Tuning expert language: current design

Kath Knobe, Intel
Mike Burke, Rice

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Expectations

- Separation of concerns
  - Domain expert's work is easier
  - The tuning facility is isolated from the domain expert
    => Doesn't have to be overly simplistic
    => Can provide a lot of control and flexibility
- One domain spec / multiple tuning specs
  - Domain spec remains as is
  - Tuning spec is in addition and separate
- Domain spec constraints are still requirements

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Expectation (cont)

- This is an assembly language for tuning
  - Do *what* I say *when* I say to.
    Burden on the tuner, not on our runtime!
  - Could be generated

- Idea
  - Does not: add parallel constructs to basically serial system
  - Does: Constrain the flow of the application through time and across the space, the platform.

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Key concept: Affinity groups

An affinity group is a group of steps to be mapped close together in both time and space

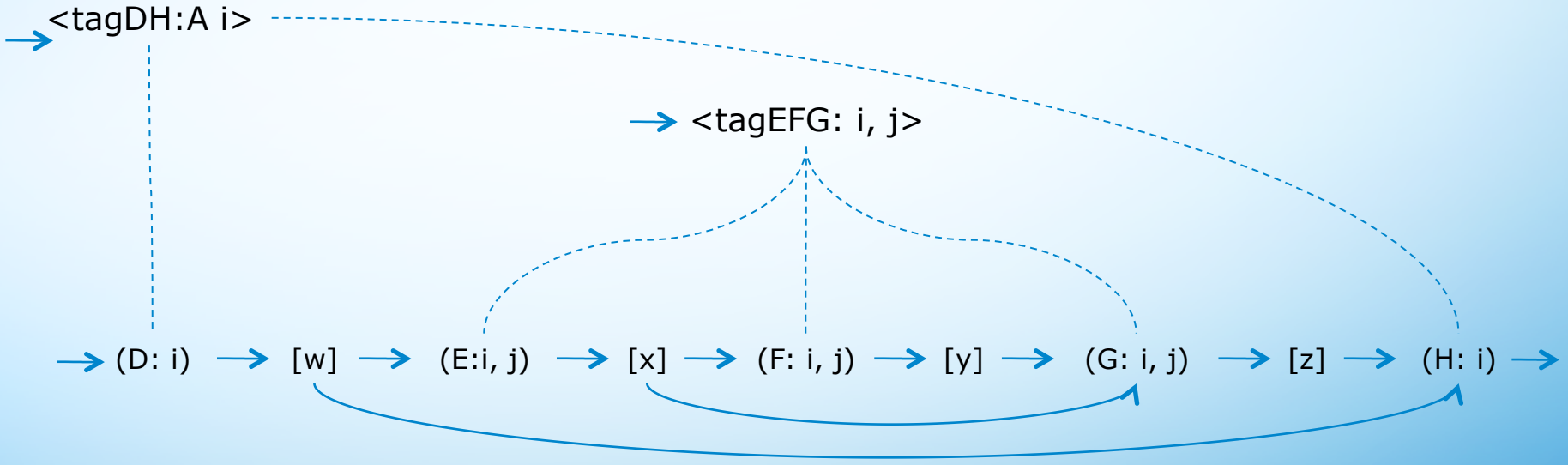
- Based on locality
- No benefit from locality if
  - too far away in time or too far away in space

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Example: domain spec

- Computation steps: (D), (E), (F), (G), (H)
- Data item names: [w], [x], [y], [z]
- Control tags: <tagDH>, <tagEFG>
- Affinity group names: {groupA}, {groupB}, …

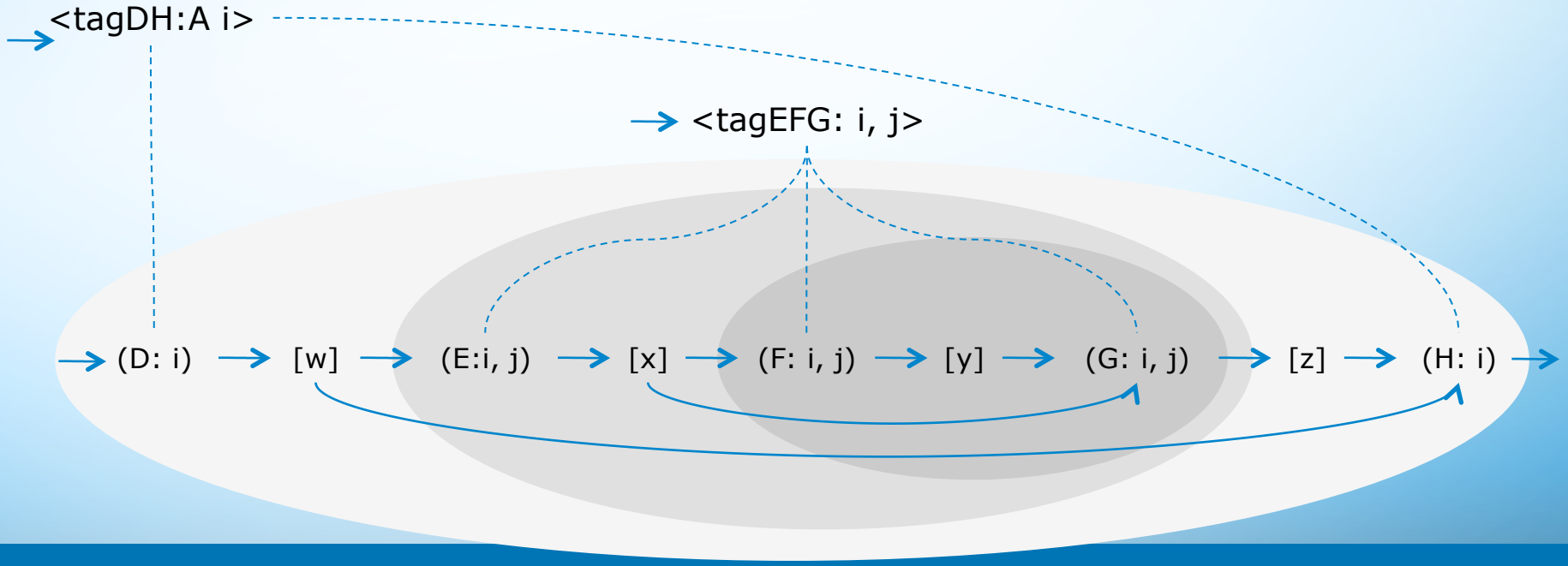


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Example: domain spec

- Computation steps: (D), (E), (F), (G), (H)
- Data item names: [w], [x], …
- Control tags: <tagDH>, <tagEFG>
- Affinity group names: {groupA}, {groupB}, …

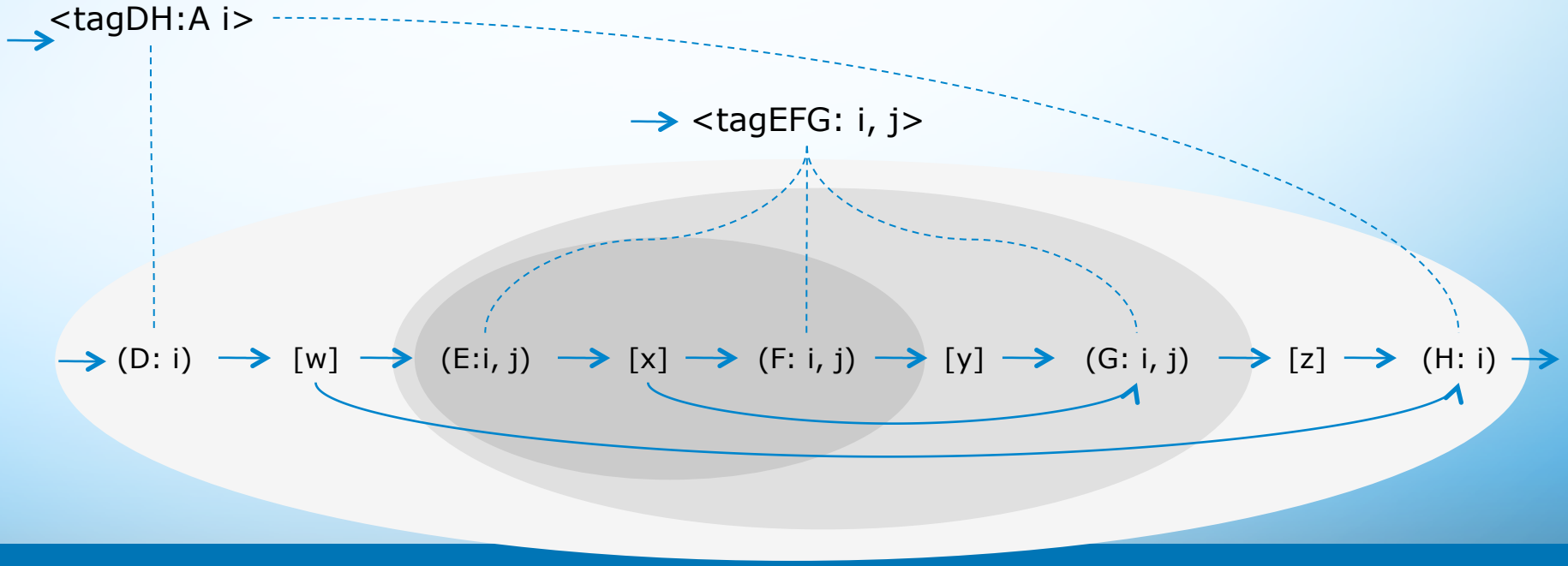


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Example: domain spec

- Computation steps: (D), (E), (F), (G), (H)
- Data item names: [w], [x], …
- Control tags: <tagDH>, <tagEFG>
- Affinity group names: {groupA}, {groupB}, …

