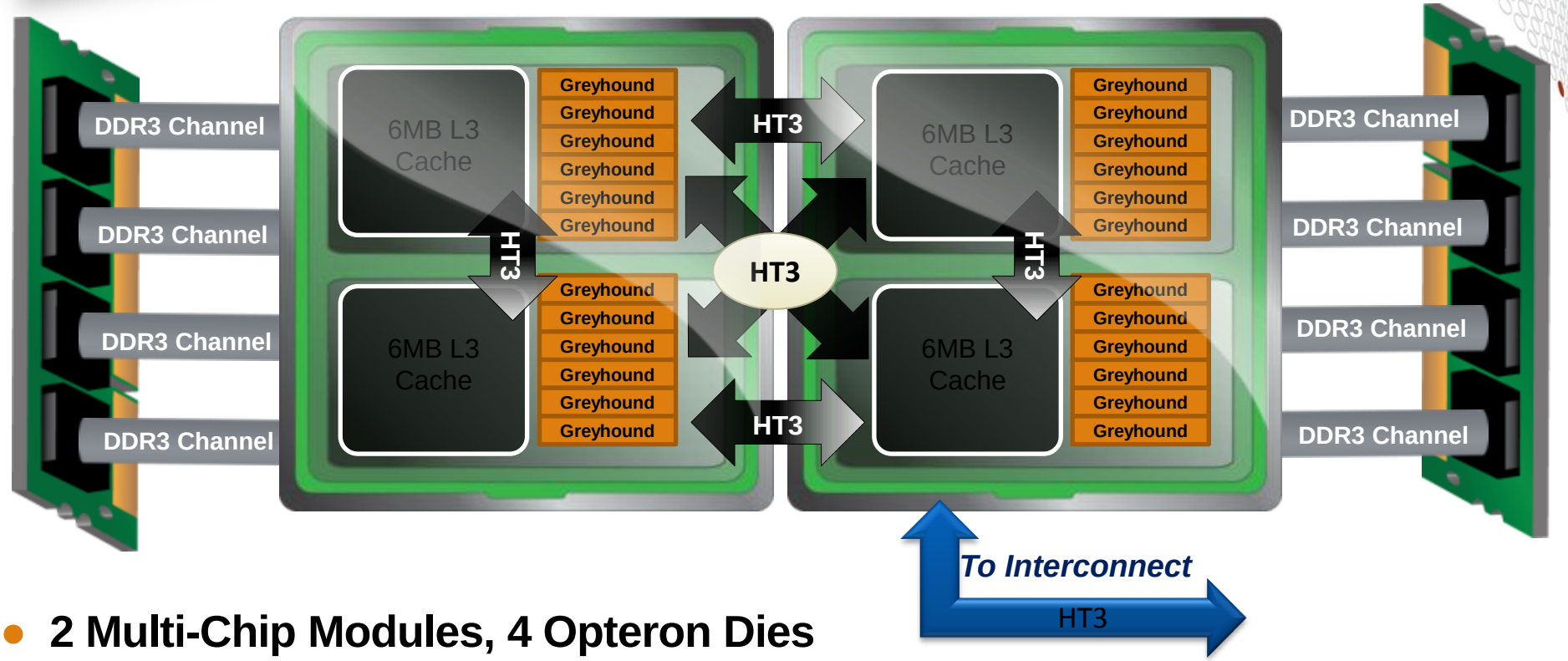
XC comparison to XE and early performance tips

Outline

- Node is similar but different
- Choice of hyperthreading is new
- Intel OpenMP on XC
- Environment is very similar
- Couple of MPI enhancements
- Hyperthreading optimization chart
- Network bandwidth, in particular global bandwidth, has improved substantially
- Brush up on your IO

