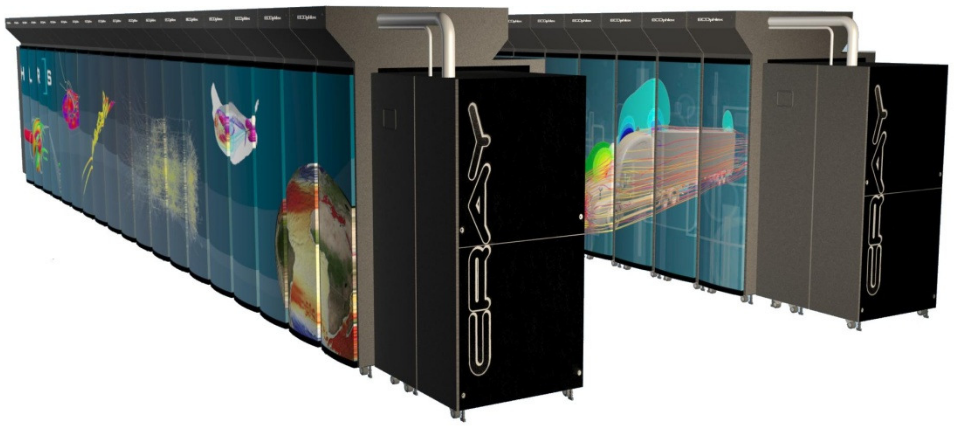
The Cray Programming Environment_2



Charles Henriet

Agenda

- Programming considerations
- Running an application
- Cray Scientific Libraries

Programming Considerations

Memory allocation

- Linux provides some environment variables to control how malloc behaves (Equivalent to using the mallopt system call)
- Returning memory to the OS is **very costly**
- MALLOC_MMAP_MAX_
  - 64 mmap regions to allow a program to return unused memory back to the system
  - no need of these regions on the XE6
  - Suggested value: `export MALLOC_MMAP_MAX_=0`

THE SUPERCOMPUTER COMPANY

Memory allocation

- MALLOC_TRIM_THRESHOLD_
  - Before malloc returns memory to the OS we need free space (at the top of the heap after a free)
  - Default setting is very small : 128 Kbytes <<< 2/4 GBytes of memory available for the application
  - Suggested value:

```
export MALLOC_TRIM_THRESHOLD_=536870912
```

Huge pages - description

- virtual memory pages which are bigger than the default base page size of 4KB
- can improve memory performance for common access patterns on large data set
- increase the maximum size of data and text in a program accessible by the high speed network.
- Access to huge pages is provided through a virtual file system called ‘hugetlbfs’
  - Link with the correct library: `-lhugetlbfs`
  - Activate the library at run time: `export HUGETLB_MORECORE=yes`
- Useful man pages:
  - man aprun
  - man intro_hugepages

Huge pages - howto

- This example requests 4000 MB of huge pages per PE
  - HUGETLB_MORECORE=yes aprun -n 8 -m4000h ./xthi
- The following example requests 4000 MB of hugepages per PE, and also specifies that a hugepage size of 16 MB is to be used
  - HUGETLB_DEFAULT_PAGE_SIZE=16m aprun -n 8 -m4000h ./xthi
- The following example terminates because the required 4000 MB of huge pages per PE are not available (hs is used)
  - aprun -n 8 -m4000hs ./xthi
- Requires 1400 MBytes of huge page memory on each node
  - `aprun -m700hs -N2 -n8 ./xthi`

More about environment on XE6(1)

MPICH_SMP_SINGLE_COPY_ON=1

All on-node messages, regardless of size, are not buffered

MPICH_GNI_RDMA_THRESHOLD

Adjusts the threshold for switching to use of the Direct Memory Access engine for transferring inter-node MPI message data.

For sh, ksh, or bash:

```
export MPICH_ENV_DISPLAY=1
export MPICH_SMP_SINGLE_COPY_ON=1
export MPICH_GNI_RDMA_THRESHOLD=2048
```

For csh or tcsh:

```
setenv MPICH_ENV_DISPLAY 1
setenv MPICH_SMP_SINGLE_COPY_ON 1
setenv MPICH_GNI_RDMA_THRESHOLD 2048
```

More about environment on XE6(2)

If your program hangs or aborts with an MPI or system library error try each of these in turn or in combination.

1. Set MPICH_GNI_DYNAMIC_CONN to "disabled".

   For sh, ksh, or bash:

   ```
   export MPICH_GNI_DYNAMIC_CONN=disabled
   ```

   For csh or tcsh:

   ```
   setenv MPICH_GNI_DYNAMIC_CONN disabled
   ```

2. Remove the MPICH_SMP_SINGLE_COPY_ON env var

3. 'module swap' back to an earlier xt-mpt module and rebuild

More about environment on XE6(3)

If your code is too slow , try the following suggestions separately, or in combination.

1. Remove the MPICH_GNI_RDMA_THRESHOLD env var. No relink or rebuild needed.

2. Increase the value of MPICH_GNI_MAX_EAGER_MSG_SIZE (the default is 8192) **and t**he value of MPICH_GNI_NUM_BUFS (default is 64). No relink or rebuild needed.

3. Set MPICH_GNI_DYNAMIC_CONN to disabled. No relink or rebuild required.

4. Try increasing the MPICH_SMP_SINGLE_COPY_SIZE   The default is 2k, setting it larger may help. Try 4k, 8k, 16k. No relink or rebuild required.

5. Try using large pages; this seems to help some applications

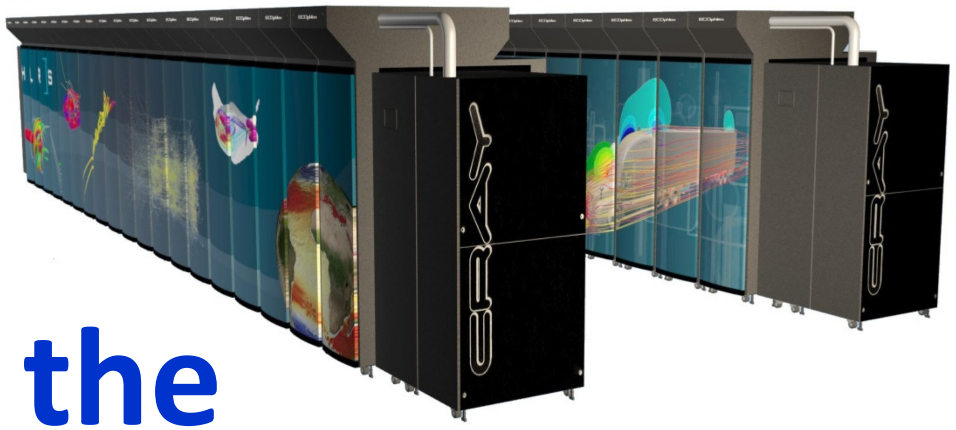
MPICH name conflict

- There is a name conflict between stdio.h and MPI C++ binding in relation to the names SEEK_SET, SEEK_CUR, SEEK_END
- If your application does not use those names:
  - work with -DMPICH_IGNORE_CXX_SEEK to come around this
- If your application does use those names:
  - Set

```
#undef SEEK_SET
<include mpi.h>
```

  - or change order of includes: mpi.h before stdio.h or iostream

Running an application on the Cray XE6



Running an application on the Cray XE

- « ALPS + aprun »
- ALPS : Application Level Placement Scheduler
- aprun is the ALPS application launcher
- aprun
  - It must be used to run application on the XT compute nodes
  - If aprun is not used, the application is launched on the login node (and might fail)
  - aprun man page contains several useful examples
  - at least 3 important parameters to control:
    - The total number of PEs :   -n
    - The number of PEs per node: -N
    - The number of OpenMP threads: -d
      More precise : The ‘stride’ between 2 PEs in a node

