ONE-SIDED COMMUNICATION, MPI ON THREADS
OVERLAP OF COMMUNICATION
OLD MPI TOPICS – NEW ANSWERS?

Dr. Heinrich Bockhorst, Intel

INTRODUCTION

Some ideas and concepts have been around since the first definition of MPI. One sided communication is an elegant way of avoiding unnecessary synchronization. MPI and Threads or MPI on Threads may also help to accomplish this goal.

Both concepts can be used to overlap computation and communication. The ultimate goal will be to hide the latency of communication. This will be the key to scalability necessary for Exascale Computing.

The picture on the poster shows 2 neighborhood exchanges. The first with overlap and the second without. We come back to this later.

Thanks to: Jeff Hammond for providing material for One-Sided and MPI + Threads

Klaus-Dieter Oertel for valuable comments

AGENDA

- MPI History
- Overlap of communication and computation
- MPI on threads
- One sided communication

MPI HISTORY: MPI-1 (1994)

- Standardized a bunch of existing messaging libraries.
- Developed when CPU much faster than network and before multicore.
- Send-Recv couple synchronization and data motion. Collectives were synchronous.
- Send-Recv requires either copy from eager buffer or partial rendezvous (to setup RDMA) due to lack of complete information about transaction on either size.
- MPI communicators are amazing for hierarchy, topology, libraries, etc.
- Good match for a lot of numerical apps…

WHY PEOPLE USED MPI-1

- Parallel programmers were using Intel NX, IBM MPL, P4, PARMACS, PVM, TCGMSG, Thinking Machines CMMD, Zipcode, etc. already.

- Standard required!

- Message passing a good match for:
  - Dense linear algebra.
  - Domain decomposition and boundary exchange.
  - Numerical solvers.
  - Monte Carlo.

- Good semantic match for *inexpensive* networks (via TCP/IP).

MPI HISTORY: MPI-2 (1997)

- First awareness of threads. Unfortunately, no one implemented THREAD_MULTIPLE efficiently until Blue Gene/Q. As a result, most applications rely upon THREAD_FUNNELED and fork-join threading.
- Dynamic processes adopted in order to make PVM to disappear. This was arguably the only useful purpose of this feature until people started to think about resilience.
- MPI-IO: parallel IO
- One-sided communication forced into horrible semantic corner by the existence of one strange but unfortunately #1 system (EarthSim), which was not cache-coherent.
- Really dropped the ball on atomics.

21 years later, we still do not have good ***implementations*** *of some of these features…*

MPI HISTORY: MPI-3 (2012)

- Nonblocking collectives! Implementations must get better and reward users for avoiding synchronization.
- Thread-safe Probe (Mprobe) – the right way to do active-messages in MPI-3.
- Topology is a first-class object with distributed graph communicators and neighborhood collectives.
- One-sided (RMA) communication is fixed. Supporting PGAS programming models like Global Arrays, UPC, and OpenSHMEM was an explicit goal.
- POSIX/SysV shared memory rolled into RMA.
- Better (non-collective) subcommunicator creation.

AGENDA

- MPI History
- Overlap of communication and computation
- MPI on threads
- One sided communication

OVERLAP OF COMPUTATION AND COMMUNCIATION

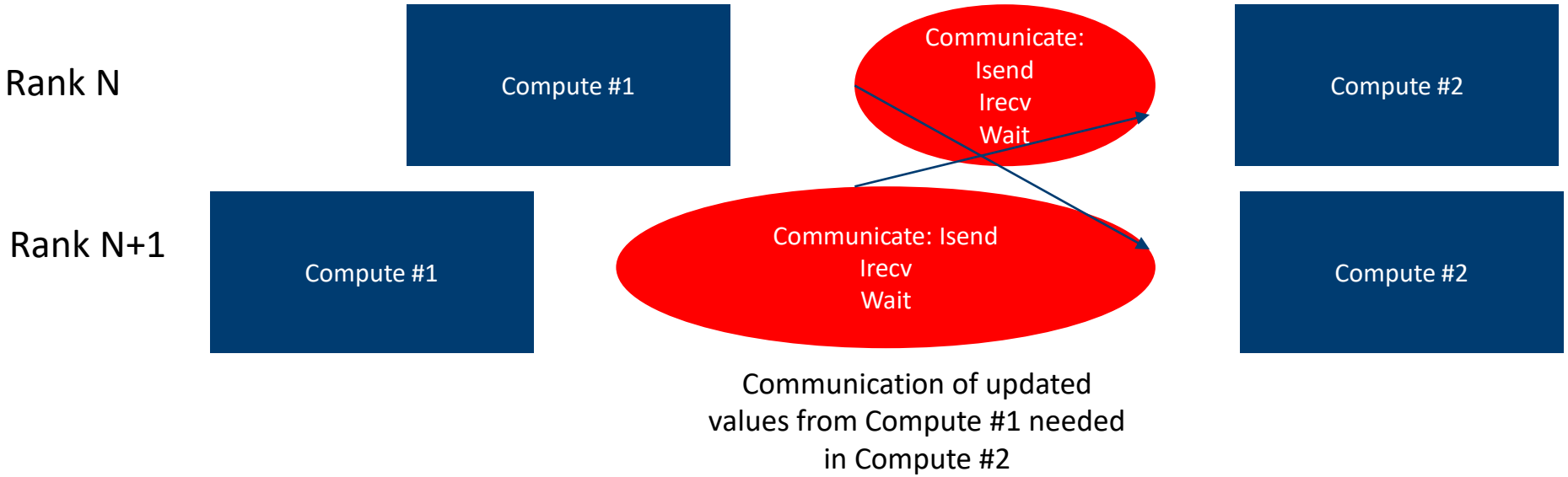
1. Initiate communication
2. Do unrelated computations. Communication should proceed in the background and provide data on the remote process. After the computation finishes the data is already in the right buffer and can be used.
3. Use/Fetch data from communication

- This is also called Latency Hiding
  (Encyclopedia of Parallel computing https://doi.org/10.1007/978-0-387-09766-4_415):

  Latency hiding improves machine utilization by enabling the execution of useful operations by a device while it is waiting for a communication operation or memory access to complete. Prefetching, context switching, and instruction-level parallelism are mechanisms for latency hiding.

