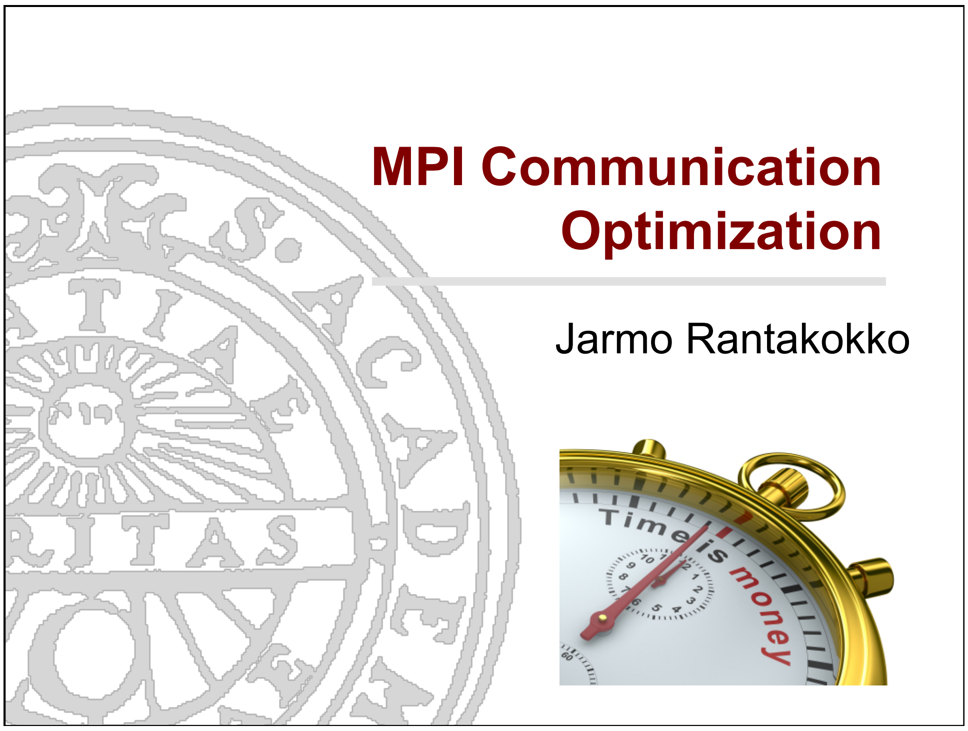
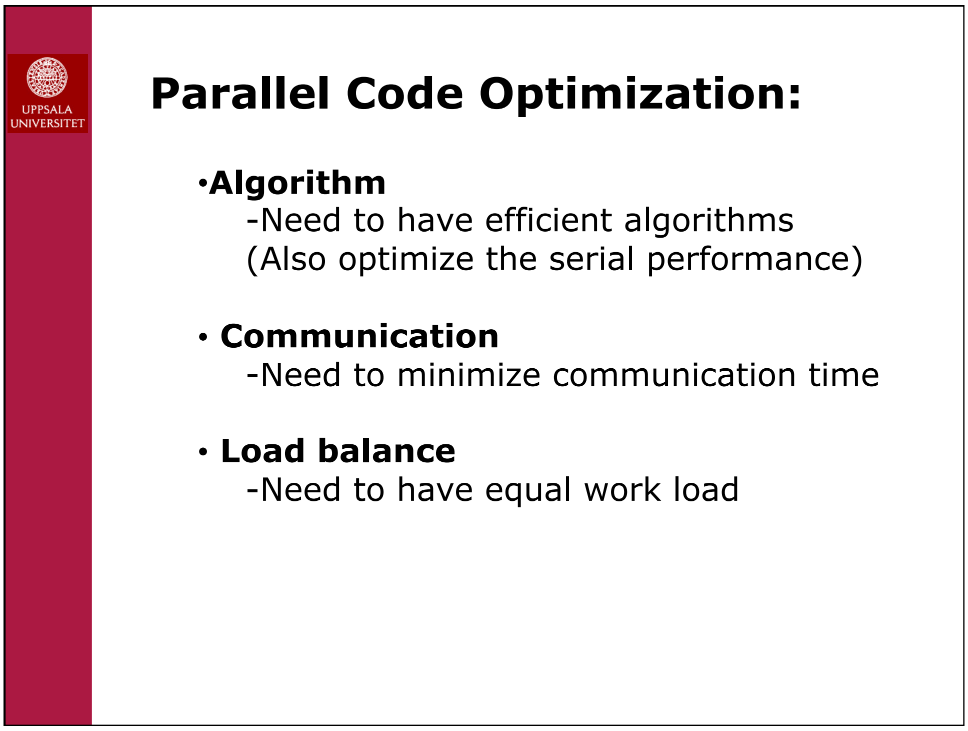


