What will be new in OpenMP 4.0?

Dr.-Ing. Michael Klemm
Software and Services Group
Intel Corporation
(michael.klemm@intel.com)

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

Copyright © 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, Phi, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may be claimed as the property of others.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: Motivation

- Allows to extend what types and operations are allowed in an OpenMP reduction clause:

```
struct point {
    int x;
    int y;
};

struct point points[N];
struct point min = { MAX_INT, MAX_INT }, max = {0,0};

#pragma omp parallel for reduction(….)
for ( int i = 0; i < N; i++ )
{
           if ( point[i].x < min.x ) min.x = point[i].x;
           if ( point[i].y < min.y )  min.y = point[i].y;
           if ( point[i].x > max.x ) max.x = point[i].x;
           if ( point[i].y > max.y ) max.y = point[i].y;
}
```

Not possible before 4.0

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: declaration

- New declarative directive

#pragma omp declare reduction(*reduction-identifier* **:** *typename-list* **:** *combiner* **)** [*initializer-clause*]

!$omp declare reduction(*reduction-identifier* **:** *type-list* **:** *combiner* **)** [*initializer-clause*]

- *reduction-identifier* is the “operator” name given to this reduction
- *combiner* specifies how to combine two elements of one the specified types
  - Only two special variables can be used:
    - omp_in
    - omp_out
  - In C/C++, an expression
  - In Fortran, either an assignment statement or subroutine name with its arguments
    - no CALL keyword

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: declaration

- *initializer-clause*
  - Specifies how to initialize the private elements of each thread
  - Special variable omp_priv represents the private element
  - Optional special variable omp_orig represents the original variable
  - No other variable is allowed
  - If not specified, a default "zero-value" is used

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: example

```
struct point {
    int x;
    int y;
};
#pragma omp declare reduction(min : struct point : \
          omp_out.x = omp_in.x > omp_out.x  ? omp_out.x : omp_in.x, \
          omp_out.y = omp_in.y > omp_out.y  ? omp_out.y : omp_in.y ) \
          initializer( omp_priv = { MAX_INT, MAX_INT } )
#pragma omp declare reduction(max : struct point : \
          omp_out.x = omp_in.x < omp_out.x  ? omp_out.x : omp_in.x, \
          omp_out.y = omp_in.y < omp_out.y  ? omp_out.y : omp_in.y ) \
          initializer( omp_priv = { 0,0 } )

struct point points[N];
struct point minp = { MAX_INT, MAX_INT }, maxp = {0,0};

#pragma omp parallel for reduction(min:minp) reduction(max:maxp)
for ( int i = 0; i < N; i++ )
{
          if ( point[i].x < minp.x ) minp.x = point[i].x;
          if ( point[i].y < minp.y )  minp.y = point[i].y;
          if ( point[i].x > maxp.x ) maxp.x = point[i].x;
          if ( point[i].y > maxp.y ) maxp.y = point[i].y;
}
```

Not really necessary

Used here as a regular **reduction**

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: omp_orig

- Allows initial value to depend on the original list item

```
class V {
    float *p;
    int n;

public:
    V( int _n ) : n(_n) {  p = new[n] float(); }
    V( const Z & m ) : n (m.n) { p = new[n] float(); }
    ~V() { delete[] p; }

    V& operator+= ( const V & );

   #pragma omp declare reduction( + : V : omp_out += omp_in ) \
        initializer(omp_priv(omp_orig))
};
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

User-defined reductions: other rules

- UDRs have scope
  - can’t be redeclared in the same scope
    - except identical declarations
  - reduction clause allows reduction operators to be qualified
    - to specify a class, namespace, …
- UDRs follow base language access rules
  - e.g., private UDRs can only be called from within class methods
- UDRs are inherited
  - the reduction operator in reduction clauses with variables of derived classes

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Some C++ examples

```
#pragma omp declare reduction( + : std::vector<int> : \
     std::transform (omp_out.begin(), omp_out.end(), \
                     omp_in.begin(), omp_in.begin(),plus<int>()))

#pragma omp declare reduction( merge : std::vector<int> : \
     omp_out.insert(omp_out.end(), omp_in.begin(), omp_in.end()))

#pragma omp declare reduction( merge : std::list<int> : \
     omp_out.merge(omp_in))
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD Support: motivation

- Provides a portable high-level mechanism to specify SIMD parallelism
  - Heavily based on Intel's SIMD directive
- Two main new directives
  - To SIMDize loops
  - To create SIMD functions