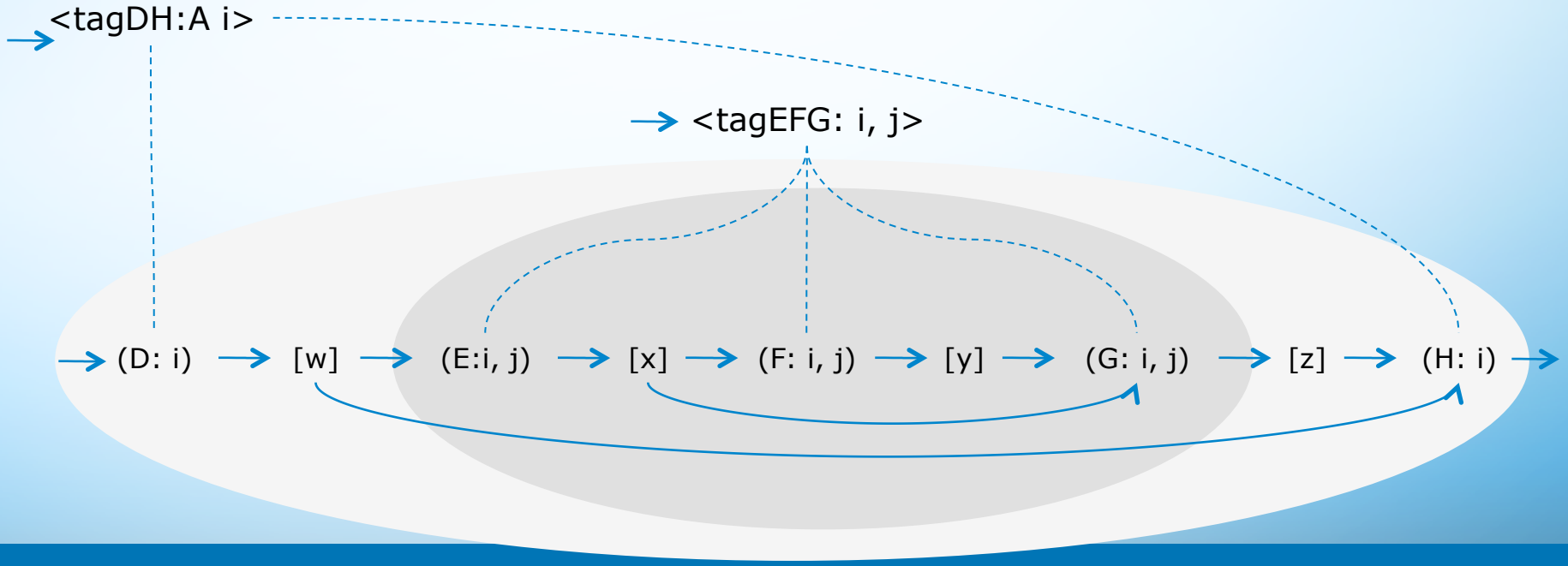


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Example: domain spec

- Computation steps: (D), (E), (F), (G), (H)
- Data item names: [w], [x], [y], [z]
- Control tags: <tagDH>, <tagEFG>
- Affinity group names: {groupA}, {groupB}, …



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Textual hierarchical structure

```
{                (D),

                 {                 (E),
                          {                 (F), (G) }
                                   },
 {group                (H)
}
```


Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Textual hierarchical structure

```
{                (D),

                 {                 (E),
                        {                 (F), (G) }
                                        },
 {group                (H)
}
```


Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Textual hierarchical structure

```
{          (D),

           {groupB   (E),
                {groupC (F), (G) }
                },
 {groupA  (H)
}
```


intel Software
Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Sets => unordered

{ (D), (H),

{ (E),

{ (F), (G) }

}

}

intel Software

Software & Services Group, Developer Products Division

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Groups are named

{**groupA** (D), (H),

{**groupB** (E),

{**groupC** (F), (G) }

}

}

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Groups are prescribed

```
<tagDH: i> ::
    {groupA  (D), (H),
        <tagEFG: i, j> ::
            {groupB   (E),
                {groupC (F), (G) }
                }
    }
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Two components of a tuning language statement

Control:
If and when the action happens.

Action:
What happens.
Identifies a group.

```
<tagDH: i> ::
    {groupA  (D), (H)
        …
    }
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Two components of a tuning language statement

Control:
If and when the action happens.

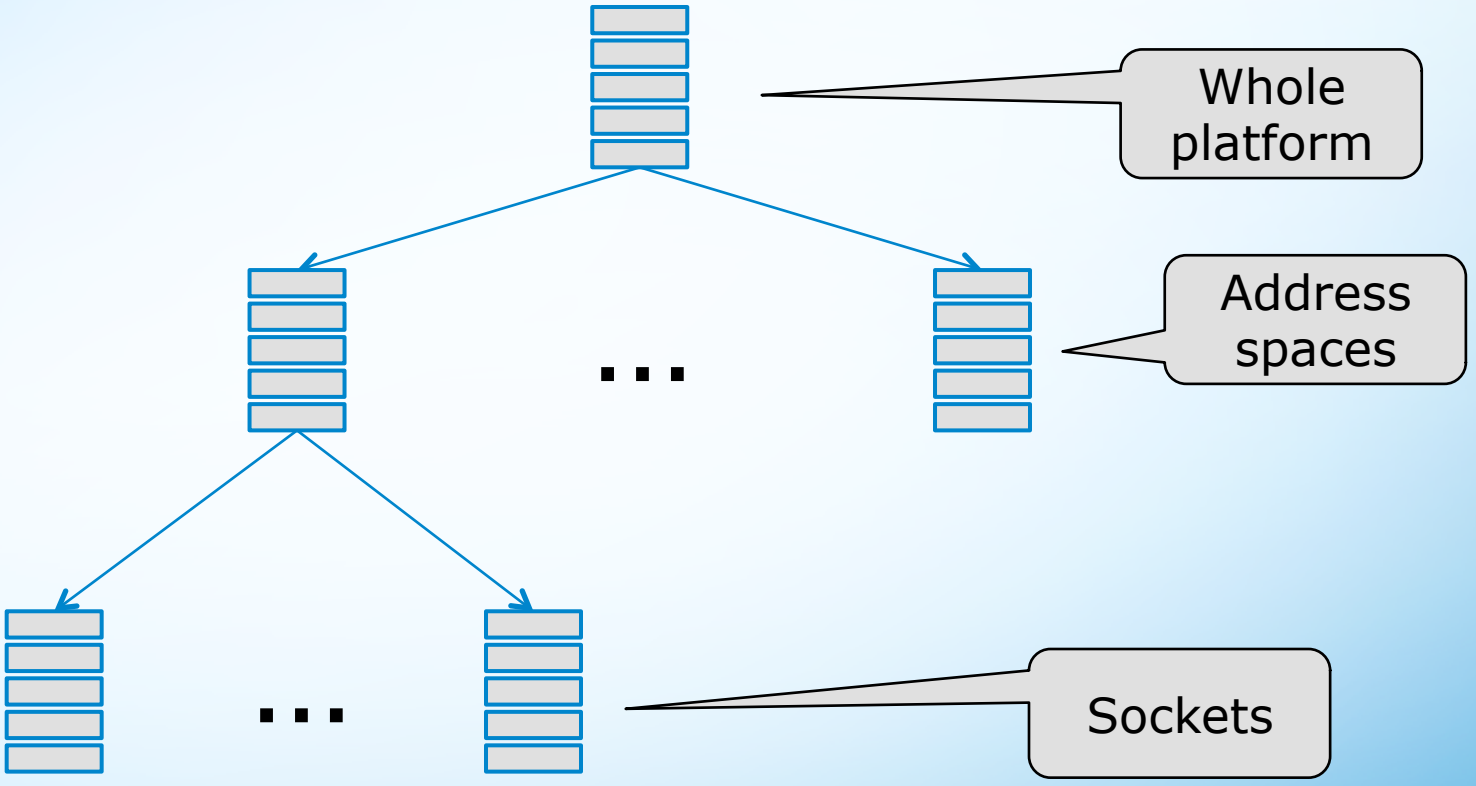
Action:
What happens.
Identifies a group.

```
<tagDH: i> ::
    {groupA  (D), (H)
        …
    }
```

These will each get more interesting.

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

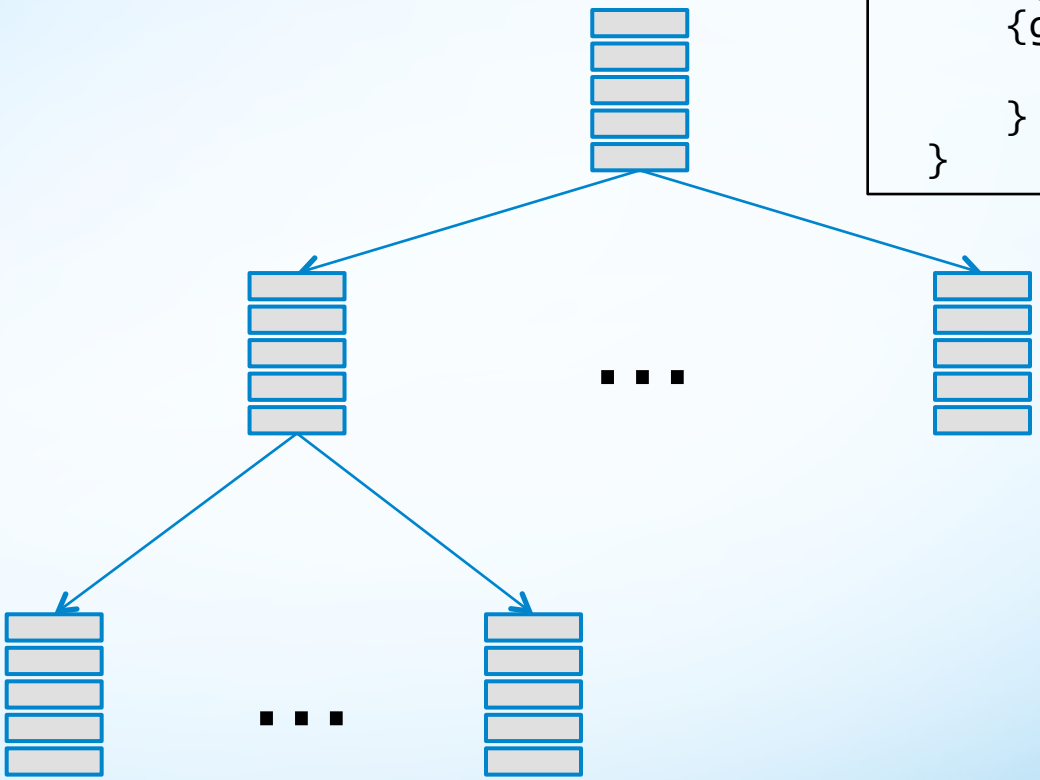
Execution model: Tree of queues



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Execution model: Tree of queues

```
<tagDH: i> ::
  {groupA  (D), (H),
    <tagEFG: i, j> ::
      {groupB   (E),
        {groupC (F), (G) }
      }
  }
```



