Introduction to High Performance Computing

MPI (2)

Collectives

Point-to-point communications do not allow interactions between multiple tasks simultanously. Therefore different communications are necessary. Available operations:

Barrier – All member of a group wait until all members reached the barrier.
Broadcast – One member sends the same data to all members of a group.
Scatter – A vector is sent and at the same time distributed within a group.
Gather – A distributed vector is sent within a group and assembled.
Reduction – Distributed data is combined to a single value within a group.

communicator and groups

- collective operations are b… groups of tasks
- groups are opaque objects with…
- groups are associated with a commu…
- all groups may be formed manipulating co… icators

no more groups in this lecture

Basic communicators:
MPI_COMM_WORLD
MPI_COMM_SELF
MPI_COMM_NULL
MPI_PROC_NULL

MPI_PROC_NULL is used to send dummy messages to non-existing rank.

communicator and groups

communicators provide communication contexts

- different communicators/contexts can not interfere
- whatever communication pattern is used in third party software does not interfere – if a different communicator is used

communicators provide group scope for collective operations

communicators maybe augmented with topological information

- inter communicators
- intra communicators

communicator

Manipulation of communicators:

```
MPI_Comm_dup(oldcomm,&newcomm);
MPI_Comm_split(oldcomm,color,key,&newcomm);
MPI_Comm_free(&comm);
```

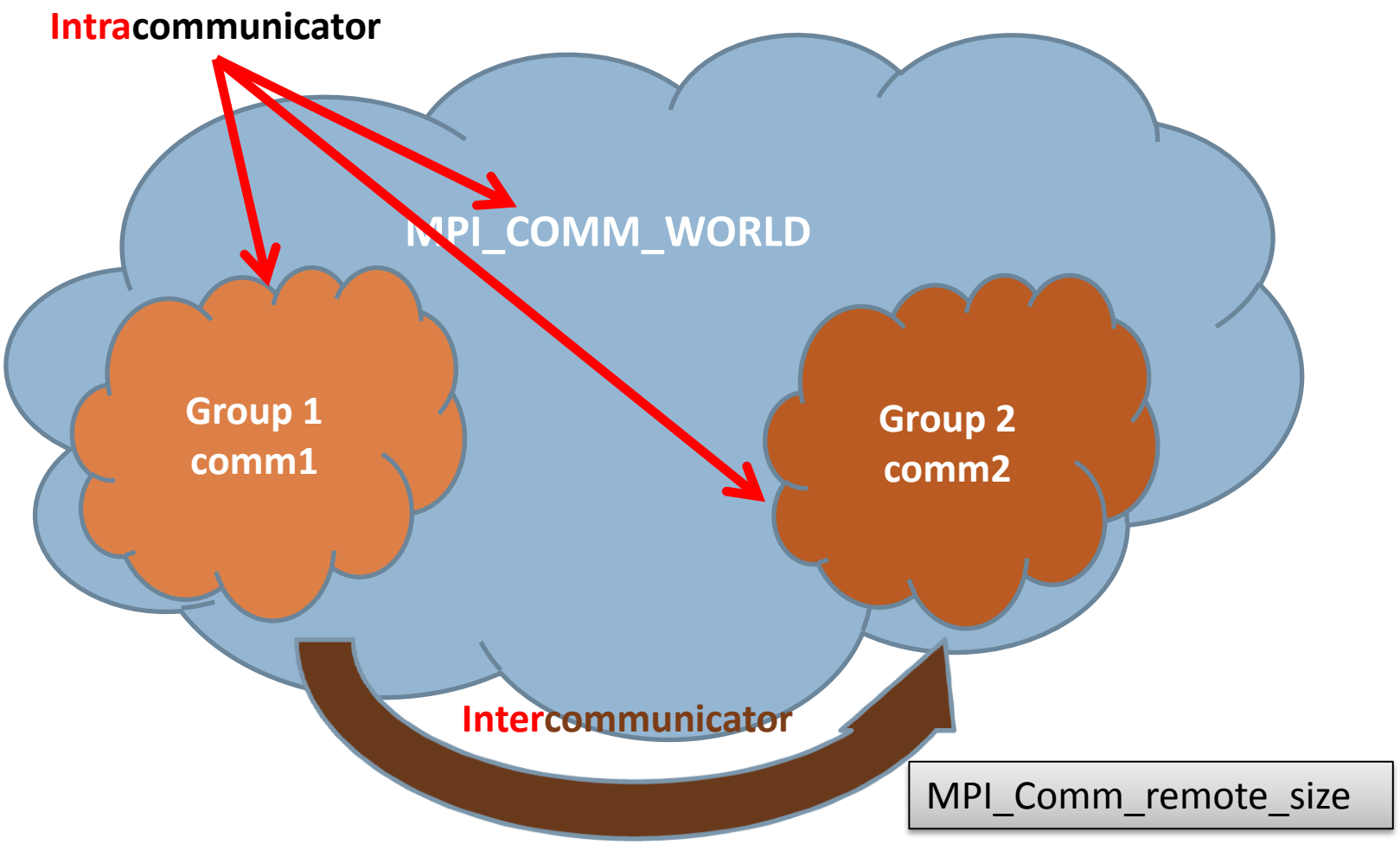
```
color=rank%3;
key=size-rank;
if(color==0)
        MPI_Comm_split(MPI_COMM_WORLD,MPI_UNDEFINED,0,&newcomm);
else
        MPI_Comm_split(MPI_COMM_WORLD,color,key,&newcomm);
```