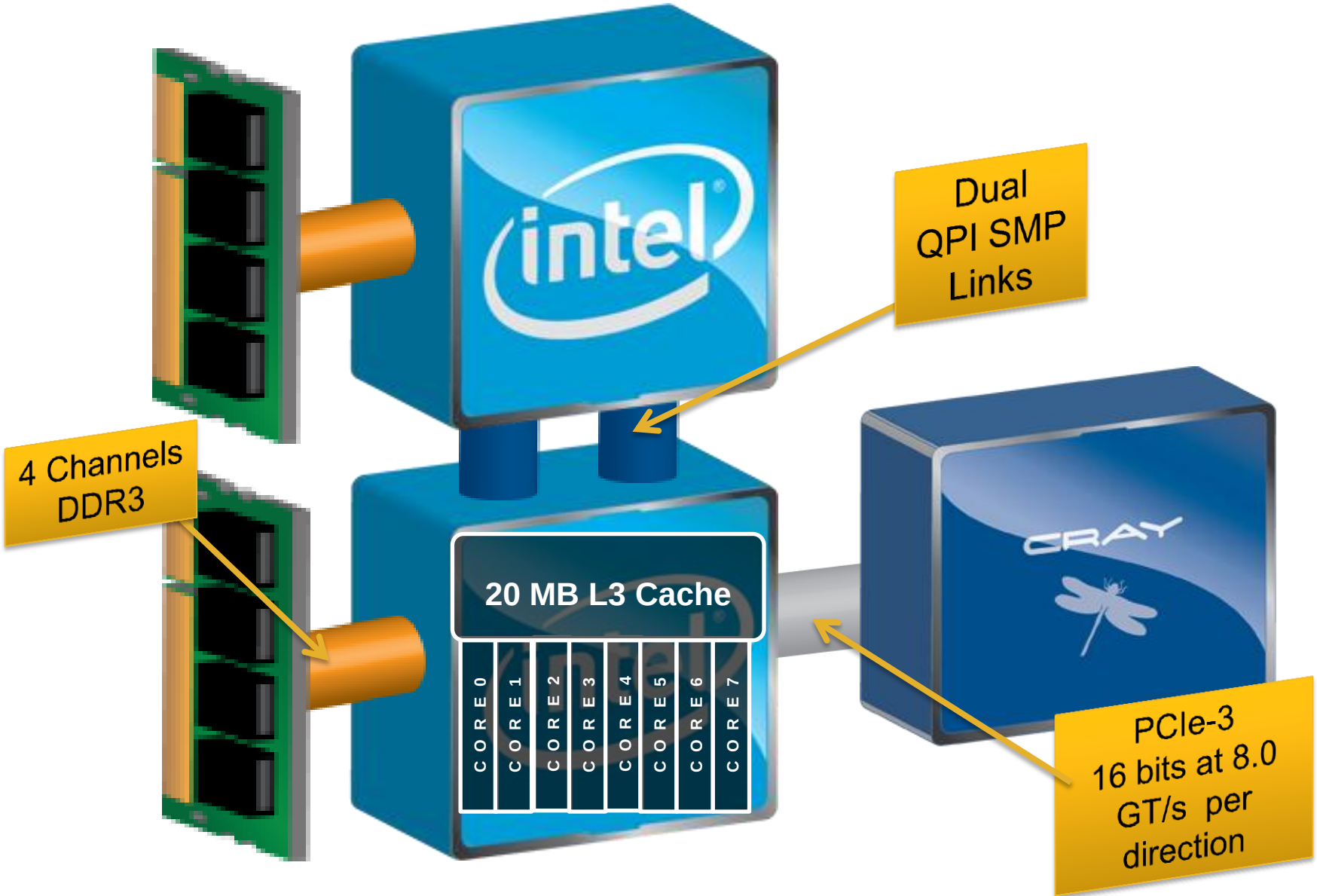
- 3D FFTS… if we have time

XE6 Compute Node Details: 24-core Magny Cours



- 2 Multi-Chip Modules, 4 Opteron Dies
- 24 (or 16) Computational Cores, 24 MB of L3 cache
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- Dies are fully connected with HT3

Cray XC30 Compute Blade Architecture



Magny Cours vs Sandybridge: bake-off

MAGNY COURS

- 6 cores per die
  - 4 die per node
- Each core has
  - 1 user thread
  - 1 SSE (vector) functional group
    - 128 bits wide
    - 1 add and 1 multiply
  - L1 cache size = 32 Kbytes
  - L2 cache size = .5 Mbytes
- L3 cache, size = 6 Mbytes
- Cache per core = .5 + 6/6 = 1.5 Mbytes
- Cache BW per core
  - L1 / L2 / L3 = 35 / 3.2 / 3.2 Gbytes/s
- Stream TRIAD BW/node = 52 Gbytes/s
- Clock speed = 2.1 Ghz
- Peak DP FP per core = 4 flops/clk
- Peak DP FP per node = 201 flops/clk
- Memory latency = 110 ns

