

Future Programming Languages

Stefano Markidis, HPCViz Department, KTH
(markidis@pdc.kth.se)



Prediction is very difficult,
especially about the future

N. Bohr

Motivation for this Lecture

- Provide reasons for improving existing programming models, and for developing new programming models.
- Give an overview of the new features in MPI and OpenMP, and of what is likely to be next in MPI and OpenMP.
- Introduce new programming languages/libraries for distributed memory systems, called PGAS, that in some sense try combine the best features of OpenMP and MPI.
- Introduce new programming approaches to make programming GPUs easier. I will briefly describe OpenACC, and using MPI to access GPU memory.

Software Losing Ground to Hardware Parallelism

High performance computing codes, in general, are **remarkable underachievers**.

A very minority (1%) of the applications **scales** on current Petascale machines. Very few codes use **accelerators** because to program them in real-world application is often difficult.

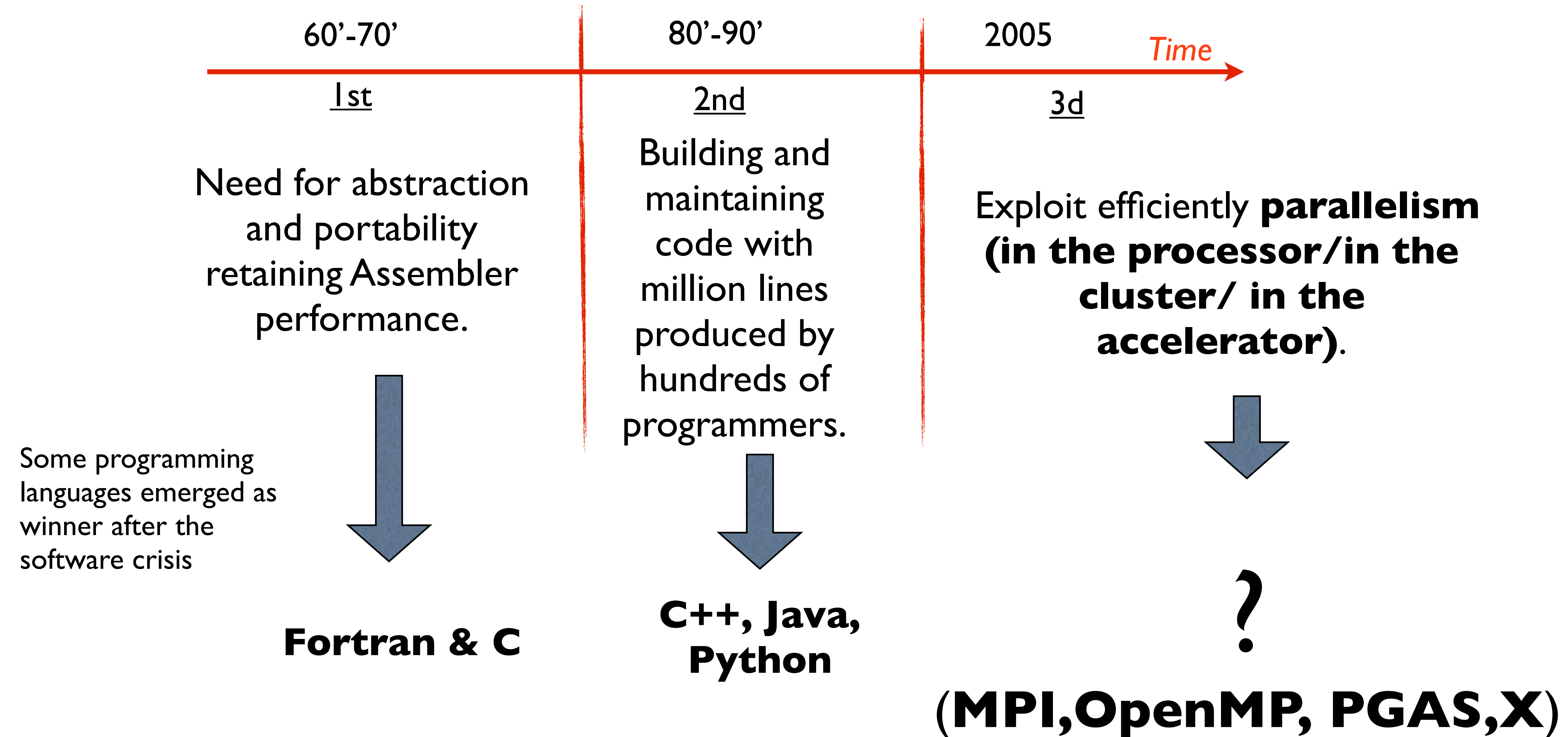


Example: Top500 #1 Tianhe-2

- 3,120,000 cores.
- Theoretical peak performance: 54.9 PetaFlops.
- Maximal performance achieved with highly optimized LINPACK: 33 PetaFlops
- With highly optimized code (LINPACK), the efficiency is 60%. Any idea of the actual performance of the average code running on HPC?

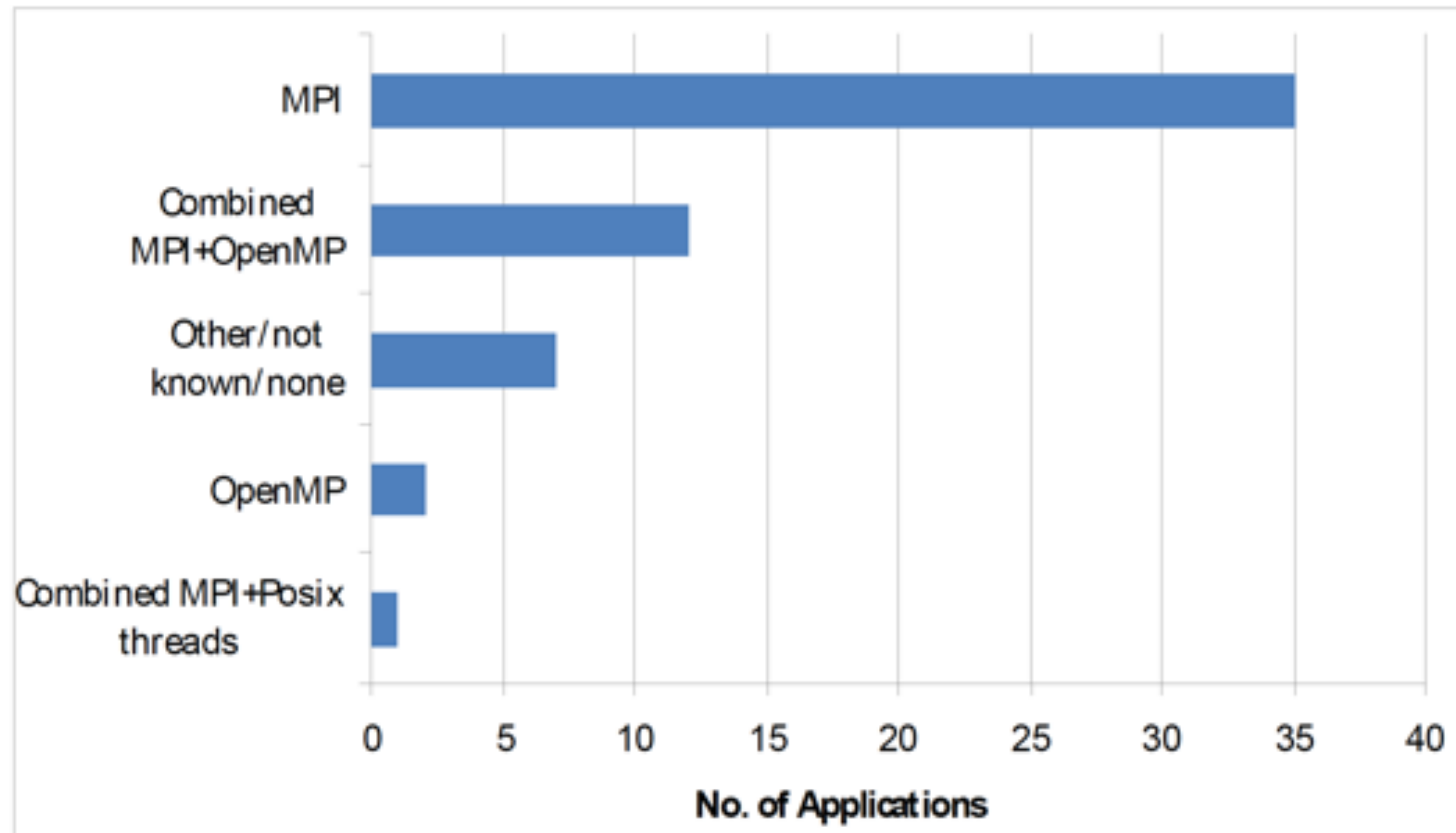


We are living the 3rd software crisis



Why did we spend so much time on MPI ?

- Almost all the codes running on supercomputer use MPI. OpenMP is used in combination with MPI. MPI and OpenMP.



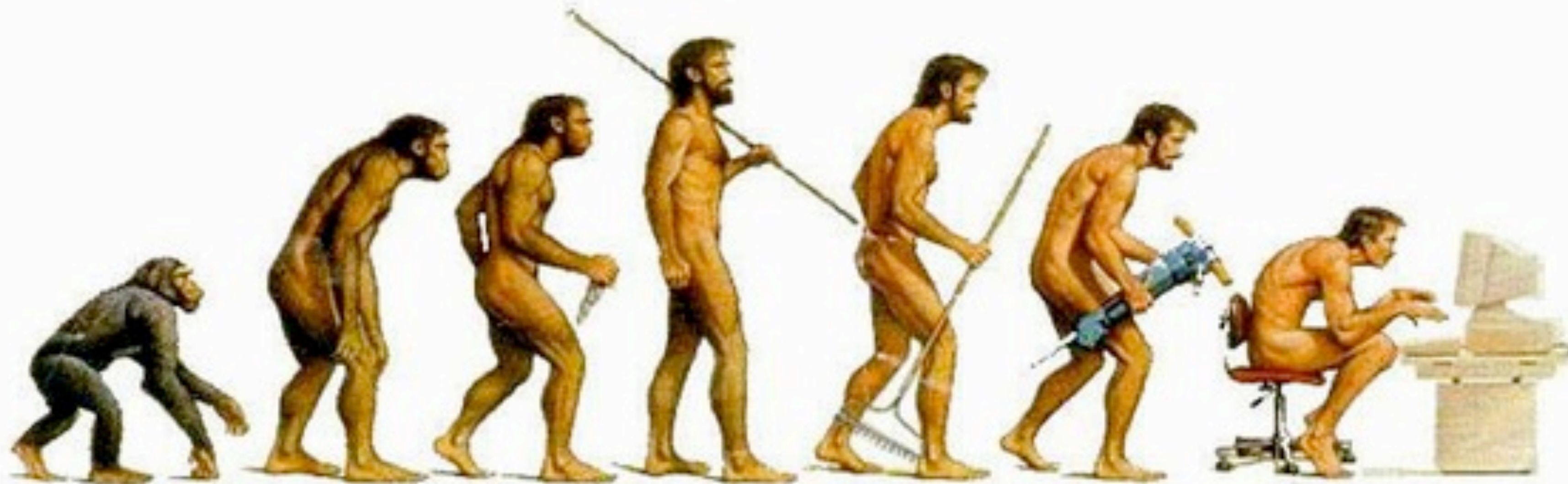
From a survey of 57 applications running on PRACE machines

MPI and OpenMP are changing

- MPI and OpenMP are in continuous evolution to keep the pace with the new challenges.

MPI-1 (1994), MPI-2 (1998), **MPI-3 (2012)**

OpenMP-1 (1997), OpenMP-2 (2000), OpenMP-3 (2008), **OpenMP 4 (July 2013)**



ENTIRE WEEK

SATURDAY

SUNDAY

MONDAY

TUESDAY

WEDNESDAY

THURSDAY

FRIDAY

Advanced MPI

SESSION: Advanced MPI

EVENT TYPE: Tutorial

TIME: 8:30AM - 5:00PM

PRESENTER(S): William Gropp, Ewing Lusk, Robert Ross, Rajeev Thakur

ROOM:

ABSTRACT:

MPI continues to be the dominant programming model for parallel scientific applications on all large-scale parallel machines, such as IBM Blue Gene and Cray XE/XK, as well as on clusters of all sizes. An important trend is the increasing number of cores per node of a parallel system, resulting in increasing interest in combining MPI with a threaded model within a node. The MPI standard is also evolving to meet the needs of future systems, and MPI 3.0 is expected to be released later this year. This tutorial will cover several advanced features of MPI that can help users program such machines and architectures effectively. Topics to be covered include parallel I/O, multithreaded communication, one-sided communication, dynamic processes, and new features being added in MPI-3 for hybrid programming, one-sided communication, collective communication, fault tolerance, and tools. In all cases, we will introduce concepts by using code examples based on scenarios found in real applications. Attendees will leave the tutorial with an understanding of how to use these advanced features of MPI and guidelines on how they might perform on different platforms and architectures.

