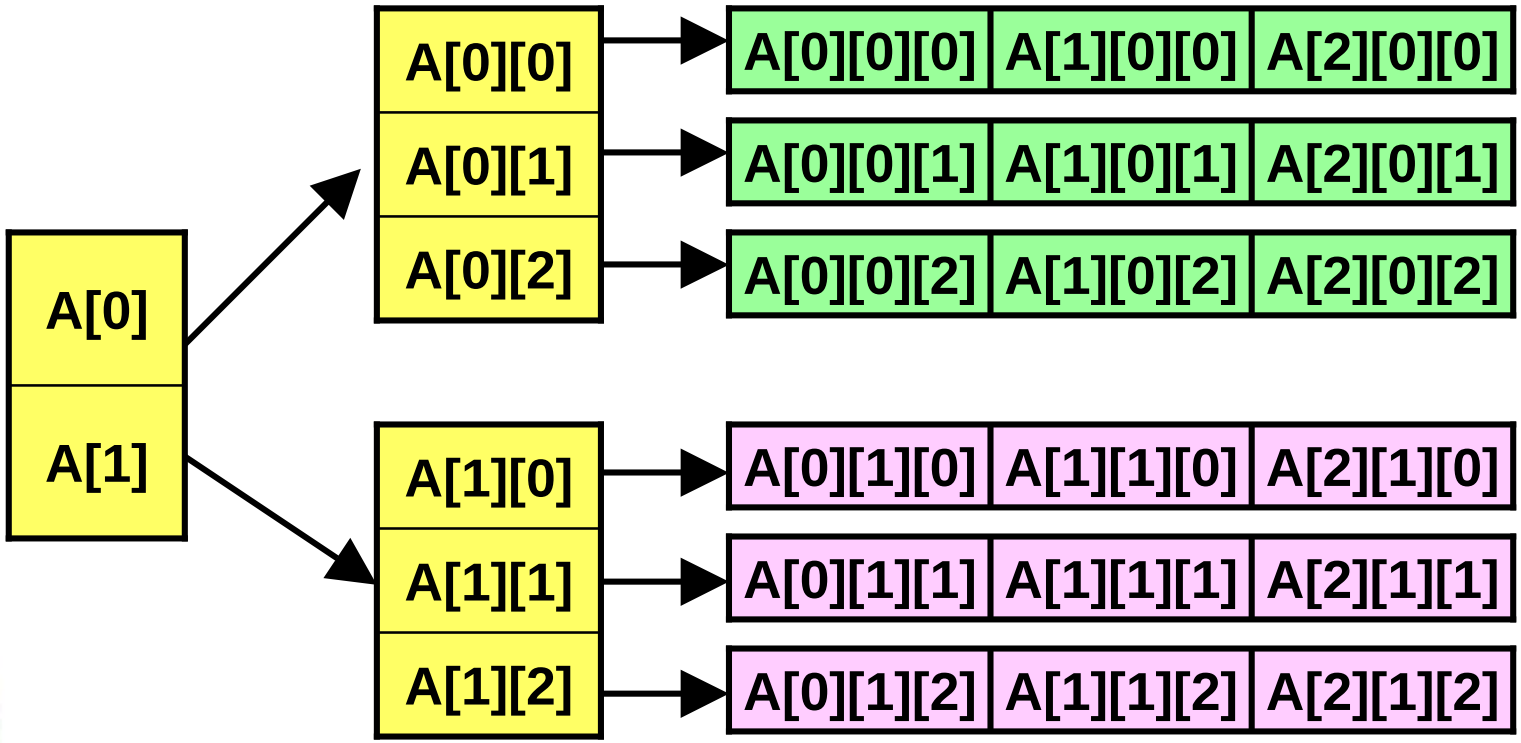
Row-pointer memory layout

Transformed Array Layout



Another Transformation

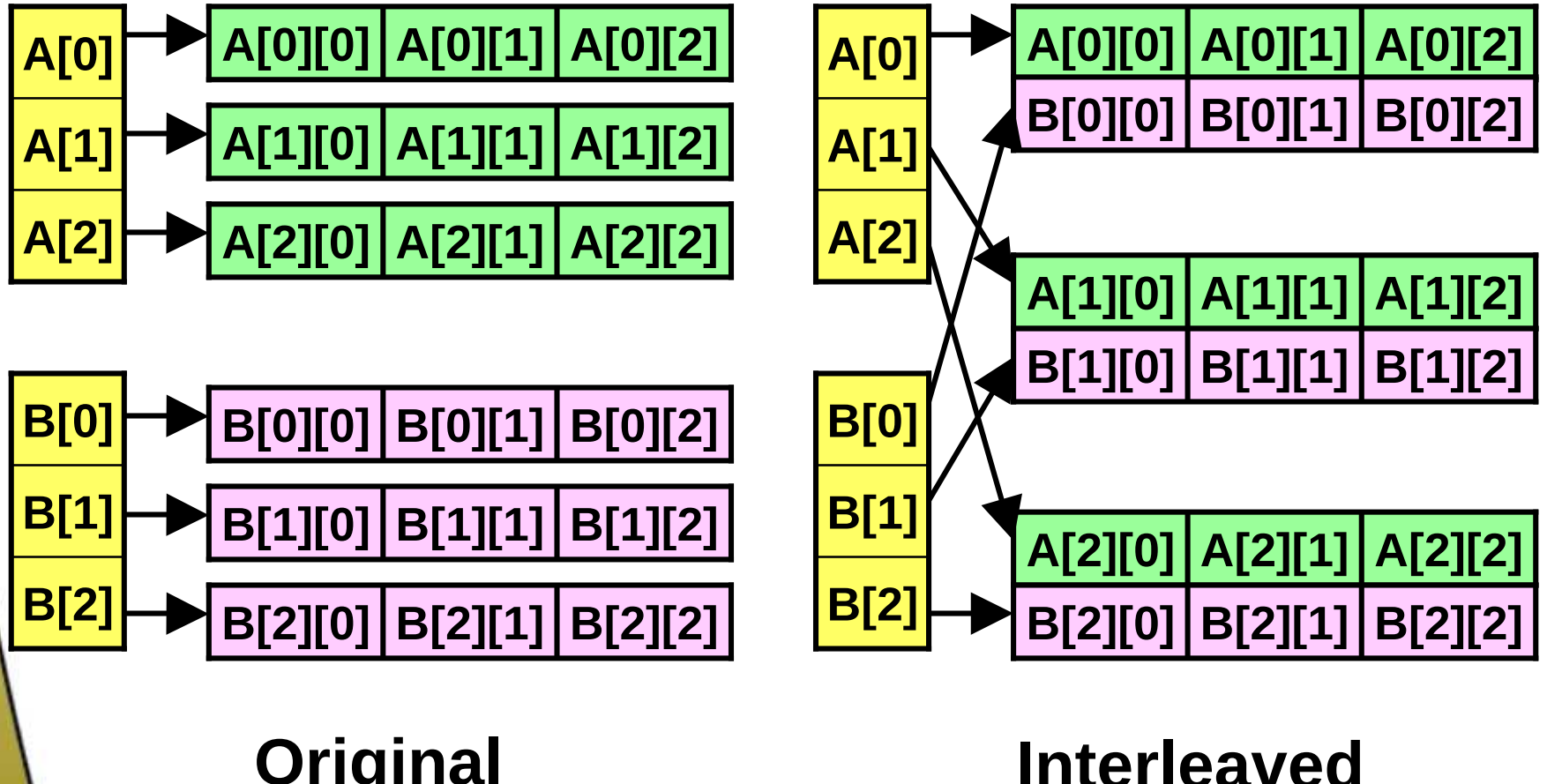
$A[N_2][N_3][N_1]$



