Introduction to OpenMP 4.0

Yonghong Yan
University of Houston
SC13, Denver CO

Acknowledgements:
OpenMP Language committee and subcommittees
Ruud van der Pas (Oracle), Barbara M. Chapman (UH)

What is OpenMP

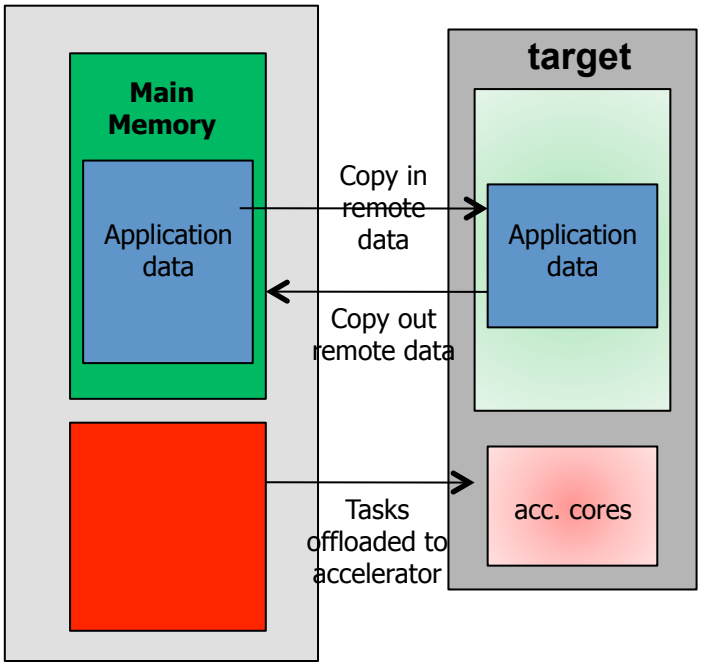
- Standard **API** to write shared memory parallel applications in C, C++, and Fortran
  - Compiler directives, Runtime routines, Environment variables
- OpenMP Architecture Review Board (ARB)
  - Maintains OpenMP specification
  - Permanent members
    - AMD, CAPS-Entreprise, Convey, Cray, Fujitsu, HP, IBM, Intel, NEC, NVIDIA, PGI, Oracle, Red Hat, Texas Instruments
  - Auxiliary members
    - ANL, ASC/LLNL, BSC, cOMPunity, EPCC, LANL, NASA, ORNL, RWTH Aachen University, Sandia, TACC, UH
  - http://www.openmp.org

OpenMP 4.0

- Released July 2013
  - http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
  - A document of examples is expected to release soon
- Changes from 3.1 to 4.0 (Appendix E.1):
  - *Accelerator: 2.9*
  - *SIMD extensions: 2.8*
  - *Places and thread affinity: 2.5.2, 4.5*
  - *Taskgroup and dependent tasks: 2.12.5, 2.11*
  - *Error handling: 2.13*
  - *User-defined reductions: 2.15*
  - *Sequentially consistent atomics: 2.12.6*
  - *Fortran 2003 support*

Accelerator (2.9): offloading

- Execution Model: Offload data and code to accelerator
- *target* construct creates tasks to be executed by devices
- Aims to work with wide variety of accs
  - GPGPUs, MIC, DSP, FPGA, etc
  - A target could be even a remote node, intentionally




```
#pragma omp target map ( … )
{
    /* it is like a new task
     * executed on a remote device */
{
```

target and map examples

```
void vec_mult(int N)
{
    int i;
    float p[N], v1[N], v2[N];
    init(v1, v2, N);
    #pragma omp target map(to: v1, v2) map(from: p)
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    output(p, N);
}
```

```
void vec_mult(float *p, float *v1, float *v2, int N)
{
    int i;
    init(v1, v2, N);
    #pragma omp target map(to: v1[0:N], v2[:N]) map(from: p[0:N])
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    output(p, N);
}
```

Accelerator: explicit data mapping

- Relatively small number of truly shared memory accelerators so far
- Require the user to explicitly *map* data to and from the device memory
- Use array region