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

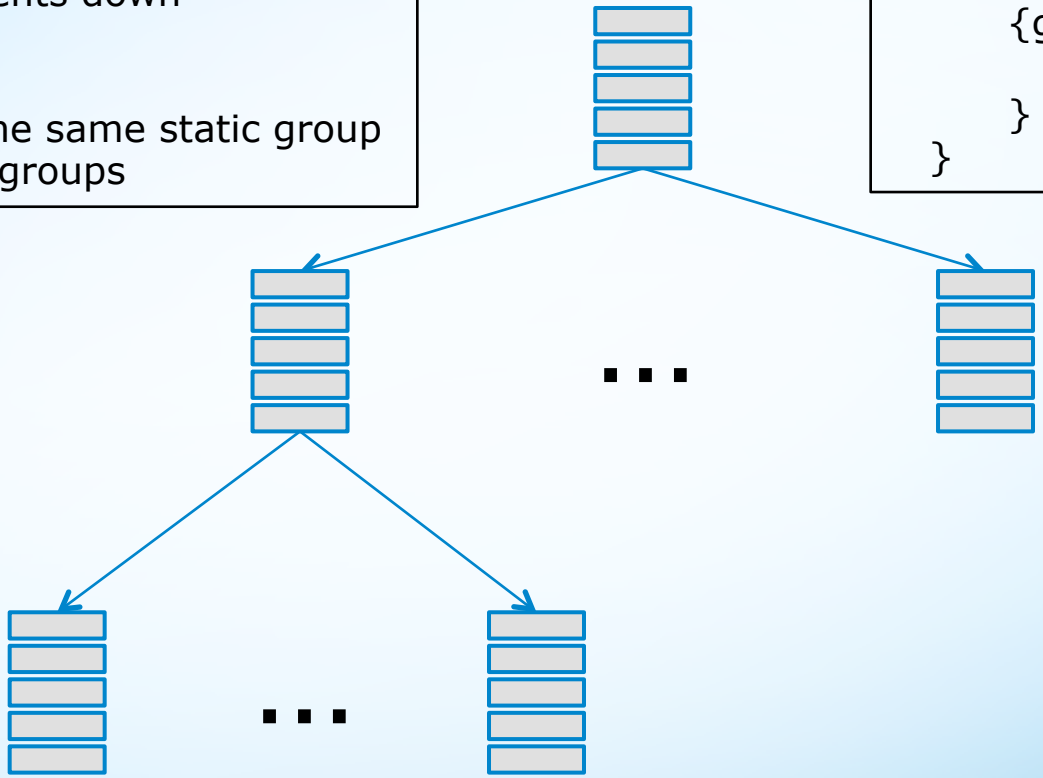
Optimization Notice

Execution model: Tree of queues

- Start outer group at the top
- Break group into components
- Move components down

Components:
- Instances of the same static group
- Distinct static groups

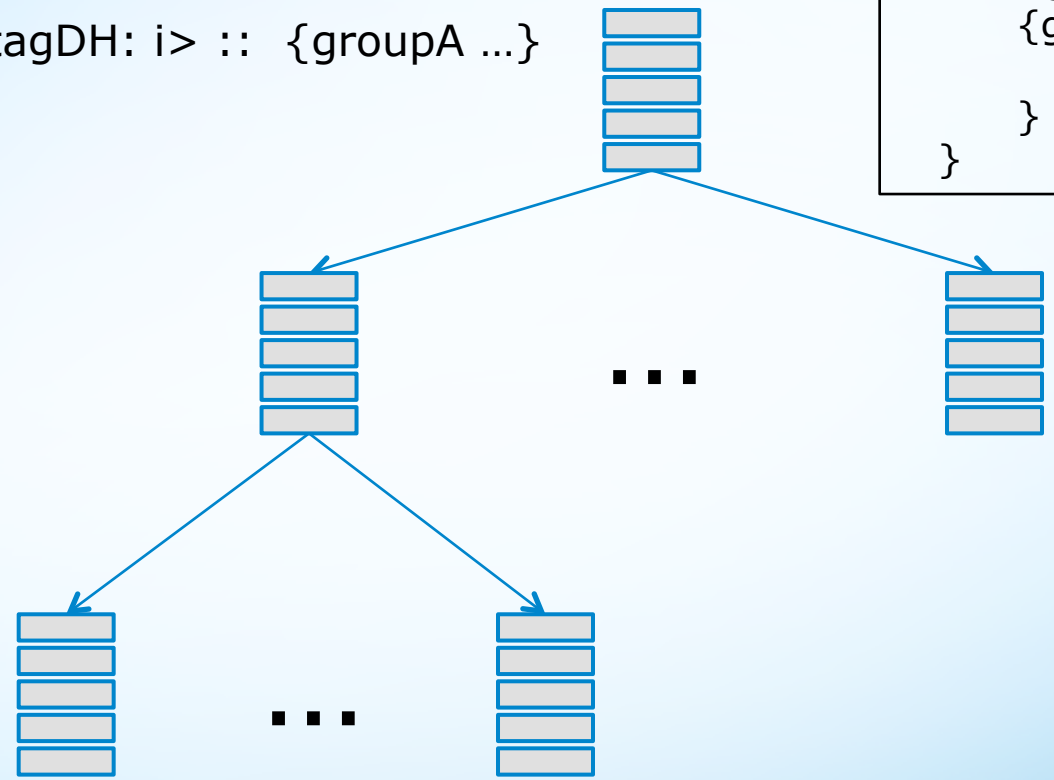
```
<tagDH: i> ::
  {groupA  (D), (H),
    <tagEFG: i, j> ::
      {groupB   (E),
        {groupC (F), (G) }
      }
  }
```



intel Software
Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Execution model: Tree of queues