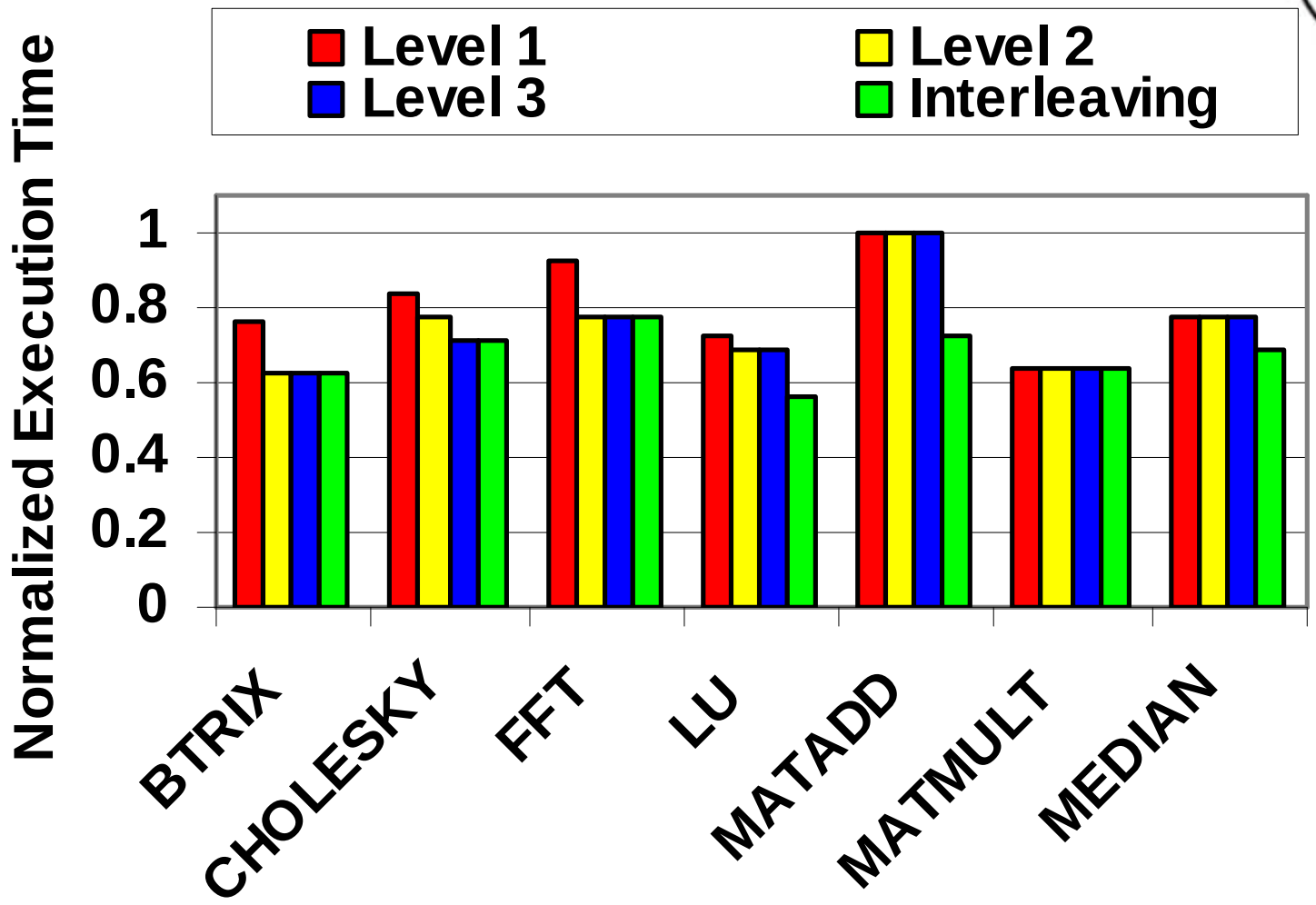
Array Interleaving

- Two arrays are accessed together in a loop
  - Conflict misses
- Array interleaving
  - Interleave data from two arrays together
  - Two arrays must belong to the same compatibility set
    - Their access frequencies are very similar
- Multiple arrays interleaving is possible