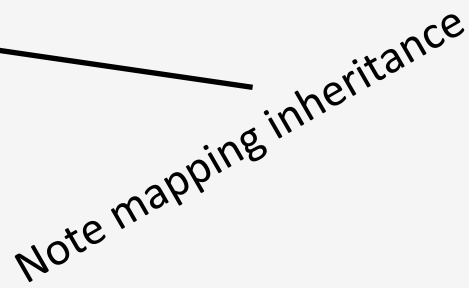
```
long a = 0x858;
long b = 0;
int anArray[100]

#pragma omp target data map(to:a) \\
  map(tofrom:b,anArray[0:64])
{
   /* a, b and anArray are mapped
     * to the device */

  /* work on the device */
#pragma omp target …
  {
        …
   }|
}
/* b and anArray are mapped
  * back to the host */
```

target date example

```
void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);
   #pragma omp target data map(from: p[0:N])
   {
      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
        p[i] = v1[i] * v2[i];
      init_again(v1, v2, N);
      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
        p[i] = p[i] + (v1[i] * v2[i]);
   }
   output(p, N);
}
```



Accelerator: hierarchical parallelism

- Organize massive number of threads
  - teams of threads, e.g. map to CUDA grid/block
- Distribute loops over teams

```
#pragma omp target

#pragma omp teams num_teams(2)
        num_threads(8)
{
    //-- creates a "league" of teams
   //-- only local barriers permitted
#pragma omp distribute
for (int i=0; i<N; i++) {

}

}
```

Only target directive makes it as accelerator region

teams and distribute loop example

```
float dotprod_teams(float B[], float C[], int N, int num_blocks,
  int block_threads)
{
    float sum = 0;
    int i, i0;
    #pragma omp target map(to: B[0:N], C[0:N])
    #pragma omp teams num_teams(num_blocks) thread_limit(block_threads)
      reduction(+:sum)
    #pragma omp distribute
    for (i0=0; i0<N; i0 += num_blocks)
        #pragma omp parallel for reduction(+:sum)
        for (i=i0; i< min(i0+num_blocks,N); i++)
            sum += B[i] * C[i];
    return sum;
}
```

Double-nested loops are mapped to the two levels of thread hierarchy (league and team)

Other directive and features

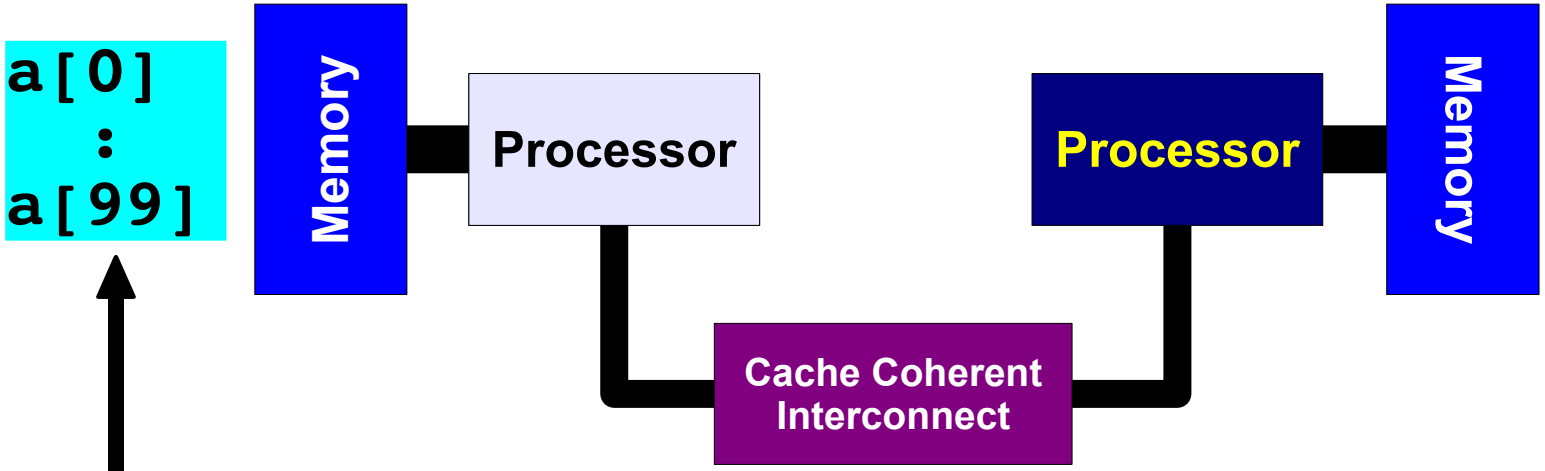
- Declare target
- Target update
- Distribute simd
- Async offloading using task
- Array region
- If clause
- Runtime routines