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD loops: syntax

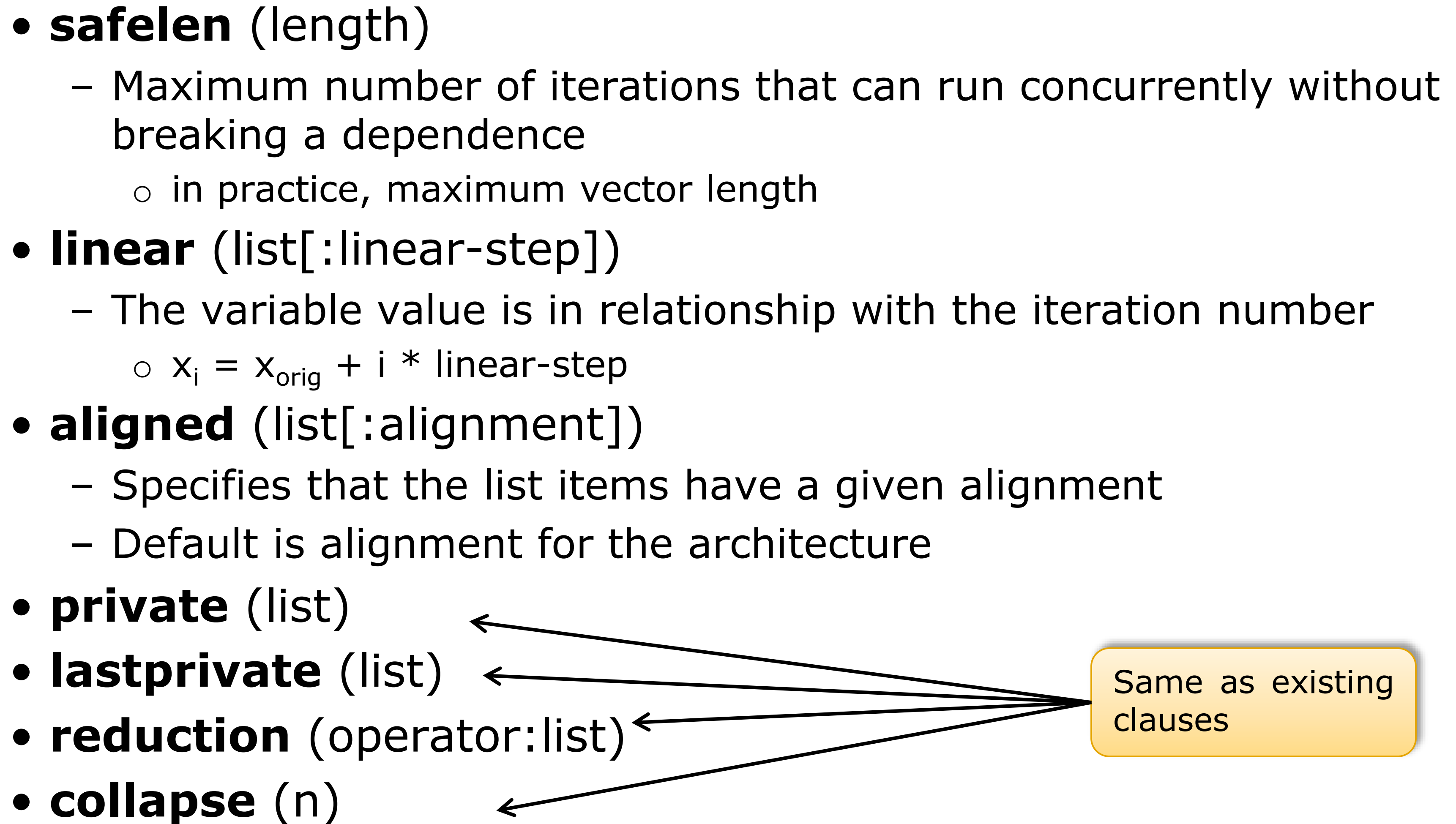
#pragma omp simd [*clauses*]
for-loop

!$omp simd [*clauses*]
do-loops
*[***!$omp end simd***]*

- Loop has to be in “Canonical loop form”
  - as do/for worksharing

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD loop clauses



Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD loop example

```
double pi()
{
    double pi = 0.0;
    double t;
#pragma omp simd private(t) reduction(+:pi)
    for (i=0; i<count; i++) {
        t = (double)((i+0.5)/count);
        pi += 4.0/(1.0+t*t);
    }
    pi /= count
    return pi;
}
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD functions: Syntax

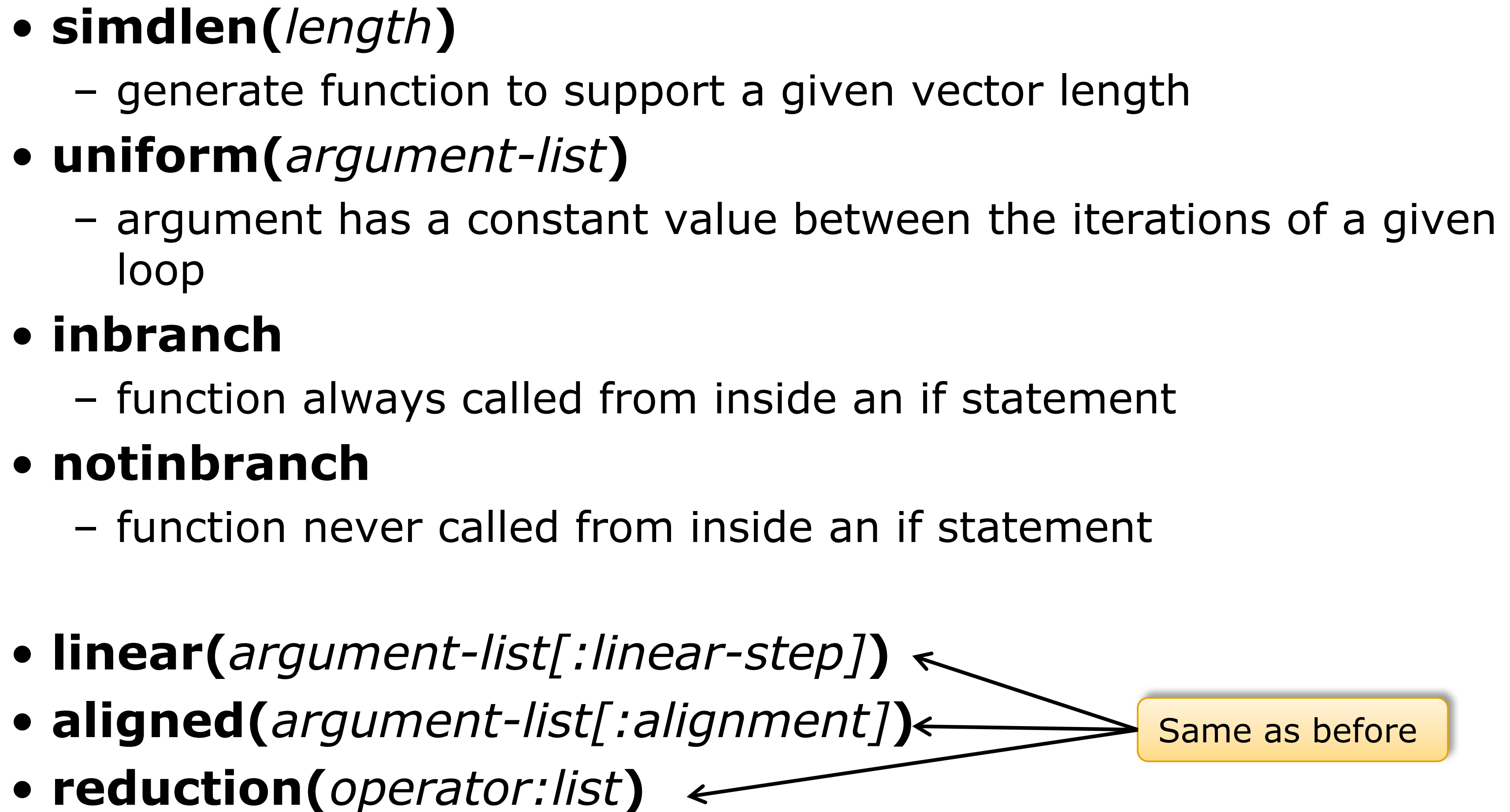
#pragma omp declare simd ***[****clauses****]***
[#pragma omp declare simd ***[****clauses****]]***
function definition or declaration

!$omp declare simd (*function-or-procedure-name***)** ***[****clauses****]***

- Instructs the compiler to
  - generate a SIMD-enabled version(s) of a given function
  - that a SIMD-enabled version of the function is available to use from a SIMD loop

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD functions: clauses



Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD combined constructs

- Worksharing + SIMD

```
#pragma omp for simd [clauses]
!$omp do simd [clauses]
[!$omp end do simd]
```

  - First vectorize the loop, then distribute the resulting iterations among threads

- Parallel + worksharing + SIMD

#pragma omp parallel for simd *[clause[[,] clause] ...]*

