

# Cray XT6 at HECToR

Kevin Roy, Jason Beech-Brandt, and Tom Edwards

---

13 Oct 2010

Manchester

## Activities

- Phase 2b – Cray XT6
  - Microprocessor
  - Interconnect
- Programming Environment
  
- Optimization Techniques
  - OpenMP
  - Environment Variables
  - Placement
  - Core Specialization
  - ...others

# Cray XT6

---

## Cray Systems at HECToR



- Cray XT4 Compute Nodes
- 3072 x 2.3 GHz Quad Core Opteron Nodes
- 12,238 Cores
- 24TB Memory
- 8GB per node – 2GB per Core

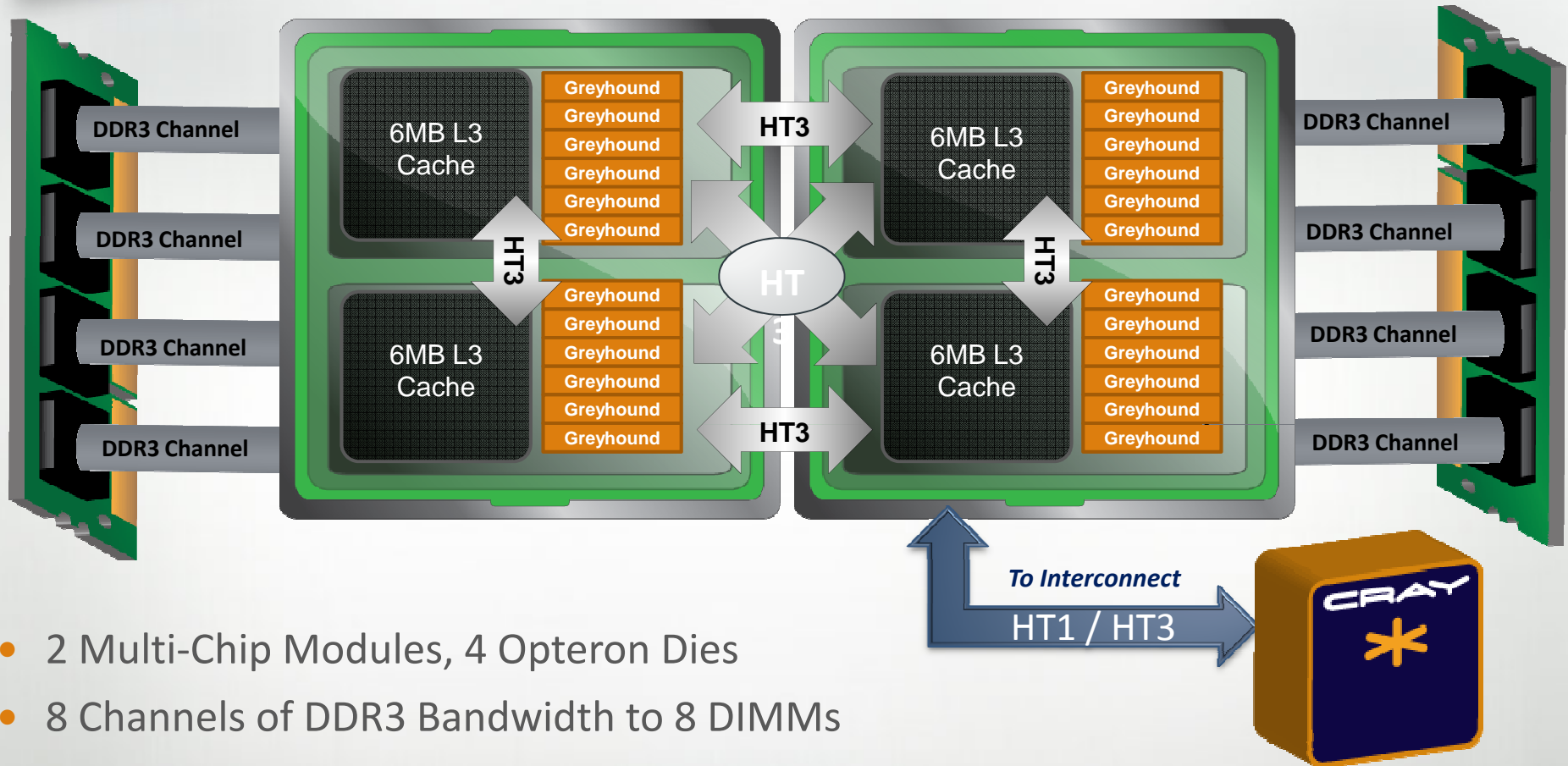
- Cray X2 Compute Nodes
- 28 x 4 Cray X2 Compute nodes
- 32 GB per node 8GB per Core



- Cray XT6 Compute Nodes
- 1856 x 2.1 GHz Dual 12 Core Opteron Nodes
- 44,544 Cores
- 59TB Memory
- 32GB per node – 1.33 GB per Core



# XT6 Node Details: 24-core Magny Cours



- 2 Multi-Chip Modules, 4 Opteron Dies
- 8 Channels of DDR3 Bandwidth to 8 DIMMs
- 24 (or 16) Computational Cores, 24 MB of L3 cache
- Dies are fully connected with HT3
- Snoop Filter Feature Allows 4 Die SMP to scale well

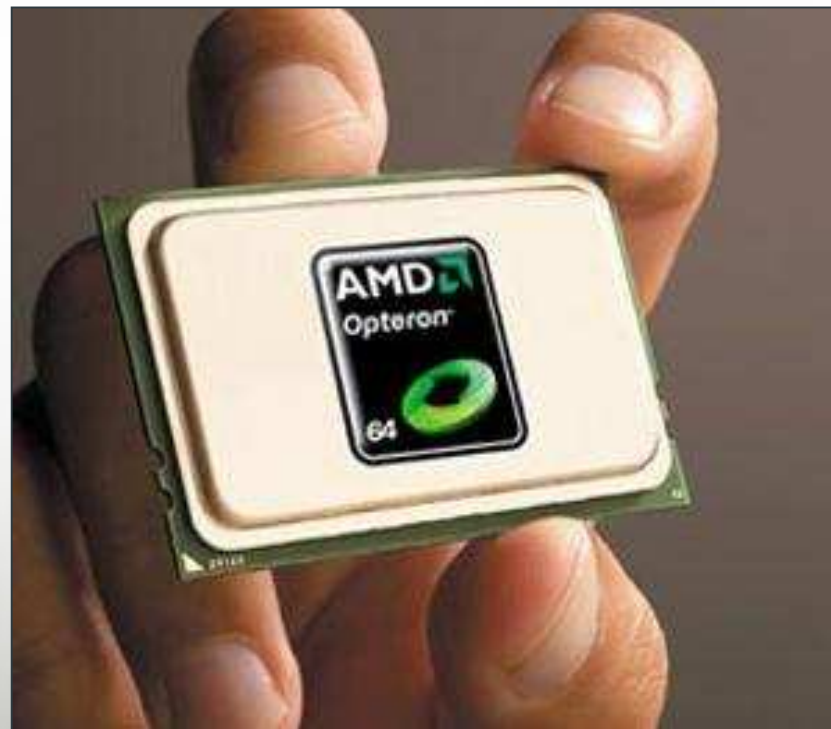


# x86 64-bit Architecture Evolution

	2003	2005	2007	2008	2009	2010
	<b>AMD Opteron™</b>	<b>AMD Opteron™</b>	<b>"Barcelona"</b>	<b>"Shanghai"</b>	<b>"Istanbul"</b>	<b>"Magny-Cours"</b>
<b>Mfg. Process</b>	130nm SOI	90nm SOI	65nm SOI	45nm SOI	45nm SOI	45nm SOI
<b>CPU Core</b>	K8 	K8 	Greyhound 	Greyhound+ 	Greyhound+ 	Greyhound+ 
<b>L2/L3</b>	1MB/0	1MB/0	512kB/2MB	512kB/6MB	512kB/6MB	512kB/12MB
<b>Hyper Transport™ Technology</b>	3x 1.6GT/.s	3x 1.6GT/.s	3x 2GT/s	3x 4.0GT/s	3x 4.8GT/s	4x 6.4GT/s
<b>Memory</b>	2x DDR1 300	2x DDR1 400	2x DDR2 667	2x DDR2 800	2x DDR2 800	4x DDR3 1333

