MPI Programming on Intel Xeon Phi™ Coprocessors

Silvio Stanzani , Raphael Cóbe , Rogério Iope

UNESP - Núcleo de Computação Científica

silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br

Agenda

- Overview
- Using Intel® MPI
- Programming Models
- Hybrid Computing

Agenda

- **Overview**
- Using Intel® MPI
- Programming Models
- Hybrid Computing

Enabling & Advancing Parallelism
High Performance Parallel Programming

- Intel tools, libraries and parallel models extend to multicore, many-core and heterogeneous computing

Code

Compiler
Libraries
Parallel Models

intel Software

Multicore

Many-core

Cluster

Multicore CPU

Multicore CPU

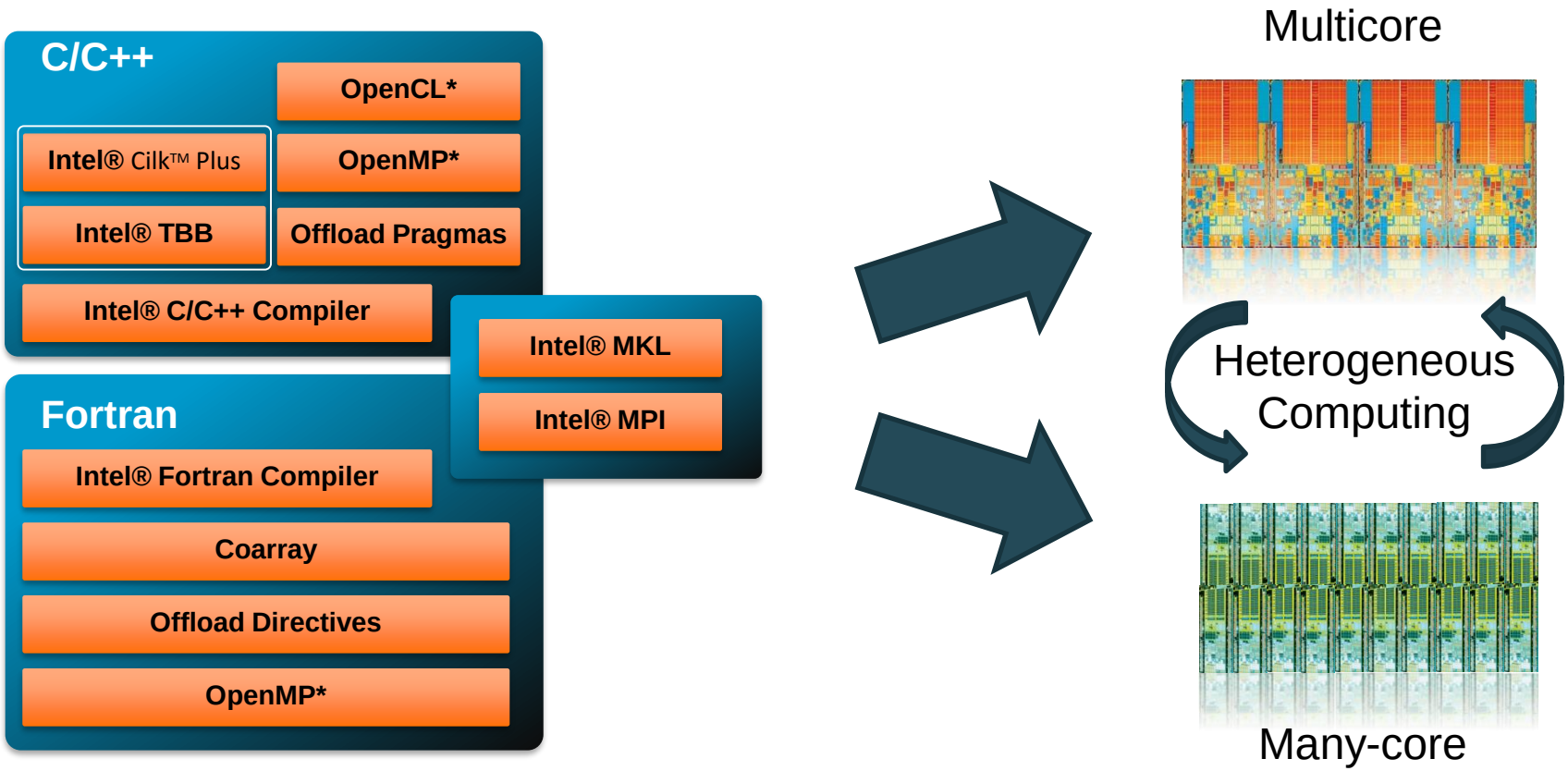
Intel® MIC Architecture Coprocessor

Multicore Cluster

Multicore & Many -core Cluster

Use One Software Architecture Today. Scale Forward Tomorrow.

