Algorithmic Generation and Evaluation of Step-code Hierarchies in CnC Applications

Nick Vrvilo
Rice University
Presented at CnC'16

September 27, 2016

Acknowledgements

Habanero Extreme-Scale Software Group:

Kath Knobe, Zoran Budimlić, Vivek Sarkar

Mayo Clinic:

Gary Delp

Purdue:

Chenyang Liu, Milind Kulkarni

Potential Tuning Applications

- Improve data locality
- Coarsen prescription granularity
  - Even-out task bookkeeping footprint over time
  - Improve temporal locality of related tasks
  - Lessen work-stealing overhead
- Automate scoping of item lifetimes

Outline

- Hierarchy-related Properties
- Example – Cholesky
- Generation of the Hierarchy Space
- Application – Locality Tuning
- Conclusion

Definition of Hierarchy

- *Hierarchy:*

  A set of valid granularity choices for the item and step instances in a CnC program

- *Hierarchy space:*

  The union of all possible *hierarchies* for a CnC program

- *Hierarchy slice:*

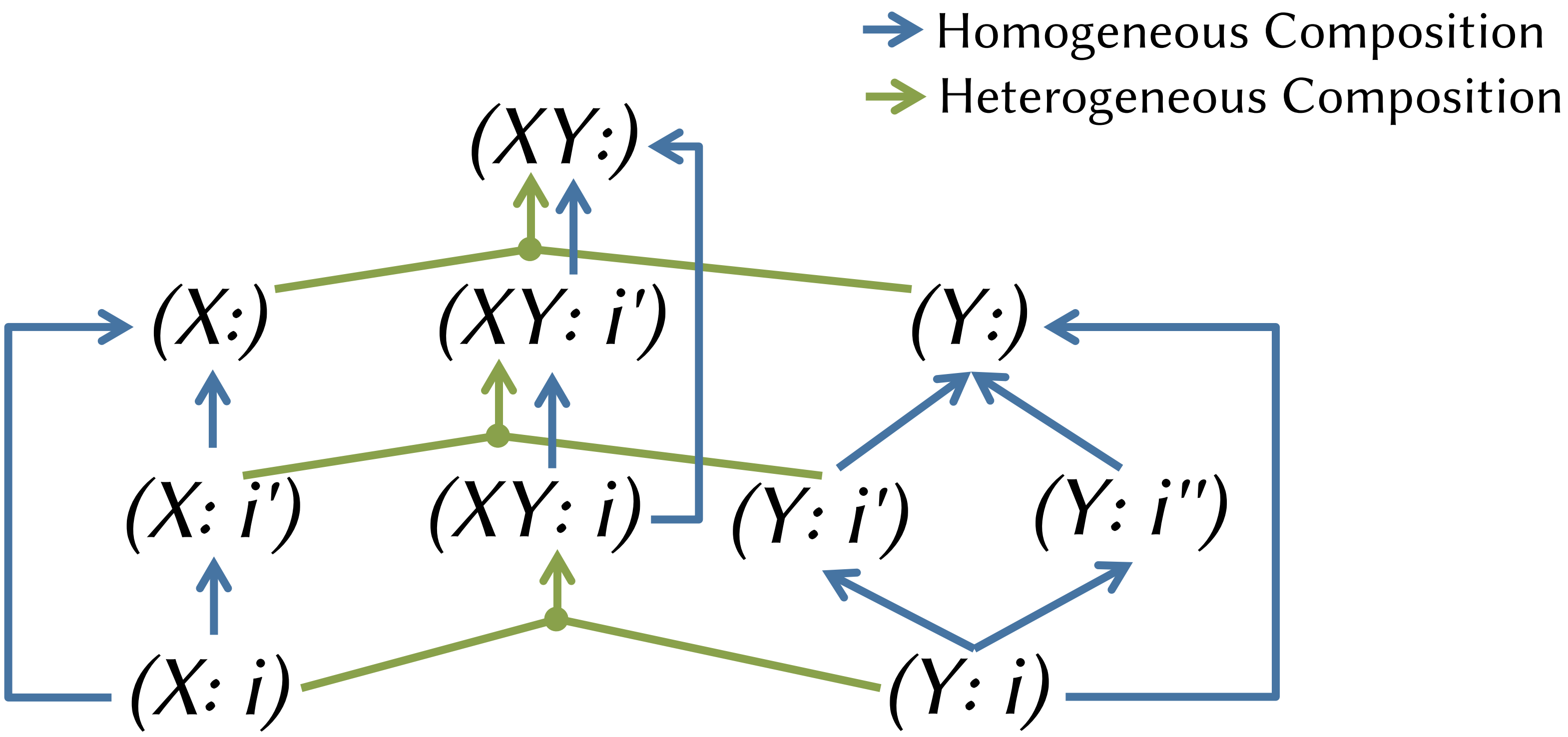
  A single granularity choice for a CnC program

Examples of Granularity Choices

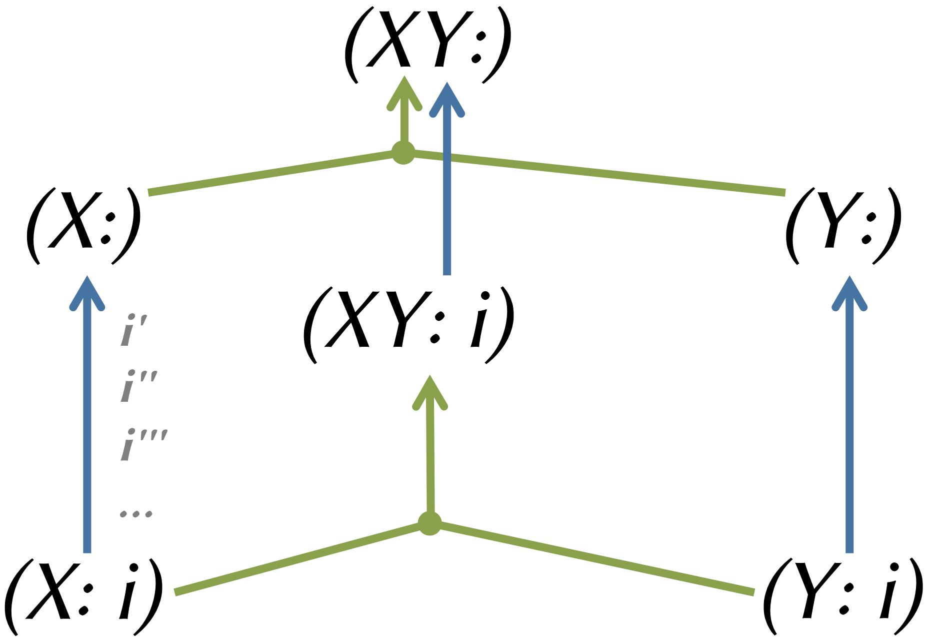
Assume we have two step collections: *(X: i) (Y: i)*