Group 1 – ranks:
1,4,7,10
New ranks:
3,2,1,0

Group 2 – ranks:
2,5,8,11
New ranks:
3,2,1,0

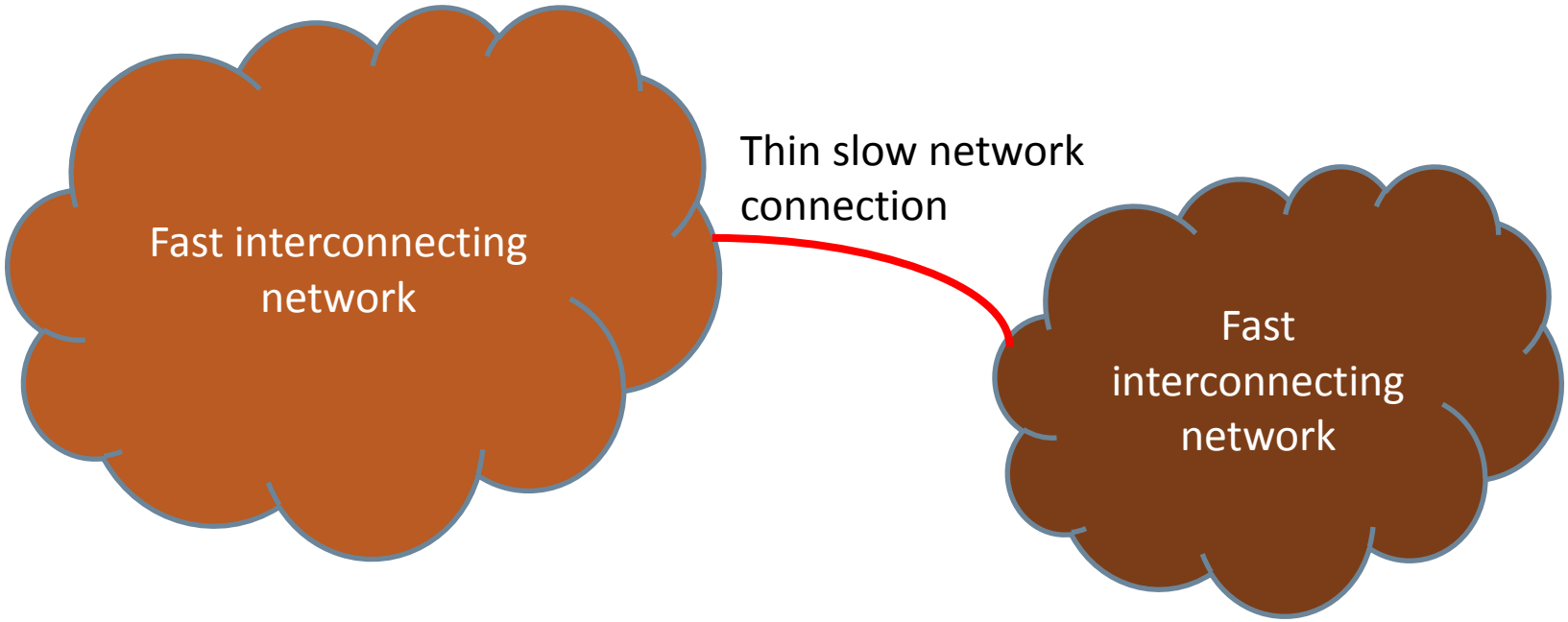
Ranks
0,3,6,9,12
MPI_COMM_NULL
no group

communicator



communicator

When to use intercommunicators?



Barrier

MPI_Barrier(communicator):
All tasks within communicator are mutually waiting until all reached the barrier. That way all tasks are synchronized.

Caveat:

```
if(myrank==0) {
 MPI_Isend(…);
 MPI_Barrier(comm);
} else {
 MPI_Barrier(comm);
}
```

Task 0

Task 1

MPI_Isend is not finished. MPI_Barrier is not equivalent with a memory fence or communication fence.

Barrier

MPI_Ibarrier(communicator,&request):
returns immediately. Only to be combined with MPI_Ibarrier. On return from a MPI_Wait(&request,MPI_STATUS)
all tasks within communicator are mutually waiting until all reached the barrier.

Example:

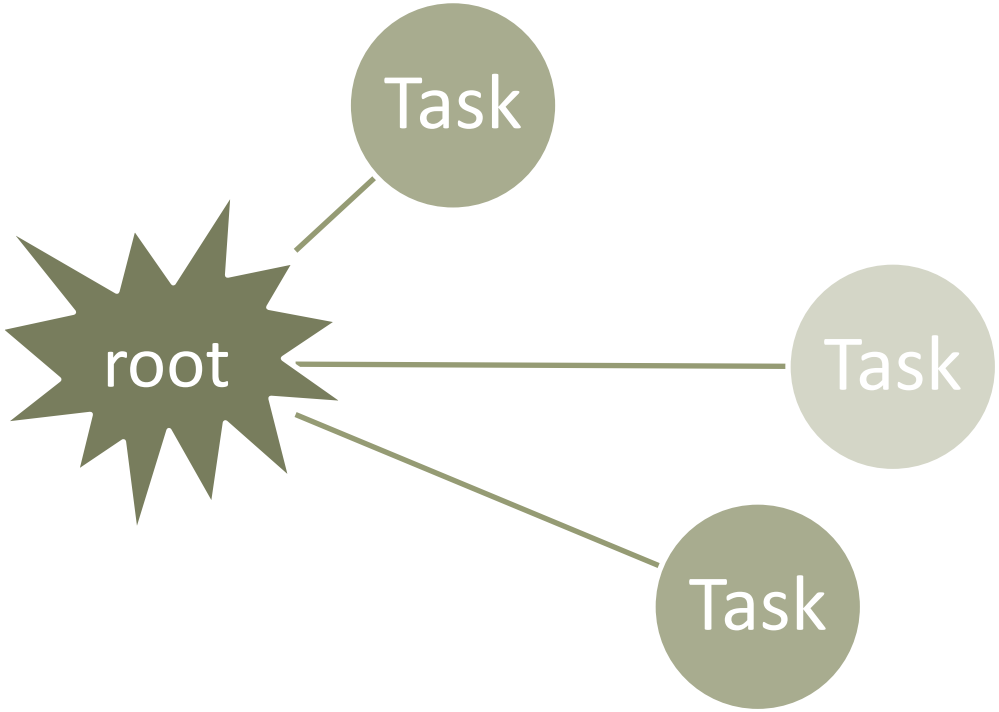
```
if(myrank==0) {
 MPI_Ibarrier(comm,&req)
 do_unrelated work;
} else {
 MPI_Ibarrier(comm,&req);
}
MPI_Wait(&req,MPI_STATUS);
continue_with_work;
```

Task 0

Task 1

Broadcast

MPI_Bcast(buffer,count,datatype,root,communicator):
All tasks use the same function call with the same arguments. Data on the task with rank root are broadcasted to all tasks within communicator. The call is blocking but not to be mistaken as synchronization.

