Parallel Programming for Distributed Memory Systems

Mikhail Brinskiy

Agenda

- Motivation
- MPI
  - Point-to-point communication
  - Collective communication
  - One-sided communication
- PGAS
- Conclusion

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

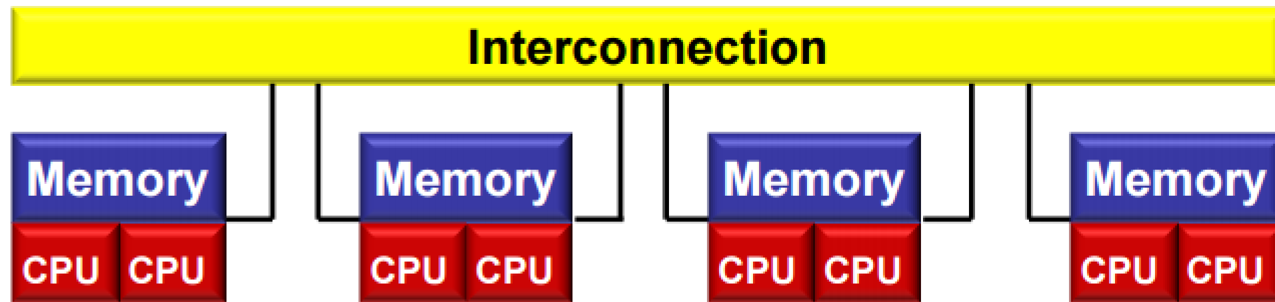
Why Parallel Programming?

- Hardware limitations:
  - CPU frequency
  - Memory size and speed
  - etc
  - So, multi-core architectures got popular

- Application requirements are increasing:
  - Scientific problem sizes become larger
  - Better accuracy required
  - New kinds of scientific problems arise

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Distributed Memory



- Data exchange between memory of different CPUs
  - Via interconnect
  - Requires explicit data transfer (message passing)

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

HPC: the current trends (1/2)

SEQUOIA	
BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	
CORES	1,572,864
RMAX (TFLOP/S)	17,173.2
RPEAK (TFLOP/S)	20,132.7
POWER (KW)	7,890

TITAN	
Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x	
CORES	560,640
RMAX (TFLOP/S)	17,590.0
RPEAK (TFLOP/S)	27,112.5
POWER (KW)	8,209

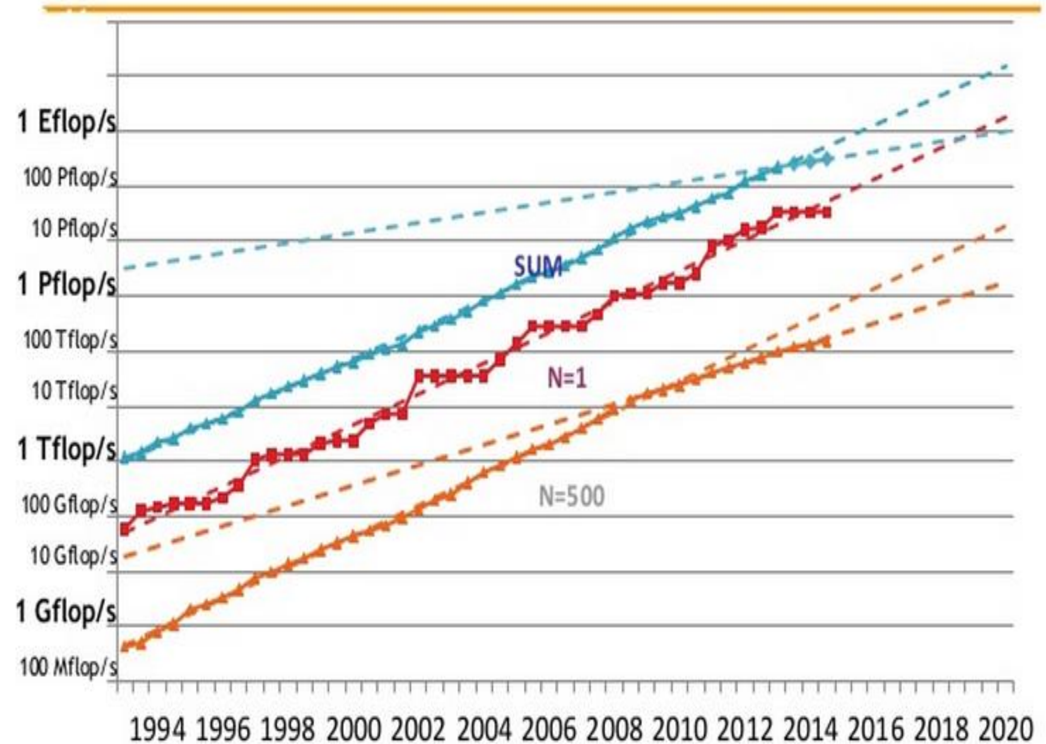
TIANHE-2 (MILKYWAY-2)	
TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P	
CORES	3,120,000
RMAX (TFLOP/S)	33,862.7
RPEAK (TFLOP/S)	54,902.4
POWER (KW)	17,808

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

HPC: the current trends (2/2)

Exascale* challenges:

- Reliability
- Power
- Scalability



*One exaflops is a quintillion, or 10^{18}, floating point operations per second

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

What is MPI?