Collections	Description
(X: i') (Y: i)	Tile instances of collection X
(X: i) (Y: i')	Tile instances of collection Y
(X: i') (Y: i')	Tile instances of X and Y symmetrically
(X: i') (Y: i'')	Tile instances of X and Y asymmetrically
(XY: i)	Compose corresponding instances of X and Y
(XY: i')	Compose corresponding tiled instances of X and Y

Example Hierarchy Space



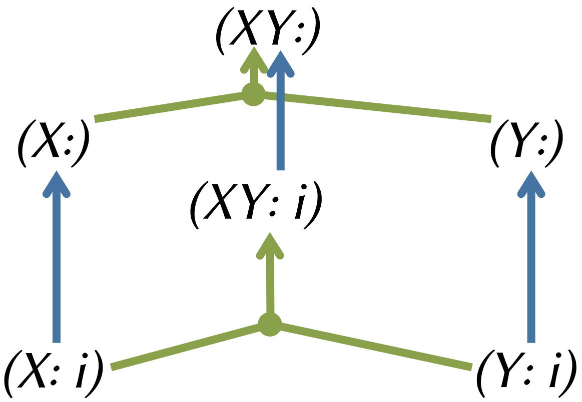
Example Hierarchy Space
(Simplified)



Example Hierarchy Space
(Simplified)

Hierarchy Space Properties

- Constitutes a *join-semilattice*
- The singleton element ⊤ is the composition of the entire graph into a single compute step



Hierarchy Properties

- Subset of the hierarchy space semilattice
- Constitutes a *forest*
- Each finest-grain element is "covered" by exactly tree in the forest
  - R = all tree root nodes in the hierarchy
  - M = minimal elements from the hierarchy space
  - $\forall\ x \in M : \exists!\ y \in R : x \leq y$

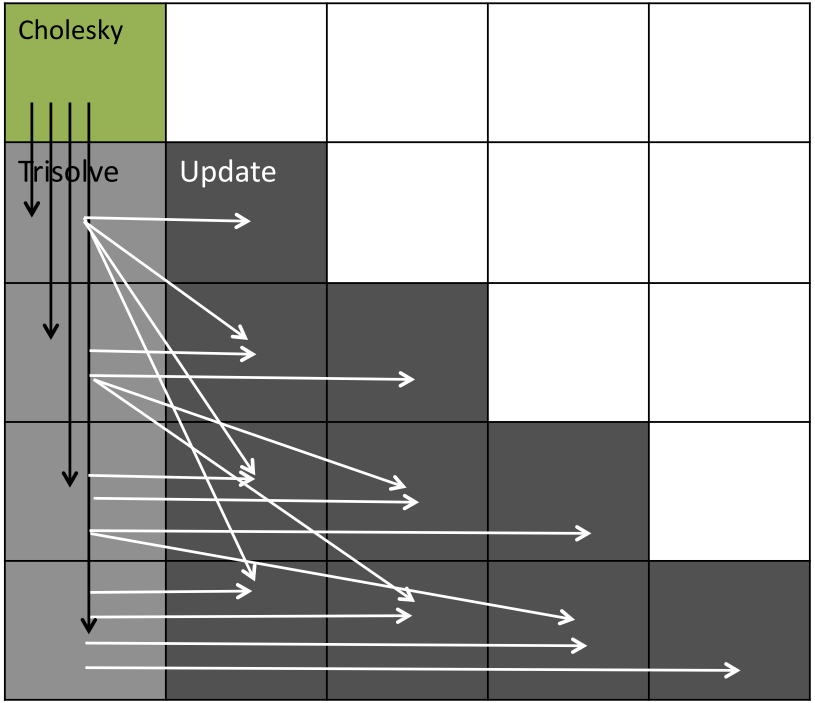
Hierarchy Slice Properties

- A *hierarchy slice* is a *single-level hierarchy*
- Each finest-grain element is “covered” by exactly one element in the hierarchy slice
  - Each element is a “tree root” of a trivial tree in the hierarchy forest
  - All hierarchy properties hold for these elements
- Any two elements in the slice are uncomparable

Outline

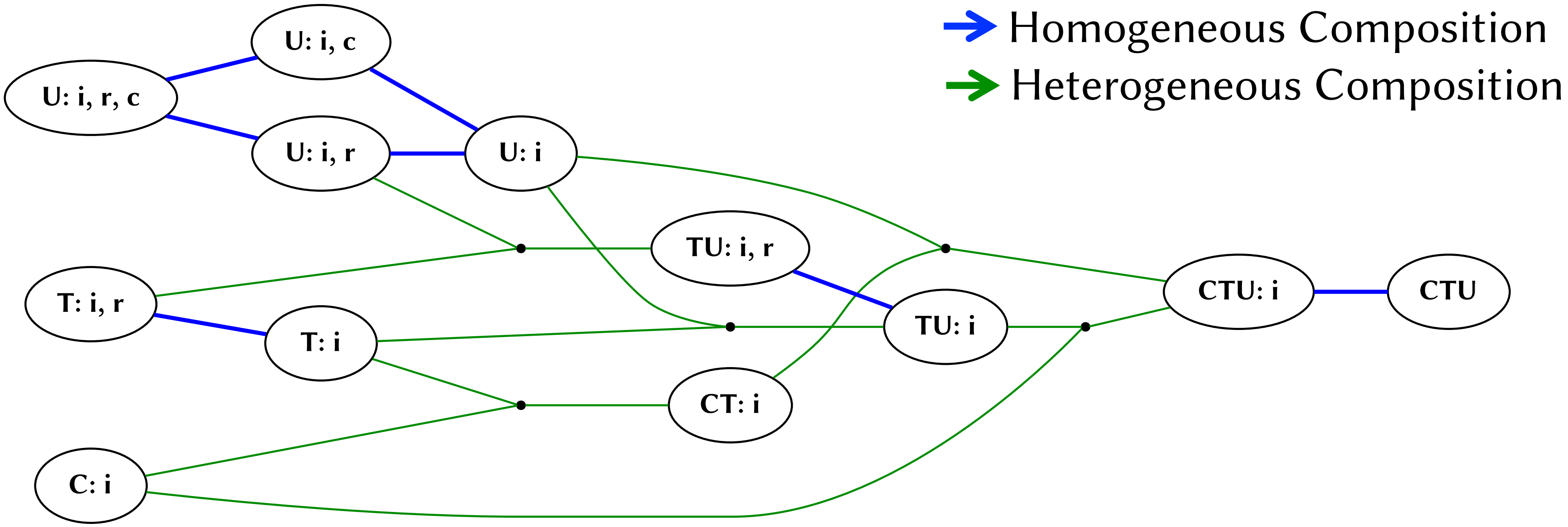
- Hierarchy-related Properties
- **Example – Cholesky**
- Generation of the Hierarchy Space
- Application – Locality Tuning
- Conclusion