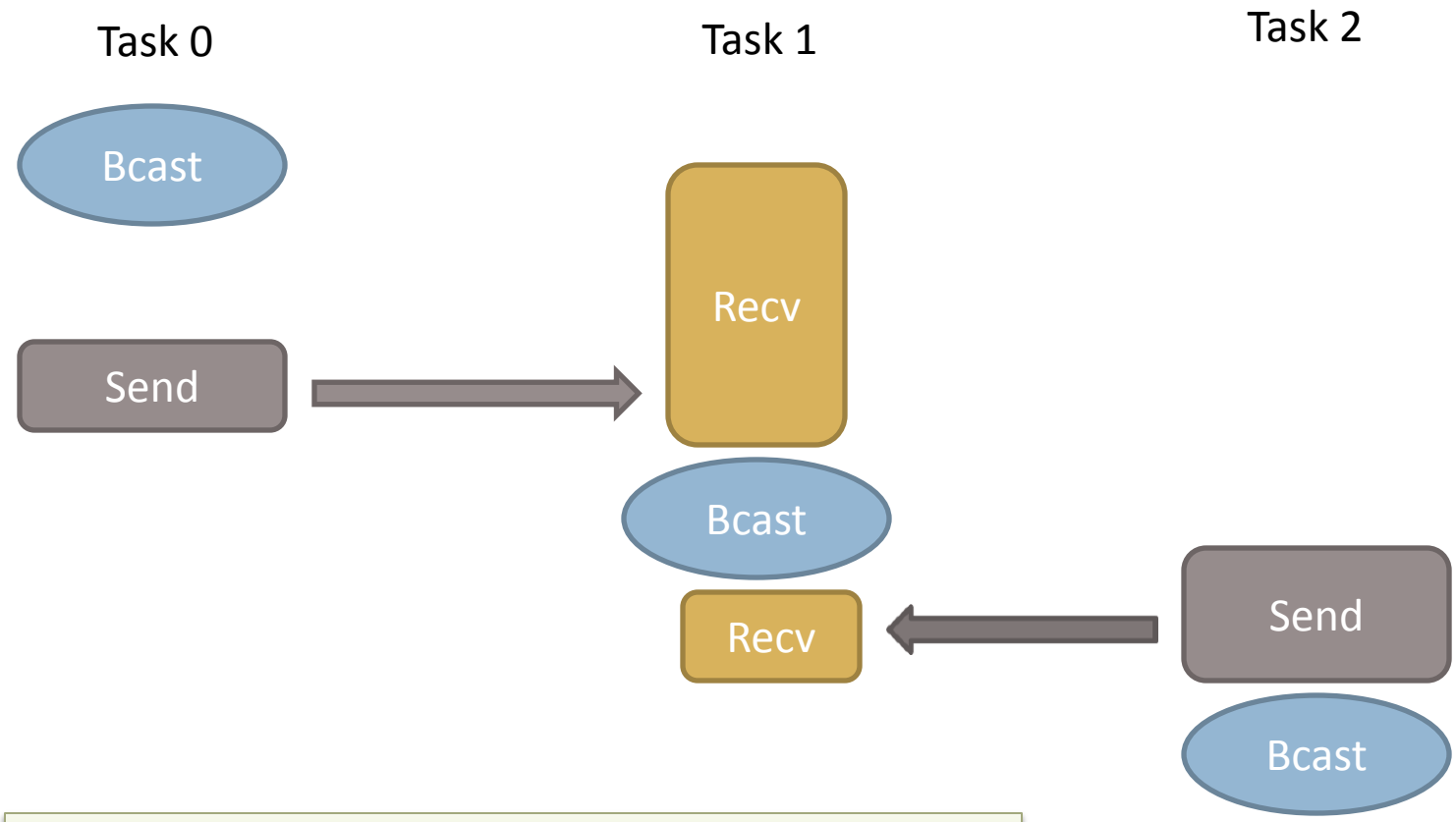


Broadcast

```
switch(myrank) {
   case 0:
        MPI_Bcast(buf1,count1,type,0,comm);
        MPI_Send(buf2,count2,type,1,tag,comm);
        break;
   case 1:
        MPI_Recv(buf2,count2,type,MPI_ANY_SOURCE,tag,comm,&status);
        MPI_Bcast(buf1,count1,type,0,comm);
        MPI_Recv(buf2,count2,type,MPI_ANY_SOURCE,tag,comm,&status);
        break;
   case 2:
        MPI_Send(buf2,count2,type,1,tag,comm);
        MPI_Bcast(buf1,count1,type,0,comm);
        break;
 }
```

- Task 2 sends buf2 to task 1
- Task 0 sends buf1 to task 1 and task 2
- Task 0 sends buf2 to task1

Broadcast



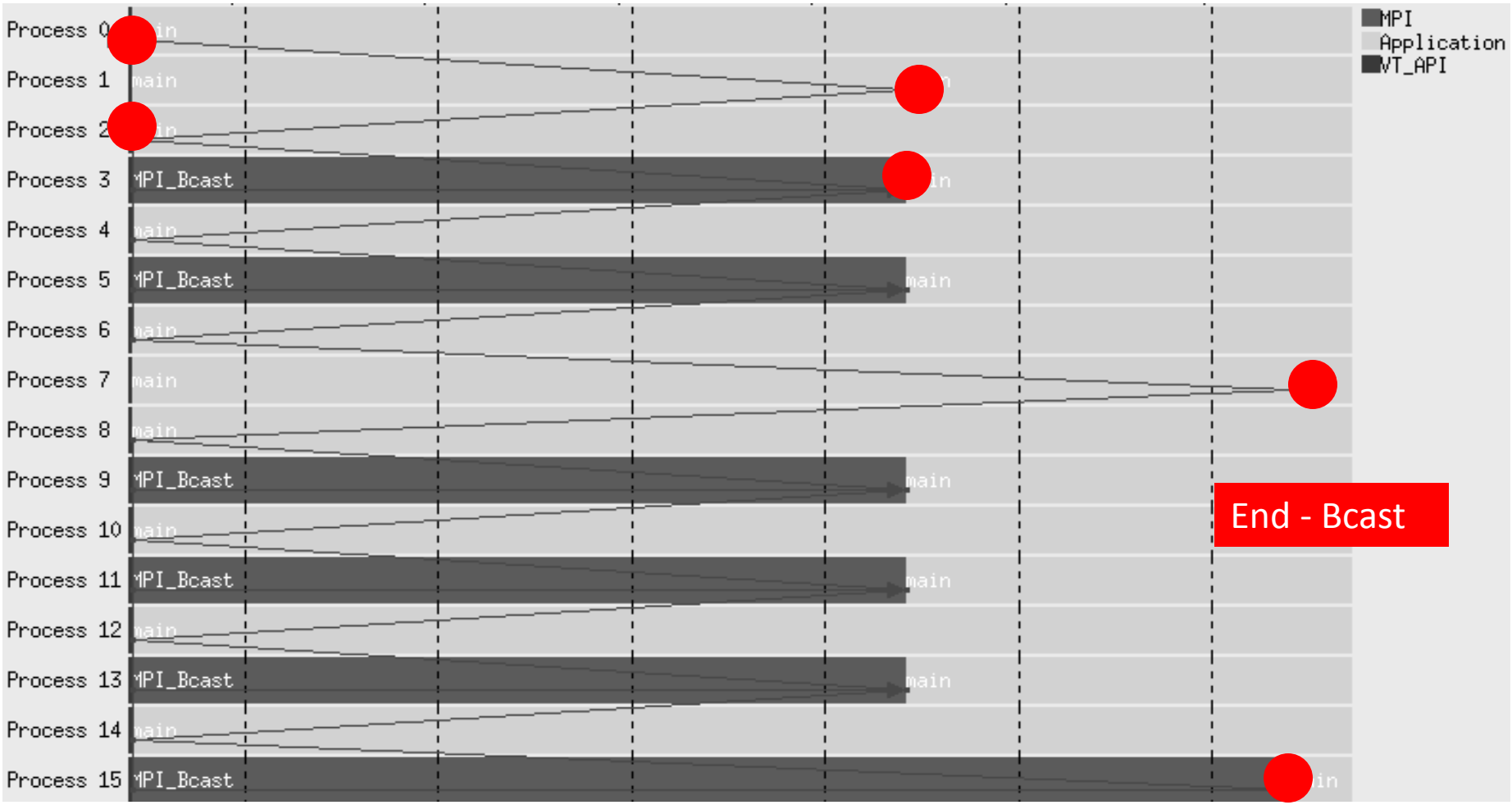
Bcast is blocking but the calls are not synchronized.