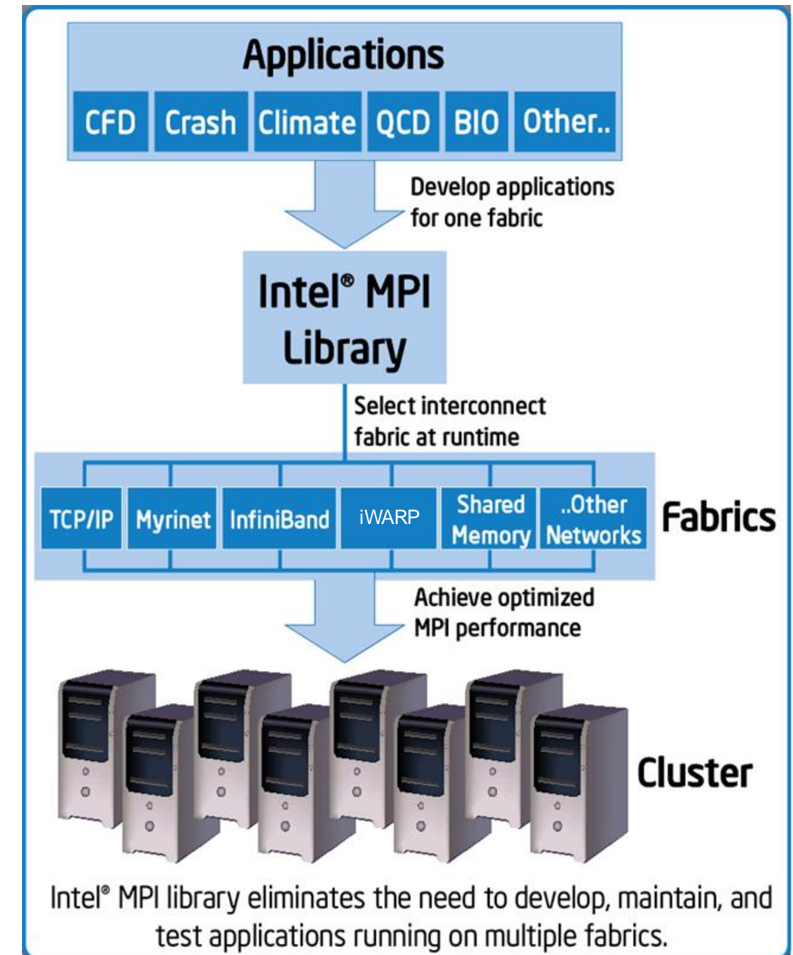
- MPI – Message Passing Interface
- Version 1.0 of the standard was released in June 1994
- Current version is MPI 3.1
- Provides language independent API for point-to-point, collectives and many other operations across distributed memory systems
- Many implementations exist (MPICH, Open MPI, Cray, IBM, Fujitsu, HPC-X, etc)

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel® MPI Library

- Optimized MPI application performance
  - Application-specific tuning
  - Automatic tuning
- Faster MPI communication
  - Optimized collectives
- Support of various interconnects
  - Novel Intel® Omni-Path Architecture
  - Mellanox InfiniBand*
  - Ethernet
  - etc
- Sustainable scalability beyond 340K cores
  - Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements



Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

MPI basis

- MPI provide a powerful, efficient and portable way for parallel programming
- MPI was explicitly designed to enable libraries
- Typically MPI supports SPMD model (MPMD possible though), i. e. same sub-program runs on each processor. The total program (all sub-programs of the program) must be started with the MPI startup tool.
- MPI program talks by means of messages (not streams)
- MPI contains quite rich API:
  - MPI Environment
  - Point-to-Point communication
  - Collective communication
  - One-sided communication (Remote Memory Access)
  - MPI Datatypes
  - Application topologies
  - Profiling interface
  - File I/O
  - Dynamic Processes

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

MPI Program

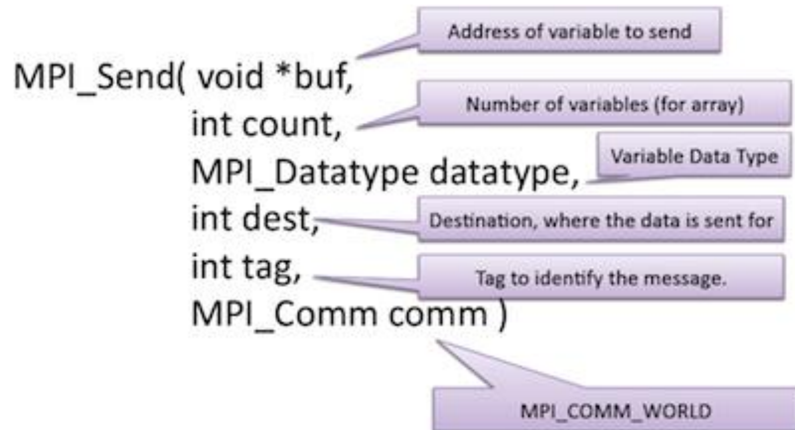
```c
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
   MPI_Init( &argc, &argv );
   printf( "Hello, world!\n" );
   MPI_Finalize();
   return 0;
}
```