Running an application on the Cray XE6

- Assuming a XE6 mc8 system (16 cores per node)
- Pure MPI application, using all the available cores in a node

```
$ aprun -n <npes>
```

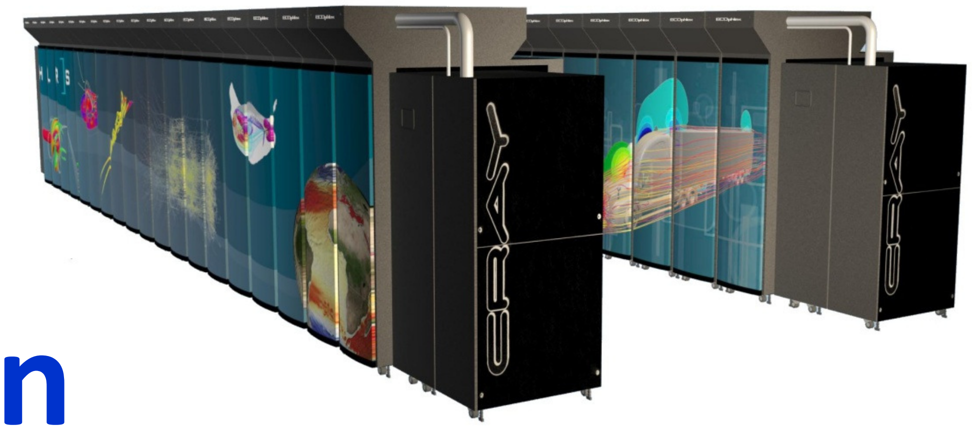
- Pure MPI application, using only 1 core per node
  - npes MPI tasks, 16*npes cores allocated, npes nodes allocated
  - Can be done to increase the available memory for the MPI tasks

```
$ aprun -N 1 -n <npes>
```

- Hybrid MPI/OpenMP application, 4 MPI ranks per node
  - npes MPI tasks, 4 OpenMP threads each
  - need to set OMP_NUM_THREADS

```
$ export OMP_NUM_THREADS=4
$ aprun -N 4 -d 4 -n <npes>
```