SIMD loop: 2.8

- omp simd: applied to a loop to indicate that multiple iterations of the loop can be executed concurrently using SIMD instructions
- omp for simd: loop chunked among team of threads and then each chunk is simdized

```
void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);
   #pragma omp target teams map(to: v1[0:N], v2[:N]) map(from: p[0:N])
   #pragma omp distribute parallel for simd
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];
   output(p, N);
}
```

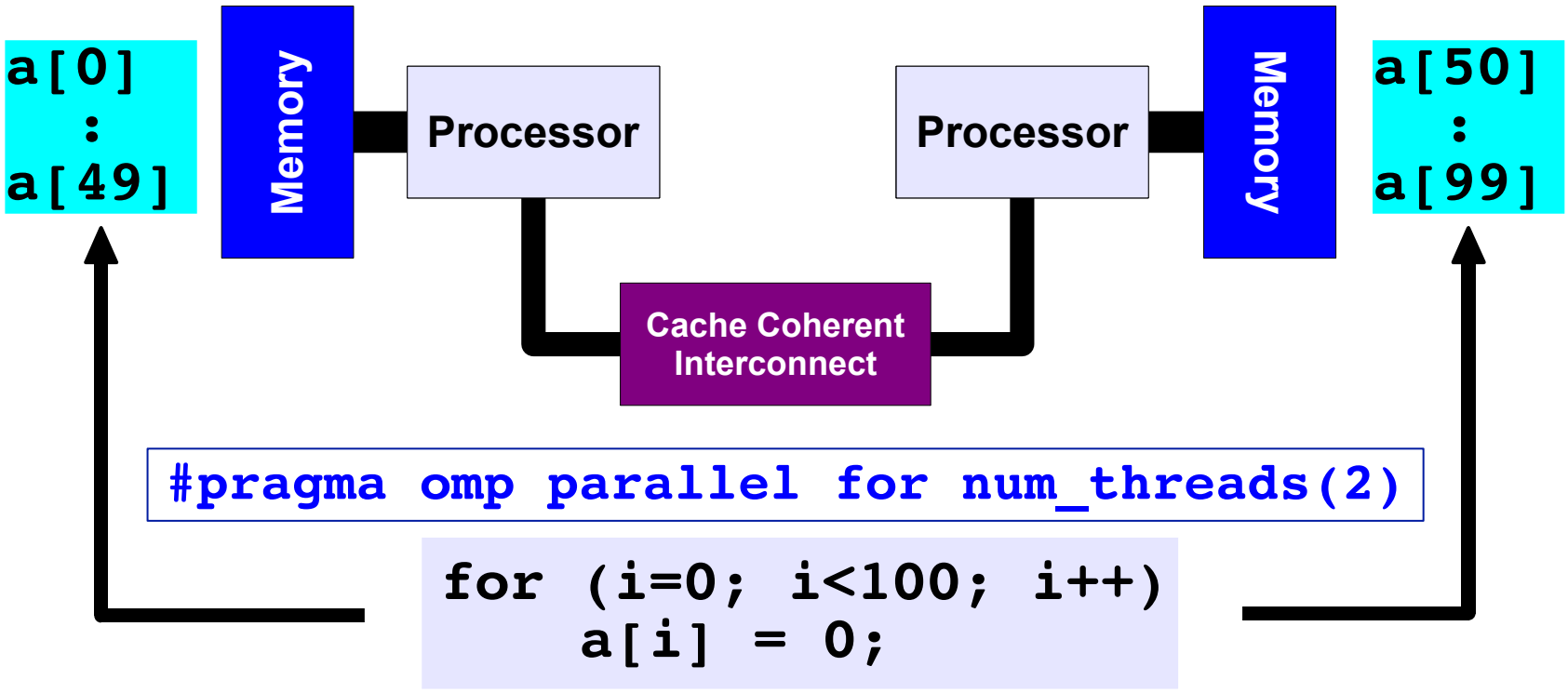
NUMA First-touch placement/1




```
for (i=0; i<100; i++)
    a[i] = 0;
```

