HPC2N, UmU

More on private data

Pedro Ojeda & Joachim Hein

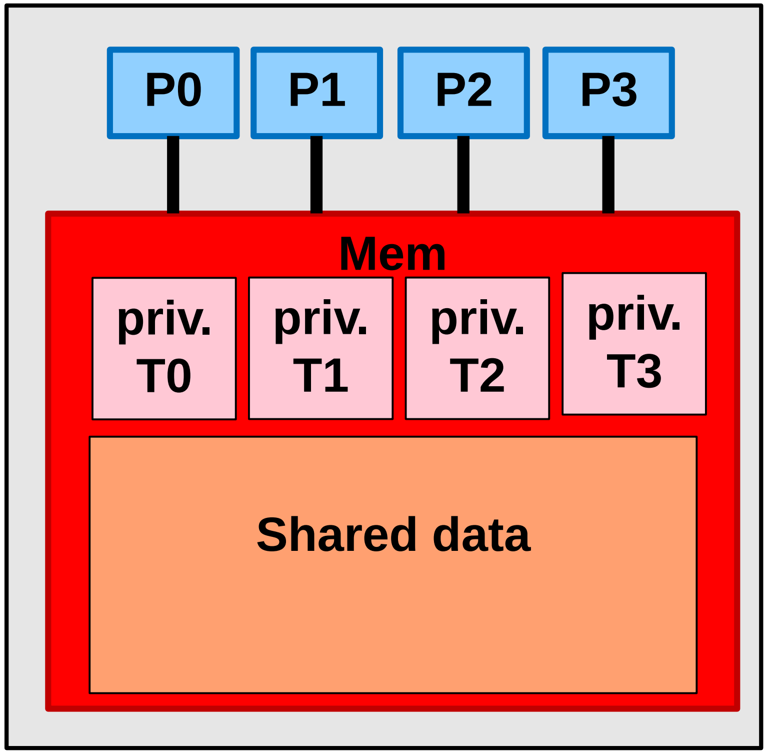
High Performance Computing Center North &
Lund University

Outline

- Special versions of private data
  - `firstprivate`
  - `lastprivate`
  - `reduction`
  - global storage: `threadprivate`

Private and shared data

- In parallel region
  - Shared and private data
- Shared data
  - Unchanged on entry to par. region
  - Survives after end of par. Region
- Private data
  - Each thread: **own private copy**
  - Normally **uninitialised** at begin of parallel region
  - Contents typically lost when parallel region finishes
  - Though connection to values before/after often needed



Clause `firstprivate`

- Private variables are **not** initialised
- Clause `firstprivate`
  - declares variable(s) private
  - initialises each private copy with the value prior to the construct

Example local accumulation:

```
integer lsum=10
!$omp parallel &
!$omp  firstprivate(lsum)

  lsum = lsum &
   + omp_get_thread_num()
  print *, lsum
!$omp end parallel
```

Clause `firstprivate`

- Private variables are **not** initialised
- Clause `firstprivate`
  - declares variable(s) private
  - initialises each private copy with the value prior to the construct

Example local accumulation:

```
int lsum=10;
#pragma omp parallel \
   firstprivate(lsum)
 {
  lsum +=
   omp_get_thread_num();
  printf(”%i\n”, lsum);
 }
```

Fortran-example: Vector norm *private*

$$\sqrt{\sum_i v(i) * v(i)}$$

```
 norm = 0.0

!$omp parallel default(none) \
  shared(vect, norm) private(i, lNorm)
    lNorm = 0.0
    !$omp do
    do i = 0, vleng
        lNorm = lNorm + vect(i)**2
    enddo
    #pragma omp atomic update
    norm += lNorm
!$omp end parallel
  norm = sqrt(norm)
```

Fortran-example: Vector norm *firstprivate*

$$\sqrt{\underset{i}{\text{å}}\, v(i) * v(i)}$$

```
 norm = 0.0
 lNorm = 0.0
!$omp parallel default(none) \
  shared(vect, norm) private(i) firstprivate(lNorm)

    !$omp do
    do i = 0, vleng
        lNorm = lNorm + vect(i)**2
    enddo
    #pragma omp atomic update
    norm += lNorm
!$omp end parallel
  norm = sqrt(norm)
```