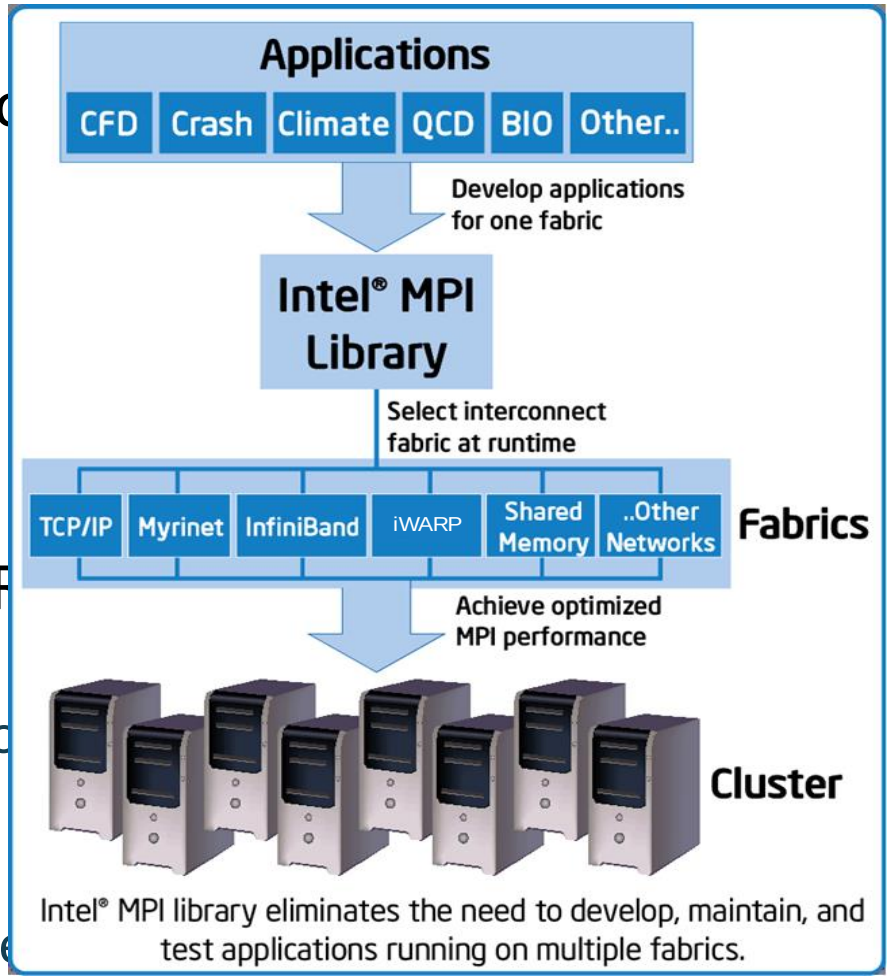
Preserve Your Development Investment
Common Tools and Programming Models for Parallelism

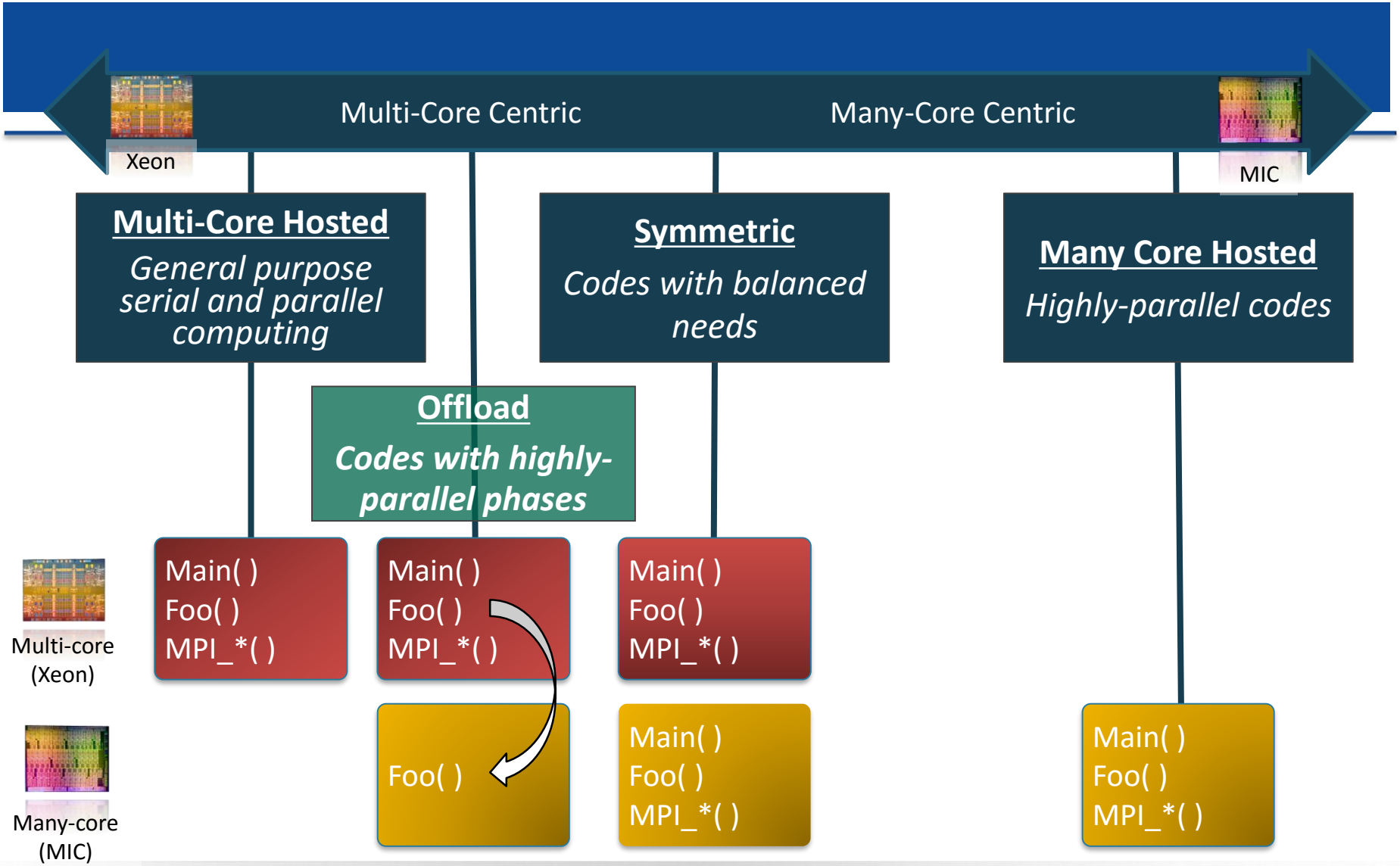


Develop Using Parallel Models that Support Heterogeneous Computing

Intel MPI Library Overview

- Intel is a leading vendor of MPI implementations and tools
- Optimized MPI application perfo
  - Application-specific tuning
  - Automatic tuning
- Lower Latency
  - ❑ Industry leading latency
- Interconnect Independence & F
  - Multi-vendor interoperability
  - Performance optimized support fo through DAPL 2.0
- More robust MPI applications
  - Seamless interoperability with Inte