!$omp parallel do simd *[clause[[,] clause] ...]*
!$omp end parallel do simd

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

SIMD functions example

```
#pragma omp simd notinbranch
float min(float a, float b) {
    return a < b ? a : b;
}

#pragma omp simd notinbrach
float distsq(float x, float y) {
    return (x - y) * (x - y);
}

#pragma omp parallel for simd
    for (i=0; i<N; i++)
        d[i] = min(distsq(a[i], b[i]), c[i]);
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup

- Allows to logically group tasks together for
  - Synchronization
  - Cancellation

- Solves long standing complain of not being able to wait for a task nest

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup syntax

```
#pragma omp taskgroup
        structured-block
```

```
!$omp taskgroup
structured-block
!$omp end taskgroup
```

- Implies a wait at the end of the region on
  - all child tasks created in the taskgroup
  - their descendants

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup vs Taskwait

```
#pragma omp task {}                 // T1
#pragma omp task                    // T2
{
     #pragma omp task {}            // T3
}
#pragma omp task {}                 // T4

#pragma omp taskwait
```

Only T1, T2 & T4 are guaranteed to have finished here

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup semantics

```
#pragma omp task {}                // T1
#pragma omp taskgroup
{
  #pragma omp task                 // T2
  {
     #pragma omp task {}           // T3
  }
  #pragma omp task {}              // T4
}
```