C-example: Vector norm *private*

$$\sqrt{\underset{i}{\text{å}}\, v(i) * v(i)}$$

```
 norm = 0.0;

#pragma omp parallel default(none) \
  shared(vect, norm) private(i, lNorm)
  { lNorm = 0.0;
    #pragma omp for
    for (i = 0; i < vleng; i++)
        lNorm += vect[i]*vect[i];
    #pragma omp atomic update
    norm += lNorm;
  }
  norm = sqrt(norm);
```

C-example: Vector norm
firstprivate

$$\sqrt{\sum_i v(i)*v(i)}$$

```
 norm = 0.0;
 lNorm = 0.0;
#pragma omp parallel default(none) \
  shared(vect, norm) private(i) firstprivate(lNorm)
  {
    #pragma omp for
    for (i = 0; i < vleng; i++)
        lNorm += vect[i]*vect[i];
    #pragma omp atomic update
    norm += lNorm;
  }
  norm = sqrt(norm);
```

Clause `lastprivate`

- Clause `lastprivate`
  - Use with **loop**, **sections**
  - Variable private
  - In the end: assigns value from last iteration or section
  - Undefined if not set in last iteration/section
- Variables can be both: `firstprivate` & `lastprivate`

Example:

```
integer i, a
!$omp parallel do &
!$omp  lastprivate(a)
 do i=1, 100
     a=i+1
     func(a)
enddo
print *,"a=", a
   ! this prints: a=101
```

Clause `lastprivate`

- Clause `lastprivate`
  - Use with **loop**, **sections**
  - Variable private
  - In the end: assigns value from last iteration or section
  - Undefined if not set in last iteration/section
- Variables can be both: `firstprivate` & `lastprivate`

Example:

```
integer i, a
#pragma omp parallel \
   for lastprivate(a)
 for (i=0; i<100; i++)
  { a=i+1;
    func(a);
  }
  printf(”a=%i\n”, a);
  // this prints: a=100
```

Reduction variables

- Frequently needed: Reduction of private variables
  - E.g.: Averages of array values, scalar products
- We have done this before: example vector norm
  - used `atomic` to protect the update
- For a `reduction`, we have to specify
  - operation, e.g.: addition, multiplication, or, …
  - one or more variables
  - A construct can have more than one `reduction`

Behavior of reduction

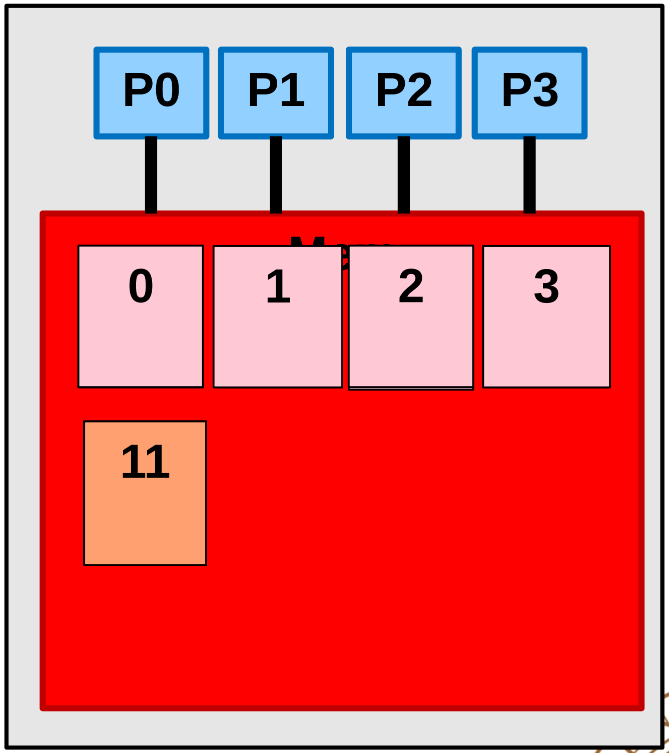
- The basic syntax

  `reduction(` *`operator`* `:` *`variable_list`* `)`

- Variables specified in `reduction`:
  - Private copy per thread
    - Initialised with default matching on operator
  - At the end of construct (e.g. parallel region)
    - Value prior to construct combined with private copies
    - Using the specified operator for combining values
    - New value available after the construct

Example:
Memory movements for reduction

```
int b;
b=5;
#pragma omp parallel \
  reduction(+:b)
{
 b+=omp_get_thread_num();
}
printf("%i\n", b);
```