CnC Cholesky

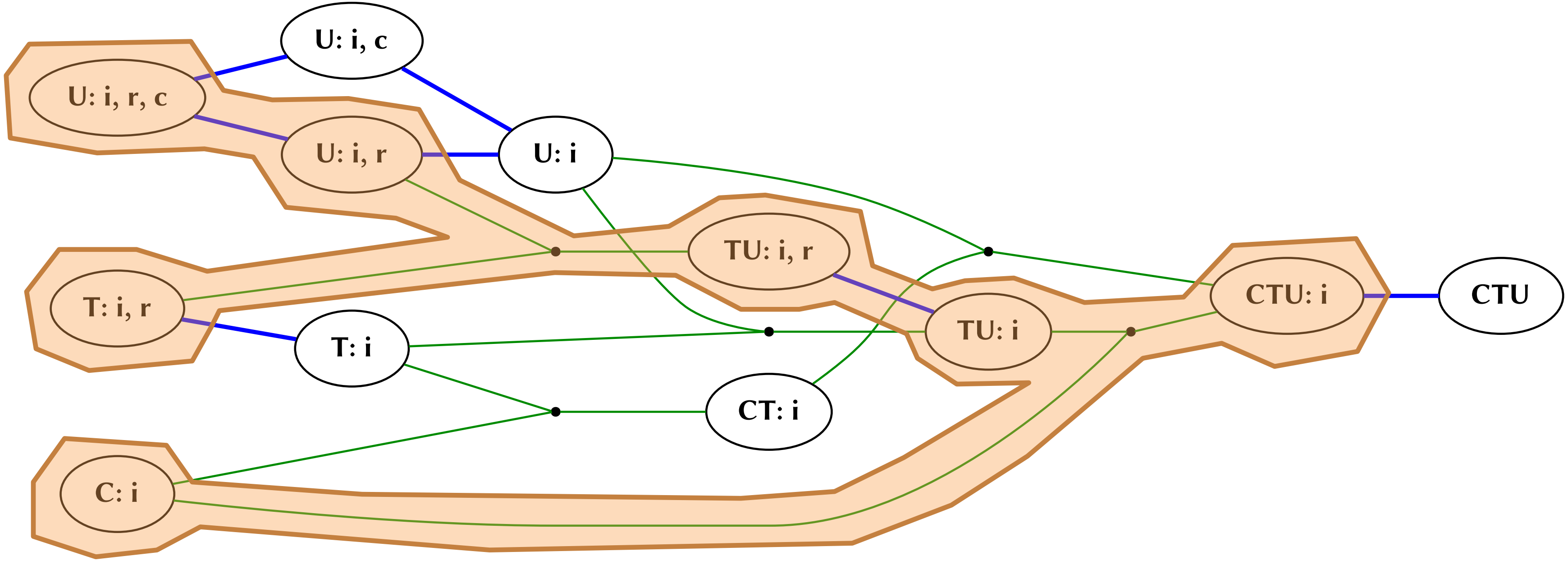



```
[ double MC[]: i ];         // tiles from C step
[ double MT[]: i, r ];      // tiles from T step
[ double MU[]: i, r, c ];   // tile from U step
( C: i )  // serial cholesky step
 <- [ MU: i, i, i ]
 -> [ MC: i+1 ];
( T: i, r )  // trisolve step
 <- [ MU: i, r, i ],
    [ MC: i+1 ]
 -> [ MT: i+1, r ];
( U: i, r, c )  // update step
 <- [ MU: i, r, c ],
    [ MT: i+1, r ] $when(r != c),
    [ MT: i+1, c ]
 -> [ MU: i+1, r, c ];
```