CHAIR/PRESENTER DETAILS:

William Gropp - University of Illinois at Urbana-Champaign

Ewing Lusk - Argonne National Laboratory

Robert Ross - Argonne National Laboratory

Rajeev Thakur - Argonne National Laboratory

MPI 3.0 released November 2012

New Developments in MPI-3



- MPI 3.0 standard posted on:

<http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>

- **Non-blocking collective operations.** Increase of performance by overlapping communication and computation in collective operations.
- **Neighborhood (aka sparse) collective operations** are extending the process topologies with additional communication power.
- **Improved support for one-sided communication**, that was already present from MPI-2 but not adopted by users (mostly for the difficulty).
- **MPI tool interface** for the development of MPI performance monitoring tools and debuggers/correctness checker.

Non-blocking Collective Example

```
MPI_Comm comm;  
int array1[100], array2[100];  
int root=0;  
MPI_Request req;  
...  
MPI_Ibcast(array1, 100, MPI_INT, root, comm, &req);  
compute(array2, 100);  
MPI_Wait(&req, MPI_STATUS_IGNORE);
```

Start a broadcast of 100
ints from process 0 to
every process in the
group, perform some
computation on
independent data, and
then complete the
outstanding broadcast
operation.

Asynchronous Barrier

- Crazy idea? The whole point of a barrier is to synchronize — how does it make sense not to block while waiting ?
- Better question: why would you block while waiting? Think of non-blocking barriers as a notification mechanism that every process has reached (or passed) a common milestone.



A barrier is like the start of a meeting. You can't start the meeting until all the participants of the meeting arrive. You can do some other work while waiting for everybody to arrive.

Why do we care about collectives?

Systems are getting very large. Top systems have tens of thousands of nodes and order 1 million cores:

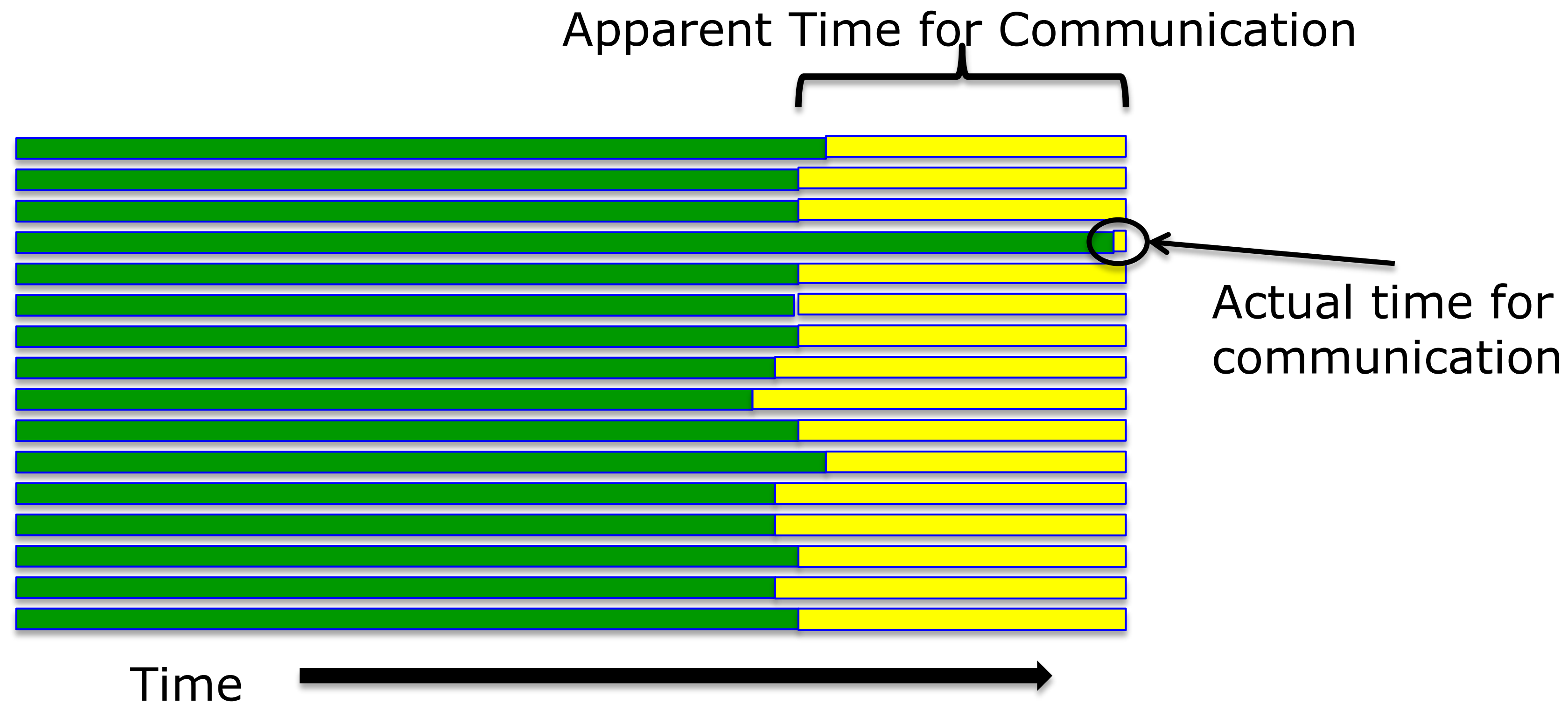
- Tianhe-2 (China) 16,000 nodes
- Sequoia (LLNL) 98,304 nodes, >1M cores

Just getting all of these nodes to agree takes time

$O(10\text{usecs})$ or about 20,000 cycles of time.

Why do we care about collectives ? 2

- What if one core (out of a million) is delayed ?

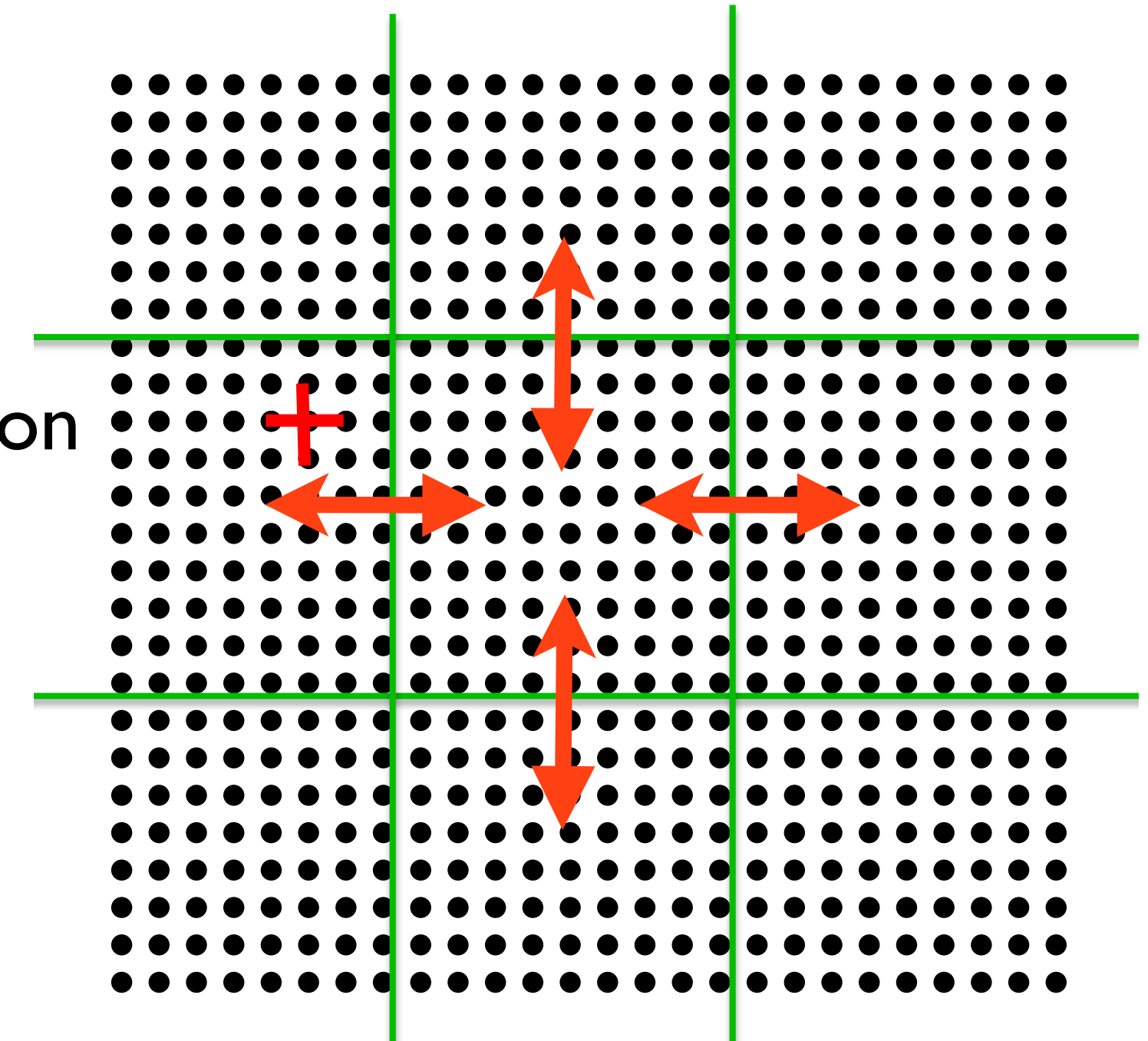


- Everyone has to wait at the synchronization point

Neighbor Collective Operations

Collective operations that are defined on process neighborhoods. Process neighborhoods are attached to the communicator on which the collectives are called.

- Many domain decompositions results in nearest neighbor communication patterns (sparse communication pattern).
- Most supercomputers (Blue-gene, Cray XT) support only sparse communication efficiently.
- 2 different neighbor collectives:
 - Gather (both blocking and non blocking)
(MPI_NEIGHBOR_ALLGATHER, MPI_NEIGHBOR_ALLGATHERV)
 - All-to-all (MPI_NEIGHBOR_ALLTOALL)