First Touch
All array elements are in the memory of the processor executing this thread

NUMA First-touch placement/2



First Touch
Both memories each have "their half" of the array

Places and thread affinity: 4.5, 2.5.2

- **OMP_PLACES** to describe a list of places (4.5) and the ***hardware threads*** of each place
  - OMP_PLACES = "{0,1},{2,3},{4,5},{6,7}”
  - OMP_PLACES = threads | cores | sockets
    - threads: place → hardware thread
    - cores: place → core (may have multiple threads)
    - sockets: place → socket (may have multiple cores)

```
setenv OMP_PLACES threads
setenv OMP_PLACES "threads(4)"
setenv OMP_PLACES "{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}"
setenv OMP_PLACES "{0:4},{4:4},{8:4},{12:4}"
setenv OMP_PLACES "{0:4}:4:4"
```

Places and thread affinity: 2.5.2, 4.5

- **`proc_bind(master|close|spread)`** clause of parallel to specify policy of assigning ***OpenMP threads*** to places (2.5.2)
  - **`master`**: All threads of the team go to the place of the master thread
  - **`close:`** assign threads to place close to the place of parent thread

```
void work2()
{
  #pragma omp parallel num_threads(12) proc_bind(close)
  {
    /* do work here */
  }
}
```

OMP_PLACES = "{0,1},{2,3},{4,5},{6,7}"

  - If master thread is in place {2,3}, then:
    - threads 0-2 execute on the place {2,3}
    - threads 3-5 execute on the place {4,5}
    - threads 6-8 execute on the place {6,7}
    - threads 9-11 execute on the place {0,1}
  - **Place set for each implicit task is still "{0,1},{2,3},{4,5},{6,7}"**

Places and thread affinity: 2.5.2, 4.5

- **`proc_bind(master|close|spread)`** clause of parallel to specify policy of assigning ***OpenMP threads*** to places (2.5.2)
  - **`spread`**: subpartition parent place sets, and then assign threads to place close to the new places

```
void work3()
{
  #pragma omp parallel num_threads(4) proc_bind(spread)
  {
    /* do work here */
  }
}
```

OMP_PLACES = "{0,1},{2,3},{4,5},{6,7}"

```
Place partition of the implicit task of thread 0: "{2,3}"
Place partition of the implicit task of thread 1: "{2,3}"
Place partition of the implicit task of thread 2: "{2,3}"
Place partition of the implicit task of thread 3: "{4,5}"
Place partition of the implicit task of thread 4: "{4,5}"
Place partition of the implicit task of thread 5: "{4,5}"
Place partition of the implicit task of thread 6: "{6,7}"
Place partition of the implicit task of thread 7: "{6,7}"
Place partition of the implicit task of thread 8: "{6,7}"
Place partition of the implicit task of thread 9: "{0,1}"
Place partition of the implicit task of thread 10: "{0,1}"
Place partition of the implicit task of thread 11: "{0,1}"
```

  - If master thread is in place {2,3}, then:
    - threads 0-2 execute on the place {2,3}
    - threads 3-5 execute on the place {4,5}
    - threads 6-8 execute on the place {6,7}
    - threads 9-11 execute on the place {0,1}