<tagDH: i> :: {groupA …}

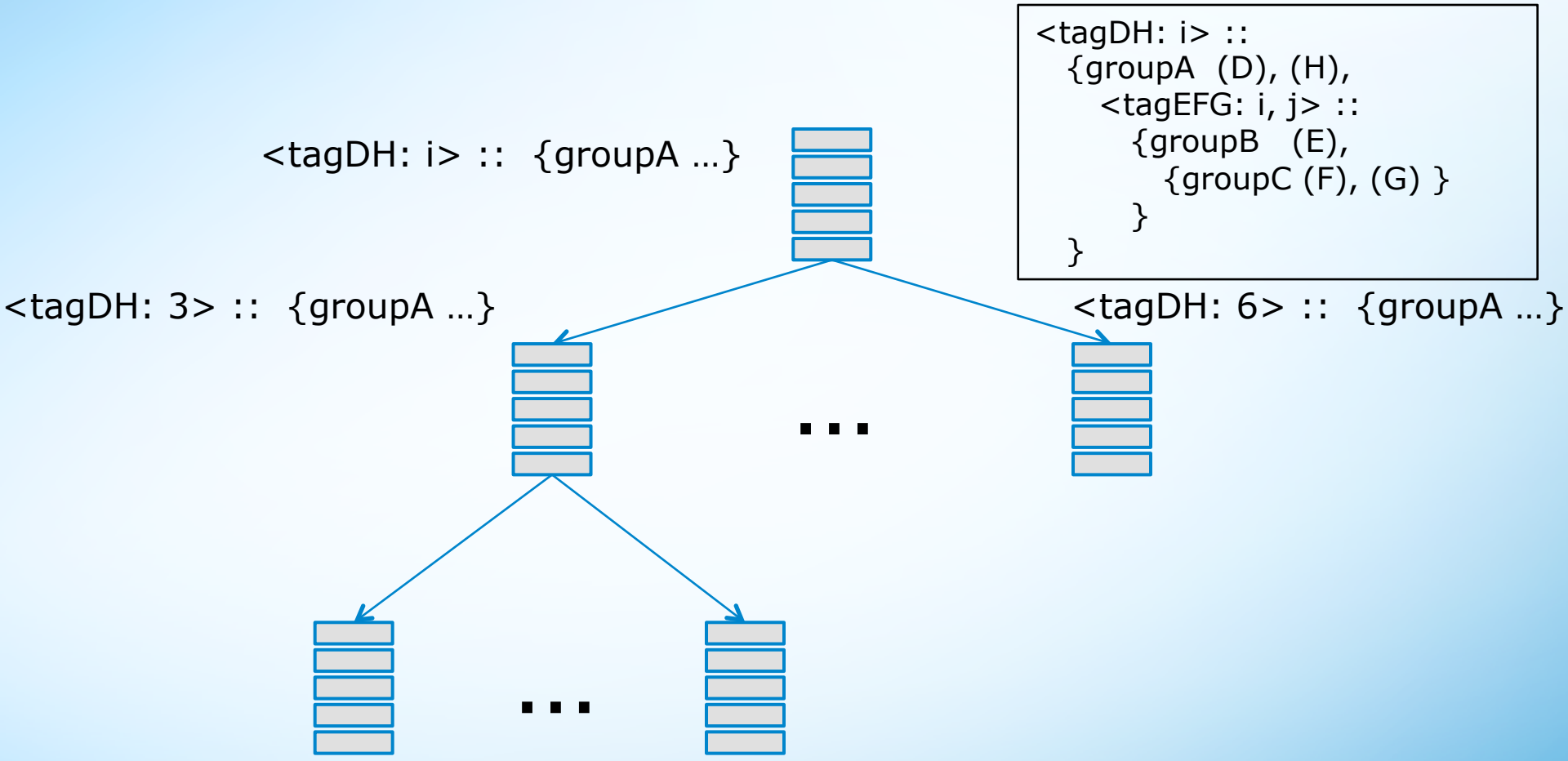


```
<tagDH: i> ::
  {groupA  (D), (H),
    <tagEFG: i, j> ::
      {groupB   (E),
        {groupC (F), (G) }
      }
  }
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

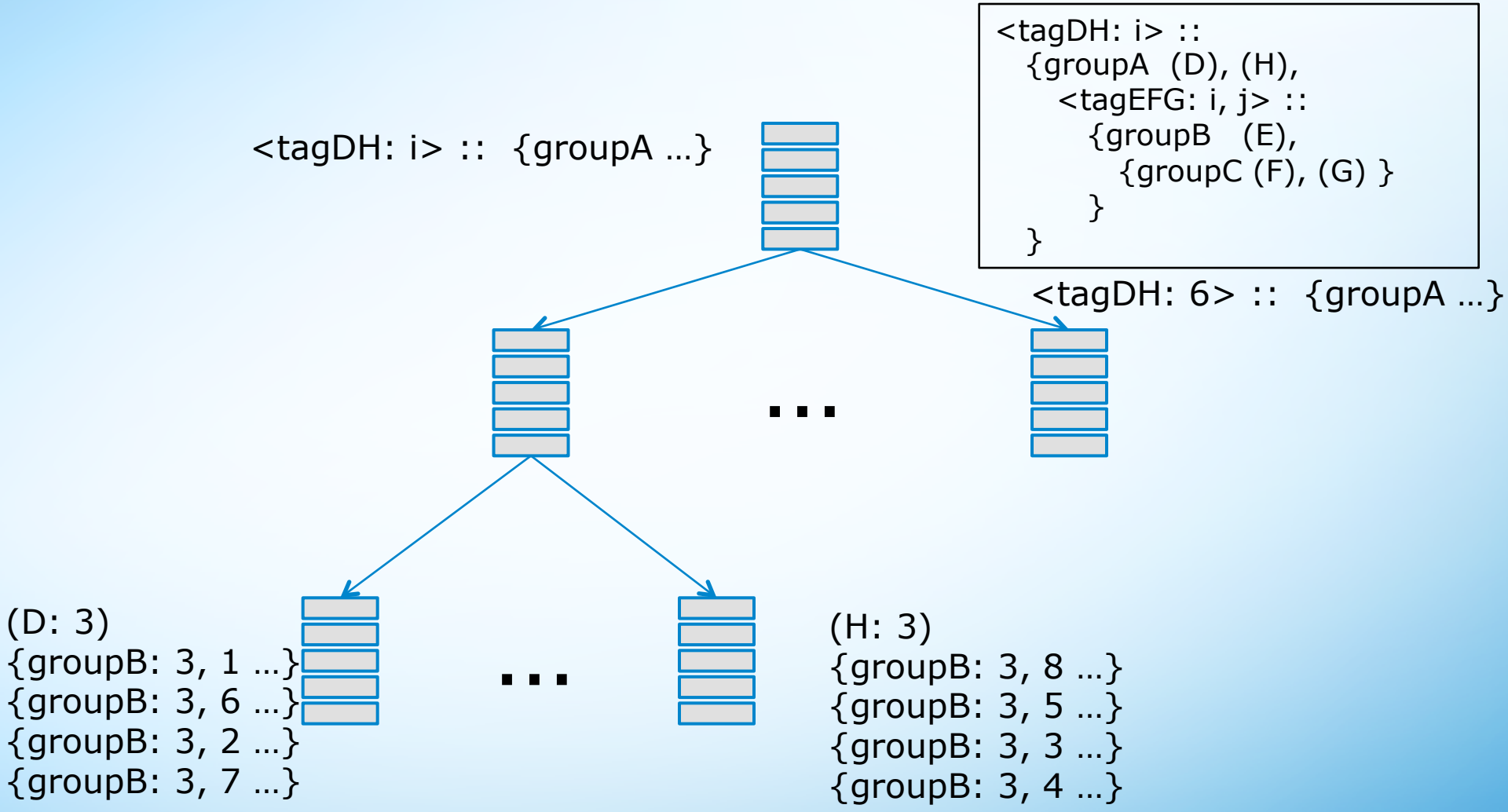
Optimization Notice

Execution model: Tree of queues



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Execution model: Tree of queues



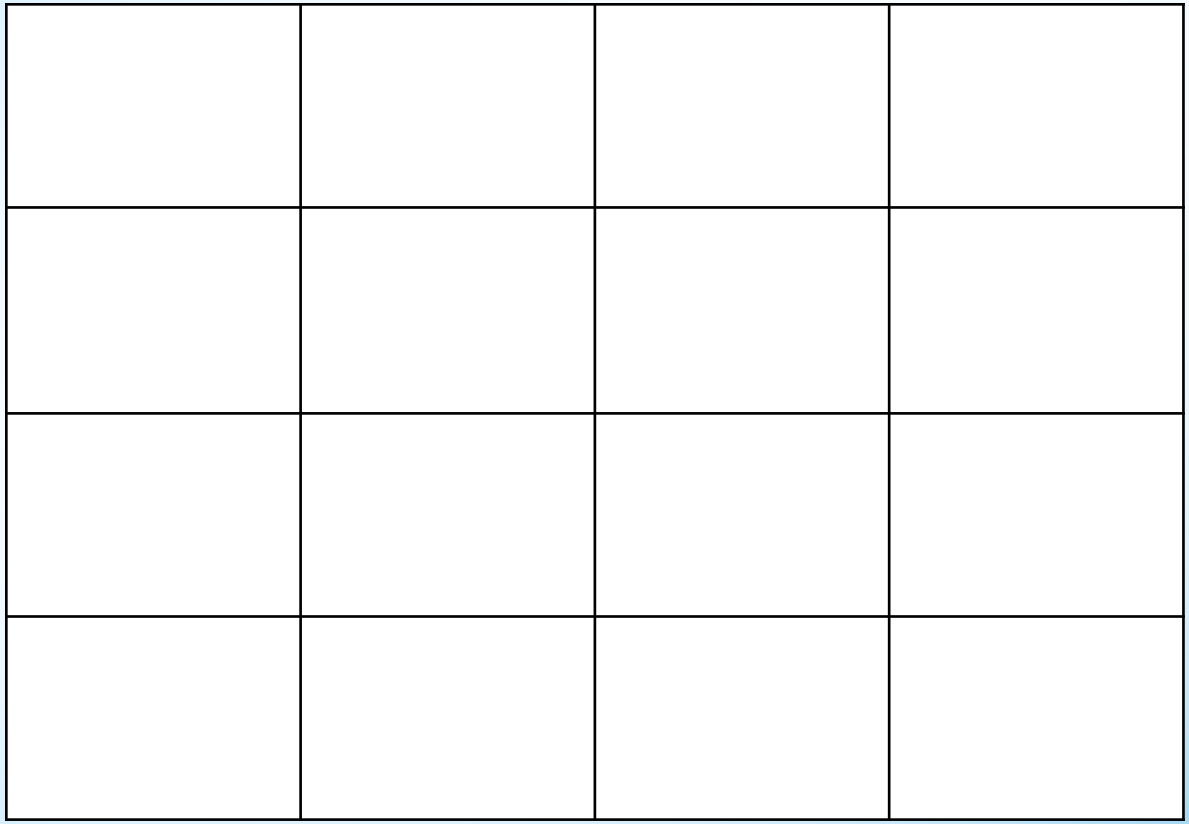
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

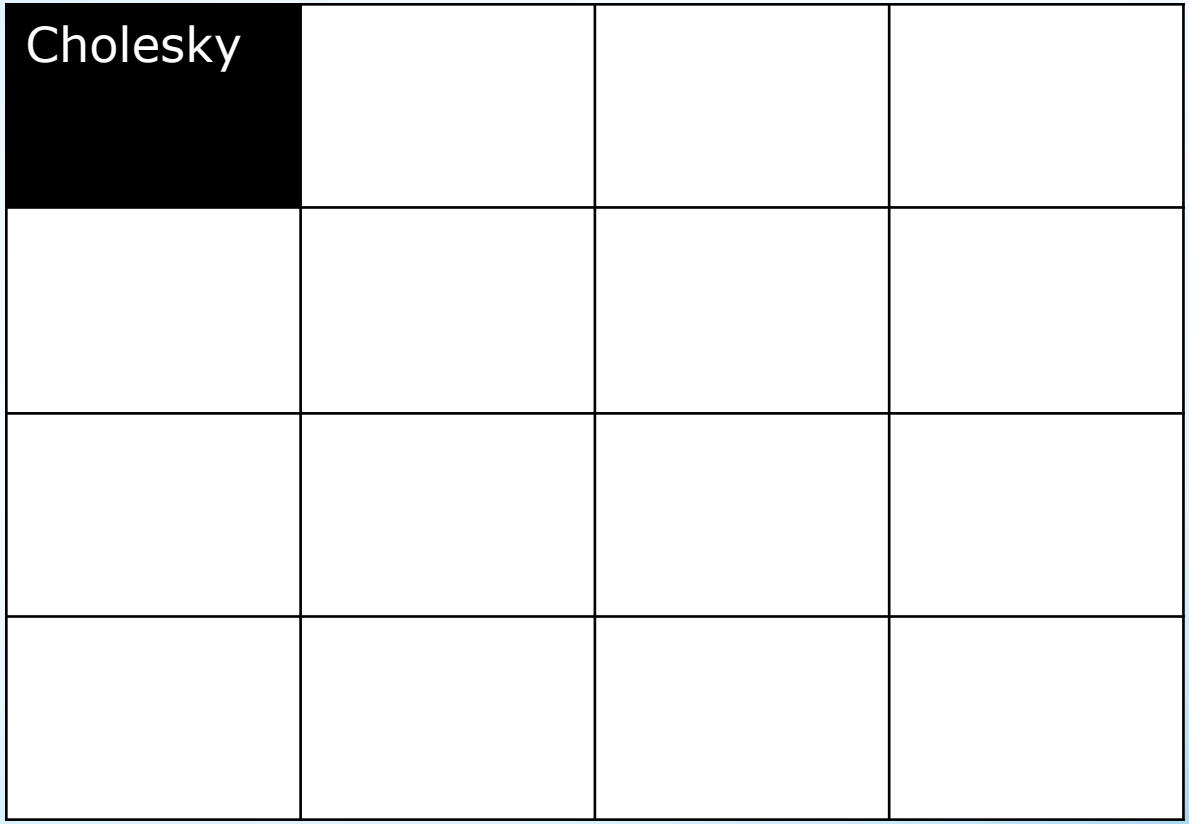
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky factorization



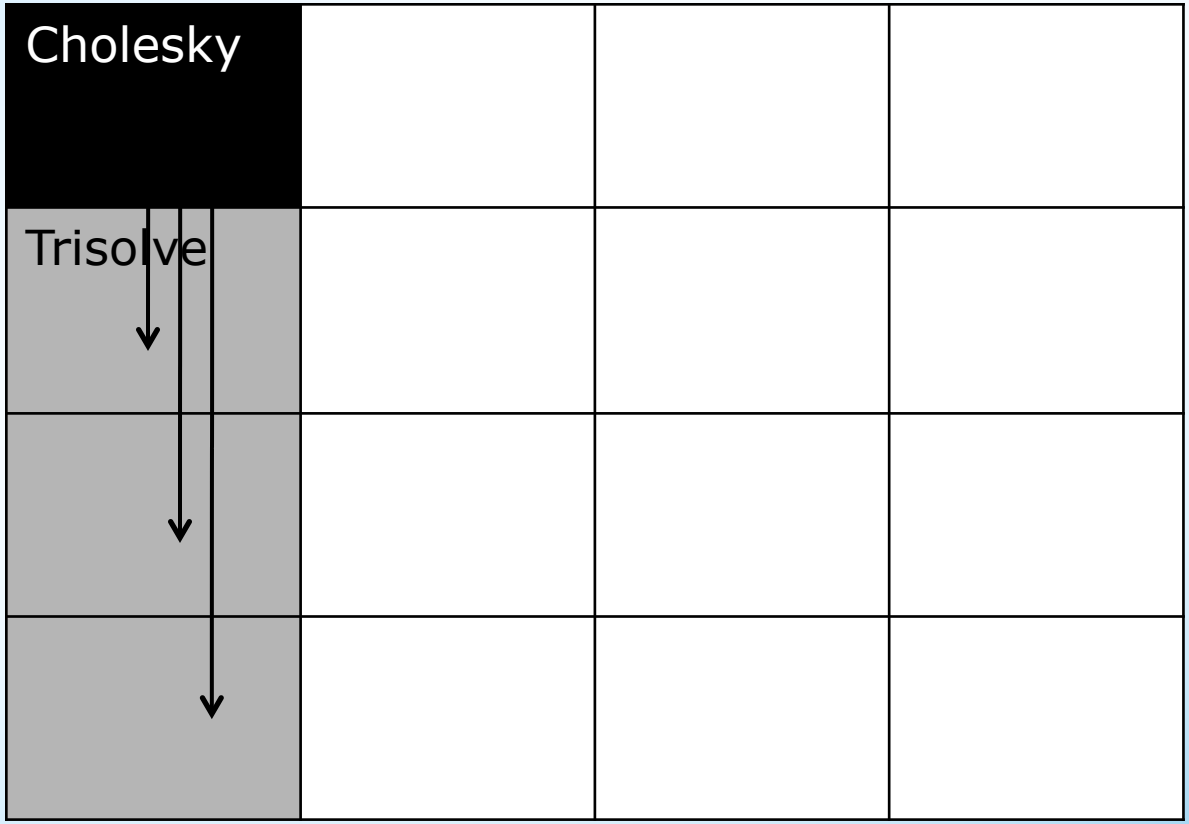