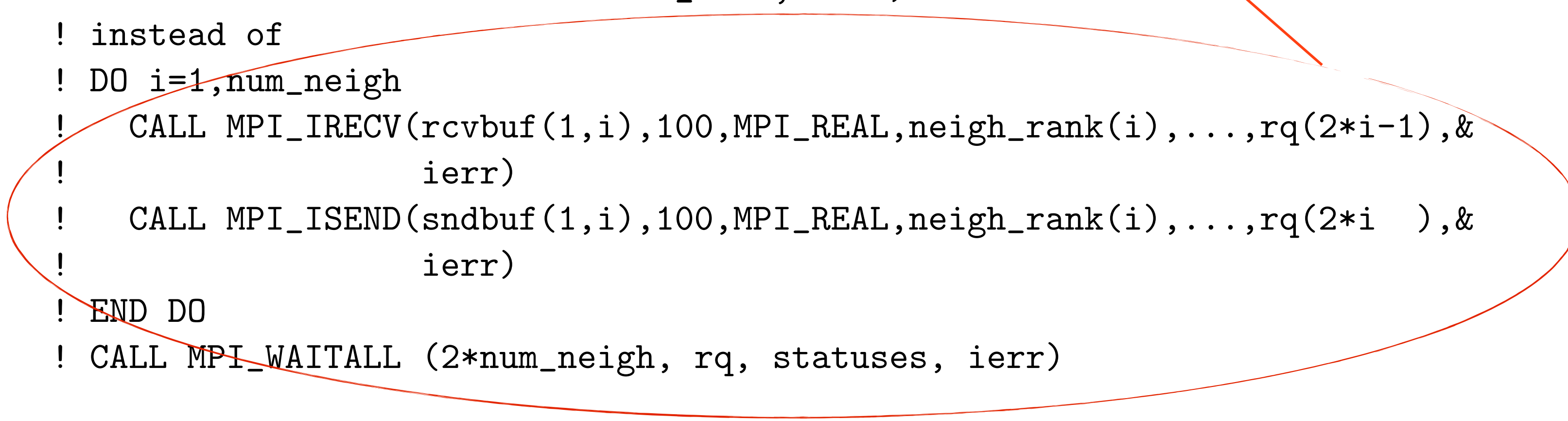


Example of Neighbor Collective

```
SUBROUTINE exchange (u, comm_cart, neigh_rank, num_neigh)
REAL u(0:101,0:101)
INTEGER comm_cart, num_neigh, neigh_rank(num_neigh)
REAL sndbuf(100,num_neigh), rcvbuf(100,num_neigh)
INTEGER ierr
sndbuf(1:100,1) = u( 1,1:100)
sndbuf(1:100,2) = u(100,1:100)
sndbuf(1:100,3) = u(1:100, 1)
sndbuf(1:100,4) = u(1:100,100)
CALL MPI_NEIGHBOR_ALLTOALL (sndbuf, 100, MPI_REAL, rcvbuf, 100, MPI_REAL, &
                           comm_cart, ierr)

! instead of
! DO i=1,num_neigh
!   CALL MPI_IRECV(rcvbuf(1,i),100,MPI_REAL,neigh_rank(i),...,rq(2*i-1),&
!               ierr)
!   CALL MPI_ISEND(sndbuf(1,i),100,MPI_REAL,neigh_rank(i),...,rq(2*i  ),&
!               ierr)
! END DO
! CALL MPI_WAITALL (2*num_neigh, rq, statuses, ierr)

u( 0,1:100) = rcvbuf(1:100,1)
u(101,1:100) = rcvbuf(1:100,2)
u(1:100, 0) = rcvbuf(1:100,3)
u(1:100,101) = rcvbuf(1:100,4)
END
```



**communication
routine
for halo
exchange**

Improvements in One-Sided Communication

- `MPI_Win_create_dynamic`
 - Window without memory attached
 - `MPI_Win_attach` to attach memory to a window
- `MPI_Win_allocate_shared`
 - Windows with shared memory
 - Allows direct loads/store accesses by remote processes
- `MPI_Rput`, `MPI_Rget`, `MPI_Raccumulate`
 - Local completion by using `MPI_Wait` on request objects
- `MPI_Get_accumulate`, `MPI_Fetch_and_op`
 - Accumulate into target memory, return old data to origin
- `MPI_Compare_and_swap`
 - Atomic compare and swap
- `MPI_Win_lock_all`
 - Faster way to lock all members of win
- `MPI_Win_flush` / `MPI_Win_flush_all`
 - Flush all RMA ops to target / window
- `MPI_Win_flush_local` / `MPI_Win_flush_local_all`
 - Locally complete RMA ops to target / window
- `MPI_Win_sync`
 - Synchronize public and private copies of window
- Overlapping accesses were “erroneous” in MPI-2
 - They are “undefined” in MPI-3

One-sided Communication Main Issue: **Memory Model**

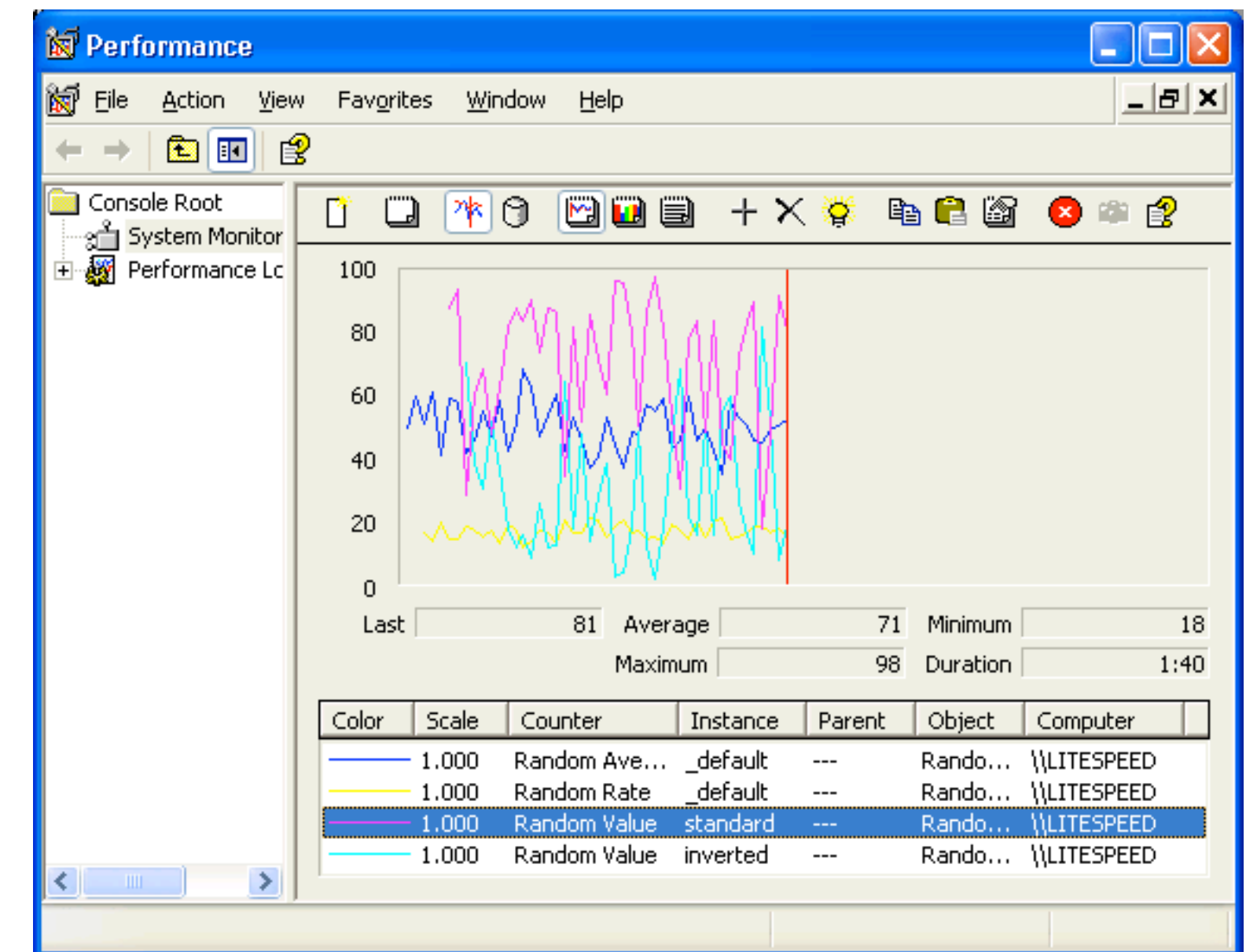
MPI-2 handled the cache coherency (consistency of data stored in local caches of a shared resource) issue **but was in many places hard to use and even harder to understand.**

MPI-3 differentiate between two memory models (essentially “cache-coherent” and “not cache-coherent” in MPI lingo “unified” and “separate” public and private window).

MPI Tools Interface

The new MPI tool interface allows the MPI implementation to expose certain internal variables, counters, and other states to the user (performance tools, debuggers).

It is very useful for tools and advanced MPI users to investigate performance issues.



MPI future- Fault Tolerance

- The trend to extend performance of supercomputers is to increase the number of computing units. As a consequence, the number of faults is expected to increase, **as one can expect that one of the millions components will always be broken.**
- The current state of the art automatic and transparent fault tolerant techniques based on **checkpoint/restart do not scale and will no longer work**, and hardware methods based on **replications are quite expensive.**
- Fault-tolerance in MPI is an active area of research and some MPI implementation already include mechanism for fault-tolerance



Future of MPI - Fault Tolerance

- **Non-collective communicator creation:** proposal to allow a group of processes to create a communicator “on their own”, i.e., without involving the full parent communicator. This would be very useful for MPI fault tolerance, where it could be used to “**fix**” a **broken communicator** (create a communicator with less processes).



Future of MPI - Allocating a Shared Memory Window

This would allow to share **data-structures across all MPI processes in a multicore node** similarly to OpenMP.

ENTIRE WEEK

SATURDAY

SUNDAY

MONDAY

TUESDAY

WEDNESDAY

THURSDAY

FRIDAY

The Future of OpenMP

SESSION: Software Development Tools II

EVENT TYPE: Exhibitor Forum

TIME: 2:00PM - 2:30PM

PRESENTER(S): Michael Wong

ROOM: 155-C

ABSTRACT:

Now celebrating its 15th birthday, OpenMP has proven to be a simple, yet powerful model for developing multi-threaded applications. OpenMP continues to evolve, adapt to new requirements, and push at the frontiers of parallelization. It is developed by the OpenMP Architecture Review Board, a group of 23 vendors and research organizations. A comment draft of the next specification version will be released at or close to SC12. It will include several significant enhancements, including support for accelerators, error handling, thread affinity, tasking extensions and support for Fortran 2003. We will give an overview of the new specifications, after having described the process to get to this new specification.

CHAIR/PRESENTER DETAILS:

Michael Wong - OpenMP ARB

OpenMP 4.0 released:

[http://www.openmp.org/mp-documents/
OpenMP4.0.0.pdf](http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf)

New Developments in OpenMP 4

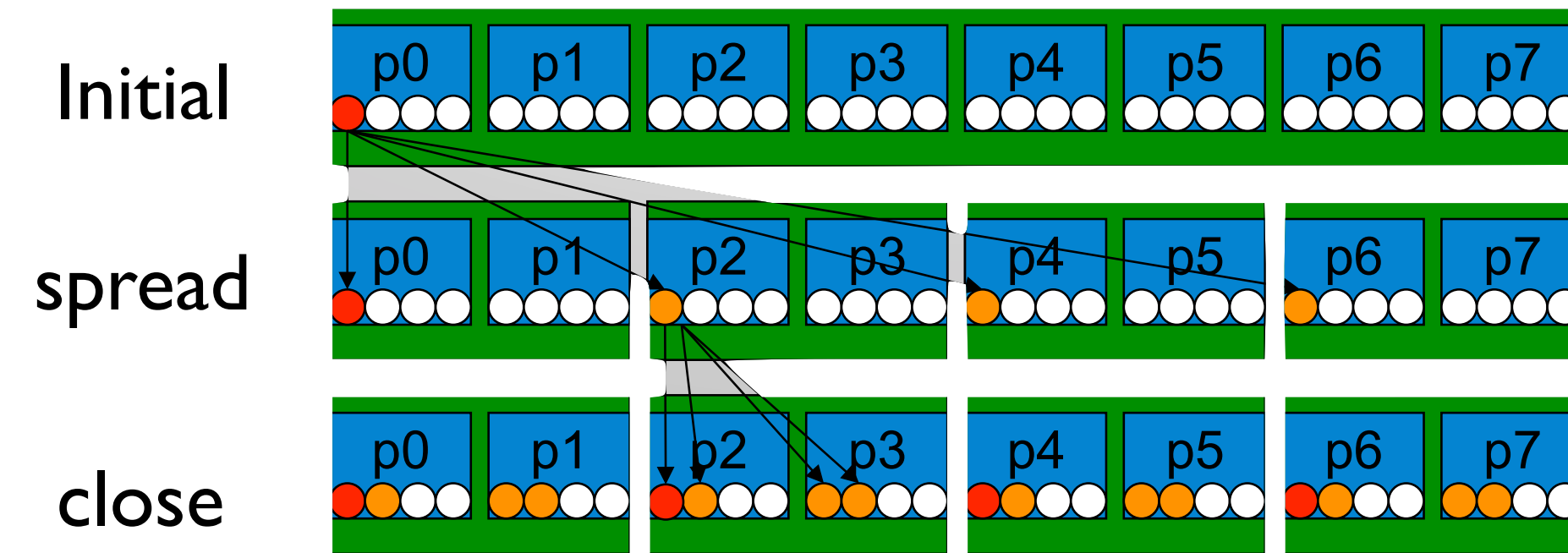
- OpenMP specifications posted on: <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>
- **Thread Affinity** (clauses and environment variables)
- **New SIMD directives.**
- **Increased support for task synchronization, and added the concept of task dependence**

Thread Affinity

Thread affinity enables **the binding and un-binding of a thread to a physical core or a range of cores**, so that the thread will execute only on the designated core or cores rather than being able to execute on any core. This can be viewed as a modification of the native central queue scheduling algorithm. Each item in the queue has a tag indicating its preferred core.