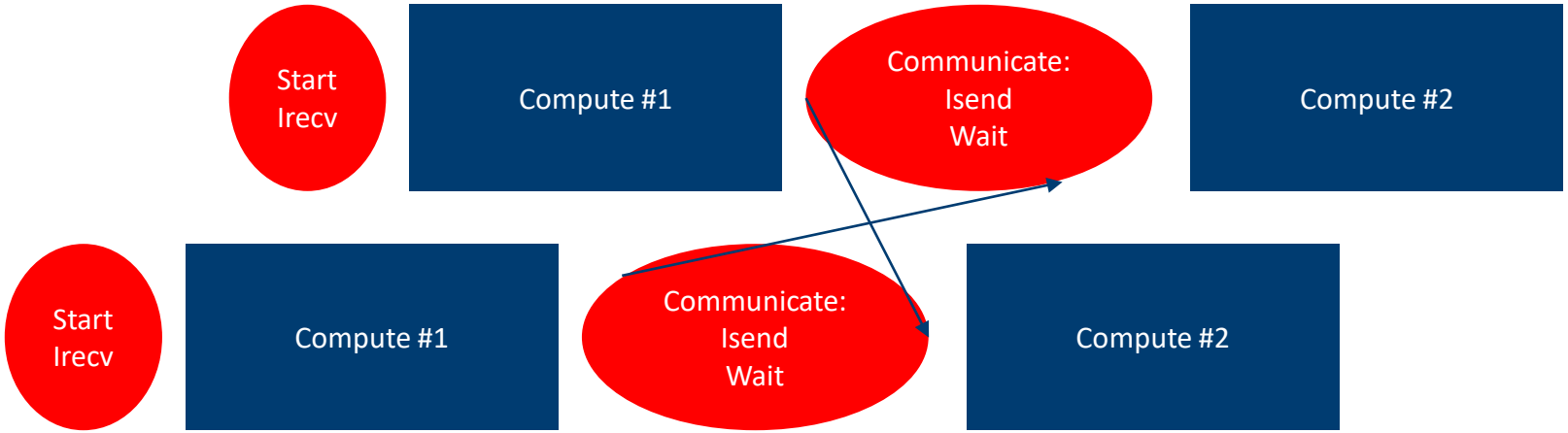
OVERLAP I/III

Implementation without overlap



OVERLAP II/III

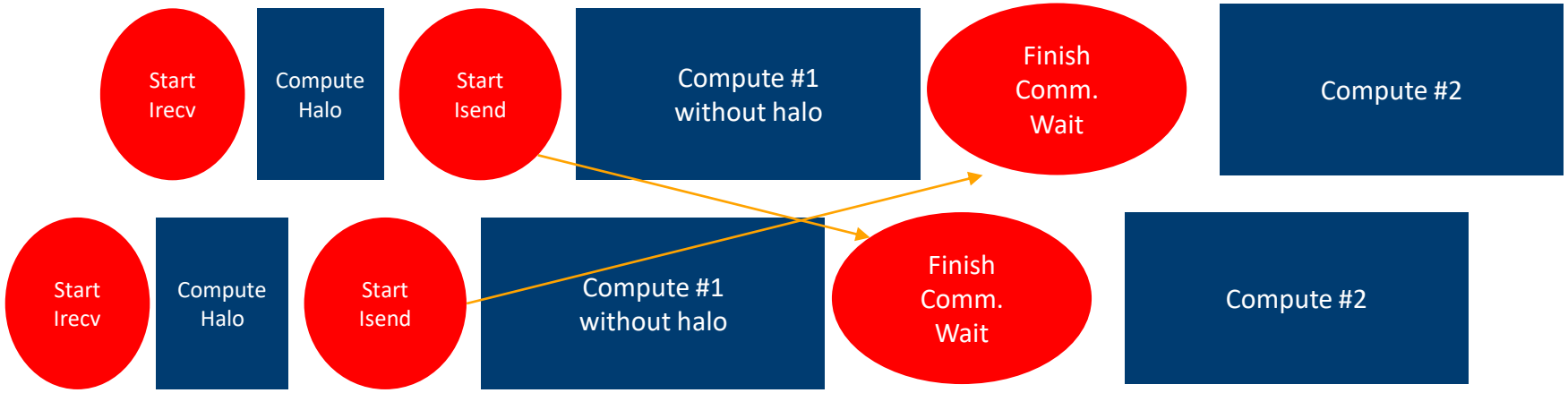
Implementation with some overlap



Receive started before computation. Good if send processor is ahead!

OVERLAP III/III

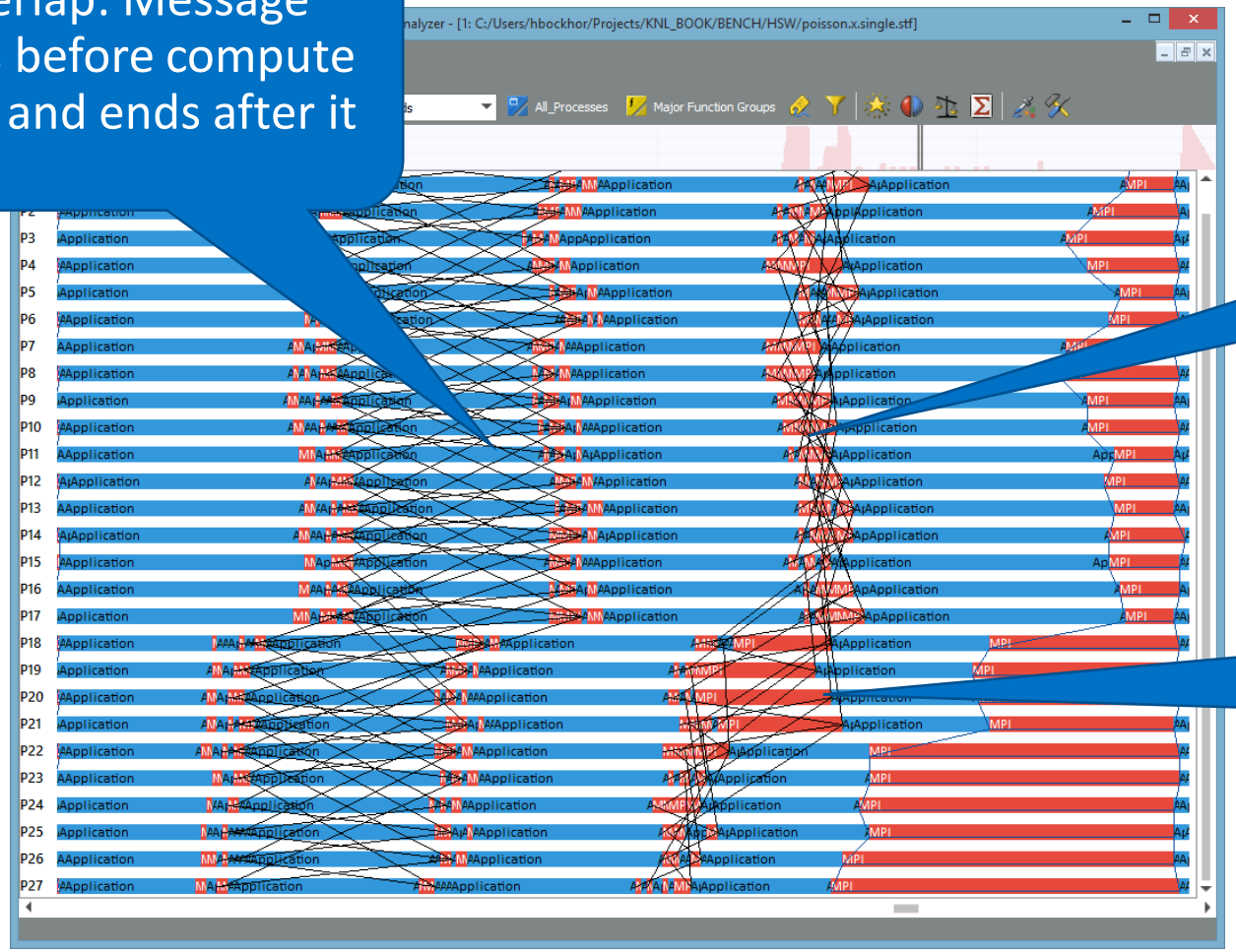
- Implementation with full overlap



Communication overlaps with inner points!

SINGLE ITERATION WITH AND WITHOUT OVERLAP

Overlap: Message starts before compute block and ends after it



NO Overlap: Message starts and is directly received

Imbalance causes MPI waiting time!

PROGRAMMING FOR OVERLAP

Original

```
red();          // update red points

exchange();     // exchange halo points

black();        // update black points

exchange();     // exchange halo points
```

Overlap

```
red_halo();     // update red halo
exchange_send_red_halo();

red-();     // update red point wo halo

exchange_wait_red_halo();
                // halo points received

black_halo();
exchange_send_black_halo();

black-();   // update black wo halo

exchange_wait_black_halo();
```

DOES MPI SUPPORT OVERLAP?