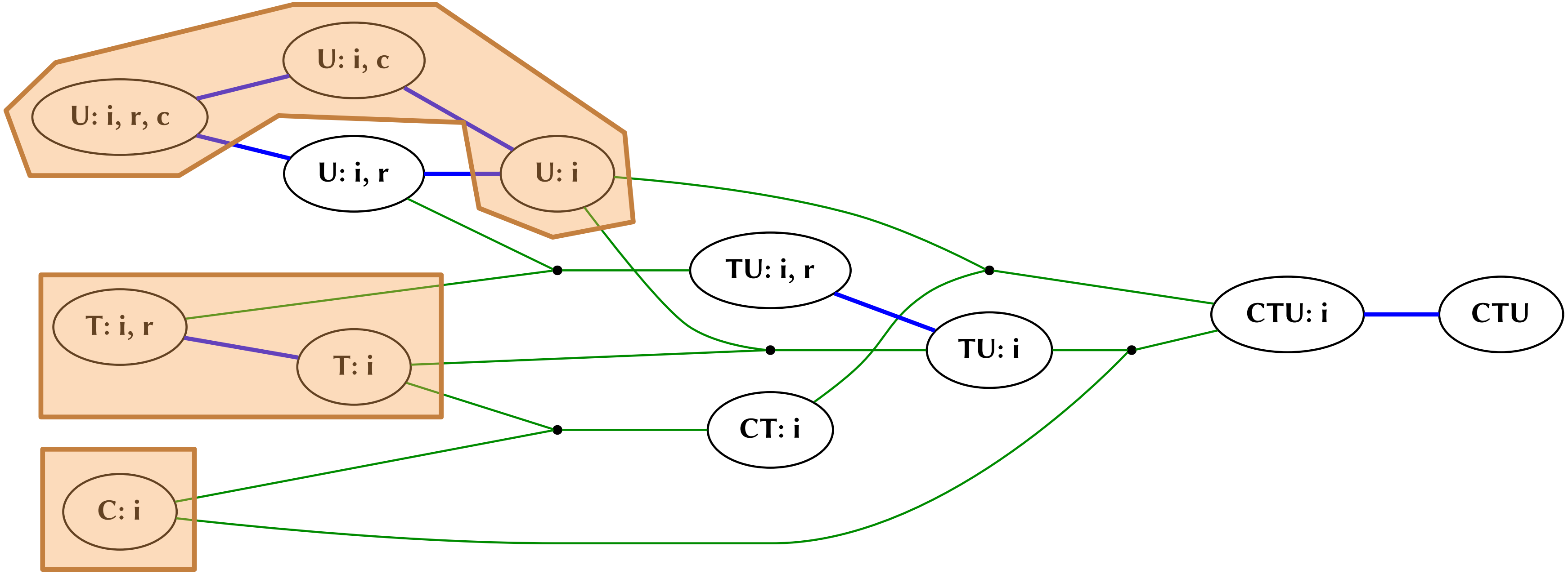
Cholesky: Hierarchy Space



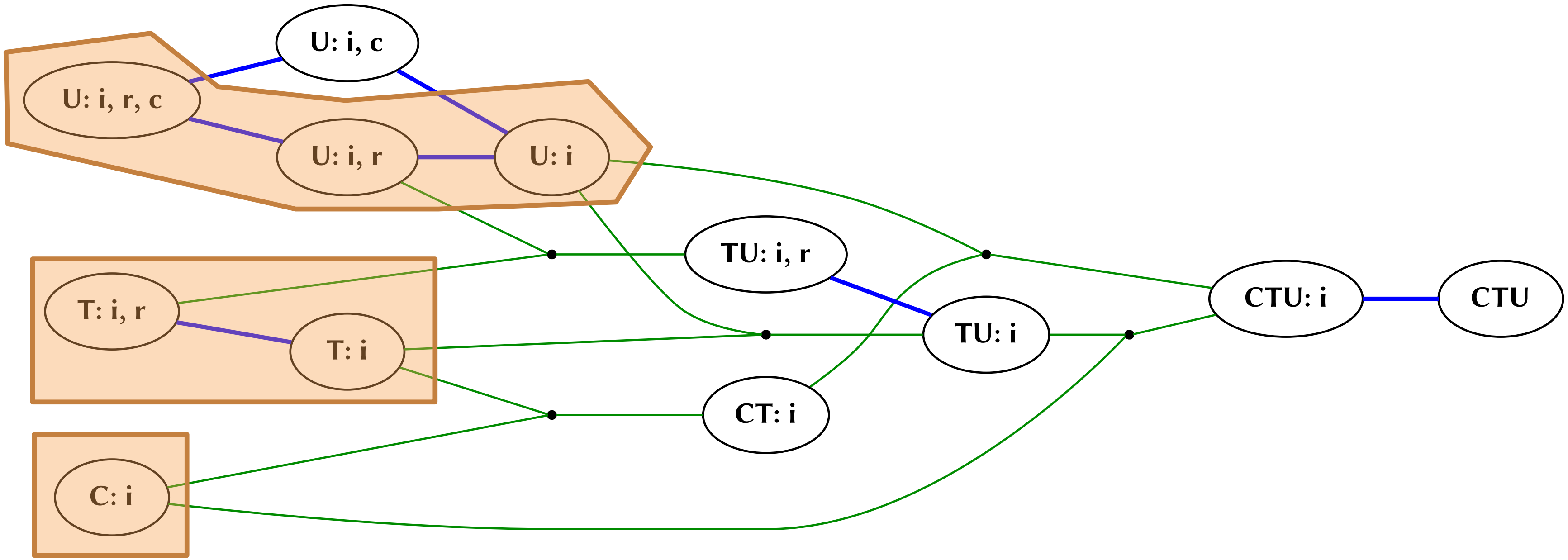
Cholesky: Sample Hierarchies



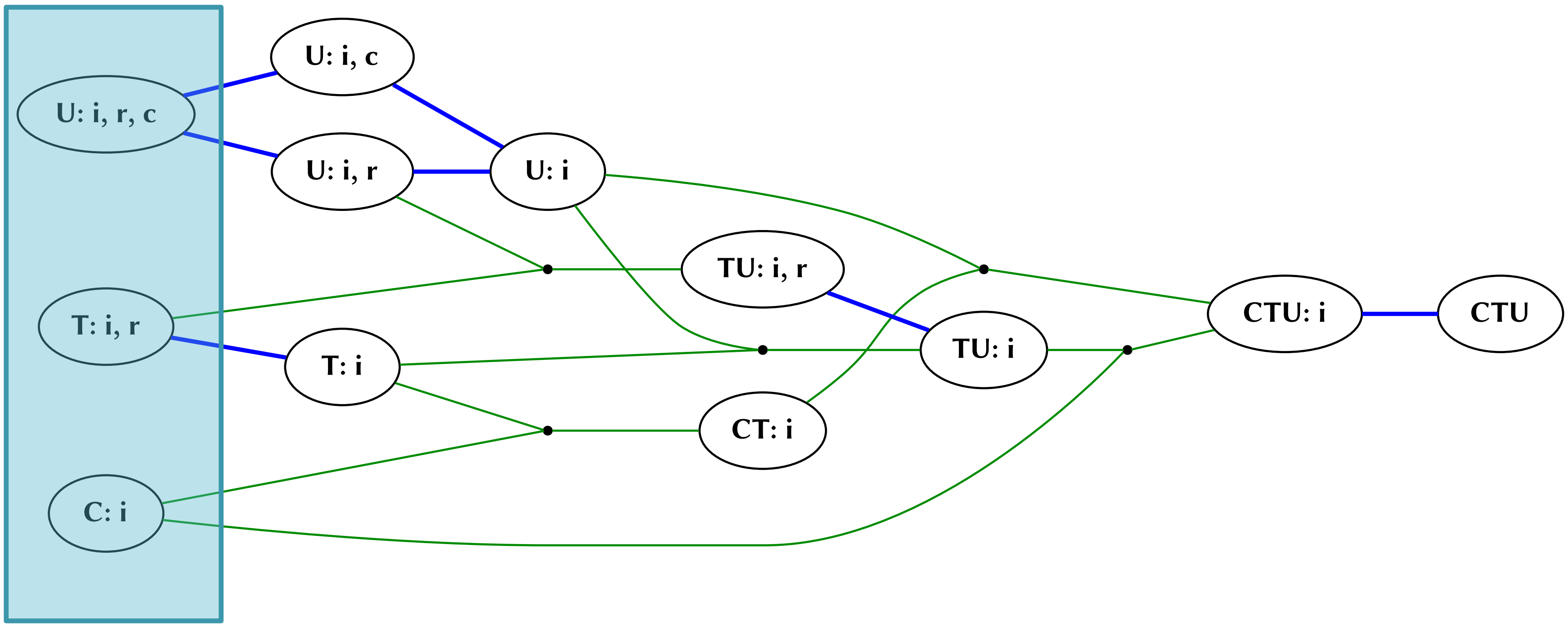
Cholesky: Sample Hierarchies



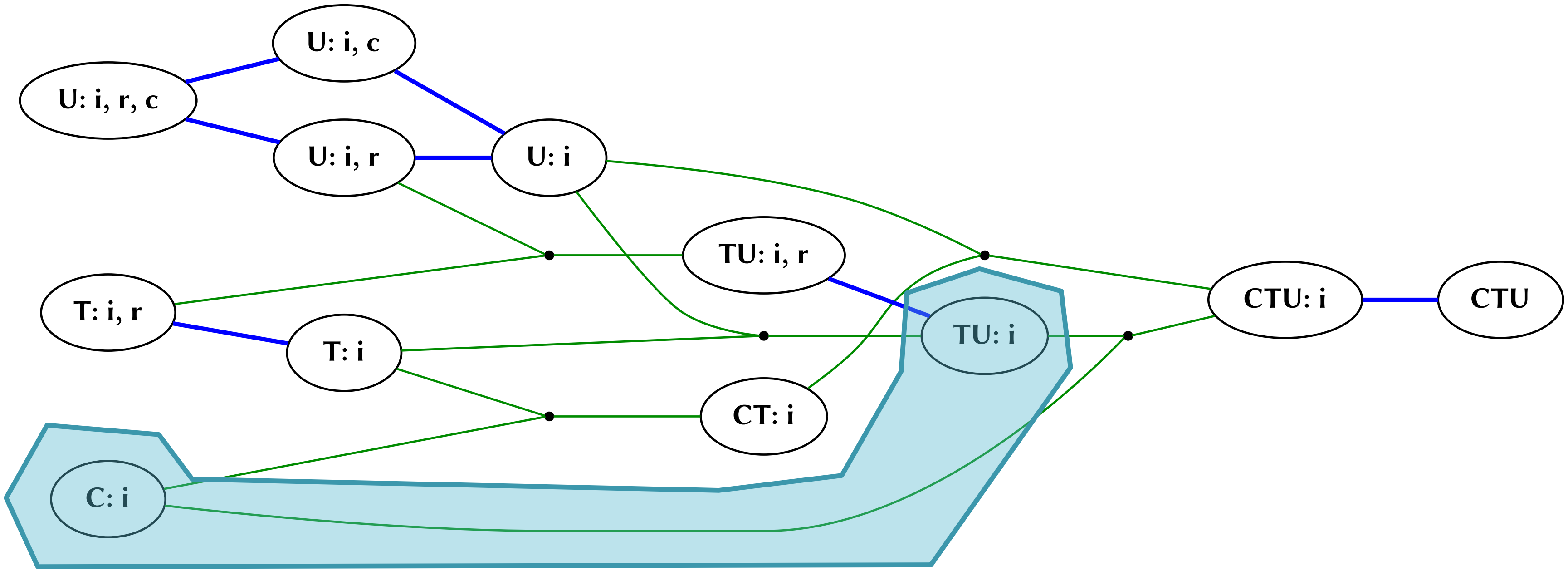
Cholesky: Sample Hierarchies



