Fundamentals of Parallelism on Intel® Architecture

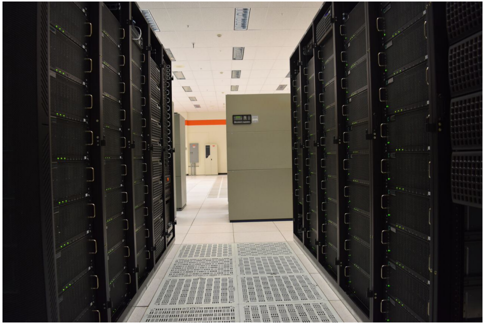
Week 5
Clusters and MPI

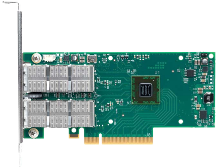
May 2017

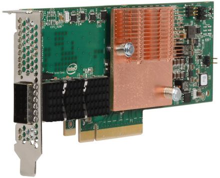
§1. Computing Clusters

Clusters

Clusters often use Gigabit Ethernet for administration and InfiniBand or Intel Omni-Path for communication.







Parallel Programming Layers

CLUSTER COMPUTING
in distributed memory

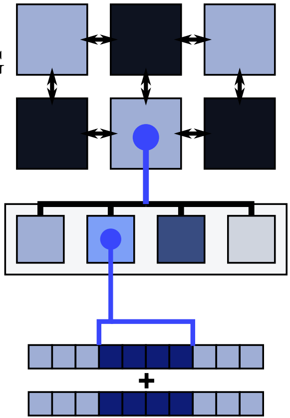
```
MPI_Sendrecv(data, k,
   MPI_DOUBLE, data2,
   ... );
```

MULTITHREADING
in shared memory

```
#pragma omp parallel for
for (j = 0; j < m; j++)
  ComputeSubset(j);
```

VECTORIZATION
of floating-point math

```
#pragma omp simd
for (i = 0; i < n; i++)
  A[i] += B[i];
```



§2. Message Passing Interface, MPI

Message Passing Interface, MPI

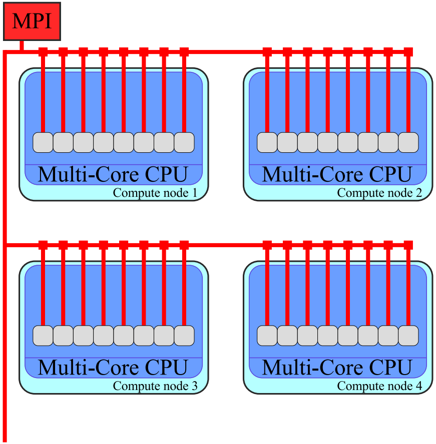
MPI

- Specification for message passing
- Multiple implementations exist

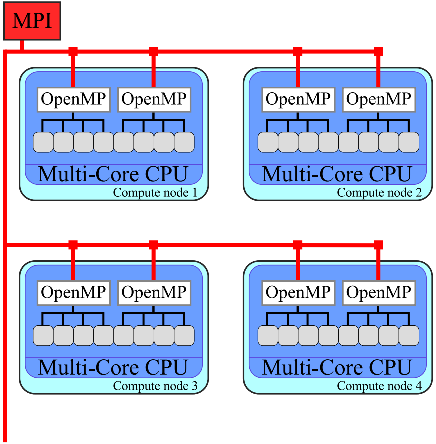
- Portable
- Efficient
- Designed for computing

- Distributed-memory computing
- Multiprocessing in shared memory

Hybrid MPI+OpenMP



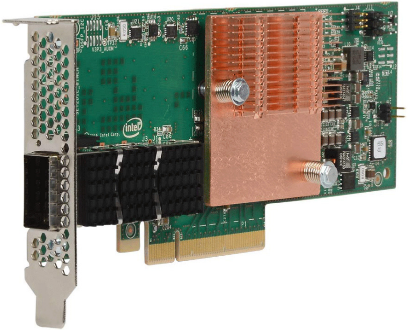
Works for low core counts



Necessary for multi-core CPUs

Intel's HPC Communication Fabric

Intel Omni-Path Architecture - low-latency, high-bandwidth, scalable communication fabric for HPC applications.