- MPI message may only start after MPI_Wait is called on the receiver side ➔ no overlap is possible!
- True overlap maybe only possible with additional helper thread doing all the spinning and polling
- MPICH environment variable MPICH_ASYNC_PROGRESS=1 will start a helper thread (Intel: I_MPI_ASYNC_PROGESS)

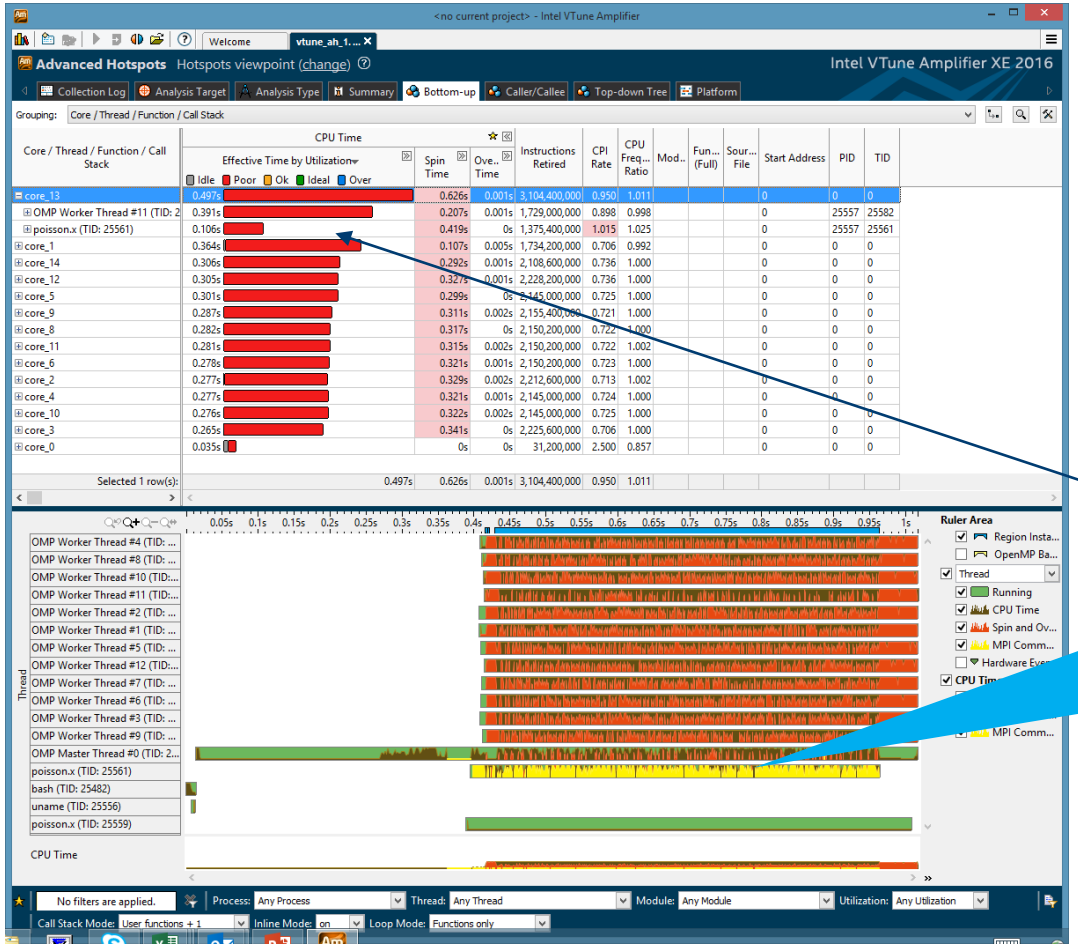
I_MPI_ASYNC_PROGRESS=1

- Additional thread is spawned from the thread doing MPI calls. Usually the OpenMP master thread

- Observation: Helper Thread does interfere with OpenMP threads. OpenMP master still has some MPI overhead (synchronization with Helper Thread)

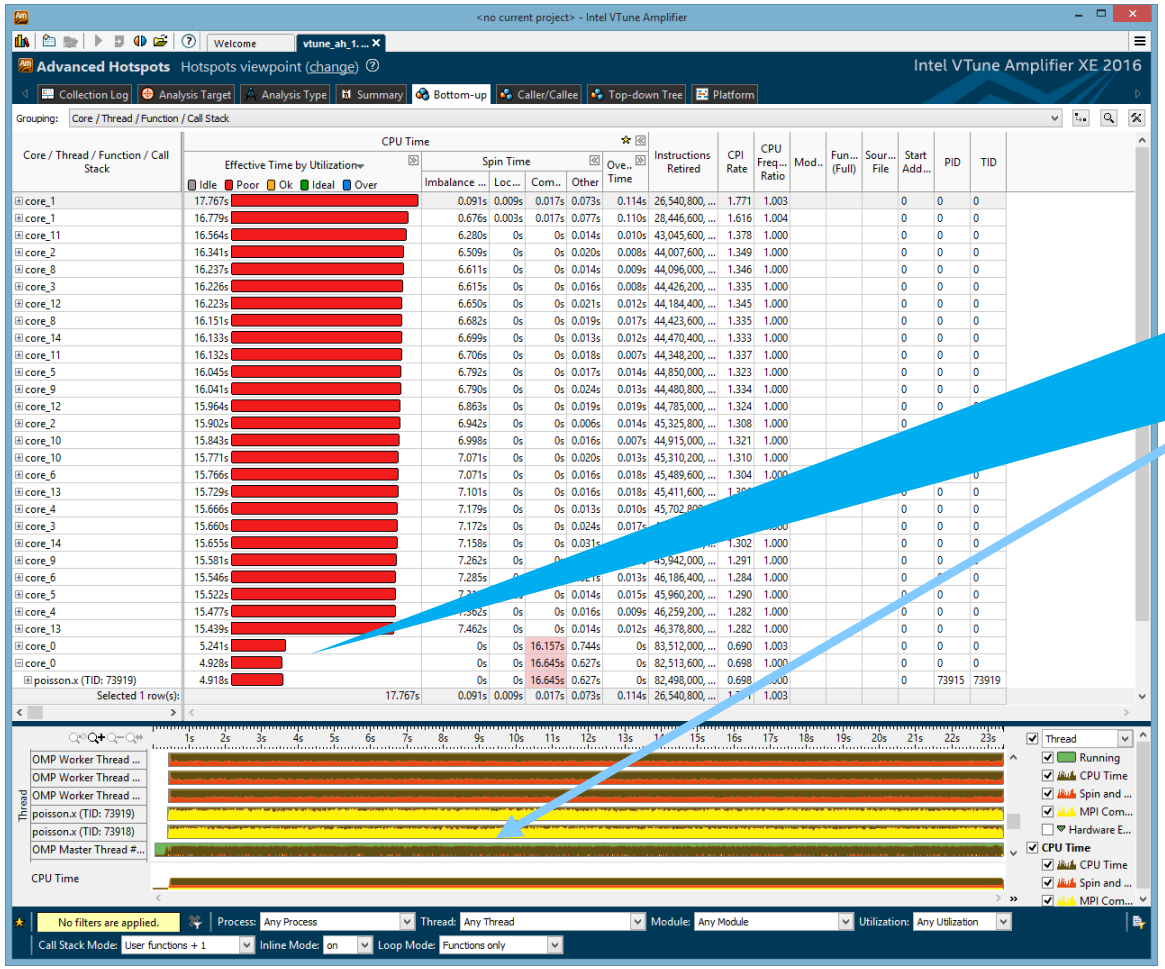
- Helper Threads can be pinned by help of variable: I_MPI_ASYNC_PROGRESS_PIN

HELPER THREAD UNPINNED



Helper Thread:
poisson.x
Is by chance located on the same core as OMP Thead # 11

PINNED HELPER THREADS; 2 MPI RANKS PER NODE



Helper Threads: poisson.x
Both helper Threads are located on core 0 of each socket

ISSUES AND FURTHER OPTIMIZATION

- Overlap version does not provide the expected benefit
- Progress engine of MPI does pause during computation on the receiver rank

- ASYNC Progress does use an additional “Helper Thread” for keeping the progress engine busy
- Helper Thread interferes with other computation and communication threads
- Explicit pinning of Helper Thread is possible but needs complicated affinity settings
- Benchmarking shows now real benefit

- Idea: reserve cores for MPI communication thread and additional Helper Threads

AGENDA

- MPI History
- Overlap of communication and computation
- MPI on threads
- One sided communication

MPI + THREADS INTRO

- MPI and threads is defined since MPI-2
- We have different levels of threading support:

MPI_THREAD_SINGLE:	only single thread will execute
MPI_THREAD_FUNNELED:	may be multithreaded but only main calls MPI
MPI_THREAD_SERIALIZED:	may be multithreaded and MPI can be called on different threads. Only one thread is allowed to call MPI at a time (serialization)
MPI_THREAD_MULTIPLE:	multiple threads can call MPI without restrictions

MPI-2 and Threads

- MPI_Init_thread(.., FUNNELED);
- #omp parallel
- {
- for (..) { Compute(..); }
- #omp master
- { MPI_Bar(..); }
- }
- MPI_Foo(..);

- MPI_Init_thread(.., SERIALIZED);
- #omp parallel
- {
- for (..) {
- Compute(..);
- #omp critical
- { MPI_Bar(..); }
- }
- }
- MPI_Foo(..);

MPI-2 and Threads II

- MPI_Init_thread(.., MULTIPLE);
- #omp parallel
- {
- Compute(..);
- MPI_Bar(..);
- }
- MPI_Foo(..);

This is the ONLY method that works reliably with more than one threading model!