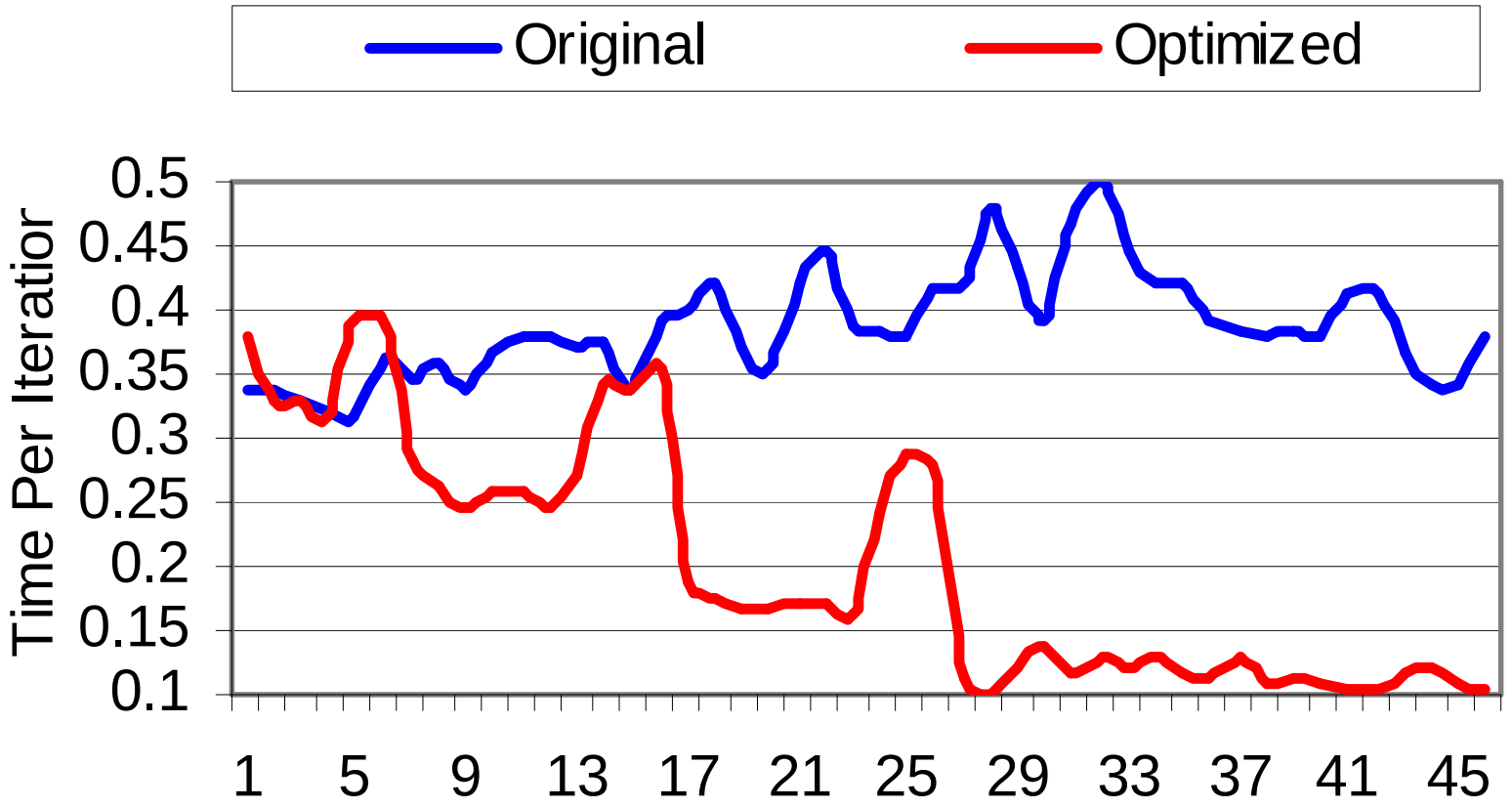
Array Interleaving



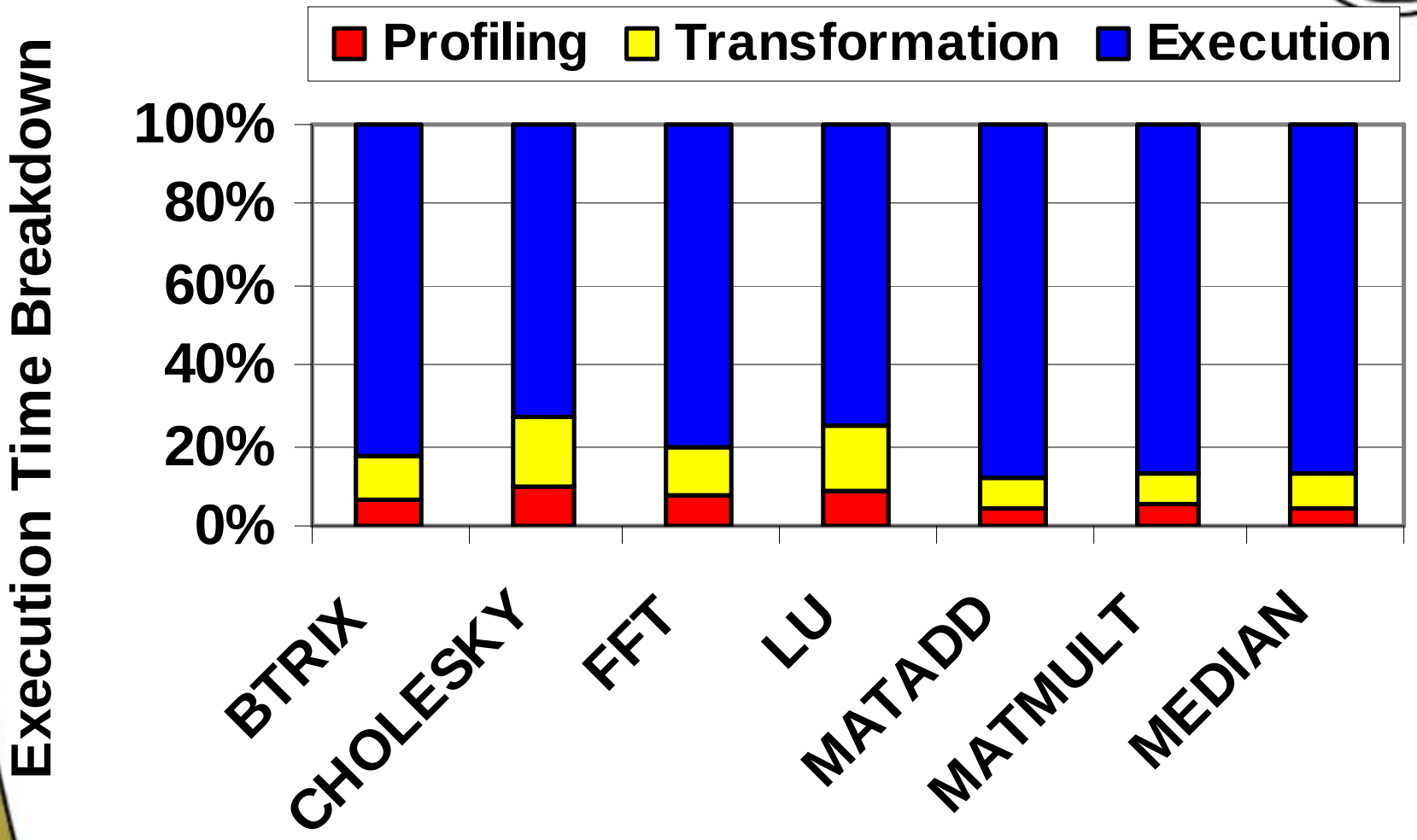
Normalized Execution Time



Time per Iteration



Execution Time Breakdown of the Optimized Codes



Conclusion and Future Work

- Run-time layout transformation for embedded Java applications
  - Reduce cache misses
- Future Work
  - More sophisticated transformations
  - Skip transformation according future usefulness
  - Port to other JVMs