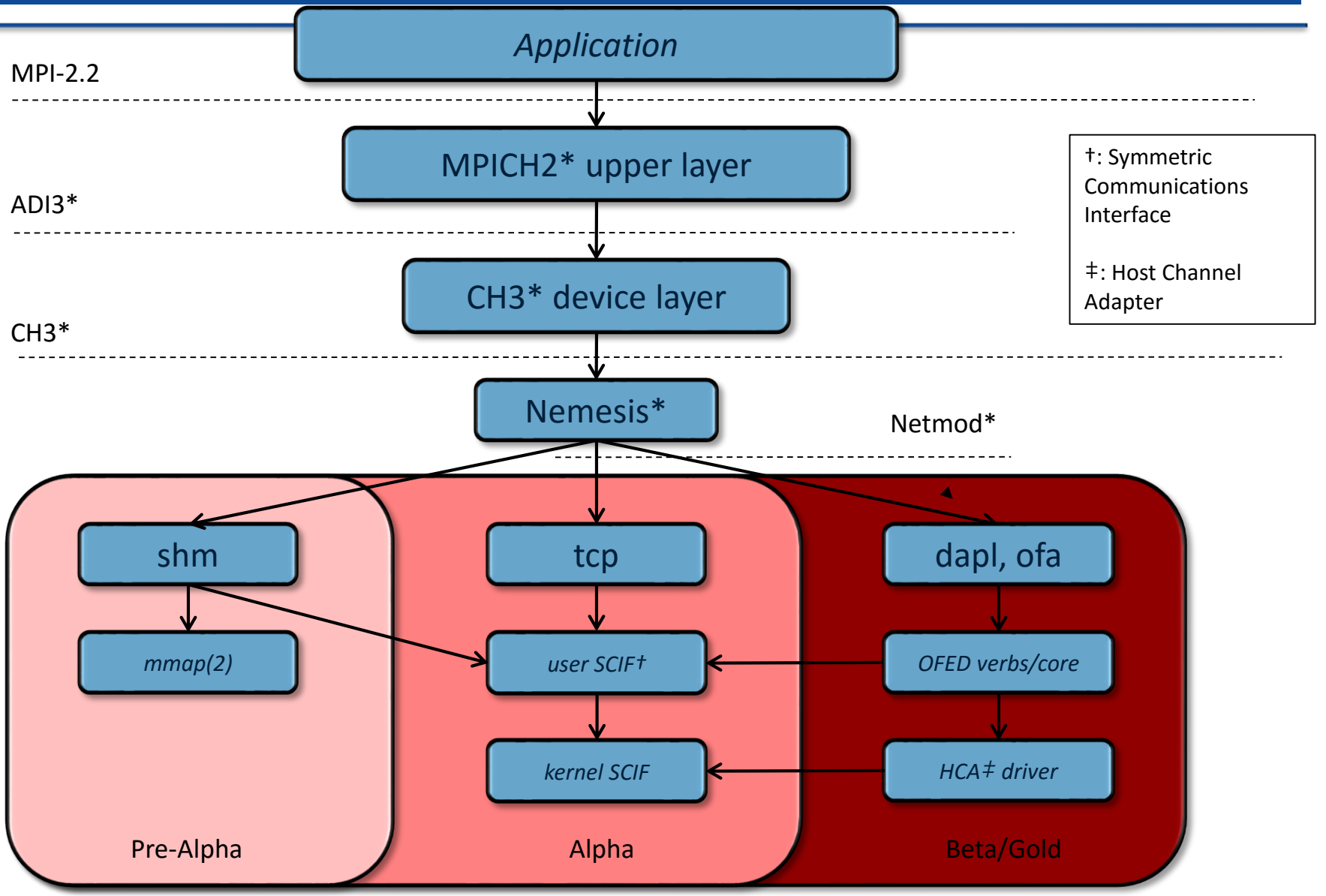
Range of models to meet application needs

Levels of communication speed

- Current clusters are not homogenous regarding communication speed:
  - Inter node (Infiniband, Ethernet, etc)
  - Intra node
    - Inter sockets (Quick Path Interconnect)
    - Intra socket

- Two additional levels to come with MIC coprocessor:
  - Host-MIC communication
  - Inter MIC communication

Intel® MPI Library Architecture & Staging



Selecting network fabrics

- Intel® MPI selects automatically the best available network fabric it can find.
  - Use I_MPI_FABRICS to select a different communication device explicitly
- The best fabric is usually based on Infiniband (dapl, ofa) for inter node communication and shared memory for intra node
- Available for KNC:
  - shm, tcp, ofa, dapl
  - Availability checked in the order shm:dapl, shm:ofa, shm:tcp (intra:inter)
- Set I_MPI_SSHM_SCIF=1 to enable shm fabric between host and MIC

Agenda

- Overview
- **Using Intel® MPI**
- Programming Models
- Hybrid Computing

Prerequisites per User

- Set the compiler environment
  - # source <compiler_installdir>/bin/compilervars.sh intel64
  - Identical for Host and MIC
- Set the Intel® MPI environment
  - # source /opt/intel/impi/4.1.0.018/intel64/bin/mpivars.sh
  - Identical for Host and MIC
- mpiexec needs ssh access to MIC!
  - ❑ Done! User‘s ssh key ~/.ssh/id_rsa.pub is copied to MIC at driver boot time.
- Debug I_MPI_DEBUG
  - ❑ Print out debugging information when an MPI program starts running. I_MPI_DEBUG=<level>[,<flags>]

Compiling and Linking for MIC

- Compile MPI sources using Intel® MPI scripts
  - For Xeon with potential offload (latest compiler)
  - # mpiicc –o test test.c
  - For Xeon with potential offload (older compiler)
  - # mpiicc –offload-build –o test test.c
  - For Xeon without potential offload as usual
  - # mpiicc [-no-offload] –o test test.c
  - For native execution on MIC add „–mmic“ flag,
    i.e. the usual compiler flag controls also the MPI compilation
  - # mpiicc –mmic –o test test.c