The application launching process



Job Launch

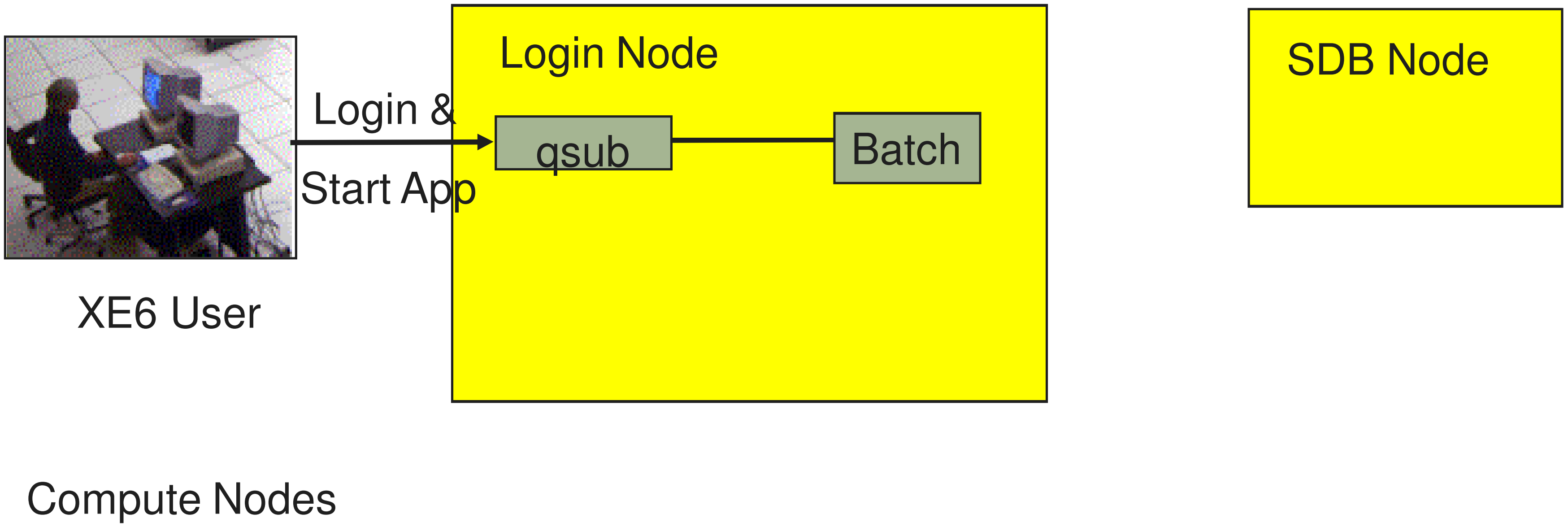
XE6 User

Login Node

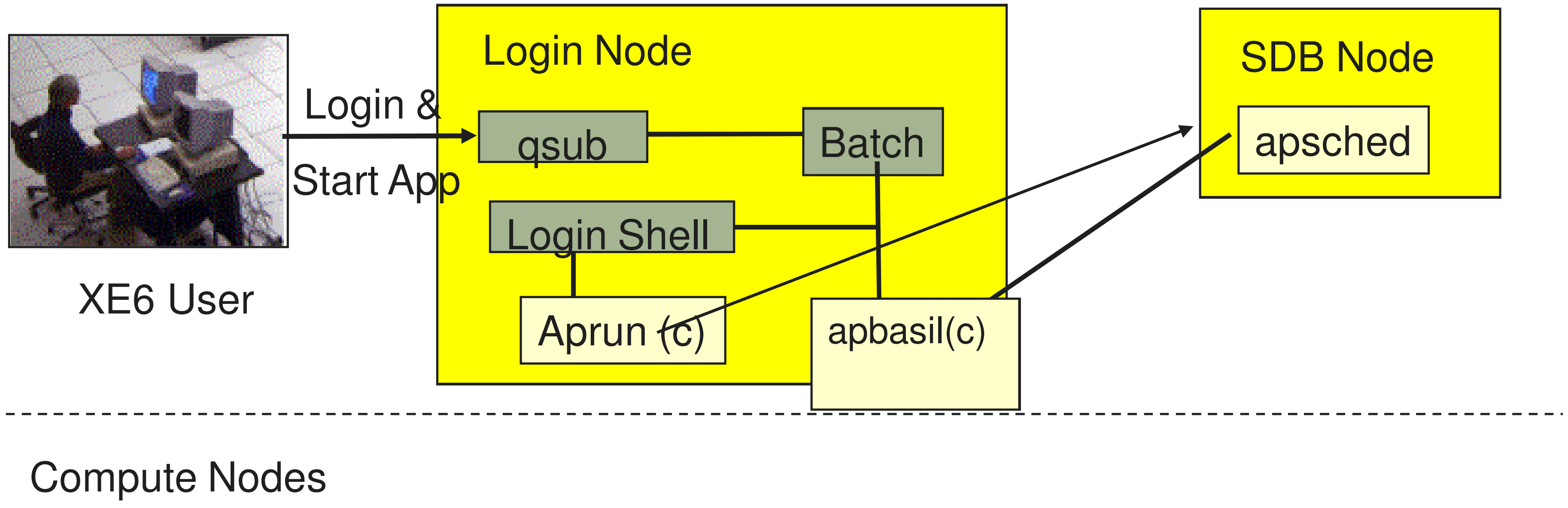
SDB Node

Compute Nodes

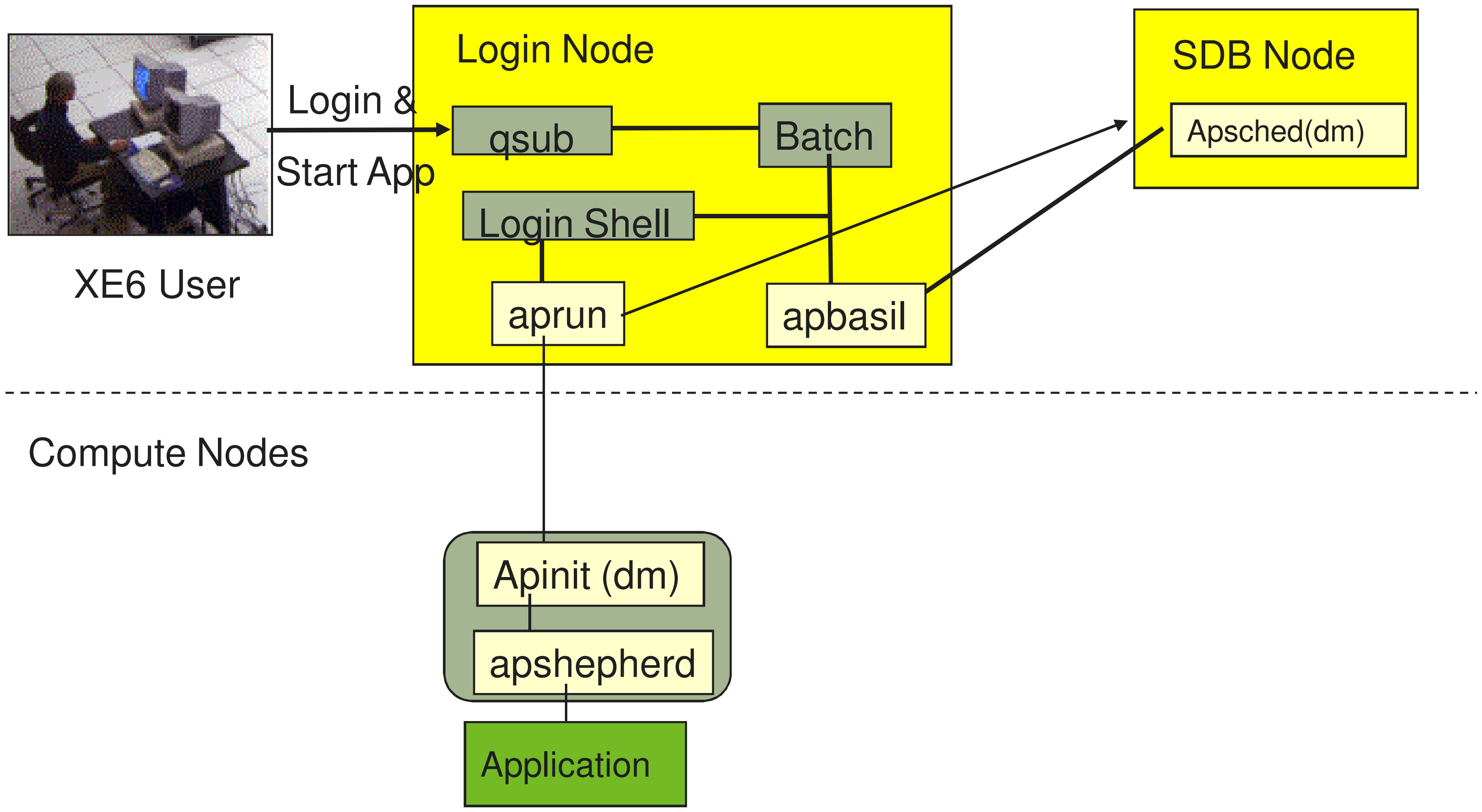
Job Launch


Login &
Start App
Login Node
qsub
Batch
SDB Node
XE6 User
Compute Nodes

Job Launch

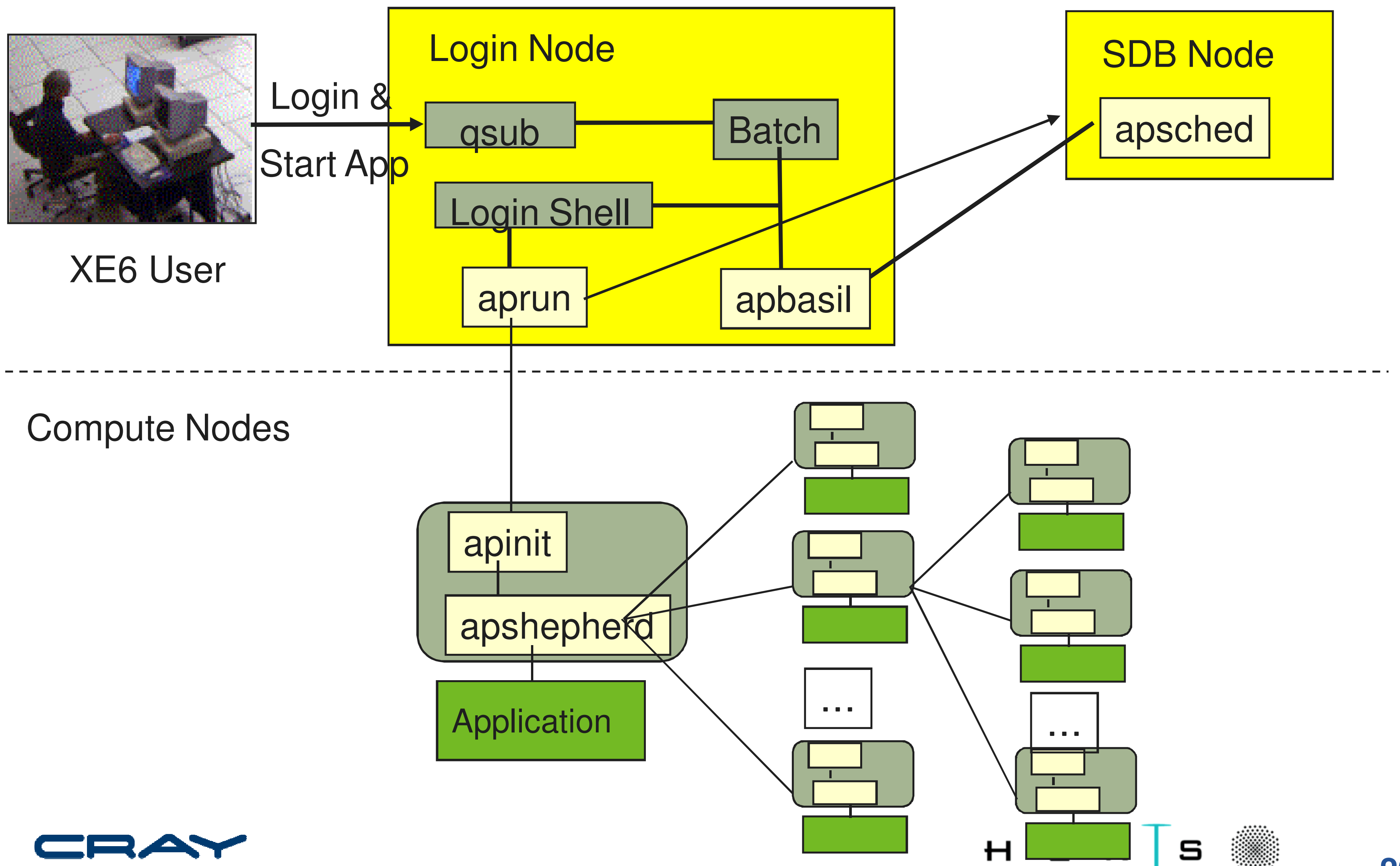

XE6 User
Login &
Start App
Login Node
qsub
Batch
Login Shell
Aprun (c)
apbasil(c)
SDB Node
apsched
Compute Nodes