Example:
Memory movements for reduction

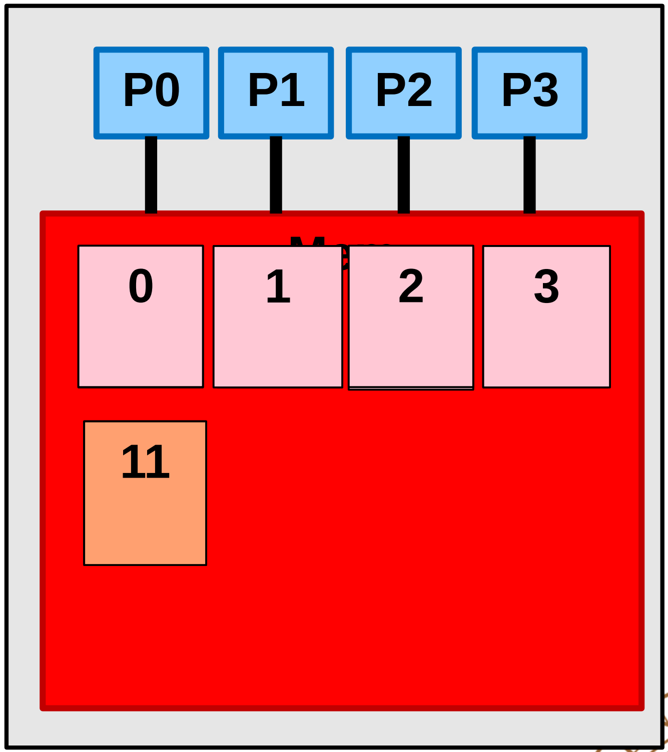
```
integer b
b=5

!$omp parallel &
!$omp  reduction(+:b)

 b=b+omp_get_thread_num()

!$omp end parallel

print *, b
```



Fortran example: Vector norm *atomic update*

$$\sqrt{\text{å}_{i} v(i) * v(i)}$$

```
 norm = 0.0
 lNorm = 0.0
!$omp parallel default(none) &
!$omp shared(vect,norm) private(i) firstprivate(lNorm)
!$omp do
    do i = 1, vleng
         lNorm = lNorm + vect(i)**2      ! priv. copy
    enddo
!$omp atomic update
    norm = norm + lNorm
!$omp end parallel                     ! comb. copies
  norm = sqrt(norm)                    ! master copy
```

Fortran example: Vector norm *reduction*

$$\sqrt{\sum_i v(i)*v(i)}$$

```
 norm = 0.0                                      ! master copy
                                                 ! lNorm gone
!$omp parallel default(none) &
!$omp shared( vect ) reduction( + : norm ) private(i)
!$omp do                                         ! priv. copy=0
    do i = 1, vleng
        norm = norm + vect(i)**2                 ! priv. copy
    enddo


!$omp end parallel                               ! comb. copies
  norm = sqrt(norm)                              ! master copy
```

Fortran example: Vector norm *reduction, parallel do*

$$\sqrt{\text{å}_{i} v(i) * v(i)}$$

```
 norm = 0.0                                  ! master copy

!$omp parallel do default(none) &
!$omp shared( vect ) reduction( + : norm )

   do i = 1, vleng
       norm = norm + vect(i)**2              ! priv. copy
   enddo


!$omp end parallel do
  norm = sqrt(norm)                          ! master copy
```

Example: Vector norm *atomic update*

$$\sqrt{\sum_i v(i) * v(i)}$$

```
 norm = 0.0;
 lNorm = 0.0;
#pragma omp parallel default(none) \
  shared(vect, norm) private(i) firstprivate(lNorm)
  {
    #pragma omp for
    for (i = 0; i < vleng; i++)
        lNorm += vect[i]*vect[i];
    #pragma omp atomic update
    norm += lNorm;
  }
  norm = sqrt(norm);
```

Example: Vector norm *reduction*

$$\sqrt{\sum_{i} v(i) * v(i)}$$

```
 norm = 0.0;                             // master copy
                                         // lNorm gone!
#pragma omp parallel default(none) \
  shared( vect ) reduction( + : norm ) private(i)
  {                                      // priv. copy: 0
    #pragma omp for
    for (i = 0; i < vleng; i++)
         norm += vect[i]*vect[i];        // private copy


  }                                      // comb. copies
  norm = sqrt(norm);                     // master copy
```