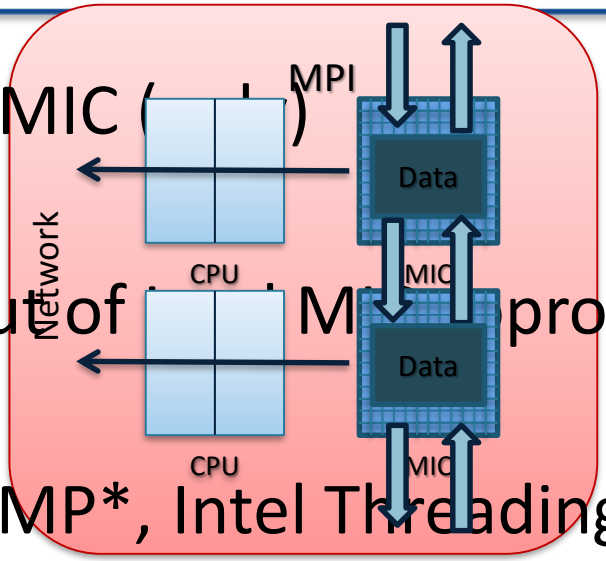
- Linker verbose mode “-v” shows
  - Without „–mmic“ linkage with intel64 libraries:
  - ld ... -L/opt/intel/impi/4.1.0.018/intel64/lib ...
  - With „–mmic“ linkage with MIC libraries:
  - ld ... -L/opt/intel/impi/4.1.0.018/mic/lib ...

Agenda

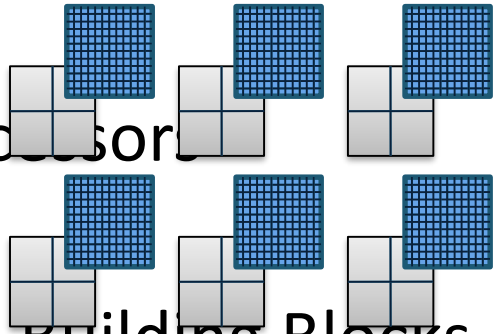
- Overview
- Using Intel® MPI
- **Programming Models**
- Hybrid Computing

Coprocessor-only Programming Model

- MPI ranks on Intel® MIC (
- All messages into/out of
- Intel Cilk Plus, OpenMP*, Intel Threading Building Blocks, Pthreads used directly within MPI processes
- Intermediate step: All MPI processes run on 1 Intel MIC Architecture only



Homogenous network of many-core CPUs



Build Intel® MIC binary using Intel® MIC compiler.

Upload the binary to the Intel® MIC Architecture.

Run instances of the MPI application on Intel® MIC nodes.

Coprocessor-only Programming Model

- MPI ranks on the MIC coprocessor(s) only
- MPI messages into/out of the MIC coprocessor(s)
- Threading possible

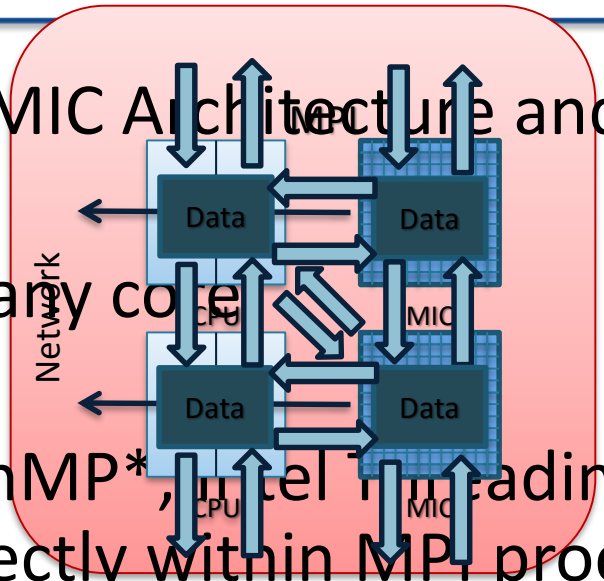
- Build the application for the MIC Architecture
- `# mpiicc -mmic -o test_hello.MIC test.c`
- Upload the MIC executable
- `# scp ./test_hello.MIC mic0:/tmp/test_hello.MIC`
  - Remark: If NFS available no explicit uploads required (just copies)!
- Launch the application on the coprocessor from host
- `# mpiexec -n 2 -wdir /tmp -host mic0 /tmp/test_hello.MIC`
- Alternatively: login to MIC and execute the already uploaded mpiexec.hydra there!

Coprocessor-only Programming Model

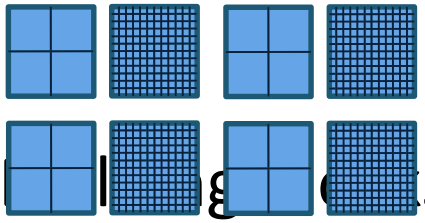
```
#include "mpi.h"
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[]) {
  int i, rank, size, namelen;

  char name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name (name, &namelen);

  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 ();
}
```

Symmetric Programming Model

- MPI ranks on Intel MIC Architecture and host CPUs
- Messages to/from any core
- Intel Cilk Plus, OpenMP*, Intel Threading Building Blocks, Pthreads* used directly within MPI processes
- Intermediate step: All MPI processes run on 1 host CPU and 1 Intel MIC Architecture only
- Available in Intel MPI Library for Intel MIC Alpha (1 host, 1 coprocessor).



Heterogeneous network of homogeneous CPUs



Build Intel® 64 and Intel® MIC Architecture binaries by using the resp. compilers targeting Intel® 64 and Intel® MIC Architecture.

Upload the Intel® MIC binary to the Intel® MIC Architecture.

Run instances of the MPI application on different mixed nodes.