Discrete

Integrated

§3. Programming with MPI

Structure of MPI Applications: Hello World

```
#include "mpi.h"
#include <cstdio>
int main (int argc, char *argv[]) {
  MPI_Init (&argc, &argv);  // Initialize MPI envirnmnt
  int rank, size, namelen;
  char name[MPI_MAX_PROCESSOR_NAME];
  MPI_Comm_rank (MPI_COMM_WORLD, &rank); // ID of current process
  MPI_Get_processor_name (name, &namelen); // Hostname of node
  MPI_Comm_size (MPI_COMM_WORLD, &size); // Number of processes
  printf ("Hello World from rank %d running on %s!\n", rank, name);
  if (rank == 0) printf("MPI World size = %d processes\n", size);
  MPI_Finalize (); // Terminate MPI environment
}
```

MPICH site contains a list of MPI 3.2 routines

§4. Compiling and Running with MPI

Compiling and Running MPI Applications on Localhost

```
u100@c005% mpiicpc -o HelloMPI HelloMPI.cc
```

Command file `mympi`:

```
#PBS -l nodes=1
cd $PBS_O_WORKDIR
mpirun -host localhost -np 2 ./HelloMPI
```

Results:

```
u100@c005% qsub mympi
2000
u100@c005% cat mympi.o2000
Hello World from rank 1 running on c005-n001!
Hello World from rank 0 running on c005-n001!
MPI World size = 2 processes
```

Running MPI Applications on Several Hosts

Command file `mydistmpi`:

```
#PBS -l nodes=2
cd $PBS_O_WORKDIR
cat $PBS_NODEFILE
mpirun -machinefile $PBS_NODEFILE ./HelloMPI
```

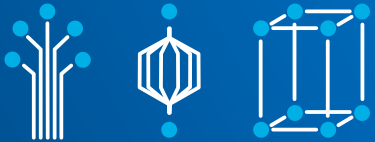
```
u100@c005% qsub mydistmpi
2001
u100@c005% cat mydistmpi.o2001
c005-n001
c005-n002
Hello World from rank 1 running on c005-n002!
Hello World from rank 0 running on c005-n001!
MPI World size = 2 processes
```

§5. Peer-to-Peer Messaging

Point to Point Communication

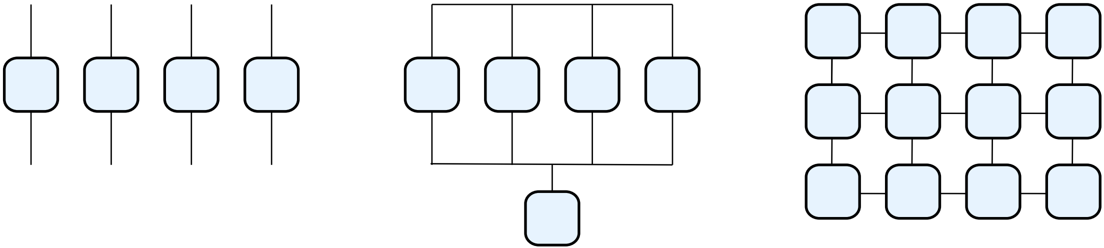
```
if (rank == sender) {

  char outMsg[msgLen];
  strcpy(outMsg, "Hi There!");
  MPI_Send(&outMsg, msgLen, MPI_CHAR, receiver, tag, MPI_COMM_WORLD);

} else if (rank == receiver) {

  char inMsg[msgLen];
  MPI_Recv (&inMsg, msgLen, MPI_CHAR, sender, tag, MPI_COMM_WORLD, &stat);
  printf ("Received message with tag %d: '%s'\n", tag, inMsg);

}
```