Only T2, T3 & T4 are guaranteed to have finished here

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup example

```
int count;

void count_tree( node * n )  {
      #pragma omp atomic
             count++;

      if ( n->left )
      #pragma omp task
            count_tree(n->left);
      if ( n -> right )
      #pragma omp task
            count_tree(n->right);
}

node * root = generate_tree();
#pragma omp parallel sections
{
     #pragma omp taskgroup
          count_tree(root);

     printf("Total count: %d\n",count);
}
```

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Taskgroup example

```
int count;

void count_tree( node * n )  {
      #pragma omp atomic
              count++;

      if ( n->left )
      #pragma omp task
            count_tree(n->left);
      if ( n -> right )
      #pragma omp task
            count_tree(n->right);
}

node * root = generate_tree();
#pragma omp parallel sections
{
     #pragma omp taskgroup
          count_tree(root);

     printf("Total count: %d\n",count);
}
```

Horrible horrible way to do this! This is just for simplicity.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Task Dependencies: Motivation

- Allows a more unstructured way of expressing task parallelism
- Pontentially allows to remove more expensive synchronizations
  - and have more work “in-flight”
  - no end of the loop synchronizations
- Several research projects have used successfully to improve linear algebra algorithms for large core counts

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Task Dependencies: Syntax

- New clause to task construct:
  depend(*dependence-type* : *list*)

- dependence-type being one of:
  - **in**
  - **out/inout**

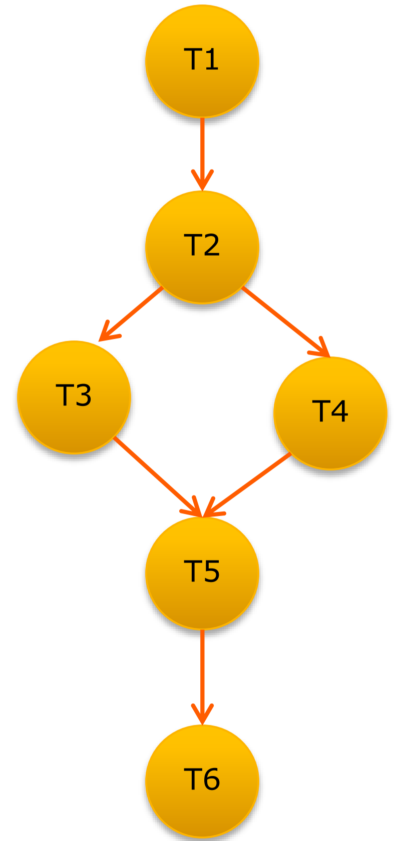
- Dependences are constructed in serial order based on the specified data relationships
  - in waits for previous out
  - out/inout wait for previous out/inout and all previous in
  - no real constraint on what the task does
  - only between sibling tasks

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Task Dependencies

```
int a;
int &b = a;

#pragma omp task depend(in:a) {}          // T1
#pragma omp task depend(out:b) {}         // T2
#pragma omp task depend(in:a) {}          // T3
#pragma omp task depend(in:b) {}          // T4
#pragma omp task depend(inout:a) {}       // T5
#pragma omp task depend(in:a) {}          // T6
```

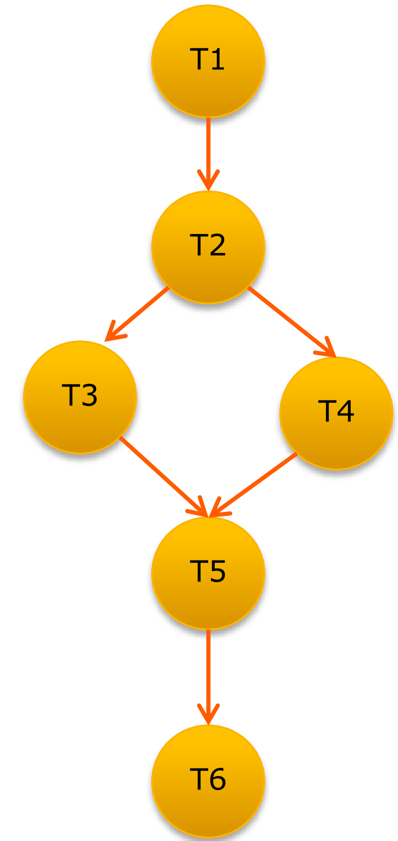


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Task dependencies

```
int a;
int &b = a;

#pragma omp task depend(in:a) {}          // T1
#pragma omp task depend(out:b) {}         // T2
#pragma omp task depend(in:a) {}          // T3
#pragma omp task depend(in:b) {}          // T4
#pragma omp task depend(inout:a) {}       // T5
#pragma omp task depend(in:a) {}          // T6
```