Broadcast – performance trace

```
switch(myrank) {
   case 1:
        sleep(1);
        MPI_Bcast(buf1,1,type,0,comm);
        break;
   case 7:
        sleep(3);
        MPI_Bcast(buf1,1,type,0,comm);
        break;
   default:
        MPI_Bcast(buf1,1,type,0,comm);
        break;
 }
```

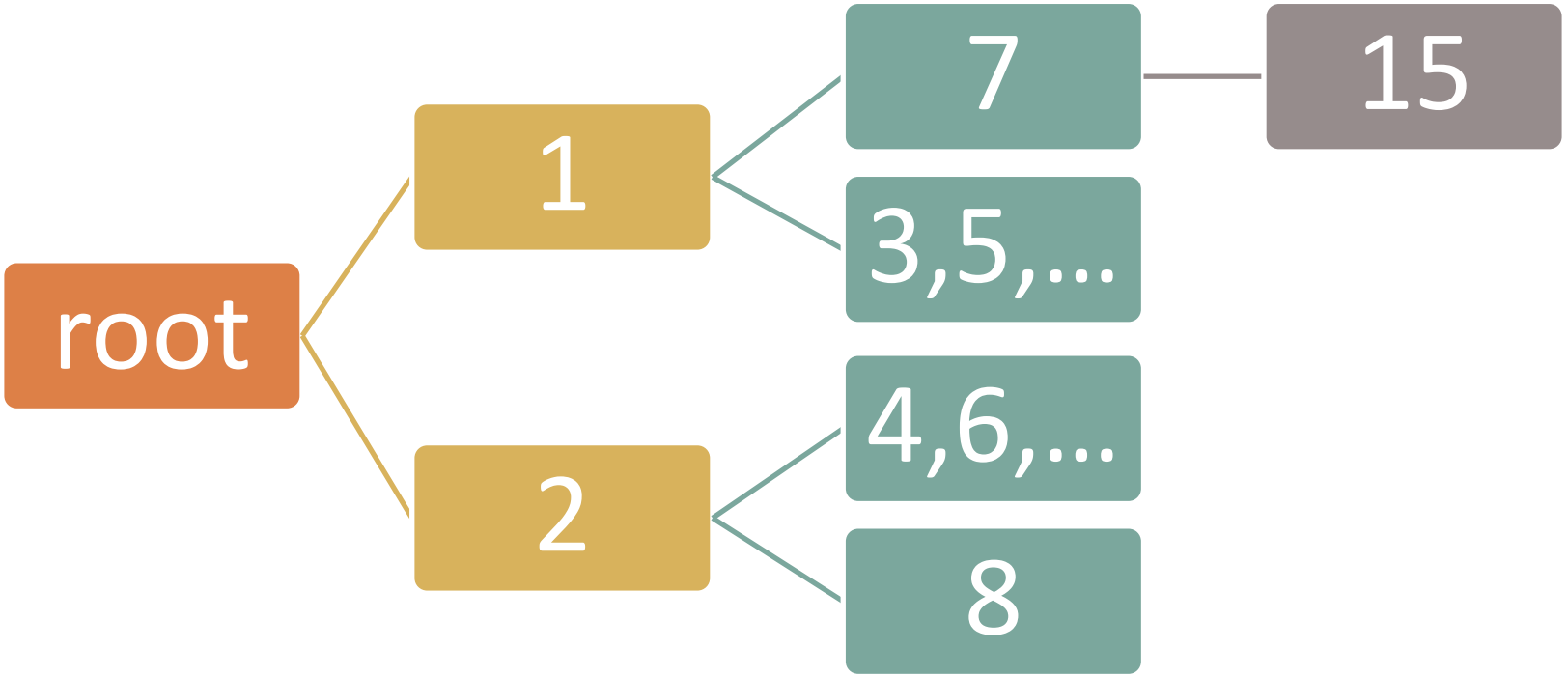
Vampir: http://tu-dresden.de/zih/vampir
Tool to visually analyze the performance of parallel applications

Broadcast – performance trace



Broadcast – performance trace

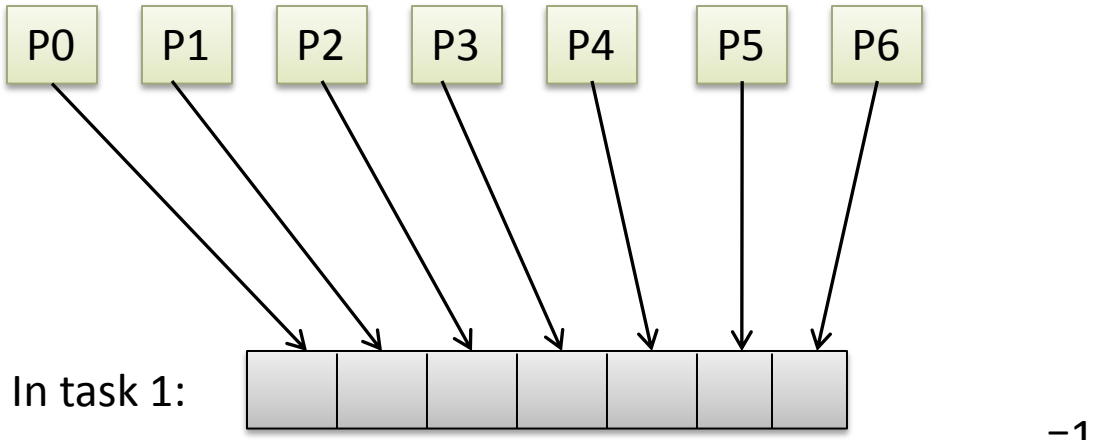
- OpenMPI implements a tree structure



non blocking broadcast

MPI_Ibcast(buffer,count,datatype,root,communicator,&request):
All tasks use the same function call with the same arguments. Data on the task with rank root are broadcasted to all tasks within communicator. root does not have to wait until an implementation likes to finish.