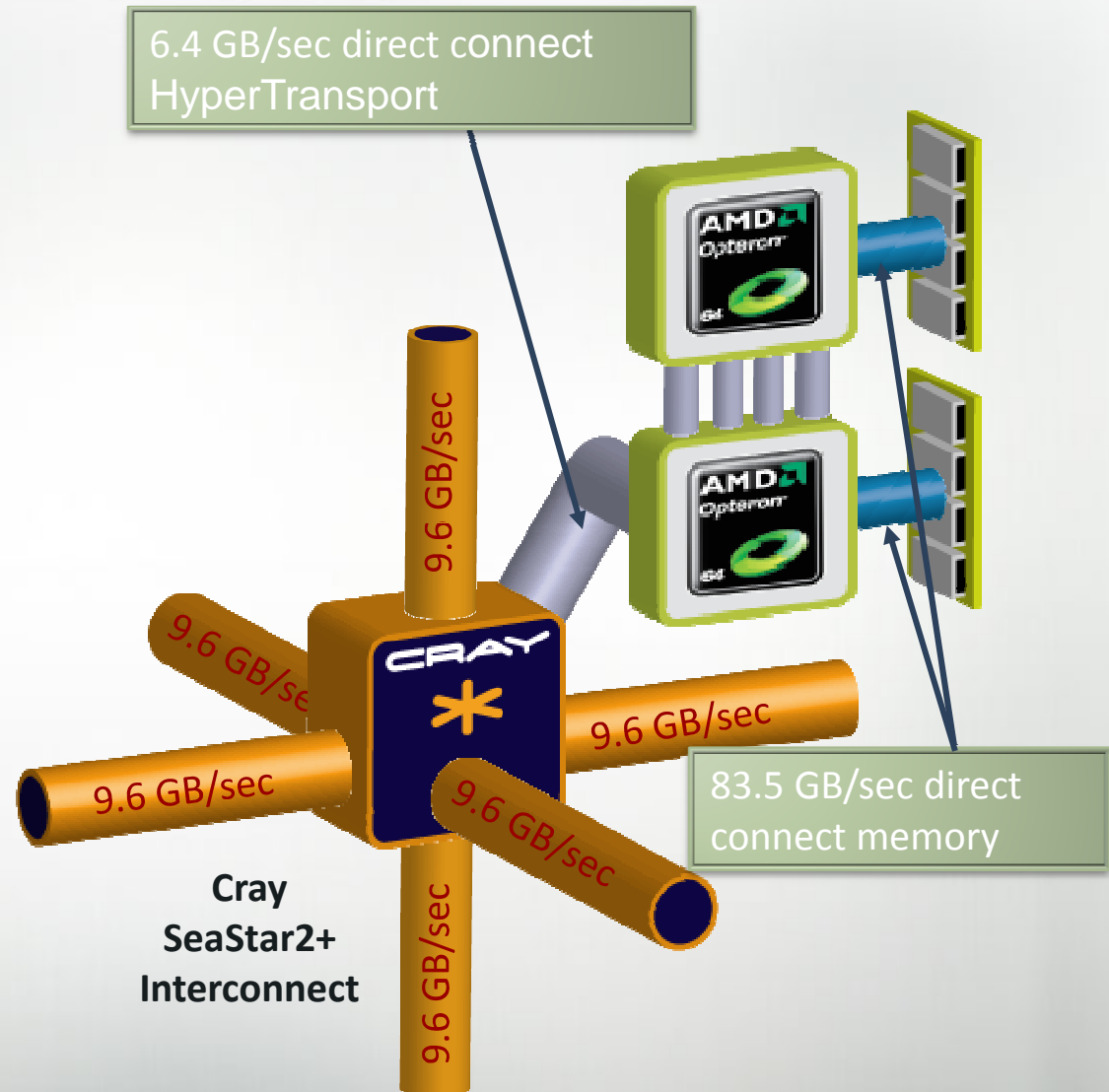
# XT Processors Properties

Processor	Cores	Frequency	Peak (Gflops)	Bandwidth (GB/sec)	Balance (bytes/flop)
Barcelona (XT4)	4	2.3	36.8	12.8	0.34
MC-12 (XT6)	12	2.1	100.8	42.6	0.42



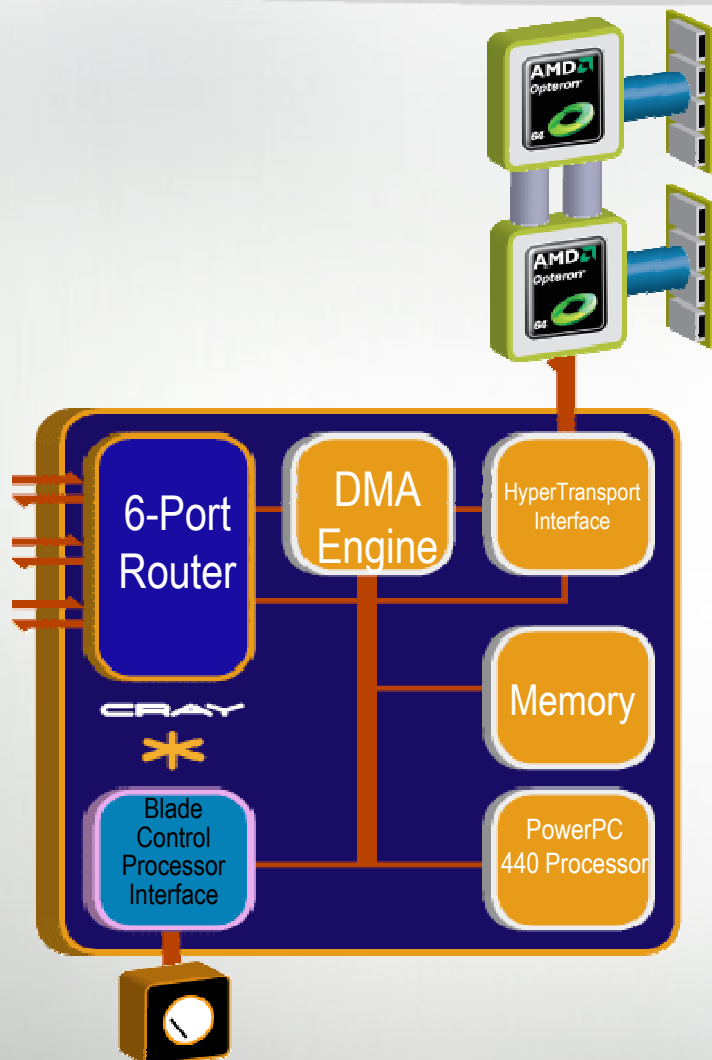
# Cray XT6 Node

Characteristics	
Number of Cores	24
Peak Performance MC-12 (2.2)	211 Gflops/sec
Memory Size	32GB per node
Memory Bandwidth	83.5 GB/sec



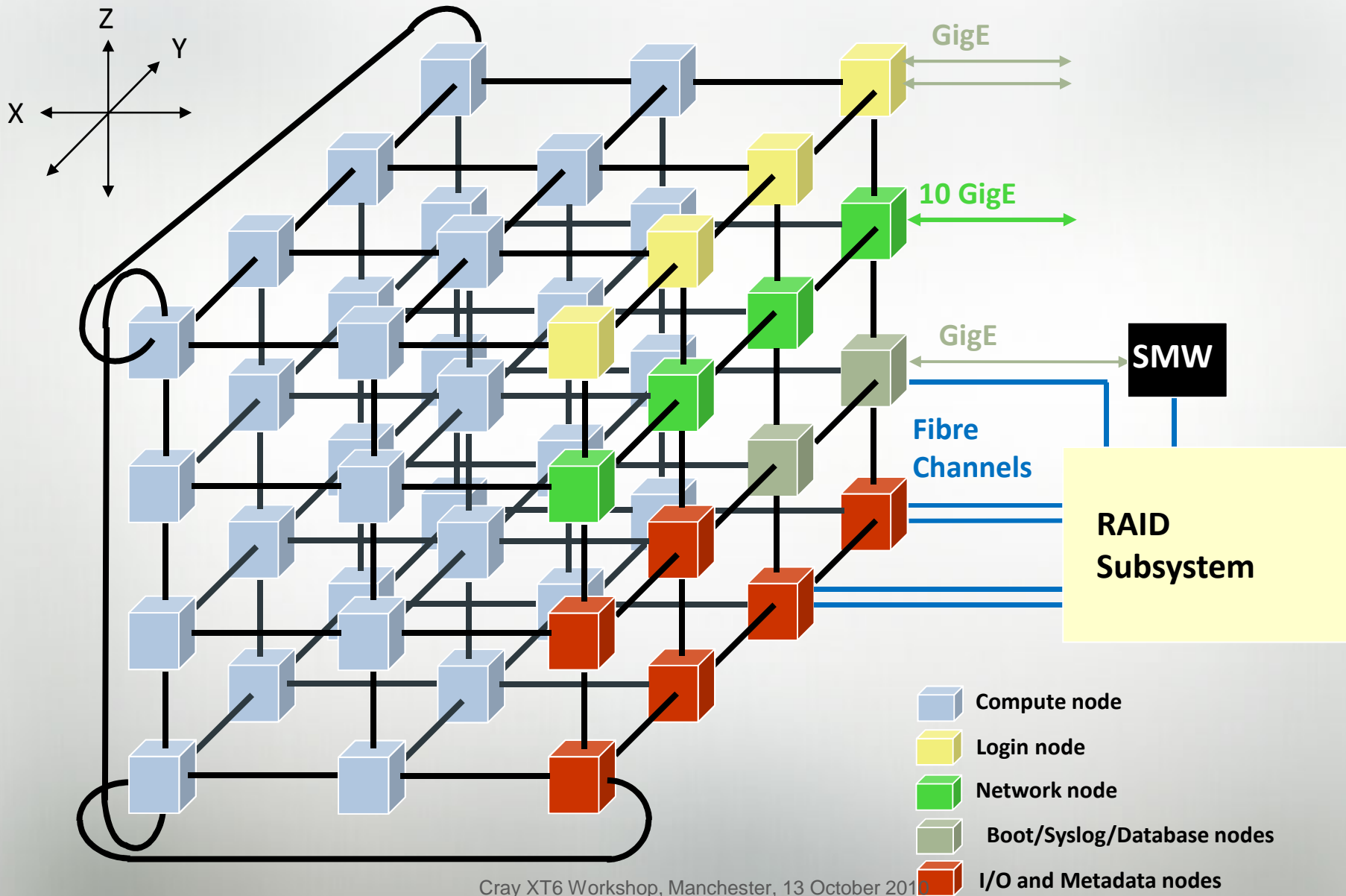


# Cray SeaStar2+ Interconnect



- Cray XT systems ship with the SeaStar2+ interconnect
- Custom ASIC
- Integrated NIC / Router
- MPI offload engine
- Connectionless Protocol
- Link Level Reliability
- Proven scalability to 225,000 cores

# XT System Configuration Example



# Programming Environment

## Modules

- The use of the compiler drivers and the xtpe-mc12 module will allow access to the fastest libraries.
- In particular this affects
  - Compiler itself
  - Libsci
- In general try the latest versions of software for improved stability and performance

**module load xtpe-mc12**

## Performance Tools

- The current performance toolset is available now under perftools.
  - `perftools` should be loaded rather than CrayPAT and PAPI
- Invocation and instrumentation behaves much as before.