The advantages of the use of thread affinity for the user are **better locality, less false sharing and more memory bandwidth**.

Increase of OpenMP performance



Increased support for Thread Affinity

Added a new clause to the parallel construct:

proc_bind(master | close | spread)

- **New policies** determine relative bindings:
 - Assign threads to same place as *master*
 - Assign threads *close* in place list to parent thread
 - Assign threads to maximize *spread* across places

High-level Affinity Support in OpenMP 4

Request binding of threads to places (already in OpenMP 3.1):

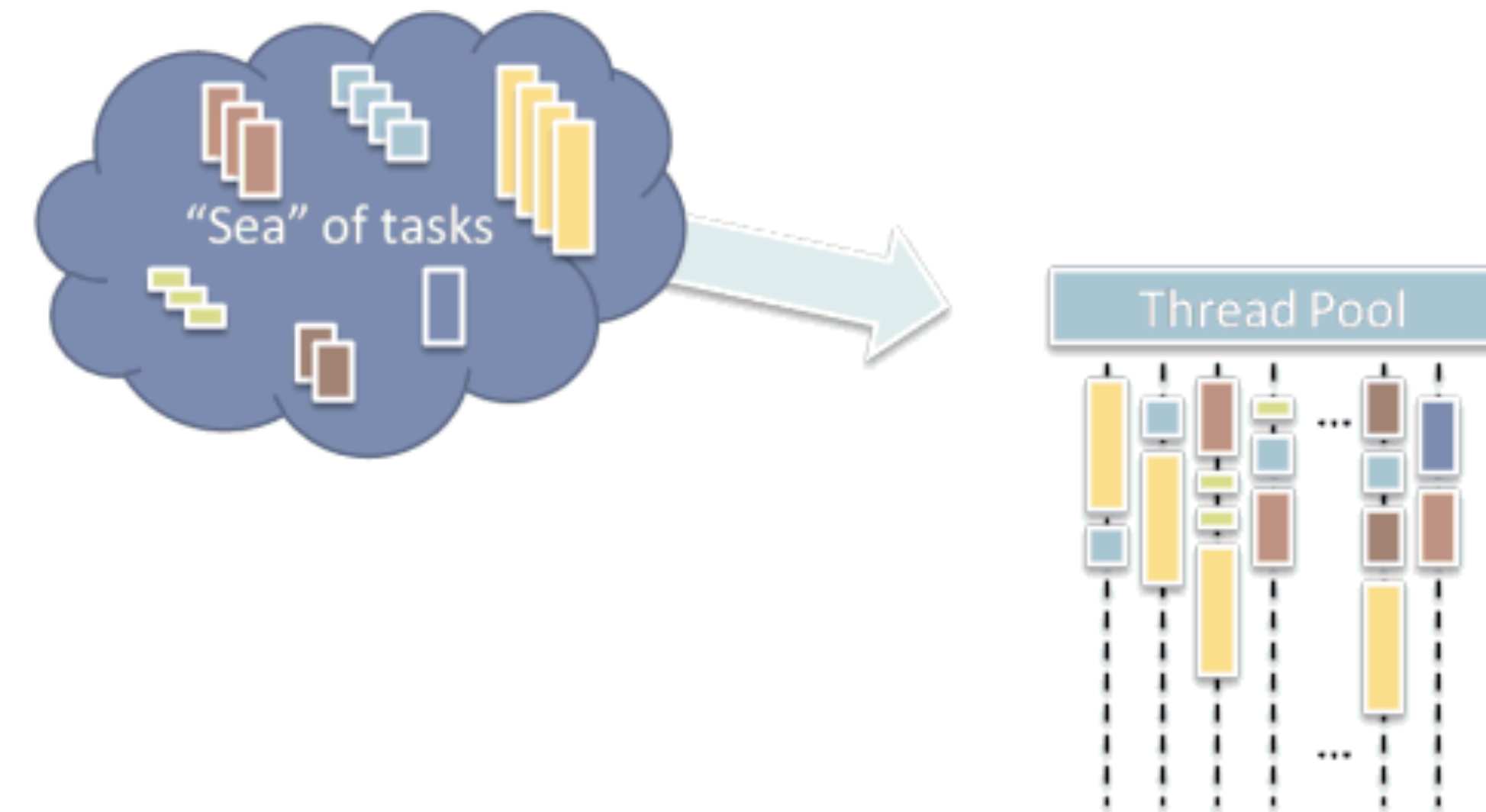
- New extensions specify thread locations
 - Increased choices for OMP_PROC_BIND
 - Can still specify true or false
 - Can now provide a list (possible item values: master, close or spread) to specify how to bind implicit tasks of parallel regions.
 - Added OMP_PLACES environment variable
 - Can specify abstract names including threads, cores and sockets
 - Can specify an explicit ordered list of places

Task Based Approaches (beyond threads)

Threads created with threading packages are **logical threads** that are mapped by the operating system onto the physical threads of the hardware.

Creating too few logical threads will undersubscribe the system, wasting some of the available hardware resources. Creating too many logical threads will oversubscribe the system, causing the operating system to incur considerable overhead as it must time-slice access to the hardware resources.

One common way to perform this difficult balancing act is to create a pool of threads that are used across the lifetime of an application. Typically, one logical thread is created per physical thread. **The application then dynamically schedules computations (tasks) on to threads in the thread pool.**



Tasks in OpenMP

OpenMP compilers have implemented a task queue construct partly inspired by the Cilk programming language and its implementation. Two new pragmas were added to OpenMP 3.0 (2009):

- **task** to specify the single-threaded tasks, or pieces of work, to be executed by a thread. (This is somewhat analogous to the section pragma, we saw in the OpenMP lectures last week).
- **task queue** to create an empty queue of tasks

Tasks in OpenMP 4

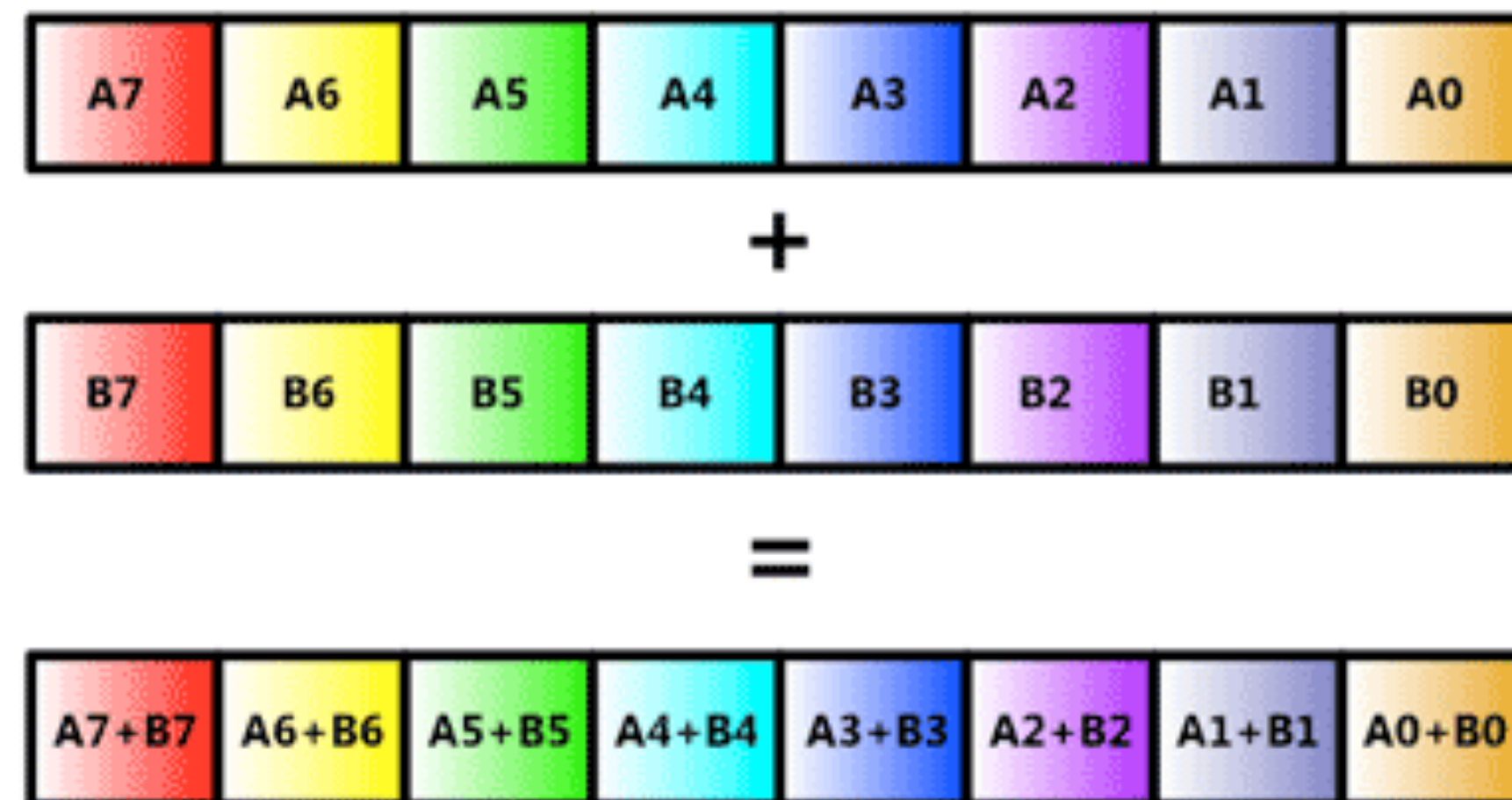
New features in OpenMP 4:

- Support simpler task synchronization
- Will add concept of task **dependence**

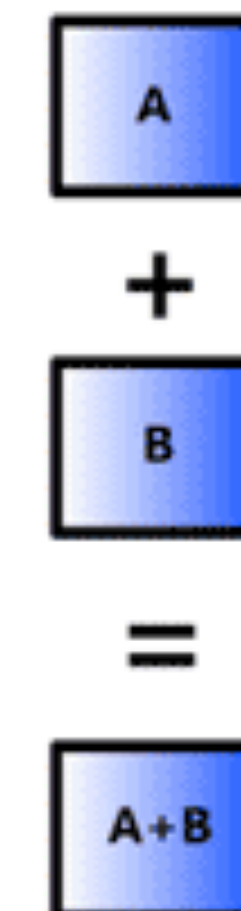
Single Instruction Multiple Data

- SIMD are a class of parallel computers with multiple processing elements that perform the same operation on multiple data points simultaneously.
- Vector instructions were used in vector supercomputers (most famous Cray') in 70s-80s (64,000 instructions at the same time) .
- Now days, **all the new CPU designs include SIMD instructions** to improve performance for graphics.

