Heterogeneous environments:

Cray XT4 at CSC

Juha Fagerholm, Jussi Hynninen, Tapani Kinnunen,
Petri Nikunen, Olli Serimaa, and Joni Virtanen

firstname.lastname@csc.fi

Contents

1. Cray XT4 system
   - General presentation of system's architecture, operating system, etc.
2. DEISA environment at CSC
   - Only UNICORE or DESHL job submission are possible
   - DEISA file systems will be available
   - Assistance is possibly needed to compile programs
   - CSC's DEISA common production environment differs from those at IBM AIX sites
3. Optimization for Cray
   - I/O optimization
   - Optimizing with compilers
   - Locating hot spots of the code (profiling)
   - Mathematical libraries
   - Single-node optimization
   - MPI optimization

CSC Cray XT4 = Louhi

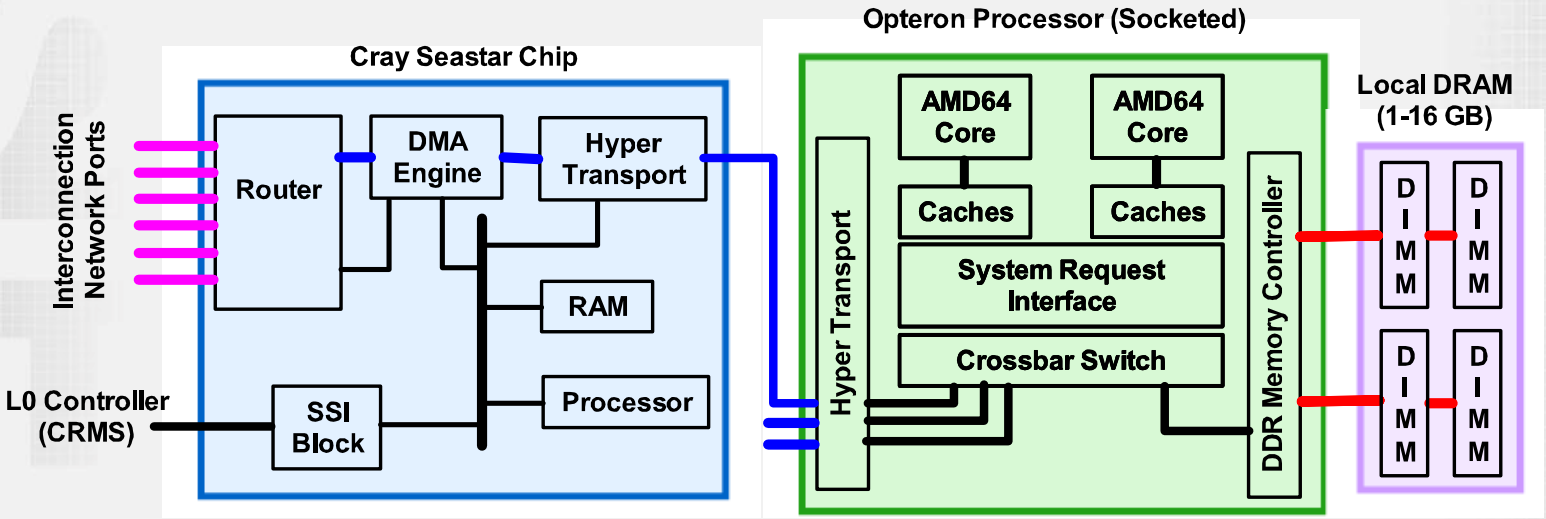
- The name of the new supercomputer is Louhi
- Louhi is one of the fictional characters in Finnish national epos Kalevala:
  - Louhi, Pohjolan Emäntä (also Pohjan Akka)
  - Lady of Norden, Mistress of Pohjola
- Also “louhi” is the imperative form of the verb “louhia”, “to mine”, “to extract”
- DNS-name: louhi.csc.fi