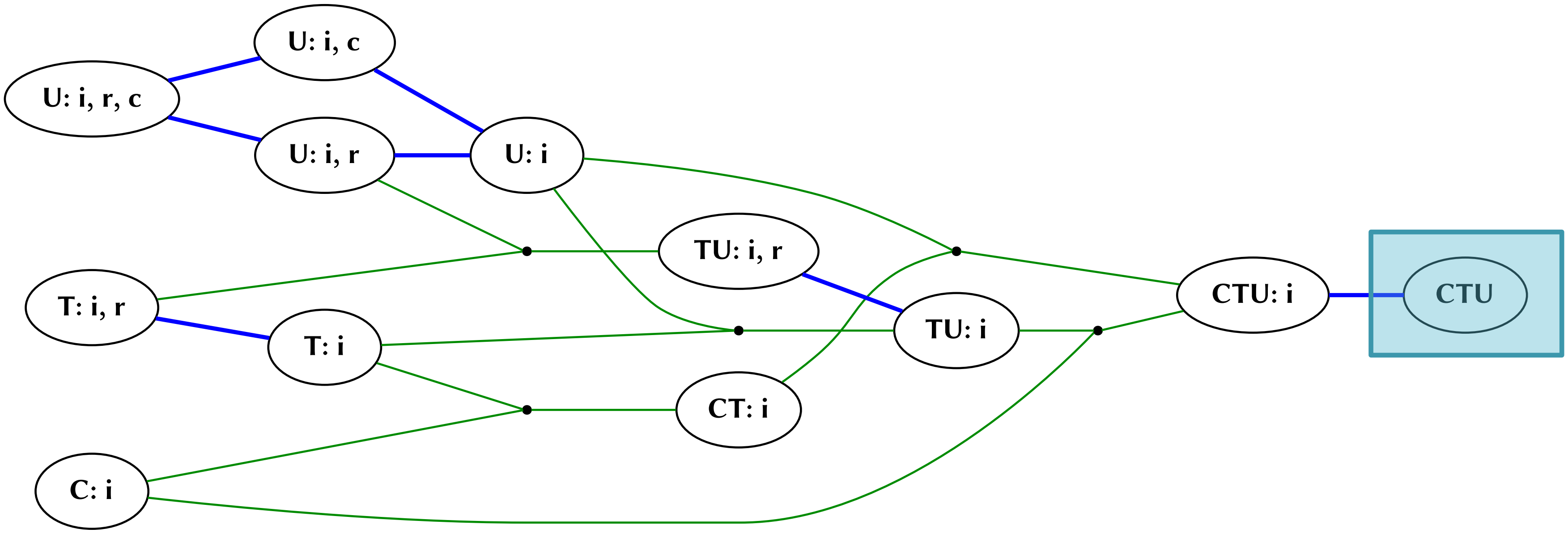
Cholesky: Sample Hierarchy Slices



Cholesky: Sample Hierarchy Slices



Cholesky: Sample Hierarchy Slices



Outline

- Hierarchy-related Properties
- Example – Cholesky
- **Generation of the Hierarchy Space**
- Application – Locality Tuning
- Conclusion

