UNIVERSITY of DELAWARE

Group Locality: Gregarious Data Restructuring

Sunil Shrestha[1], Joseph Manzano[2], Andres Marquez[2] and Guang R. Gao[1]

[1]University of Delaware

[2]Pacific Northwest National Laboratory

Pacific Northwest
NATIONAL LABORATORY

Introduction

- Memory access latency, a bottleneck to higher performance
- Abundant processing resources, limited shared resources
- Execution strategy
  - Coarse grain: Resource Underutilization
  - Fine-grain: Locality Agnostic
- Access Pattern crucial for cache utilization
- Thread Collaboration essential for higher performance

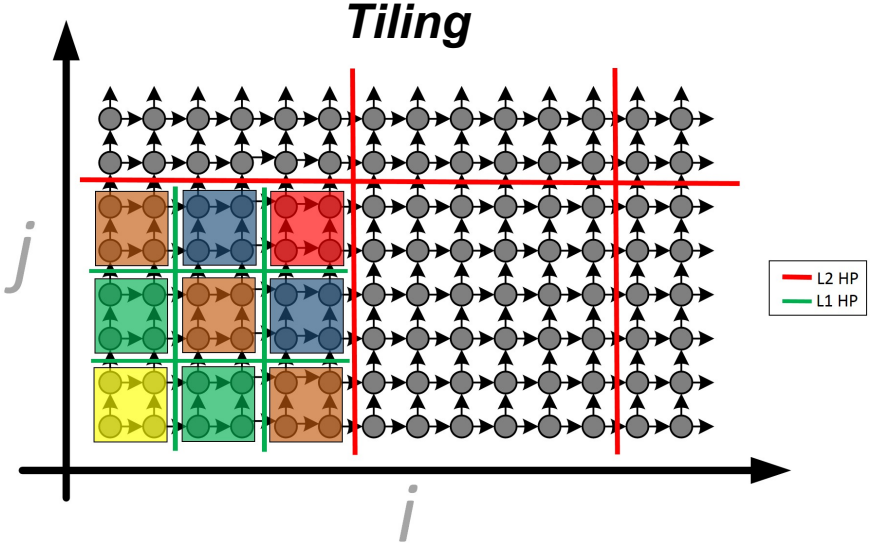
Contributions

A restructuring framework that takes access pattern into account and restructure data for group benefits. Contributions can be summarized as:

1. Collaborative data restructuring for group reuse.
2. Low overhead transformation technique.

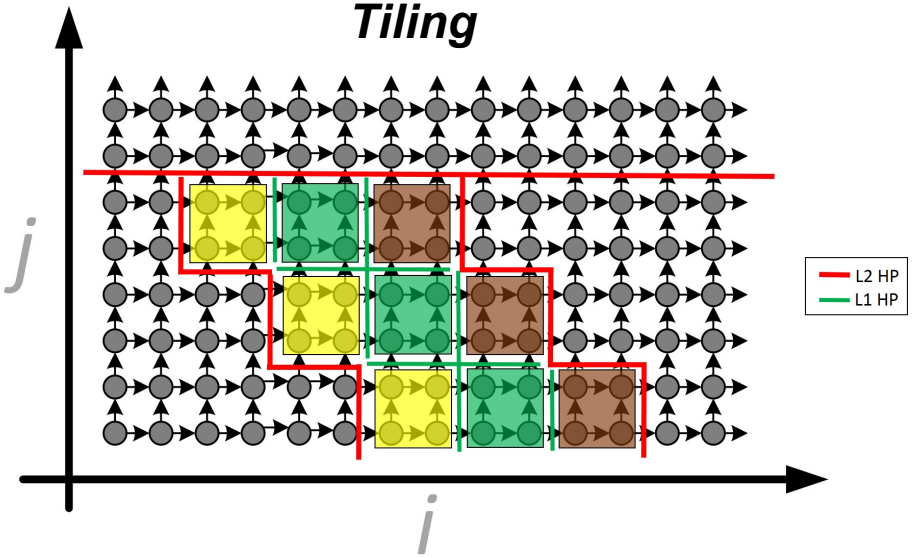
Recap: Jagged Tiling for Pipeline Start

Traditional Hierarchical Tiling



L1 HP : (1,0) and (0,1)
L2 HP : (1,0) and (0,1)

Jagged Hierarchical Tiling

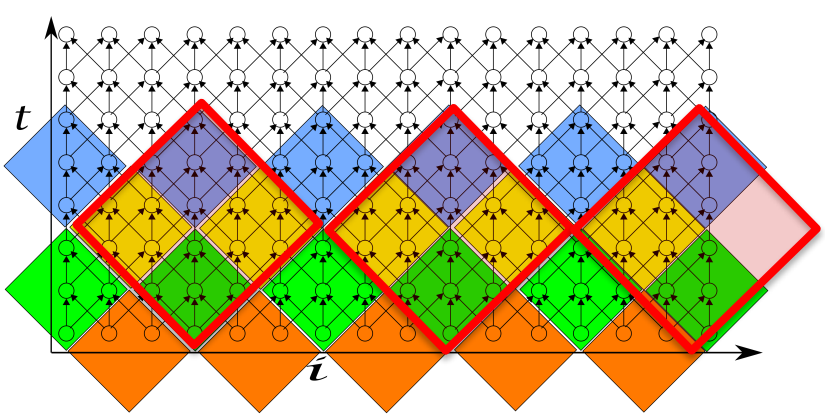


L1 HP : (1,0) and (0,1)
L2 HP : (1,1) and (0,1)