## Libsci - Cray Scientific Library

- Libsci provides cray optimised versions of common libraries
  - BLAS – Basic Linear Algebra Subroutines
  - LAPACK – Linear Algebra Routines
  - ScaLAPACK – Parallel Linear Algebra Routines
  - BLACS – Basic Liner Algebra Communication Subprograms
  - IRT – Iterative Refinement Toolkit
  - SuperLU – Sparse Solver Routines
  - CRAFFT – Cray Adaptive Fast Fourier Transform
- `module load xt-libsci`
- Load appropriate Opteron module
  - `xtpe-barcelona, xtpe-mc12`



## Libsci - Threaded

- Libsci uses OpenMP threads as of `xt-libsci/10.4.2` and `asynce/3.9`
  - Can still be used for serial applications as default `OMP_NUM_THREADS=1`
  - Can be used inside parallel regions without spawning additional threads
  - Requires the appropriate `xtpe-<barcelona|mc12>` module to be loaded

## Cray Compiling Environment

- As part of the XT6 installation CCE was made available on all systems
- Latest version is 7.2.7
- CCE can provide excellent on node performance compared to other compilers.
- CCE provides one of the most standard coherent compiling environments available.
- Provides the ability to do PGAS languages.
  - Very important as we look forward to the Gemini upgrade which will provide hardware support for PGAS

## Cray Compiler Environment (CCE): Brief History of Time

- Cray has a long tradition of high performance compilers
  - Vectorization
  - Parallelization
  - Code transformation
  - More...
- Began internal investigation leveraging an open source compiler called LLVM
- Initial results and progress better than expected
- Decided to move forward with Cray X86 compiler
- Initial x86 release, 7.0, in December 2008
- 7.2.7 is the latest release (many improvements to PGAS support)

## Cray Opteron Compiler: Current Capabilities

- Fortran, C and C++ support
- Excellent Vectorization
  - Vectorize more loops than other compilers
- OpenMP - 3.0 standard
- PGAS: Functional UPC and CAF available today.
- Excellent Cache optimizations
  - Automatic Blocking
  - Automatic Management of what stays in cache
- Prefetching, Interchange, Fusion, and much more...

## Cray Opteron Compiler: Future Capabilities

- Enhanced C++ Front end
- More Automatic Parallelization
  - Modernized version of Cray X1 streaming capability
  - Interacts with OMP directives
- Optimized PGAS for Gemini
- Improved Vectorization
- Improve Cache optimizations

## Cray compiler flags

- Module available  
**module swap PrgEnv-pgi PrgEnv-cray**
- **Overall Options**
  - -ra creates a listing file with optimization info
- **Preprocessor Options**
  - -eZ runs the preprocessor on Fortran files
  - -F enables macro expansion throughout the source file
- **Optimisation Options**
  - -O2 optimal flags [ enabled by default ]
  - -O3 aggressive optimization
  - -O ipa<n> inlining, n=0-5



## Cray compiler flags

- **Language Options**

- -f free                    process Fortran source using free form specifications
- -s real64                treat REAL variables as 64-bit
- -s integer64            treat INTEGER variables as 64-bit
- -hbyteswapio            big-endian files in Fortran; XT4 is little endian  
this is a link time option

- **Parallelization Options**

- -O omp                    Recognize OpenMP directives [ enabled by default ]
- -O thread<n>            n=0-3, aggressive parallelization, default n=2

**man pages: [crayftn](#), [intro\\_directives](#), [intro\\_pragmas](#), [assign](#)**

[http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-3901-71;idx=books\\_search;this\\_sort=;q=3901;type=books;title=Cray%20Fortran%20Reference%20Manual](http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-3901-71;idx=books_search;this_sort=;q=3901;type=books;title=Cray%20Fortran%20Reference%20Manual)

## Cray programming environment: explain

- Displays the explanation for an error message: compile, run time
- Compiler error

```
ftn-1615 crayftn: WARNING TEST, File = fail.F90, Line = 24, Column = 16
```

```
Procedure "ADD" is defined at line 1 (fail.F90). Dummy argument "B" is an array argument.
This argument is scalar.
```

```
>explain ftn-1615
```

```
Warning : Procedure "%s" is defined at line %s. Dummy argument "%s" is an
array argument. This argument is scalar.
```

The scope of a global name is the entire compilation, so a global (or external) name must be defined and referenced consistently throughout the compilation. The compiler has found that a reference or definition of this global name differs with another reference or definition for this name.

The compiler is comparing two definitions or a definition and a reference to the listed procedure. If the compiler is comparing two definitions, then the compiler has found that in one definition, a dummy argument is an array argument, but the corresponding dummy argument in the second definition is a scalar argument. The arguments must be the same. If the compiler is comparing a reference with its definition, then it has found an array dummy argument associated with a scalar actual argument or vice versa. Again, the arguments must either both be scalar or both be arrays. (Note: In a reference, an array element is considered a scalar.)

## Cray programming environment: explain

- I/O runtime error

Open IOSTAT=5016

```
rosa> explain lib-5016
```

An EOF or EOD has been encountered unexpectedly.

The file terminated unexpectedly, perhaps without the proper end-of-file record or end-of-data control information. The file specification may be incorrect, or the file may be corrupted.

Ensure that the file specification is correct. If the file is corrupted, create the data again, if possible.

See the man pages for assign(1) and asgcmd(1).

The error class is UNRECOVERABLE (issued by the run time library).

# Optimization Techniques

# 13. Load the proper xtpc-<arch>

And try every available compiler!

## What is xtpe-arch

```
: ) module show xtpe-istanbul
```

```
-----  
/opt/cray/xtpe-arch/xtpe-istanbul:
```

**It'd probably be a really bad idea to load two architectures at once.**

```
conflict
```

```
conflict
```

```
xtpe-quadcore
```

```
conflict
```

```
xtpe-shanghai
```

```
conflict
```

```
xtpe-mc8
```

```
conflict
```

```
xtpe-mc1
```

**I should build for the right compute-node architecture.**

```
prepend-path
```

```
PE_PRODUCT_LIST XTPE_ISTANBUL
```

```
setenv
```

```
XTPE_ISTANBUL_ENABLED ON
```

```
setenv
```

```
INTEL_PRE_COMPILE_OPTS -msse3
```

```
setenv
```

```
PATHSCALE_PRE_COMPILE_OPTS -
```

**Oh yeah, let's link in the tuned math libraries for this architecture too.**



## Starting Points for Each Compiler

- PGI
  - `-fast -Mipa=fast(,safe)`
  - If you can be flexible with precision, also try `-Mfprelaxed`
  - Compiler feedback: `-Minfo=all -Mneginfo`
  - `man pgf90; man pgcc; man pgCC; or pgf90 -help`
- Cray
  - `<none, turned on by default>`
  - Compiler feedback: `-rm (Fortran) -hlist=m (C)`
  - If you know you don't want OpenMP: `-xomp` or `-Othread0`
  - `man crayftn; man craycc ; man crayCC`
- Pathscale
  - `-Ofast` Note: this is a little looser with precision than other compilers
  - Compiler feedback: `-LNO:simd_verbose=ON`
  - `man eko ("Every Known Optimization")`
- GNU
  - `-O2 / -O3`
  - Compiler feedback: good luck
  - `man gfortran; man gcc; man g++`

**12. Try MPICH\_PTL\_MATCH\_OFF  
For Latency-Sensitive Codes.**

---

## What is MPICH\_PTL\_MATCH\_OFF?

- If set => Disables Portals matching
  - Matching happens on the Opteron
  - Requires extra copy for EAGER protocol
- Reduces MPI\_Recv Overhead
  - Helpful for latency-sensitive application
    - Large # of small messages
    - Small message collectives (<1024 bytes)
- When can this be slower?
  - When extra copy time longer than post-to-Portals time
  - Lots of pre-posted Receives can slow it down
  - For medium to larger messages (16k-128k range)

# 11. Set MPICH\_FAST\_MEMCPY

## What is MPICH\_FAST\_MEMCPY?

- If set, enables an optimized memcpy routine in MPI. The optimized routine is used for local memory copies in the point-to-point and collective MPI operations.
  - This can help performance of some collectives that send large (256K and greater) messages.
    - Collectives are almost always faster
    - Speedup varies by message size
    - Example: If message sizes are known to be greater than 1 megabyte, then an optimized memcpy can be used that works well for large sizes, but may not work well for smaller sizes.
  - Default is not enabled (because there are a few cases that experience performance degradation)
- Ex: PHASTA at 2048 processes: reduction from 262s to 195s