Impossible to build back arches and have deadlocks

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Array sections

- OpenMP extends array subscript notation in C/C++ to allow array sections
  - [*lower bound* : *length*]
    - specifies a section of length elements starting at lower bound
  - if no lower bound is specified defaults to zero
  - if no length is specified defaults to the remaining elements of that dimension for the array
    - only can be omitted if the size of the dimension is known

```
int a[10], *b;

a[:]    //legal
b[:]    // illegal
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

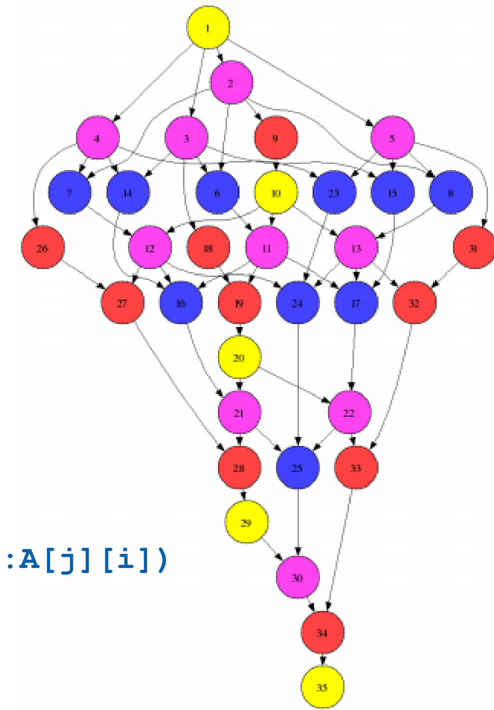
Array sections

- Array sections can be used only in
  - The **depend** clause of the **task** construct
    - Sections cannot partially overlap
  - The **map** clause from the **target** and **target data** constructs
  - Other clauses may allow them in the future
    - probably not in the 4.0 timeframe

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Task dependencies

```
void blocked_cholesky( int NB, float A[NB][NB] ) {
   int i, j, k;
   for (k=0; k<NB; k++) {
     #pragma omp task depend(inout:A[k][k])
        spotrf (A[k][k]) ;
     for (i=k+1; i<NT; i++)
       #pragma omp task depend(in:A[k][k]) depend(inout:A[k][i])
          strsm (A[k][k], A[k][i]);
       // update trailing submatrix
       for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
           #pragma omp task depend(in:A[k][i],A[k][j]) depend(inout:A[j][i])
              sgemm( A[k][i], A[k][j], A[j][i]);
         #pragma omp task depend(in:A[k][i]) depend(inout:A[i][i])
            ssyrk (A[k][i], A[i][i]);
       }
   }
}
```



* image from BSC

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

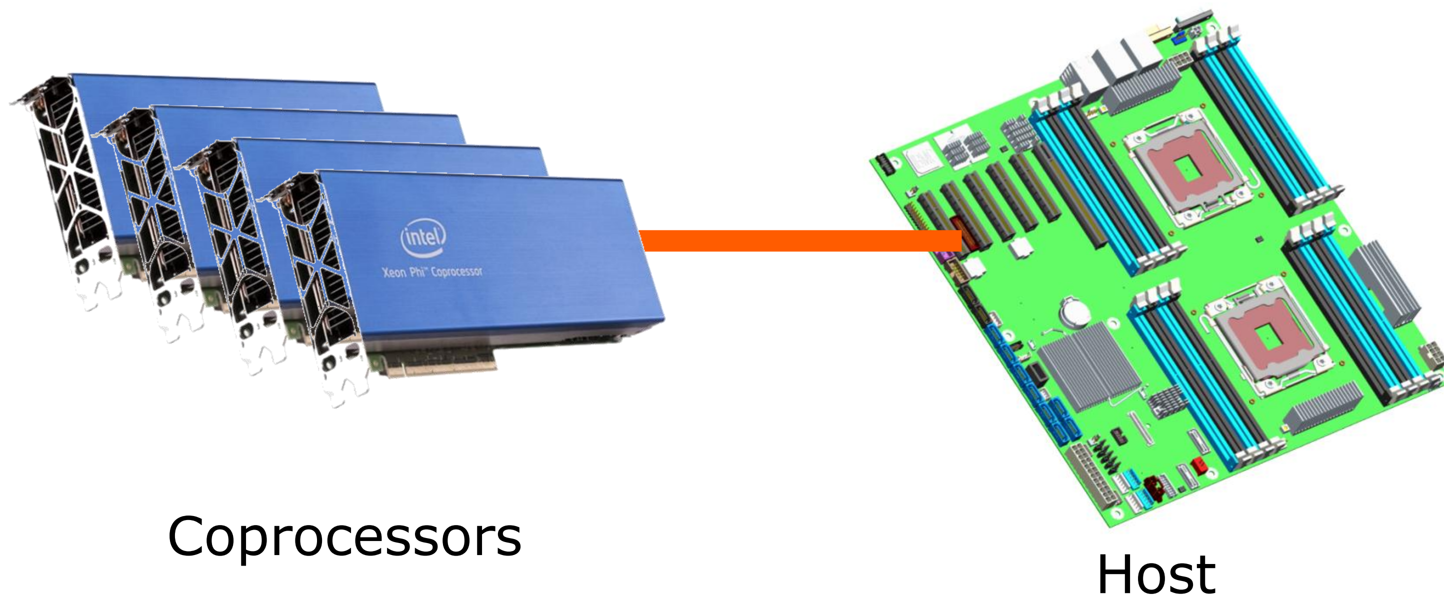
Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Device Model

- OpenMP 4 will support accelerators and coprocessors
- Device model:
  - One host
  - Multiple accelerators/coprocessors of the same kind



Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Terminology

- Device:
  an implementation-defined (logical) execution unit
- League:
  the set of threads teams created by a `teams` construct
- Contention group:
  threads of a team in a league and their descendant threads
- Device data environment:
  Data environment as defined by `target data` or `target`
  constructs
- Mapped variable:
  An original variable in a (host) data environment with a
  corresponding variable in a device data environment
- Mapable type:
  A type that is amenable for mapped variables.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

`target declare` Clauses

- **C/C++**

  #pragma omp declare target *new-line*

  [function-definition-or-declaration]

  #pragma omp end declare target *new-line*

- **Fortran**

  !$omp declare target *[(proc-name-list | list)] new-line*

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

`target` Clauses

#pragma omp target *[clause[[,] clause],...] new-line*
structured-block

Clauses: **device(***scalar-integer-expression***)**
map(alloc | to | from | tofrom: *list***)**
if(*scalar-expr***)**

#pragma omp target data *[clause[[,] clause],...] new-line*
structured-block

Clauses: **device(***scalar-integer-expression***)**
map(alloc | to | from | tofrom: *list***)**
if(*scalar-expr***)**

#pragma omp target update *[clause[[,] clause],...] new-line*

Clauses: **to(** *list* **)**
from(*list* **)**
device(*integer-expression* **)**
if(*scalar-expression* **)**

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

OpenMP* Data Environment Examples

