Colorado State University

Introducing the Sparse Polyhedral Framework (SPF)

Michelle Mills Strout,
Larry Carter, Jeanne Ferrante, and Alan LaMielle

FRACTAL 12/5/09

The Problem

- Sparse computations are important
  - Molecular dynamics simulations, finite element analysis, manipulation of sparse matrices, …
- Sparse computations are SLOW
  - Indirect memory accesses A[B[i]] make compile-time rescheduling impossible and prefetching difficult
- Inspector/executor strategies help, but their application has not been automated

Inspector

Traverses index array

Generates data reordering function σ

Reorder data and updates index array

```
for i=0,7
  ... r[i] ...
for j=0,7
  sigma[j] = ...

for j=0,7
  Z'[sigma[j]]=Z[j]
  r'[j]=sigma[r[j]]
```

Original Code

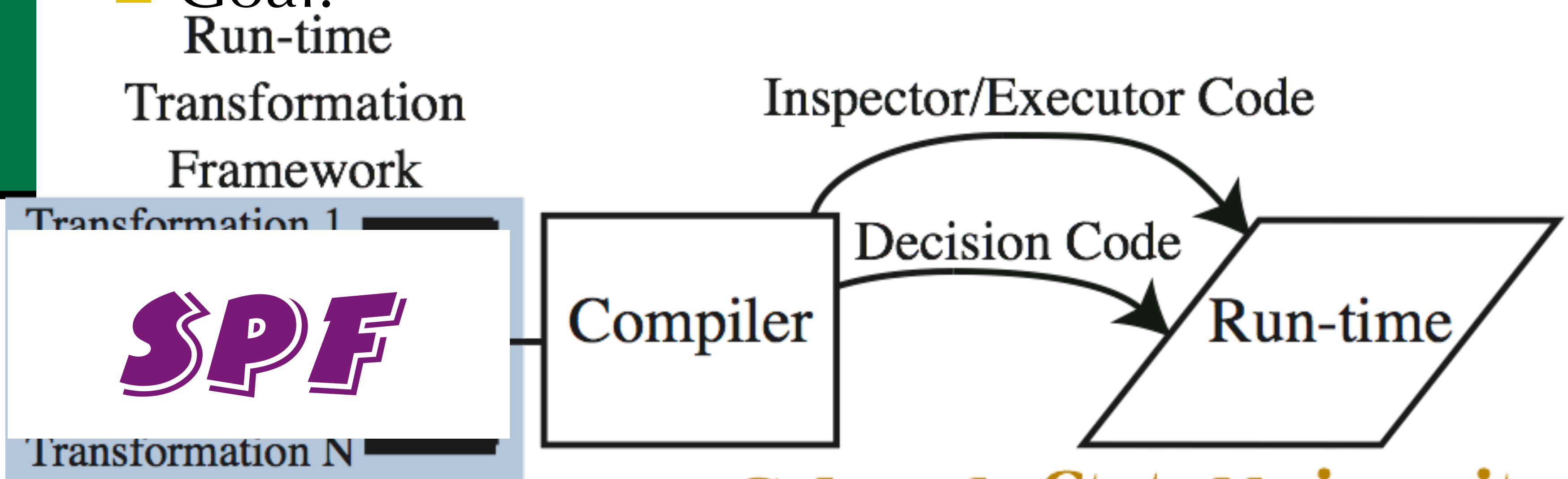
```
for i=0,7
  Y[i] = Z[r[i]]
```

Executor

```
for i=0,7
  Y[i] = Z'[r'[i]]
```

Solution: Compiler/Run-time System for Irregular Applications

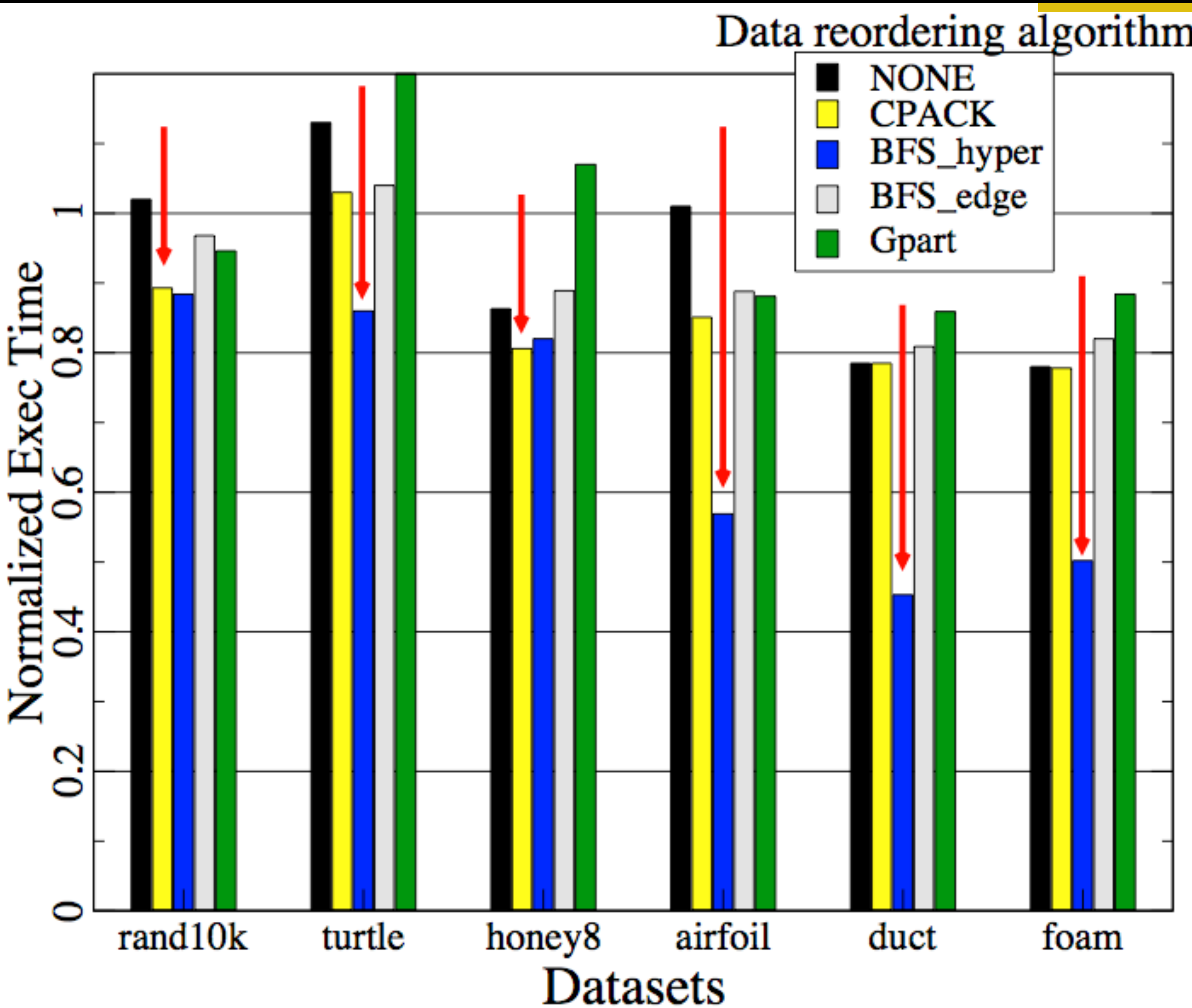
- Challenge: unable to effectively reorder data and computation at compile-time in irregular applications
- Approach: run-time reordering transformations
- Goal:



Example Inspector/Executor Strategies

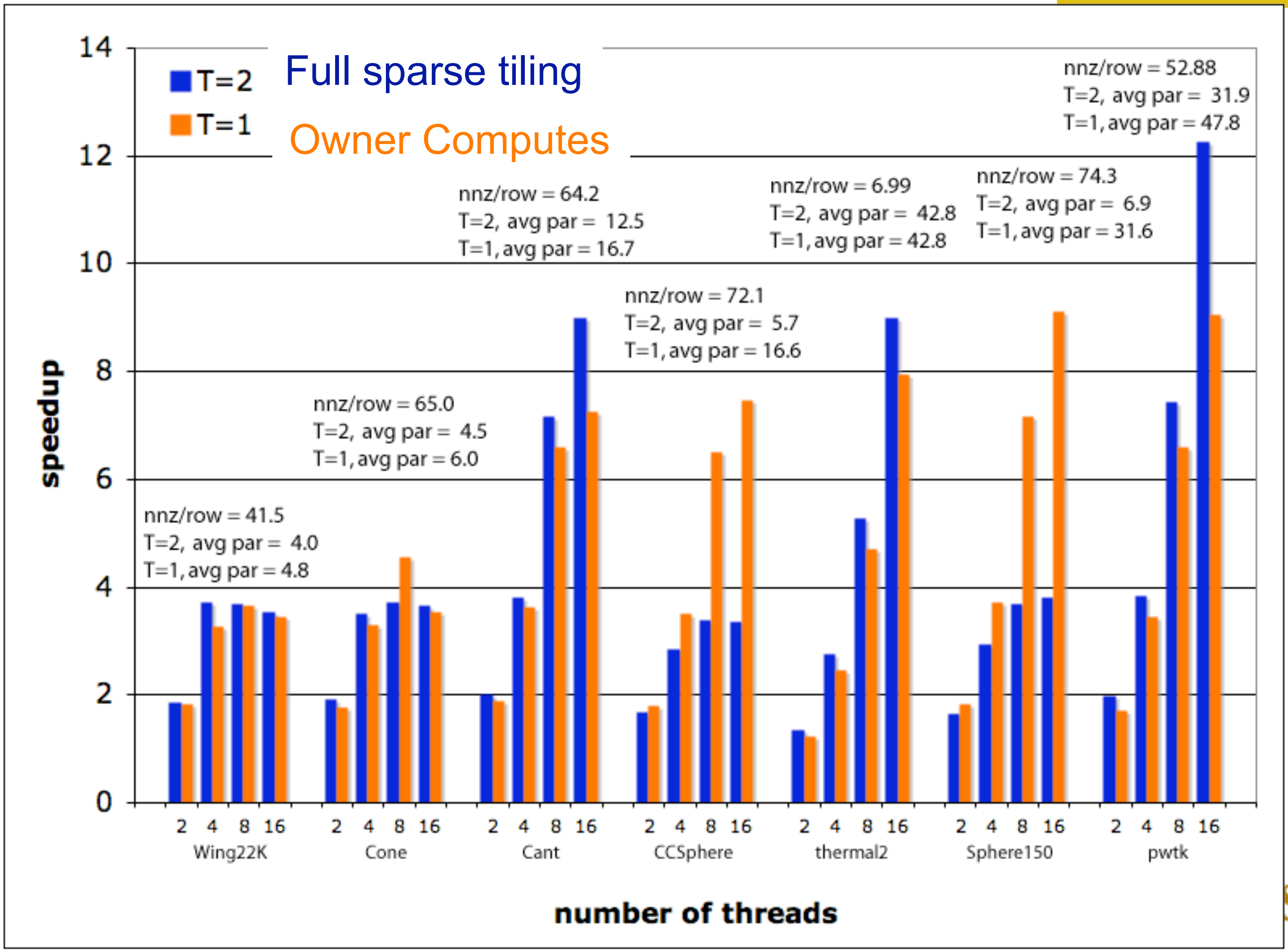
- Gather/scatter parallelization [Saltz et al. 1994]
- Cache blocking [Im and Yelick]
- Irregular cache blocking [Douglas and Rude]
- Full sparse tiling (ICCS 2001)
- Communication avoiding [Demmel et al 2008]
- Run-time data and iteration permutation [Chen and Kennedy 99, Mitchell 99, …]
- ***Compositions*** of the above (PLDI 2003)

Effect of Reorderings on FeasNewt Xeon Pentium 4, 2.2GHz



Parallelization of Gauss-Seidel

IBM Power 5+, 1.9GHz, 1.9MB L2 cache, 36MB L3 cache



sity

Inspector/Executor Strategies show great promise but …

- Only a couple have been automated
- There is library support for some I/E strategies, but matching sparse data structures in non-trivial
- Most I/E strategies are only at the stage of hand-written prototypes
- *How can we automate or semi-automate the application of I/E strategies?*

Loop Transformation Frameworks

- Currently are used in some compilers to …
  - abstractly represent loops, memory accesses, and data deps in loops
  - abstract loop transformations and their effect
  - generate code for transformed loop
- Examples
  - Unimodular framework [Banerjee 90, Wolf & Lam 91]
  - Polyhedral framework [Feautrier, Pugh, Rajopadhye, Cohen, …]

Sparse Polyhedral Framework (SPF)