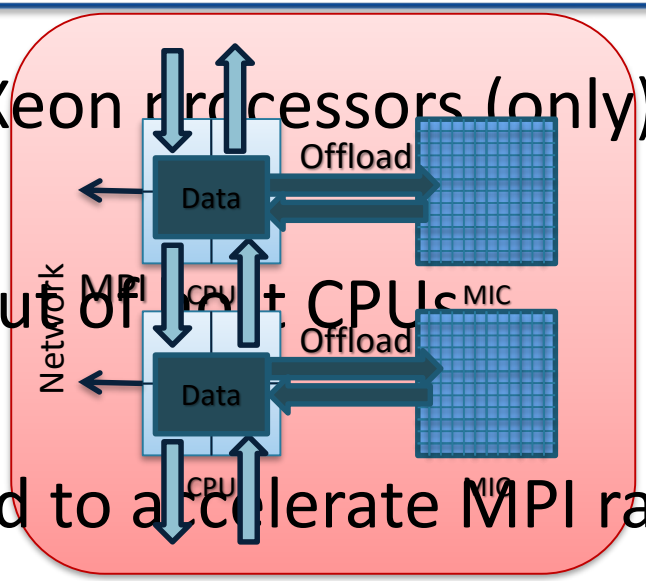
Symmetric model

- MPI ranks on the MIC coprocessor(s) and host CPU(s)
- MPI messages into/out of the MIC(s) and host CPU(s)
- Threading possible

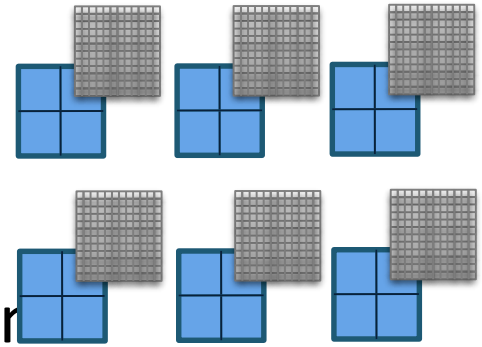
- Build the application for Intel®64 and the MIC Architecture separately
- `# mpiicc -o test_hello test.c`
- `# mpiicc -mmic -o test_hello.MIC test.c`
- Upload the MIC executable
- `# scp ./test_hello.MIC mic0:/tmp/test_hello.MIC`
- Launch the application on the host and the coprocessor from the host
- `# mpiexec -n 2 -host <hostname> ./test_hello : -wdir /tmp -n 2 -host mic0 /tmp/test_hello.MIC`

MPI+Offload Programming Model

- MPI ranks on Intel Xeon processors (only)
- All messages into/out of Intel CPUs
- Offload models used to accelerate MPI ranks
- Intel Cilk Plus, OpenMP*, Intel Threading Building Blocks, Pthreads* within Intel MIC



Homogenous network of heterogeneous nodes



Build Intel® 64 executable with included offload by using the Intel® 64 compiler.

Run instances of the MPI application on the host, offloading code onto MIC.

Advantages of more cores and wider SIMD for certain applications

MPI+Offload Programming Model

```
#include "mpi.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#include <unistd.h>

int main (int argc, char *argv[]) {
  int i, rank, size, namelen;

  char name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name (name, &namelen);

  printf ("Hello World from rank %d running on %s!\n", rank, name);

  if (rank == 0) {
    printf("MPI World size = %d processes\n", size);
int thid;
  char hn[600];
  char hn2[600];

  #pragma offload target(mic)
  {
   gethostname(hn2,600);

   #pragma omp parallel private(thid)
   {
    thid=omp_get_thread_num();
    printf("P10 hello from thread %d host %s rank %d
     \n",thid, hn2, rank);
   }
  }
 }
 MPI_Finalize ();
}
```

MPI+Offload Programming Model

- MPI ranks on the host CPUs only
- MPI messages into/out of the host CPUs
- Intel MIC Architecture as an accelerator

- Compile for MPI and internal offload
- `# mpiicc -o test test.c`
- Latest compiler compiles by default for offloading if offload construct is detected!
  - Switch off by `-no-offload` flag
  - Previous compilers needed `-offload-build` flag
- Execute on host(s) as usual
- `# mpiexec -n 2 ./test`
- MPI processes will offload code for acceleration

Offloading to Intel MIC Architecture Examples

- C/C++ Offload Pragma
- #pragma offload target (mic)
- #pragma omp parallel for reduction(+:pi)
- for (i=0; i<count; i++) {
- float t = (float)((i+0.5)/count);
- pi += 4.0/(1.0+t*t);
- }
- pi /= count;

- MKL Implicit Offload
- //MKL implicit offload requires no source code
- changes, simply link with the offload MKL Library.

- MKL Explicit Offload
- #pragma offload target (mic) \
- in(transa, transb, N, alpha, beta) \
- in(A:length(matrix_elements)) \
- in(B:length(matrix_elements)) \
- in(C:length(matrix_elements)) \
- out(C:length(matrix_elements)alloc_if(0))
- sgemm(&transa, &transb, &N, &N, &N, &alpha,
- A, &N, B, &N, &beta, C, &N);

Fortran Offload Directive

```
!dir$ omp offload target(mic)
!$omp parallel do
      do i=1,10
      A(i) = B(i) * C(i)
      enddo
!$omp end parallel
```

C/C++ Language Extensions

```
class _Shared common  {
    int data1;
    char *data2;
    class common *next;
    void process();
};
_Shared class common  obj1, obj2;

…
_Cilk_spawn _Offload obj1.process();
_Cilk_spawn          obj2.process();

…
```

Agenda

- Overview
- Installation of Intel® MPI
- Programming Models
- **Hybrid Computing**

Traditional Cluster Computing

- MPI is »the« portable cluster solution

- Parallel programs use MPI over cores inside the nodes
  - Homogeneous programming model
  - "Easily" portable to different sizes of clusters
  - No threading issues like »False Sharing« (common cache line)
  - Maintenance costs only for one parallelization model

