ICHEC

Irish Centre for High-End Computing

Parallel Models

Paradigms V

Threads vs Process

Process	Thread
Instance of a program running on a computer	A dispatchable unit of work within a process
Heavy weight operation	Light weight operation
Every process has its own memory space	Threads use the memory of the process they belong to
Inter process communication is slow as processes have different memory address	Inter thread communication as fast as threads of the same process share the same memory address of the process they belong to
Context switching between process is more expensive	Context switching between threads is less expensive
Processes don’t share the memory with other processes	Threads share the same memory with other threads of the same process

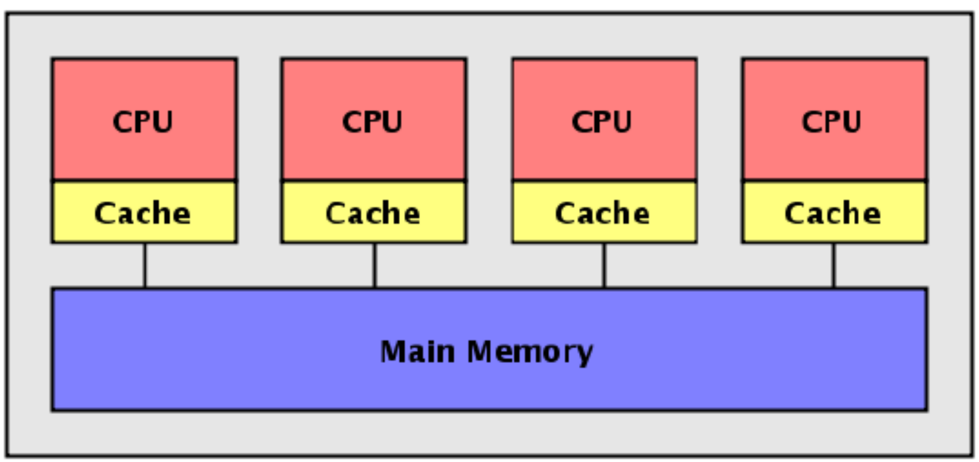
Shared Memory Programming

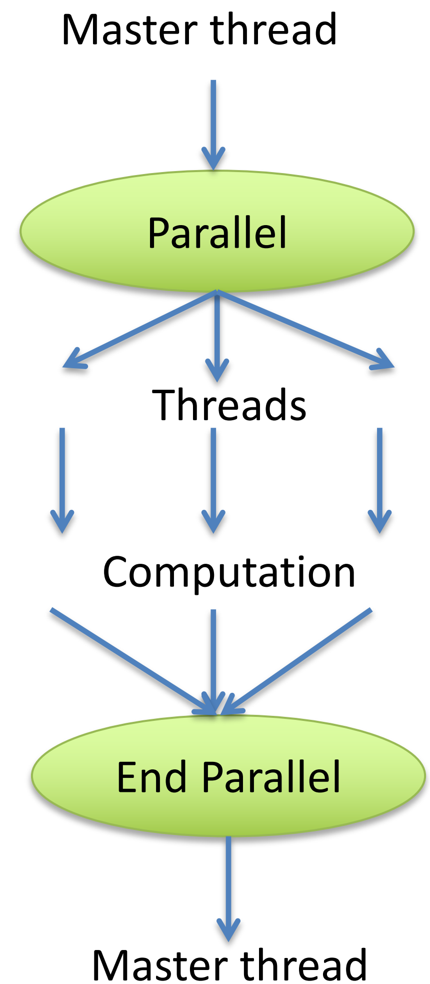
Advantages:

- The notion of data ownership is lacking, so there is no need to specify explicitly the communication of data between tasks
- Program development can often be simplified

Example: OpenMP

Fork-join Model





OpenMP Example C

```
double dot(double *A, double *B, int n)
{
  int i;
  double s=0.0d;
#pragma omp parallel for reduction(+:s),
shared(A,B), private(i)
  for(i=0; i<n; ++i) {
    s+=A[i]*B[i];
  }
  return s;
}
```

OpenMP Example Fortran

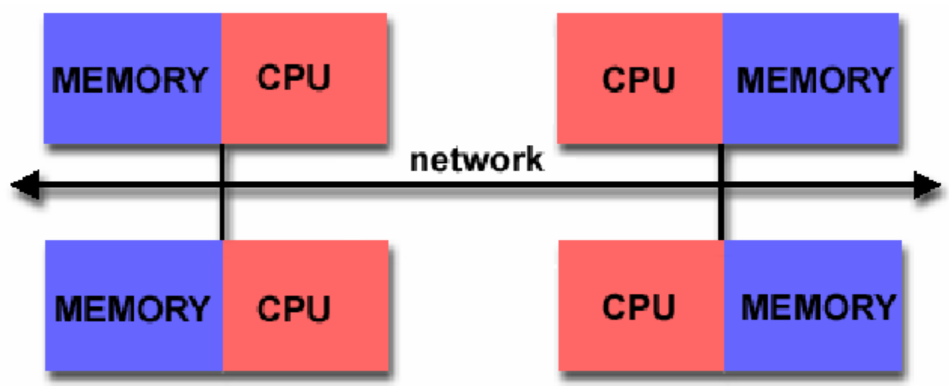
```
  real(kind=8) function dot(a,b,n)
    real(kind=8), intent(in) :: a(:),b(:)
    integer, intent(in) :: n

    integer :: i
    real(kind=8) :: s
    s=0.0_8
!$omp parallel do reduction(+:s), shared(A,B),
private(i)
    do i=1,n
      s=s+a(i)*b(i)
    end do
!$omp end parallel do
    dot=s
  end function dot
```