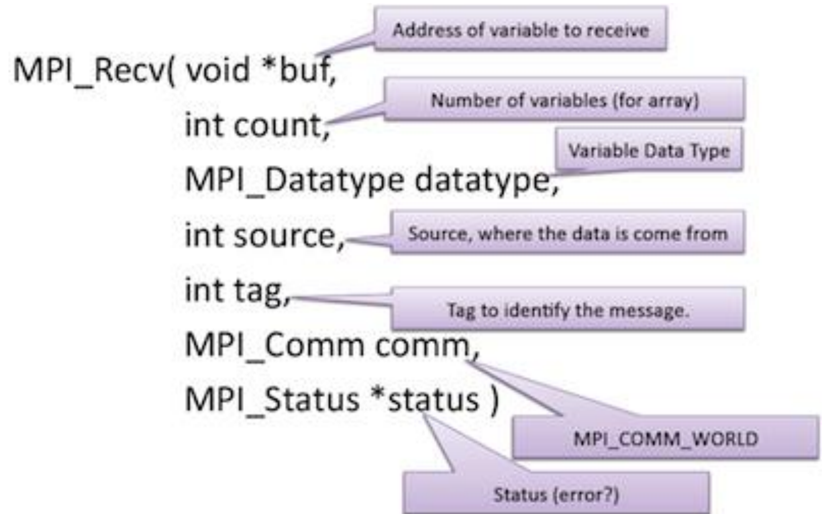
```fortran
program main
use MPI
integer ierr

call MPI_INIT( ierr )
print *, 'Hello, world!'
call MPI_FINALIZE( ierr )
end
```


Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication





- Messages are matched by triplet of source, tag and communicator
- Tag is just a message mark (MPI_Recv may provide MPI_ANY_TAG to match message with any tag)
- MPI_Recv may receive from any process by using MPI_ANY_SOURCE as source
- Communicator represents two things: the group of processes and communication context

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication

Ok, we have the following program:

```
if (rank == 0) {
      buf[0] = 19;
      buf[1] = 31;
      MPI_Send (&buf[0], 1, MPI_INT,    1, 55, MPI_COMM_WORLD);
      MPI_Send (&buf[1], 4, MPI_CHAR , 1, 55, MPI_COMM_WORLD);
 } else {
      buf[0] = buf[1] = -1;
      MPI_Recv (&buf[0], 4, MPI_CHAR, 0, 55, MPI_COMM_WORLD, &stat);
      MPI_Recv (&buf[1], 1, MPI_INT,    0, 55, MPI_COMM_WORLD, &stat);
      printf ("buf[0] = %d, buf[1] = %d \n", buf[0], buf[1]);
 }
```

What will be the output?

buf[0] = ?, buf[1] = ?

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication

Ok, we have the following program:

```
if (rank == 0) {
    buf[0] = 19;
    buf[1] = 31;
    MPI_Send (&buf[0], 1, MPI_INT,   1, 55, MPI_COMM_WORLD);
    MPI_Send (&buf[1], 4, MPI_CHAR , 1, 55, MPI_COMM_WORLD);
} else {
    buf[0] = buf[1] = -1;
    MPI_Recv (&buf[0], 4, MPI_CHAR, 0, 55, MPI_COMM_WORLD, &stat);
    MPI_Recv (&buf[1], 1, MPI_INT,  0, 55, MPI_COMM_WORLD, &stat);
    printf ("buf[0] = %d, buf[1] = %d \n", buf[0], buf[1]);
}
```

What will be the output?

buf[0] = **19**, buf[1] = **31**

Message matching check source, tag and communicator, right? MPI does **not** care about datatypes matching

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication

Ok, we have the following program (assuming MPI_Send will not block):

```
if (rank == 0) {
      buf[0] = 19;
      buf[1] = 31;
      MPI_Send (&buf[0], 1, MPI_INT,    1, 55, MPI_COMM_WORLD);
      MPI_Send (&buf[1], 4, MPI_CHAR, 1, 56, MPI_COMM_WORLD);
 } else if (rank == 1){
      buf[0] = buf[1] = -1;
      MPI_Recv (&buf[0], 4, MPI_CHAR, 0, 56, MPI_COMM_WORLD, &stat);
      MPI_Recv (&buf[1], 1, MPI_INT,    0, 55, MPI_COMM_WORLD, &stat);
      printf ("buf[0] = %d, buf[1] = %d \n", buf[0], buf[1]);
 }
```

What will be the output?

buf[0] = ?, buf[1] = ?