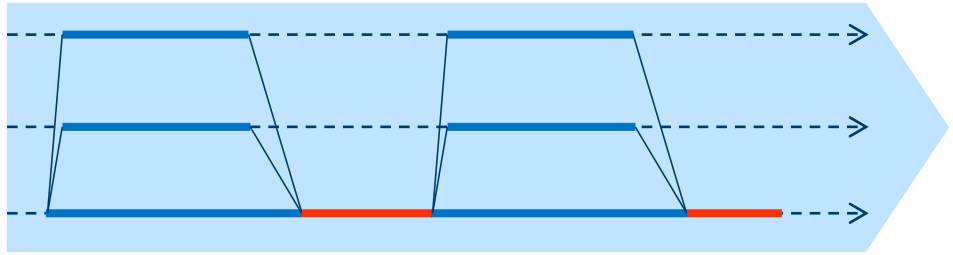
Common Implementation

- int MPI_Bar(..)
- {
- if (MULTIPLE) Lock(Mutex);
- rc = MPID_Bar(..);
- if (MULTIPLE) Unlock(Mutex);
- return rc;
- }

Optimized

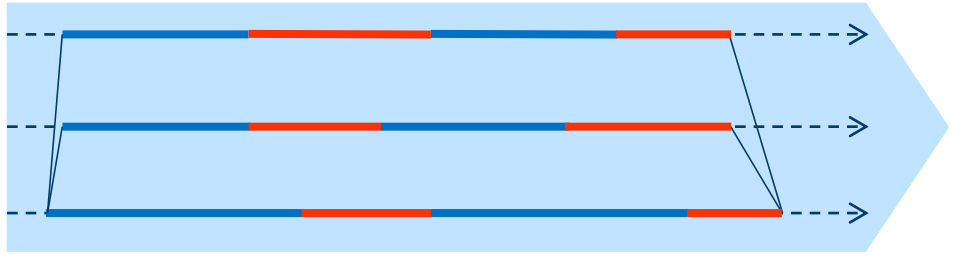
```
int MPI_Bar(..)
{
  return MPID_Bar(..);
  /* ^ fine-grain locking
          inside of this call… */
}
```

MPI + THREADS PROGRAMMING MODELS



Funneled Model

MPI_Init_thread(MPI_THREAD_SINGLE)
The best MPI can do now



True Multi-threaded Model (MPI-MT)

MPI_Init_thread(MPI_THREAD_MULTIPLE)
Why is MPI so slow ???

Communication thread(s) model

MPI_Init_thread(MPI_THREAD_SINGLE) or
MPI_Init_thread(MPI_THREAD_MULTIPLE)

Timeline diagram: Light green arrow - rank, Dotted arrow - thread, Blue - computation, Red - MPI communication, Connectors - parallel region

MPI IN MULTI-THREADS - WHAT'S WRONG

MPI 3.1 Standard: *"threads are not separately addressable: a rank in a send or receive call identifies a process, not a thread"*

- Any thread may send/receive a message to a peer
- Single global receive queue has to be supported
  - Global, or fine grained lock imposed
  - Overheads are more than just serialization
- MPI objects are global
  - Object management overhead incurred

MPI ON MULTIPLE THREADS – HOW TO RESOLVE

- Message from different threads may be distinguished by providing a tag that encodes the thread id

- Message from different threads may also be given different communicators

INTEL® MPI 2019: ENHANCED SUPPORT FOR HYBRID PROGRAMMING MODELS

- New MPI_THREAD_MULTIPLE optimization
  - Available with release_mt library version: I_MPI_THREAD_SPLIT=1

- The optimization allows to achieve the following:
  - Improve aggregated communication bandwidth & message rate
  - Communicate as soon as data is ready, not waiting for the slowest thread
  - Avoid implied bulk synchronization threading barriers, overhead on parallel sections start/stop

- Full stack is available with Intel® Omni-Path Fabric Software 10.5 and OFI 1.5.0

Improved flexibility for fine grain collective operations tuning

MPI-MT: HOW TO GET PERFORMANCE

- Reduced/narrowed MPI_THREAD_MULTIPLE model introduced: *MPI_THREAD_SPLIT (or **thread-split** model)*
  - Non-standard, available in Intel MPI with I_MPI_THREAD_SPLIT=1
- Threads are addressed with a dedicated communicator - now separate per-thread queues possible
  - Matching the same message from concurrent threads not allowed
  - Explicit/implicit thread addressing sub-models apply (next slide)
- MPI request objects cannot be accessed from >1 thread at the same time - now can be in TLS
  - No multi-threads access penalty

PROGRAMMING WITH THE THREAD-SPLIT MODEL

Implicit sub-model

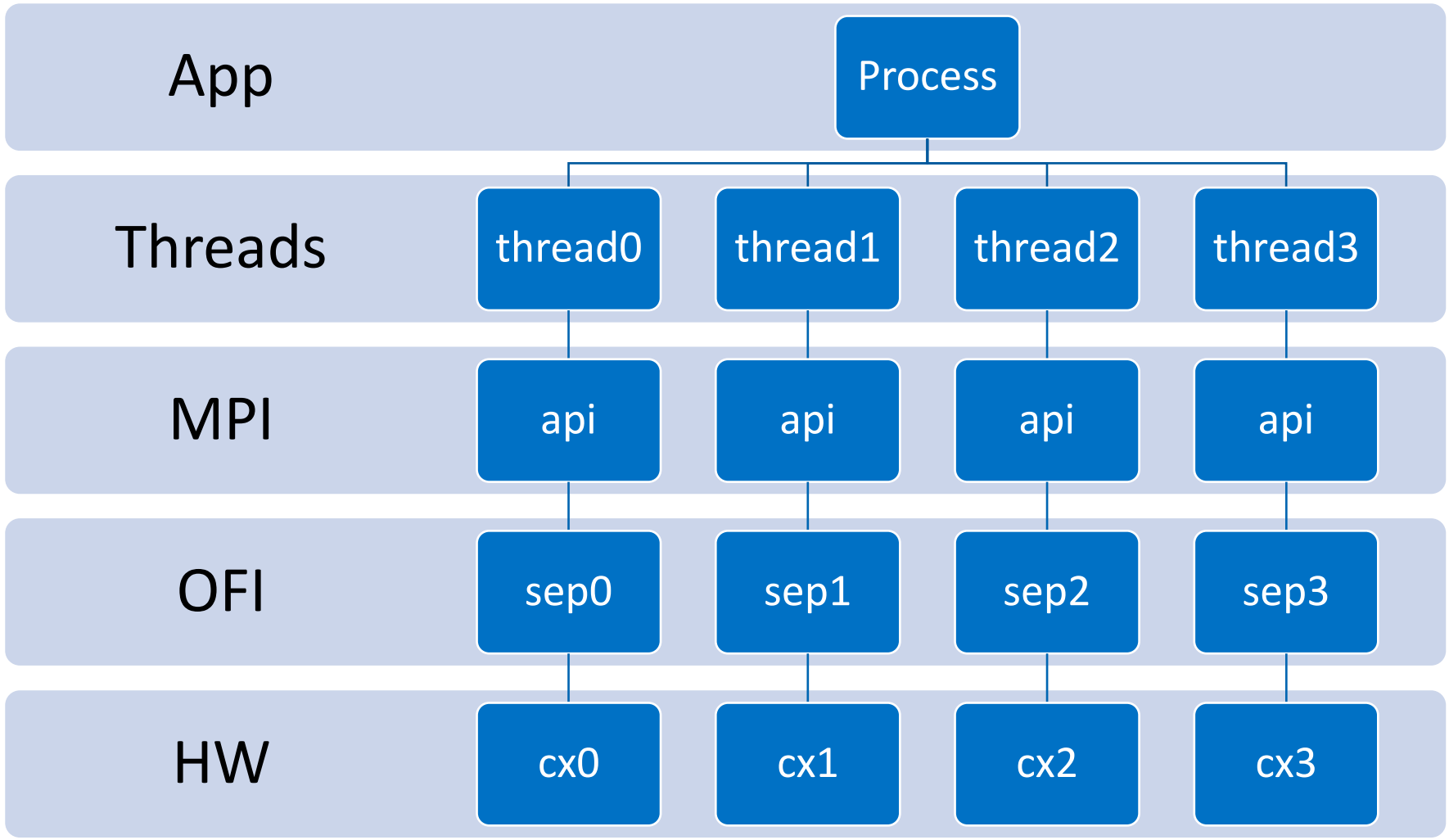
```
#define n 2
MPI_Comm split_comm[n];
int main() {
    int i, j, provided;
    MPI_Init_thread(NULL, NULL, MPI_THREAD_MULTIPLE,
                    &provided);
    for (i = 0; i < n; i++)
        MPI_Comm_dup(MPI_COMM_WORLD, &split_comm[i]);
#pragma omp parallel for private(j) num_threads(n)
    for (i = 0; i < n; i++)
        MPI_Allreduce(&i, &j, 1, MPI_INT, MPI_SUM,
                      split_comm[i]);
    MPI_Finalize();
}

export OMP_NUM_THREADS=2
export I_MPI_THREAD_SPLIT=1
```