Example: Vector norm *reduction, parallel for*

$$\sqrt{\sum_{i} v(i) * v(i)}$$

```
 norm = 0.0;                                 // master copy

#pragma omp parallel for default(none) \
  shared( vect ) reduction( + : norm )


    for (i = 0; i < vleng; i++)
         norm += vect[i]*vect[i];    // private copy



  norm = sqrt(norm);                         // master copy
```

Supported operators and initial values for reduction in Fortran (OpenMP 3.0)

Name	Symbol	Initial Value of local copy
add	`+`	`0`
multiply	`*`	`1`
subtract	`-`	`0`
logical AND	`.and.`	`.true.`
logical OR	`.or.`	`.false.`
EQUIVALENCE	`.eqv.`	`.true.`
NON-EQUIV.	`.neqv.`	`.false.`
maximum	`max`	smallest representable number
minimum	`min`	largest representable number
bitwise AND	`iand`	All bits on
bitwise OR	`ior`	`0`
bitwise XOR	`ieor`	`0`

Supported operators and initial values for reduction variables in C (OpenMP 3.0)

Name	Symbol	Initial Value of local copy
add	`+`	`0`
multiply	`*`	`1`
subtract	`-`	`0`
bitwise AND	`&`	`~0`
bitwise OR	`|`	`0`
bitwise XOR	`^`	`0`
logical AND	`&&`	`1`
logical OR	`||`	`0`

Restrictions and comments on reduction

- Arrays are unsupported as reduction variables in C/C++
- No pointer or reference types in C/C++

- Fortran ALLOCATABLE must be allocated at the beginning of construct and must not be de-allocated during construct
- No Fortran pointers or assumed size arrays

- No order of threads is specified
  - Repeated runs are typically not bit-identically (common issue in parallel computing)
  - This is a *race condition*, which is typically tolerated!

- OpenMP 4.0: declare your own reductions

User defined reductions

- Allows to define you own reduction operations

- Particularly useful with derived data types, examples:
  - C/C++: struct
  - Fortran: type

- You need:
  - Combiner: combines thread private results to final result
  - Initialiser: initialise private contributions at outset

A case study:
Maximum value and its postion

- Problem:
  - We have a large array
  - Determine the maximum value
  - Location of the maximum in the array

- Parallelisation
  - Assign portion of array to each thread
  - Thread determines maximum and position in its part
  - User defined reduction to determine final result

Example in Fortran

- Consider the type:

```
type :: mx_s
      real value
      integer index
end type
```

- Declare a reduction operator named `maxloc`:

```
!$omp declare reduction(maxloc: mx_s: &
!$omp  mx_combine(omp_out, omp_in) ) &
!$omp  initializer(mx_init(omp_priv, omp_orig))
```

- The operation can be triggered by the name “`maxloc`”
- Utilises subroutine `mx_combine and mx_init`
- Acts on object of type: `mx_s`

The intitialiser

- Subroutine or assignment statement – here: subroutine
- Acts on variables:

  `omp_priv`: reference to variable to be initialised

  `omp_orig`: reference to original variable prior to construct
- Example: Initialise from value prior to construct:

```
subroutine mx_init(priv, orig)
    type(mx_s), intent(out) :: priv
    type(mx_s), intent(in) :: orig
    priv%value = orig%value
    priv%index = orig%index
  end subroutine mx_init
```

The combiner

- Subroutine or assignment statement – here: subroutine
- Acts on variables:

  `omp_in`: reference to contribution from thread

  `omp_out`: reference to combined result
- Example: replace if contribution is larger

```
subroutine mx_combine(out, in)
    type(mx_s), intent(inout) :: out
    type(mx_s), intent(in) :: in
    if ( out%value < in%value ) then
       out%value = in%value
       out%index = in%index
    endif
  end subroutine mx_combine
```

How to use it:

- You can use it similar to predefined reductions:

```
mx%value = val(1)
mx%index = 1

!$omp parallel do reduction(maxloc: mx)
do i=2, count
   if (mx%value < val(i)) then
      mx%value = val(i)
      mx%index = i
   endif
enddo
```

- Easily readable code
- Similar to what one would do in serial programming

Example in C

- Consider the type:

```
struct mx_s {
        float value;
        int index:
 };
```