Cray XT4 hardware

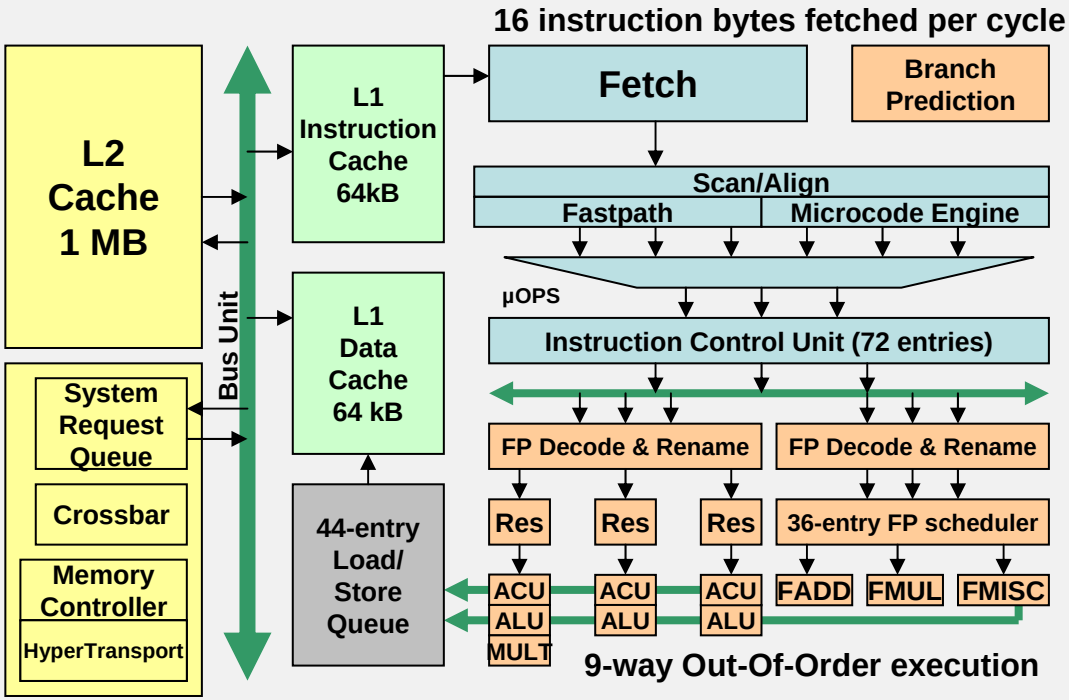
- First phase in 2007 (1Q2007)
  - 1012 compute nodes, 2024 cores, 11 cabinets
  - AMD Opteron DualCore 2.6 GHz processors
  - memory 1 GB/core, partly extended (4Q2007)
  - Cray SeaStar 2 interconnect (3D torus)
  - 10.6 TF peak, 8.75 TF HPL
- Second phase in 2008:
  - AMD Opteron QuadCore 2.6 GHz processors
  - 6736 cores, 18 cabinets
  - 70 TF peak