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky factorization



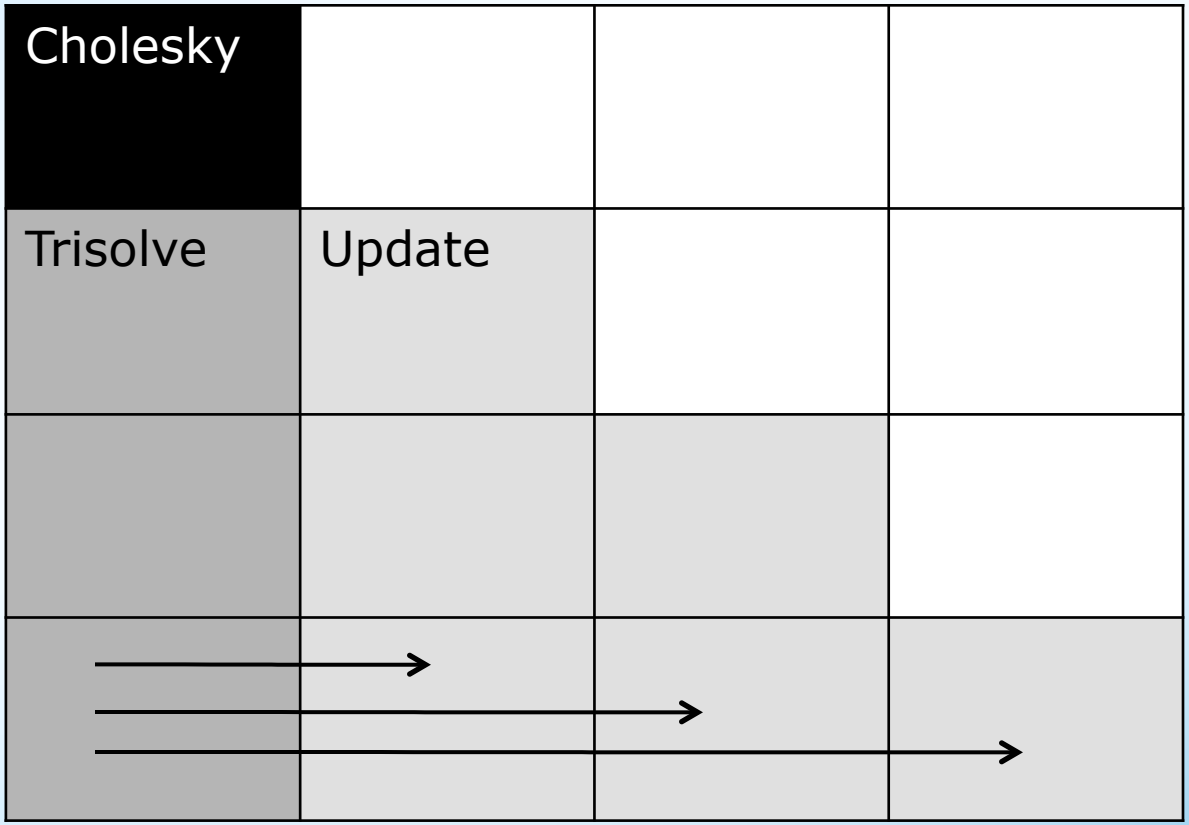
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky factorization



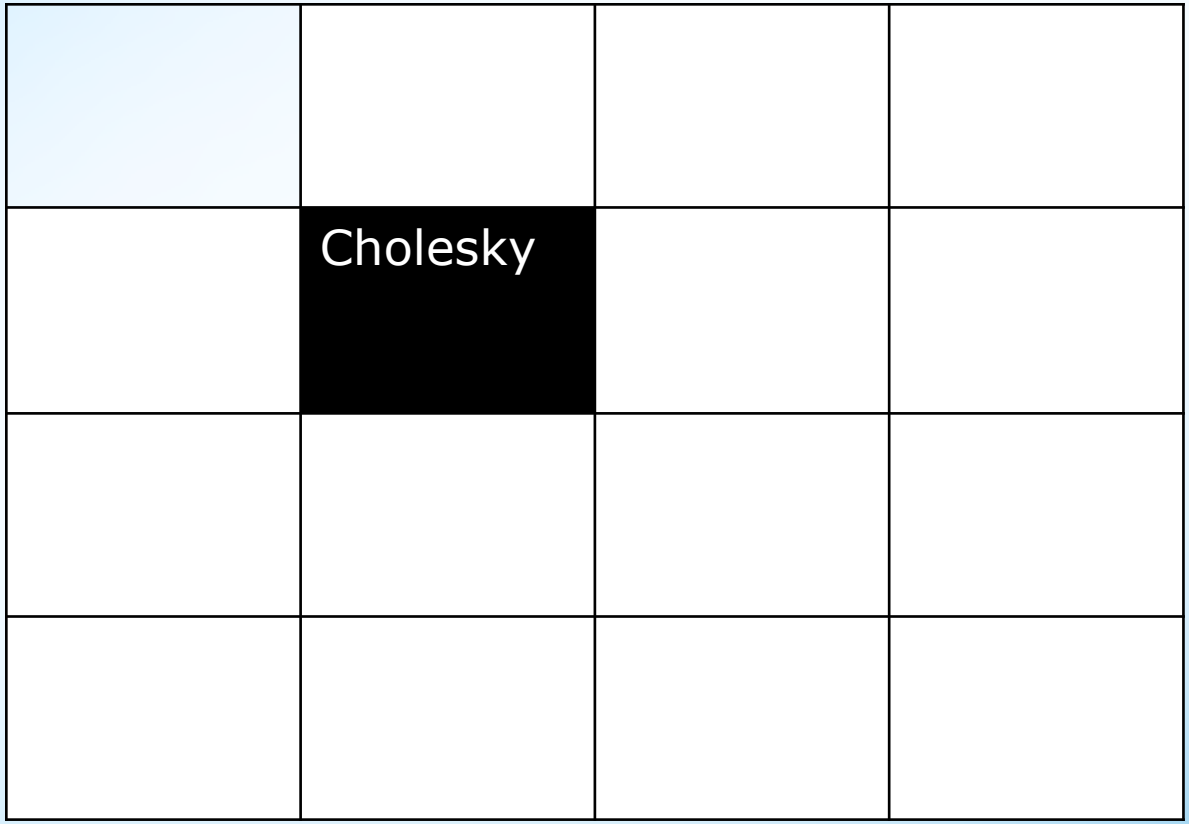
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky factorization



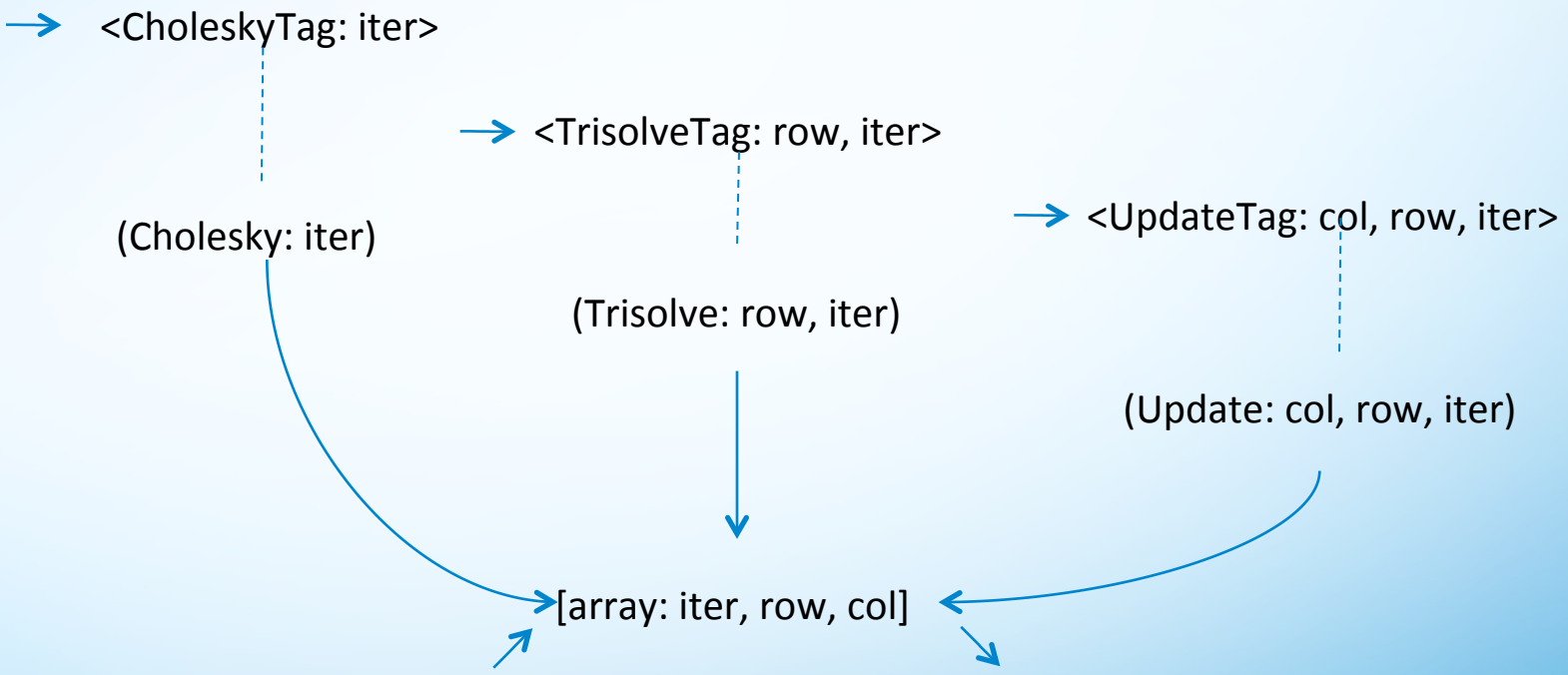
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky factorization



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky domain spec



intel Software
Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Cholesky: obvious grouping

Outer group {Cho-Tri-Up} is one whole iteration
Inner group {Tri-Up} is a row within an iteration