# 10. Pre-post receives

---

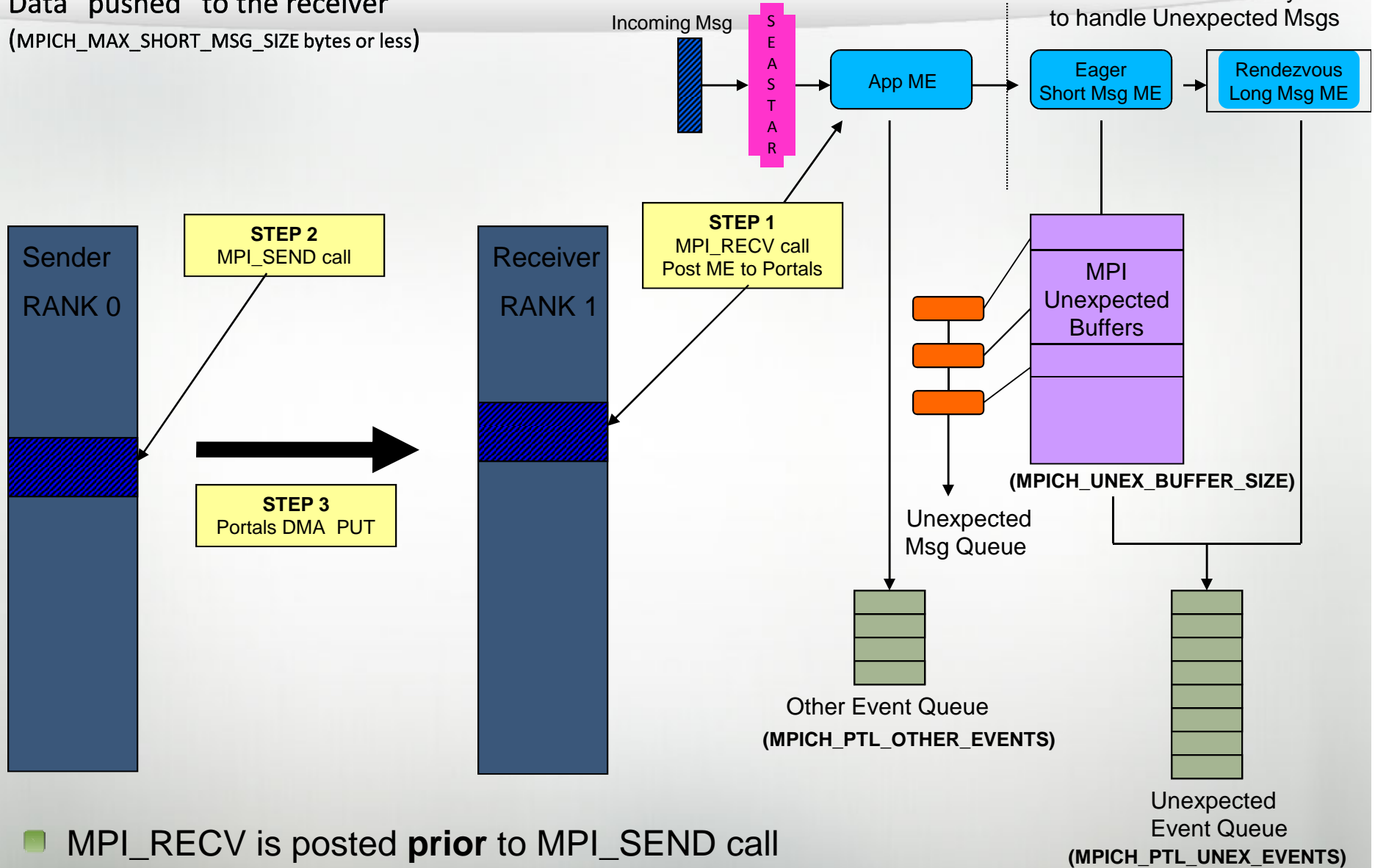


## Cray MPI XT Portals Communications

- Short Message **Eager** Protocol
  - The sending rank “pushes” the message to the receiving rank
  - Used for messages `MPICH_MAX_SHORT_MSG_SIZE` bytes or less
  - Sender assumes that receiver can handle the message
    - Matching receive is posted - or -
    - Has available event queue entries (`MPICH_PTL_UNEX_EVENTS`) and buffer space (`MPICH_UNEX_BUFFER_SIZE`) to store the message
- Long Message **Rendezvous** Protocol
  - Messages are “pulled” by the receiving rank
  - Used for messages greater than `MPICH_MAX_SHORT_MSG_SIZE` bytes
  - Sender sends small header packet with information for the receiver to pull over the data
  - Data is sent only after matching receive is posted by receiving rank

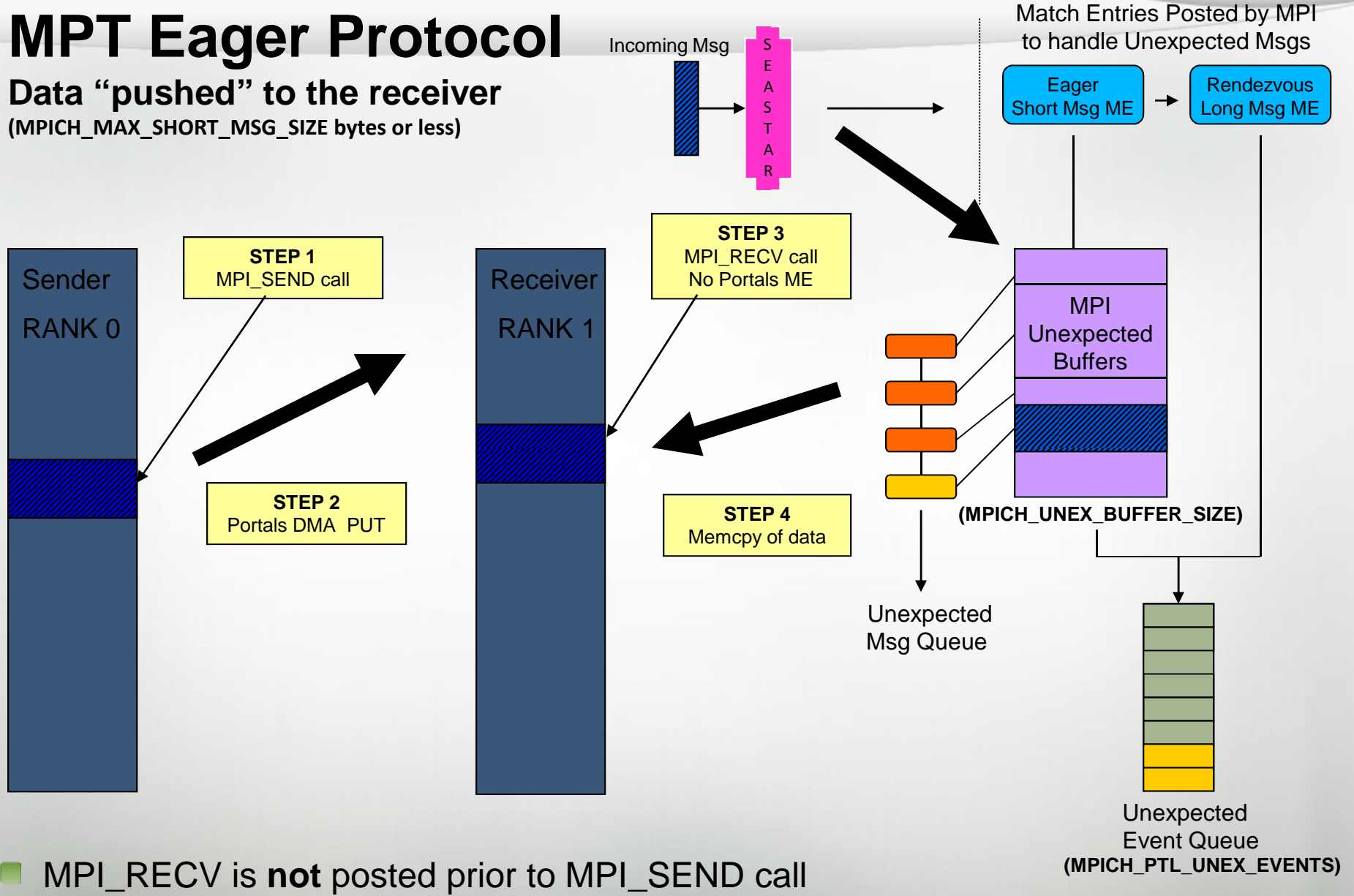
# MPT Eager Protocol

Data "pushed" to the receiver  
(MPICH\_MAX\_SHORT\_MSG\_SIZE bytes or less)