- Adds uninterpreted function symbols to the polyhedral framework
  - polyhedral includes affine inequality constraints to represent iteration spaces
  - SPF adds constraints such as $x=f(y)$, where f is a function and its input domain and output range are polyhedra
- Code generation for SPF results in inspector and executor code

SPF Example (MOLDYN)

```
for s=1,T
  for i=1,n
    ... = ...Z[i]
  endfor

  for j=1,m
    Z[l[j]] = ...
    Z[r[j]] = ...
  endfor

  for k=1,n
    ... += Z[k]
  endfor
endfor
```

Access Relation for i loop

$$A_{I_0 \to Z_0} = \{[i] \to [i]\}$$

Access Relation for j loop

$$A_{J_0 \to Z_0} = \{[j] \to [i] \quad | \quad l(j) \vee i = r(j)\}$$

Data Dependences between i and j loop

$$D_{I_0 \to J_0} = \{[i] \to [j] \mid (i = l(j)) \vee (i = r(j))\}$$

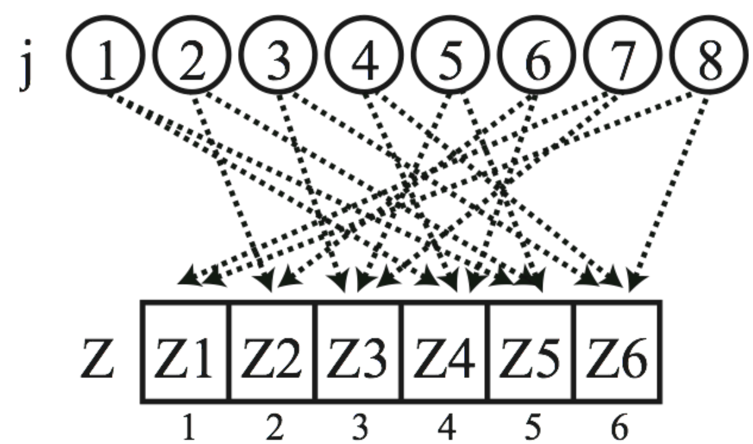
Data Permutation Reordering
(Equations are compile-time abstraction)

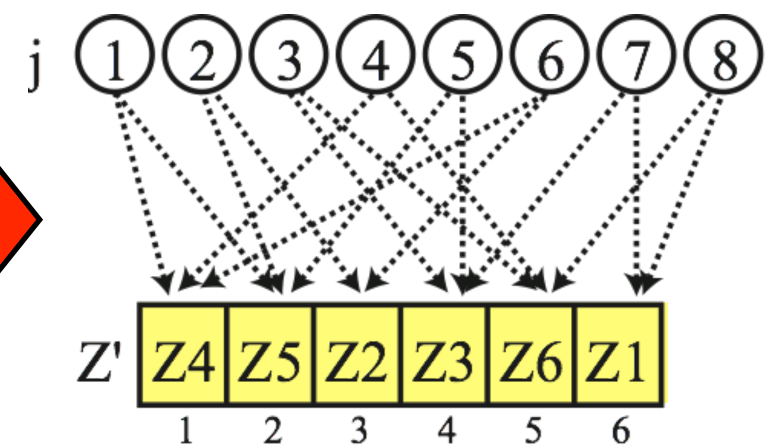
$$R_{Z_0 \to Z_1} = T_{I_0 \to I_1} = \{[i] \to [\sigma(i)]\}$$

CPACK reordering heuristic [Ding & Kennedy 99]

$$A_{J_0 \to Z_0} = \{[j] \to [i] \quad | \quad l(j) \vee i = r(j)\}$$

$$A_{J_0 \to Z_1} = \{[j] \to [i] \quad | \quad i = \sigma(l(j)) \vee i = \sigma(r(j))\}$$