SIMD Mode



Scalar Mode



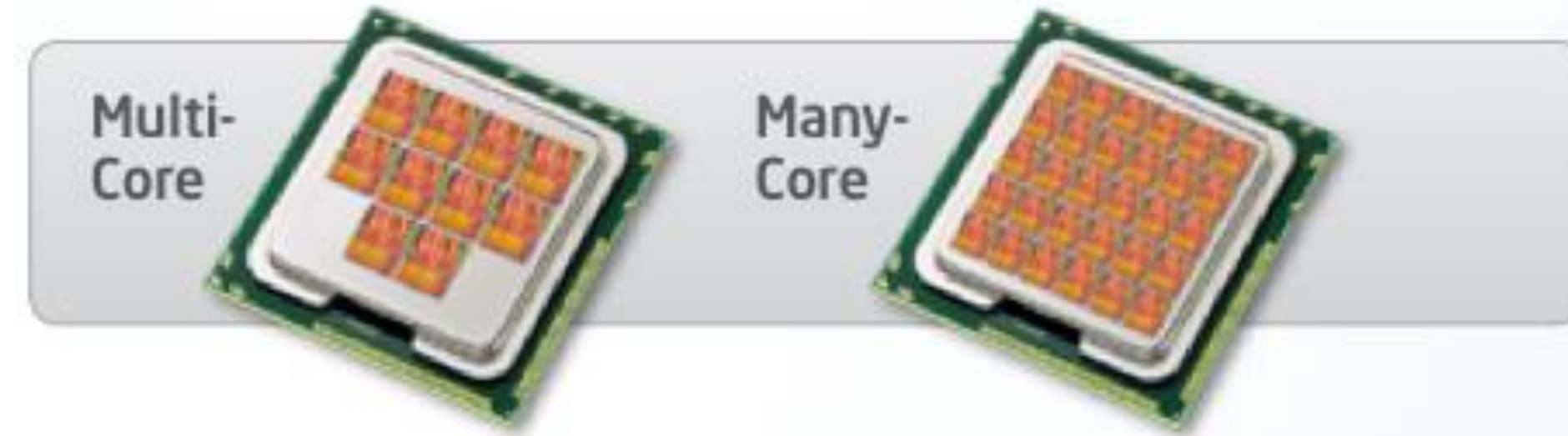
Introduction of SIMD directive in OpenMP

- In order to use SIMD instructions, the **simd directive** is introduced in OpenMP to indicate a loop should be SIMDized.
- Execute iterations of loops in SIMD chunks:
 - SIMD chunk is set of iterations executed concurrently by a SIMD lanes

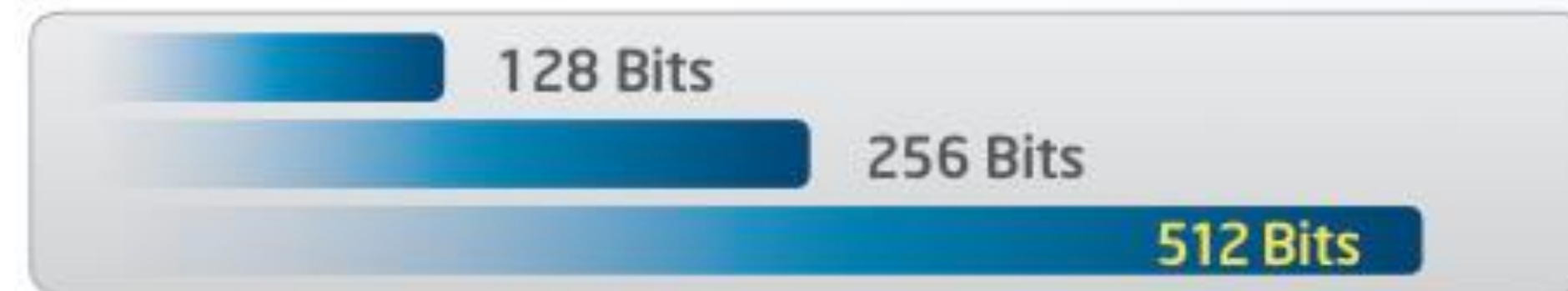
SIMD and Intel Many Integrated Core (MIC)

- Intel processors have extensions that support SIMD. These instructions operate on a vector of data in parallel. The vector width, and therefore the number of elements that can be accessed in parallel. The Intel MIC can execute **16 single-precision or 8 double-precision (DP) operations per cycle.**
- Intel stand-alone/accelerator with more than 50 cores. Prototype products, named Knights Ferry (2010), Knights Corner commercial release is in production with the name of Xeon Phi since 2012.
- Xeon Phi provides up to 61 cores, and 1.2 TeraFlops. It used in Thiane-2 and Stampede.

More Cores



Wider Vectors



Future of OpenMP

- Support for memory affinity
- Support for accelerator programming
- Incorporating tool support

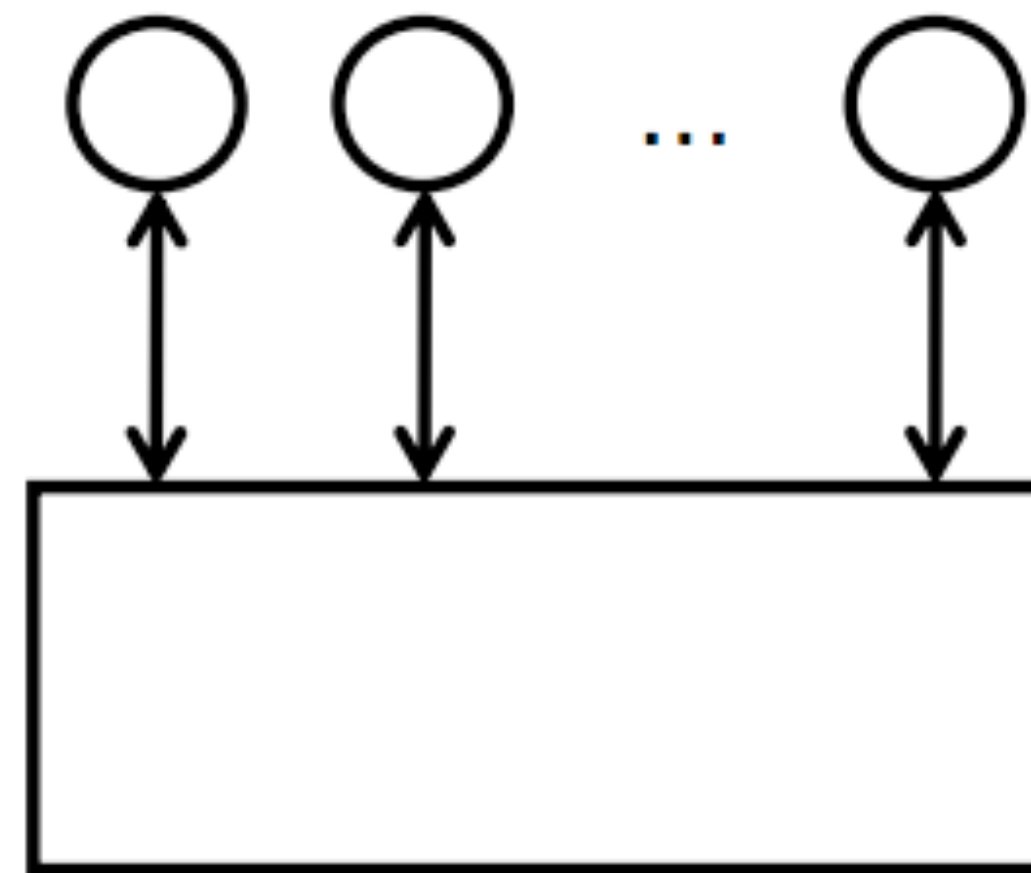
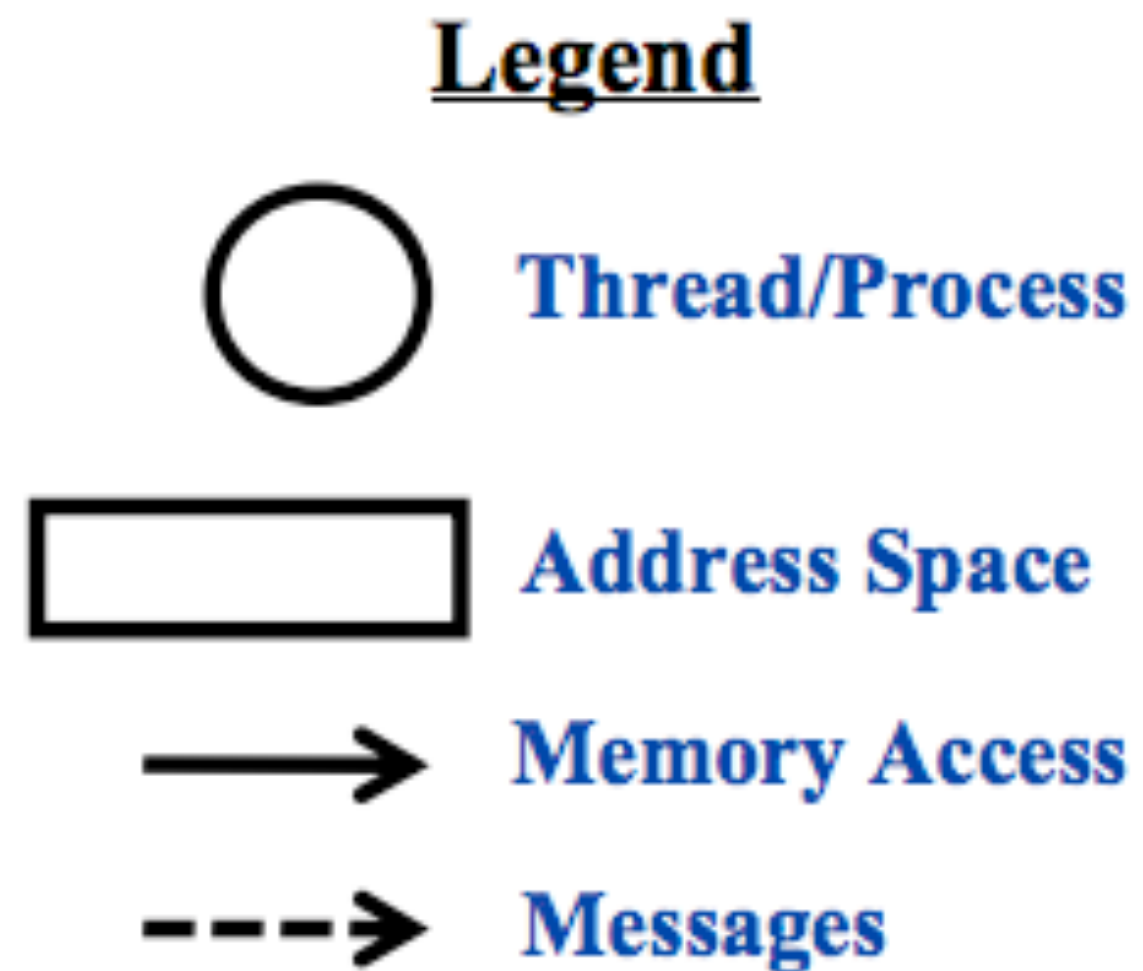
Can Parallel Computing be more Productive?

- PGAS (Partitioned Global Address Space) languages are born from **High Productivity Computing Systems initiative** (a DARPA program, 2002-2010). They are an approach for programming distributed memory systems.
- MPI functions are too low level: it's tricky to match send/receive avoiding deadlock; non-blocking communication can be tricky also.
- PGAS languages address this problem of MPI complexity. Communication is not explicit like in MPI but it just an access to a remote memory.
- PGAS languages include UPC (C extension), CoArray Fortran (Fortran extension).

Before this lecture we saw ...

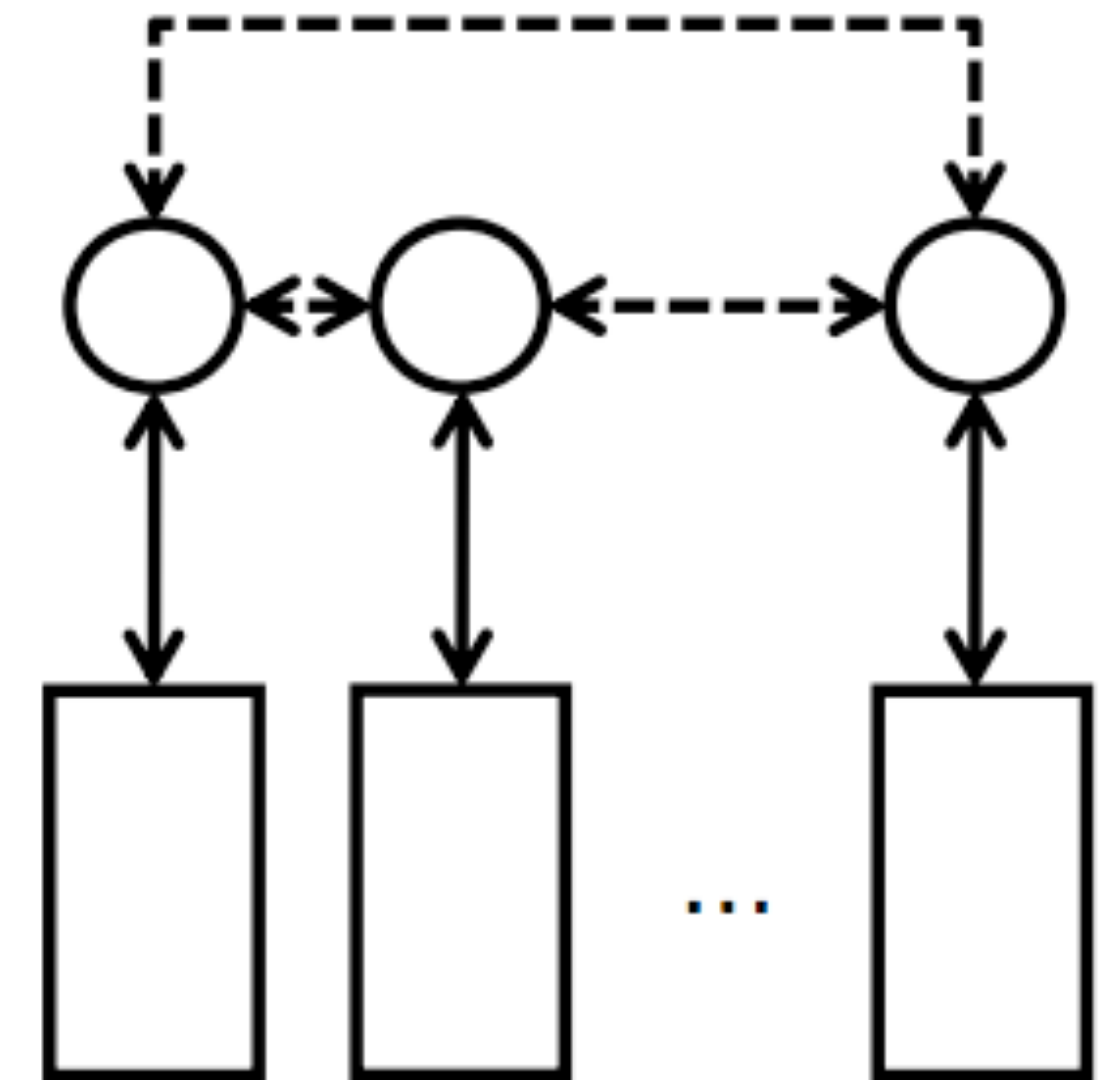
Shared Memory Model

(last week)