Gather



MPI_Gather(sbuf,sanz,styp,rbuf,ranz,rtyp,root,comm)

alternatively:

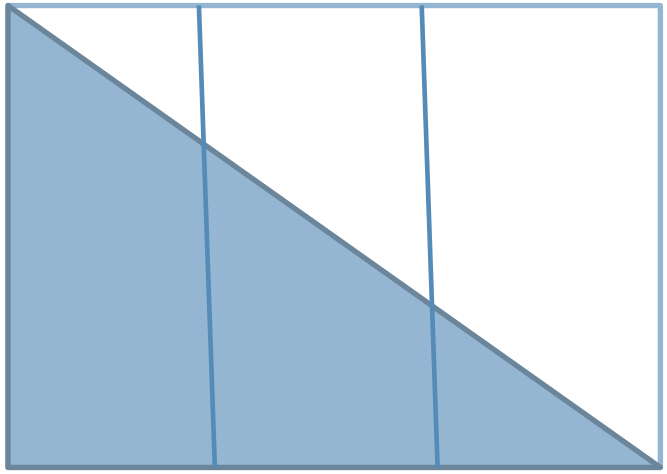
```
MPI_Send(sbuf,sanz,styp,tag,root,comm);
if(myrank==root) {
 if(i=0;i<group_size;i++)
  MPI_Recv(rbuf+i*ranz*extent(typ),ranz,rtyp,
                          i,tag,comm,&status);
}
```

Gatherv

- used for irregular patterns (inversed order, permutation, ….)
- different length for different tasks

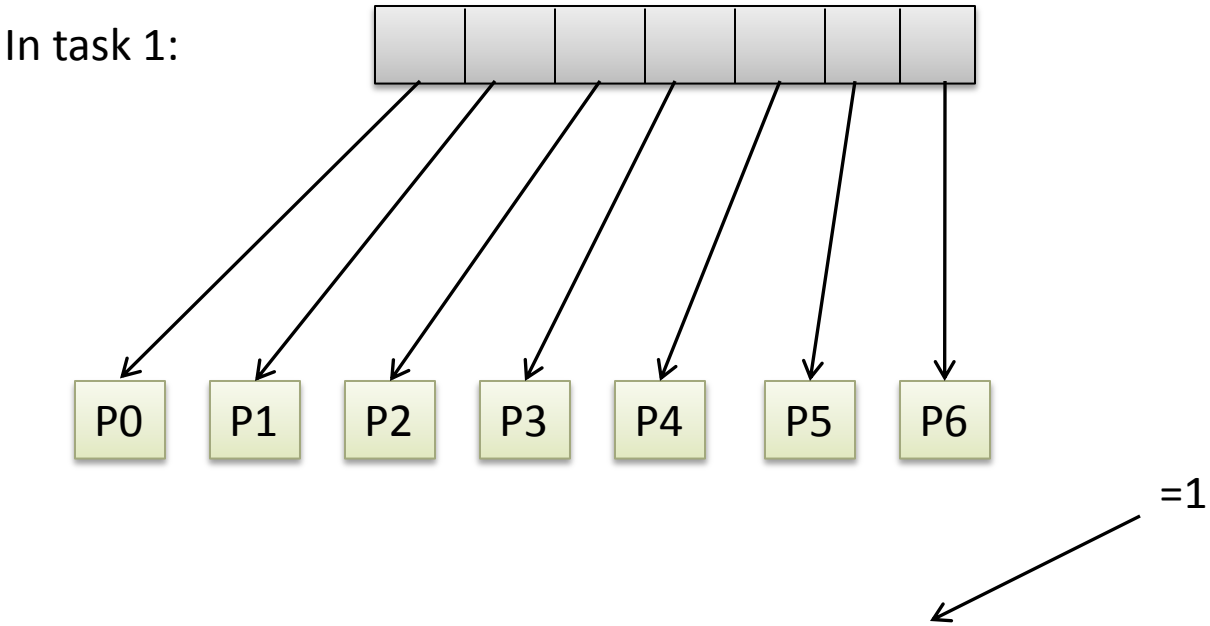
```
MPI_Gatherv(sbuf,sanz,styp,
            rbuf,{ranz},{displs},rtyp,root,comm)
```

Example:



Copying a distributed symmetric matrix into an array on task 0

Scatter



MPI_Scatter(sbuf,sanz,styp,rbuf,ranz,rtyp,root,comm)

MPI_Scatterv(sbuf,{sanz},{displs},styp,rbuf,ranz,rtyp,root,comm)

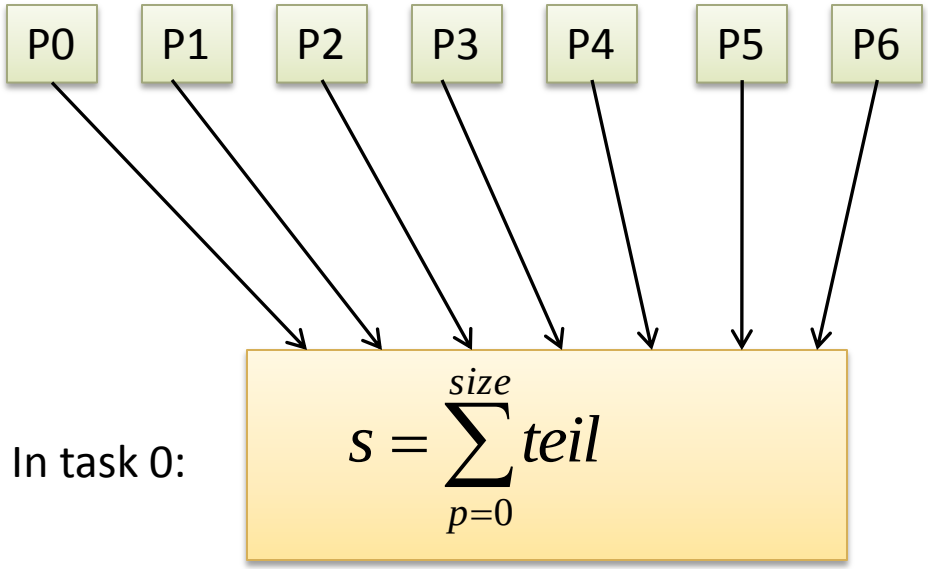
Scatterv

```
/*initialization */
if(myrank==root)  init(x,N);

/* organize distribution of work and data */
MPI_Comm_size(comm,&size);
Nopt=N/size;
Rest=N-Nopt*size;
displs[0]=0;
for(i=0;i<N;i++) {
  sanz[i]=Nopt;
  if(i>0) displs[i]=displs[i-1]+sanz[i-1]*sizeof(double);
  if(Rest>0) { sanz[i]++; Rest--;}
}

/*  distribute data */
MPI_Scatterv(x,sanz,displs,MPI_DOUBLE,y,sanz[myrank],MPI_DOUBLE,root,comm);
```

non blocking

MPI_Igather, MPI_Igatherv
MPI_Iscatter, MPI_Iscatter v

Reduction



root task 0 owns result:

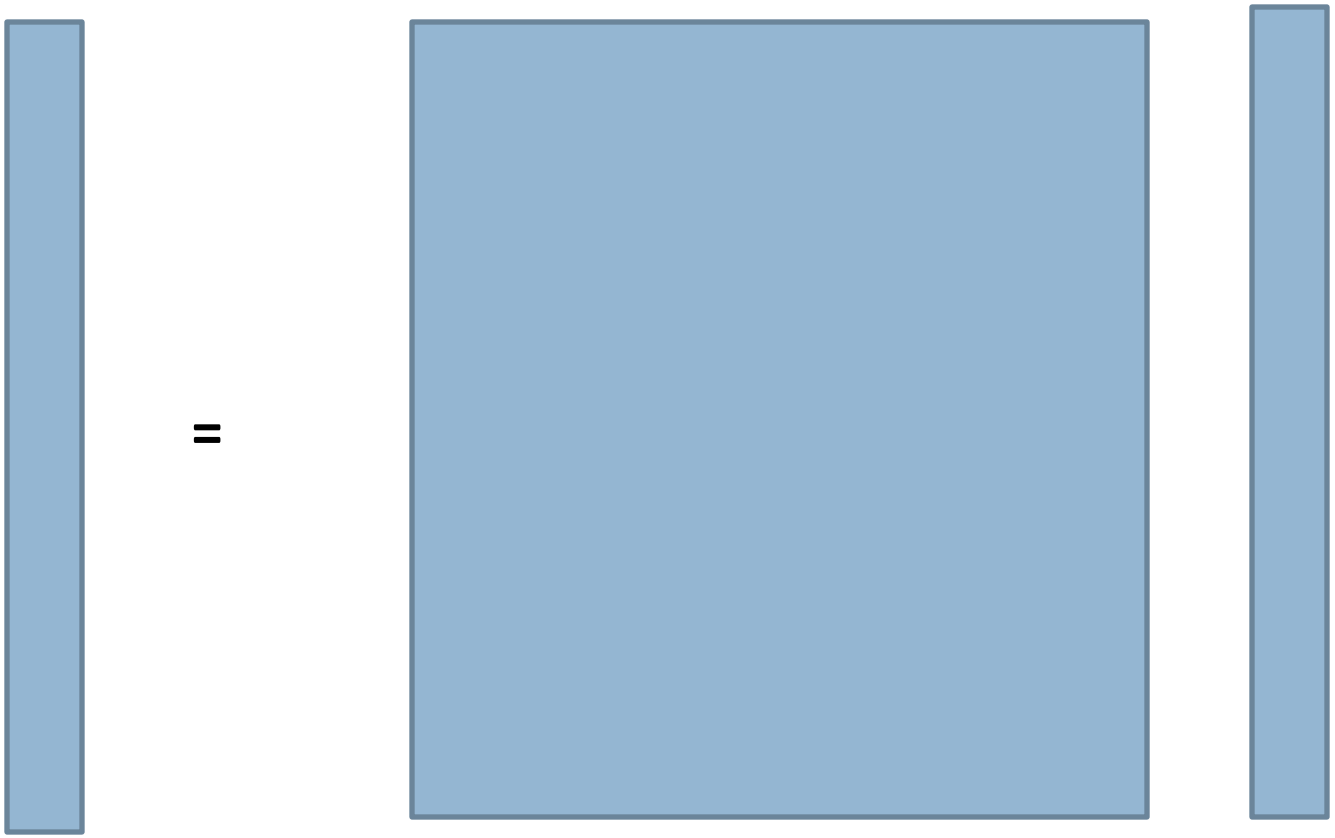
```
MPI_Reduce(teil,s,1,MPI_DOUBLE,MPI_SUM,0,comm)
```

all tasks own result:

```
MPI_Allreduce(teil,s,1,MPI_DOUBLE,MPI_SUM,comm)
```

Reduction

Multiply a matrix with a vector



Multiply a matrix with a vector

Basic algorithm, encapsulated in function, callable for submatrices
function(&matrix[begin],height,width,…)

```
for(i=0;i<M;i++) {
  s=0;
  for(j=0;j<N;j++)
    s+=A[i*lda+j]*x[j];
  y[i]=s;
}
```

Multiply a matrix with a vector

OpenMP

```
for(i=0;i<M;i++) {
  s=0;
#pragma omp for private(j) reduction(+:s)
  for(j=0;j<N;j++)
    s+=A[i*lda+j]*x[j];
  y[i]=s;
}
```

Multiply a matrix with a vector

OpenMP

```
#pragma omp for private(i,j,s)
for(i=0;i<M;i++) {
  s=0;
  for(j=0;j<N;j++)
    s+=A[i*lda+j]*x[j];
  y[i]=s;
}
```

Easy and simple to write – **but**:

- Which one will perform better?

Multiply a matrix with a vector

MPI

```
for(i=0;i<M;i++) {
   s=0;
   for(j=0;j<N;j++)
      s+=A[i*lda+j]*x[j];
   y[i]=s;
}
```