# MPT Eager Protocol

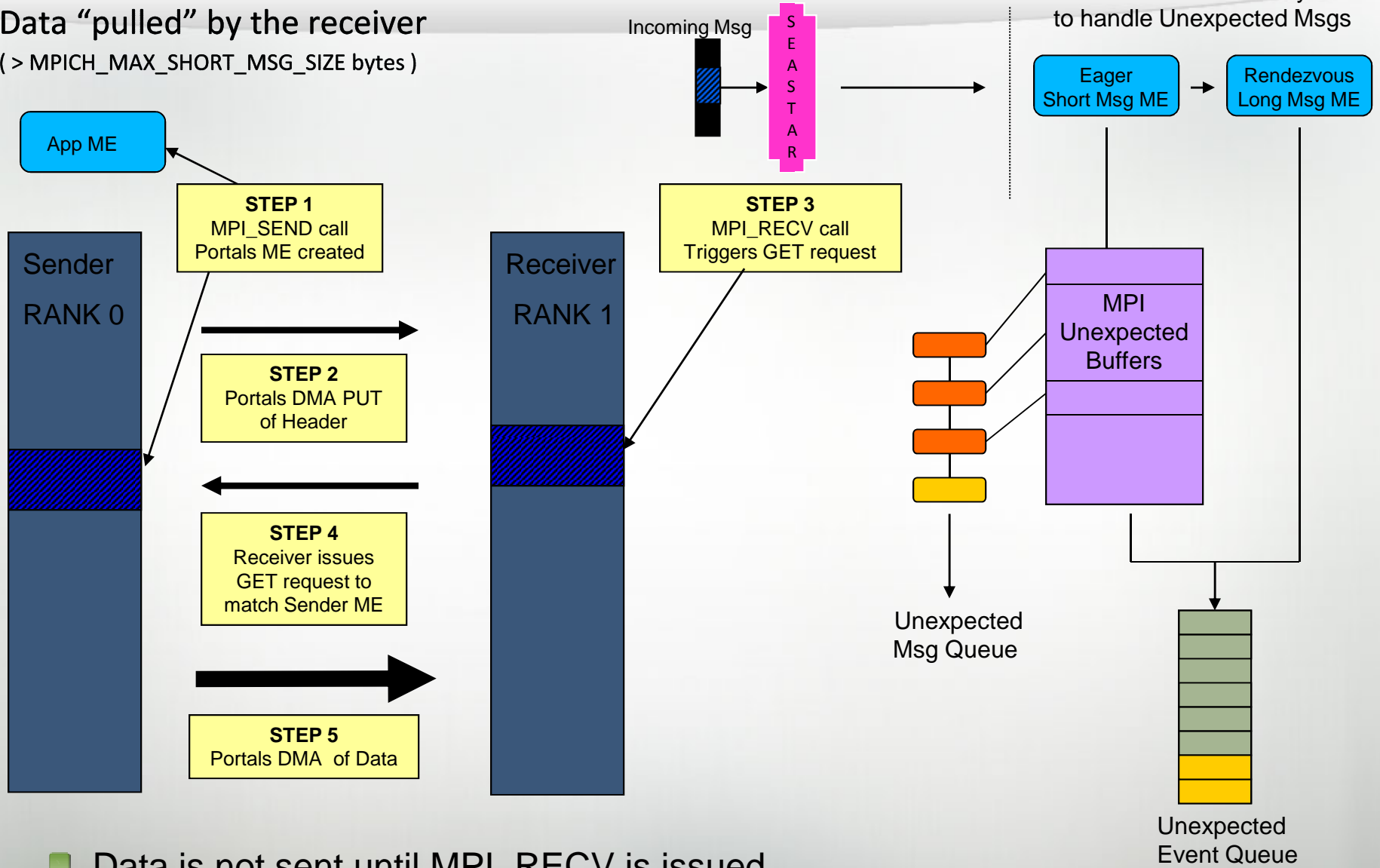
Data "pushed" to the receiver  
(MPICH\_MAX\_SHORT\_MSG\_SIZE bytes or less)



# MPT Rendezvous Protocol

Data "pulled" by the receiver

( > MPICH\_MAX\_SHORT\_MSG\_SIZE bytes )



## 9. Tweak

# MPICH\_MAX\_SHORT\_MSG\_SIZE

## Adjusting MPICH\_MAX\_SHORT\_MSG\_SIZE

- Controls message sending protocol
  - Message sizes  $\leq$  MSG\_SIZE: Use EAGER
  - Message sizes  $>$  MSG\_SIZE: Use RENDEZVOUS
  - Increasing this variable may require that MPICH\_UNEX\_BUFFER\_SIZE be increased
- Increase MPICH\_MAX\_SHORT\_MSG\_SIZE if App sends large messages and receives are pre-posted
  - Can reduce messaging overhead via EAGER protocol
  - Can reduce network contention
  - Can speed up the application
- Decrease MPICH\_MAX\_SHORT\_MSG\_SIZE if:
  - App sends lots of smaller messages and receives not pre-posted, exhausting unexpected buffer space

## 8. Stripe lustre directories.

---

## Lustre

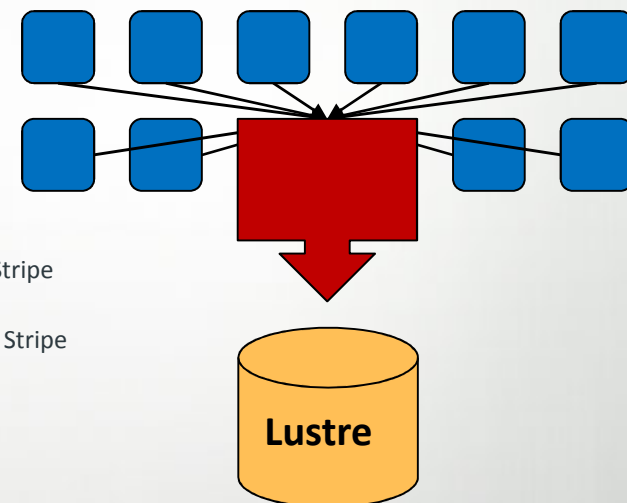
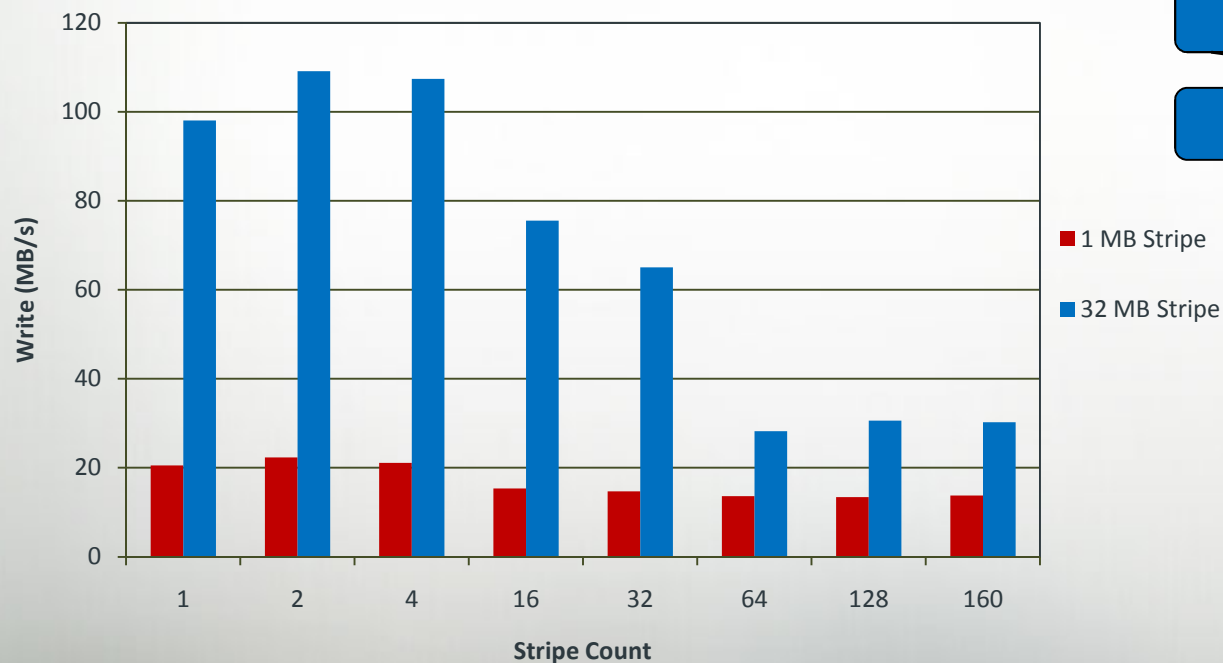
- The current XT4 has 72 OSTs
- The current XT6 has 12 OSTs
- esFS will have 84 OSTs so more parallelism
  
- It can be quite simple to get some significant performance benefits.