Compute node architecture

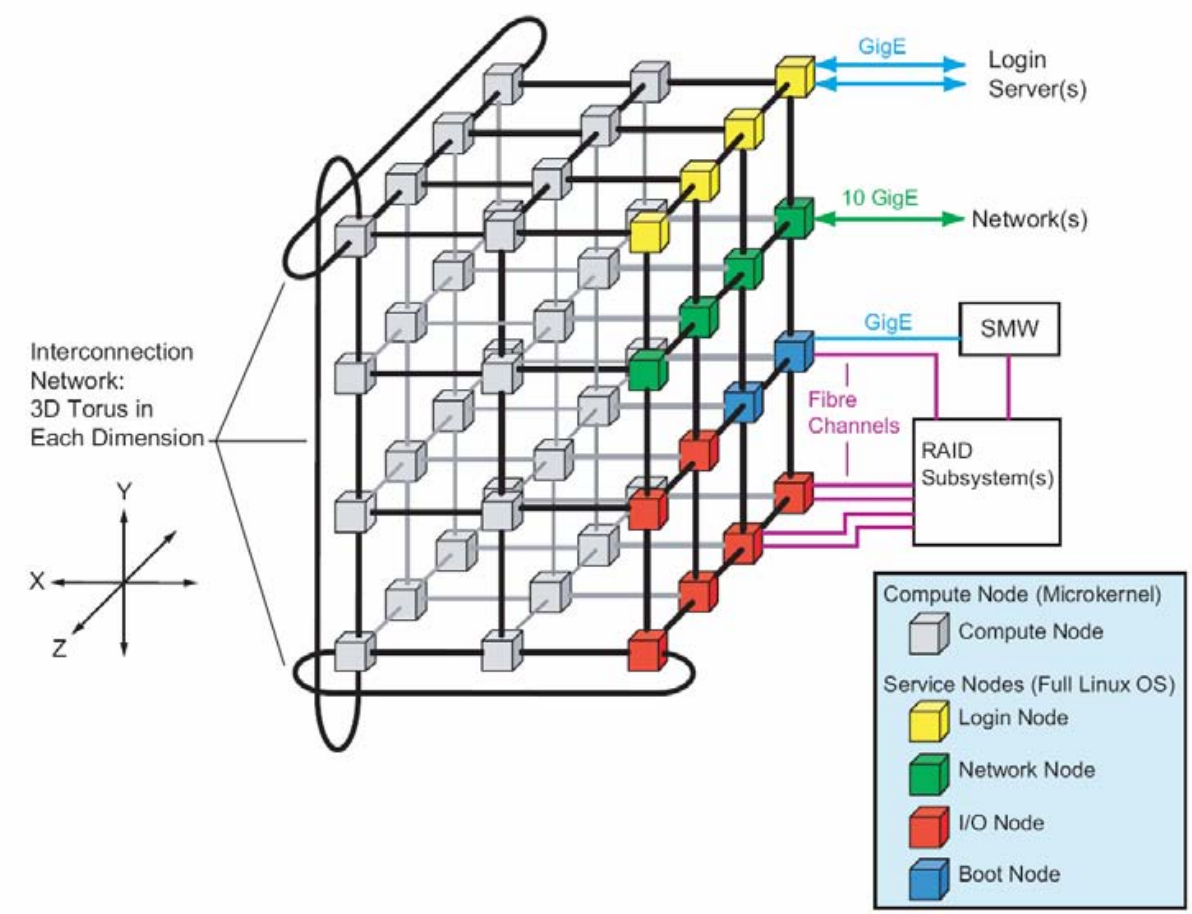
- AMD Opteron DualCore 2.6 GHz
- Cray SeaStar 2 Chip



AMD Opteron Processor



Cray 3D torus interconnect



Cray XT4 system software

- Unicos/lc (SUSE Linux, LWK Catamount in compute nodes)
- PBS Pro as batch job management system
- Lustre 1.5 parallel file system (based on 1.4.6)
  - 74 TB local disk space
  - $DEISA_SCRATCH is located on the Lustre system

Program development tools

- PGI compilers 6.2.5
- PathScale 2.5, GNU compilers 3.2.3 & 4.1.1
- CrayPat Performance Analysis Tool collection (profiling, HW counters, etc.)
- Cray Apprentice2 (visualizer for CrayPat)
- TotalView debugger
- Optimized libraries
  - 64 bit AMD Core Math library (ACML)
  - Level 1,2,3 of BLAS, LAPACK, FFT
  - SciLib: Scalapack, BLACS, SuperLU
- MPI library derived form MPICH-2
- Also SHMEM, but no OpenMP/threads

Contents

1. Cray XT4 system
   - General presentation of system's architecture, operating system, etc.
2. DEISA environment at CSC
   - Only UNICORE or DESHL job submission are possible
   - DEISA file systems will be available
   - Assistance is possibly needed to compile programs
   - CSC's DEISA common production environment differs from those at IBM AIX sites
3. Optimization for Cray
   - I/O optimization
   - Optimizing with compilers
   - Locating hot spots of the code (profiling)
   - Mathematical libraries
   - Single-node optimization
   - MPI optimization