```
#pragma omp target map(to:b[0:count])) map(to:c,d) map(from:a[0:count])
  {
#pragma omp parallel for
    for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
    }
  }
```

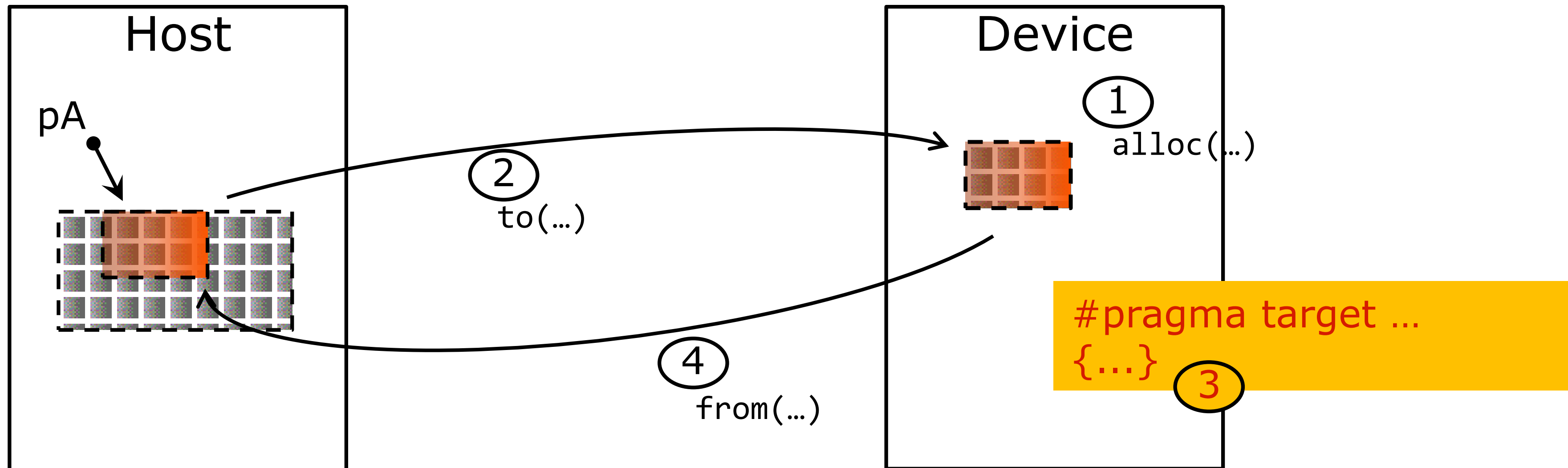
```
#pragma omp target data device(0) map(alloc:tmp[0:N]) map(to:input[:N])) map(from:result)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    do_some_other_stuff_on_host();

#pragma omp target device(0)
#pragma omp parallel for reduction(+:result)
    for (i=0; i<N; i++)
      result += final_computation(tmp[i], i)
  }
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Execution Model



- The `target` construct transfers the control flow to the target device
  - The transfer clauses control direction of data flow
  - Array notation is used to describe array length
- The `target data` construct creates a *scoped* device data environment
  - The transfer clauses control direction of data flow
  - Device data environment is valid through the lifetime of the `target data` region
- Use `target update` to request data transfers from within a `target data` region

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

`team` Constructs

#pragma omp team *[clause[[,] clause],...] new-line*
structured-block

Clauses:
num_teams(*integer-expression* **)**
num_threads(*integer-expression* **)**
default(shared | none)
private(*list* **)**
firstprivate(*list* **)**
shared(*list* **)**
reduction(*operator* **:** *list* **)**