- Declare a reduction operator named `maxloc`:

```
# pragma omp declare reduction(maxloc:         \
   struct mx_s: mx_combine(&omp_out, &omp_in)) \
   initializer(mx_init(&omp_priv, &omp_orig))
```

- The operation can be triggered by the name “`maxloc`”
- Utilises subroutine `mx_combine and mx_init`
- Acts on object of type: `mx_s`

The intitialiser in C

- Expression – here: implemented with a function
- Acts on variables:

  `omp_priv`: reference to variable to be initialised

  `omp_orig`: reference to original variable prior to construct
- Example: Initialise from value prior to construct:

```
void mx_init(struct mx_s *priv, struct mx_s *orig)
{
    priv->value = orig->value;
    priv->index = orig->index;
}
```

The combiner in C

- Expression – here implemented with a function
- Acts on variables:

  omp_in: reference to contribution from thread

  omp_out: reference to combined result
- Example: replace if contribution is larger

```
void mx_combine(struct mx_s *out, struct mx_s *in)
{
    if ( out->value < in->value ) {
       out->value = in->value;
       out->index = in->index;
    }
}
```

How to use it in C:

- You can use it similar to predefined reductions:

```
mx->value = val[0];
mx->index = 0;

#pragma omp parallel for reduction(maxloc: mx)
for (i=1; i < count; i++) {
   if (mx.value < val[i])
   {
      mx.value = val[i];
      mx.index = i;
   }
}
```

- Easily readable code
- Similar to what one would do in serial programming

Declaring a reduction operation
Syntax summary

- Basic syntax in C

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

- Basic syntax in Fortran

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

Dealing with global storage

- By default global storage is **`shared`**
- C/C++ examples for global storage:
  - file scope variables
  - static variable
- Fortran examples for global storage:
  - `COMMON` blocks
  - `module` data
  - variable with `save` attribute
- Not always what is needed

Directive: `threadprivate` in C

- Directive `threadprivate`
- Each thread gets private copy
- Outside `parallel`: modify copy of `master`
- Example prints:
  - 4 on master thread
  - 1 else

```
int g_var=1;
#pragma omp \
      threadprivate(g_var)

int main{
  g_var = 4;
#pragma omp parallel
  {
    printf(”%d\n”, g_var);
  }
return 0;
}
```

Directive: `threadprivate` in Fortran

- Directive `threadprivate`
- Each thread gets private copy
- Outside `parallel`: modify copy of `master`
- Example prints:
  - 4 on master thread
  - 1 else

```
module gmod
 integer g_var=1
 !$omp threadprivate(g_var)
end module gmod

Program example
  use gmod
  g_var = 4
!$omp parallel
    print *,g_var
!$omp end parallel
End program example
```

Clause: `copyin` to intitialise `threadprivate` in C

- Initialise threadprivate data form master: `copyin` clause

- Example prints:
  - 4 on all threads

```
int g_var=1;
#pragma omp \
      threadprivate(g_var)

int main{
  g_var = 4;
#pragma omp parallel \
    copyin(g_var)
  {
    printf(”%d\n”, g_var);
  }
return 0; }
```

Directive: `threadprivate` in Fortran

- Initialise threadprivate data form master: `copyin` clause
- Example prints:
  - 4 on all threads

```
module gmod
 int g_var=1
 !$omp threadprivate(g_var)
end module gmod

Program example
  use gmod
  g_var = 4
!$omp parallel copyin(g_var)
    print *,g_var
!$omp end parallel
End program example
```

More on `threadprivate`

- `threadprivate` data **unchanged** between `parallel` regions if:
  - Neither region nested inside other parallel region
  - Both regions have same thread count
  - Internal variable *dyn-var* is false in both regions (use function `omp_set_dynamic` to set)
- In Fortran one can make a `COMMON` block `threadprivate`

```
integer :: a, b, c
COMMON/abccom/a,b,c
!$OMP threadprivate(/abccom/)
```

Summary

- Discussed special private variables
  - `firstprivate`: initialisation of private variables
  - `lastprivate`: set value of private variable to value of last loop iteration or last section at end of construct
  - `reduction`: Calculating sums, products etc. in parallel
  - `threadprivate`: privatise global storage

- **Remark:** The above will handle standard situations
  - Constructs of the earlier examples for special cases
    - Initialise `private` variable from `shared` variable
    - `Atomic/critical` writes of shared variables