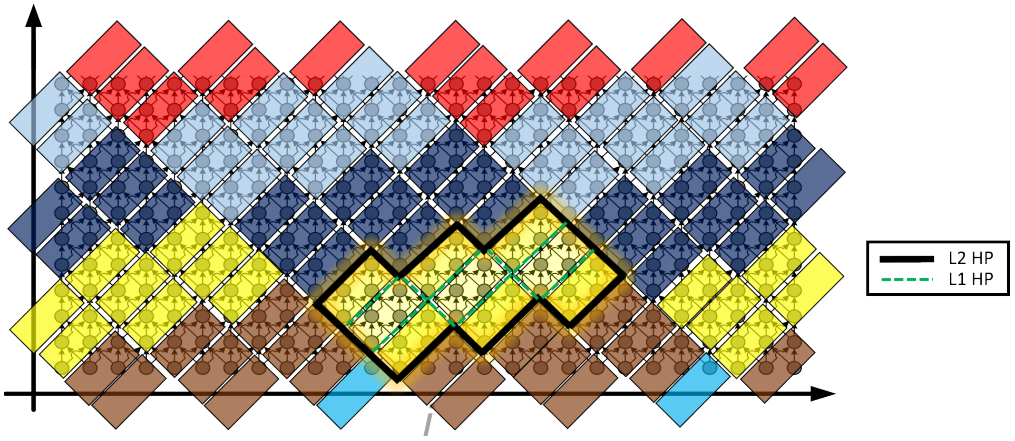
```
for (i=1; i<n;i++) {
   for (j=1; j<n;j++){
      A[i][j]=A[i-1][j]+A[i][j-1];
   }
}
```