- If specified, a **teams** construct must be contained within a **target** construct. That **target** construct must contain no statements or directives outside of the **teams** construct.
- **distribute**, **parallel**, **parallel loop**, **parallel sections**, and **parallel workshare** are the only OpenMP constructs that can be closely nested in the **teams** region.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Distribute Constructs

#pragma omp distribute *[clause[[,] clause],...] new-line*

for-loops

Clauses: **private(** *list* **)**
firstprivate(*list* **)**
collapse(*n* **)**
dist_schedule(*kind[, chunk_size]* **)**

- A **distribute** construct must be closely nested in a **teams** region.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Examples

```
#pragma omp target device(0)
#pragma omp teams num_teams(60)  num_threads(4) // 60 physical cores, 4 h/w threads each
{
#pragma omp distribute  // this loop is distributed across teams
  for (int i = 0; i < 2048; i++) {
#pragma omp parallel for // loop is executed in parallel by all threads (4) of the team
    for (int j = 0; j < 512; j++) {
#pragma omp simd // create SIMD vectors for the machine
      for (int k=0; k<32; k++) {
        foo(i,j,k);
      }
    }
  }
}
```


Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Execution Environment Routines and ENV

- **void omp_set_default_device(int** *device_num* **)**
- **int omp_get_default_device(void)**
- **int omp_get_num_devices(void);**
- **int omp_get_num_teams(void)**
- **int omp_get_team_num(void);**

- **OMP_DEFAULT_DEVICE ENV Variable**
  - The **OMP_DEFAULT_DEVICE** environment variable sets the device number to use in target constructs by setting the initial value of the *default-device-var* ICV.
  - The value of this environment variable must be a non-negative integer value.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Agenda

- User-defined Reductions
- Support for SIMD Parallelism
- Task Extensions
- Support for Accelerators and Coprocessors
- Thread-affinity Support

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

OpenMP Affinity

- OpenMP 3.1 introduced a thread binding interface
- OpenMP 4 extends the interface to allow for full thread affinity
  - Vendor-agnostic affinity settings (e.g. no more KMP_AFFINITY)
  - Easier interaction between environment and OpenMP runtime
- Terminology:
  - Place: unordered set of processors
  - Place list: ordered list of available places for execution
  - Place partition: Contiguous interval in the place list
- Policies / affinity types:
  - Master: keep worker threads in the same place partition as the master thread
  - Close: keep worker threads “close” to the master thread in contiguous place partitions
  - Spread: create a sparse distribution of worker threads across the place partitions

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

OpenMP Affinity

- Additional clause for parallel regions: proc_bind(*affinity-type*)

- Environment variables control the affinity settings:
  - OMP_PROC_BIND
    e.g., export OMP_PROC_BIND=“spread,spread,close”
  - OMP_PLACES
    e.g., export OMP_PLACES=“{0,1,2,3},{4,5,6,7},{8:4},{12:4}”

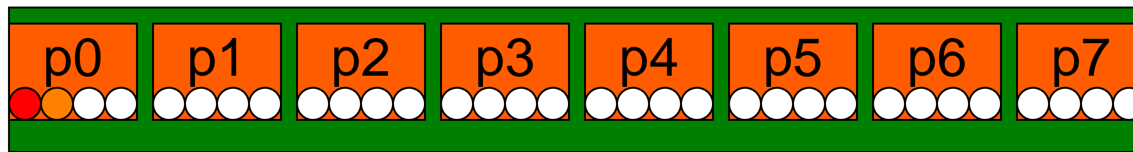
- Places are system-specific and are not defined by OpenMP

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

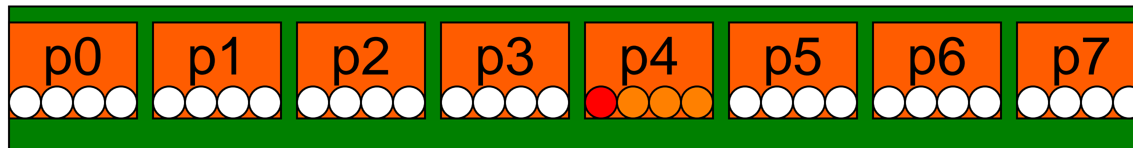
Examples: `master`

- For best data locality
  - select OpenMP threads in the same place as the master