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication

Ok, we have the following program:

```
if (rank == 0) {
      buf[0] = 19;
      buf[1] = 31;
      MPI_Send (&buf[0], 1, MPI_INT,     1, 55, MPI_COMM_WORLD);
      MPI_Send (&buf[1], 4, MPI_CHAR , 1, 56, MPI_COMM_WORLD);
 } else if (rank == 1) {
      buf[0] = buf[1] = -1;
      MPI_Recv (&buf[0], 4, MPI_CHAR, 0, 56, MPI_COMM_WORLD, &stat);
      MPI_Recv (&buf[1], 1, MPI_INT,     0, 55, MPI_COMM_WORLD, &stat);
      printf ("buf[0] = %d, buf[1] = %d \n", buf[0], buf[1]);
 }
```

What will be the output?

buf[0] = **31**, buf[1] = **19**

Message matching check source, **tag** and communicator, right?

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Point-to-Point Communication

Actually there are more Point-to-point functions (look thru the MPI standard – it's interesting!!!):

MPI_Bsend() (*buffered send*): buffers the message, so the call does not depend on the receiver. Thus it is a local operation

MPI_Ssend() (*synchronous send*): blocks until receiver confirms reception of the message

MPI_Rsend() (*ready send*): The user should ensure that matching receive already posted. Like rendezvous protocol, but without handshaking. The aim of this function is potential performance benefit, by eliminating rendezvous overhead.

All of them have their own corresponding NB counterparts.

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Process Groups

- It is an ordered set of unique MPI processes
- Every process can be a member of arbitrary number of groups
- Every process has an index id in the particular group. This id is known to be a "rank" of the interested MPI process. Ranks are contiguous and start from 0
- Quite rich API for managing groups: unions, intersections, comparisons, inclusions, etc. This provides basis for communicators

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Communicators

Communicator represents two entities:

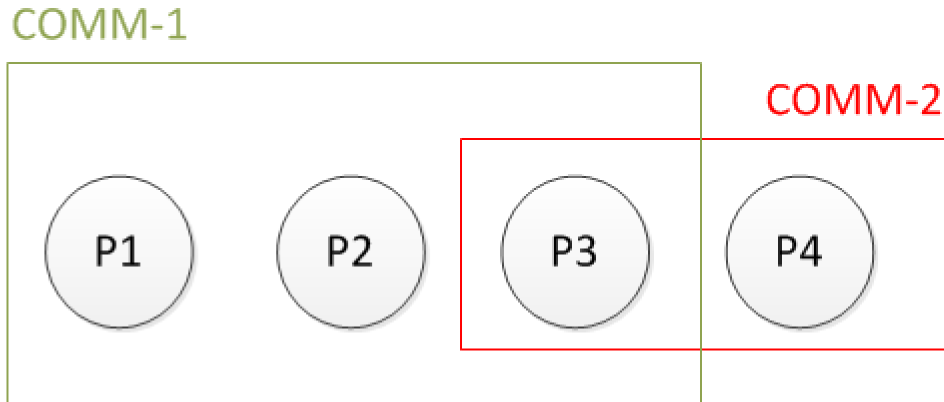
- Process group
- Communication context (one of the pillars in matching triplet). Provides partitioning of the communication space

No wildcards allowed, but there are two pre-defined communicators:

- MPI_COMM_WORLD: contains all MPI processes
- MPI_COMM_SELF: contains just a process itself

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Communicators



What would be the arguments for:

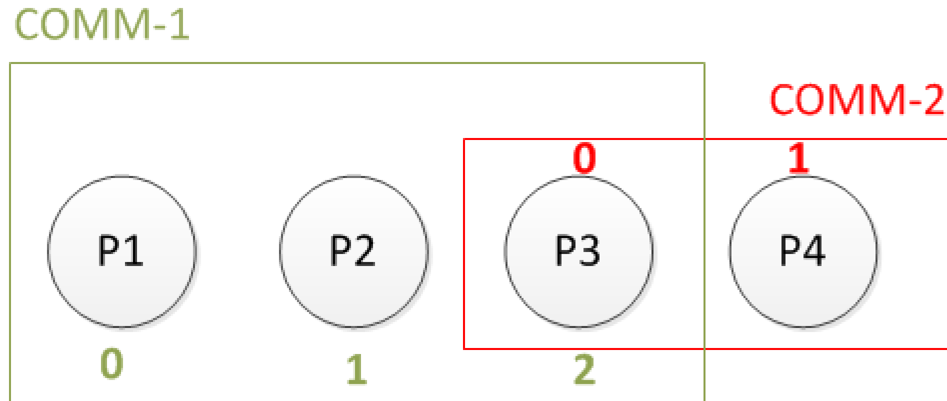
P1---->P**?**: MPI_Send (&buf, 1, MPI_INT, **2**, some_tag, **COMM-1**);

P4---->P**?**: MPI_Send (&buf, 1, MPI_INT, **0**, some_tag, **COMM-2**);

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Communicators



What would be the arguments for:

P1---->P**3**: MPI_Send (&buf, 1, MPI_INT, 2, some_tag, COMM-1);

P4---->P**3**: MPI_Send (&buf, 1, MPI_INT, 0, some_tag, COMM-2);

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Communicators

Take aways:

- Communicator defines an ordered set of unique processes with unique context id
- Communications performed on different comms do not interfere
- Communicator defines the scope of collective operations: collective operation must be called by all processes from the given comm
- MPI_COMM_WORLD is pre-defined comm, containing all processes
- Every process may have different ranks in different comms
- Regular point-to-point and collective operations performed on the same comm do not interfere
- Two types of comms: intra and inter. Intra is commonly used, inter is kinda an advanced topic