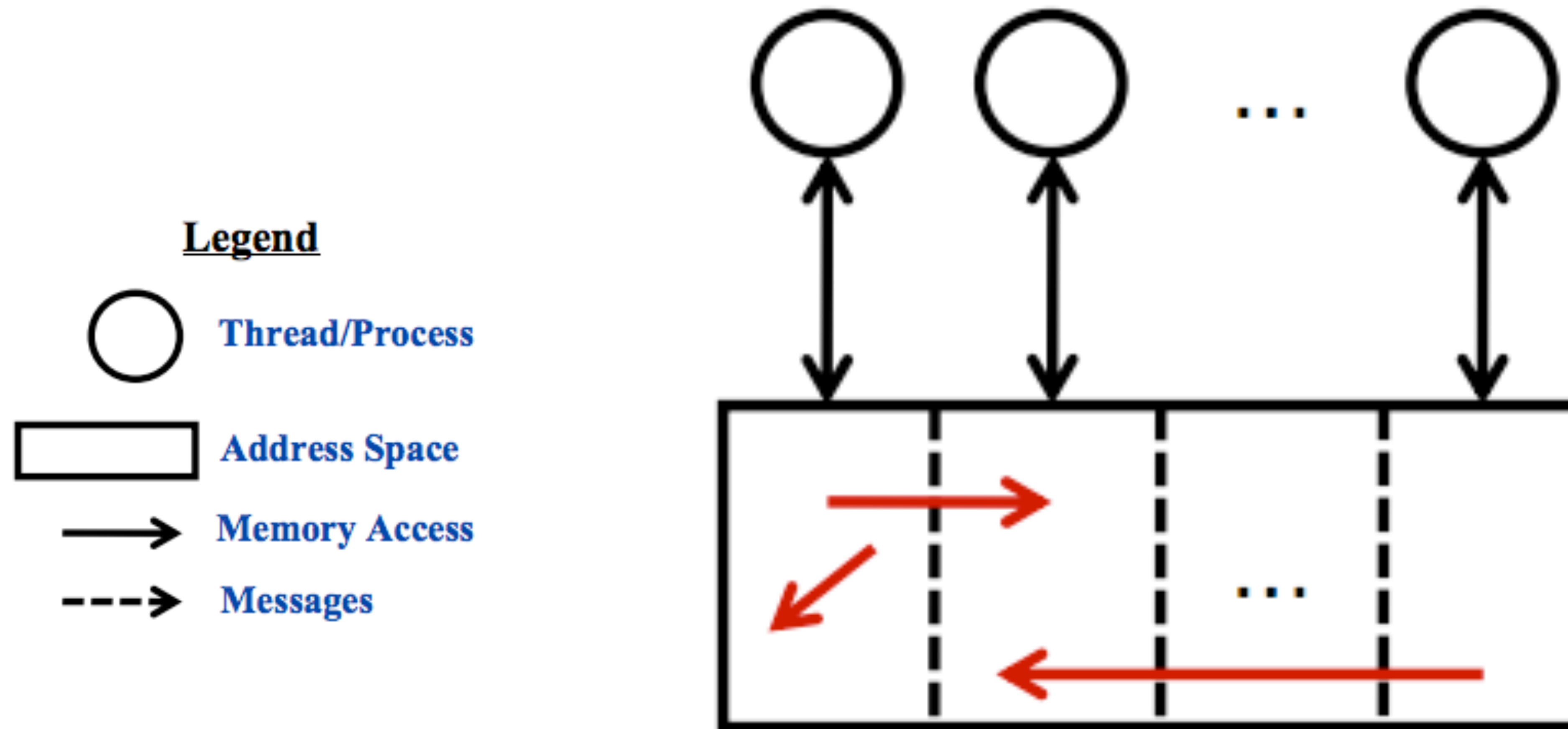


Message Passing Model

(this week)



PGAS model



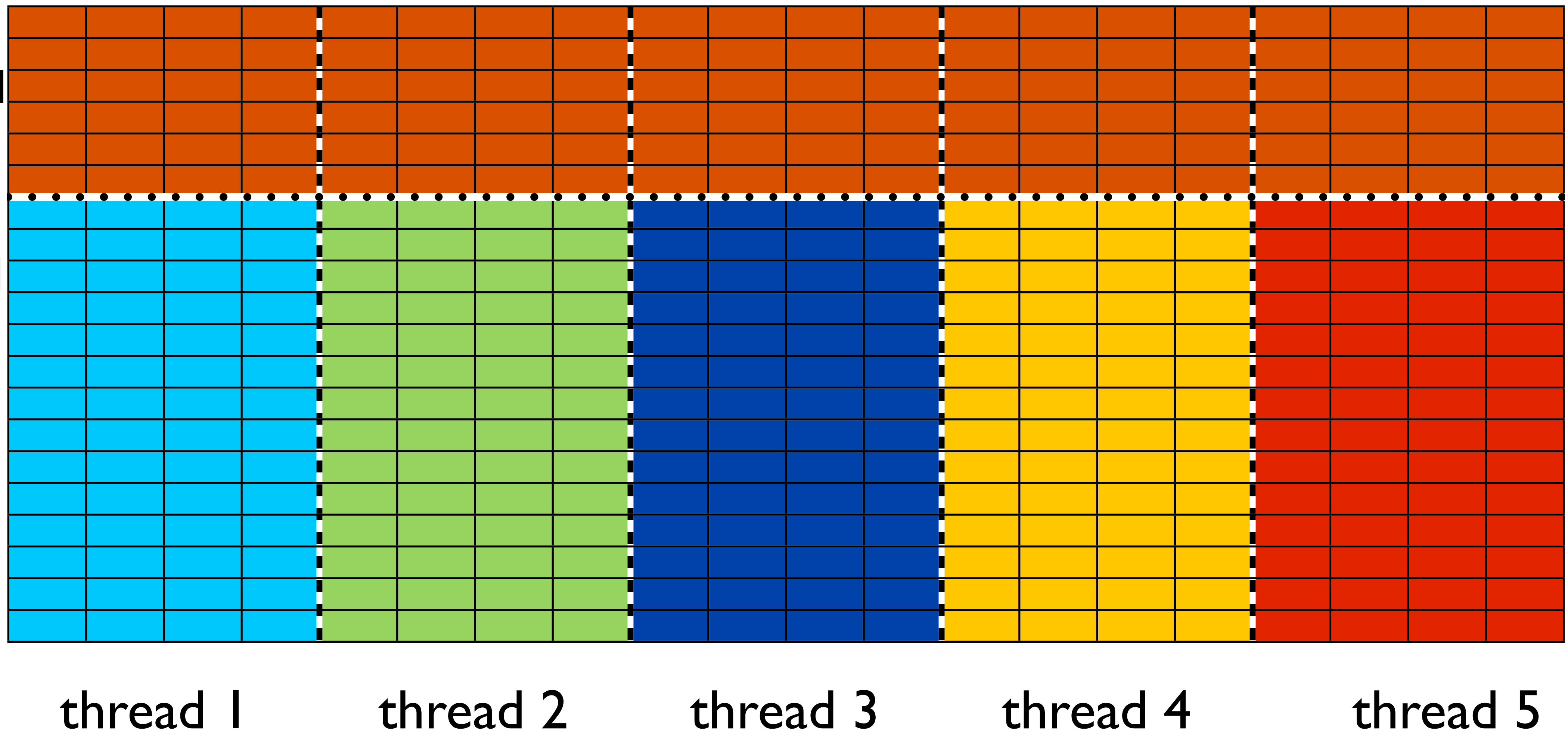
Partitioned Global Address Space

Global

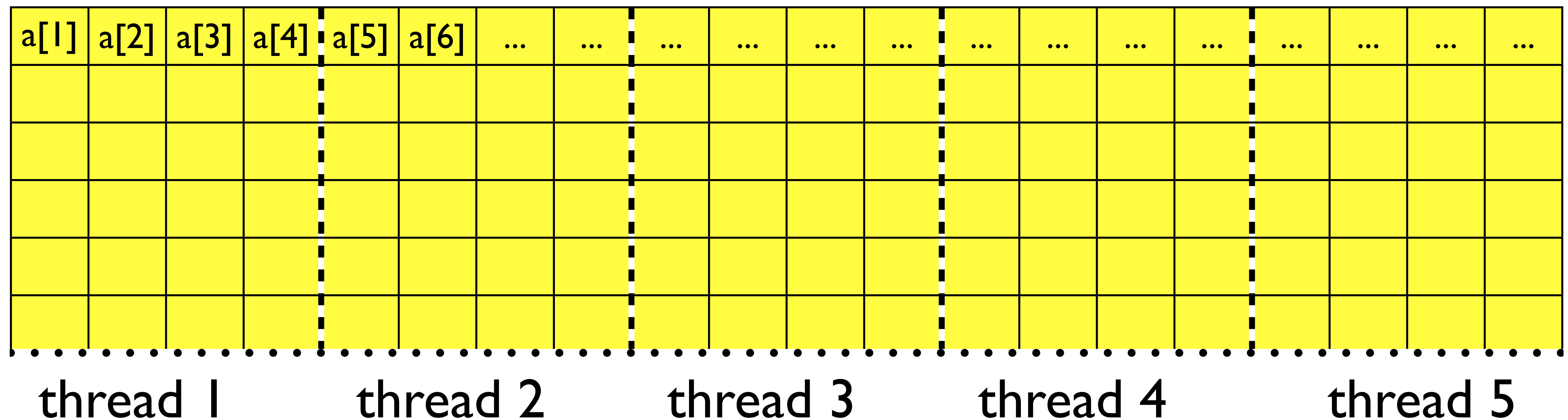
located on one thread but accessible by all the other threads

Local located on one thread but **not** accessible by all the other threads. Very fast.

MEMORY LAYOUT



Implicit communication is an access to global memory that is located on a different thread



i.e. communication (handled by the language) occurs when we want to calculate:

$$a[1] = a[4] + a[5];$$

the value $a[5]$ is transferred to thread 1 by implicit communication and summed to $a[4]$ and set in thread 1 as $a[1]$

UPC (Unified Parallel C)



UPC is a PGAS model and programs operate in **Single Program, Multiple Data** (like MPI) fashion: multiple processes execute the same program, but the execution paths can differ. UPC uses the term *thread*.

To allow threads to access both local and remote memory, UPC provides at the program level the concept of two memory spaces: private and shared. Objects declared in private memory space use regular C declarations, e.g.

```
int x; // private variable
```

and **are only accessible by a single thread**. Objects declared in shared memory space using the “**shared**” identifier, e.g.

```
shared int y; // shared variable
```

are accessible by all threads.