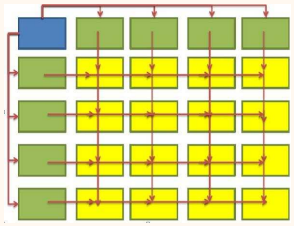
OpenMP 4.0 task extension

- Specifying dependencies of tasks sharing the same parent: 2.11.1.1

  #pragma omp task depend (**out**:t_1, t_2 ,…, t_n) depend (**in**:t_1 ,t_2, …, t_n) depend (**inout**:t_1 ,t_2, …, t_n)

- ```
  #pragma omp taskgroup new-line
    structured-block
  ```
  2.12.5
  - Join all tasks created within the group → x10 finish
  - taskwait vs taskgroup
    - Taskwait only joins the child tasks
    - Taskgroup joins the child and decedent tasks
  ```

in|out|inout task dependency

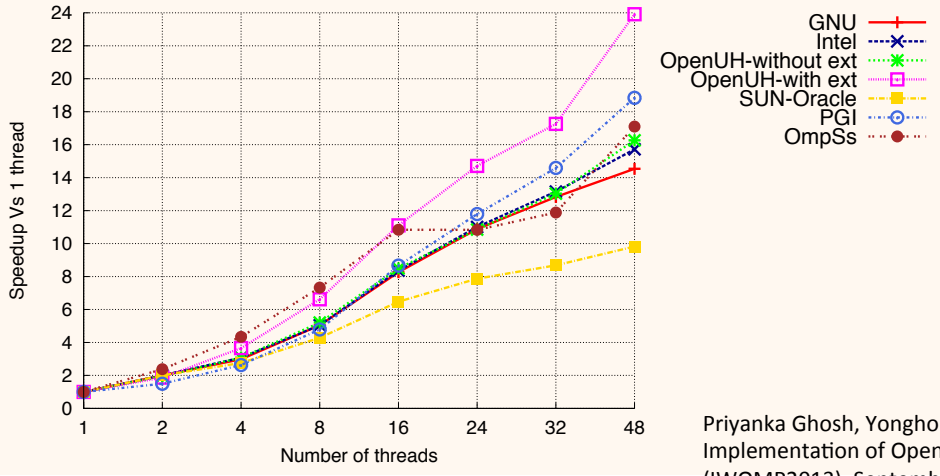


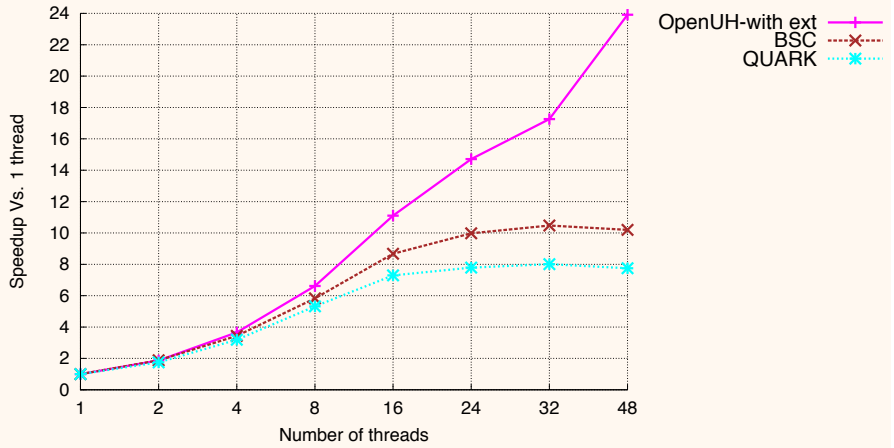
```
1  #pragma omp parallel
2  {
3   #pragma omp master
4   {
5     for ( i=0; i<matrix_size; i++ ) {
6
7     /**** Processing Diagonal block ****/
8     ProcessDiagonalBlock(.....);
9
10    for (i=1;i<M;i++){
11
12    #pragma omp task out(2*i) /** Processing block on column **/
13      ProcessBlockOnColumn(.....);
14
15    #pragma omp task out(2*i+1) /** Processing block on row **/
16      ProcessBlockOnRow(.....);
17      }
18
19 /*** Elimination of Global Synchronization point ********/
20
21 /**** Processing remaining inner block ****/
22    for (i=1;i<M;i++)
23      for (j=1;j<M;j++){
24    #pragma omp task in(2*i) in(2*j+1)
25    ProcessInnerBlock(.....);
26        }
27    #pragma omp taskwait
28      }
```

#pragma omp task depend (out: t1, t2, …) depend (in: t4, t5)

Runtime

- Avoid the use of global locks
- Work with workstealing
- Decentralized dependency setup and resolution

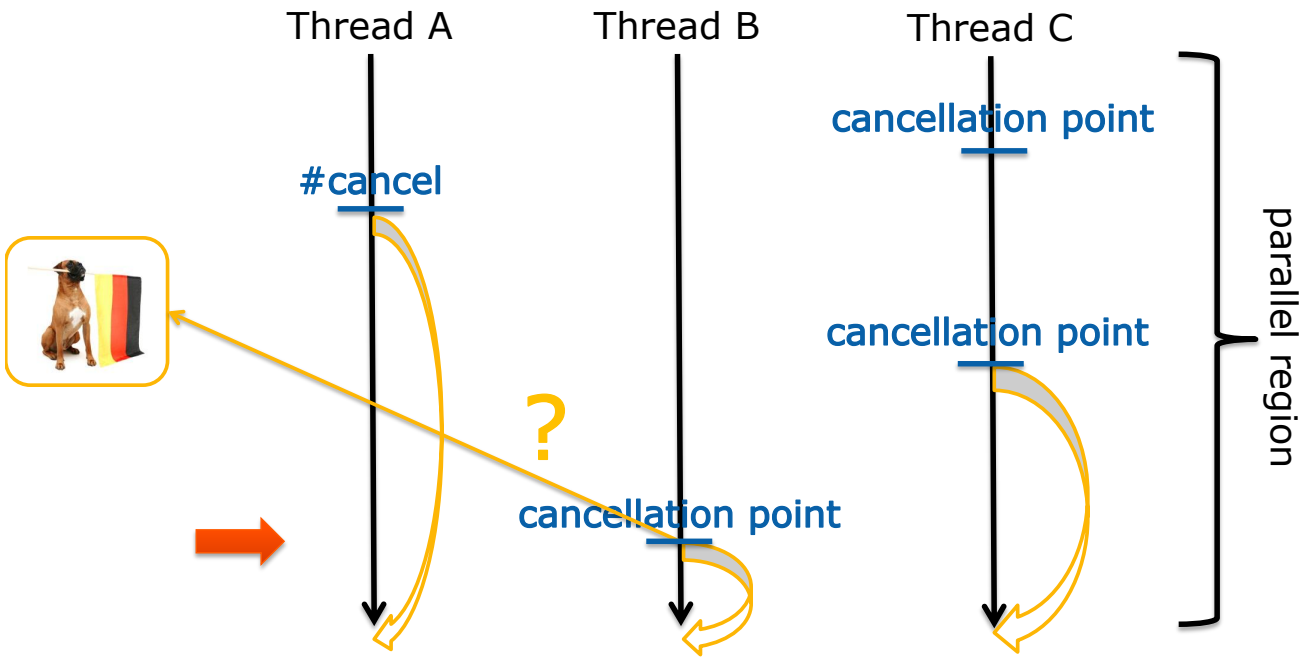