Explicit sub-model

```
#define n 2
int thread_id[n];
MPI_Comm split_comm[n];
pthread_t thread[n];
void *worker(void *arg) {
    int i = *((int*)arg), j;
    MPI_Allreduce(&i, &j, 1, MPI_INT, MPI_SUM, split_comm[i]);
}
int main() {
    ...
    MPI_Init_thread(NULL, NULL, MPI_THREAD_MULTIPLE,
                    &provided);
    MPI_Info_create(&info);
    for (i = 0; i < n; i++) {
        MPI_Comm_dup(MPI_COMM_WORLD, &split_comm[i]);
        ...
        MPI_Comm_set_info(split_comm[i], info);
        pthread_create(&thread[i], NULL, worker,
                       (void*) &thread_id[i]);
    }
    for (i = 0; i < n; i++) pthread_join(thread[i], NULL);
    ...
}
export I_MPI_THREAD_MAX=2
export I_MPI_THREAD_SPLIT=1
```

Refer to the Intel® MPI Library Developer Guide for the full set of examples

INTEL® MPI 2019: ENHANCED SUPPORT PROVIDED BY ALL LIBRARIES



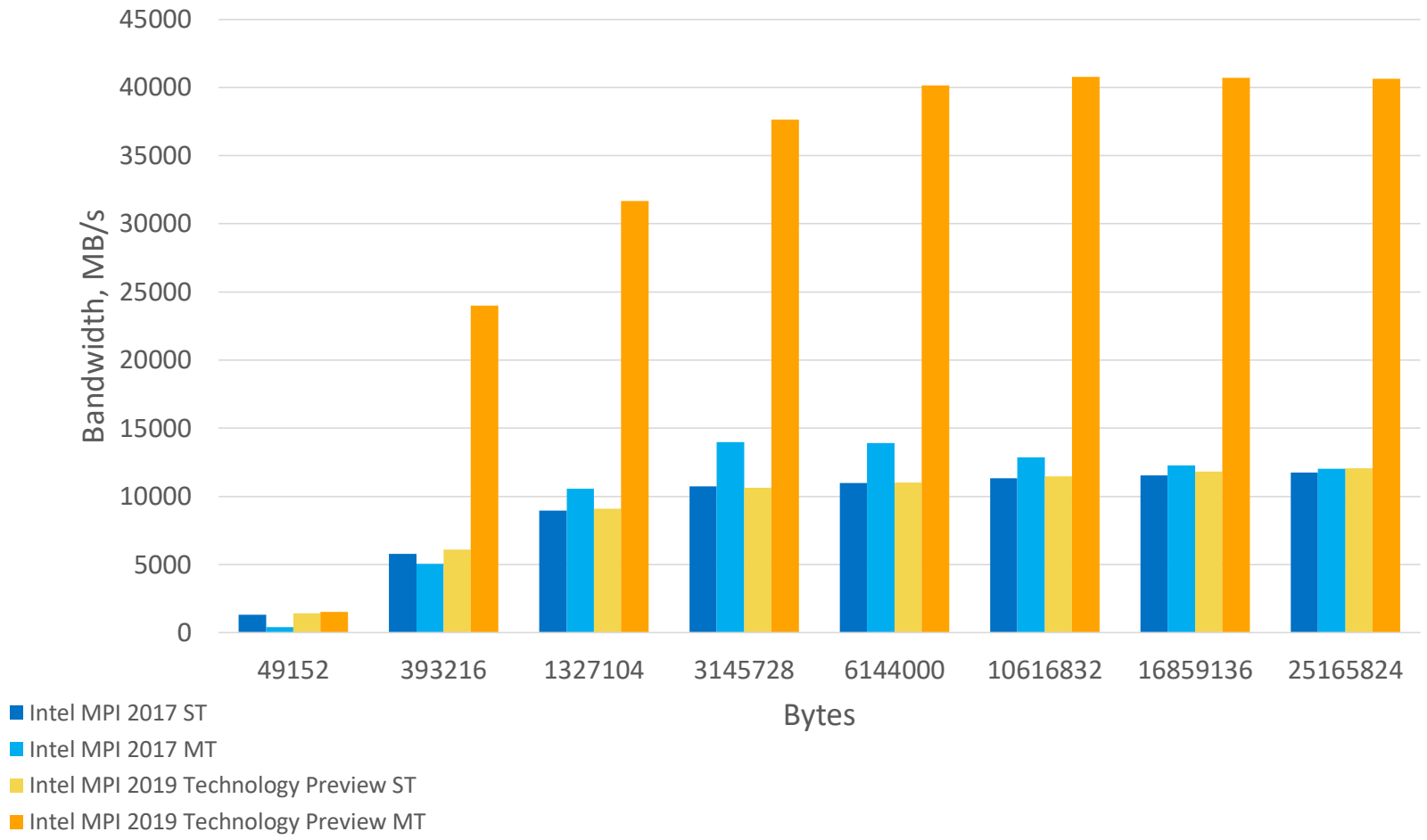
- OFI scalable endpoint

- Independent HW context

The I_MPI_THREAD_SPLIT feature allow to get minimal amount of serialization

INTEL® MPI 2019 TECHNICAL PREVIEW: HYBRID PROGRAMMING MODEL IMPACT EXAMPLE

Grid QCD Performance Benchmark (courtesy of Prof. Peter Boyle)



Hardware:
- Intel® Xeon® Platinum 8170 processor
- Intel® Omni-Path Host Fabric Interface, dual-rail

Software:
- Intel® MPI Library 2017 Update 4
- Intel® MPI Library 2019 Technology Preview
- Libfabric 1.5 (OFI)
- IFS 10.5
- Grid benchmark1
- huge memory pages

[1] The benchmark software uses the Grid QCD library https://github.com/paboyle/Grid and the peformance benchmark benchmarks/Benchmark_comms.cc is used.
The tests are run on 16 nodes system is divided into a four dimensional Cartesian communicator with 24 ranks, and one rank per node. These dimensions are called {x, y, z, t}. Each node is given a four dimensional volume L^4, and the global volume is $G^4 = (2L)^4$. The code mimics the communications pattern for a halo exchange PDE arising in quantum chromodynamics.
Each node sends packets of size the surface L^3 to neighbors in each of +x, -x, +y, -y, +z, -z, +t, and -t directions, while concurrently receiving the neighbors data.