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

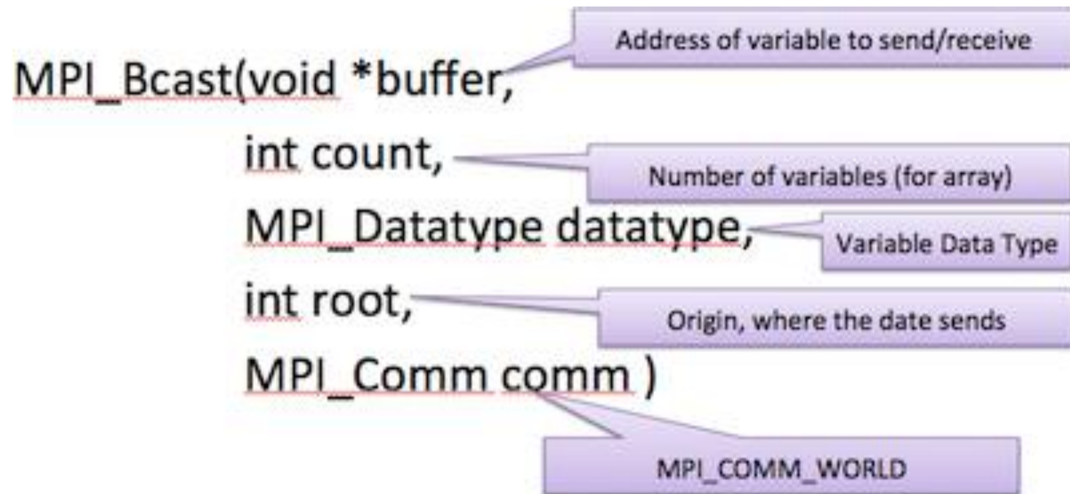
Collective communication

- Represent different communication patterns, which may involve an arbitrary number of ranks
- Why would not plain send and receive be enough? O-p-t-i-m-i-z-a-t-i-on
- All collective operations involve every process in a given communicator
- MPI implementations may contain several algorithms for every collective
- Typically based on point-to-point functionality (but not necessary)
- Can be divided into 3 categories: one-to-all, all-to-one, all-to-all
- There are regular, nonblocking and neighbor collectives.

Optimization Notice

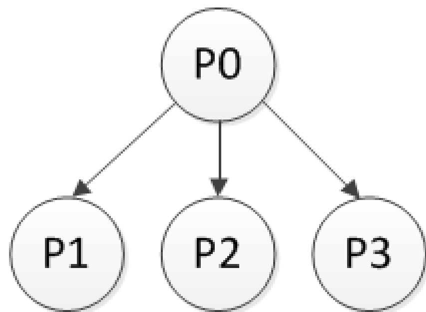
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Collective communication

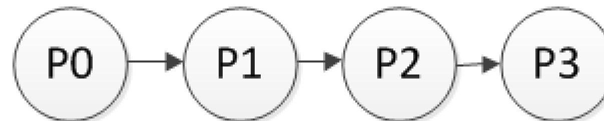


MPI_Bcast: one process (root) sends some chunk of data to the rest of processes in the given communicator

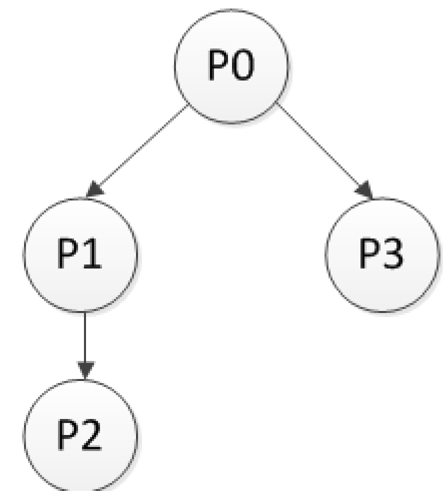
Possible algorithms:



(a) flat tree



(b) chain (ring)



(c) binomial

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

intel

Collective communication

Busting general myths:

- Collective operations DO NOT guarantee processes synchronization (except MPI_Barrier which is intended for that). However some of them synchronize
- Many of the collective operations/algorithms implies unequal amount of work per every process
- MPI_Barrier guarantees not to return until all processes in the given communicator have entered the barrier
- Be aware that most of the MPI implementation tune their collective algorithms for different platforms, so different algorithms may be chosen by MPI for the same program executing on different HW

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Collective communication

Interesting facts:

- Some of collective operations have the corresponding vector counterpart. For instance MPI_Gatherv allows each rank to send different amount of data to the root (MPI_Gather assumes that every rank send equal portion of data)
- These vector counterparts usually have “v” in the end of its name (e.g., MPI_Scatter<->MPI_Scatterv). One exception is MPI_Reduce_scatter<->MPI_Reduce_scatter_block
- MPI_Alltoallw is the only operation which allows ranks to specify different datatypes