Job Launch

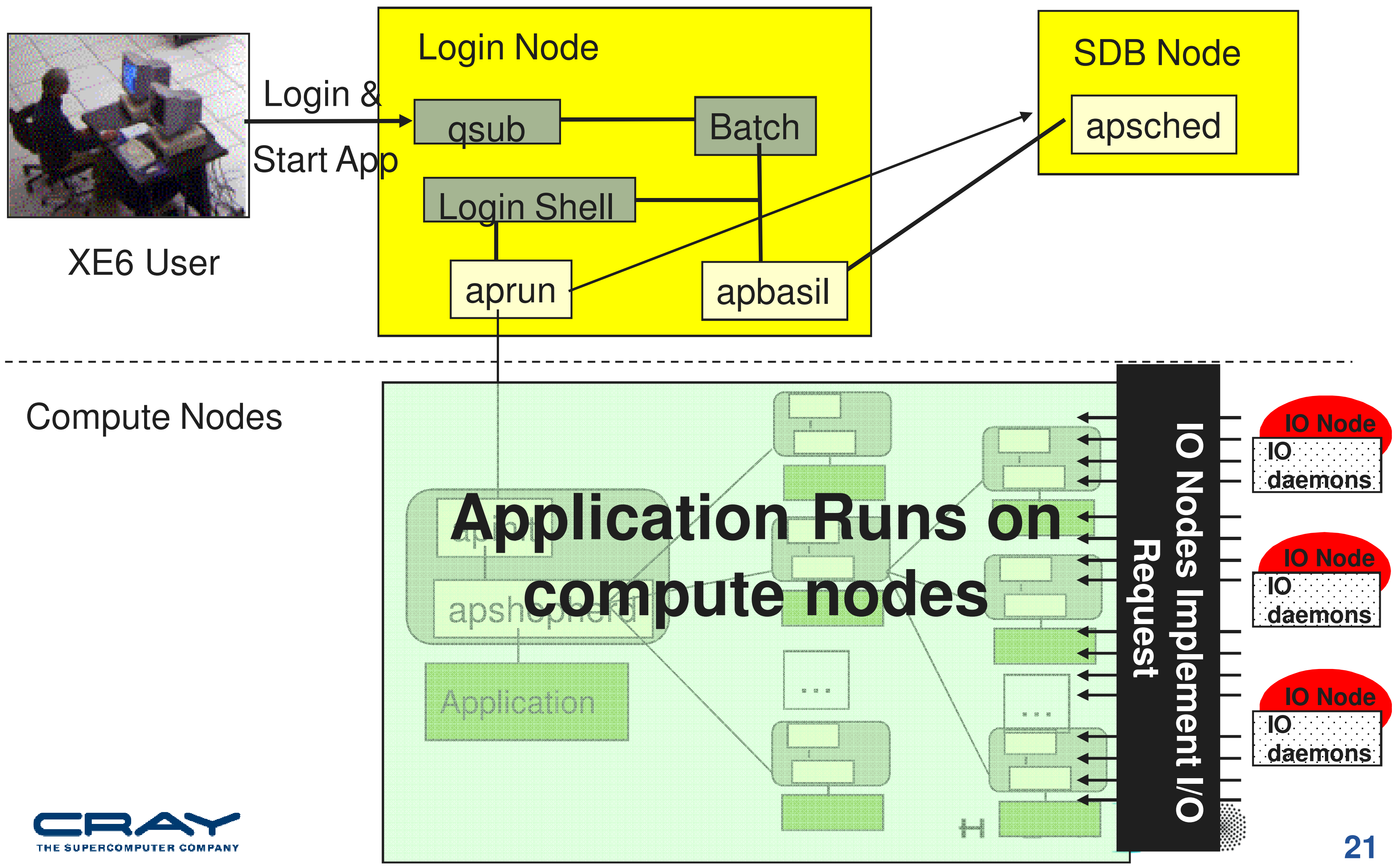


Login &
Start App
XE6 User
Login Node
qsub
Batch
Login Shell
aprun
apbasil
SDB Node
Apsched(dm)
Compute Nodes
Apinit (dm)
apshepherd
Application

Job Launch

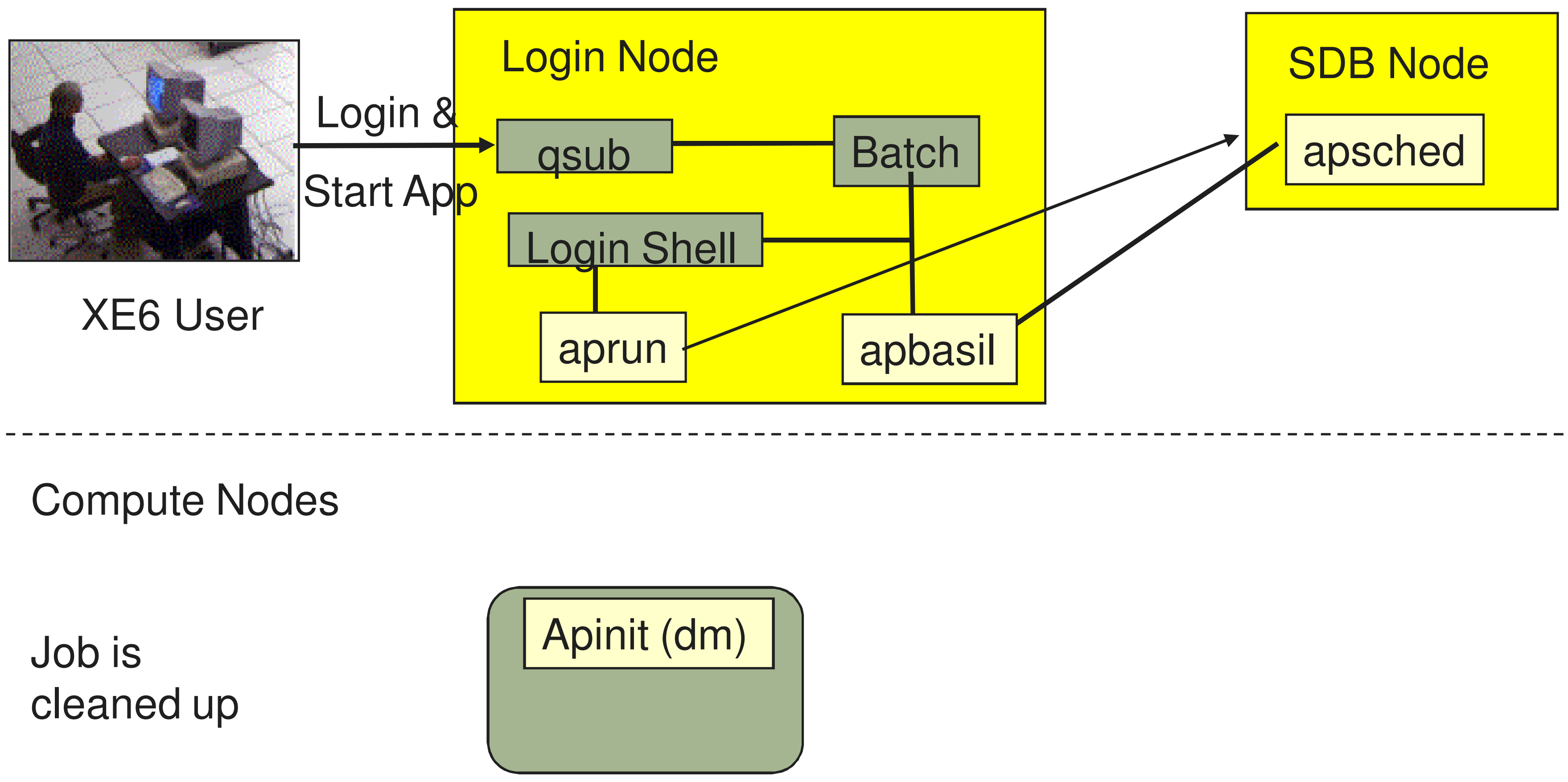

XE6 User
Login &
Start App
Login Node
qsub
Batch
Login Shell
aprun
apbasil
SDB Node
apsched
Compute Nodes
apinit
apshepherd
Application
...
...