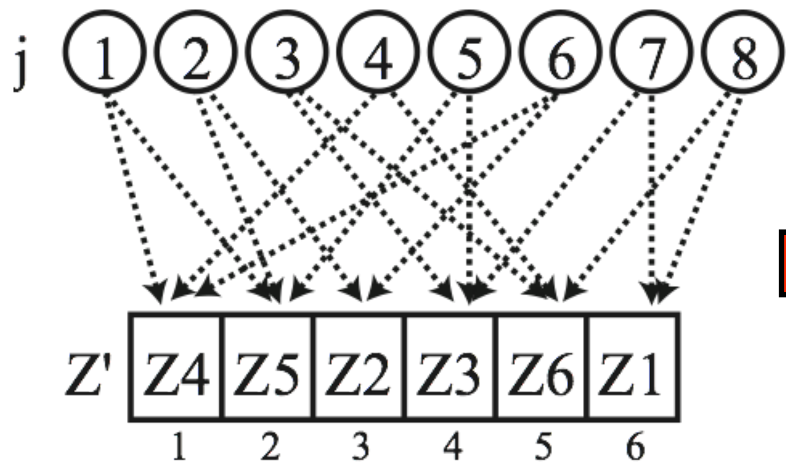


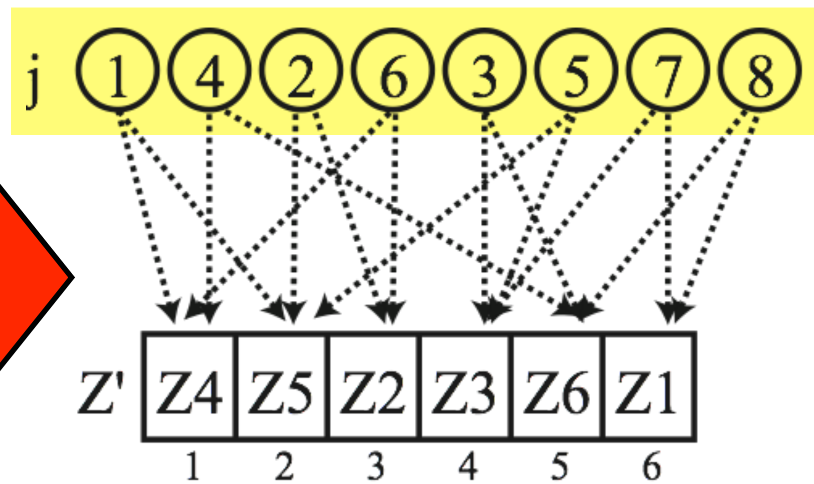
Iteration Permutation Reordering

$$T_{J_0 \to J_1} = \{[j] \to [x] \quad | \quad x = \delta(j)\}$$

$$A_{J_0 \to Z_1} = \{[j] \to [i] \quad | \quad i = \sigma(l(j)) \vee i = \sigma(r(j))\}$$

$$A_{J_1 \to Z_1} = \{[j] \to [i] \quad | \quad i = \sigma(l(\delta^{-1}(j))) \vee i = \sigma(r(\delta^{-1}(j)))\}$$





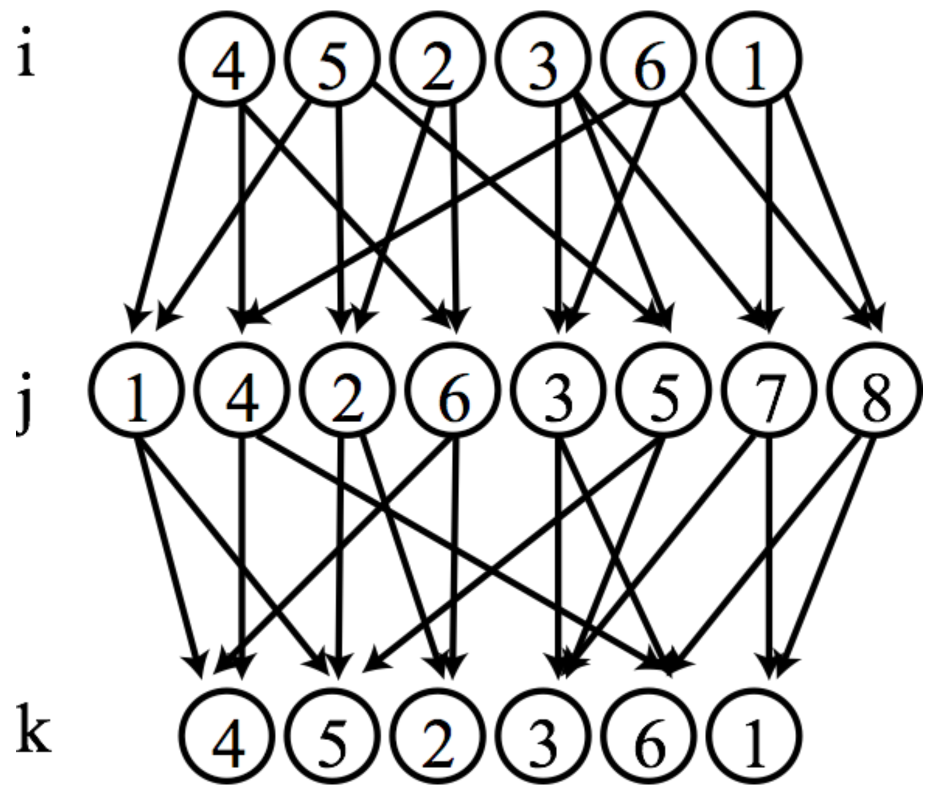
Dependences Between Loops after other transformations

```
for s=1,T
  for i=1,n
    ... = ...Z[i]
  endfor

  for j=1,m
    Z[l[j]] = ...
    Z[r[j]] = ...
  endfor

  for k=1,n
    ... += Z[k]
  endfor
endfor
```