Hello world in UPC

- Any legal C program is also a legal UPC program.
- If you compile and run it as UPC with P threads, it will run P copies of the program.
- Number of threads specified at compile-time or run-time; available as program variable THREADS
- MYTHREAD specifies thread index (0..THREADS-1)

```
#include <upc.h> /* needed for UPC extensions */
#include <stdio.h>

main() {
    printf("Thread %d of %d: hello UPC world\n",
        MYTHREAD, THREADS);
}
```


Calculate PI in UPC

```
main(int argc, char **argv) {  
    int i, hits, trials = 0;  
    double pi;
```

Each thread gets its own
copy of these variables

```
    if (argc != 2) trials = 1000000;  
    else trials = atoi(argv[1]);
```

Each thread can use
input arguments

```
    srand(MYTHREAD*17);
```

Initialize random in
math library

```
    for (i=0; i < trials; i++) hits += hit();  
    pi = 4.0*hits/trials;  
    printf("PI estimated to %f.", pi);
```

Each thread calls "hit" separately

- Required includes:

```
#include <stdio.h>  
#include <math.h>  
#include <upc.h>
```

- Function to throw dart and calculate where it hits:

```
int hit(){  
    int const rand_max = 0xFFFFFFFF;  
    double x = ((double) rand()) / RAND_MAX;  
    double y = ((double) rand()) / RAND_MAX;  
    if ((x*x + y*y) <= 1.0) {  
        return(1);  
    } else {  
        return(0);  
    }  
}
```

Co-Array Fortran (CAF)

Fortran Co-Arrays are an example of a PGAS model and a relatively new mechanism for performing communications in parallel Fortran applications.

Like UPC, Co-array Fortran programs follow a **SPMD model**. Each **replication is referred to as an “image”, and images are executed asynchronously**.

The execution path may differ from image to image: each image has a unique identifier that can be used in control statements. A new, **co-dimension syntax** is used in addition to the standard array dimension syntax. For example,

real :: x(10)[*]

declares a **co-array which has a dimension of size 10 on each image**.

Advantages of PGAS

- Easy and therefore more productive programming approach for distributed memory systems (MPI)
- PGAS are based on one-sided model:
 - Modern networking infrastructure (e.g., InfiniBand, Quadrics, BlueGene) provide native one-sided RDMA and one-sided atomic primitives that allow efficient one-sided communication.

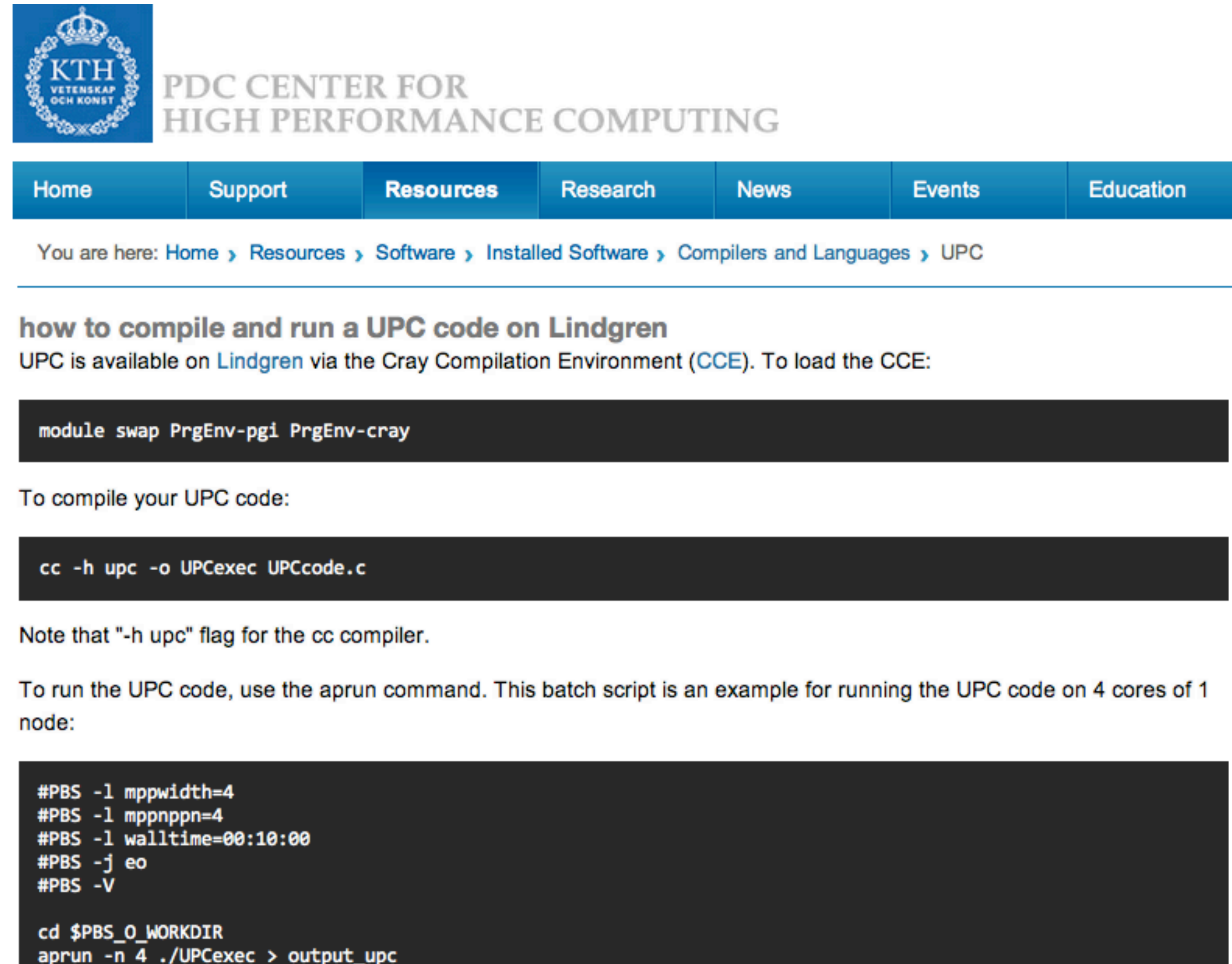


How to use UPC/CAF on Lindgren

Information on how to compile and run UPC/CAF codes on Lindgren can be found at pages:

- UPC (<http://www.pdc.kth.se/resources/software/installed-software/compilers-and-languages/upc>)
- CAF (<http://www.pdc.kth.se/resources/software/installed-software/compilers-and-languages/coarray-fortran>)

Example codes are available too.



The screenshot shows the PDC Center for High Performance Computing website. The header includes the KTH logo and the text "PDC CENTER FOR HIGH PERFORMANCE COMPUTING". A navigation bar contains links: Home, Support, Resources, Research, News, Events, and Education. Below the navigation bar, a breadcrumb trail reads: "You are here: Home > Resources > Software > Installed Software > Compilers and Languages > UPC". The main content area is titled "how to compile and run a UPC code on Lindgren" and states: "UPC is available on Lindgren via the Cray Compilation Environment (CCE). To load the CCE:". A code block shows the command: `module swap PrgEnv-pgi PrgEnv-cray`. Below this, it says "To compile your UPC code:" followed by a code block: `cc -h upc -o UPCexec UPCcode.c`. A note states: "Note that '-h upc' flag for the cc compiler." It then says: "To run the UPC code, use the aprun command. This batch script is an example for running the UPC code on 4 cores of 1 node:" followed by a code block containing a PBS batch script and the execution command: `#PBS -l mppwidth=4`
`#PBS -l mppnppn=4`
`#PBS -l walltime=00:10:00`
`#PBS -j eo`
`#PBS -V`

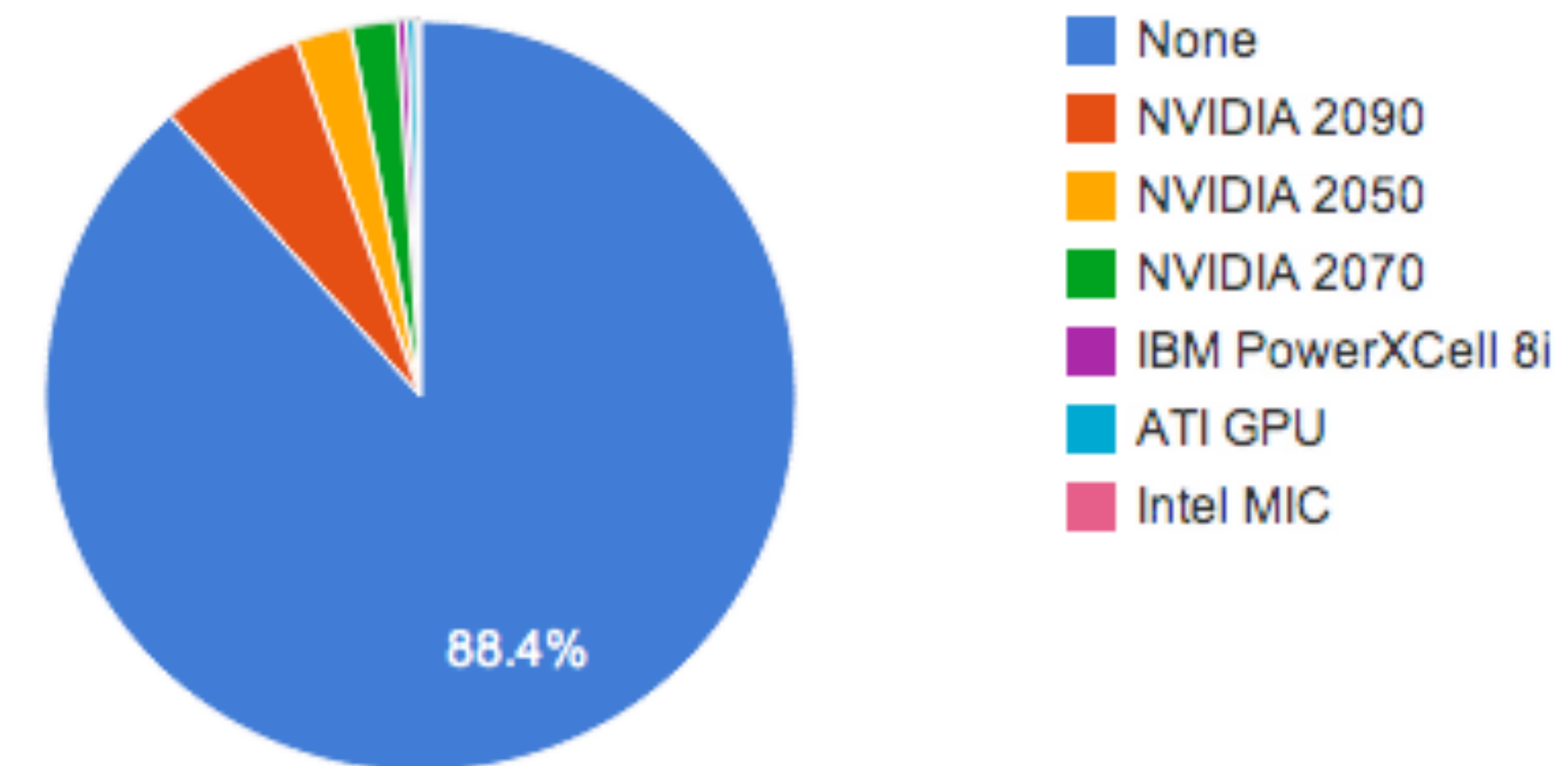
`cd $PBS_O_WORKDIR`
`aprun -n 4 ./UPCexec > output_upc`