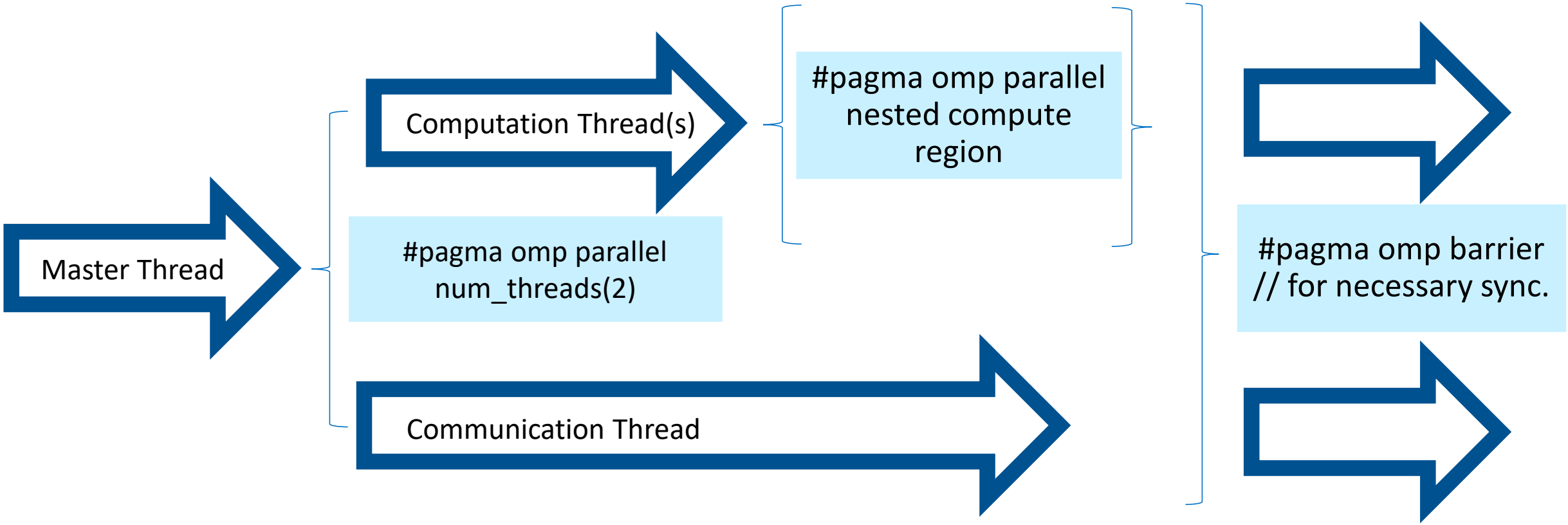
MULTI-EP SOFTWARE RECIPE

- Requires components from:
  - Intel® Omni-Path Fabric Suite Fabric Manager (IFS) version 10.5 or newer
  - OpenFabrics Interfaces (OFI) Libfabric version 1.5 or newer
  - Intel® MPI Library 2019 (or 2019 Technical Preview)
- Example execution, using 16 nodes, 1 MPI rank per node with 4 endpoints:
  - source /opt/intel/impi/2019.0.070/bin64/mpivars.sh release_mt -ofi_internal
  - export I_MPI_THREAD_SPLIT=1
  - export I_MPI_THREAD_RUNTIME=openmp
  - export PSM2_MULTI_EP=1

  - mpirun -np 16 -ppn 1 -hostfile 16nodes -genv I_MPI_FABRICS shm:ofi -genv OMP_NUM_THREADS=4 ./myapplication

PSM - Performance Scaled Messaging

MPI + THREADS ADVANCED (COMMUNICATION THREAD MODEL)

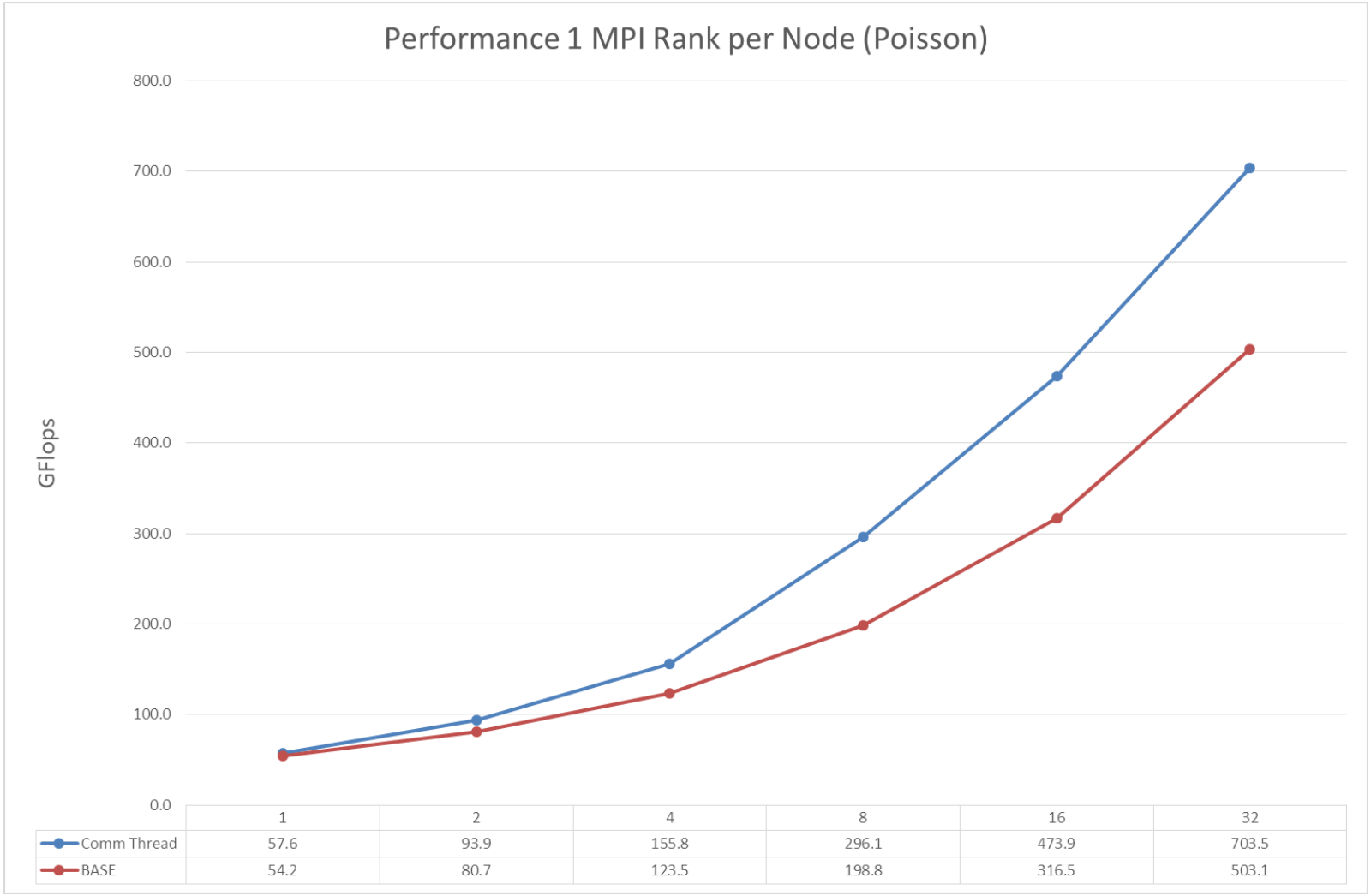


ENVIRONMENT VARIABLES FOR NESTED OPENMP

- OMP_NESTED=true # without this the nested region is serialized
- KMP_HOT_TEAMS_MAX_LEVEL=2 (Intel) # maximum nested level used. Keeps threads of nested # regions ready for next region
- KMP_HOT_TEAMS_MODE (Intel) # = 0 : extra threads are freed and put into pool (default) # = 1 : extra threads kept in team, for faster reuse
- OMP_NUM_THREADS=2,MAXTH-1 # MAXTH=(#cpus/#mpi_ranks)

- Setting these variables is necessary to get any performance. Hot Teams will ensure that the same threads will run on nested parallel regions of compute threads
- The two level OMP_NUM_THREADS was the missing part to make it work!