Priyanka Ghosh, Yonghong Yan, Deepak Eachempati and Barbara Chapman, "A Prototype Implementation of OpenMP Task Dependency Support", The 9th International Workshop on OpenMP-(IWOMP2013), September 16-18,2013, Canberra, Australia

Error handling: 2.13

- **`cancel` Directive**
  - **`#pragma omp cancel`** *[clause[[,]clause] ...]*
  - **`!$omp cancel`** *[clause[[,]clause] ...]*
  - Clauses: **`parallel`**, **`sections`**, **`for`**, **`do`**



User defined reduction: 2.15

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

where:

- *reduction-identifier* is either a base language identifier or one of the following operators: **`+`**, **`-`**, **`*`**, **`&`**, **`|`**, **`^`**, **`&&`** and **`||`**
- *typename-list* is list of type names
- *combiner* is an expression
- *initializer-clause* is **`initializer (`** *initializer-expr* **`)`** where *initializer-expr* is **`omp_priv =`** *initializer* or *function-name* **`(`** *argument-list* **`)`**

Atomic extensions: 1.12.6

- capture clause: to allow new atomic update and capture the original or result value, e.g. cas operation
- seq_cst clause: to support sequentially consistent atomic operations, i.e. force a flush

Notes on compiler implementations

- Intel:
  - Has similar offloading (MIC) and SIMD support, lots of materials on Intel website
- ROSE:
  - OpenMP accelerator for GPGPUs
  - see our IWOMP13 paper: www.iwomp13.org
- OpenUH:
  - Task dependency support
  - see our IWOMP13 paper
- OpenACC compiler (PGI, CAPS, OpenUH, etc)
  - Could be easily added to support OpenMP acc
- Ongoing:
  - Oracle, Cray, etc ?