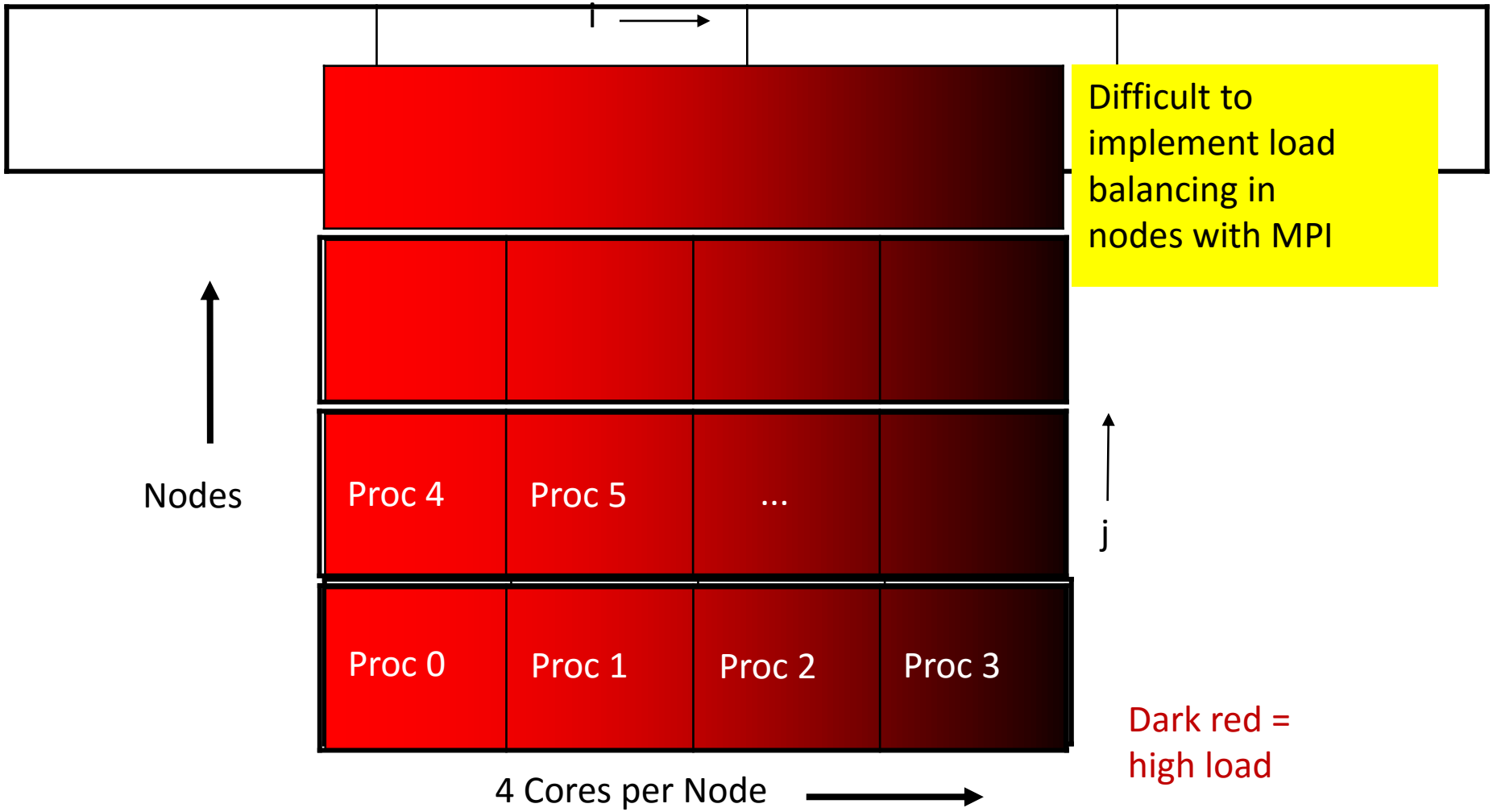
Traditional Cluster Computing (contd.)

- Hardware trends
  - Increasing number of cores per node - plus cores on coprocessors
  - Increasing number of nodes per cluster
- Consequence: Increasing number of MPI processes per application
- Potential MPI limitations
  - Memory consumption per MPI process, sum exceeds the node memory
  - Limited scalability due to exhausted interconnects (e.g. MPI collectives)
  - Load balancing is often challenging in MPI