MPI Communication Optimization

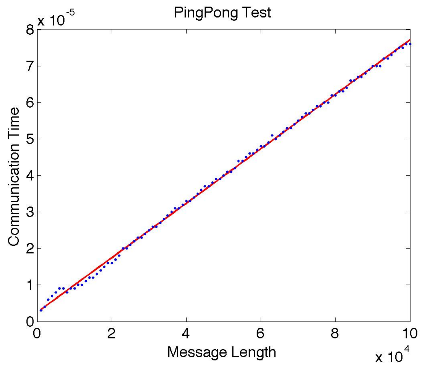
Jarmo Rantakokko



Parallel Code Optimization:

- **Algorithm**
  - -Need to have efficient algorithms (Also optimize the serial performance)
- **Communication**
  - -Need to minimize communication time
- **Load balance**
  - -Need to have equal work load

Communication overhead

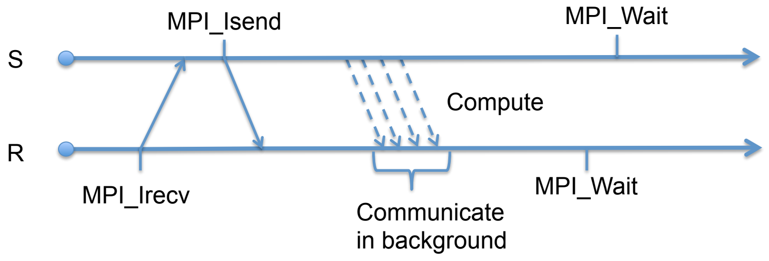


$T_{Comm}(n)=t_s+n\ t_w$, t_s=2.5 μs, β=10.7 GB/s, (t_w=1/β)
⇒ T_{Comm}(1)=2500 ns + 0.7ns, latency>>transfer time
Compare: 37GFlop/s => 92500 operations/latency

⇒ **Need to minimize and/or hide the latency**
(and keep the total communication volume down) **!**

Communication optimization:

1. Non-blocking/buffered communication
Overlap computations and communication



⇒ Avoid synchronization and hide latency (and data transfer time).

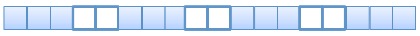
UPPSALA UNIVERSITET

2. Derived datatypes

Send non-contiguous data in one call

- MPI_Type_vector(…) -- Regular distribution
- MPI_Type_indexed(…) -- Irregular distribution
- MPI_Type_struct(…) -- Different datatypes

=> Reduce number of communication calls (latency)



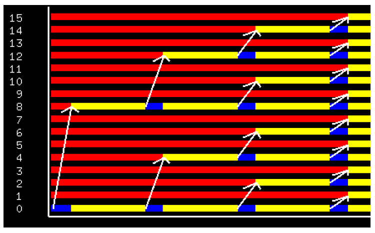
```
MPI_Type_vector(4,3,5,MPI_DOUBLE,&strided);
MPI_Type_commit(&strided);
MPI_Send(A,1,strided,to,tag,comm);
```

UPPSALA UNIVERSITET

3. Collective communication calls

Uses efficient algorithms to communicate with many processors in one call:

- MPI_Bcast(…)
- MPI_Scatter(…)
- MPI_Gather(…)
- MPI_Allgather(…)
- MPI_Alltoall(…)
- MPI_Reduce(…)

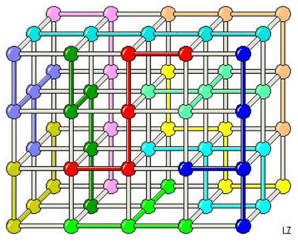


⇒ Parallelize the communication (reduce latency)

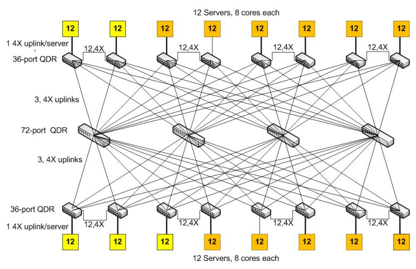
4. Virtual topologies

Communicator with structure. Makes a mapping to physical topology optimizing (localizing) the communication. Helps in algorithm construction reflecting the communication pattern.