§6. Collective Communication



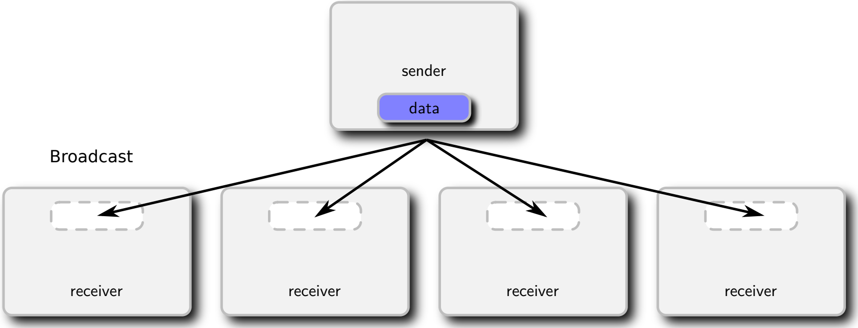
Parallel Patterns

Common parallel patterns call for collective communication



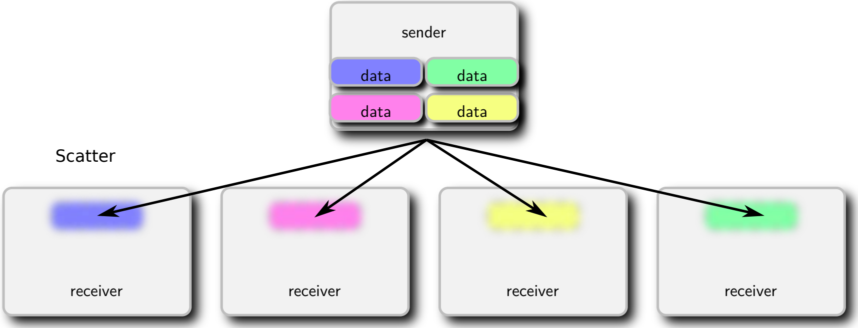
Collective Communication: Broadcast

```
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype,
    int root, MPI_Comm comm );
```



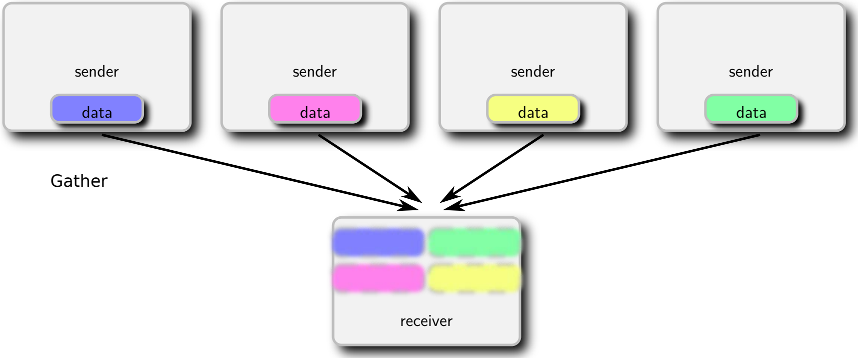
Collective Communication: Scatter

```
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recv
    int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm);
```



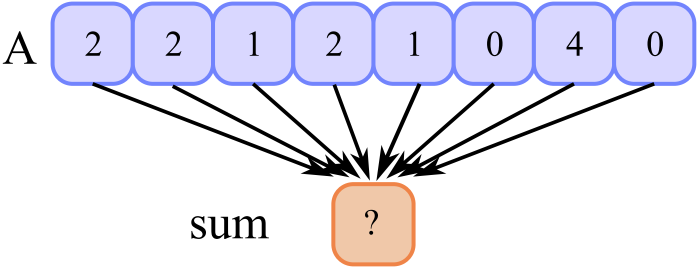
Collective Communication: Gather

```
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype,
    void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm co
```