Pacific Northwest NATIONAL LABORATORY

Recap: Jagged Tiling for Concurrent Start



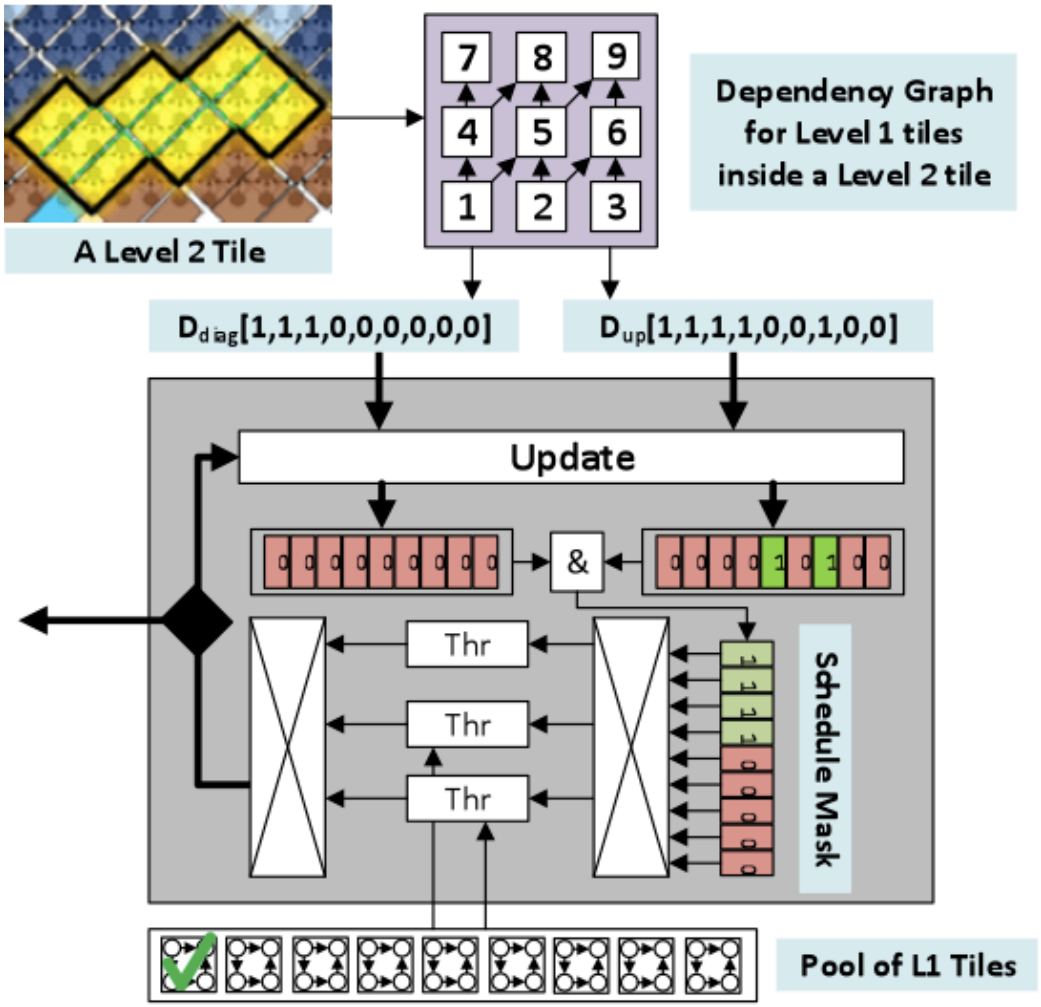
Hierarchical-Level



Heat-1d

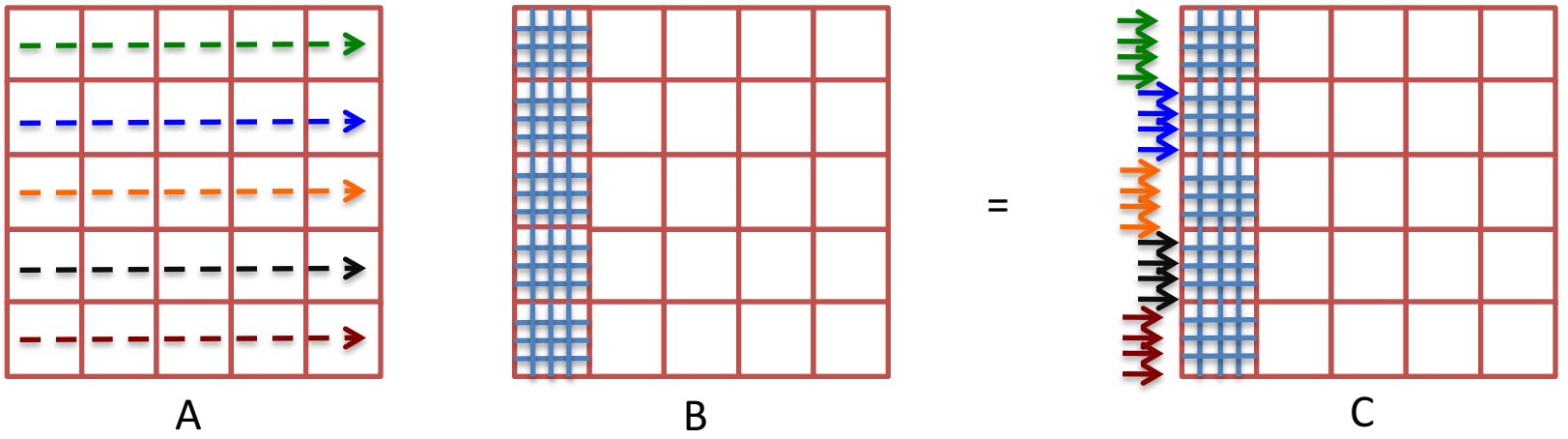
```
for (t=0; t<T;t++)
  for (i=1; i<N;i++)
    A[t+1][i]=α*(A[t][i+1]-
        β*A[t][i] + A[t][i-1])
```

Recap: Fine-Grain Execution



- **Thread Grouping**
- Take advantage of outer tile(L2) locality.
- Uses low overhead atomic operation
- Hierarchical

Motivational Example: Matrix Multiplication

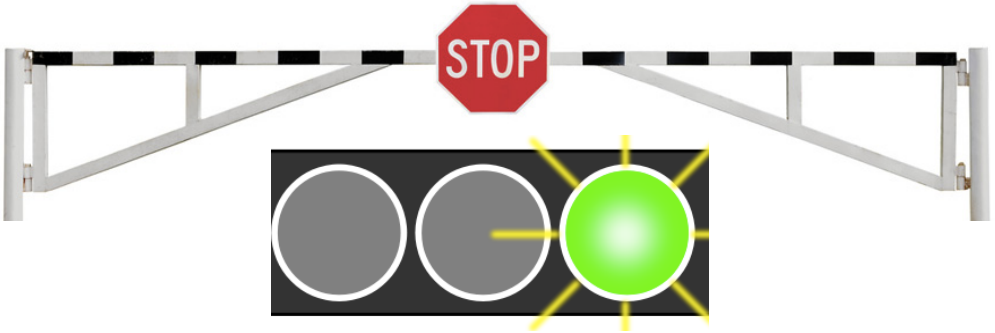


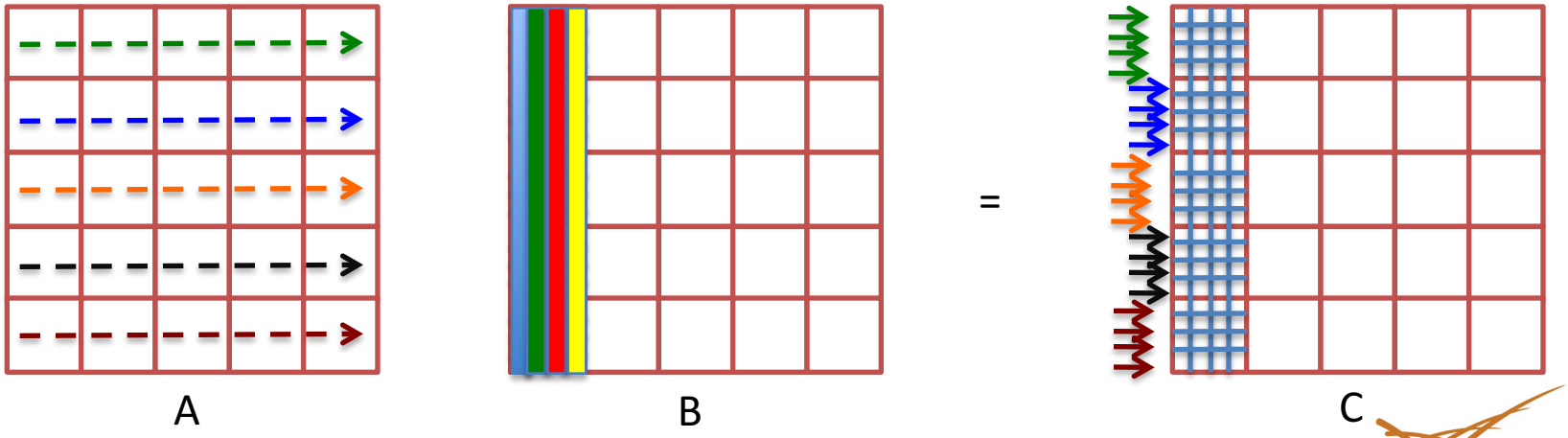
- Strided access for B matrix
- Penalty paid by all threads accessing data
- Unnecessary eviction in caches

```
for (i=0; i<N;i++)
  for (i=1; i<N;i++)
    for (k=1; k<N;k++)
      C[i][j]+=A[i][k]*B[k][j];
```