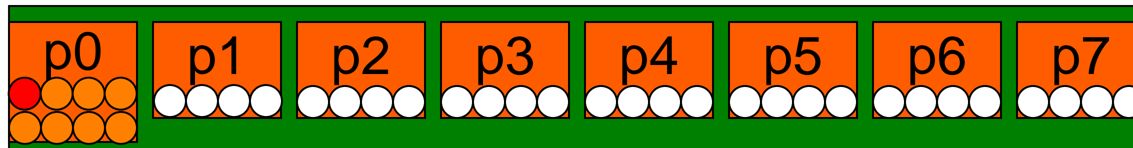
- Examples
  - master 2*



  - master 4



  - master 8



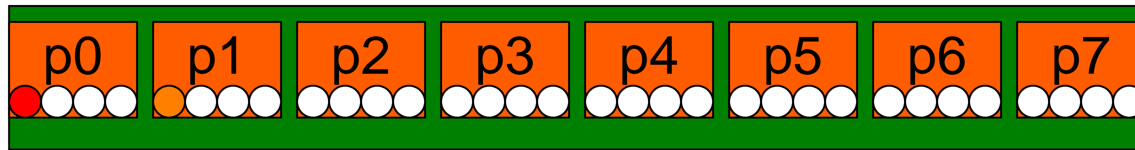
master worker partition

Software & Services Group, Developer Relations Division
Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

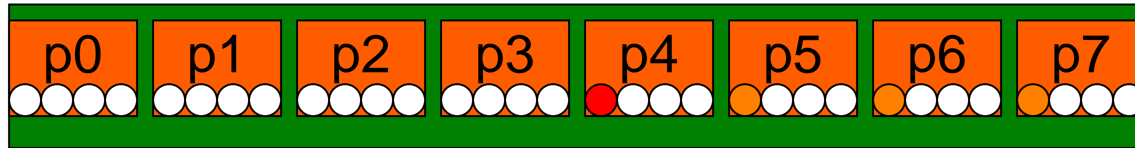
Example: `close`

- For data locality, load-balancing, and more dedicated-resources
  - select OpenMP threads near the place of the master
  - wrap around once each place has received one OpenMP thread

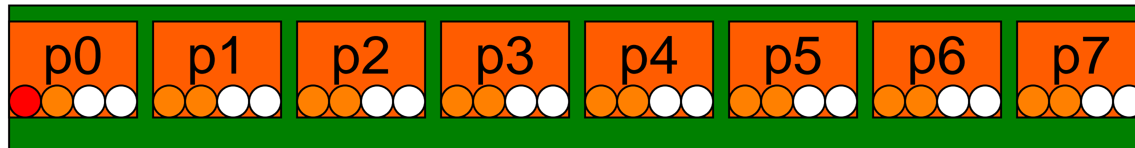
- Examples
  - close 2*



  - close 4



  - close 8



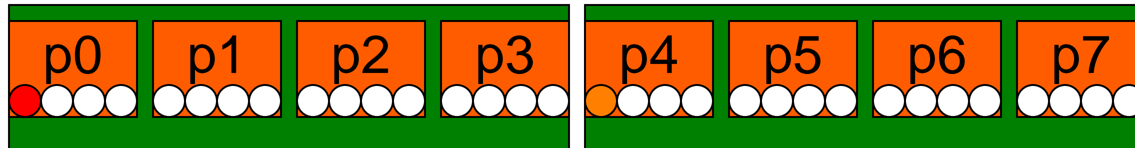
master worker partition

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

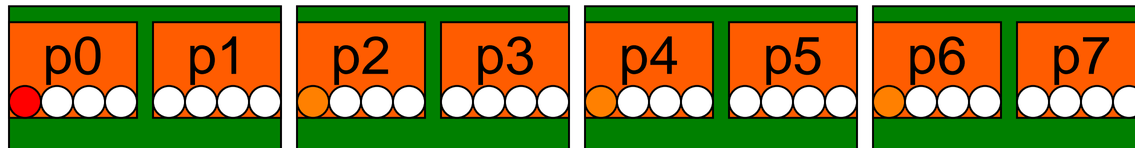
Example: spread

- For load balancing, most dedicated hardware resources
  - spread OpenMP threads as evenly as possible among places
  - create sub-partition of the place list
    - subsequent threads will only be allocated within sub-partition

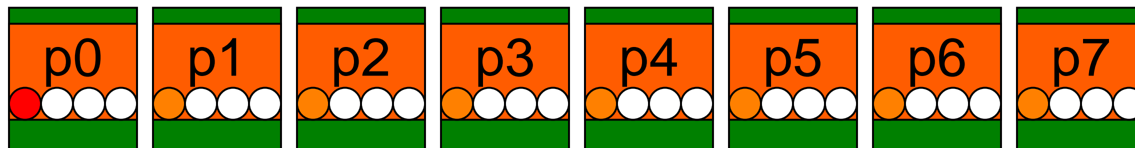
- Examples
  - spread 2*



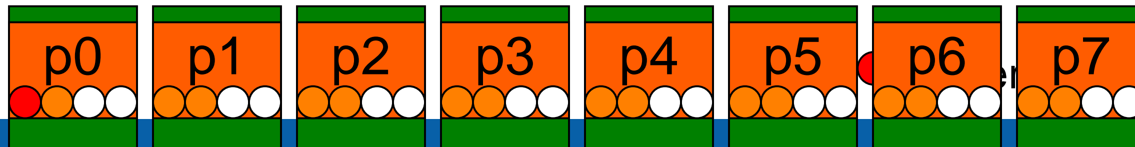
  - spread 4



  - spread 8



  - spread 16



partition

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Questions?

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.

Copyright© 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.