Job submission

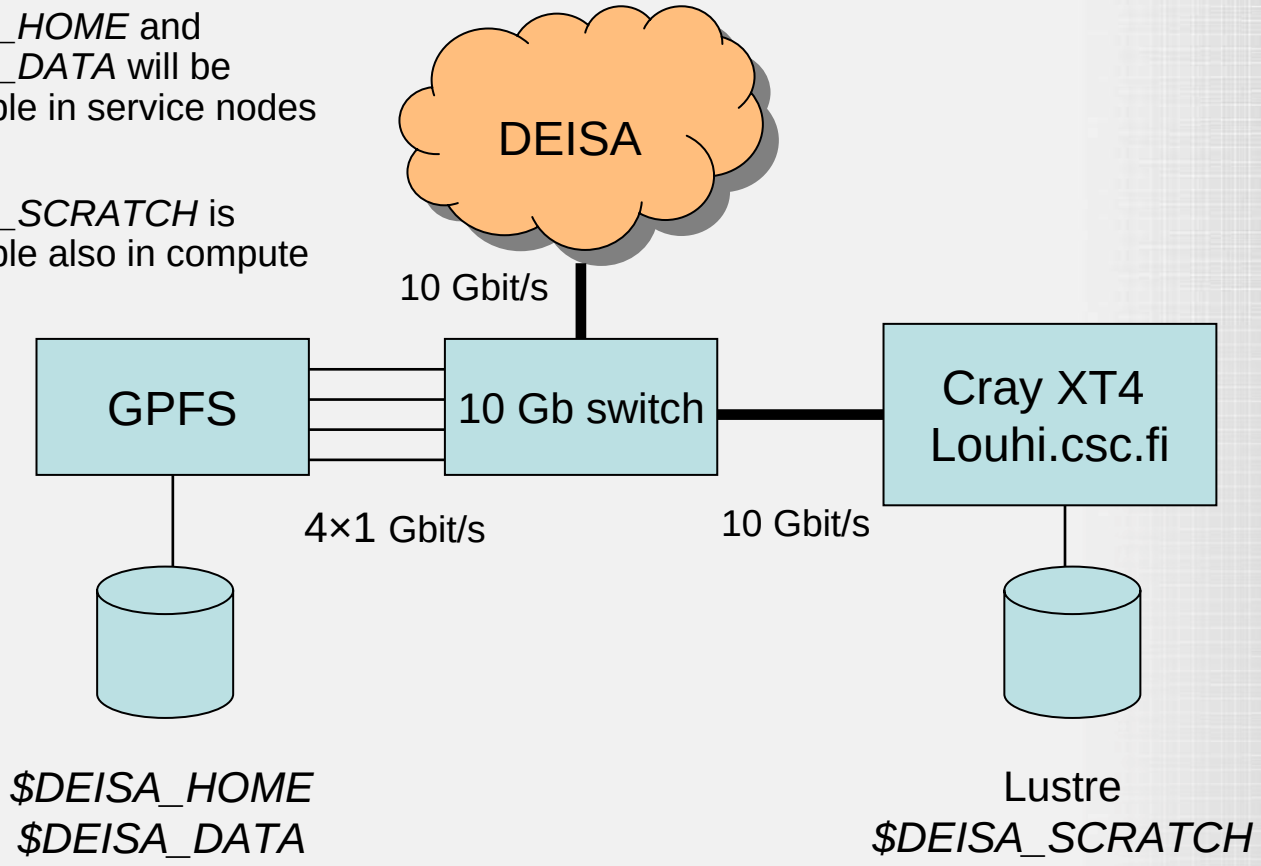
- Only UNICORE (/DESHL) job submission initially possible
  - The XT4 system uses PBS Pro as its scheduler
  - As the LSF/Universus installation proceeds, job submission utilizing the DEISA metascheduler (LSF) will be an option
- DESHL will also be available on the XT4 via the modules environment
  - *module load deshl*
- Since the DEISA GPFS directories will not be available on the compute nodes, data staging is mandatory
  - Users will have to copy their work files under *$DEISA_SCRATCH* (a local Lustre file system) before starting the actual job and back to *$DEISA_DATA* / UNICORE uspace / etc.

DEISA file systems

- The DEISA GPFS file systems not available yet
  - *$DEISA_HOME, $DEISA_DATA*
  - Users will have to transfer their data to the system by other means
  - *$DEISA_SCRATCH* always points to a local file system (as described in DEISA Primer) – Lustre is used as scratch
- GPFS tests on a Linux cluster currently ongoing
  - When found stable enough, GPFS will be made available on the XT4 (this summer, most likely)
  - Only on login/service nodes → data staging needed
    - The compute nodes currently have the Catamount kernel, GPFS not available
    - Possible switch to Compute Node Linux later this year → GPFS may be implemented on the compute nodes as well

DEISA directories in Cray (GPFS)

- DEISA GPFS
  - *$DEISA_HOME* and *$DEISA_DATA* will be accessible in service nodes
- Lustre
  - *$DEISA_SCRATCH* is accessible also in compute nodes