MPI ON THREADS BENCHMARKING (SIMPLE 2D POISSON SOLVER)



	1	2	4	8	16	32
Comm Thread	57.6	93.9	155.8	296.1	473.9	703.5
BASE	54.2	80.7	123.5	198.8	316.5	503.1

KNL NODES of JURECA Booster

KNL 7250F with OPA

MPI ON OPENMP TASKS - PLAN FOR NEAR FUTURE

- OpenMP/TBB tasks are getting more interest because of more opportunities for dynamic, unstructured workloads
- Tasks offer dependencies for minimizing synchronization instead of only barriers
- Affinity with HW resources is hard to achieve
- IDEA: Run MPI task(s) that synchronizes with compute tasks via dependencies
- Flow Graph Analyzer is a tool with experimental support of OMP tasks

SWIFT CODE - UNIVERSITY OF DURHAM

- Astrophysics code. Competition to Gadget
- (Smoothed Particle Hydrodynamics) SPH with Inter-dependent fine-grained Tasking: http://swift.dur.ac.uk/
- Task Library can be used for other problems like Molecular Dynamics (tried by TU Munich)
- MPI is used on send and receive Tasks. Issue was the too large number of tags used. 2019 version shows much lower MPI usage times.

AGENDA

- MPI History
- Overlap of communication and computation
- MPI on threads
- One sided communication

ONE-SIDED COMMUNICATION INTRO

- MPI One-Sided Communications were introduced already with the MPI2 standard. There was no great adoption of the new routines because syntax and missing hardware support were preventing well performing implementations.

- The great advantage of One-Sided Communication is to avoid synchronization. A process can put data into the memory of another process without an explicit receive call. In MPI-2 a lot of bulk collective synchronization is necessary (MPI_Win_fence)

- MPI-3 allows more fine grained synchronization between pairs of processors

- The conditions have changed now with the MPI3 standard and more appropriate hardware (Cray DMAPP, Intel Omni-Path)

- Book: Using Advanced MPI, Chapter 3+4

MPI2: ONE-SIDED PROGRAMMING

Typical pattern for MPI_Get usage:

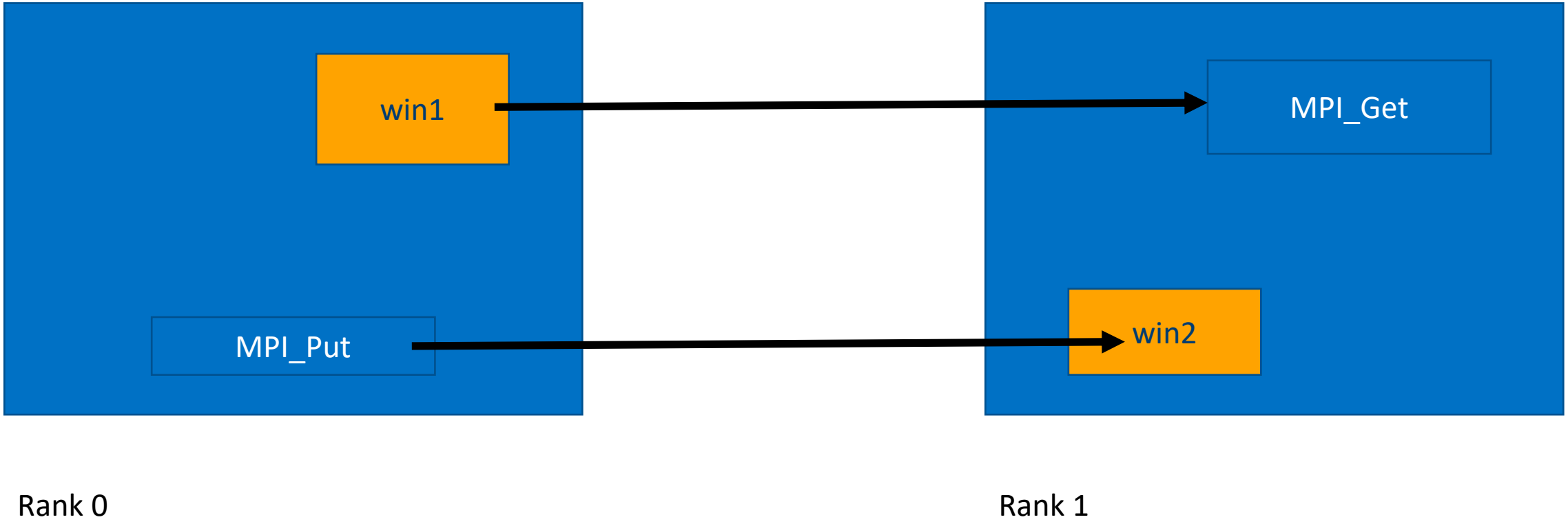
```
MPI_Win_Create(buff, size, ..., win)
MPI_Win_Fence(0, win)                    // collective operation on all PE using win
MPI_Get(&buff2, size1, ..,from_PE, ..., win)
// computation....
// more non overlapping MPI_Get calls
MPI_Win_Fence(0,win)                     // collective operation on all PE using win
```

➔ High synchronization cost, not scalable

ONE-SIDED PROGRAMMING - PICTURE

Remote Access Memory (RMA) windows

Local Memory



DISADVANTAGE OF MPI-2 ONE-SIDED COMMUNICATION

- Collective calls to MPI_Win_fence are necessary to separate RMA operations from local loads and stores

- The target process is actively involved because it has to call MPI_Win_fence: active target synchronization

- This cumbersome syntax was chosen to ensure that One-Sided Communication could be implemented on a wide range of HW with or without Cache Coherence.

MPI -3 : ONE-SIDED COMMUNICATION

- One of the main targets was to relax the memory model of MPI-2 One-Sided Communications

- Passive target synchronization in MPI-3 does not need an explicit call on the target processor

- An communication access epoch is started by a MPI_Win_lock call and ended by a MPI_Win_Unlock call . These calls can be used to access remote memory on a specific target processor. MPI_Win_lock_all gives access to all remote processes of a memory window

- The name “lock” may be confusing because It does not behave like a shared memory lock we know from threading. The lock/unlock is more like a start/end of RMA operation for the specified window.

MPI-3 PROGRAMMING EXAMPLE – SYNCHRONIZATION EPOCHS

```
MPI_Win w;
/* construct window */
MPI_Win_lock_all(MPI_MODE_NOCHECK,w); /* “PGAS mode” */
{
      …
      MPI_Put(..,pe,w);     /* all RMA communications are nonblocking */
      MPI_Win_flush_local(pe,w);      /* local completion */
      MPI_Win_flush (pe,w);  /* remote completion = global visibility */
      …
}
MPI_Win_unlock_all(w);
MPI_Win_free(w);
```

MPI-3 PROGRAMMING : REQUEST BASED

- The request based versions MPI_Rput and MPI_Rget have a MPI_Request as additional argument.
- Programming MPI_Requests is well known from MPI_Isend, MPI_Irecv, MPI_Wait,…
- The additional advantage is that we can use MPI_Test etc to determine the state of the communication procedure