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

One-sided communication

- One process to specify all communication parameters, both for the sending side and for the receiving side
- Separate communication and synchronization
- No matching
- Process that initiates a one-sided communication operation is the *origin process* and the process that contains the memory being accessed is the *target process*
- Memory is exposed via *window* concept
- Quite rich API: a bunch of different window creation, communication, synchronization and atomic routines,
- Example of communication call:

```
MPI_Put(const void *origin_addr,  int origin_count,  MPI_Datatype    origin_datatype,
int target_rank,   MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype,
        MPI_Win win)
```


Optimization Notice

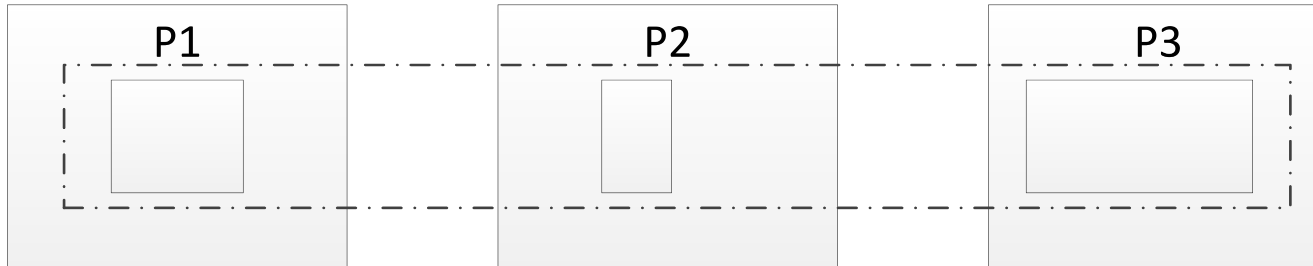
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

One-sided communication

Window

Memory that a process allows other processes to access via one-sided communication is called a window

Group of processes specify their local windows to other processes by calling the collective function (i. e. MPI_Win_create, MPI_Win_allocate, etc)



Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

One-sided communication

Epoch

- Accesses must occur within an epoch
  - Lock(window, rank) … Unlock(window, rank)
  - Access mode can be exclusive or shared
  - Operations are not ordered within an epoch (except MPI_Accumulate calls)
- An epoch starts within an RMA synchronization call
- RMA communication calls must occur only within an *access epoch*
- In active mode a target window can be accessed by RMA operations only within an *exposure epoch*
- In passive mode there is no concept of *exposure epoch*

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

One-sided communication

Active target communication

- Both processes are explicitly involved in the communication - similar to 2-sided message passing, except that
  - all the data transfer arguments are provided by one process,
  - the second process only participates in the synchronization.

Passive target communication

- Only the origin process is explicitly involved in the transfer
- Two origin processes may communicate by accessing the same location in a target window
- This communication paradigm is closest to a shared memory model, where shared data can be accessed by all processes, irrespective of location

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Passive mode

```
int b1 = 1, b2 = 2;
int winbuf = -1;
MPI_Win win;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Win_create(&winbuf, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win );

if (rank == 0)
{
   MPI_Win_lock(MPI_LOCK_SHARED, 1, 0, win);
   MPI_Put(&b1, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
   MPI_Put(&b2, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
   MPI_Win_unlock( 1, win );
}
else if (rank == 1)
{
   printf("My win %d\n", winbuf);
}
```


Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

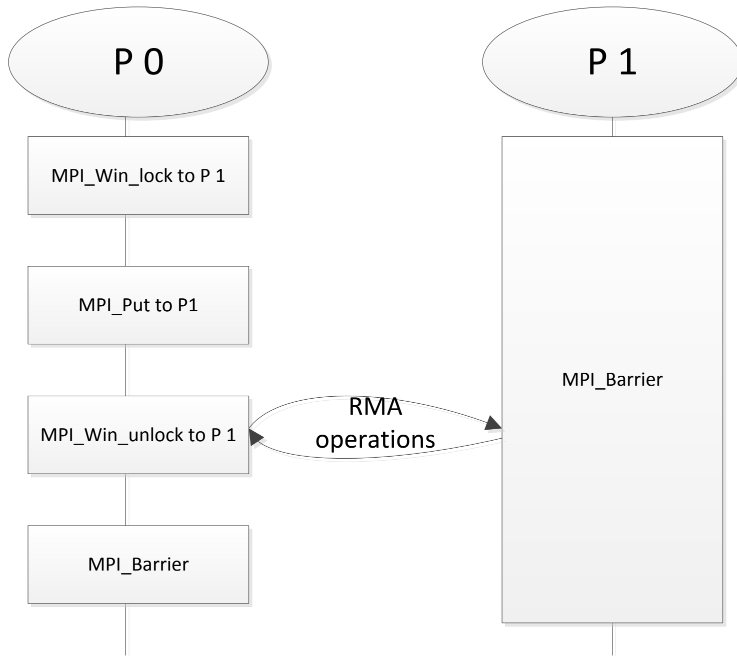
Passive mode

```
int b1 = 1, b2 = 2;
int winbuf = -1;
MPI_Win win;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Win_create(&winbuf, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win );

if (rank == 0)
{
   MPI_Win_lock(MPI_LOCK_SHARED, 1, 0, win);
   MPI_Put(&b1, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
   MPI_Put(&b2, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
   MPI_Win_unlock( 1, win );
}
MPI_Barrier(MPI_COMM_WORLD);
if (rank == 1)
{
   printf("My win %d\n", winbuf);
}
```