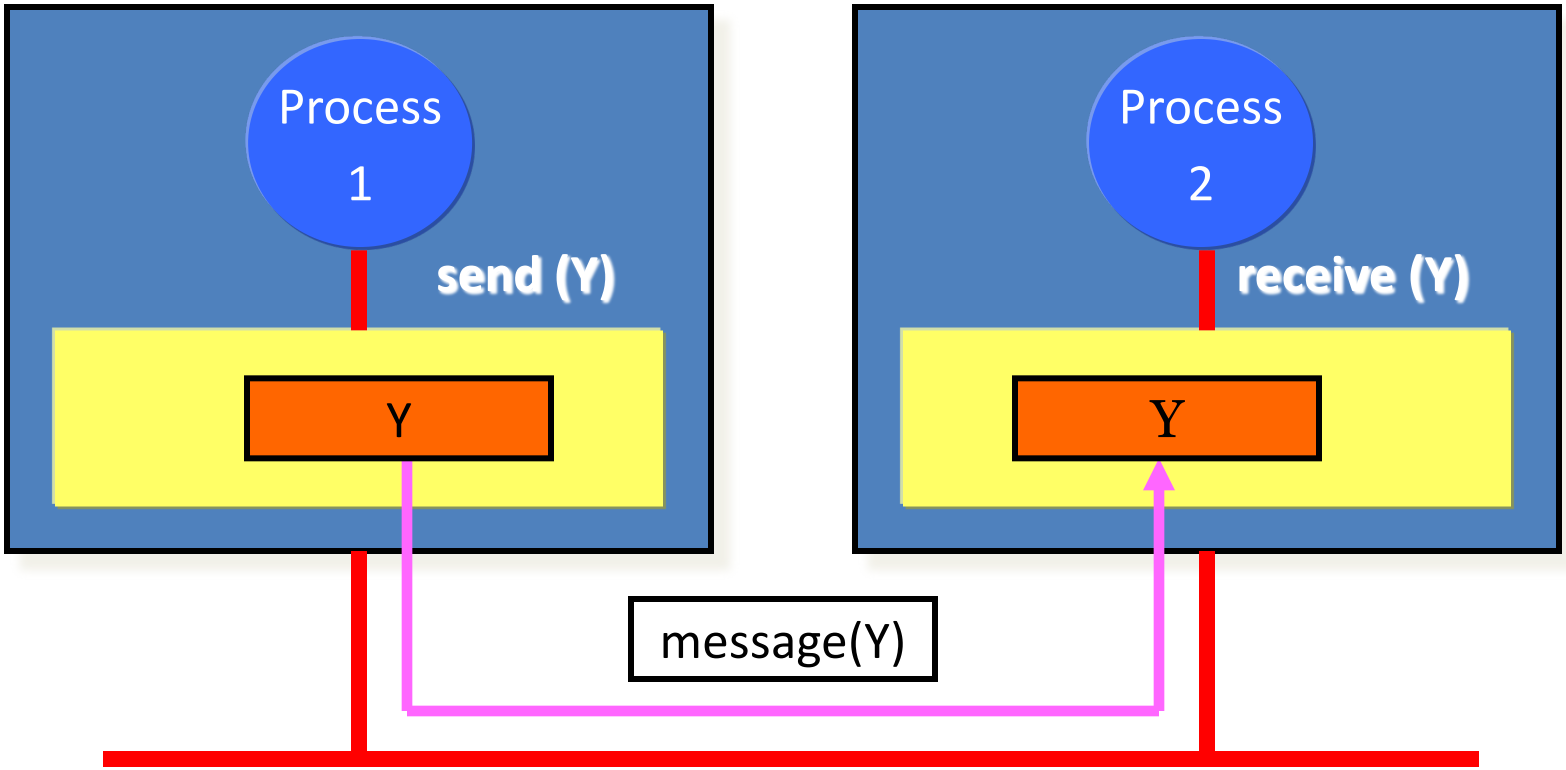
Distributed Memory Programming

The message passing model demonstrates the following characteristics:

- A set of processors that use their own local memory during computation.
- Processors exchange data through communications by sending and receiving messages.
- Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.
- Example: MPI



Send/Receive



MPI Example C

```
int main(int argc, char **argv) {
   float buf[10];
   int ierr, rank, sendto, tag;
   MPI_Status stat;

   ierr = MPI_init(&argc, &argv);
   ierr = MPI_comm_rank(MPI_COMM_WORLD, &rank);
   if (rank == 0) {
      sendto = 1;
      ierr = MPI_send(buf, 10, MPI_FLOAT, sendto, tag, MPI_COMM_WORLD);
   } else {
      recfrom = 0;
      ierr = MPI_recv(buf, 10, MPI_FLOAT, recfrom, tag, MPI_COMM_WORLD, &stat);
   }
   ierr = MPI_finalize();
}
```

MPI Example Fortran

```
Program SendEx
    use omp_lib
    real(kind=4) :: buffer(10)
    integer  :: ierr, rank, sendto, tag, stat(MPI_STATUE_SIZE)

    call MPI_init(ierr)
    call MPI_comm_rank(MPI_COMM_WORLD,rank,ierr)
    if (rank .eq. 0) then
       sendto = 1
       call MPI_send(buf,10,MPI_REAL4,sendto,tag,MPI_COMM_WORLD,ierr)
    else
       recfrom=0
       call
MPI_recv(buf,10,MPI_REAL4,recfrom,tag,MPI_COMM_WORLD,stat,ierr)
    endif
    call MPI_finalize(ierr)
end program SendEx
```

Summary

1. Parallelization breaks the problem into smaller chunks which are then solved concurrently
2. There are two programmatical classes: shared and distributed memory.
3. Shared memory allows vectorization and thread based parallelization, mostly handled by the compiler.
4. Distributed memory approach requires more input by the programmer.