Introduction to OpenMP

Dr. Christian Terboven
Data Scoping
Dr. Christian Terboven

Introduction to OpenMP
Managing the Data Environment is the challenge of OpenMP.

**Scoping in OpenMP: Dividing variables in shared and private:**

- **private**-list and **shared**-list on Parallel Region
- **private**-list and **shared**-list on Worksharing constructs
- General default is **shared** for Parallel Region, **firstprivate** for Tasks.
- Loop control variables on **for**-constructs are **private**
- Non-static variables local to Parallel Regions are **private**
- **private**: A new uninitialized instance is created for the task or each thread executing the construct
  - **firstprivate**: Initialization with the value before encountering the construct
  - **lastprivate**: Value of last loop iteration is written back to Master
- Static variables are **shared**
Scoping Rules

- Managing the Data Environment is the challenge of OpenMP.
- **Scoping** in OpenMP: Dividing variables in *shared* and *private*:
  - *private*-list and *shared*-list on Parallel Region
  - *private*-list and *shared*-list on Worksharing constructs
  - General default is *shared* for Parallel Region, *firstprivate* for Tasks.
  - Loop control variables on *for*-constructs are *private*
  - Non-static variables local to Parallel Regions are *private*
  - *private*: A new uninitialized instance is created for the task or each thread executing the construct
    - *firstprivate*: Initialization with the value before encountering the construct
    - *lastprivate*: Value of last loop iteration is written back to Master
  - Static variables are *shared*
Privatization of Global/Static Variables

- Global / static variables can be privatized with the `threadprivate` directive
  - One instance is created for each thread
    - Before the first parallel region is encountered
    - Instance exists until the program ends
    - Does not work (well) with nested Parallel Region
  - Based on thread-local storage (TLS)
    - `TlsAlloc` (Win32-Threads), `pthread_key_create` (Posix-Threads), keyword `__thread` (GNU extension)

---

### C/C++
```
static int i;
#pragma omp threadprivate(i)
```

### Fortran
```
SAVE INTEGER :: i
 !$omp threadprivate(i)
```
Privatization of Global/Static Variables

- Global / static variables can be privatized with the `threadprivate` directive
  - One instance is created for each thread
    - Before the first parallel region is encountered
    - Instance exists until the program ends
    - Does not work (well) with nested Parallel Region
  - Based on thread-local storage (TLS)
    - TlsAlloc (Win32-Threads), pthread_key_create (Posix-Threads), keyword `__thread` (GNU extension)

```
C/C++
static int i;
#pragma omp threadprivate(i)

Fortran
SAVE INTEGER :: i
 !$omp threadprivate(i)
```

Really: try to avoid the use of threadprivate and static variables!
Back to our bad scaling example

C/C++

```c
int i, s = 0;
#pragma omp parallel for
for (i = 0; i < 100; i++)
{
    #pragma omp critical
    { s = s + a[i]; } 
}
```
It's your turn: Make It Scale!

```c
#pragma omp parallel
{

#pragma omp for
for (i = 0; i < 99; i++)
{
    s = s + a[i];
}

#do i = 0, 24
s = s + a(i)
end do

#do i = 25, 49
s = s + a(i)
end do

#do i = 50, 74
s = s + a(i)
end do

#do i = 75, 99
s = s + a(i)
end do

} // end parallel
```
The Reduction Clause

In a reduction-operation the operator is applied to all variables in the list. The variables have to be shared.

- reduction(operator:list)
- The result is provided in the associated reduction variable

C/C++

```c
int i, s = 0;
#pragma omp parallel for reduction(+:s)
for(i = 0; i < 99; i++)
{
    s = s + a[i];
}
```

- Possible reduction operators with initialization value:
  - + (0), * (1), - (0), & (~0), | (0), && (1), || (0), ^ (0), min (largest number), max (least number)
int a=0;

#pragma omp parallel
#pragma omp for reduction(+:a)
for (int i=0; i<100; i++)
{
    a+=i;
}

local copies for computation

update is written to the shared variable

reduction computes final result in the shared variable
Questions?