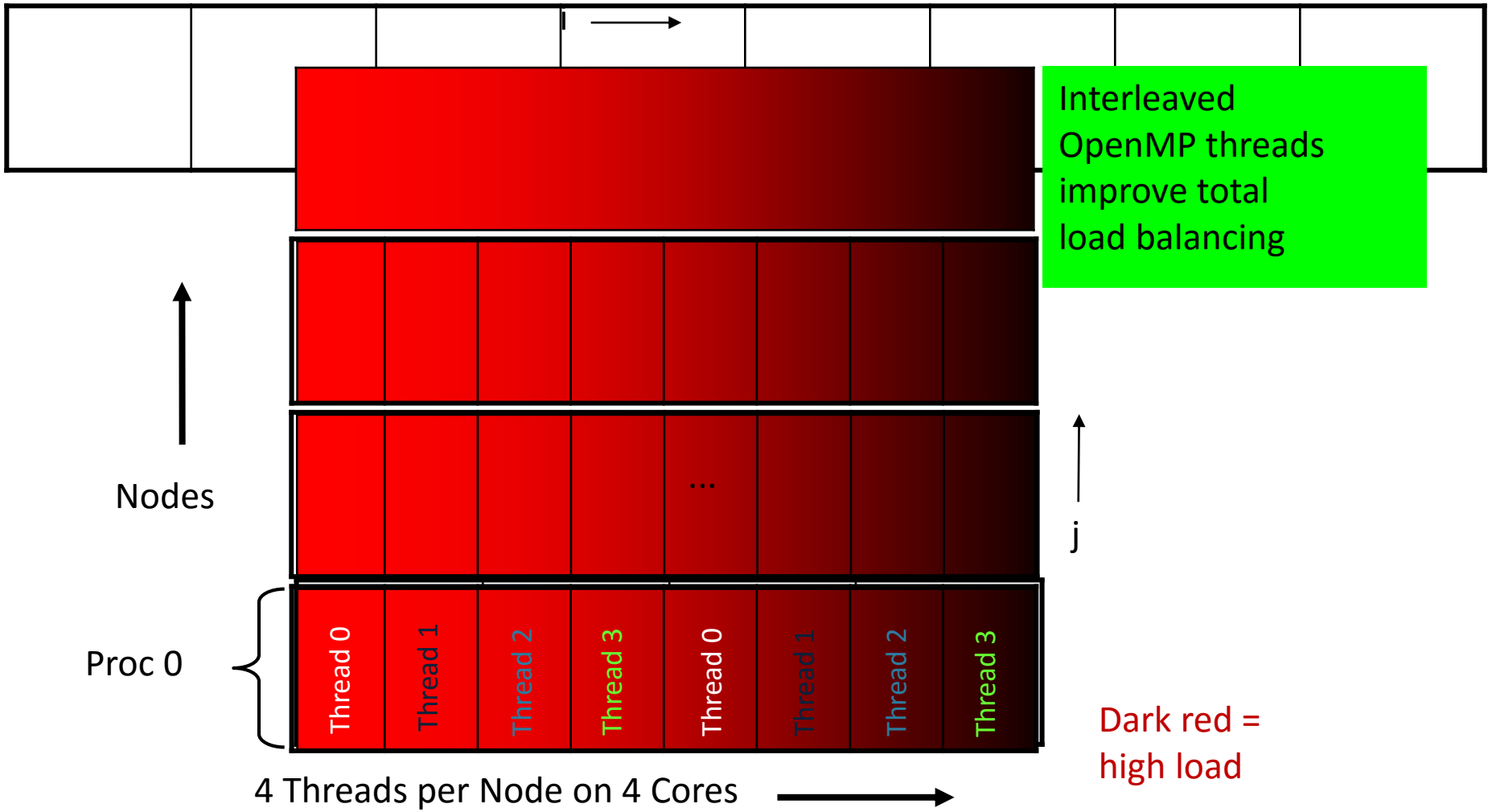
Hybrid Computing

- Combine MPI programming model with threading model
- Overcome MPI limitations by adding threading:
  - Potential memory gains in threaded code
  - Better scalability (e.g. less MPI communication)
  - Threading offers smart load balancing strategies
- Result: Maximize performance by exploitation of hardware (incl. coprocessors)

Example: MPI Load Imbalance



Example: Hybrid Load Balance



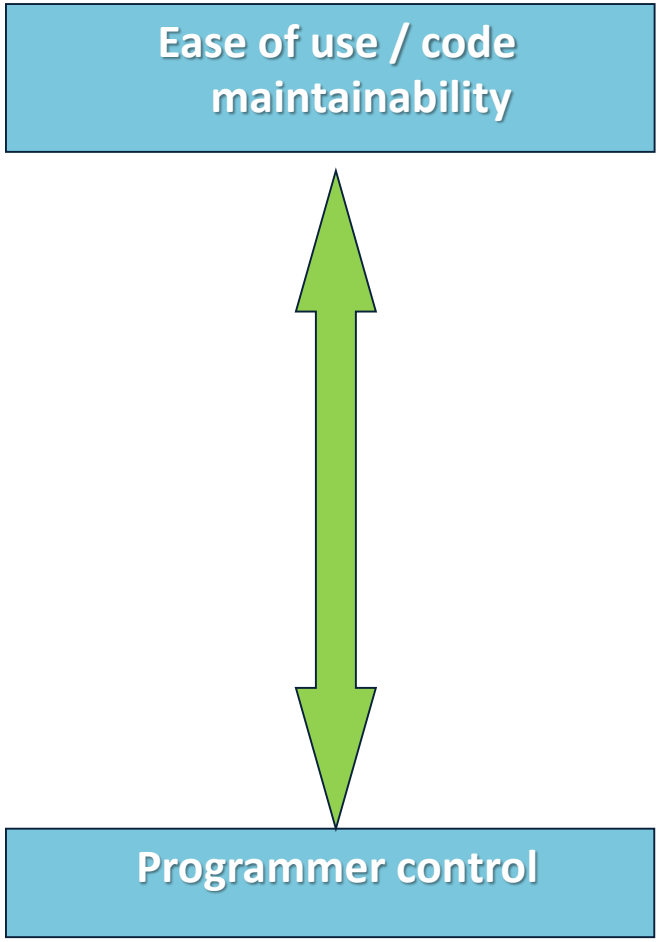
Options for Thread Parallelism

Intel® Math Kernel Library

OpenMP*

Intel® Threading Building Blocks
Intel® Cilk™ Plus

Pthreads* and other threading libraries



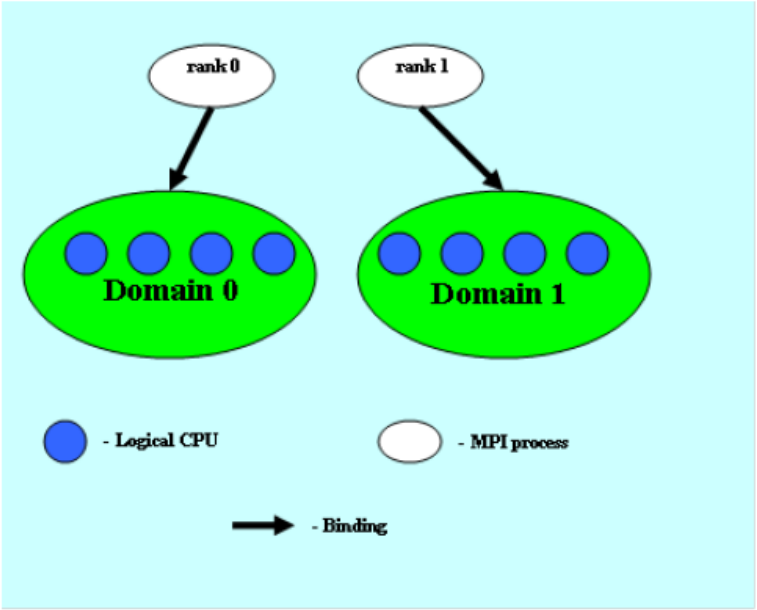
Choice of unified programming to target Intel® Xeon and Intel® MIC Architecture!