Sandybridge

- 8 cores per die
  - 2 die per node
- Each core has
  - <u>1 or 2 user threads</u>
  - 1 SSE (vector) functional group
    - 256 bits wide
    - 1 add and 1 multiply
  - L1 cache size = 32 Kbytes
  - L2 cache size = 256 kbytes
- L3 cache, size = 20 Mbytes
- Cache per core= 20/8 = 2.5 Mbytes
- Cache BW per core
  - L1 / L2 / L3 = 105 / 42 / 26 Gbytes/s
- Stream TRIAD BW / Node = 77 Gbytes/s
- Clock speed = 2.6 Ghz
- Peak DP FP per core = 8 flops/clk
- Peak DP FP per node = 332 flops/clk
- Memory latency = 82 ns

Single Stream vs Dual Stream

- **Cray compute nodes booted with hyperthreads always ON**
- **User can choose to run with one or two ranks/pes/threads per core**
- **Choice made at runtime**

- **aprun –n### -j1 … -> Single Stream mode, one rank per core**
- **aprun –n### -j2 … -> Dual Stream mode, two ranks per core**

- **Default is Single Stream**
- **Dual Stream often better**
  - if throughput is more important OR…
  - If performance per node is more important OR…
  - if you code scales extremely well
- **Single Stream often better**
  - Single job performance matters more
  - Per core performance matters most (code does not scale well)

- **Cray ended up running 4 or the 7 "SSP" codes in dual stream mode to maximize overall system score**

Running with OpenMP and the Intel PE

- **An extra thread created by the Intel OpenMP runtime interacts with the CLE thread binding mechanism and causes poor performance**

- **To work around this issue cpu-binding should be turned off**
  - Allows user compute threads to spread out over available resources
  - Helper thread will no longer impact performance

- **Note: This is only an issue for running OpenMP programs that were compiled and linked with the Intel compiler**

Examples of using MPI and OpenMP with Intel PE

- **Running when "depth" divides evenly into the number of "cpus" on a socket**

  export OMP_NUM_THREADS="<=depth"

  aprun -n npes -d "depth" -cc numa_node a.out

- **Running when "depth" does not divide evenly into the number of "cpus" on a socket**

  export OMP_NUM_THREADS="<=depth"

  aprun -n npes -d "depth" -cc none a.out

- **When running default # of cpus = # of cores**
- **When running using –j2 # of cpus = # of cores X 2 hyperthreads**
- **These "-cc" options turn off cpu binding**
  - Your process/thread may switch cores in the middle of execution

Cray Software: XE6 – Cascade Continuity

	Cray XE6 Platform	Cray Cascade Platform
Cray Programming Environment	Cray MPT (MPI and SHMEM)	
	Cray Compiler (incl. Fortran/UPC/CAF/Chapel)	
	Cray Tools (CrayPat, Cray Apprentice2, Cascade Debugging Support)	
	Cray Math and Science Libraries	
	Third Party Libraries (ACML, MKL, IO, Math and Science)	
	Third Party (Python, Totalview, DDT)	
	PGI/GNU/Intel Compilers	GNU/Intel Compilers
Cray Linux Environment	MOAB/LSF/PBS Pro	
	ALPS	
	Lustre/DVS/NFS	
	CLE	
	SuSE components	
		Intel Processors Components
	Gemini Networks	Gemini API's on Aries Network
	Cray System Mgmt	
	HSS Framework	

Cray Developed

Other Party Developed

MPI Features / Functionality for XC

- MPI on XC behaves essentially the same as MPI on XE
  - UGNI interface is the same for XE and XC
  - MPICH2 code base is nearly the same
    - Messaging Paths (VSHORT, EAGER, RENDEZVOUS) are Identical
- Enhanced Features for XC
  - Modified MPI Asynchronous Progress Engine Threads
    - Threads can be placed on unused Intel hyper thread cores
    - More on this in a moment…
  - XC Hardware Collective Engine (CE)
    - XC supports hardware-offload of Barrier & Allreduce collectives
    - Plan to invoke these via MPICH_USE_DMAPP_COLL env variable
    - Must also link libdmapp into your application

MPI - Async Progress Engine Support

- Used to improve communication/computation overlap
  - Each MPI rank starts a “helper thread” during MPI_Init
- Helper threads progress MPI engine while application computes
- Only inter-node messages that use Rendezvous Path are progressed (relies on BTE for data motion)
- To enable on XC when using 1 stream per core:
  - export MPICH_NEMESIS_ASYNC_PROGRESS=1
  - export MPICH_MAX_THREAD_SAFETY=multiple
  - export MPICH_GNI_USE_UNASSIGNED_CPUS=enabled
  - Run application: aprun –n XX a.out
- To enable on XC when using 2 streams per core recommend running with the corespec option:
  - export MPICH_NEMESIS_ASYNC_PROGRESS=1
  - export MPICH_MAX_THREAD_SAFETY=multiple
  - Run application with corespec: aprun –n XX -r [1-2] a.out
- 10% or more performance improvements with some apps

Hyperthreading optimization chart

Single Stream Mode – No MPI Async:

Collect performance baseline here

Maximize per cpu performance

Little to no MPI communication overlap for medium size messages

Dual Stream Mode – Without MPI Async

Goals:
Optimizing per node performance
or
Maximizing performance by using many PEs

Is this "better"?