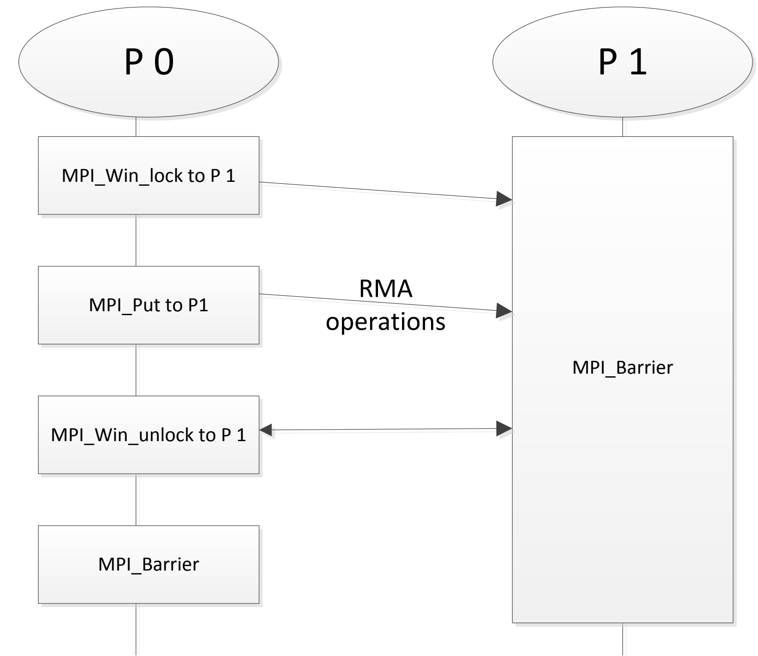

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Passive mode



(a)



(b)

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Active mode

```
int b1 = 1,
int winbuf = -1;
MPI_Win win;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Win_create(&winbuf, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD,
&win );

MPI_Win_fence( 0, win );
if (rank == 0)
{
   MPI_Put(&b1, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
}
MPI_Win_fence( 0, win );
if (rank == 1)
{
   printf("My win %d\n", winbuf);
}
```

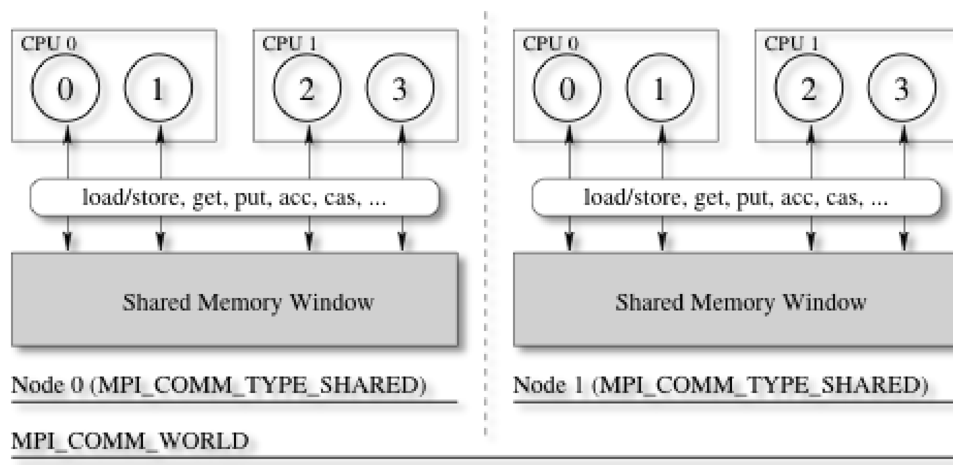

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Active mode

```
int b1 = 1,
int winbuf = -1;
MPI_Win win;
MPI_Group group;

MPI_Comm_group(MPI_COMM_WORLD, &group);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Win_create(&winbuf, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD,
&win );

if (rank == 0){
   MPI_Win_start(group, 0, win);
   MPI_Put(&b1, 1, MPI_INT, 1, 0, 1, MPI_INT, win );
   MPI_Win_complete(win);
}  else if (rank == 1) {
   MPI_Win_post(group, 0, win);
   MPI_Win_wait(win);
   printf("My win %d\n", winbuf);
}
```


Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

MPI+MPI: Shared Memory Windows




```
MPI_Comm_split_type(TYPE_SHARED)
MPI_Win_allocate_shared(shr_comm)
MPI_Win_shared_query(&baseptr)

MPI_Win_lock_all(shr_win)
// access baseptr[]
MPI_Win_sync()
// ...
MPI_Win_unlock_all(shr_win)
```