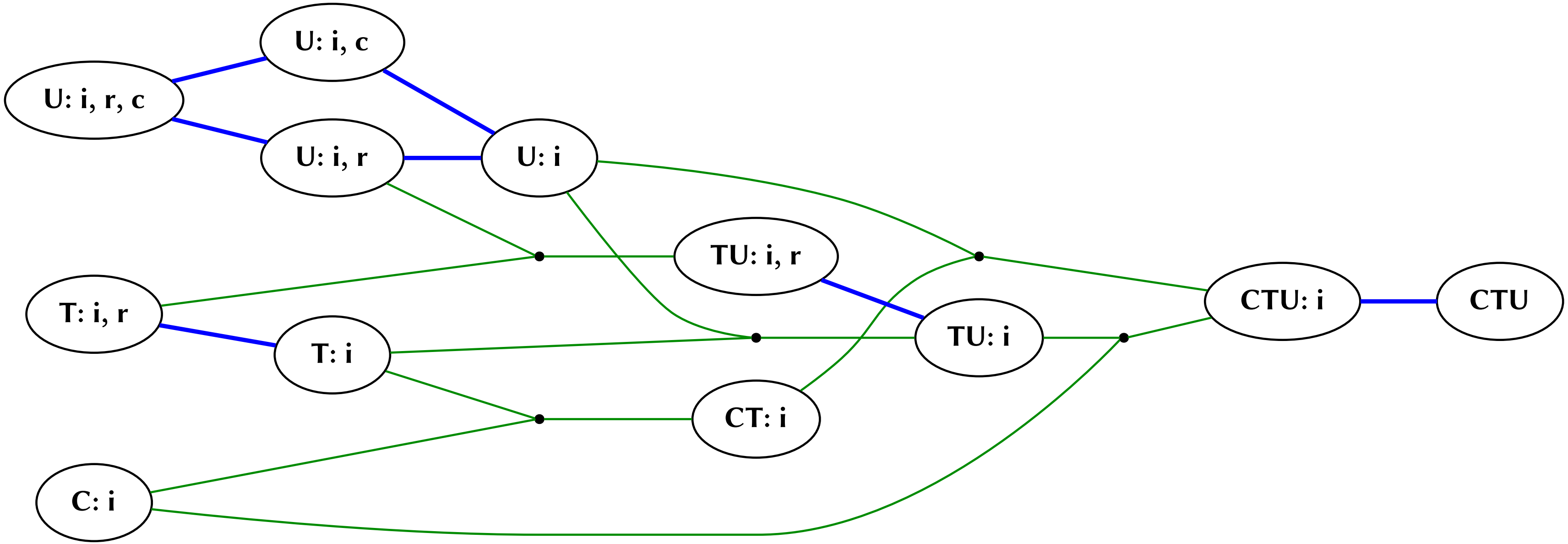
Hierarchy Space Algorithm

```
worklist ← queue(graph.step_collections)
until worklist.is_empty():
    S ← worklist.dequeue()  // pop a step
    for T in S.tag_components():  // homogeneous comps
        if S.can_compose_component(T):
            worklist.enqueue(S.compose_component(T))
    for U in all_step_collections:  // heterogeneous comps
        if S.can_compose_with(U):
            worklist.enqueue(S.compose_with(U))
```

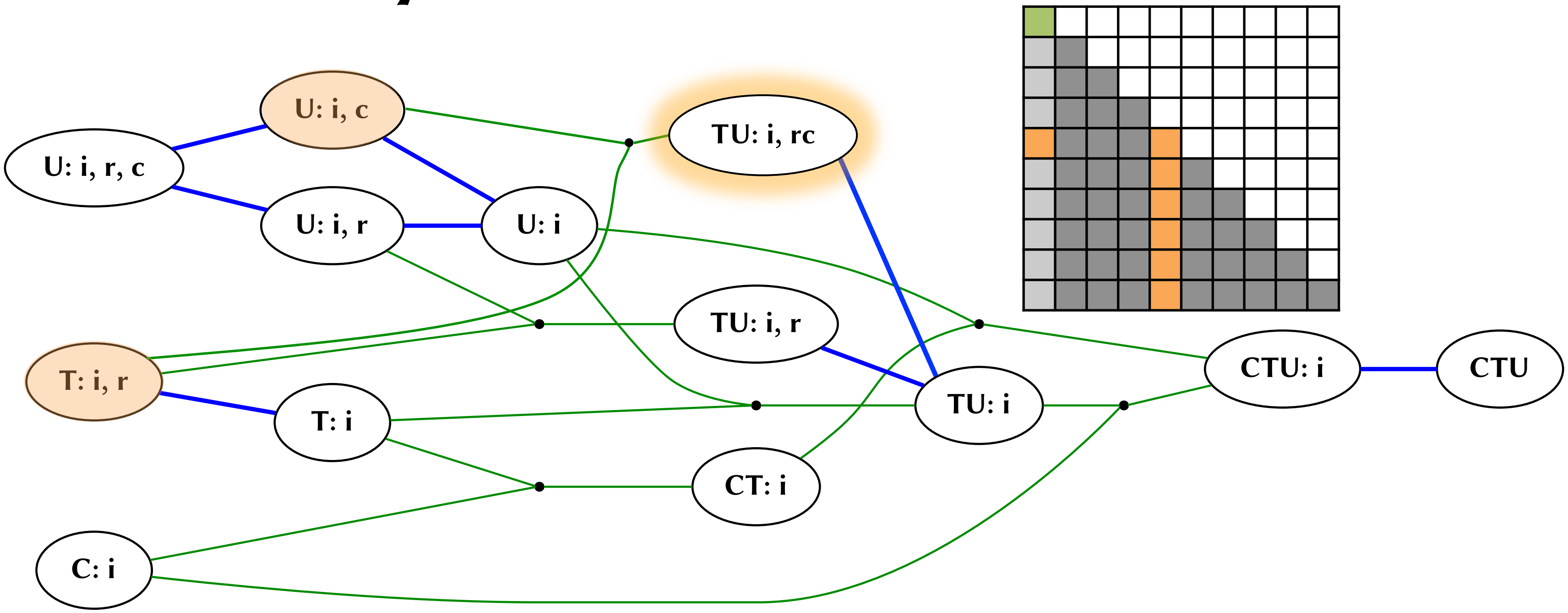
Assumptions & Limitations

- All dependence functions are known
- Only composing (no decomposing)
  - Input graph is the finest grain
  - No “peeling” instances from collections
- Only heterogeneously compose in pairs (can’t simultaneously compose triplets, etc.)