Single Stream Mode – With MPI Async

Goals:

Maximize per cpu performance

Improve communication performance

Does overall performance improve?

Dual Stream Mode – With MPI Async

Goals:
Optimizing per node perf. or
Maximizing perf. using many Pes
and
Improve communication performance…

But give up using 1 or 2 hyperthreads

Is this "better"?

Significant Improvement in Network Bandwidth

- **There has been a significant improvement in global bandwidth moving from the XE to the XC**
  - Phase 2 Edison will likely have ~4-5 times global bandwidth of Hopper if running on the full system
  - Group structure and adaptive routine will likely mean the improvement will be even greater for jobs that take 100+ nodes
- **Adaptive routing should prevent hotspots in the network**
  - Relatively easy to generate hot spots in hopper
  - Higher dimensional nearest-neighbor patterns see contention in the 3D torus network in Hopper
  - Edison should be able to handle these patterns much better
- **Global bandwidth intensive and higher dimensional nearest-neighbor codes should benefit from dragonfly**

Linking with MKL and PrgEnv-cray

- **PrgEnv-cray compatible with sequential, not threaded, MKL**
- **Examples assume you have loaded the intel module (to define the env var INTEL_PATH)**
  - Typical case: You want to use MKL BLAS and/or LAPACK

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
-Wl,--end-group
```

  - Another typical case: You want to use MKL serial FFTs/DFTs

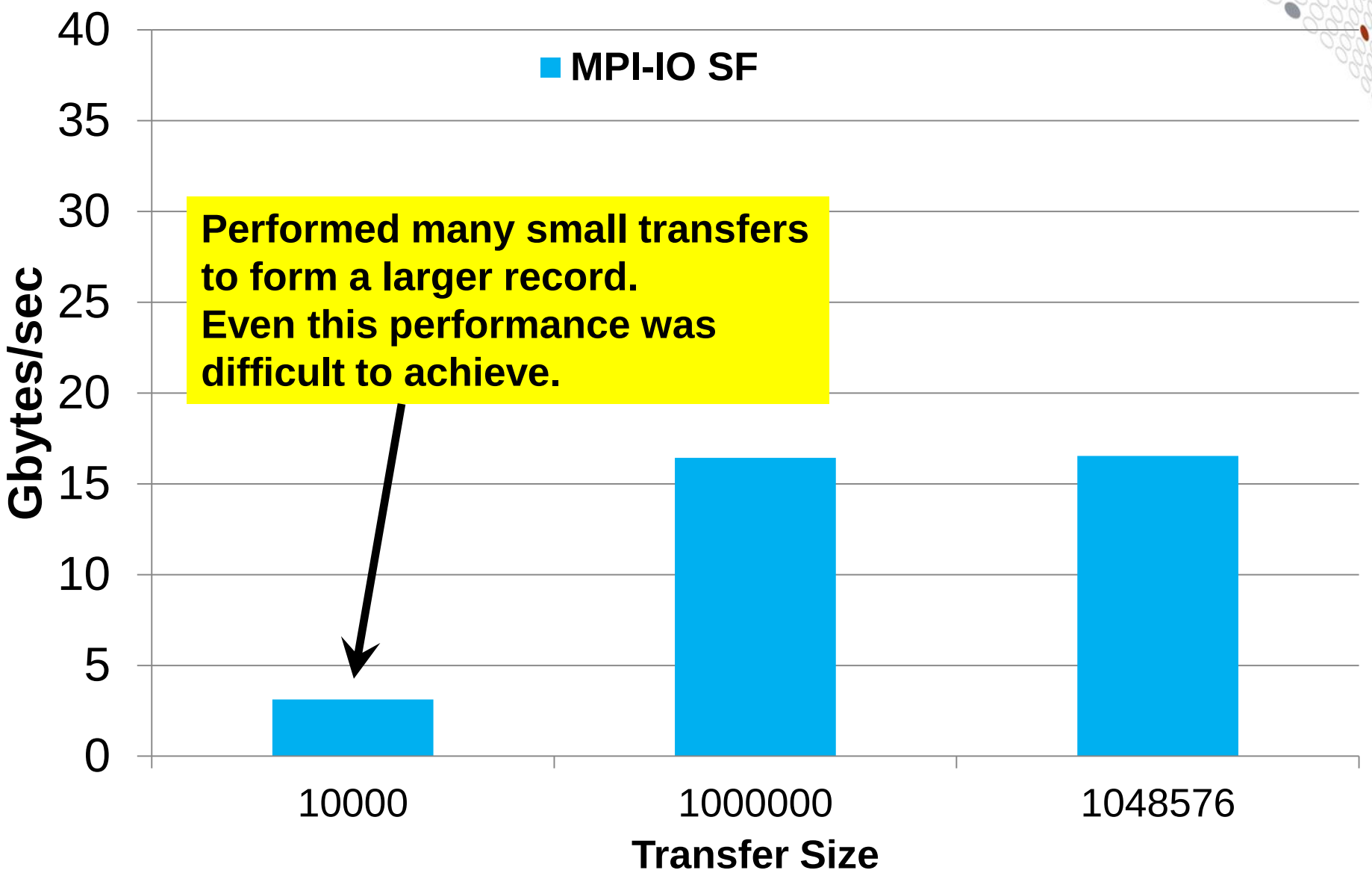
Same as above (need more for FFTW interface)

  - A less typical case: You want to use MKL distributed FFTs