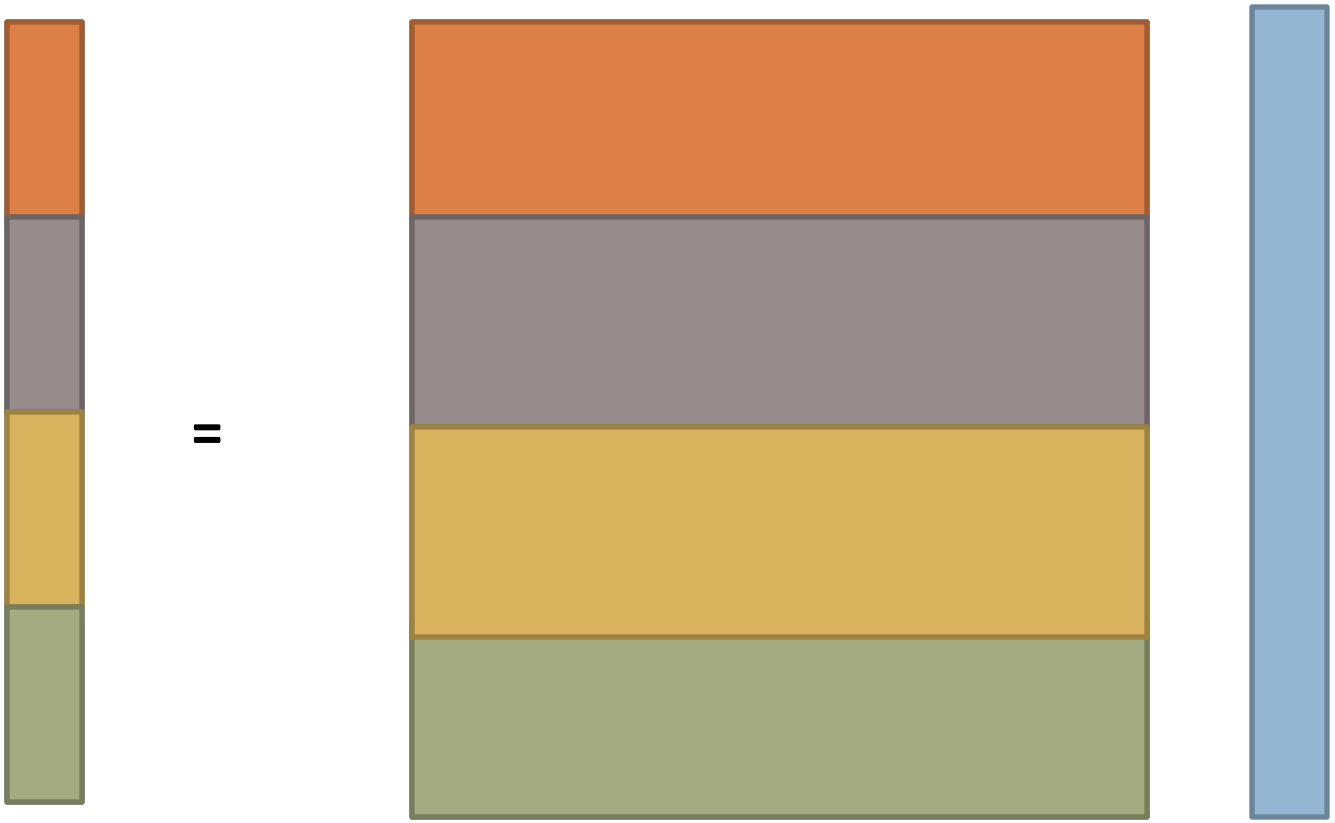
parallel

One distinct block per task

rowwise blocks per task

entire vector x
required on all tasks

Multiply a matrix with a vector



Multiply a matrix with a vector

Building block: matrix-vector multiplication (BLAS: dgemv)

```
void local_mv(N,M,y,A,lda,x) {
double x[N],A[N*M],y[M],s;
for(i=0;i<M;i++) {
  s=0;
  for(j=0;j<N;j++)
    s+=A[i*lda+j]*x[j];
  y[i]=s;
}
}
```

times:

arithmetic	2*N*M*Ta
Memory access	
X	M*Tm(N,1)
Y	Tm(M,1)
A	M*Tm(N,1)

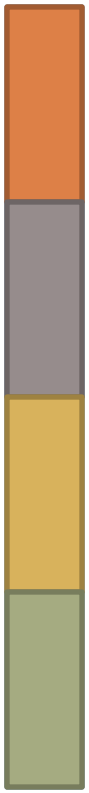
optimizable

Multiply a matrix with a vector

So far for the organization – now distribute the data:

Start: All data is owned by task 0, result resides on task 0

vector y:

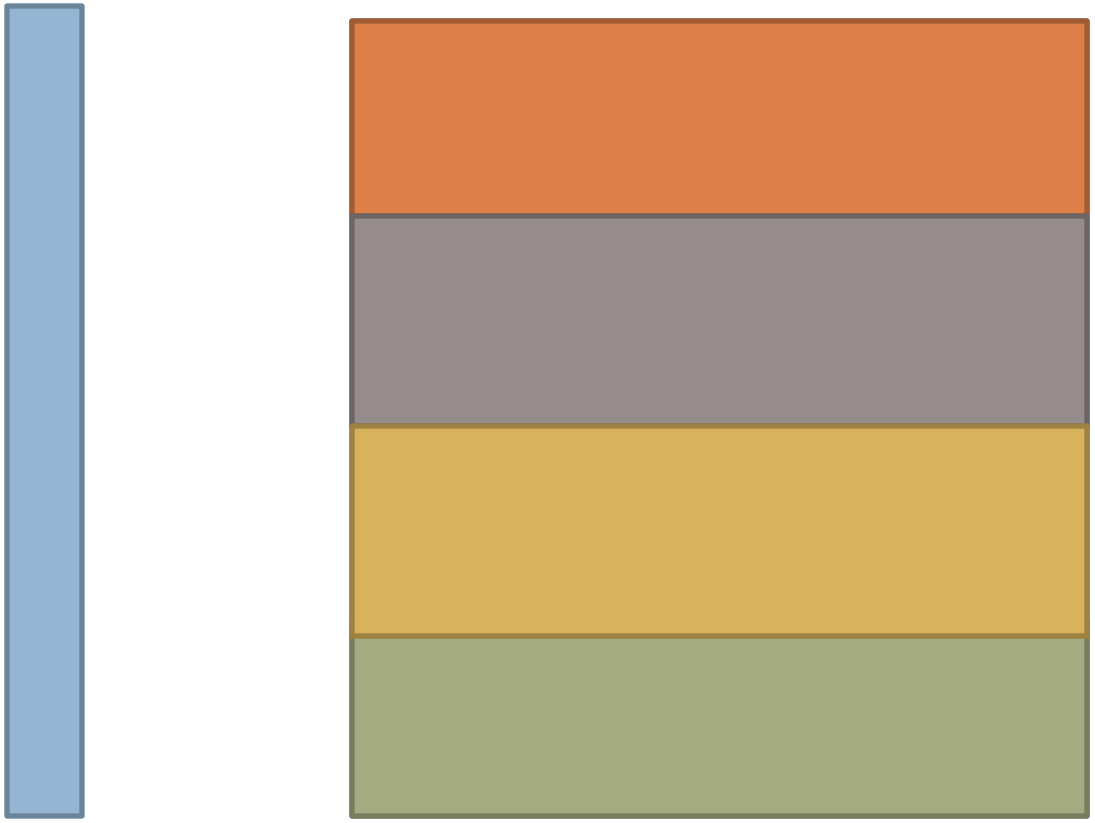


MPI_Gather bzw. MPI_Gatherv

Estimated time (p-1 times receive, M block size):
(p-1)*Tc(M/p)