Job Launch

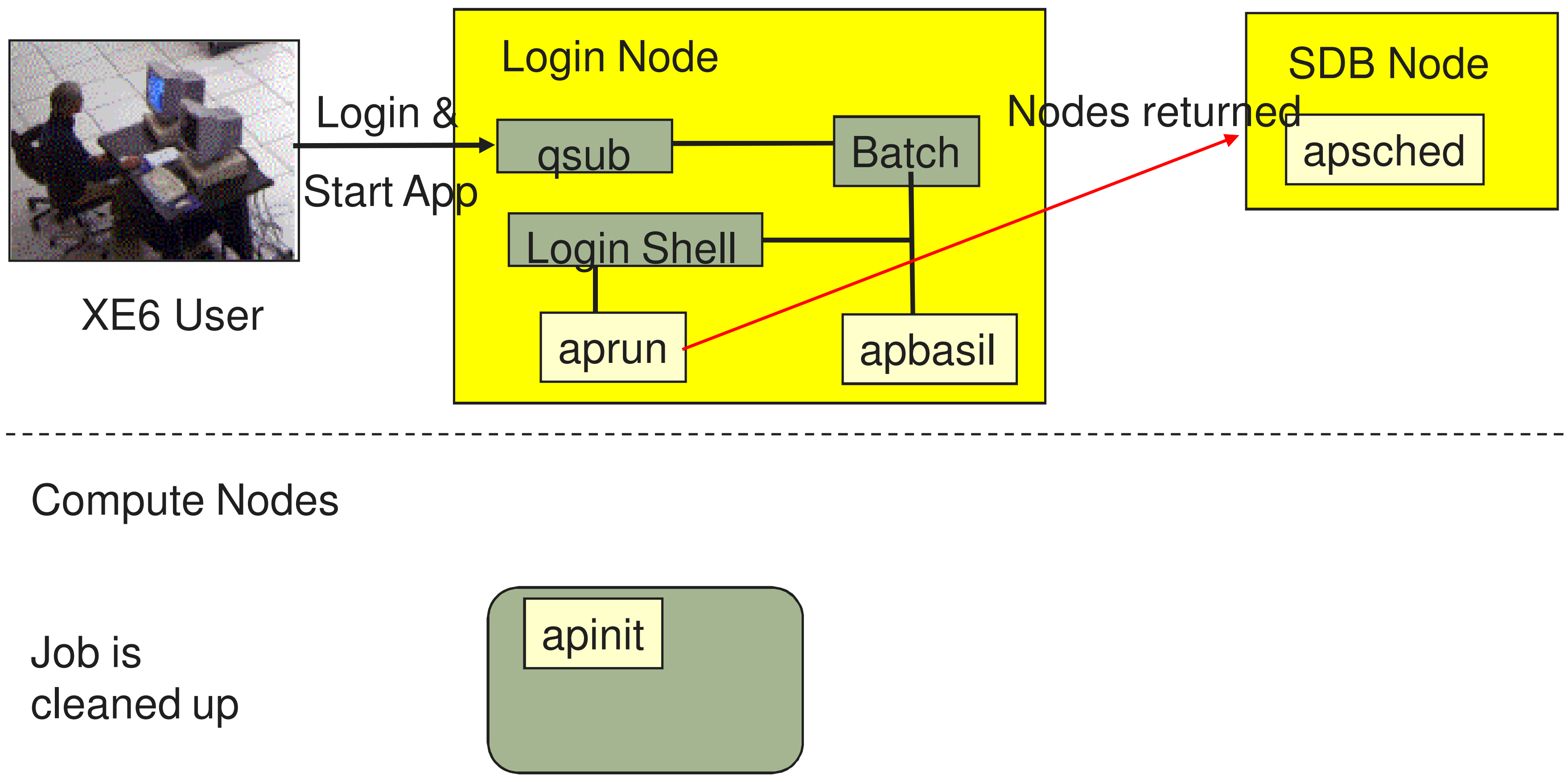


CRAY THE SUPERCOMPUTER COMPANY

Job Launch


Login &
Start App
XE6 User
Login Node
qsub
Batch
Login Shell
aprun
apbasil
SDB Node
apsched
Compute Nodes
Job is
cleaned up
Apinit (dm)

Job Launch



Login &
Start App
XE6 User
Login Node
qsub
Batch
Login Shell
aprun
apbasil
Nodes returned
SDB Node
apsched
Compute Nodes
Job is
cleaned up
apinit

Job Launch : Done

XE6 User

Login Node

SDB Node

Compute Nodes

aprun CPU Affinity control

- CNL can **dynamically** distribute work by allowing PEs and threads to migrate from one CPU to another within a node
- In some cases, moving PEs or threads from CPU to CPU increases cache and translation lookaside buffer (TLB) misses and therefore **reduces** performance
- CPU affinity options enable to bind a PE or thread to a particular CPU or a subset of CPUs on a node
- aprun CPU affinity option (see man aprun)
  - suggested settings: -cc cpu (default)
  - The cpu keyword binds each PE to a CPU within the assigned NUMA node

aprun CPU Affinity control

- Pathscale compiler provide its own control of cpu affinity which is turned on by default :
  this should be disabled to avoid interference with ALPS
  - export PSC_OMP_AFFINITY=FALSE
- The Intel RTE starts an extra thread when using OpenMP threads. This confuses the defaults ALPS affinity control
  - aprun –cc numa_node
  - aprun –cc none

Further aprun affinity control

- Cray XE6 systems use dual-socket compute nodes
  - Each die (4 or 6 cores) is considered a NUMA-node
- **Remote-NUMA-node memory references**, can adversely affect performance.
- aprun memory affinity options (see man aprun)
  - Suggested setting is -ss
  - -ss : a PE can allocate only the memory local to its assigned NUMA node

Running an application on the Cray XT - MPMD

- aprun supports MPMD – Multiple Program Multiple Data

- Launching several executables on the same MPI_COMM_WORLD
```
$ aprun -n 128 exe1 : -n 64 exe2 : -n 64 exe3
```

- Notice : Each exacutable needs a dedicated node, exe1 and exe2 cannot share a node.

  Example : The following commands needs 3 nodes
```
$ aprun -n 1 exe1 : -n 1 exe2 : -n 1 exe3
```

- Use a script to start several serial jobs on a node :
```
$ aprun -a xt -n 3 script.sh
```

```
>cat script.sh
./exe1&
./exe2&
./exe3&
wait
>
```

Running a batch application with Torque

- The number of required nodes and cores is determined by the parameters specified in the job header

  ```
  #PBS -l mppwidth=256
  #PBS -l mppnppn=4
  ```

  This example uses 256/4=64 nodes
- The job is submitted by the qsub command
- At the end of the exection output and error files are returned to submission directory
- PBS environment variable: $PBS_O_WORKDIR
  Set to the directory from which the job has been submitted
- man qsub for env. variables

Other Torque options

- #PBS -N job_name

  the job name is used to determine the name of job output and error files
- #PBS -l walltime=hh:mm:ss

  Maximum job elapsed time

  should be indicated whenever possible: this allows Torqu to determine best scheduling startegy
- #PBS -j oe

  job error and output files are merged in a single file
- #PBS -q queue

  request execution on a specific queue

THE SUPERCOMPUTER COMPANY

Core specialization

- System 'noise' on compute nodes may significantly degrade scalability for some applications
- Core Specialization can mitigate this problem
  - 1 core per node will be dedicated for system work (service core)
  - As many system interrupts as possible will be forced to execute on the service core
  - The application will not run on the service core
- Use aprun -r to get core specialization

```
$ aprun -r -n 100 a.out
```