Matrix Multiplication contd…

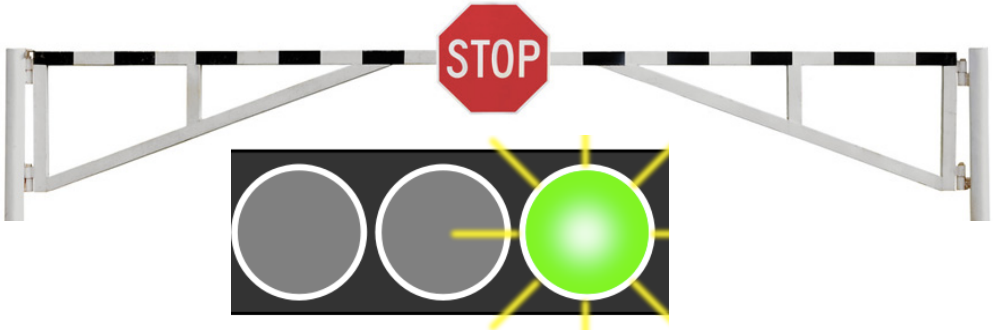
- Solution is transpose
- Wait (Sync)
- Use restructured data.
- Benefit has to outweigh overhead for performance

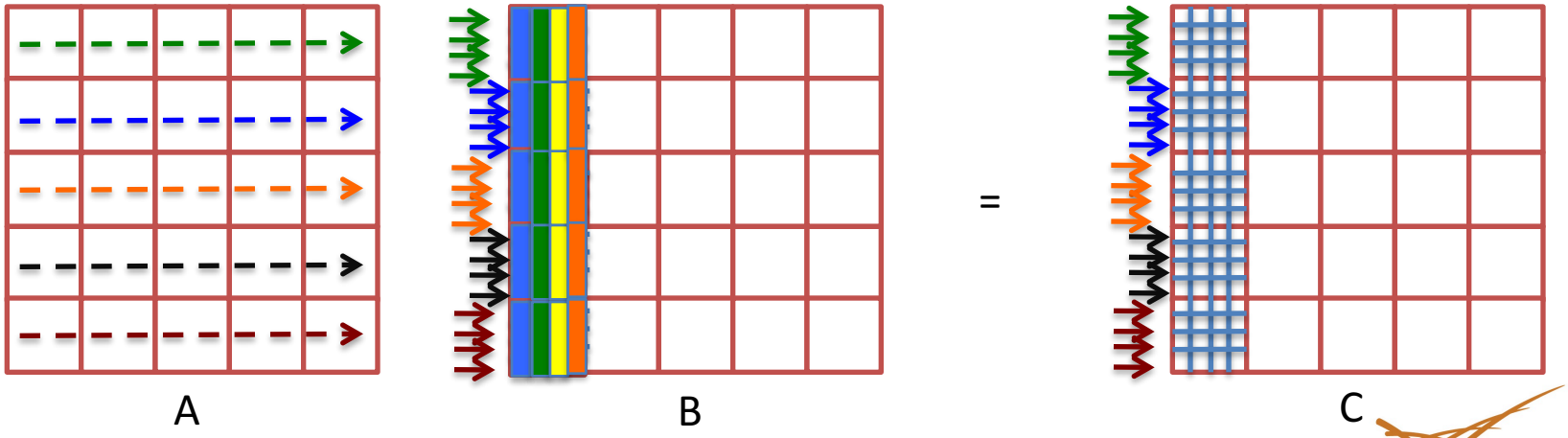




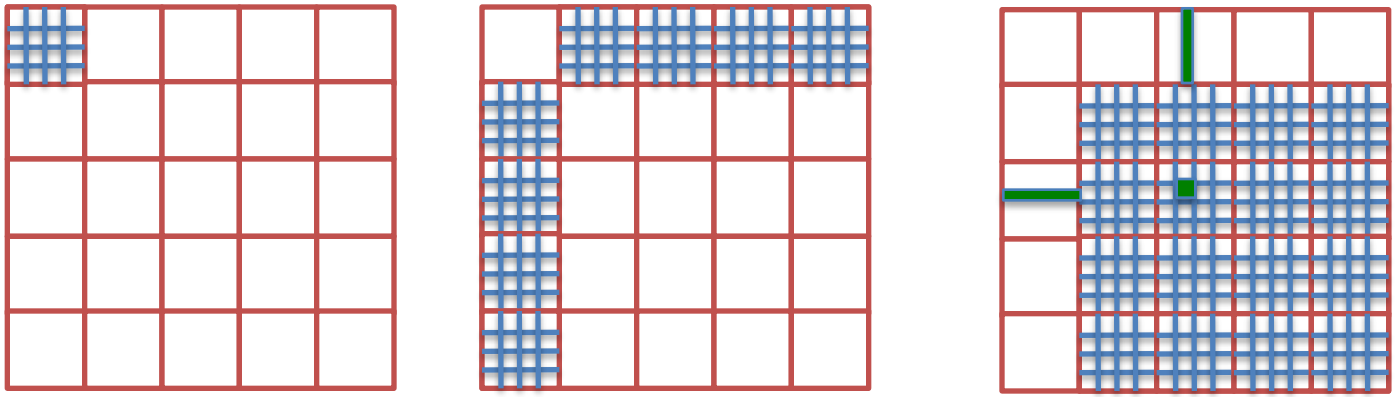
Matrix Multiplication (Our Approach)