Multiply a matrix with a vector

Vector x – (p-1) times send
MPI_Bcast
(p-1)*Tc(N)



matrix A:
MPI_Scatter bzw.
MPI_Scatterv

(p-1)*Tc(M/p*N)

Multiply a matrix with a vector

- communication

  (p-1)*Tc(M/p) + (p-1)*Tc(N) + (p-1)*Tc(M/p*N)

- arithmetic

  2*N*M*Ta

- memory accesses

  M*Tm(N,1) + Tm(M,1) + M*Tm(N,1)

- Estimation

$$\approx 4M * N * Ta + (p-1)(3Ts + M / p * N / BB)$$

$$\approx \frac{4N^2 * Ta}{p} + 3(p-1)Ts + \frac{N^2}{p * BB}$$

Multiply a matrix with a vector

$$\frac{4N^2 * Ta}{p} + 3(p-1)Ts + \frac{N^2}{p * BB}$$

	1Gb (LAN)	20Gb (Infiniband)
N=1000, p<	10ms (Ts,BB)	1ms(BB,Speicher)
N=1000, p=100	**20ms(Ts,BB)**	<1ms(Ts,BB)
N=100.000, p<	82s (BB,Speicher)	6s(BB,Speicher)
N=100.000, p=1000	80s (BB)	4s(BB)
N=1000	200 Mflops	>2 Gflops
N=100.000	250 Mflops	5 Gflops

sequential library: > 1Gflops !

Multiply a matrix with a vector

MPI

```
for(i=0;i<M;i++) {
  s=0;
  for(j=0;j<N;j++)
    s+=A[i*lda+j]*x[j];
  y[i]=s;
}
```

parallel

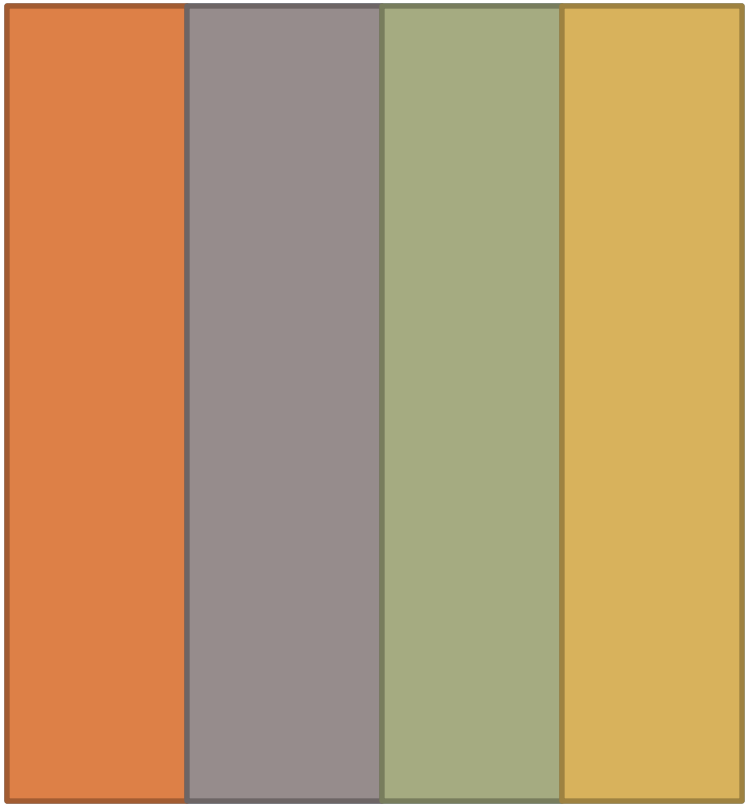
Remains on task 0

x distributed in blocks

columnwise blocks per task

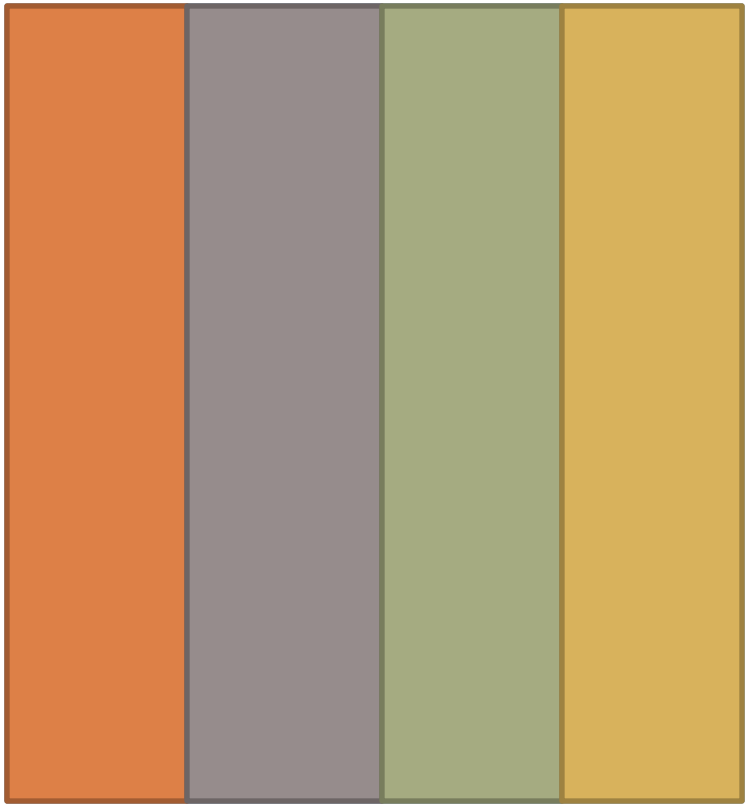
Multiply a matrix with a vector

vector x: MPI_Scatter (p-1)*Tk(N/p)



How ?

Multiply a matrix with a vector



Pack blocks into buffers and send them:

memory access (copy):
N*Tm(M,1) + M*Tm(N,1)

Send:
(p-1)Tc(M*N/p)

Multiply a matrix with a vector

Reduction (y): log(p)(Tc(N/p)+NTa+2Tm(N/p,1))

arithmetic: 2*N*M*Ta

memory: N/p*Tm(M,1) + Tm(N/p,1) + N*Tm(M,1)

Estimation: may be a bit faster

Parallelisation is only meaningful if distribution of data is not part of the problem but accomplished in advance!

Einführung in das Hochleistungsrechnen
Introduction to High Performance Computing

VIELEN DANK
THANK YOU