DEISA CPE Packages on Cray

- **DEISA CPE Program Packages on CSC Cray Computer:**
- **Most of the programs in DEISA CPE are already installed (shown GREEN) or will be installed**
  - **TBI = to be installed in the tables below (shown BLUE)**
- **Basically only IBM specific programs and libraries will be left out (shown RED)**
  - **And some of them will be replaced by similar Cray/Linux specific program or library**

DEISA CPE, Shells and Compilers

Description	Status	Version	Remarks
Shells			
Bourne Again Shell (Bash)	**OK**	2.05b.0(1)	Cray
Extended C Shell (Tcsh)	**OK**	6.12.0	Cray
Compilers			
Portland C compiler (pgcc)	**OK**	6.2-5	64-bit on x86-64 Linux
Portland C++ compiler (pgCC)	**OK**	6.2-5	64-bit on x86-64 Linux
Portland Fortran compiler (pgf90, f77)	**OK**	6.2-5	64-bit on x86-64 Linux
GNU C compiler (gcc)	**OK**	3.3.3	SUSE Linux
GNU C++ compiler (g++)	**OK**	3.3.3	SUSE Linux
GNU Fortran compiler (g77)	**OK**	3.3.3	SUSE Linux (no Fortran 90)
Sun Java compiler	**OK**	1.6	1.4, 1.5 and 1.6, SUSE Linux

DEISA CPE, Libraries 1

	Description	Status	Version	Remarks
LIBRARIES				
blacs	Basic Linear Algebra Communication Subroutines	**OK**	3.6	Included in Cray's ACML (module acml)
blas	Basic Linear Algebra Subroutines	**OK**	3.6	Included in Cray's ACML
libsci	numerical subroutine library	**OK**	3.6	Included in Cray's ACML, Cray only
lapack	Linear Algebra PACKage	**OK**	3.6	Included in Cray's ACML
scalapack	SCAlable Linear Algebra PACKage	**OK**	3.6	Included in Cray's ACML
fftw	Fast Fourier Transform library	**OK**	3.1.1	also version 2.1.5; incompatible!
nag	Numerical Algorithms Group	**TBI**	20/21?	commercial
hdf5	Hierarchical Data Format (version 5)	**TBI**	?	
netcdf	Network Common Data Format	**OK**	3.6.1	

DEISA CPE, Libraries 2

	Description	Status	Version	Remarks
LIBRARIES				
blacssmp	Basic Linear Algebra Communication Subroutines	**NO**		IBM only, SMP
blassmp	Basic Linear Algebra Subroutines	**NO**		IBM only, SMP
essl	IBM Engineering and Scientific Subroutine Library	**NO**		IBM only, replaced by Cray's libsci
pessl	IBM Parallel ESSL	**NO**		IBM only
mass	IBM basic numerical library	**NO**		IBM only
wsmp	IBM Watson Sparse Matrix Package	**NO**		IBM only, commercial

DEISA CPE, Tools

	Description	Status	Version	Remarks
Tools				
emacs	Emacs Text editor	**OK**	**21.3.1**	
gmake	GNU make	**OK**	**3.80**	
nedit	Nedit Text editor	**OK**	**5.5**	
omniorb	OmniOrb CORBA implementation utilities	**OK**	**4.0.1**	
OpenSSH	Open Secure Shell	**OK**	**4.1p1**	
OpenSSL	Open Secure Shell utilities	**OK**	**0.9.7d**	17 Mar 2004
perl	PERL (Practical Extraction and Report Language)	**OK**	**5.8.3**	Login nodes only
python	Development language	**OK**	**2.3.3**	Login nodes only
tcl	Tool Command Language (TCL)	**OK**	**8.4**	
tk	GUI toolkit for Tcl	**OK**	**8.4**	
totalview	TotalView parallel debugger (Etnus)	**OK**	**7.3.0-0**	commercial (module totalview)
cray pat	Cray Performance Analysis Tool (PAT)	**OK**	**3.1.2**	also v3.1.1 (module craypat)
hpm	IBM Hardware Profiler Monitor	**NO**		IBM only, replaced by Cray PAT

DEISA CPE, Applications

	Description	Status	Version	Remarks
Applications				
cpmd	Molecular dynamics (Car-Parrinello)	**TBI**	3.11.1	
cpmd2cube	Post-processing of CPMD density/wannier files	**TBI**		Login nodes only
gopenmol	Post-processing of cubes-files (g94cube2pl)	**TBI**	3.00	Login nodes only
namd	Molecular dynamics	**TBI**	?	
wien2k	Electronic structure calculations of solids	**OK**	06.4	testing needed
torb	Turbulence	**NO**		IBM only
torbwsmp	Turbulence (using WSMP)	**NO**		IBM only