Generating Hierarchies



Cholesky: One More Grain Choice



Finding Hierarchy Slices

```
def find_recursive(nodes = ∅): // finds all slices
    slices ← ∅
    for X in hierarchy_space.nodes():
        if covered(nodes) ∩ covered(X) ≠ ∅: continue
        nodes’ ← nodes ∪ { X }
        if graph.is_covered_by(nodes’):
            slices ← slices ∪ { nodes’ }
        else: slices ← slices ∪ find_recursive(nodes’)
    return slices
```

Outline

- Hierarchy-related Properties
- Example – Cholesky
- Generation of the Hierarchy Space
- **Application – Locality Tuning**
- Conclusion

Evaluation of Distribution Tuning with Cholesky Hierarchy Slices

- Use Cholesky hierarchy slices to determine step distributions across a cluster
- Indirectly choose item distributions based on the placement of the producer
- Evaluate all slices in the hierarchy space

Setup for Evaluation

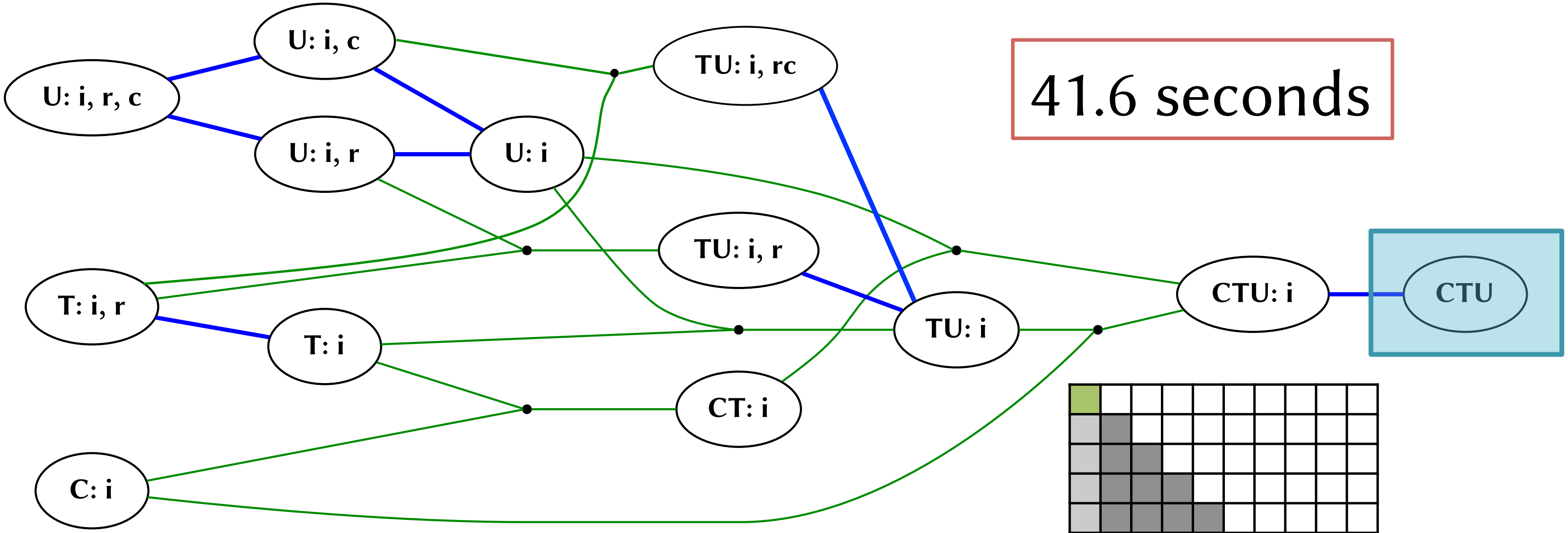
- 8 nodes, 16 cores/node
- 8-core Intel Xeon CPUs @ 2.90GHz
- Habanero CnC Framework
- Intel CnC + Intel MPI
- Cholesky: 8100×8100 matrix, tiles of 50×50

Selection of Hierarchy-based Distribution Results for Cholesky

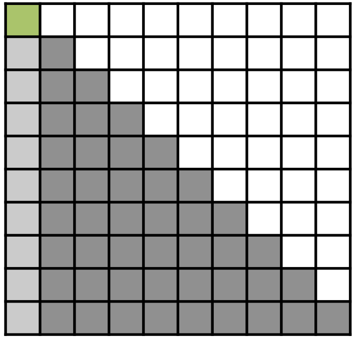
Hierarchy Slice	Run-time*
(CT: i) + (U: i, c)	3.2 seconds
(C: i) + (T: i, r) + (U: i, c)	5.6 seconds
(CT: i) + (U: i, r)	9.0 seconds
(CTU:)	41.6 seconds
(C: i) + (T: i, r) + (U: i)	60.3 seconds

* Averaged over five runs

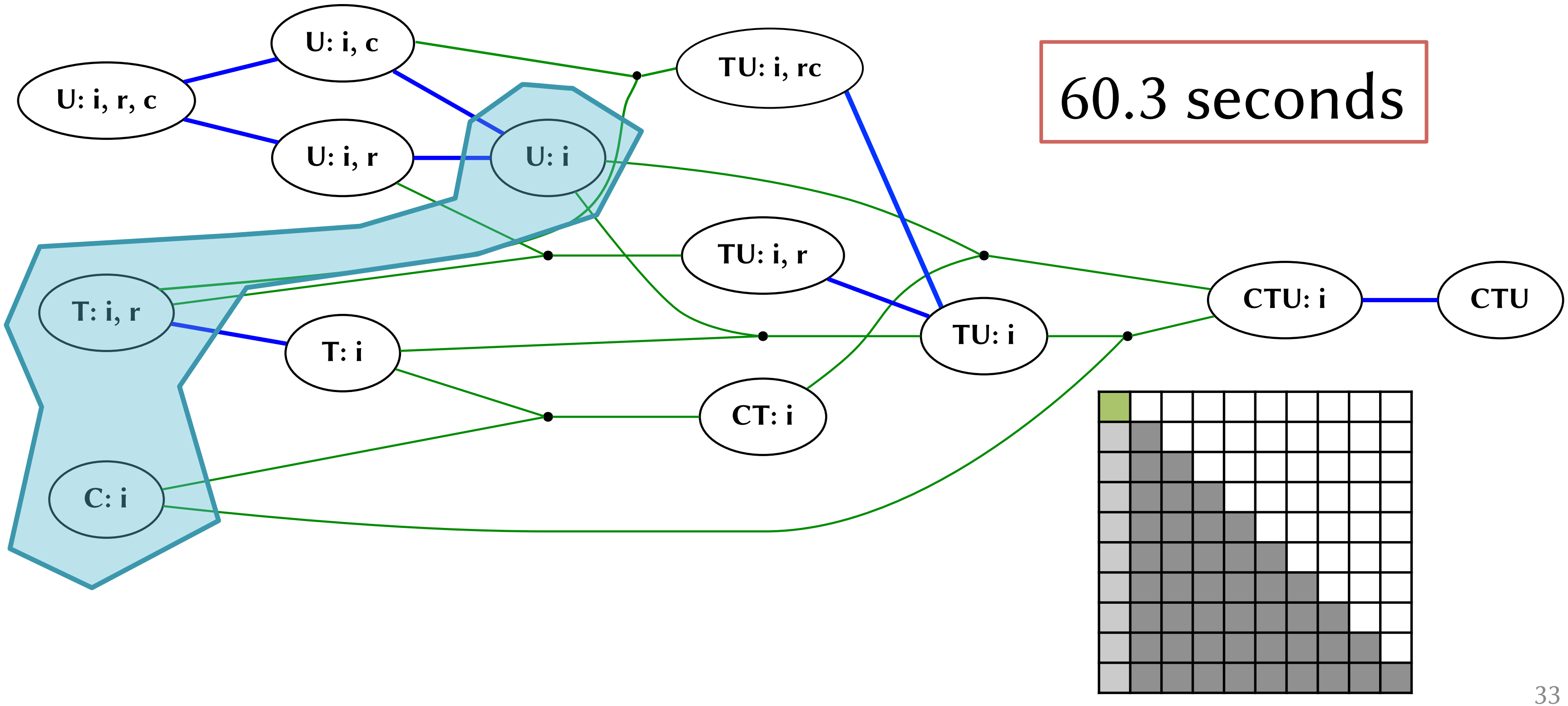
Cholesky: Singleton Slice (Bad)



