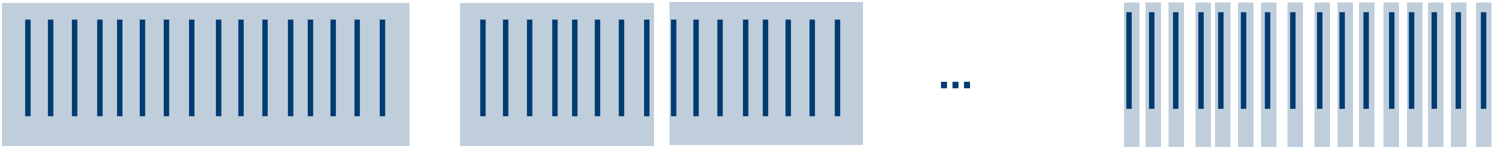
MULTIPLE ENDPOINTS FOR IMPROVED MPI PERFORMANCE ON A LATTICE QCD CODE

Larry Meadows (Intel), Ken-Ichi Ishikawa (University of Hiroshima), Taisuke Boku (University of Tsukuba) and **Masashi Horikoshi** (Intel)

January 31, 2018 (IXPUG Workshop at HPC Asia 2018)

Multi-EP: What and Why

A Hybrid OpenMP-MPI program with 16 cores can be laid out in 5 different ways:



1 MPI 16 OpenMP — 2 MPI 8 OpenMP — … — 16 MPI 1 OpenMP

Without multi-EP, only one thread per rank can do MPI calls

With one MPI thread per rank, how many MPI ranks to use?
Pros and cons
Fewer MPI:

- Fewer ranks = less overhead for collectives (allreduce etc)
- Fewer MPI calls/core, less MPI overhead per rank
- Fewer ranks may require less aggregate transfer (surface-to-volume ratio smaller; algorithm dependent)

More MPI:

- Less effort required to improve threading because fewer threads/rank
- More simultaneous MPI gives higher aggregate fabric throughput

Multi-EP allows more than one MPI thread per rank:

- Saturate fabric with fewer ranks
- Application has more choice in layout
- Enables MPI in OpenMP regions, reducing fork-join and improving cache locality

Multi-EP in the Intel® MPI Library

https://software.intel.com/sites/default/files/managed/c7/80/130384-hpcdevcon2017-High-performance-Threaded-Intel-MPI-Library-on-Intel-Omni-Path-Architecture-Under-the-Hood.pdf

Supports high performance with MPI_THREAD_MULTIPLE (multiple threads in MPI library simultaneously). Needed to saturate Intel® Omni-path Architecture bandwidth.

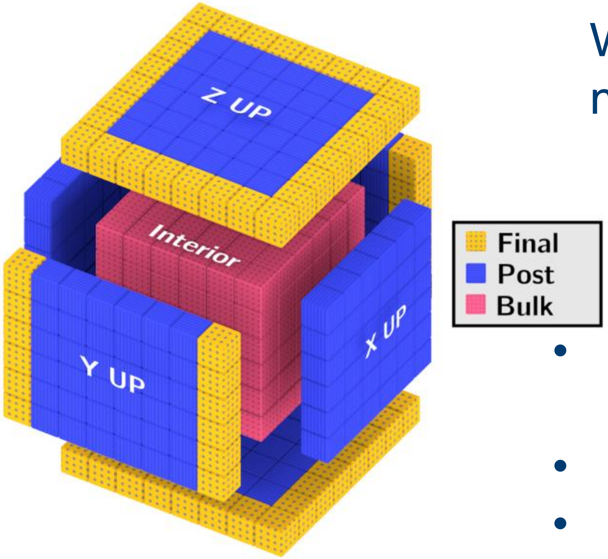
Each thread uses a separate communicator so there is no locking in the library

Use `MPI_Init_thread(...MPI_THREAD_MULTIPLE...)`

Set some environment variables:

```
I_MPI_THREAD_SPLIT=1

I_MPI_THREAD_RUNTIME=openmp

PSM2_MULTI_EP=1
```