- Reduce cache conflicts by placing all elements of a tile in contiguous memory.
- Use low overhead transformation technique
- Parallel data movement





Motivational Example: LU Decomposition

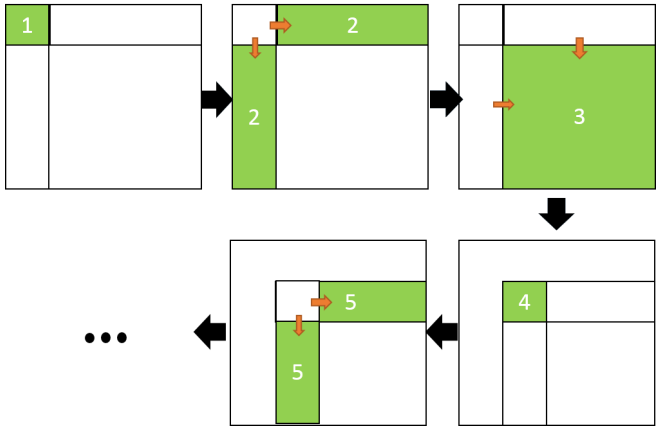


LU Inner Block Calculation (DGEMM)

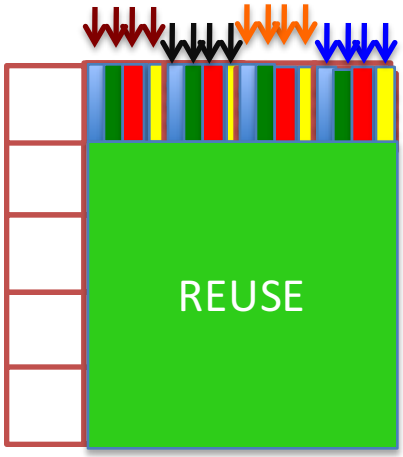
```
f o r ( j j = 1 6 * J ;……) {
   for (kk = 16 * K ;……){
       f o r ( i i = 1 6 * I ; ….) {
              A[ jj ][kk]=A[ jj ][ kk]−A[ ii ][kk]*A[ jj ][ ii ];
       }
    }
}
```

- Strided Access when calculating inner block.

LU Decomposition contd…



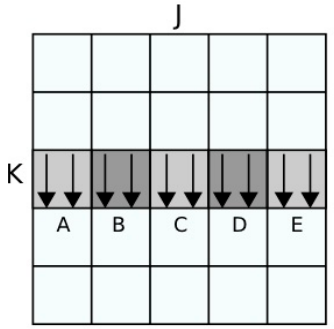
- Plenty of data sharing/reuse during DGEMM
- Opportunity for Data Restructuring
- Restructuring can benefit all participating threads