$$D_{I_1 \to J_1} = \{[0,i] \to [1,j] \quad | \quad i = \sigma(l(\delta^{-1}(j))) \\ \vee i = \sigma(r(\delta^{-1}(j)))\}$$

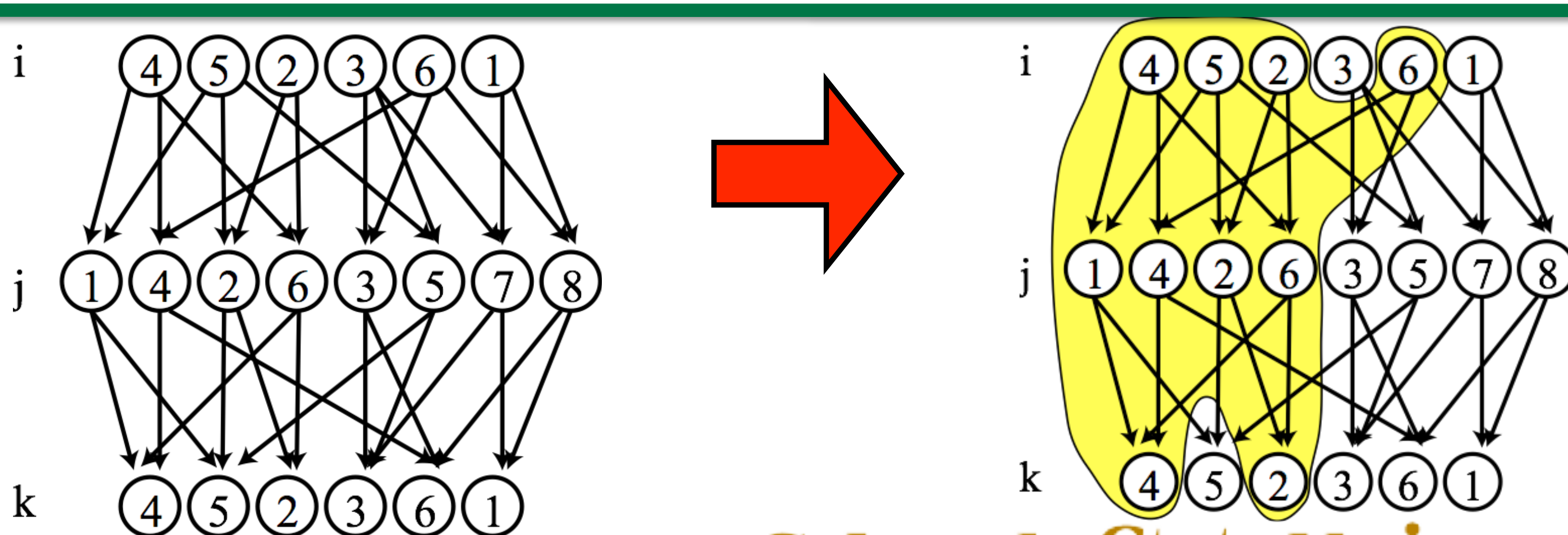


Full Sparse Tiling (FST)

$$
\begin{aligned}
T_{F_1 \rightarrow F_2} &= \{[s,0,i] \rightarrow [s,0,t,0,i] \mid t = \Theta(0,i)\} \\
&\cup \{[s,1,i] \rightarrow [s,0,t,1,j] \mid t = \Theta(1,j)\} \cdots
\end{aligned}
$$

$$F_1 = \{[s,0,t,0,i]\} \cup \{[s,0,t,1,j]\} \cup \{[s,0,t,1,k]\}$$

$$F_2 = \{[s,0,t,0,i] \mid t = \Theta(0,i)\} \cup \{[s,0,t,1,j] \mid t = \Theta(1,j)\} \cdots$$