The Application: Lattice QCD MULT operator



Wilson-Clover Dirac hopping matrix multiplication operator (MULT)

- 4d +/-1 stencil, distributed in 3 or 4 dimensions (X,Y,Z,T)
- Figure shows 3d distribution
- Interior (bulk) points can be computed while exchanging surfaces with MPI
- In this formulation “post” and “final” iterations are computed in the same loop nest

Static Workstealing scheduler*

C++ object that does initial static distribution with workstealing using OpenMP threads:

- Declare an opaque per-core data structure to coordinate threads:

```
Percore cores[maxCores];
```

- Instantiate a Sched object with an initial static distribution per core, possibly assigning zero iterations to MPI threads:

```
Sched sch(int niters, Percore cores, int nMPIthreads=0)
```

- Replace the OpenMP for with a while loop:

```
while ((block = sch.nextiter()) != 1) { work_on_block(block); }
```

(*Larry Meadows and Ken-ichi Ishikawa. 2017. *OpenMP Tasking and MPI in a Lattice QCD Benchmark*. Springer International Publishing, Cham, 77–91. https://doi.org/10.1007/978-3-319-65578-9_6)

MULT Algorithm Pseudo-code

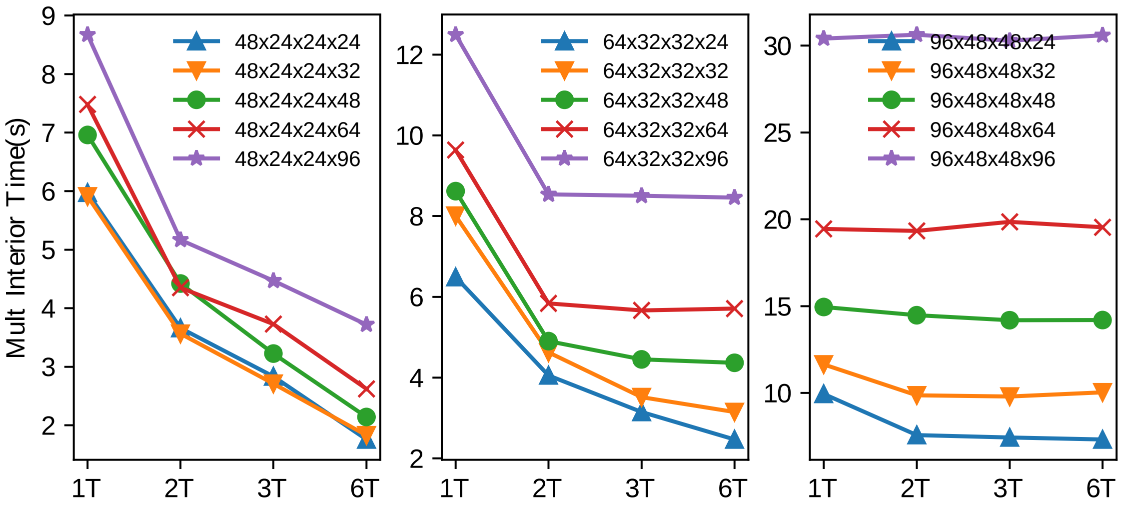
```
#pragma omp parallel
{
  // compute face contributions to send
  #pragma omp barrier

  Sched sch(n, cores, ncommthreads);
  // comm threads do MPI then fall through to steal
  mic_combuf_exchange();
  // other threads immediately enter compute loop
  while ((block = sch.nextiter()) != -1) {
    // compute interior iterations
  }
  #pragma omp barrier

  // apply face contributions from other nodes

} // end parallel
```

Communications Pseudo-code

```
mic_combuf_exchange() {
  int tid = omp_get_thread_num();
  if (tid >= ncommthreads) return;
  // Divide the faces among the comm threads
  for (int i = tid; i < nfaces; i += ncommthreads) {
   MPI_Irecv(rbuffers[i], sizes[i], MPI_BYTE,
     dest_rank, tag, comms[tid], &req1)
   MPI_Isend(sbuffers[i], sizes[i], MPI_BYTE,
     src_rank, tag, comms[tid], &req2);
   MPI_Wait(req1, MPI_STATUS_IGNORE);
   MPI_Wait(req2, MPI_STATUS_IGNORE);
  }
// return to fall through to steal any remaining
// interior iterations
}
```