Restructuring Cases

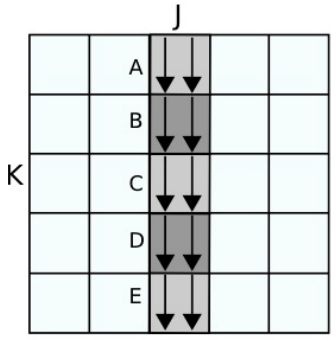
Case 1

- Outer tiles are arranged by row
- Tile elements are accessed by column

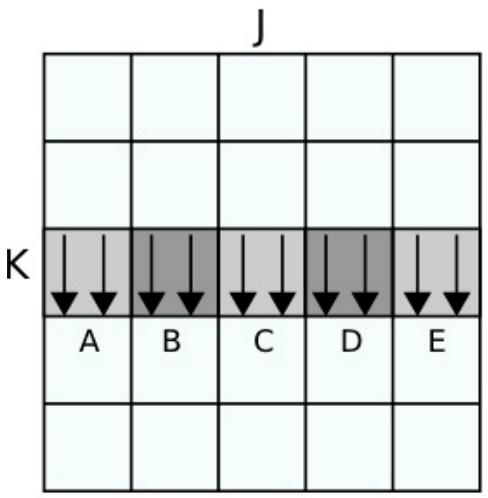


Case 2

- Outer tiles are arranged by column
- Tile elements are accessed by column



Data Restructuring: Case 1



Subtract Lower Bound

TILE INDEX

$K' = 0 \longrightarrow K' = K - K$

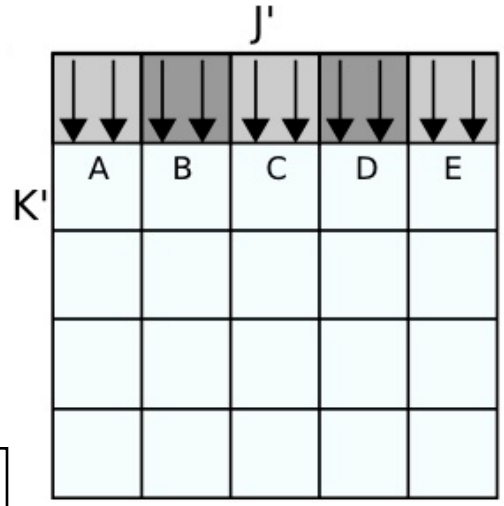
$J' = J$

Translate to reflect element index

ELEMENT INDEX

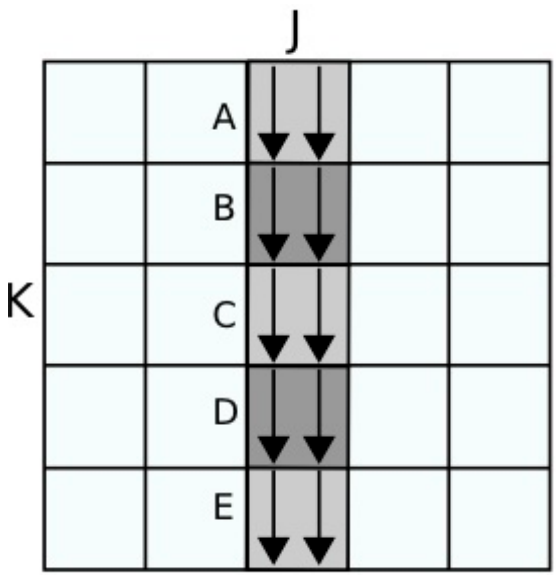
$k' = 0 \longrightarrow k' = k - K*ts_k$

$j' = j$



New Index = Original index +/- Offset

Data Restructuring: Case 2

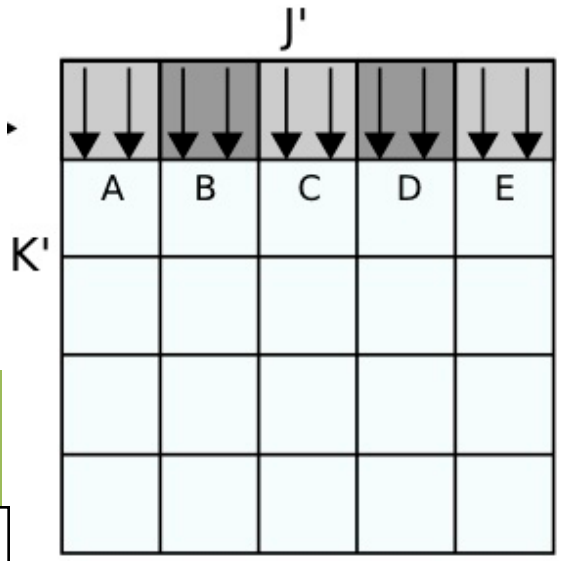


Subtract Lower Bound and Transpose

TILE INDEX

$K' = 0 \longrightarrow K' = K - K$

$J' = K \longrightarrow J' = J + K-J$



Translate to reflect element index

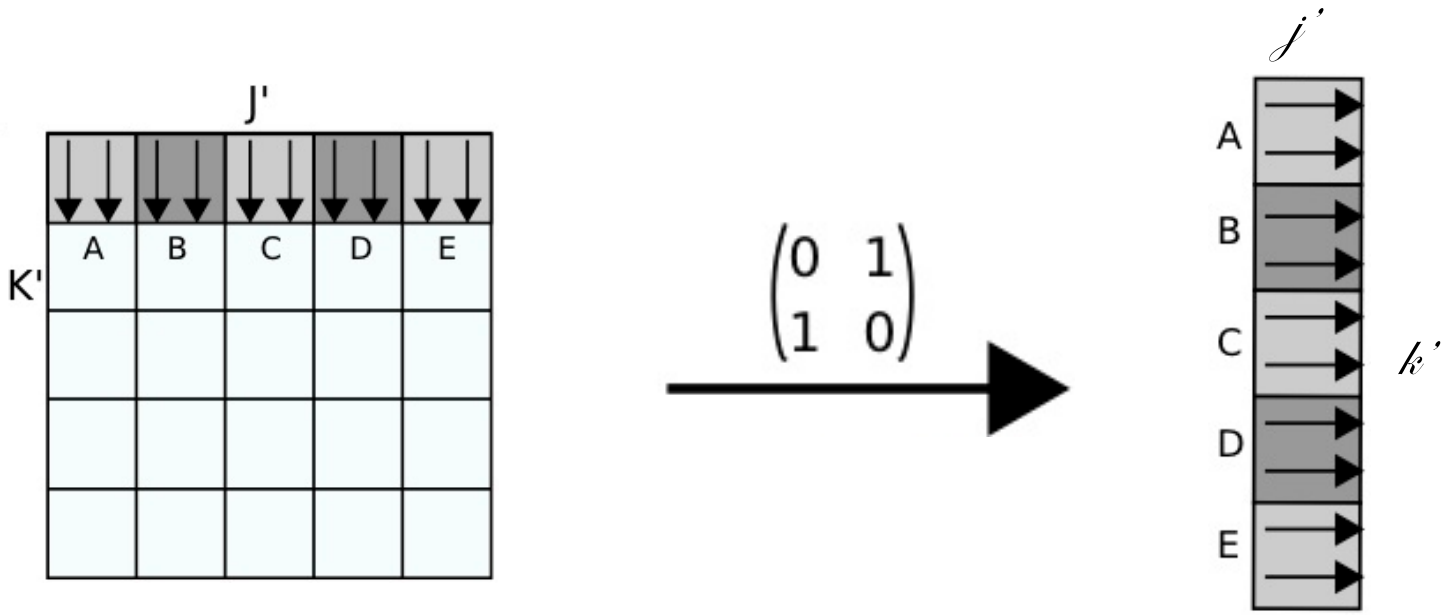
ELEMENT INDEX

$k' = 0 \longrightarrow k' = k - K*ts_k$

$j' = j + (K-J)*ts_j$

New Index = Original index +/- Offset

Data Restructuring contd…



Restructuring space is allocated as a contiguous block in memory.