```
<CholeskyTag: iter> :: {Cho-Tri-Up
    ;; weak affinity within an iteration
    (Cholesky: iter)
    <TrisolveTag: row, iter> :: {Tri-Up
        ;; tight affinity for a given row within an iteration
        ;; for a row, the trisolve and the updates are together
        (Trisolve: row, iter)
        <UpdateTag: col, row, iter> ::
                    (Update: col, row, iter)
            }
}
```


intel Software
Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Cholesky: tiled grouping

N = 100
blockSize = 25
colBlock is 1,2,3, or 4
colOfBlock is 1,2,3,4,5,6….25
The function: Col(colOfBlock, colBlock)
Converts the blocked version:
    <colOfBlock, ColBlock>
back to the original column #
    example: colOfBlock = 24
                colBlock = 2
    Col(colOfBlock, colBlock) is 49

env -> [blockSize], [N] -> (computeAffinityTags)
;; tag for each column block, row and iteration
-> <blockTags: colBlock, row, iter>,
;; tag for each column within block, block, row, iteration
<colOfBlockTags: colOfBlock, colBlock, row, iter>

<CholeskyTag: iter>:: {Cho-Tri-Up
;; all iters have weak affinity
(Cholesky: iter)
<TrisolveTag: row, iter> :: {Tri-Up
;; for <iter> all rows have medium affinity
(Trisolve: row, iter)
<blockTags: colBlock, row, iter>:: {UpBlock
;; for <row, iter> colBlocks have tight affinity
<colOfBlockTags: colOfBlock, colBlock, row, iter> :: {UpOfBlock
;; for <colOfBlock, row, iter> columns within a block have very tight affinity
(Update: **Col(colOfBlock, colBlock),** row, iter)
} } } }

intel Software
Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Cholesky: tiled grouping

env -> [blockSize], [N] -> (computeAffinityTags)
;; tag for each column block, row and iteration
-> <blockTags: colBlock, row, iter>,
;; tag for each column within block, block, row, iteration
<colOfBlockTags: colOfBlock, colBlock, row, iter>

N = 100
blockSize = 25
colBlock is 1,2,3, or 4
colOfBlock is 1,2,3,4,5,6….25
The function: Col(colOfBlock, colBlock)
Converts the blocked version:
<colOfBlock, ColBlock>
back to the original column #
example: colOfBlock = 24
colBlock = 2
Col(colOfBlock, colBlock) is 49

<CholeskyTag: iter>:: {Cho-Tri-Up
;;
(C
<

Like tiling for execution.
No easier. No harder.

;; for <colOfBlock, row, iter> columns within a block have very tight affinity
(Update: **Col(colOfBlock, colBlock),** row, iter)
} } } }

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Obvious and Tiled versions:

two tunings of the same domain spec

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

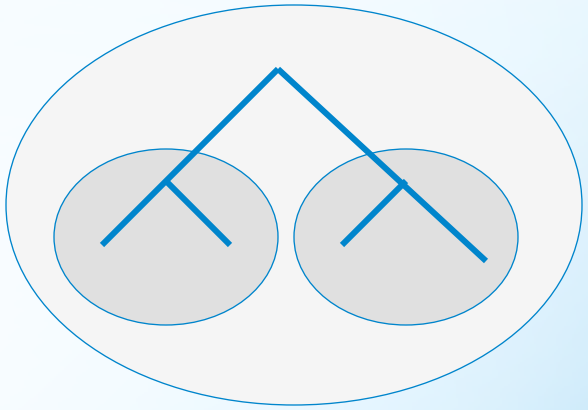
Optimization Notice

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Recursive specification (breadth-first) affinity groups

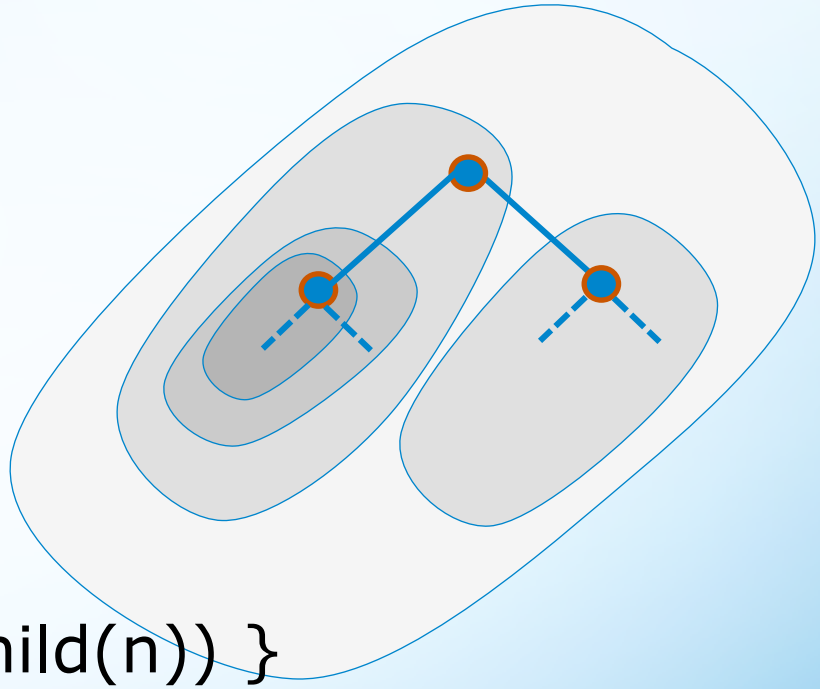


```
{treeGroup          (step: n)
                    {treeGroup (leftChild(n)}
                    {treeGroup (rightChild(n)}
}
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