- apcount provided to compute total number of cores required

```
$ qsub -l mppwidth=$(apcount -r 1 1024 16)job
  aprun -n 1024 -r 1 a.out
```

Running a batch application with Torque

- The number of required nodes can be specified in the job header
- The job is submitted by the qsub command
- At the end of the exection output and error files are returned to submission directory
- Environment variables are inherited by **#PBS -V**
- The job starts in the home directory. $`PBS_O_WORKDIR` contains the directory from which the job has been submitted

Hybrid MPI + OpenMP

```
#!/bin/bash
#PBS -N hybrid
#PBS –lwalltime=00:10:00
#PBS –lmppwidth=128
#PBS –lmppnppn=4

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=4
aprun –n32 –d4 –N4 a.out
```

Starting an interactive session with Torque

- An interactive job can be started by the –I argument
  - That is <capital-i>

- Example: allocate 4 nodes on a mc8 system and exporting the environment variables to the job (-V)

```
$ qsub -I -V -lmppwith=64 -lmppnppn=16
```

Watching a launched job on the Cray XE

- **xtnodestat**
  - Shows XE nodes allocation and aprun processes
  - Both interactive and PBS
- **apstat**
  - Shows aprun processes status
  - apstat overview
  - apstat –a[apid]info about all the applications or a specific one
  - apstat –n info about the status of the nodes
- Batch **qstat** command
  - shows batch jobs

Accounting at HLRS

- Currently the XE6 is run in test mode, it‘s free to use
  - Accounting is allready enabled for testing purposes
- Accounting is done by examining the Torque log files and is based on the unix group id a user belongs to
  - Normally the user don‘t have to do anything
- If a user is involved in several projects, he has to select the correct one by setting the group id in the batch script :
  - #PBS -W group_list=<group name>

Starting 512 MPI tasks (PEs)

```
#PBS -N MPIjob
#PBS -l mppwidth=512
#PBS -l mppnppn=16
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

aprun  -n 512 -cc cpu -ss ./a.out
```

Starting an OpenMP program

```
#PBS -N OpenMP
#PBS -l mppwidth=1
#PBS -l mppdepth=16
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=16

aprun -n1 -d $OMP_NUM_THREADS -cc cpu -ss ./a.out
```

Starting a hybrid job
single node, 4 MPI tasks, each with 4 threads

```
#PBS -N hybrid
#PBS -l mppwidth=4
#PBS -l mppnppn=4
#PBS -l mppdepth=4
#PBS -l walltime=01:00:00
#PBS -j oe

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=4

aprun -n4 -N4 -d $OMP_NUM_THREADS -cc cpu -ss ./a.out
```

CRAY THE SUPERCOMPUTER COMPANY HLRS

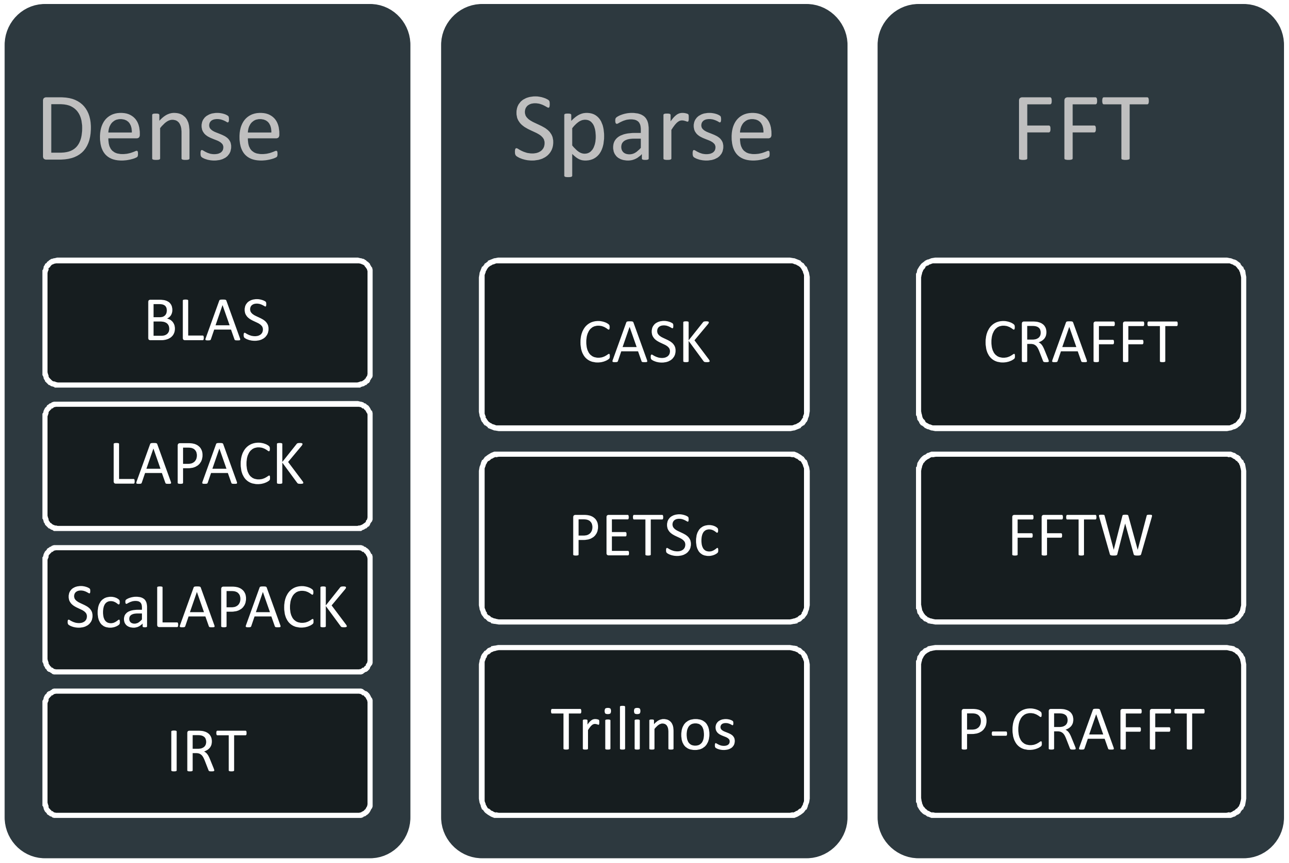
Starting a MPMD job on a non-default projectid using 1 master, 16 slaves, each with 4 threads

```
#PBS -N hybrid
#PBS -l mppwidth=80 ! Note : 5 nodes * 16 cores = 80 cores
#PBS -l mppnppn=16
#PBS -l walltime=01:00:00
#PBS -j oe
#PBS -W group_list=My_Project

cd $PBS_O_WORKDIR

export MPICH_ENV_DISPLAY=1

export MALLOC_MMAP_MAX_=0
export MALLOC_TRIM_THRESHOLD_=536870912

export OMP_NUM_THREADS=4
id # Unix command ‚id`, to check group id

aprun -n1 -d16 -N1 ./master.exe : -n 16 -N4 -d
  $OMP_NUM_THREADS  -cc cpu -ss ./slave.exe
```

THE SUPERCOMPUTER COMPANY

Cray Scientific Libraries (Libsci)

Cray Scientific/Math Libraries



Dense

Bend over backwards to keep everything the same despite increases in machine complexity.