Intel® MPI Support of Hybrid Codes

- Intel MPI is strong in mapping control
- Sophisticated default or user controlled
  - I_MPI_PIN_PROCESSOR_LIST for pure MPI
  - For hybrid codes (takes precedence):
  - I_MPI_PIN_DOMAIN =<size>[:<layout>]
  - <size> =
    omp Adjust to OMP_NUM_THREADS
    auto #CPUs/#MPIprocs
    <n> Number
  - <layout> =
    platform According to BIOS numbering
    compact Close to each other
    scatter Far away from each other
- Naturally extends to hybrid codes on MIC

Intel® MPI Support of Hybrid Codes

- Define I_MPI_PIN_DOMAIN to split logical processors into non-overlapping subsets
- Map



er 1 domain

Pin OpenMP threads inside the domain with KMP_AFFINITY (or in the code)

I_MPI_PIN_DOMAIN

- I_MPI_PIN_DOMAIN=<multi-core-shape>

<mc-shape>	Define domains through multi-core terms.
core	Each domain consists of the logical processors that share a particular core. The number of domains on a node is equal to the number of cores on the node.
socket | sock	Each domain consists of the logical processors that share a particular socket. The number of domains on a node is equal to the number of sockets on the node. This is the recommended value.
node	All logical processors on a node are arranged into a single domain.
cache1	Logical processors that share a particular level 1 cache are arranged into a single domain.
cache2	Logical processors that share a particular level 2 cache are arranged into a single domain.
cache3	Logical processors that share a particular level 3 cache are arranged into a single domain.
cache	The largest domain among cache1, cache2, and cache3 is selected.

I_MPI_PIN_DOMAIN

- I_MPI_PIN_DOMAIN=<Explicit shape <size>:<layout>>

<size>	Define a number of logical processors in each domain (domain size)
omp	The domain size is equal to the OMP_NUM_THREADS environment variable value. If the OMP_NUM_THREADS environment variable is not set, each node is treated as a separate domain.
auto	The domain size is defined by the formula size=#cpu/#proc, where #cpu is the number of logical processors on a node, and #proc is the number of the MPI processes started on a node
<n>	The domain size is defined by a positive decimal number *<n>*

<layout>	Ordering of domain members. The default value is compact
platform	Domain members are ordered according to their BIOS numbering (platform-depended numbering)
compact	Domain members are located as close to each other as possible in terms of common resources (cores, caches, sockets, etc.). This is the default value
scatter	Domain members are located as far away from each other as possible in terms of common resources (cores, caches, sockets, etc.)

Intel® MPI Environment Support

- The execution command mpiexec of Intel® MPI reads argument sets from the command line:
  - Sections between „:“ define an argument set
    (also lines in a configfile, but not yet available in Beta)
  - Host, number of nodes, but also environment can be set independently in each argument set
  - # mpiexec –env I_MPI_PIN_DOMAIN 4 –host myXEON ...
    : -env I_MPI_PIN_DOMAIN 16 –host myMIC

- Adapt the important environment variables to the architecture
  - OMP_NUM_THREADS, KMP_AFFINITY for OpenMP
  - CILK_NWORKERS for Intel® CilkTM Plus

Intel® MPI Environment Support

- #debug
- export I_MPI_DEBUG=4

- #mapping MPI only compact
- mpirun -env I_MPI_PIN_DOMAIN auto:compact -env I_MPI_DEBUG 4 -host mic0 -n 4 /tmp/mpitest

- #mapping MPI only compact
- mpirun -env I_MPI_PIN_DOMAIN auto:scatter -env I_MPI_DEBUG 4 -host mic0 -n 4 /tmp/mpitest

- #thread per core
- mpirun -env I_MPI_PIN_DOMAIN 3 -env I_MPI_DEBUG 4 -host localhost -n 2 ./mpiExample/mpiExample : -env I_MPI_PIN_DOMAIN 4 -env I_MPI_DEBUG 4 -host mic0 -n 2 /tmp/mpiExample.mic

Coprocessor-only and Symmetric Support

- Full hybrid support on Intel® Xeon from Intel ® MPI extends to Intel ® MIC
- KMP_AFFINITY=balanced (only on MIC) in addition to scatter and compact

- Recommendations:
  - Explicitly control where MPI processes and threads run in a hybrid application
    (according to threading model and application)
  - Avoid splitting cores among MPI processes, i.e.
    I_MPI_PIN_DOMAIN should be a multiple of 4
  - Try different KMP_AFFINITY settings for your application

Mpi openmp

```
#include "mpi.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <omp.h>
#include <unistd.h>

int main (int argc, char *argv[]) {
  int i, rank, size, namelen;

  char name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name (name, &namelen);

  printf ("Hello World from rank %d running on %s!\n", rank, name);

  if (rank == 0)
    printf("MPI World size = %d processes\n", size);
int thid;
 char hn[600];
 char hn2[600];

 gethostname(hn2,600);

 #pragma omp parallel private(thid)
 {
  thid=omp_get_thread_num();
  printf("P10 hello from thread %d %s\n",thid, hn2);
 }

 MPI_Finalize ();
}
```

Mpi-openmp Thread Affinity Mapping

- #mapping MPI openMP scatter

- mpirun -env I_MPI_PIN_DOMAIN auto:compact -env OMP_NUM_THREADS 2 -env KMP_AFFINITY granularity=fine,scatter,verbose -env I_MPI_DEBUG 4 -host mic0 -n 2 /tmp/mpiopenMPExample.mic

- #mapping MPI openMP compact

- mpirun -env I_MPI_PIN_DOMAIN auto:compact -env OMP_NUM_THREADS 2 -env KMP_AFFINITY granularity=fine,compact,verbose -env I_MPI_DEBUG 4 -host mic0 -n 2 /tmp/mpiopenMPExample.mic

MPI+Offload Support

- How to control MIC mapping of threads?
  - How do I avoid that offload of first MPI process interferes with offload of second MPI process, i.e. by using identical MIC cores/threads?
  - Default: No special support (now). Offloads from MPI processes handled by system like offloads from independent processes (or users).

- Define thread affinity manually per single MPI process (pseudo syntax!):
  # export OMP_NUM_THREADS=4
  - # mpiexec –env KMP_AFFINITY=[1-4] –n 1 –host myMIC ... :
    –env KMP_AFFINITY=[5-8] –n 1 –host myMIC ... :
    ...