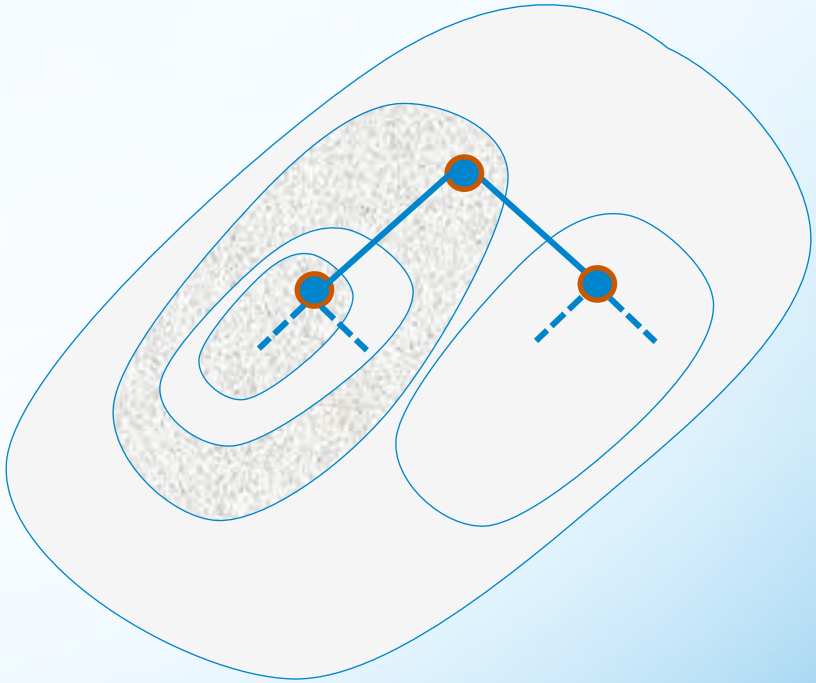
Recursive specification (depth-first) affinity groups



```
{treeGroup: n
      {treeGroupLeft: n
            (step: n)
            {TreeGroup(leftChild(n)) }
      }
      {treeGroup(rightChild(n)) }
}
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Recursive specification
2 group types

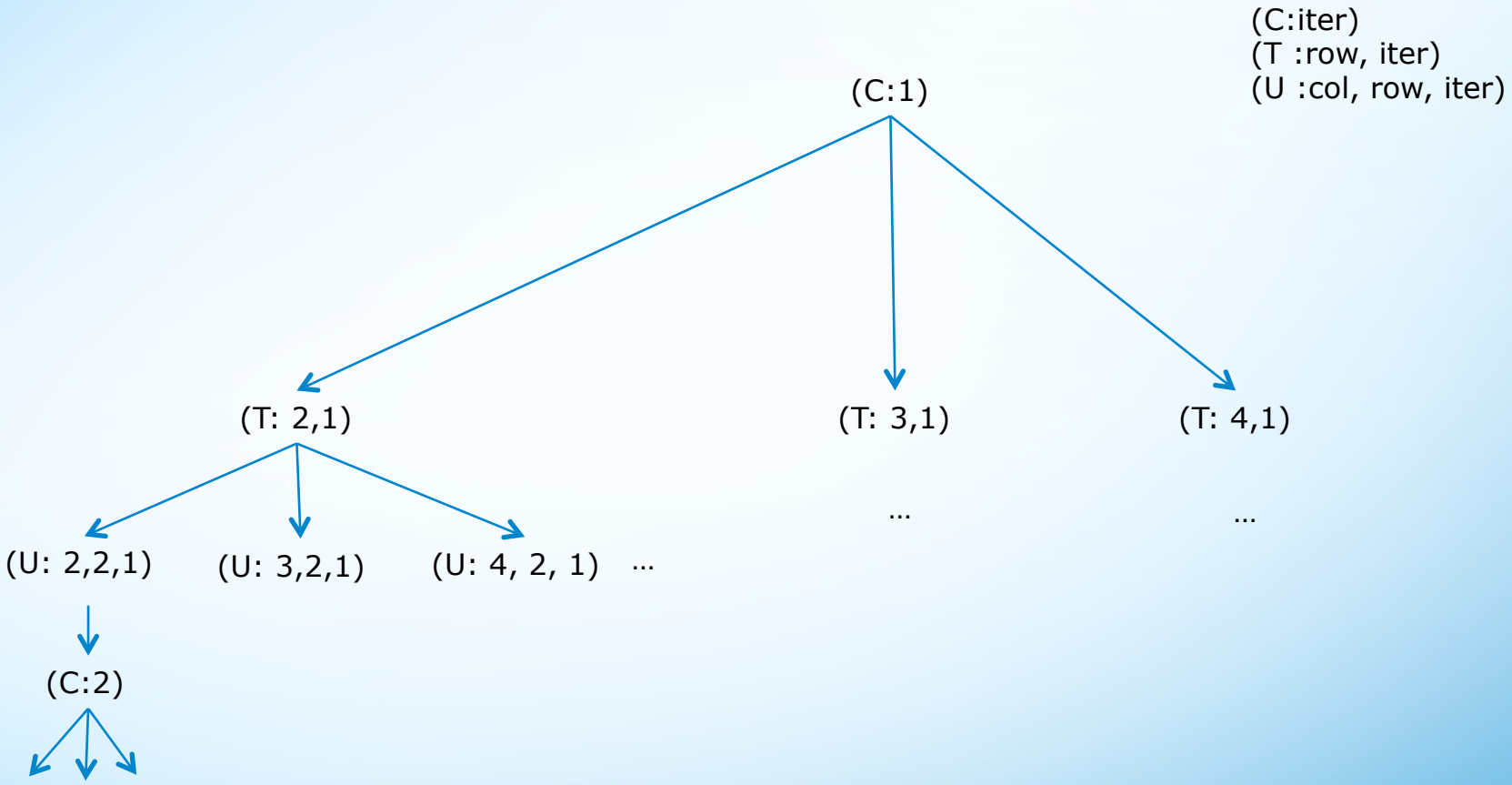


```
{treeGroup: n
      {treeGroupLeft: n
            (step: n)