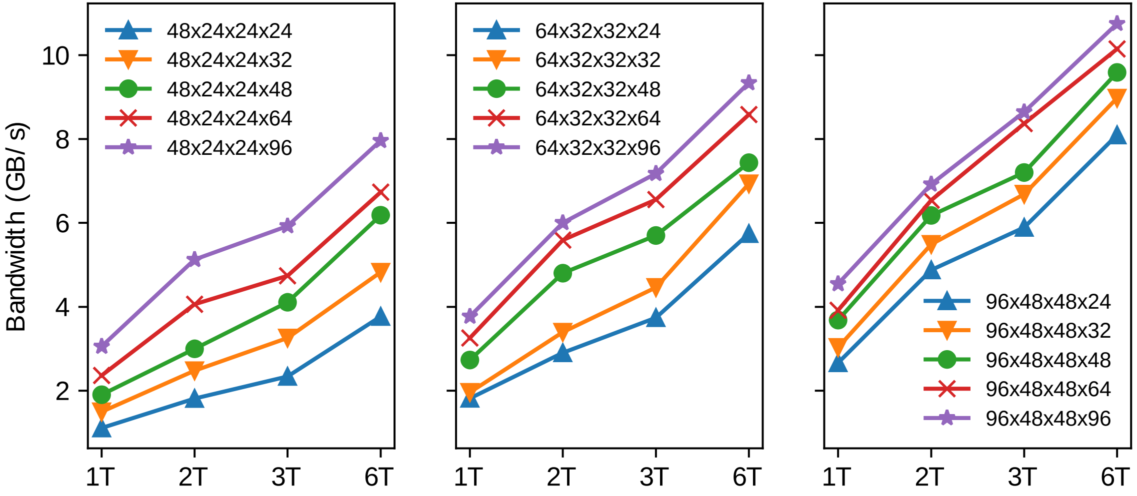
`comms[tid]` is a separate communicator for each comms thread, created by `MPI_Comm_dup`

Performance: Overlap of MPI with Compute (8 node, 1 rank/node, 64 OMP on Oakforest-PACS)



- X axis is number of comm threads, Y axis is total time for interior + (overlapped) MPI
- # comm threads must divide number of distributed faces for load balance
- We continue to see speedup as comm threads are added until the compute time dominates

Performance: Network BW vs. #comm threads (8 node, 1 rank/node, 64 OMP on Oakforest-PACS)



- Bandwidth increases with total transfer size
- Bandwidth increases with number of comm threads
- Best bandwidth is 10.75GB/sec, 86% of peak Intel® Omni-path Architecture undirectional BW of 12.5 GB/sec

Conclusions and Future Work

Static workstealing scheduler + Multi-EP allows multiple MPI threads per MPI rank with minimal changes to existing code

For CCS-QCD, a single rank per node is optimal because it avoids over-decomposition, thus reducing surface to volume ratio

We are still not achieving full OPA bandwidth, further investigation is needed

Similar results were obtained by Peter Boyle in the Grid implementation of Lattice QCD

Legal Disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Atom, Xeon, Xeon Phi, 3D Xpoint, Iris Pro and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

Legal Disclaimers

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 SSE3 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