- **LAPACK, ScaLAPACK**
  - Algorithmic tuning : increased performance by exploiting algorithmic improvement : Sub-blocking, new algorithms
- **BLAS, FFT**
  - Kernel tuning: Improve the numerical kernel performance in assembly language
- **ScaLAPACK**
  - Parallel tuning : exploit Cray's custom network interfaces and MPT

IRT

- Iterative Refinement Toolkit
- Solves linear systems in single precision
- Obtaining solutions accurate to double precision
  - For well conditioned problems
- Serial and Parallel versions of LU, Cholesky, and QR
- 2 usage methods
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable IRT_USE_SOLVERS =1
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
      - Allows
        - condition number estimation
        - error bounds return
        - minimization of either forward or backward error
        - 'fall back' to full precision if the condition number is too high
        - max number of iterations can be altered by users
- man intro_irt

Sparse

Adoption of near-standard interfaces/Assume near-standards and promote those:

PETSc

- Portable, Extensible Toolkit for Scientific Computation
  http://www-unix.mcs.anl.gov/petsc/petsc-as
- Serial and Parallel versions of sparse iterative linear solvers
- Large user community: DoE Labs, PSC, CSCS, CSC, ERDC, AWE and more.
- To use Cray-PETSc on CRAY XE :
  - module load petsc or module load petsc-complex
  - no need to load a compiler specific module
  - treat the Cray distribution as your local PETSc installation
- Cray provides state-of-the art scientific computing packages to strengthen the capability of PETSc
  - Hypre: scalable parallel preconditioners
  - ParMetis: parallel graph partitioning package
  - MUMPS: parallel multifrontal sparse direct solver
  - SuperLU: sequential version of SuperLU_DIST

Trilinos

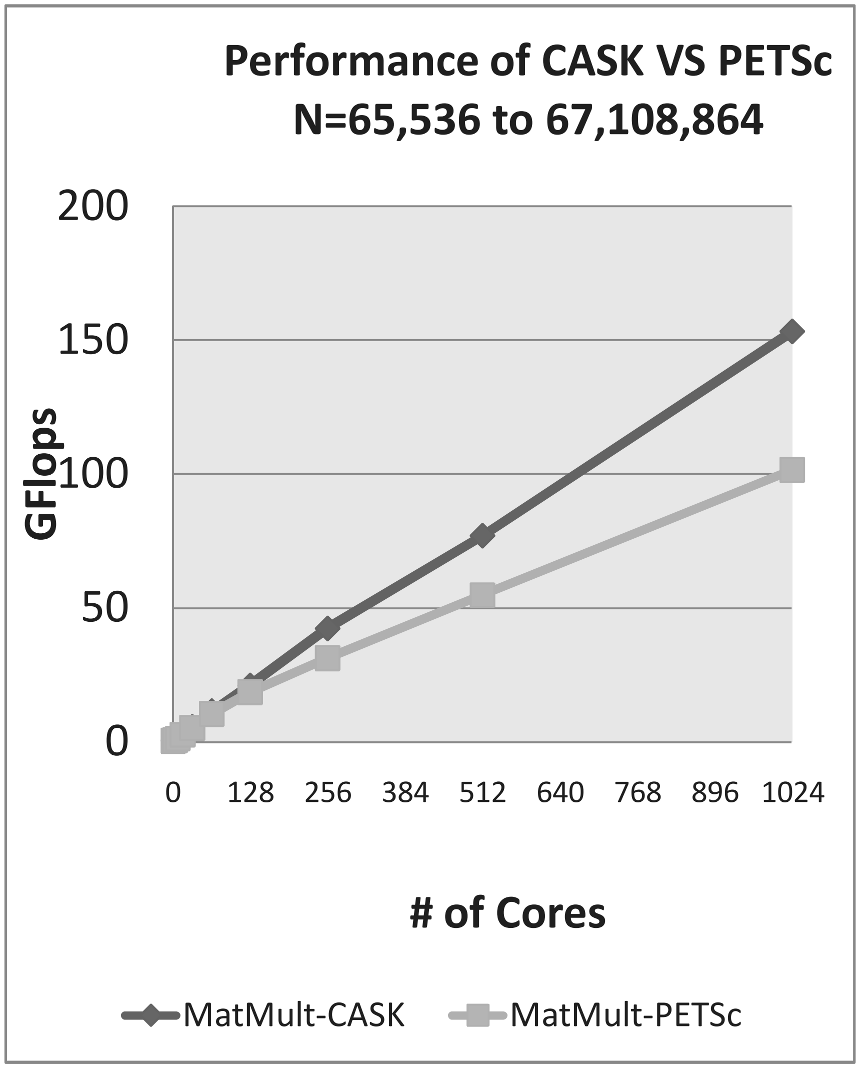
- The Trilinos Project
  - http://trilinos.sandia.gov/
  - “an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems”
- A unique design feature of Trilinos is its focus on packages.
- Very large user-base and growing rapidly.
- Cray’s optimized Trilinos released
  - Includes 50+ trilinos packages
  - Optimized via CASK
  - Any code that uses Epetra objects can access the optimizations
- Usage : module load trilinos

CASK

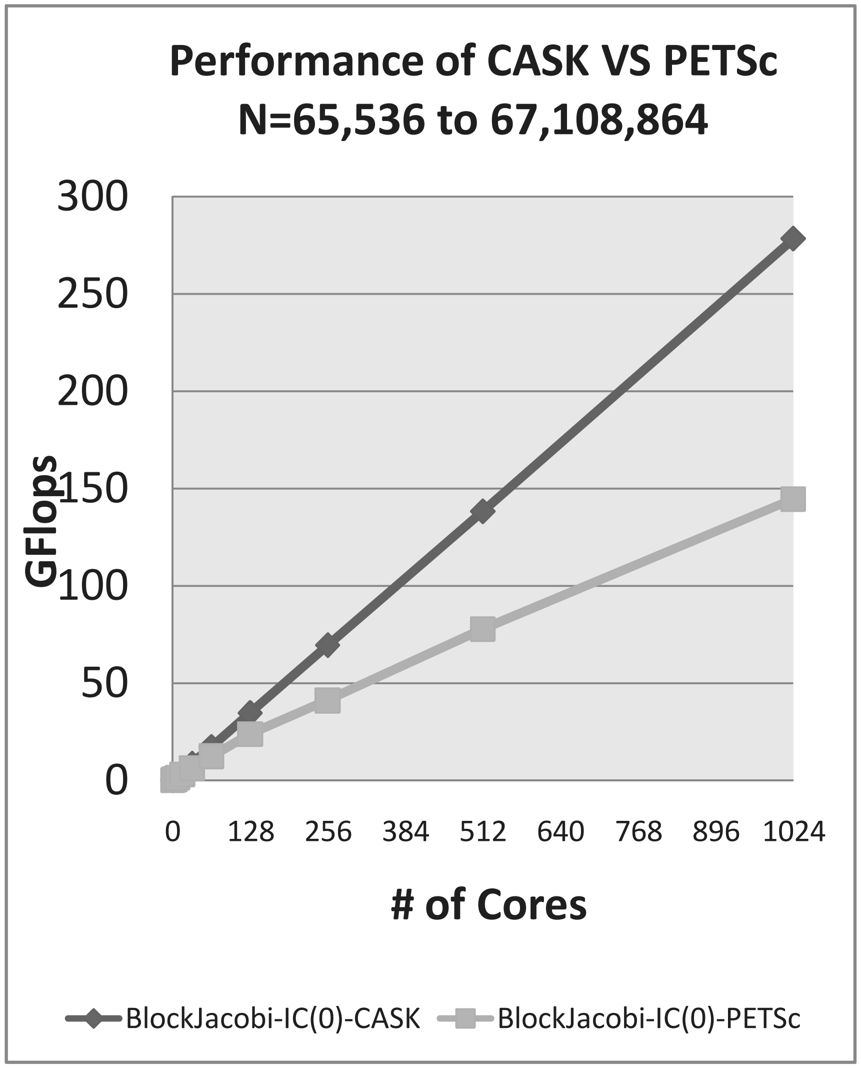
- Cray Adaptive Sparse Kernel
- CASK is a product developed at Cray using the Cray Auto-tuning Framework (Cray ATF)
- The CASK Concept :
  - Analyze matrix at minimal cost
  - Categorize matrix against internal classes
  - Based on offline experience, find best CASK code for particular matrix
  - Previously assign "best" compiler flags to CASK code
  - Assign best CASK kernel and perform Ax
- CASK silently sits beneath PETSc on Cray systems
  - Trilinos support coming soon

CASK + PETSc Scalability (XT4)

SpMV



Block Jacobi Preconditioning



CRAY THE SUPERCOMPUTER COMPANY

HLRS

Cray Adaptive FFT (CRAFFT)

- In FFTs, the problems are
  - Which library choice to use?
  - How to use complicated interfaces (e.g., FFTW)

- Standard FFT practice
  - Do a plan stage
    - Deduced machine and system information and run micro-kernels
    - Select best FFT strategy
  - Do an execute

Our system knowledge can remove some of this cost!