Accelerators and Supercomputing

From top500.org (list of 500 fastest supercomputers) **June 2012**

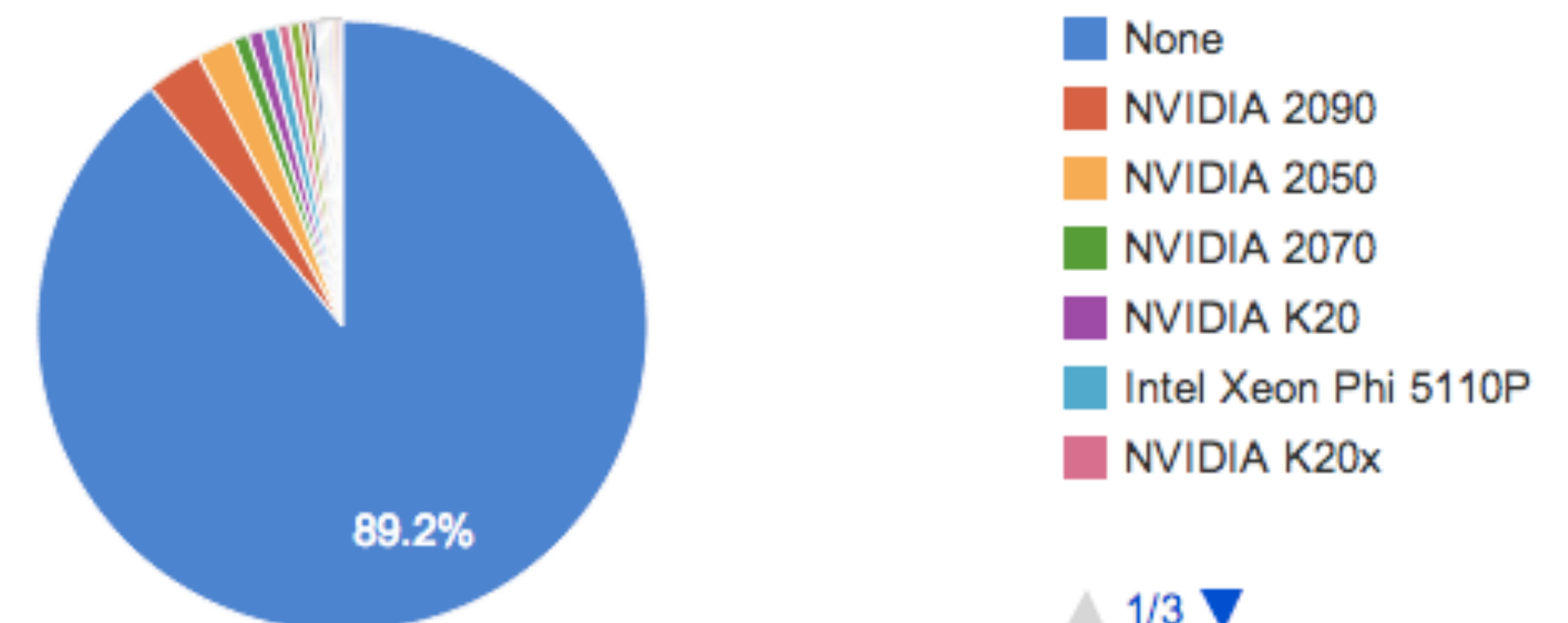
- Majority of the supercomputers (89%) doesn't have accelerators.
- NVIDIA GPUs dominate the accelerator business in HPC.
- 1% Cell accelerator in 2012 not present anymore.
- 1 machine with Intel MIC in 2012, now much more common

Accelerator/Co-Processor System Share



From top500.org (list of 500 fastest supercomputers) **June 2013**

Accelerator/Co-Processor System Share



Why only 12% of top500 has accelerators ?

- The main problem is port code (originally written for CPU) to accelerator architecture. A particular challenge is caused by **accelerators having their own memory space**, so the programmer must manage the memory transfers to and from the accelerator.
- **Can accelerator programming be made easier, without requiring to write new code or rewrite old ones?**
- Rather than rewriting, a better approach is to provide a mechanism for compilers to generate executables that can run on the GPU from the original source code.

Compiler Directives for GPU Programming

- The most promising approach is to instruct the compiler where to run part of the code (CPU/GPU) and let the compiler handle the memory transfer and code translation for GPU.
- OpenACC, the future openMPI, provide a collection of compiler directives to use GPU.
- The basic idea of compiler directives is that the **applications developer does not have to be concerned with the details of the underlying hardware of accelerators.**



Standard for Accelerator Compiler Directives

There are currently two standardization efforts ongoing:

- The first is via the established OpenMP Architecture Review Board (ARB) standards committee. A subcommittee was established to **develop an extension to the existing OpenMP 3.0 standard** that would target a **wide class of possible accelerators**. This would include **GPUs**, but also address other accelerators e.g. digital signal processors (DSPs). OpenMP will include support for accelerators.
- However, there was a need for a minimal, **interim standard to serve early adopters of the directive programming model**. To this end, the OpenACC standard launched in **November 2011**, with support from **NVIDIA** and compiler developers **Cray, PGI and CAPS**.

OpenACC

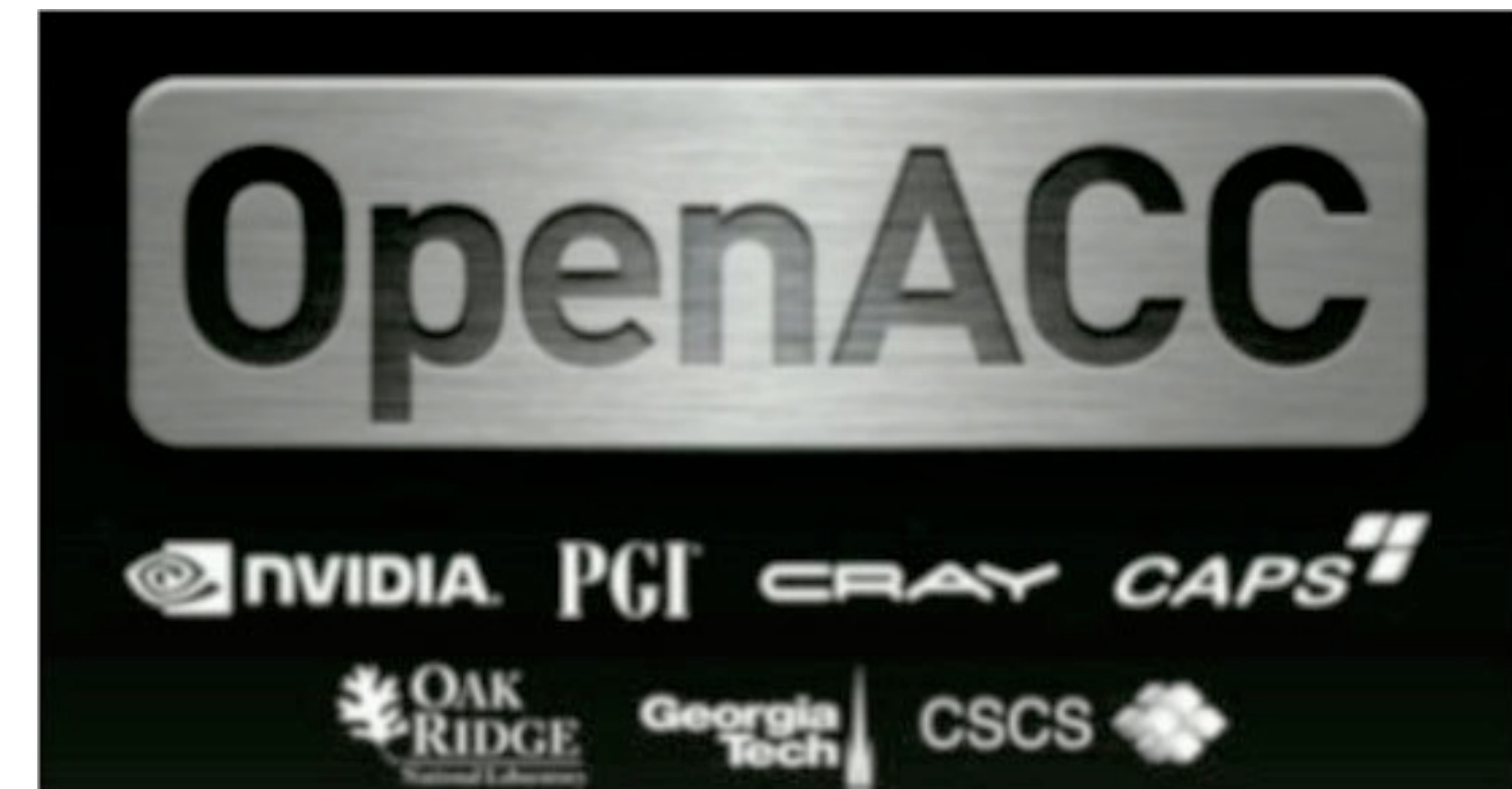
- A set of compiler directives (#pragma)
- Offload specific loops or parallelizable sections in code onto accelerators

```
#pragma acc parallel {  
    for(i = 0; i < size; i++) {  
        A[i] = B[i] + C[i];  
    }  
}
```

- Routines to allocate/free memory on accelerators

```
buffer = acc_malloc(MYBUFSIZE);  
acc_free(buffer);
```

- Supported for **C, C++ and Fortran**
- Huge list of modifiers – **copy, copyout, private, independent, etc..**



Looks like OpenMP!

(http://www.openacc.org/sites/default/files/OpenACC_API_QuickRefGuide.pdf)

General Syntax

C

```
#pragma acc directive [clause [,] clause...] new-line
```

FORTRAN

```
!$acc directive [clause [,] clause...]
```

An OpenACC construct is an OpenACC directive and the immediately following statement, loop or structured block.

Parallel Construct

An accelerator **parallel** construct launches a number of gangs executing in parallel, where each gang may support multiple workers, each with vector or SIMD operations.

C

```
#pragma acc parallel [clause [,] clause...] new-line  
{ structured block }
```

FORTRAN

```
!$acc parallel [clause [,] clause...]  
    structured block  
!$acc end parallel
```

Any data clause is allowed.

Data Clauses Memory Transfer

The description applies to the clauses used on parallel constructs, kernels constructs, data constructs, declare constructs, and update directives.

copy(list)

Allocates the data in *list* on the accelerator and copies the data from the host to the accelerator when entering the region, and copies the data from the accelerator to the host when exiting the region.

copyin(list)

Allocates the data in *list* on the accelerator and copies the data from the host to the accelerator when entering the region.

copyout(list)

Allocates the data in *list* on the accelerator and copies the data from the accelerator to the host when exiting the region.

Kernels Construct

An accelerator **kernels** construct surrounds loops to be executed on the accelerator, typically as a sequence of kernel operations.

C

```
#pragma acc kernels [clause [,] clause...] new-line  
{ structured block }
```

FORTRAN

```
!$acc kernels [clause [,] clause...]  
    structured block  
!$acc end kernels
```

Any data clause is allowed.

Loop Construct

A **loop** construct applies to the immediately following loop or nested loops, and describes the type of accelerator parallelism to use to execute the iterations of the loop.

C

```
#pragma acc loop [clause [,] clause...] new-line
```

FORTRAN

```
!$acc loop [clause [,] clause...]
```

CLAUSES

collapse(n)

Applies the associated directive to the following *n* tightly nested loops.

seq

Executes this loop sequentially on the accelerator.

private(list)

A copy of each variable in *list* is created for each iteration of the loop.

reduction(operator:list)

See **reduction** clause for **parallel** construct.

Example of OpenACC matrix-matrix multiplication $a \times b = c$

```
double precision a(n1, n2, m), b(n2, n3, m), c(n1, n3, m)
double precision tmp
!$acc data copyin(a,b) copyout(c)
!$acc kernels loop independent
do imat = 1, m
!$acc loop independent
do j = 1, n3
!$acc loop independent
do i = 1, n1
tmp = 0.0
!$acc loop
do k = 1, n2
tmp = tmp + a(i,k,imat)*b(k,j,imat)
end do
c(i, j, imat) = tmp
end do
end do
!$acc end kernels
end do
!$acc end data
```

The outermost ‘data’ region ensures that the input matrices **A and **B** are copied to the GPU, and that the result **C** is copied back to the host.** The outermost loop over ‘imat’ is marked as being the place to start parallelisation using the ‘kernels’ directive. All of the four loops are marked as parallelizable.

MPI for GPU-GPU communication

- An implementation of MPI, MVApich2 supports GPU to GPU communication (in parallel GPU systems), enabling MPI_Send, MPI_Recv from/to GPU memory



At Sender: `MPI_Send(s_device, size, ...);`

At Receiver: `MPI_Recv(r_device, size, ...);`

Conclusions

- Traditional programming models are evolving fast (MPI 3.0, OpenMP 4.0):
 - Non-blocking collectives, neighbor collectives, increased support for one sided communication (MPI)
 - thread affinity, SIMD pragmas (particularly important for Intel MIC), more support task-based approach,
- PGAS languages/libraries are an alternative to MPI. One-sided communication occurs implicitly as remote access to a shared global memory.
- Compiler directives, such OpenACC, will make programming GPUs and accelerators easier.
- First MPI implementation (mvapich) to communicate to GPU memory directly, and support GPU-GPU communication.