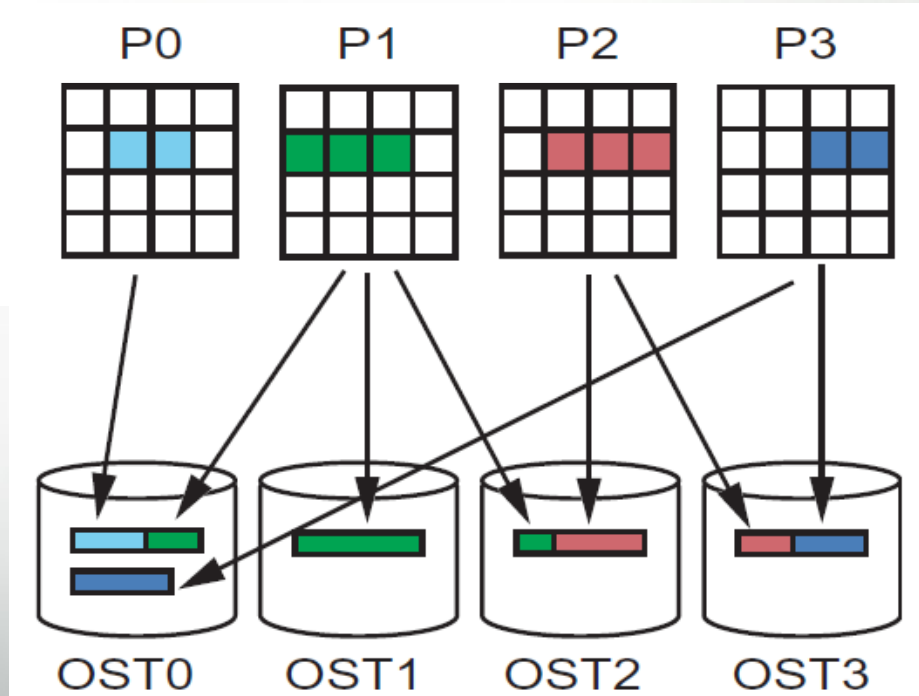
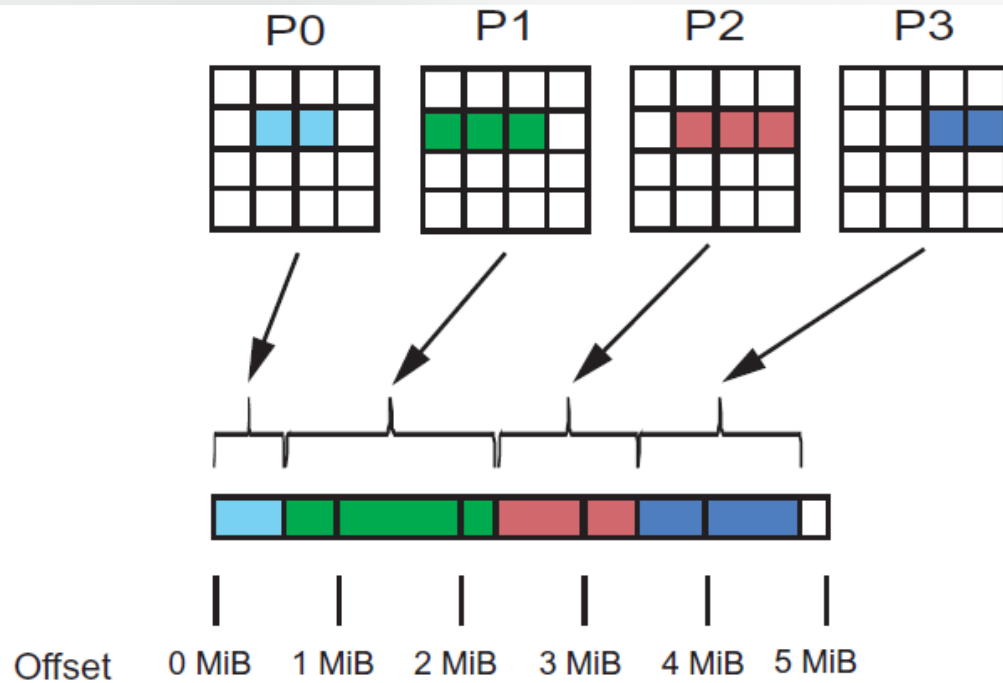
## Single writer performance and Lustre

- 32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size
  - Unable to take advantage of file system parallelism
  - Access to multiple disks adds overhead which hurts performance

**Single Writer  
Write Performance**



# File Striping: Physical and Logical Views

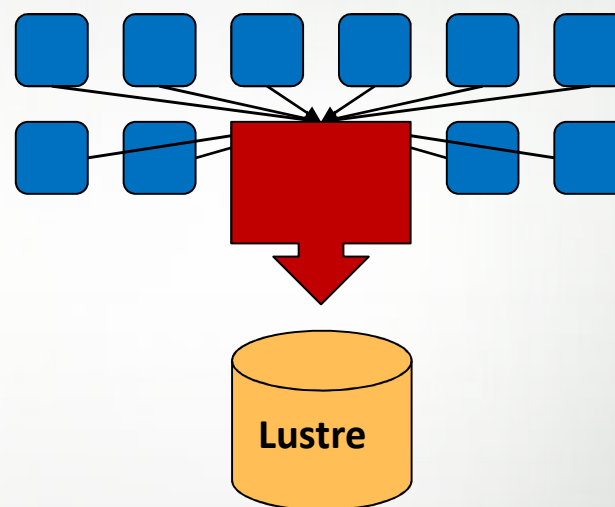


## Lustre: Important Information

- Use the `lfs` command, `libLUT`, or `MPIIO` hints to adjust your stripe count and possibly size
  - `lfs setstripe -c -1 -s 4M <file or directory>` (160 OSTs, 4MB stripe)
  - `lfs setstripe -c 1 -s 16M <file or directory>` (1 OST, 16M stripe)
  - `export MPICH_MPIIO_HINTS='*: striping_factor=160'`
- Files inherit striping information from the parent directory, this **cannot** be changed once the file is written
  - Set the striping before copying in files

## Standard Output and Error

- Standard Output and Error streams are effectively serial I/O.
- All STDIN, STDOUT, and STDERR I/O serialize through aprun
- Disable debugging messages when running in production mode.
  - “Hello, I’m task 32000!”
  - “Task 64000, made it through loop.”



# 7. Tune malloc.

---

# GNU Malloc

- GNU malloc library
  - malloc, calloc, realloc, free calls
    - Fortran dynamic variables
- Malloc library system calls
  - Mmap, munmap => for larger allocations
  - Brk, sbrk => increase/decrease heap
- Malloc library optimized for low system memory use
  - Can result in system calls/minor page faults

# Improving GNU Malloc

- Detecting “bad” malloc behavior
  - Profile data => “excessive system time”
- Correcting “bad” malloc behavior
  - Eliminate mmap use by malloc
  - Increase threshold to release heap memory
- Use environment variables to alter malloc
  - `MALLOC_MMAP_MAX_ = 0`
  - `MALLOC_TRIM_THRESHOLD_ = 536870912`
- Possible downsides
  - Heap fragmentation
  - User process may call mmap directly
  - User process may launch other processes
- PGI’s `-Msmartalloc` does something similar for you at compile time

## 6. Use the best communication layer possible

---



# 5. OpenMP

Either as an extra level or replacement level of parallelism

## OpenMP

- OpenMP can be used to better exploit the power of the Magny-Cours processor.
- It can be beneficial in a number of situations:
  - Where the code is limited in parallelism, i.e., a 3<sup>rd</sup> dimension could be parallelised with OpenMP
  - It may be simpler and more efficient to parallelise a certain algorithm with OpenMP compared to MPI
  - Where MPI begins to dominate the runtime.
  - Use of available OpenMP libraries

## OpenMP

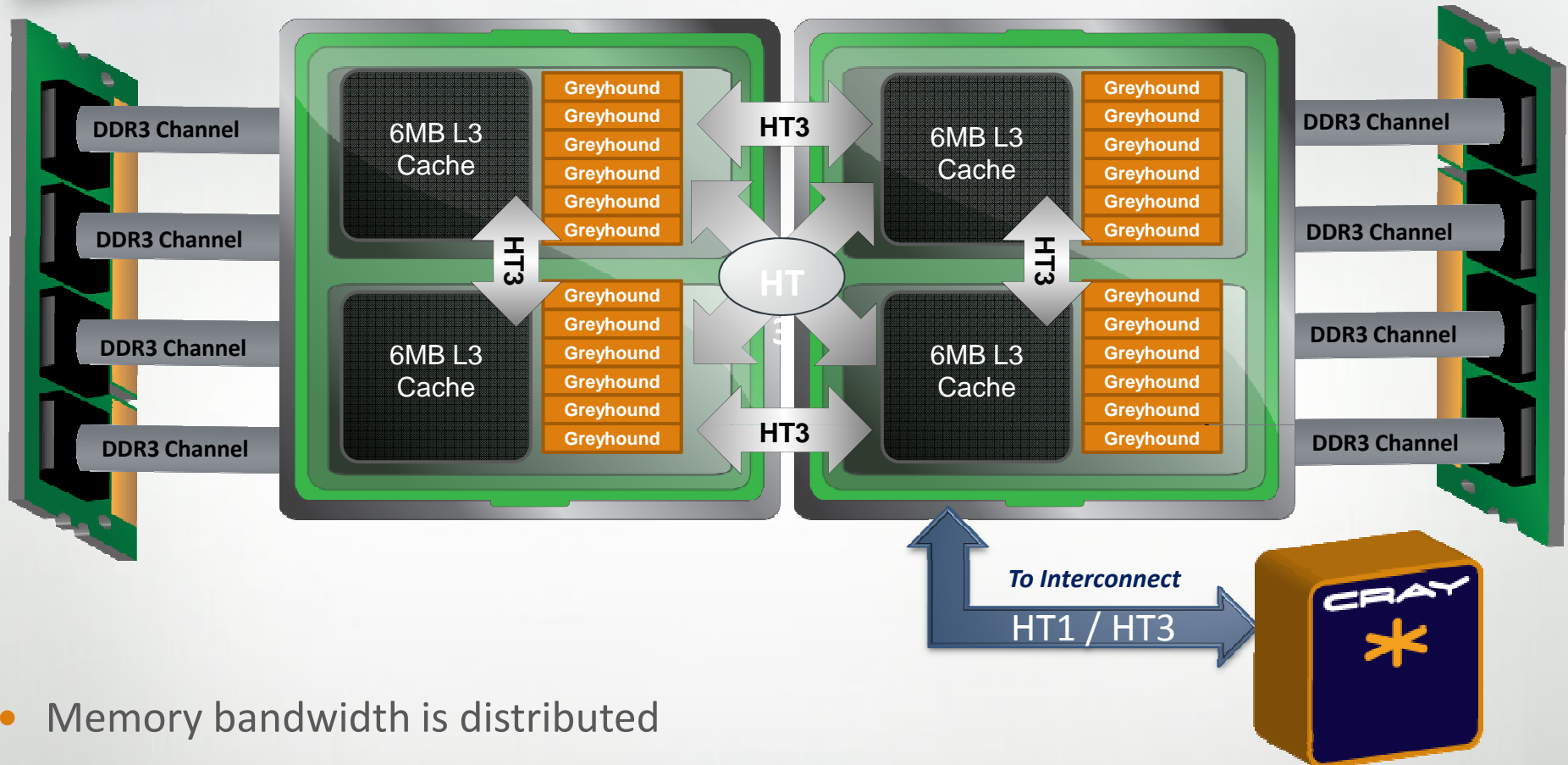
- The processes avoid the overhead of MPI for communication
- It is important that a big enough problem is available on node
  - Enough iterations for OpenMP to parallelise.
  - Enough work for each thread to continue to get high use of vectorization
  - It is best to produce OpenMP directives at a high level with sufficient granularity for good performance
- Castep is a great example where swapping OpenMP for MPI can produce great performance wins
  - See Chris Armstrong's talk.