Remarks

- Some of the programs run only with (SUSE) Linux
  - That means that they run on login/service nodes only
  - They don't run on compute nodes (with Catamount)
  - That can change later, if and when compute node Linux arrives
- For compute nodes, one must compile Catamount compliantly
  - We'll try to "hide" this from (DEISA) users, using compiler wrappers and module system
  - Avoid using compilers "directly" (i.e. without wrappers)
- Most numeric libraries are implemented by Cray's version of ACML (AMD Core Math Library) and Cray's LibSci
  - There are several different versions of ACML (versions 3.0 or 3.6, for Portland or GCC compiler, large arrays, and some mp version)
  - And hence also several different modules for ACML

- Situation may (will) change!
- so check!
- Primer v1.5
- DEISA site:
  - http://www.deisa.org/
  - (http://www.deisa.eu/)
- CSC web site, Louhi user guide:
  - http://www.csc.fi/english/pages/louhi_guide/

Contents

1. Cray XT4 system
   - General presentation of system's architecture, operating system, etc.
2. DEISA environment at CSC
   - Only UNICORE or DESHL job submission are possible
   - DEISA file systems will be available
   - Assistance is possibly needed to compile programs
   - CSC's DEISA common production environment differs from those at IBM AIX sites
3. Optimization for Cray
   - I/O optimization
   - Optimizing with compilers
   - Locating hot spots of the code (profiling)
   - Mathematical libraries
   - Single-node optimization
   - MPI optimization

General optimization process

- In order of importance
  1. When a better algorithm is available, use it.
  2. If I/O is a significant part of the program, tune for it next.
  3. Use the best set of compiler optimization flags.
  4. Locate the hot spots in the program (profiling).
  5. Use the ACML and LibSci mathematical libraries when possible.
  6. Hand-tune the code for the Cray XT4 architecture
     (most optimization tricks work on all modern platforms).

I/O optimization

- General principles
  - Use unformatted I/O when possible.
  - Write whole arrays or strings rather than one element at a time.
  - Use memory for intermediate results.
- Use Buffered I/O
  - Considerably faster than unbuffered I/O.
  - Use buffers of at least 1-10 MBytes.
  - Load iobuf module and relink the program *(module load iobuf)*.
- Consider using Parallel I/O
  - MPI-IO gives optimal performance for large I/O operations.
  - The format of the parallel file is identical to that produced by a serial program.
  - MPI-IO is part of the MPI-2 standard.

Optimizing with compilers

- Portland Group (PGI) compiler options
  - First try
    - *cc -fastsse -Mipa=fast prog.c* *(cc,CC, ftn, f77)*
  - Next try some of these
    - *-Mipa=fast,inline*
    - *-O3* or *-O4*
    - *-tp k8-32* (32-bit programs run sometimes faster)
  - Keep in mind that **not all** optimizations make a program run faster and some may even produce wrong results!
  - To check for vectorization, inlining, loop unrolling, etc.
    - *-Minfo* or *-Minfo=all*

yod options for performance

- You should almost always use small pages
  - *yod -sz 256 -VN –small_pages a.out*
  - Especially important when the program makes large jumps in memory
    - Indirect addressing
    - Sparse matrices

Locating hot spots of the code

- Locating hot spots of the code (profiling)
  - Profiling is important. Do not waste time tuning the code that is infrequently executed.
  - The profile depends on input data, but not too much in many cases.
  - The profile depends on the platform on which it is executed, but usually not much.
  - On the Cray XT4 system, you can use CrayPat performance analysis tool to profile programs
    - See Louhi user's guide for details: http://www.csc.fi/english/pages/louhi_guide/

Mathematical libraries

- The most important mathematics libraries on CSC's Cray system are
  - ACML (AMD Core Math Library)
    - BLAS : Basic Linear Algebra Subprograms
    - LAPACK : Linear Algebra Package for solving linear equations and eigenvalue problems
    - FFT : a set of routines for Fast Fourier Transform
    - RNG : a set of Random Number Generators and statistical distribution functions
    - Fast vectorized versions of standard mathematical functions, such as sin, cos, exp, log, etc.