Experimental Platform

- Tile-Gx36
- 36 Cores
- L1 Cache:
  - 32 KB
  - 2-Way Associative
- L2 Cache:
  - 256 KB
  - 8-Way Associative
- L3 Cache:
  - Shared L2 gives an impression of L3
- Grouping of physical cores

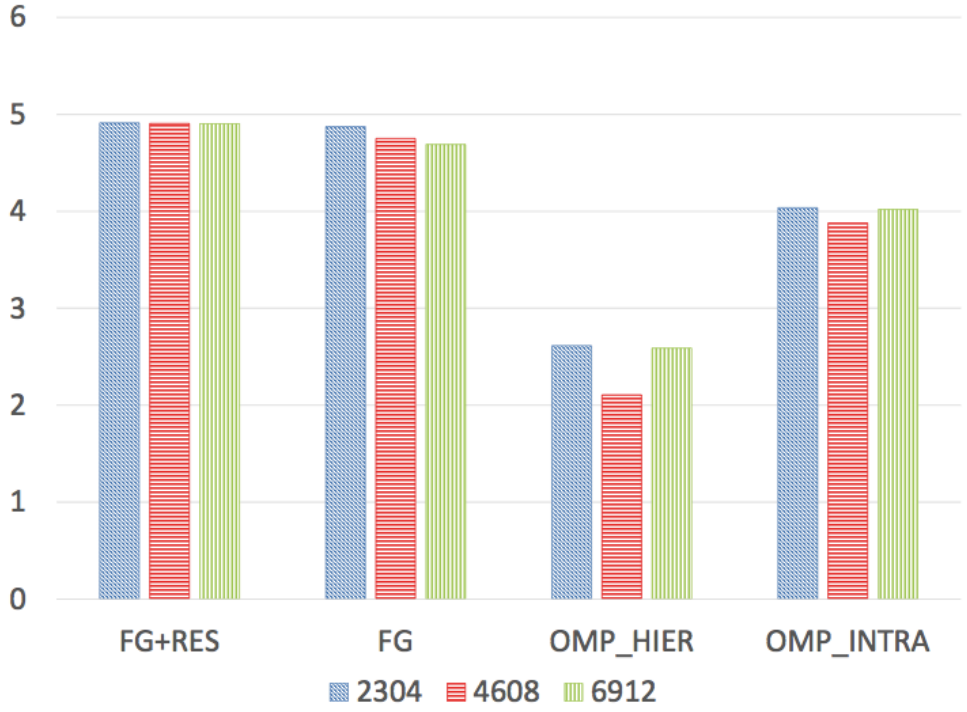
Tile-Gx36 Hardware Counters

Performance Counter	Description	Short-hand
READ_MISS	Level 1 Cache misses for reads	L1RM
WRITE_MISS	Level 1 Cache misses for writes	L1WM
LOCAL_DATA_READ_MISS	Local Level 2/3 Cache misses for reads	L2RM
LOCAL_WRITE_MISS	Local Level 2/3 Cache misses for writes	L2WM
REMOTE_DATA_READ_MISS	Remote Level 2/3 Cache misses for reads	RRM
REMOTE_WRITE_MISS	Remote Level 2/3 Cache misses for writes	RWM

Matrix Multiplication Performance

26.5% over OMP

4.5% over Fine-Grain



FG : Fine-grain
FG+RES : Fine-grain with data restructuring
OMP_HIER: Hierarchically tiled OMP
OMP_INTRA: OMP with intra-tile parallelism

Pacific Northwest NATIONAL LABORATORY

Matrix Multiplication Counters (Normalized)