      {TreeGroup(leftChild(n)) }
      }
      {treeGroup(rightChild(n)) }
}
```

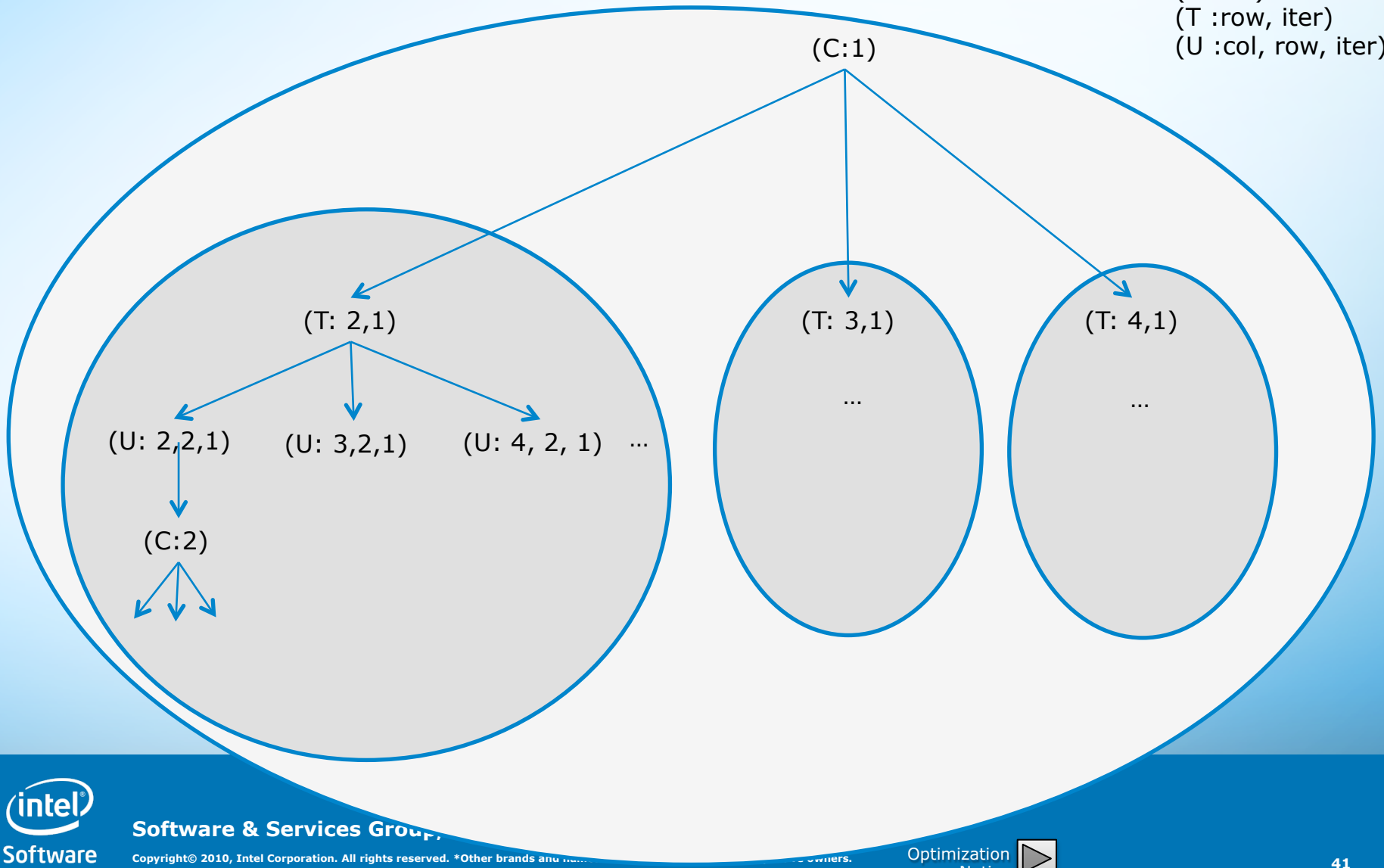

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky as a tree



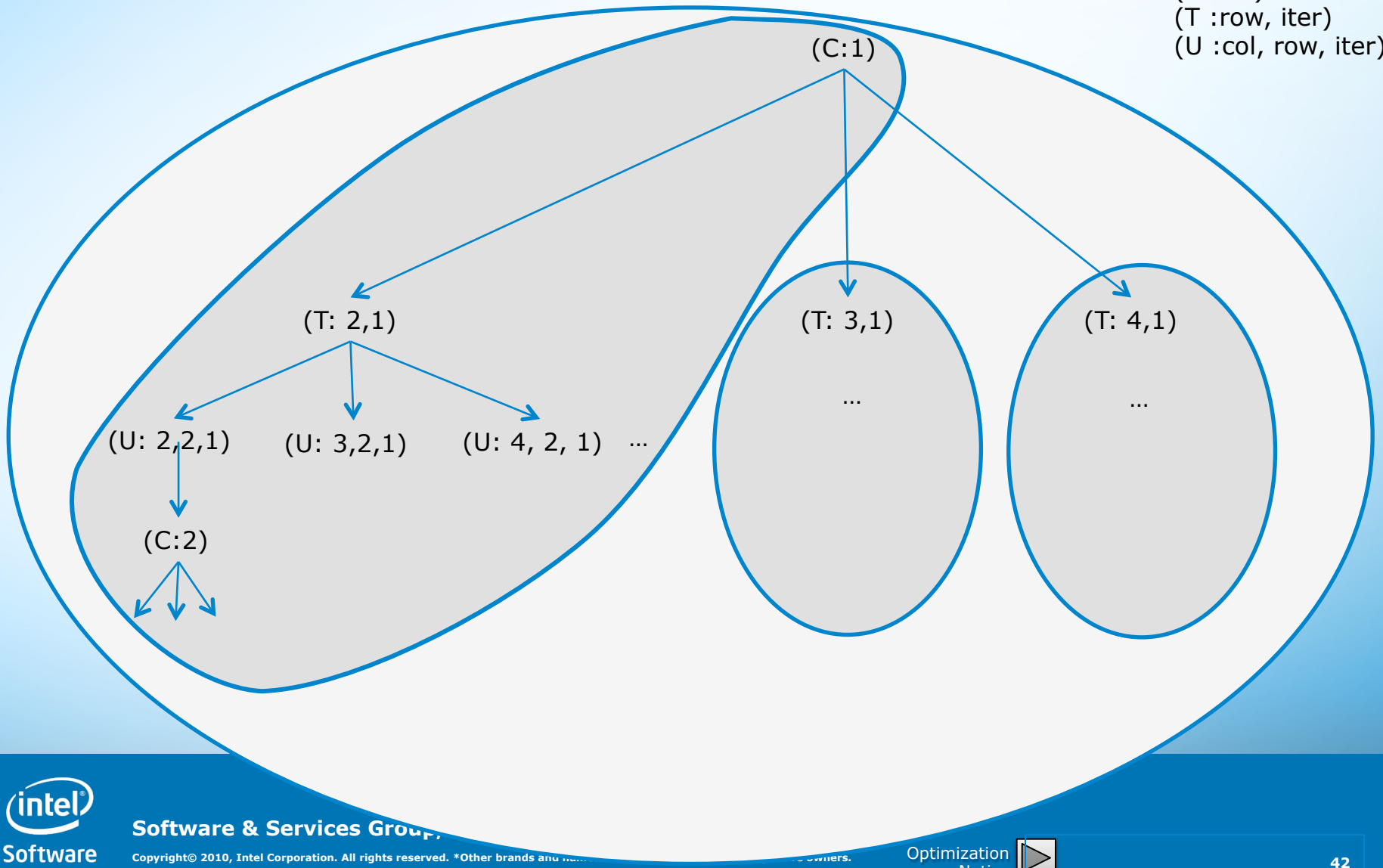
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Cholesky as a breadth-first tree



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and ... owners.

Cholesky as a depth-first tree



Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and
owners.
Optimization Notice

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Control via attributes expressions

- Control

  <groupTag: i> :: {group … }

  Do action {group: 3}

  If/when 3 is in the prescribing collection

- CnC semantic attribute expressions

  A general mechanism - specifies if/when in a CnC execution

  Examples:

  <groupTag: i>.available :: {group … }

  (foo: i).executed and [x: i].available :: {group …}

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Control via attribute expressions in Cholesky

```
<CholeskyTag: iter>.available and [X: iter, iter, iter-1].available :: {Cho-Tri-Up
    ;; weak affinity within an iteration
    (Cholesky: ite
    <TrisolveT        w, iter> :: {Tri-Up
        ;;            nity for a given row within an iteration
                                                  s are together
                    (Update: col, row, iter)
        }
}
```

If: All the tags are known from the beginning

Software & Services Group, Developer Products Division
Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Control via attribute expressions in Cholesky

```
<CholeskyTag: iter>.available and [X: iter, iter, iter-1].available :: {Cho-Tri-Up
      ;; weak affinity w thin an iteration
      (Cholesky: ite
      <TrisolveT      w, iter> :: {Tri-Up
                     ity for a given row within an iteratio
                         (Update: col, row, iter)
            }
}
```

If: All the tags are known from the beginning

When: delay until input is available

intel Software

Software & Services Group, Developer Products Division

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Optimization Notice

Explicit space mappings

```
<CholeskyTag: iter> :: {Cho-Tri-Up replicate across address-spaces
    ;; weak affinity within an iteration
    (Cholesky: iter)
    <TrisolveTag: row, iter> :: {Tri-Up distribute across sockets
        ;; tight affinity for a given row within an iteration
        ;; for a row, the trisolve and the updates are together
        (Trisolve: row, iter)
        <UpdateTag: col, row, iter> ::
                    (Update: col, row, iter)
            }
}
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Explicit space mappings

```
<CholeskyTag: iter> :: {Cho-Tri-Up replicate across address-spaces
    ;; weak affinity within an iteration
    (Cholesky: iter)
    <TrisolveTag: row, iter> :: {Tri-Up distribute across sockets
                                        socket# = f(row, iter)
        ;; tight affinity for a given row within an iteration
        ;; for a row, the trisolve and the updates are together
        (Trisolve: row, iter)
        <UpdateTag: col, row, iter> ::
                (Update: col, row, iter)
            }
}
```


Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Basic idea for mapping to time

We already have ways of expressing

- Before and after:
  - in domain spec
  - in tuning language
- Close: via affinity groups

What else do we need?

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Basic idea for mapping to time

What else do we need?

Relative time among components within a group.

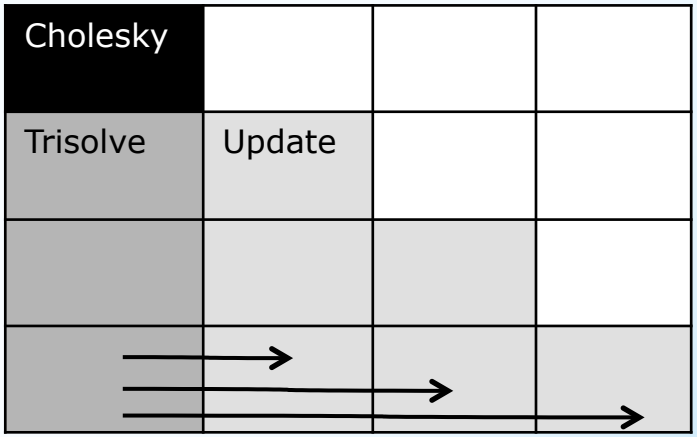
	ordered	unordered
Non-overlapping	serial/barrier	exclusive
overlapping	priority	arbitrary

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Mapping to time in Cholesky

Obvious affinity grouping

```
<CholeskyTag: iter> :: {Cho-Tri-Up
      ;; all iters have weak affinty
      (Cholesky: iter)
      <TrisolveTag: row, iter> :: {Tri-Up
            ;; for <iter> all rows have medium affinity
            (Trisolve: row, iter)
             <UpdateTag: col, row, iter> :: {Up
                   ;; for <row, iter> cols have tight affinity
                   (Update: col, row, iter)
              }
      }
}
```



Priorities

{Tri-Up: row, iter … **priority func(row)**
}

This will execute the rows of trisolve and update : in order with possible overlap

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Optimization Notice

Examples of space-time combinations

- {big_memory_footprint: i}

  Distribute across address spaces (space)
  Within each address space exclusive (time)

- {video_processing: frameID}

  cyclic distribution among address spaces(space)
  ordered & overlapping within address spaces (time)

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Outline

- Basics of affinity groups
  syntax and execution model
- Tuning Cholesky
- Recursively defined affinity groups
- Advanced topics
- Wrap-up

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Summary of possibilities for controls and actions

Control

- Prescription
- Attribute expression

Actions

- Define a group
- Map group with space-specific mappings
- Map group with time-specific mappings
- Include items in affinity groups; Map items to space
- Add steps, items and tags used only for tuning
- Set/ref tuner-defined attributes (e.g., demanded, dead)
- Set/ref tuner-defined values (e.g., optimize for power/time)
- Explicitly move items

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Related work

- Control of time
  - Various types of parallelism
    - Data parallel, task parallel, streaming, serial, …
- Control of space
  - Support for distributions
    - Hierarchical Place Trees,HPF, …

We can do each of these separately.

But we can also blur the distinction between them.

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Status

- CnC is a significant part of our Ubiquitous High performance Computing (UHPC) project. Tuning is a key component.
- Working on tuning spec examples
- Working out design details
  - with Feng Qiao and Chih-Ping Chen
- Before full implementation
  - developing a stylized way of adding to domain spec to simulate a specific tuning spec
- Soliciting feedback from our community

Call to action

- Ask the naïve/hard questions
- Point out all problems/concerns
- Try writing tuning spec for your favorite app
- Give us all feedback

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Thanks

- Feng Qiao – Intel/SSG
  Visiting professor from Beijing Technical University
- Chih-Ping Chen – Intel/SSG
- Frank Schlimbach – Intel/SSG
- Steve Lang – Intel/DAP

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

End

Copyright© 2010, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.