Collective Communication: Reduction

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype
    MPI_Op op, int root, MPI_Comm comm);
```



Available reducers: max/min, minloc/maxloc, sum, product,
AND, OR, XOR (logical or bitwise).

§7. Example: Stencil Code

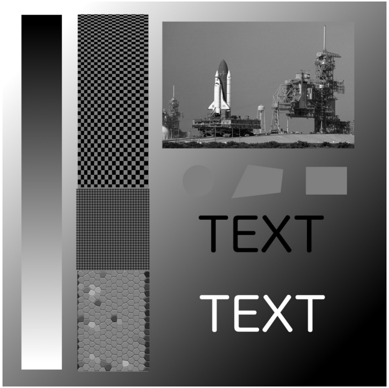
Stencil Operators

- Linear systems of equations
- Partial differential equations

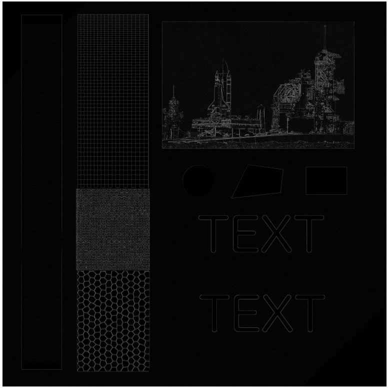
$$
\begin{aligned}
Q_{x,y} = \; & c_{00} P_{\mathrm{x-1,y-1}} && + \; c_{01} P_{\mathrm{x,y-1}} && + \; c_{02} P_{\mathrm{x+1,y-1}} && + \\
& c_{10} P_{\mathrm{x-1,y}} && + \; c_{11} P_{\mathrm{x,y}} && + \; c_{12} P_{\mathrm{x+1,y}} && + \\
& c_{20} P_{\mathrm{x-1,y+1}} && + \; c_{21} P_{\mathrm{x,y+1}} && + \; c_{22} P_{\mathrm{x+1,y+1}} &&
\end{aligned}
$$

Fluid dynamics, heat transfer, image processing (convolution matrix), cellular automata.

Edge Detection



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \rightarrow$$

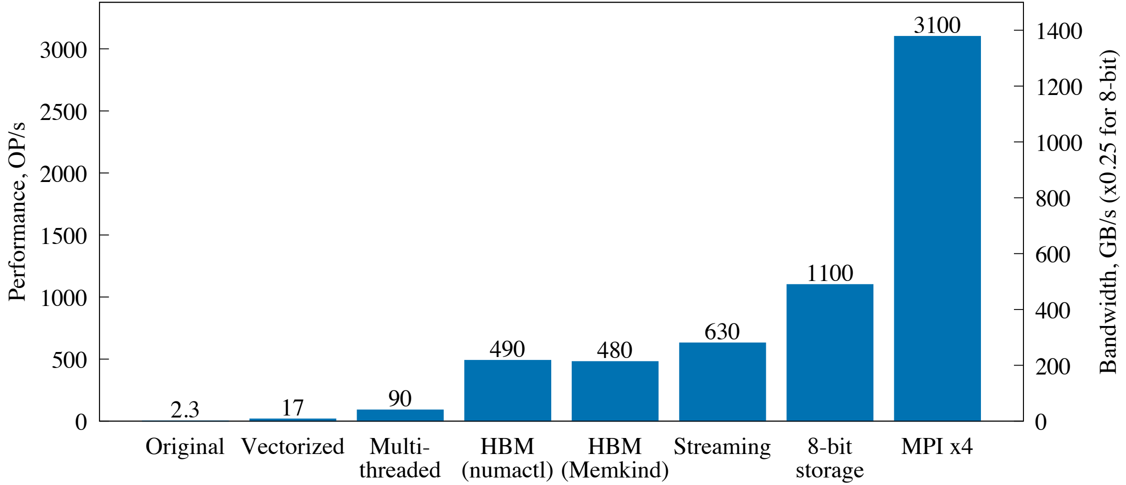


Clustering with MPI

```
#PBS -l nodes=4:flat
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE ./stencil test-image.png
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
MPI_Comm_size(MPI_COMM_WORLD, &nRanks);

const double rowsPerProcess = double(height-2)/double(nRanks);
const int myFirstRow = 1 + int(rowsPerProcess*myRank);
const int myLastRow  = 1 + int(rowsPerProcess*(myRank+1));