```
-L ${INTEL_PATH}/mkl/lib/intel64/ \
-Wl,--start-group \
-lmkl_cdft_core -lmkl_intel_lp64 -lmkl_sequential \
-lmkl_core -lmkl_blacs_intelmpi_lp64 \
-Wl,--end-group
```

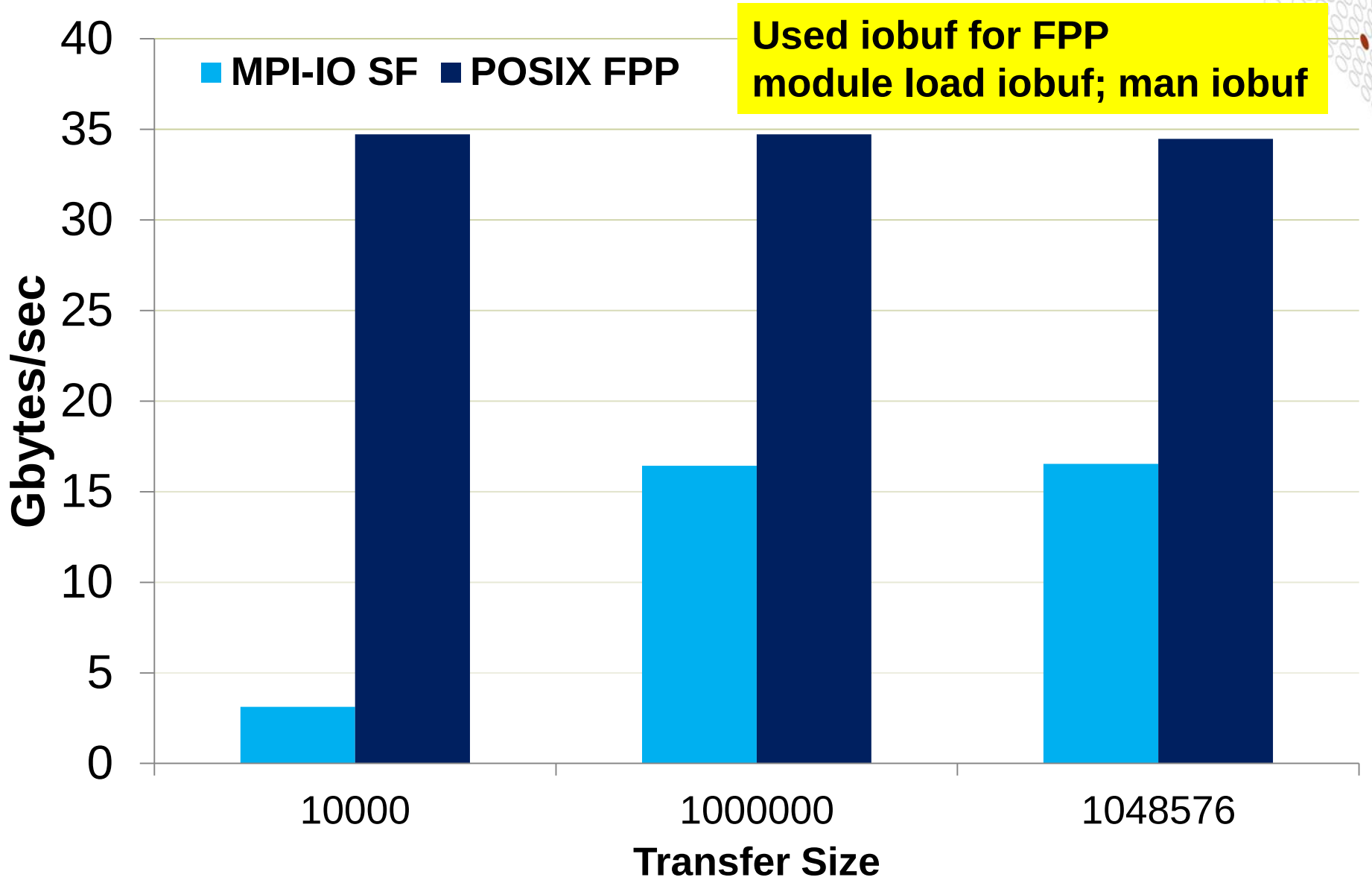
- **The Intel MKL Link Line Advisor can tell you what to add to your link line**
  - http://software.intel.com/sites/products/mkl/

Write Transfer Speeds for Sequential IO Patterns



Performed many small transfers to form a larger record. Even this performance was difficult to achieve.

Write Transfer Speeds for Sequential IO Patterns



IO performance summary

- **File per process performance can benefit from using IOBUF, especially for small record sizes.**
- **Shared file performance that is 40-50% of file per process performance is possible, depending on the file access pattern.**
  - During shared file IO the file system must do file locking to preserve consistency semantics.
- **Shared file performance may or may not benefit from MPI I/O collective buffering, depending on the file access pattern.**
  - See the Cray manual "Getting Started on MPI I/O" (S-2490), and in particular, section 5.2, for some simple write examples.
- **To get a summary of the file access pattern:**

  export MPICH_MPIIO_STATS=1

  - For apps with MPI I/O calls using module cray-mpich2/5.6.0 (or later)
- **In general, large records with no gaps performs best, small records with large gaps performs worst.**
  - And collective buffering helps most on writes for small records by many processes to a single region of a shared file.

3D FFT case study

3D FFT Story: 1D decomposition

- Customer had a code whose main computational section was a 3D FFT
- "Original" version had a 1D decomposition; series of planes
  - 3 compute sections
  - 1 communication section of a large all-to-all across all PEs in the job
    - Communication used shmem
- Two major problems