## OpenMP

- OpenMP can only improve the computational aspects if the code is communication bound then there will be little improvement.
  - If a code takes 100 seconds of which only 20 seconds can be sped up using OpenMP then the best you can achieve on 24 threads is 80.83s
  - If when run on fewer cores (by a factor of 24 so 1 MPI rank per node) the code takes much less time in MPI. In this case 500 seconds maybe realistic , of which perhaps 480 seconds can be attacked with OpenMP. Performance will now be 40 seconds!
  
- Need also to be aware of locality (see next optimization), if you are using OpenMP within libraries then your application may control data placement.



# XT6 Node Details: 24-core Magny Cours



- Memory bandwidth is distributed
- If all data lies in the first dies memory then memory bandwidth is limited that available on the first die (two DDR3 links)
  - Instant 4x hit in available memory bandwidth!!

# 4. Touch your memory to improve locality

---

# Memory Allocation: Make it local

- Linux has a “first touch policy” for memory allocation
  - \*alloc functions don’t actually allocate your memory
  - Memory gets allocated when “touched”
- Problem: A code can allocate more memory than available
  - Linux assumed “swap space,” we don’t have any
  - Applications won’t fail from over-allocation until the memory is finally touched
- Problem: Memory will be put on the core of the “touching” thread
  - Only a problem if thread 0 allocates all memory for a node
- Solution: Always initialize your memory immediately after allocating it
  - If you over-allocate, it will fail immediately, rather than a strange place in your code
  - If every thread touches its own memory, it will be allocated on the proper socket

# 3. Try different MPI Rank Orders

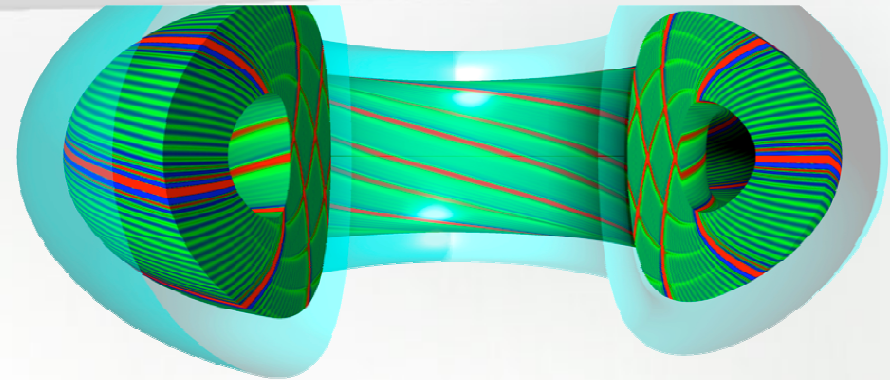


## Rank Placement

- The default ordering can be changed using the following environment variable:  
MPICH\_RANK\_REORDER\_METHOD
- These are the different values that you can set it to:
  - 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: SMP-style placement – Sequential ranks fill up each node before moving to the next.
  - 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: Custom ordering. The ordering is specified in a file named MPICH\_RANK\_ORDER.
- When is this useful?
  - Point-to-point communication consumes a significant fraction of program time and a load imbalance detected
  - Also shown to help for collectives (alltoall) on subcommunicators (GYRO)
  - Spread out IO across nodes (POP)

## Reordering example - GYRO

- GYRO 8.0
  - B3-GTC problem with 1024 processes
- Run with alternate MPI orderings
  - Custom: profiled with with `-O apa` and used reordering file `MPICH_RANK_REORDER`.



Reorder method	Comm. time
Default	11.26s
0 – round-robin	6.94s
2 – folded-rank	6.68s
d-custom from apa	8.03s


CrayPAT  
suggestion  
almost right!

## Reordering example - TGYRO

- TGYRO 1.0
  - Steady state turbulent transport code using GYRO, NEO, TGLF components
- ASTRA test case
  - Tested MPI orderings at large scale
  - Originally testing weak-scaling, but found reordering very useful

Reorder method	TGYRO wall time (min)		
	20480	40960	81920
Default	99m	104m	105m
Round-robin	66m	63m	72m

Huge win!



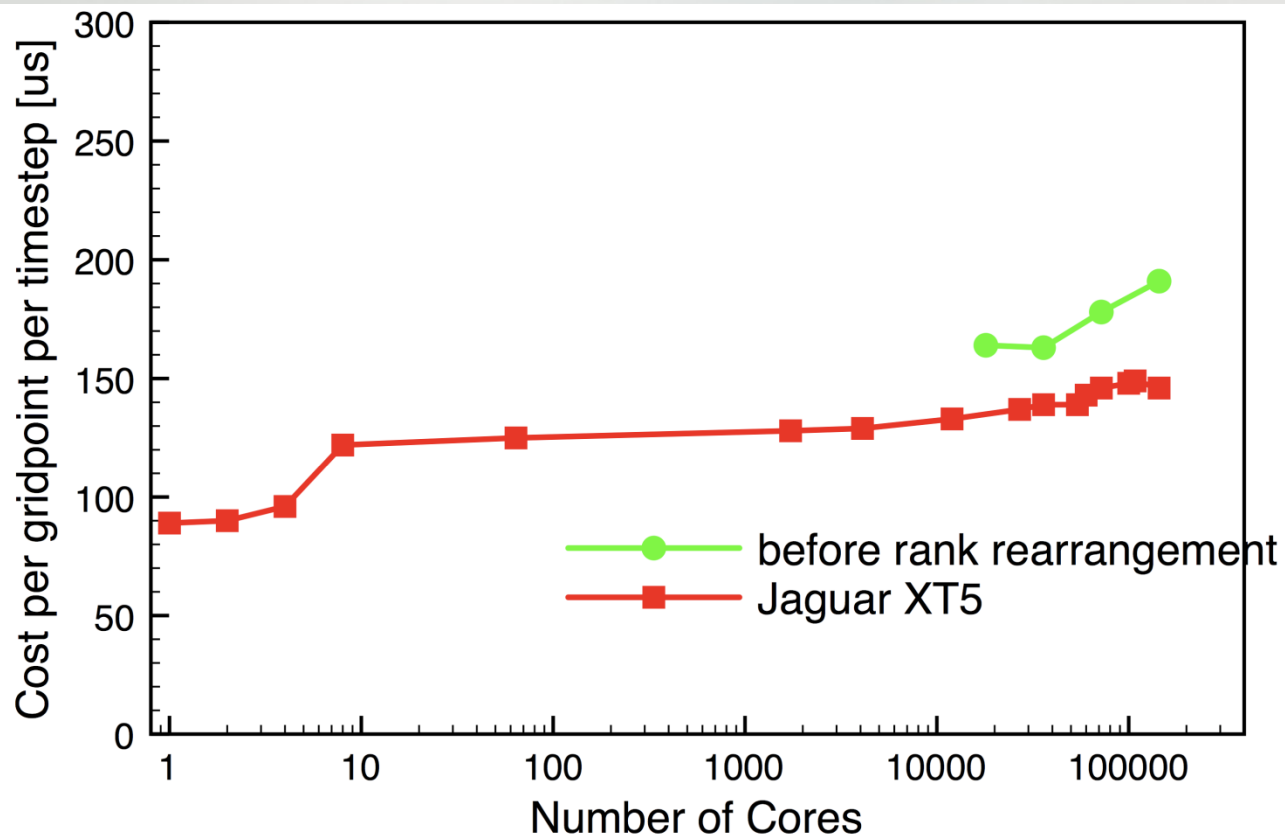
## Rank Reordering Case Study

Application data is in a 3D space, X x Y x Z.

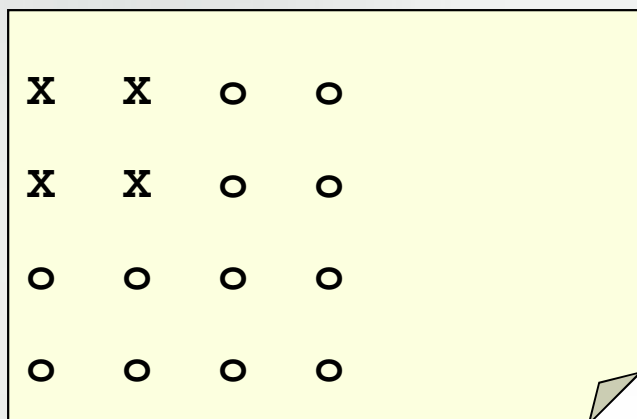
Communication is nearest-neighbor.