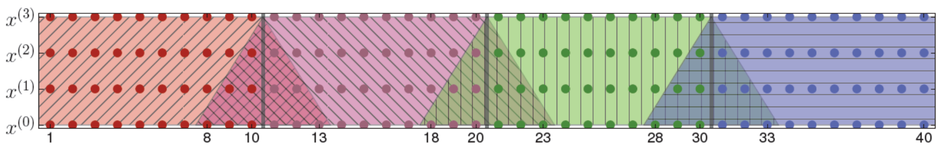


Key Insights in SPF

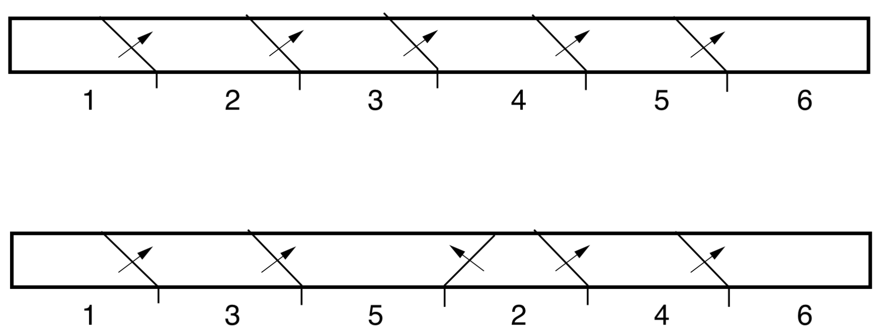
- The inspectors ***traverse*** the data mappings and/ or the data dependences
- We can ***express*** how the data mappings and data dependences will change
- Subsequent inspectors ***traverse the new*** data mappings and data dependences
- Use polyhedral code generator (Cloog) for outer loops and deal with sparsity in inner loops and access relations

Goal: Code Generation for Parameterized Scheduling Strategies

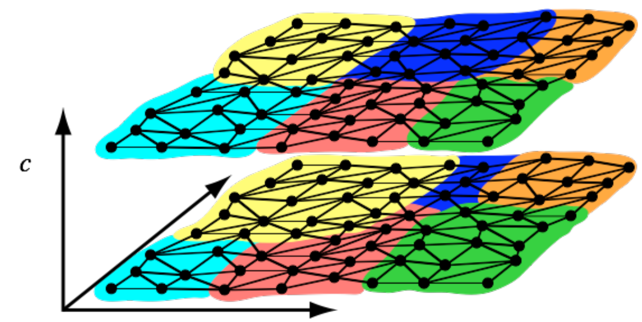
communication avoiding



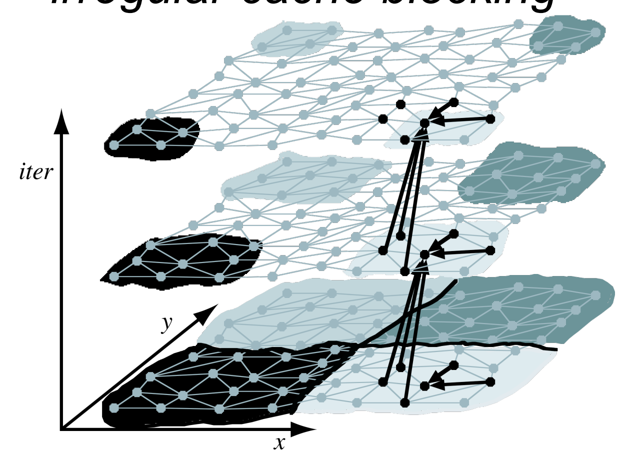
full sparse tiling variants



full sparse tiling



irregular cache blocking



Parameters: sparse tile width and height, graph partitioner, etc.

Conclusions

- Sparse Polyhedral Framework provides abstractions needed to automate performance transformation of irregular/sparse apps
- Inspector/executor code generator (IEGen) is under development
- Long term research: How can we move transformations frameworks out of the compiler and into the programming model?