41.6 seconds

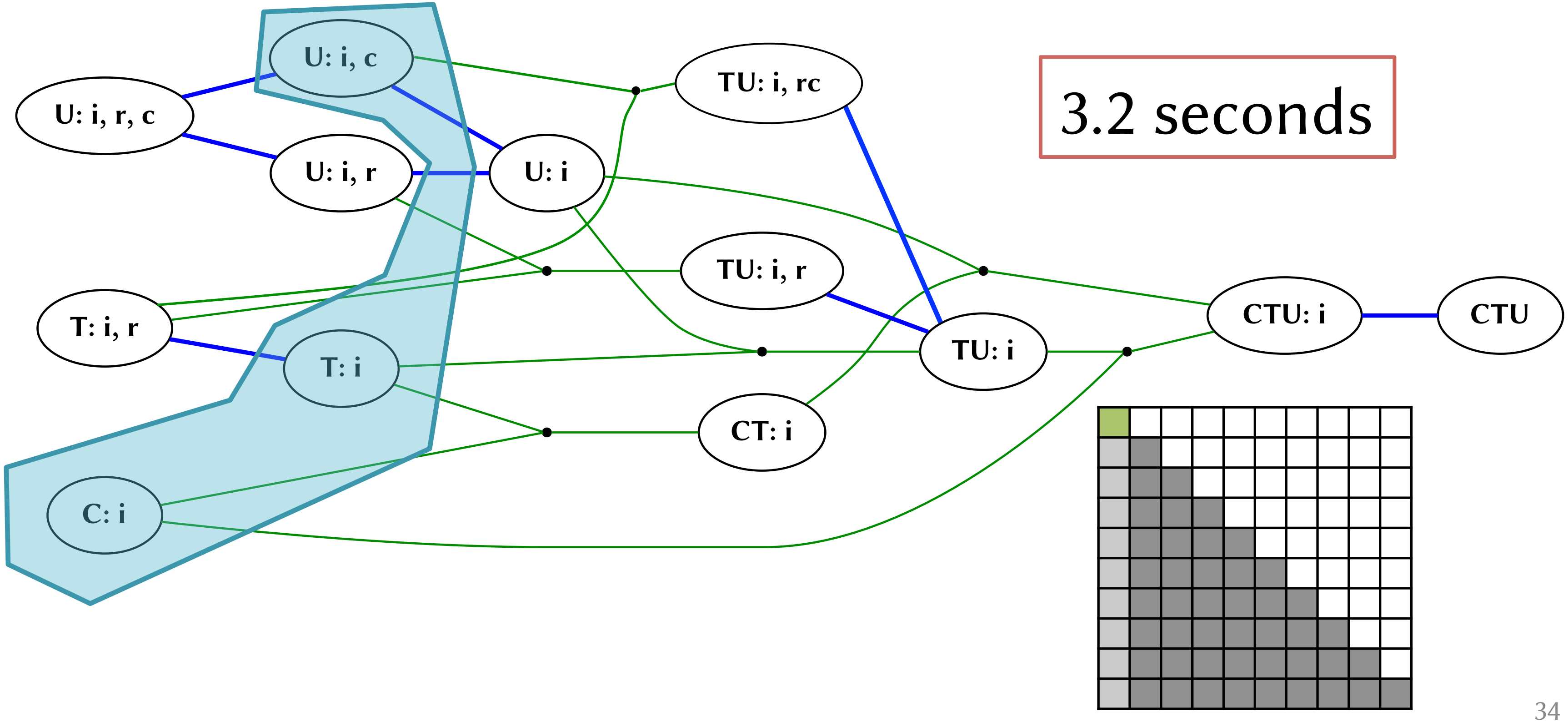


Cholesky: Worst Hierarchy Slice



60.3 seconds

Cholesky: Best Hierarchy Slice



3.2 seconds

Outline

- Hierarchy-related Properties
- Example – Cholesky
- Generation of the Hierarchy Space
- Application – Locality Tuning
- **Conclusion**

Summary

- CnC programs can be framed in terms of:
  - Hierarchy Spaces
  - Hierarchies
  - Hierarchy Slices
- Automatic hierarchy-space generation is more accurate (and less tedious) than manual
- Automatically-generated hierarchy slices can be effectively used in auto-tuning

Future Directions

- Reify multiple levels of *hierarchy* in a single program (not just a single slice)
- Remove algorithm restrictions on decomposition
- Change granularities (not just placement)
  - Related: Chenyang's talk tomorrow
- Explore software-engineering applications

Source Code on GitHub

- github.com/habanero-rice/cnc-framework
- Tag: cnc16-auto-hierarchy

Ω

Hierarchy-based Distribution Results for Cholesky

Hierarchy Slice	Run-time*
(CT: i) + (U: i, c)	3.2 ± 0.2 seconds
(C: i) + (T: i, r) + (U: i, c)	5.6 ± 0.3 seconds
(CT: i) + (U: i, r)	9.0 ± 1.8 seconds
(CTU:)	41.6 ± 4.9 seconds
(C: i) + (T: i, r) + (U: i)	60.3 ± 2.5 seconds

* Averaged over five runs