Default ordering results in 12x1x1 block on each node.

A custom reordering is now generated: 3x2x2 blocks per node, resulting in more on-node communication



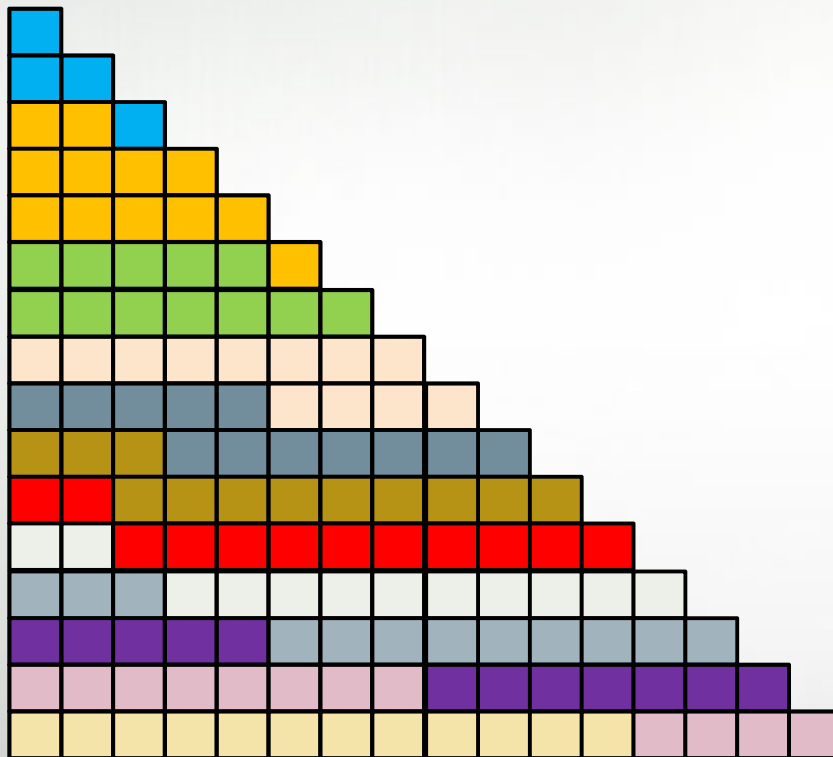
# Rank order choices



- Nodes marked X heavily use a shared resource
  - If memory bandwidth, scatter the X's
  - If network bandwidth to others, again scatter
  - If network bandwidth among themselves, concentrate
- Check out pat\_report, grid\_order, and mgrid\_order for generating custom rank orders based on:
    - Measured data
    - Communication patterns
    - Data decomposition

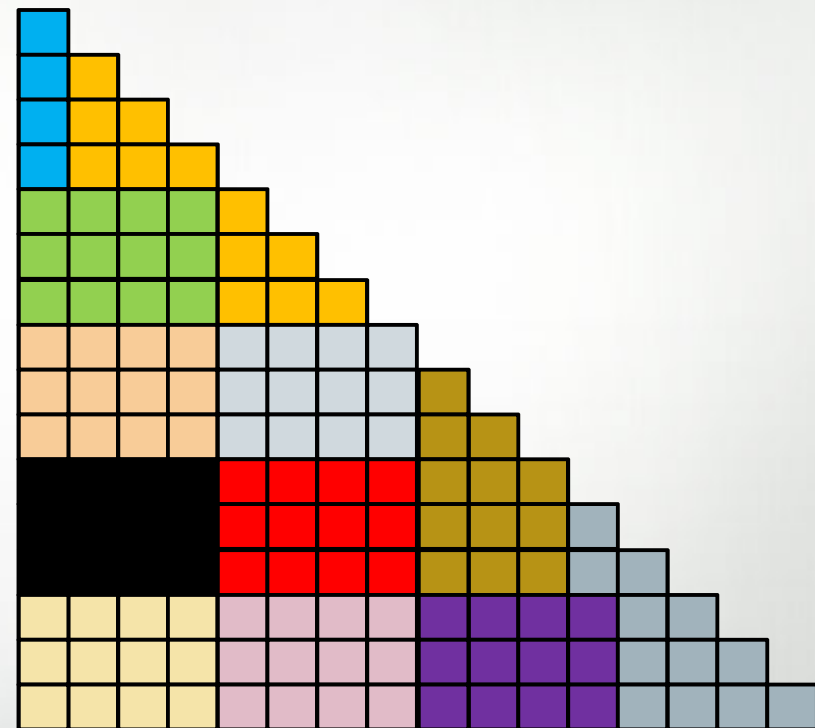
# Custom Placement

- Default Placement



- 7.15 seconds per iteration

- Custom Placement

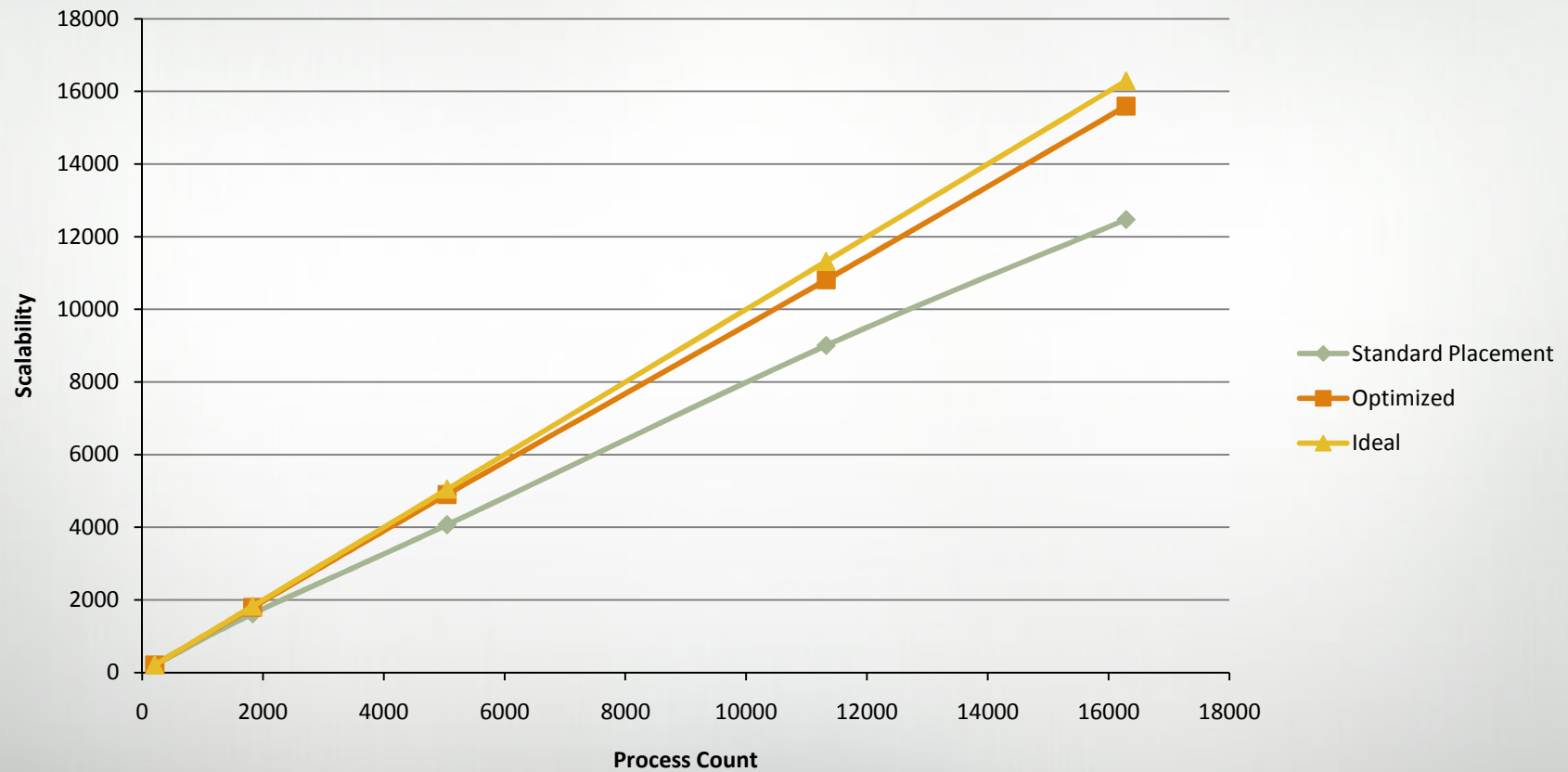


- 5.58 seconds

**22% Improvement**

# Custom Placement Results

## Scalability Improvements Achieved with Process Placement



## 2. SSE Vecotorization

---



## SSE Vectorization

- This is still important and has been stressed before.
- If you can vectorize your code you can double the performance.
- If you can halve the precision for some solvers (and the solver vectorizes) you can double the performance.
  - The SSE instruction can perform two 64 bit calculations simultaneously
  - It can perform four 32 bit calculations.
- This reduction in precision of the real can be done within some Lapack routines with the use of IRT.

# 1. Optimize for Cache.

---

- Questions?
  
- Kevin Roy: [kroy@cray.com](mailto:kroy@cray.com)
- Jason Beech-Brandt: [jason@cray.com](mailto:jason@cray.com)
- Tom Edwards: [tedwards@cray.com](mailto:tedwards@cray.com)