  LibSci (Cray Scientific Library) For parallelized codes
    - BLACS : Basic Linear Algebra Communication Subprograms
    - ScaLAPACK : a parallel version of LAPACK
    - SuperLU : for solving large sparse linear systems
    - FFT : parallel FFT routines

Mathematical libraries

- Why use ACML and LibSci?
  - They provide functionality that is not so easy to code yourself. Do not reinvent the wheel!
  - Mathematical libraries are better tested and thus more reliable than typical users' codes.
  - ACML and LibSci are optimized for the Cray XT4 design. It is extremely difficult to beat their performance.

Single-node optimization

- General coding guidelines for performance Not Cray-specific!
  - Use optimized data types
    - *int / INTEGER*4*
    - *long / INTEGER*8*
    - *float / REAL*4*
    - *double / REAL*8*
    - *COMPLEX*8*
    - *COMPLEX*16*
  - Use smallest floating-point precision appropriate to your program
    - Use *long double / REAL*16 / COMPLEX*32* only when extremely high precision is required!

Single-node optimization

- General coding guidelines for performance (cont.)
  - Use local variables whenever possible
    - The compiler can analyze the use of local variables but it has to make several worst-case assumptions about global variables. These assumptions tend to restrict optimization.
  - Avoid using pointers
    - Keeping track of pointers during optimization is difficult and thus prevents most memory optimization.
  - Access data sequentially
    - In loops, access data arrays with unit stride to improve cache utilization.

C	···	a[n][1]	a[n][2]	a[n][3]	a[n][4]		···
Fortran	···	a(1,n)	a(2,n)	a(3,n)	a(4,n)		···

Single-node optimization

– Example

C	…	a[n][1]	a[n][2]	a[n][3]	a[n][4]		…
Fortran	…	a(1,n)	a(2,n)	a(3,n)	a(4,n)		…

Untuned

```
DO I=1,N
   DO J=1,N
      A(I,J)=B(I,J)
   ENDDO
ENDDO
```

Tuned

```
DO J=1,N
   DO I=1,N
      A(I,J)=B(I,J)
   ENDDO
ENDDO
```

Single-node optimization

- General coding guidelines for performance (cont.)
  - To allow for vectorization, loops should not include
    - *if* statements and other conditionals
    - divides
    - function and subroutine calls

Untuned

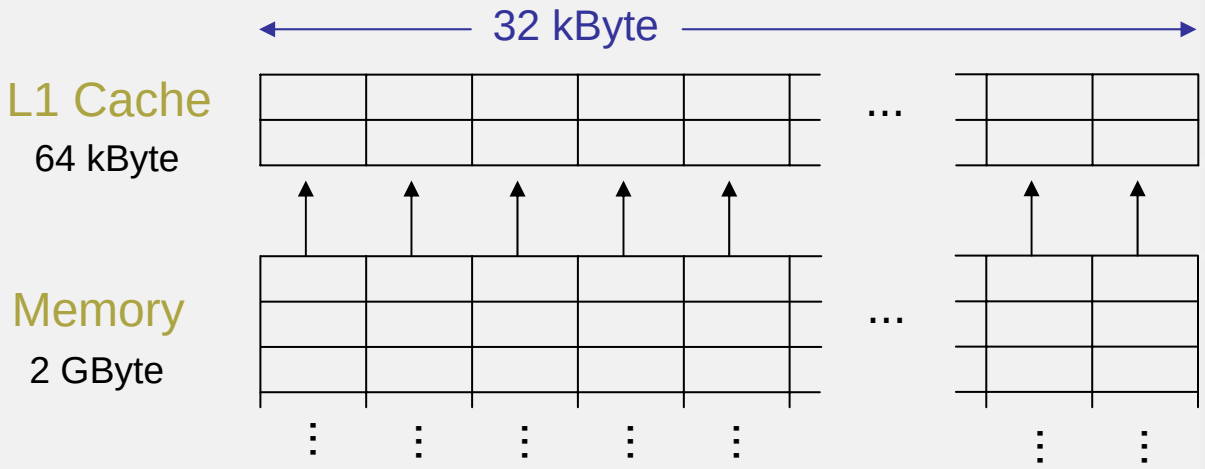
```
DO I=1,N
   DO J=1,N
      A(J,I)=SIN(X(J))
   END DO
ENDDO
```