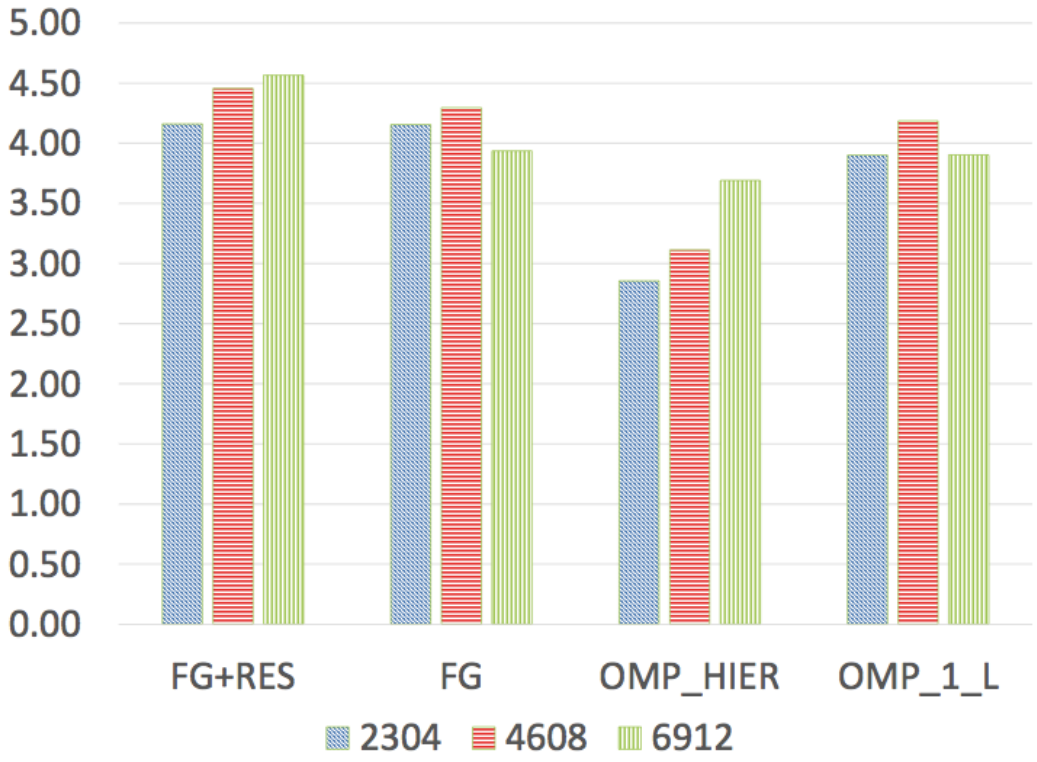
Compared to OMP:
Cache Misses:23.53% to 89.81% (reduced)

Remote Write Misses: 138% (increased)

LU Decomposition Performance

31.4% over OMP

15.9% over Fine-Grain

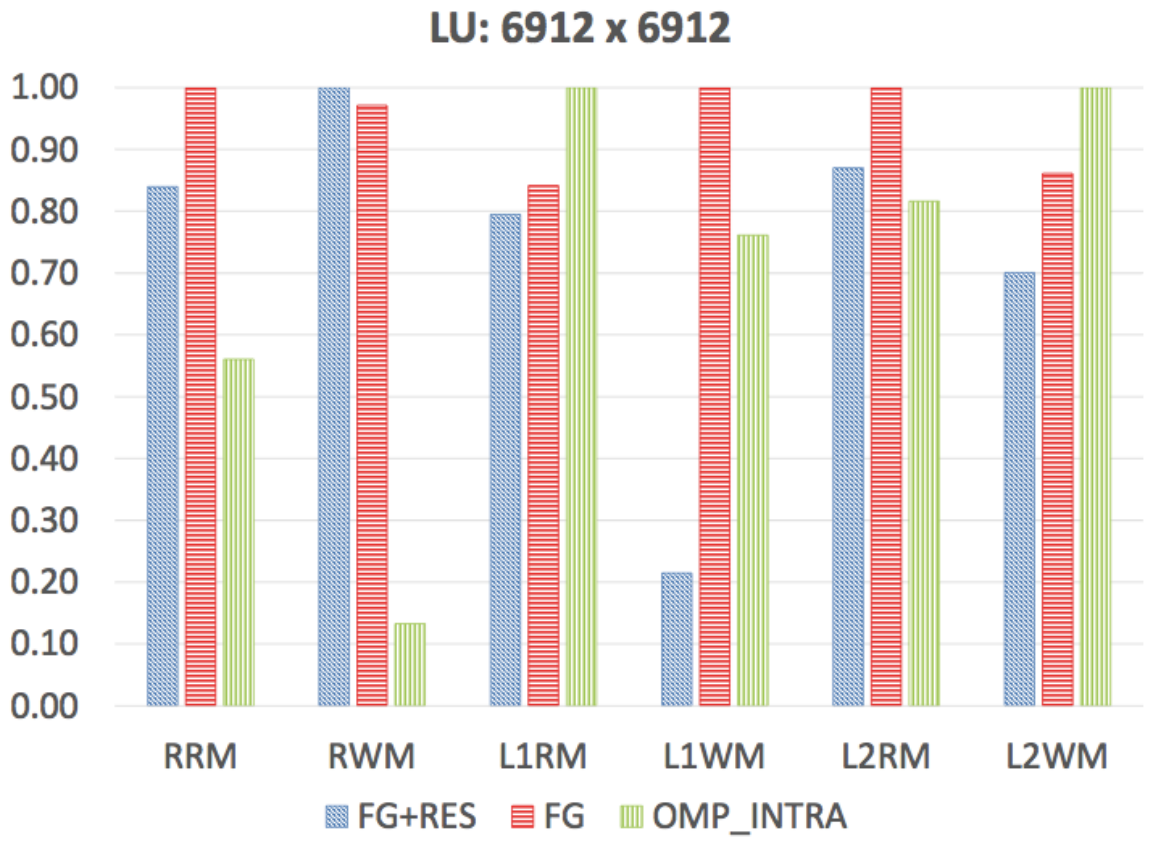


FG : Fine-grain
FG+RES : Fine-grain with data restructuring
OMP_HIER: Hierarchically tiled OMP
OMP_1_L: OMP one level tiling

LU Decomposition Counters

Compared to OMP: Cache Misses:17.8% to 72.79% (reduced)

Remote Write Misses: 267.23% (increased)



Future Work and Conclusion

- Investigate interplay between cache properties and restructuring space.
- Memory Access still bottleneck, Shared resources (bandwidth, memory) limited.
- Collaborative view with thread grouping and restructuring help improve performance.