- MPI_Cart_create(…)
- MPI_Graph_create(…)



Virtual topology



Physical topology

5. Persistent communication

Initiate communication once and activate repeatedly
⇒ Reduces initialization overhead (latency), useful when we have many calls and complicated patterns.

```
MPI_Send_init(…);
MPI_Recv_init(…):
etc (many calls)

for (i=0; i<n; i++)
{
   MPI_Startall(…);
   Compute
   MPI_Waitall(…)
   Use data
}
```

6. Message probing
Reduce synchronization OH by not ordering messages, use first-come first-serve.
Avoid sending length in a separate call.

```
while (i<nproc)
{
   MPI_Probe(MPI_ANY_SOURCE,… , &stat);
   MPI_Get_count(&stat,type,&len);
   MPI_Recv(data,len,type,stat.MPI_SOURCE,… );
   i++;
}
```

Example: Numerical PDE Solver

Consider the Hyperbolic PDE:

$$u_t + u_x + u_y = F(t,x,y) \quad 0 \le x \le 1, 0 \le y \le 1$$

$$\begin{cases} u(t,0,y) = h_1(t,y) & 0 \le y \le 1 \\ u(t,x,0) = h_2(t,x) & 0 \le x \le 1 \end{cases} \quad \textit{Boundary Conditions}$$

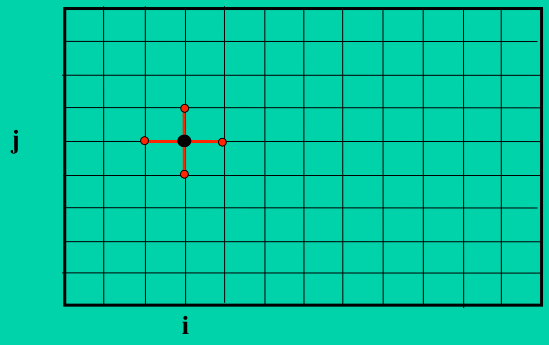
$$u(0,x,y) = g(x,y) \quad \textit{Initial Conditions}$$

Solve with explicit finite difference method, for example Leap-Frog

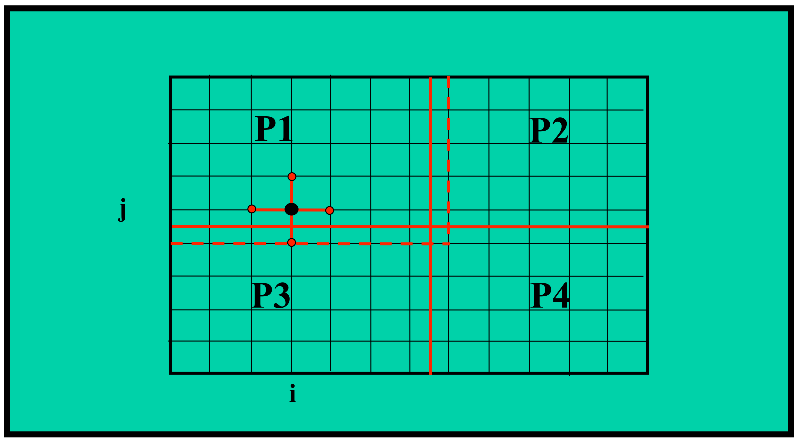
Core of the computations:

```
do k=2,Nt
   t=k*dt; Uold=U; U=Unew;
   do j=1,Ny-1
      do i=1,Nx-1
        x=i/Nx; y=j/Ny
        Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                (U(i+1,j)-u(i-1,j))/(2*dx)-
                (U(i,j+1)-U(i,j-1))/(2*dy))
      end do
   end do
end do
```

Computational stencil:



Parallelization, partition grid:



Message Passing Version:

```
do k=2,Nt
   t=k*dt; Uold=U; U=Unew;
   update partition boundary - communicate
   do j=j1,j2
      do i=i1,i2
         x=i/Nx; y=j/Ny
         Unew(i,j)=Uold(i,j)+2*dt*(F(t,x,y)-
                  (U(i+1,j)-u(i-1,j))/(2*dx)-
                  (U(i,j+1)-U(i,j-1))/(2*dy))
      end do
   end do
end do
```

```
Compute 'left, right, up, down' node

! Send and receive left-right
if ('not at left boundary')
   call mpi_send(left, ... )
end if

if ('not at right boundary')
   call mpi_recv(right, ... )
   call mpi_send(right, ... )
end if

if ('not at left boundary')
   call mpi_recv(left, ... )
end if

! Send and receive up-down
...
```

Note:

- **Structured domain decomposition**
  Use cartesian processor topology
  (simplifies communication)

- **The communication is repeated**
  Use persistent communication objects
  (mpi_send_init, mpi_startall, mpi_waitall)

- **Inner domain independent of neighbors**
  Overlap communication with computations
  (asynchronous communication)

```
! Create virtual topology
mpi_cart_create( ... )
mpi_cart_shift(dim1,left,right, ... )
...

! Initiate communication
mpi_send_init(left, ... )
mpi_recv_init(left, ... )
mpi_send_init(right, ... )
...

! Timestepping
do k=2,Nt
   mpi_startall( ... )
   update inner points
   mpi_waitall( ... )
   update partition boundary
end do
```

PDE Solver (lab MPI):

Program files:

• wave.c	-- Main program
• initcomm.c	-- Persistent communication
• diffop.c	-- Computational core

(+ some additional files)

Task: Study implementation details and measure the parallel performance for different problem sizes. Compare performance running within one server and using several servers (out of node communication).