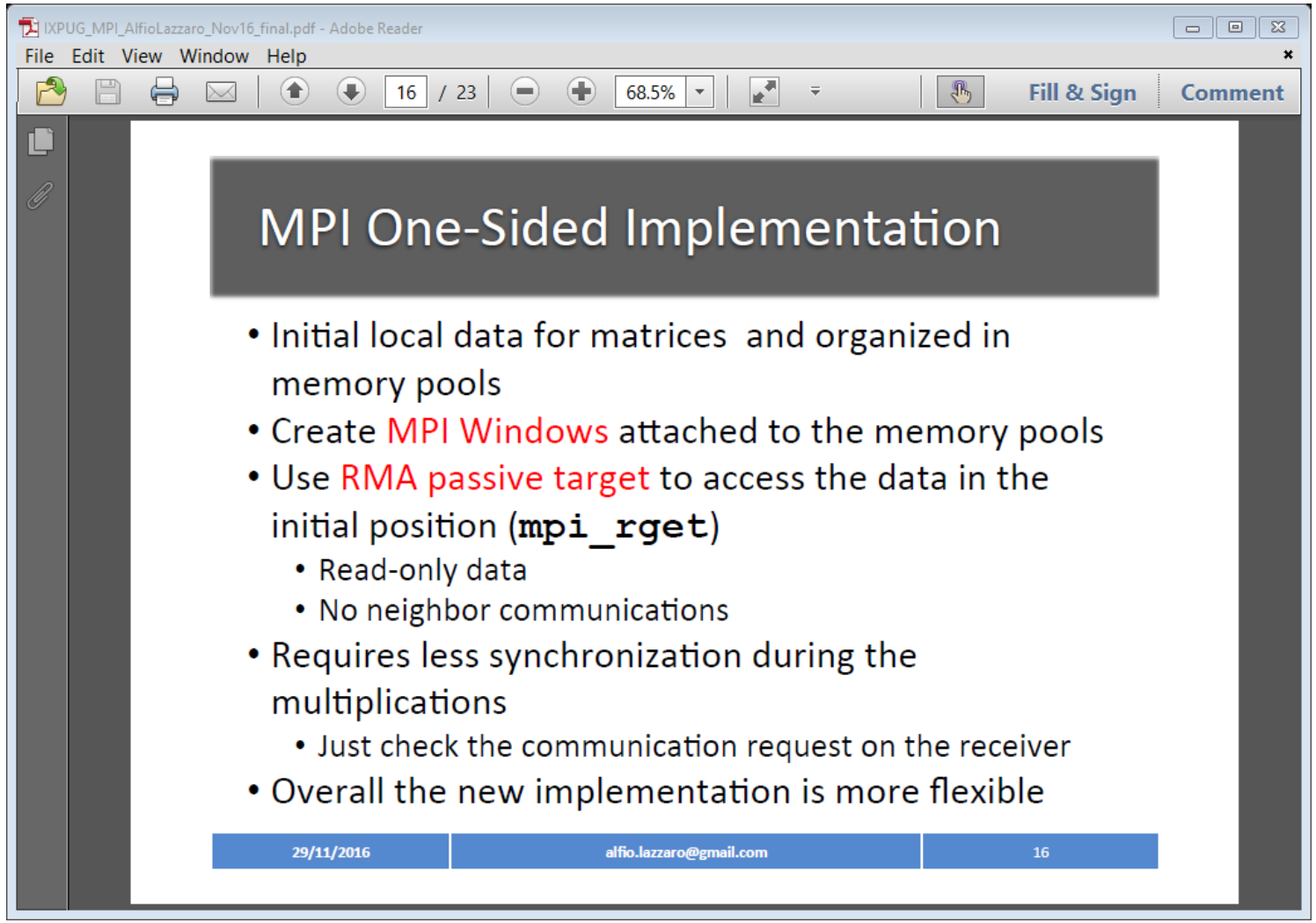
MPI-3 REQUEST BASED CODING SKETCH

```
MPI_Request request;
MPI_Win_allocate(size, …,comm ,&buf, &win); // collective win. alloc.

MPI_Win_lock_all(0,win);                    // access win on all pe of comm

MPI_Rget(&data, …, target_pe, .., win, &request)

//Compute unrelated to data to be overlapped

MPI_Wait(&request, MPI_STATUS_IGNORE);  // wait on rget to be finished

compute(&data,…);                           // use the data

MPI_Win_unlock_all(win);
```

MPI_RGET/MPI_RPUT ADVANTAGE

- Multi-Buffering can be used to overlap communication and computation. Each buffer will be associated with a request. Buffer can be read and reused after request is finished by MPI_Wait
- MPI_Test can be used to check the status of an request
- MPI_Waitall/MPI_Waitany can be used to finish all requests or find a finished request and start a new MPI_Rget/MPI_Rput using the “free” request
- Existing logic from programs with MPI_Isend, MPI_Irecv, MPI_Wait may be used and improved because only MPI_Rget + MPI_Wait or MPI_Rput + MPI_Wait are necessary!

CP2K BY ALFIO LAZZARO, ETH ZÜRICH I/II



https://arxiv.org/pdf/1705.10218.pdf

CP2K BY ALFIO LAZZARO, ETH ZÜRICH II/II

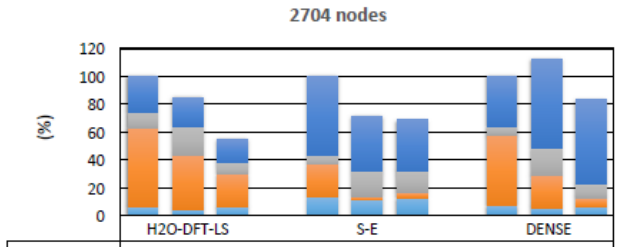
IXPUG_MPI_AlfioLazzaro_Nov16_final.pdf - Adobe Reader

File Edit View Window Help

19 / 23 68.5% Fill & Sign Comment

Performance results

- CRAY MPI + DMAPP
- Normalized values to Point-to-Point timings
  - First column: Point-to-Point implementation
  - Second column: One-sided implementation with single layer
  - Third column: One-sided implementation with 4 layers



	H2O-DFT-LS			S-E			DENSE		
ETC	26	21	17	57	39	36	36	64	60
Sync	12	20	8	6	18	16	7	20	10
Wait comm	57	39	24	24	3	4	51	23	6
Computation	6	4	6	13	11	13	7	6	6

- Good speed-up from one-sided implementation
- Further speed-up with layers
- Memory footprint under control (<3GB per rank)

29/11/2016 | alfio.lazzaro@gmail.com | 19

PRESENTATION AND PROJECT LINKS ABOUT ONE-SIDED

- Florian Wende HLRN ZIB: https://www.hlrn.de/twiki/pub/NewsCenter/ParProgWorkshopFall2017/ws_mpi3_onesided.pdf

- Jeff Hammond, Intel: ARMCI-MPI is a completely rewritten version of ARMCI based on MPI-3 RMA: https://github.com/jeffhammond/armci-mpi. The paper http://www.mcs.anl.gov/publication/supporting-global-arrays-pgas-model-using-mpi-one-sided-communication describes ARMCI-MPI using MPI-2 RMA. This won't be particularly helpful since the major result of the paper is that we fixed MPI-3 RMA to make libraries like Global Arrays (and DDI) work better

- Jeff Hammond, Intel: OSHMPI is an implementation of OpenSHMEM written in MPI-3 RMA: https://github.com/jeffhammond/oshmpi. https://github.com/jeffhammond/oshmpi/blob/master/docs/iwosh-paper.pdf?raw=true is a paper about OSHMPI that describes MPI-3 RMA and how it maps to SHMEM, in case you have any familiarity with SHMEM

- William Gropp: lectures on One-Sided: http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture34.pdf and http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture35.pdf

Optimization work on threaded MPI

- P. Balaji, D. Buntinas, D. Goodell, W. D. Gropp, and R. Thakur. 2010. *Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming*. Int. J. High Perform. Comput. Appl. 24 (Feb. 2010), 49–57.
- D. Goodell, P. Balaji, D. Buntinas, G. Dozsa, W. Gropp, S. Kumar, B. R. de Supinski, and R. Thakur. 2010. *Minimizing MPI Resource Contention in Multithreaded Multicore Environments*. In Proceedings of the 2010 IEEE International Conference on Cluster Computing (CLUSTER '10). IEEE Computer Society, Washington, DC, USA, 1–8.
- G. Dozsa, S. Kumar, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, J. Ratterman, and R. Thakur. 2010. *Enabling Concurrent Multithreaded MPI Communication on Multicore Petascale Systems*. In Proceedings of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface (EuroMPI'10). Springer-Verlag, Berlin, Heidelberg, 11–20.
- A. Amer, H. Lu, Y. Wei, P. Balaji, and S. Matsuoka. 2015. *MPI+Threads: runtime contention and remedies*. In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 239–248.
- K. Vaidyanathan, D. Kalamkar, K. Pamnany, J. Hammond, P. Balaji, D. Das, J. Park, and B. Joo. SC15. *Improving concurrency and asynchrony in multithreaded MPI applications using software offloading.* http://dx.doi.org/10.1145/2807591.2807602

Legal Disclaimer & Optimization Notice

- Performance results are based on testing as of 05.11.2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.
- 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. For more complete information visit www.intel.com/benchmarks.
- 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.
-

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


intel
Software