- Leverage RMA to incorporate node-level programming
- RMA provides portable atomics, synchronization, …
- Eliminates “X” in MPI+X, when only shared memory is needed
- Memory/core is not increasing
- Allows NUMA-aware mapping
- Each window piece is associated with the process that allocated it

http://htor.inf.ethz.ch/publications/img/mpi_mpi_hybrid_programming.pdf

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

PGAS

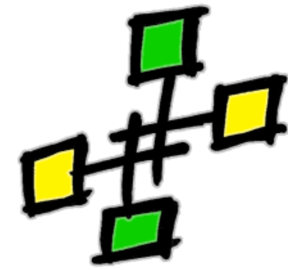
Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

GPI and MCTP

- GPI - Global Address Space Programming Interface (Implements GASPI standard)
- MCTP - Multi-Core Thread Package
- Developed by Fraunhofer Institute for Industrial Mathematics ITWM GPI has been evolving since 2005 (it was called FVM:Fraunhofer Virtual Machine prior 2009)
- GPI is intended for distributed memory systems. MCTP supplements GPI on the node level.
- GPI/MCTP completely replaced MPI in Fraunhofer Institutes
- They have big customers like Shell, some government assignments
- About 40 people working in HPC area in Fraunhofer

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

OpenSHMEM



- Specification API for programming in the PGAS (plus reference implementation of this API). Specification is about 80 pages
- It is an attempt to standardize different SHMEM implementations (Cray, SGI, Quadrics, HP, IBM, etc)
- It is a library (like MPI), available for C and Fortran
- Main features: one-sided communication, natural overlap of computation and communication, atomic memory operations, collective calls, etc
- There are several implementations available:
  - Mellanox – based on OpenMPI
  - TSHMEM - Over Tilera Tile-Gx architecture and libraries (many core processes). Developed prototype to deal with xeon phi, which will be integrated to TSHMEM soon
  - OpenSHMEM on top of MPI-3. First official release in Jan 2014. Shows quite poor performance values

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

CAF

- CAF (Coarray Fortran) was an extension of Fortran 95/2003, since 1990s
- Fortran 2008 standard includes coarrays, but with slightly different than original CAF syntax
- Utilize concept of images, each image executes the same program independently from the others
- Coarray Fortran is usually implemented on top of MPI for better portability
- Coarray Fortran 2.0 is being developed by the Rice University. It includes several additional features compared to the emerging Fortran 2008
- Several Implementations exist:
  - Cray CAF
  - Los Alamos Computer Science Institute
  - Intel Fortran Compiler XE supports CAF
  - OpenUH compiler supports coarrays (in the form of Fortran 2008)

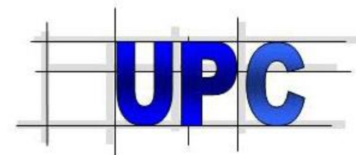
Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

intel

UPC

- UPC (Unified Parallel C): is an extension of the C language, which assumes SPMD programming model
- The desired amount of threads (UPC entity) can be specified either at compile time or in run-time
- UPC distinguishes two types of data: private and shared. Shared data accessible from all UPC threads.
- Also UPC provides some collectives, parallel I/O, synchronization primitives, etc
- The latest specification version is 1.3 (16 Nov 2013)
- Several Implementations exist:
  - GWU UPC: also provides UPC specification as well as UPC collective and parallel I/O specifications
  - Berkeley UPC
  - GCC UPC
  - Florida UPC
  - MTU UPC



Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Some More PGAS

- Chapel: PGAS language, core language (not an extension)
- X10: Object oriented programming language (IBM, open sourced). CAN Use MPI as a network transport and MPI collectives : upcoming in X-10 2.4.1 (dec 2013)
- XcalableMP (U. of Tsukuba, U. of Tokyo, U of Kyoto, etc) – directive based language extension similar to OpenMP. SPMD execution model
- Charm++: C++ based language
- ARMCI (Aggregate Remote Memory Copy Interface): library
- Global Arrays: library
- etc

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Questions?

Pavan Balaji (Computer Scientist and Group Lead Argonne National Laboratory):

- *I don't know what tomorrow's scientific computing language will look like, but I know it will be called Fortran*
- *I don't know what tomorrow's parallel programming model will look like, but I know it will be called MPI (+X)*

(http://seec.cs.vt.edu/files/2014-07-25-vt-exampi.pdf)

Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Links

- Parallel Computing Why & How?
- Parallel Programming Introduction to MPI and OpenMP
- Introduction to the Message Passing Interface
- How I Learned to Stop Worrying about Exascale and Love MPI
- An Introduction to MPI Parallel Programming with the Message Passing Interface
- PGAS tutorial
- And Others

Optimization Notice

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

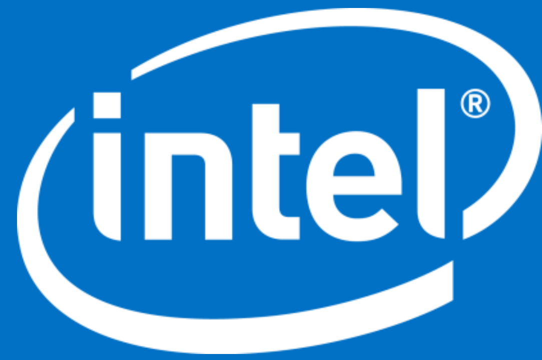
Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.


intel