Tuned

```
DO J=1,N
   SINX(J)=SIN(X(J))
ENDDO
DO I=1,N
   DO J=1,N
      A(J,I)=SINX(J)
   END DO
ENDDO
```

Single-node optimization

- Set associativity
  - A given memory line can only map to a small set of cache locations
  - For example, Cray XT4's L1 cache is 2-way set associative:
    - 2 cache lines are available for a given memory line



Single-node optimization

- Implication of the set associative cache structure:
  - Strides which are large powers of 2 are **very unfavorable**—the effective cache size reduces to just 2 cache lines.
  - Smaller powers of 2 also reduce effective cache size, but not as badly.
- Solution
  - pad arrays with empty space to avoid having a dimension which is a large power of 2.

Untuned

```
REAL*8 :: A(4096,N)
```

2^{15} Byte

Tuned

```
REAL*8 :: A(4104,N)
```

2^{15} + 64 Byte

Single-node optimization

- Memory allocation performance
  - Default *malloc* is Catamount custom implementation of the *malloc()*
    - Tuned to Catamount's non-virtual-memory operating system.
    - Favors applications allocating large, contiguous data arrays.
    - Not always the fastest—if you experience performance problems with Catamount *malloc*, try glibc *malloc()* also *(module load glibc)*.

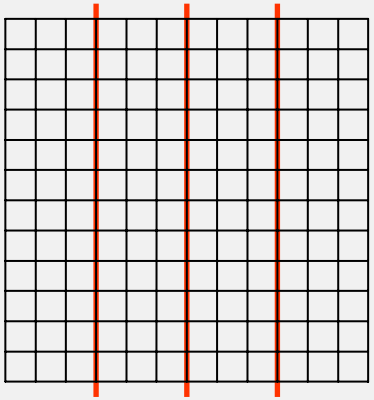
MPI optimization

- Cray XT4 provides two kinds of parallelism
  - Cray Shared Memory Access (SHMEM)
    - between cores within nodes
  - Message Passing Interface (MPI)
    - between cores within nodes
    - between cores on different nodes
- The basic parallel programming model for Cray XT4 is MPI.

MPI optimization

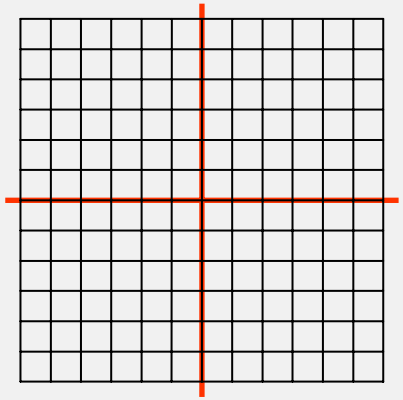
- For peak MPI performance
  - Minimize data to be passed
    - Example: domain decomposition with 4 partitions

Untuned



72 border cells

Tuned



44 border cells

MPI optimization

- For peak MPI performance (cont.)
  - Balance load between tasks
    - With unbalanced load, some cores are idle for some of the time, which increases the total runtime of the program.
  - Collect many small messages into a single large message
    - This reduces the latency overhead per byte of information.
    - Packing cost is typically smaller than the latency per byte.
  - Start receives before sends
    - This eliminates unnecessary buffering.
  - Overlap computing with communication
    - For this, use non-blocking routines and check completion after computation.
    - Do not synchronize unless it is absolutely necessary (avoid *MPI_BARRIER* calls).

The last rule

- Last but not least, do not **excessively** hand-optimize your code
  - It may
    - reduce the portability of the code
    - confuse the compiler (and other programmers)

"Premature optimization is the root of all evil."

(Tony Hoare, inventor of the famous Quicksort algorithm)

Books and articles on optimization

- Optimization with the PGI compiler
  - Portland Group: PGI User's guide (http://www.pgroup.com/doc/pgiug.pdf)
- This is not for Cray XT4, but a very book nice anyway
  - IBM redbooks: *The POWER4 Processor Introduction and Tuning guide*
- A nice article about optimization of scientific and engineering codes
  - John Feo: *How I Got 15x Improvement Without Really Trying: A Case Study in Program Optimization* (http://developers.sun.com/sunstudio/articles/Feo/feo_content.htm)