CRAFFT library

- CRAFFT is designed with simple-to-use interfaces
  - Planning and execution stage can be combined into one function call
  - Underneath the interfaces, CRAFFT calls the appropriate FFT kernel

- CRAFFT provides both offline and online tuning
  - Offline tuning
    - Which FFT kernel to use
    - Pre-computed PLANs for common-sized FFT
      - No expensive plan stages
  - Online tuning is performed as necessary at runtime as well

- At runtime, CRAFFT will **adaptively select the best** FFT kernel to use based on both offline and online testing (e.g. FFTW, Custom FFT)

CRAFFT usage

1. Load module fftw/3.2.0 or higher.
2. Add a Fortran statement “use crafft”
3. call crafft_init()
4. Call crafft transform using none, some or all optional arguments (as shown in red)

   In-place, implicit memory management :

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign)
```

   in-place, explicit memory management

```
call crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,isign,work)
```

   out-of-place, explicit memory management :

```
crafft_z2z3d(n1,n2,n3,input,ld_in,ld_in2,output,ld_out,ld_out2,isign,work)
```

Note : the user can also control the planning strategy of CRAFFT using the **CRAFFT_PLANNING environment variable** and the do_exe optional argument, please see the intro_crafft man page.

Performance of one CRAFFT feature -
3-d FFT times using FFTW wisdom under-the-covers

	128x128	256x256	512x512
FFTW plan	74	312	2758
FFTW exec	0.105	0.97	9.7
CRAFFT plan	0.00037	0.0009	0.00005
CRAFFT exec	0.139	1.2	11.4

Parallel CRAFFT

- CRAFFT includes distributed parallel transforms
- Uses the CRAFFT interface prefixed by “p”, with optional arguments
- Can provide performance improvement over FFTW 2.1.5
- Currently implemented
  - complex-complex
  - Real-complex and complex-real
  - 3-d and 2-d
  - In-place and out-of-place
- Upcoming
  - C language support for serial and parallel

parallel CRAFFT usage

1. Add “use crafft” to Fortran code
2. Initialize CRAFFT using crafft_init
3. Assume MPI initialized and data distributed
4. Call crafft, e.g. (optional arguments in red)

   2-d complex-complex, in-place, internal mem management :

   ```
   call crafft_pz2z2d(n1,n2,input,isign,flag,comm)
   ```

   2-d complex-complex, in-place with no internal memory :

   ```
   call crafft_pz2z2d(n1,n2,input,isign,flag,comm,work)
   ```

   2-d complex-complex, out-of-place, internal mem manager :

   ```
   call crafft_pz2z2d(n1,n2,input,output,isign,flag,comm)
   ```

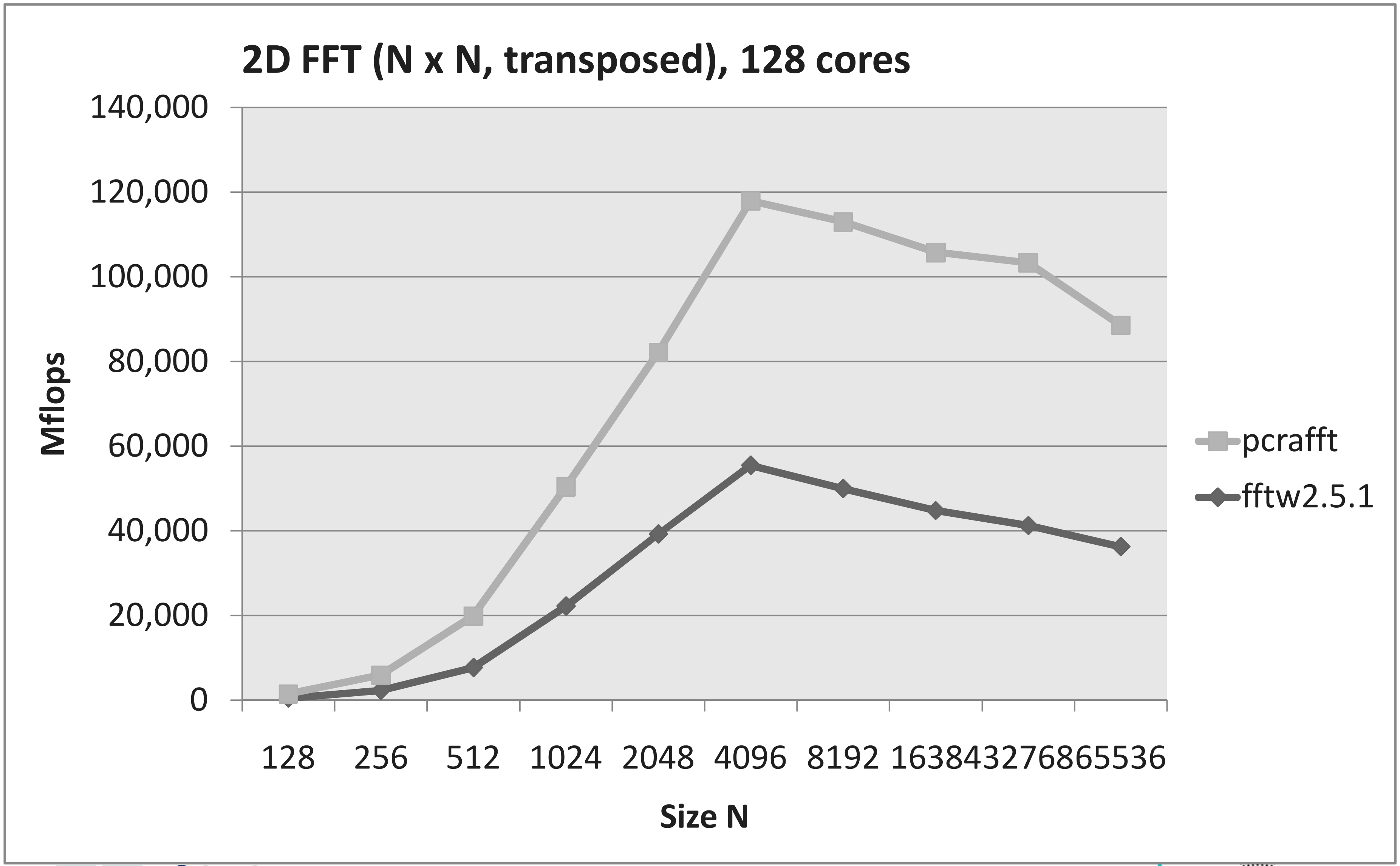
   2-d complex-complex, out-of-place, no internal memory :

   ```
   crafft_pz2z2d(n1,n2,input,output,isign,flag,comm,work)
   ```

Each routine above has manpage.

Also see 3d equivalent : `man crafft_pz2z3d`

Parallel CRAFFT performance



Documentation

- Cray docs site

  http://docs.cray.com

- Starting point for Cray XE info

http://docs.cray.com/cgi-bin/craydoc.cgi?mode=SiteMap;f=xe_sitemap

- Twitter ?!?

  http://twitter.com/craydocs

End Part_2