#pragma omp parallel for
for (int i = myFirstRow; i < myLastRow; i++)
  ...
```

Performance



§8. Example: Numerical Integration

Midpoint Rectangle Method

$$I(a,b) = \int_0^a f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n-1} f\left(x_{\mathrm{i}+\frac{1}{2}}\right)\Delta x,$$

where

$$\Delta x = \frac{a}{n}, \qquad x_{\mathrm{i}+\frac{1}{2}} = \left(i + \frac{1}{2}\right)\Delta x.$$

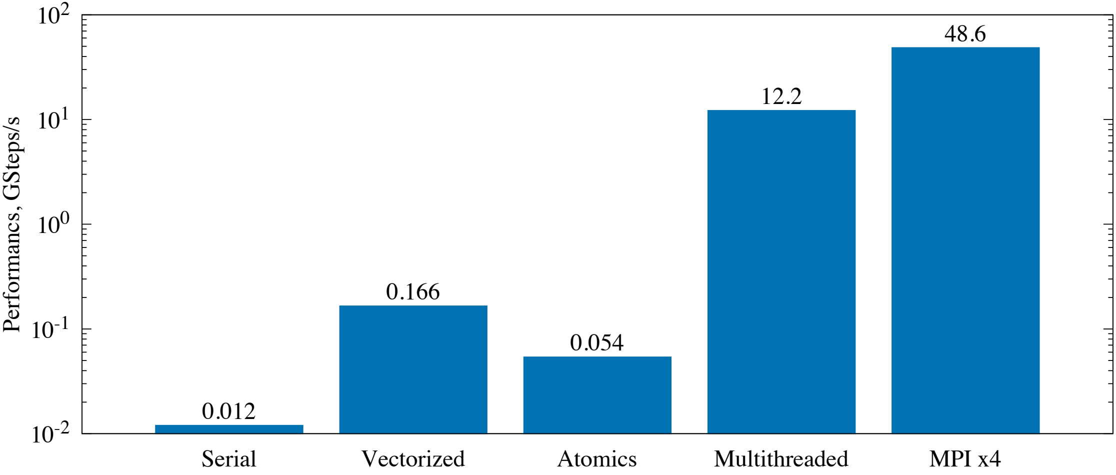
Single-node Implementation

```
const double dx = a/(double)n;
double integral = 0.0;
#pragma omp parallel for simd reduction(+: integral)
for (int i = 0; i < n; i++) {
  const double xip12 = dx*((double)i + 0.5);
  const double dI = BlackBoxFunction(xip12)*dx;

  integral += dI;
}
```

Multi-node Implementation

```
const int iStart = double(n)/double(nRanks)*double(rank);
const int iEnd   = double(n)/double(nRanks)*double(rank+1);

const double dx = a/(double)n;
double integral_partial = 0.0, integral = 0.0;
#pragma omp parallel for simd reduction(+: integral_partial)
for (int i = iStart; i < iEnd; i++) {
  const double xip12 = dx*((double)i + 0.5);
  const double dI = BlackBoxFunction(xip12)*dx;
  integral_partial += dI;
}

MPI_Allreduce(&integral_partial, &integral, 1, MPI_DOUBLE,
              MPI_SUM, MPI_COMM_WORLD);
```

Performance



§9. Learn More

MPI Concepts and Functions

Communication modes – (non-)blocking, (a)synchronous, ready mode

Message buffers – application, system, user space

Communicators, Groups – creation, manipulation, usage

Data types – built-in, derived

Collective communication – patterns

Hybrid programming – co-existence with threads: safety, performance

One-sided communication – remote memory access

Click for links from the MPI Forum.
More information in our book and MPI tutorial from the LLNL

Summary on MPI

- Framework for distributed-memory programming
- Hides from developer complexity of programming a variety of fabrics
- Collective communication may use functions of the fabric
- Intel Omni-Path Architecture is a native solution for Xeon Phi
- Intel tool for tuning load balance, communication: ITAC