Real example: Combustion simulation

HPC User: Michael Liberman, Physics dept UU

Problem:
Navier-Stokes 2D, chemical reaction terms
Modeling of deflagration-to-detonation transition
(i.e. knocking which must be avoided in engines)
=> Efficient burning and good fuel economy

Needs:
Requires a very accurate numerical resolution
of the flame front => Efficiently parallelized code
for a large number of processors

Results:

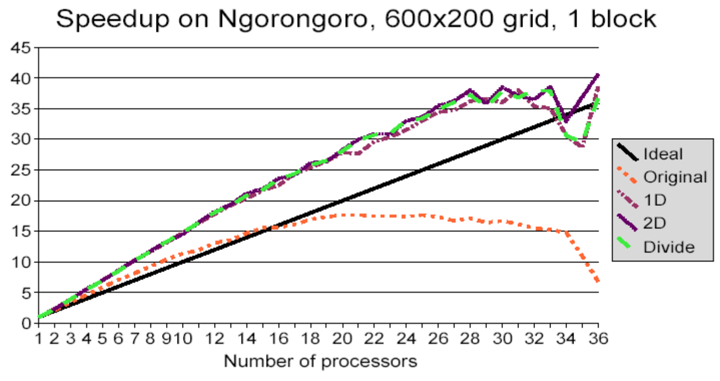
Orginal code parallelized with MPI, standard
MPI_Send/MPI_Recv, and 1D partitioning

- Changed standard calls to non-blocking
  persistent communication objects
- Extended to 2D partitioning, virtual topology
- Extended to multi-block partitioning,
  (allows for AMR and two-level parallelization)
- Efficient load balancing for multi-block grids

=> Very high parallel efficiency
(Super linear on SunFire 15K)

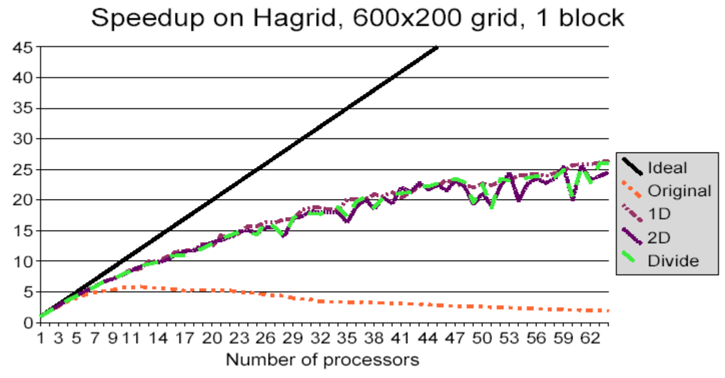
UPPSALA UNIVERSITET

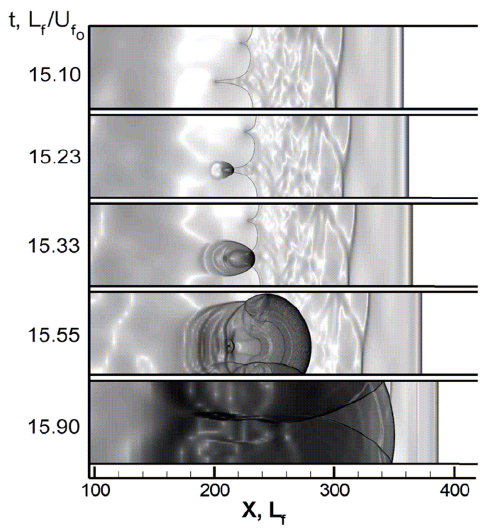
SunFire 15K (shared memory)



UPPSALA UNIVERSITET

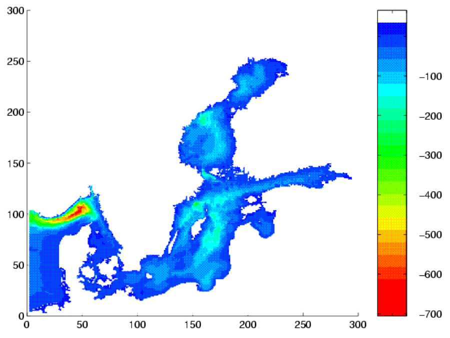
SweGrid I (PC-cluster)





631x2100 cells, 140 procs on UPPMAX, 212 hours

Real world example 2: Oceanographic model



- Irregular shape
- Irregular work load (due to sea depth)

⇒ Need to partition data in a way that gives an equal work load on the processor and minimizes communication