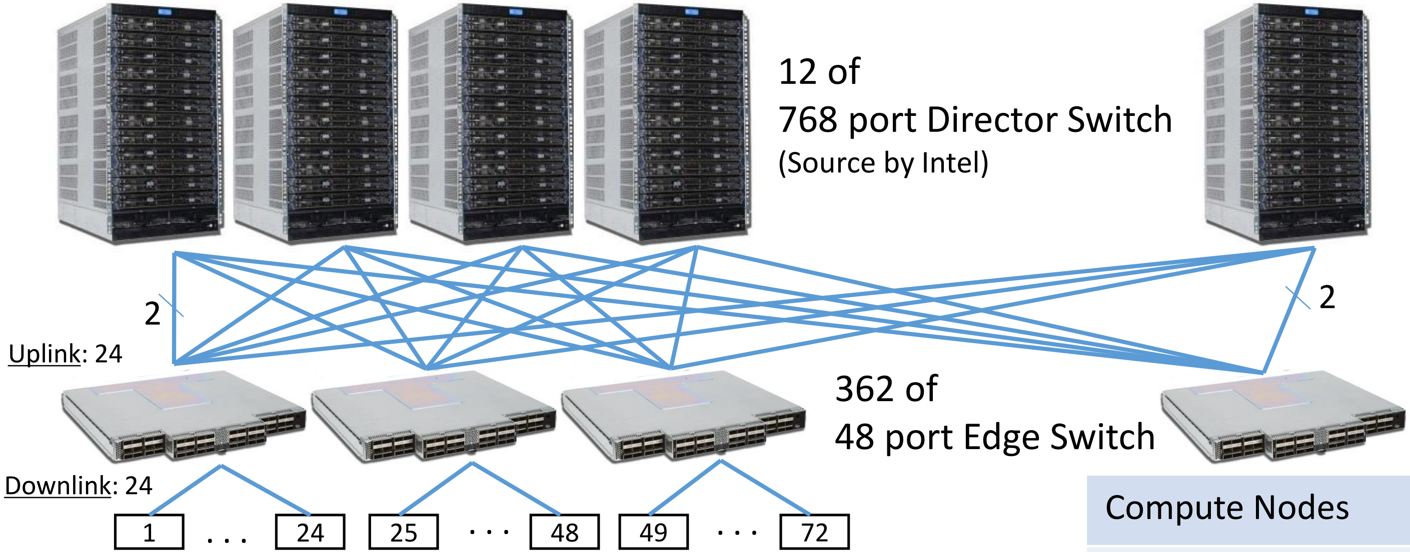
[Pre-Patch Disclaimer] Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

Specification of Oakforest-PACS system

Total peak performance			25 PFLOPS
Total number of compute nodes			8,208
Compute node	Product		Fujitsu PRIMERGY CX600 M1 (2U) + CX1640 M1 x 8node
	Processor		Intel® Xeon Phi™ 7250 (Code name: Knights Landing), 68 cores, 1.4 GHz
	Memory	High BW	16 GB, 490 GB/sec (MCDRAM, effective rate)
		Low BW	96 GB, 115.2 GB/sec (peak rate)
Interconnect	Product		Intel® Omni-Path Architecture
	Link speed		100 Gbps
	Topology		Fat-tree with (completely) full-bisection bandwidth

Slide courtesy of Prof. Toshihiro Hanawa and Prof. Taisuke Boku

Full bisection bandwidth Fat-tree by Intel® Omni-Path Architecture



Firstly, to reduce switches&cables, we considered :
- All the nodes into subgroups are connected with FBB Fat-tree
- Subgroups are connected with each other with >20% of FBB

But, HW quantity is not so different from globally FBB, and globally FBB is preferred for flexible job management.

Compute Nodes	8208
Login Nodes	20
Parallel FS	64
IME	300
Mgmt, etc.	8
Total	**8600**

Specification of Oakforest-PACS system (Cont'd)

Parallel File System	Type	Lustre File System
	Total Capacity	26.2 PB
	Product	DataDirect Networks ES14K
	Aggregate BW	500 GB/sec (50 GB/sec x 10 OSS)
	Metadata	MDS x 12, MDT x 3, 3 DNE (Distributed Namespace)
File Cache System	Type	Burst Buffer, Infinite Memory Engine (by DDN)
	Total capacity	940 TB (NVMe SSD, including parity data by erasure coding)
	Product	DataDirect Networks IME14K
	Aggregate BW	1,560 GB/sec (with 25 x2 IME servers)
Power consumption		4.2 MW (including cooling)
# of racks		102