  - Limited parallelism: Real problems <=10k in 1 dimension
  - As #PEs increased, message size decreased
    - More of a problem on XT (Seastar) than XE (Gemini)

3D FFT Story: 2D decomposition

- "New" version had a 2D decomposition; pencils
  - 3 compute sections
  - 2 communication sections of a large all-to-all across subsets of PEs in the job
- New version was not running "as well as he would like"

WHY?

3D FFT Story: 2D decomposition

- Reason 1: Moving twice as much data
  - Two sections that picked up and set down “all the data”
  - At scale communication accounted for the vast majority of the time
  - Communicating twice was taking twice as long

- What can we do about it?

- What about the decomposition?

3D FFT Story: 2D decomposition

- Initial decomposition was 100 x 100 or 200 x 50
  - First number is the number of “logically contiguous” PEs which do the first all-to-all
  - Second number is the number of simultaneous all-to-alls done between groups
- Second all-to-all was global bandwidth bound
  - See previous slides for optimization techniques
- First all-to-all still put significant amounts of data onto the network.

3D FFT Story: 2D decomposition

- What if we reduced the size of the first dimension?
  - Smaller size kept higher % of data on node
  - Smaller size meant fewer, “likely closer” nodes were communicating during that phase
  - 24 x M decomposition: communication for the 1st all-to-all is entirely on node
    - Time associated with the first all-to-all essentially disappeared

3D FFT Story: 2D decomposition

- Reason 2: Using too few cores
  - Initial comparisons where made on < 4k cores
  - 1D version was performing just fine
    - Had no parallelism limitations
    - Smaller messages were running fine on XE
  - 2D version “designed” to run on more core
    - Designed to scale to core counts that 1D could not achieve