Cray XT for Dummies
(A Reference for the Rest of Us)

Jeff Larkin
Cray Supercomputing Center of Excellence
larkin@cray.com

NCCS Users Meeting 2007

CRAY
THE SUPERCOMPUTER COMPANY

Building your executable

- Choosing your Compiler
  - By default, you will be using PGI
  - To change from PGI to GNU
    - `module swap PrgEnv-pgi PrgEnv-gnu`
  - To change from GNU to PGI
    - `module swap PrgEnv-gnu PrgEnv-pgi`
  - If you do not swap these modules you could link against the wrong libraries!
- Using the Cray Compiler Wrappers
  - Fortran: `ftn <-target=catamount>`
  - C: `cc <-target=catamount>`
  - C++: `CC <-target=catamount>`
  - While using the mpi*** scripts may work, it is not recommended

PGI Compiler Options

- Option help: -help <option>
  - Must call PGI compiler directly for –help.
- Listing file: -Mlist (sorry, not as nice as the X1E)
- Additional compile-time information: -Minfo,-Mneginfo
  - =[inline|ipa|loop|opt|stat|time|all]
- -byteswapio: Swap byte-order for unformatted input/output
  - Useful when moving data between jaguar and phoenix
  - Can be used file-by-file, if you want to read one endianness and write the other
- -r[4,8] –i[2,4,8]: Controls interpretation of real and integer sizes.
  - This may affect library compatibility.
  - Use –r8 with –default64 to get proper system libraries

PGI Optimization Flags

- -O0-4: Set optimization level, -O0 to -O4, default -O2
- -fast: “Good” optimizations
  - -O2 -Munroll=c:1 -Mnoframe -Mlre
- -fastsse: “Good” SSE optimizations
  - -fast -Mvect=sse -Mscalarsse -Mcache_align –Mflushz
  - You should be striving for at least this level for future multi-core processors
- More Advanced Options
  - -Mvect: Control automatic vector pipelining
  - -Mipa: Enable Interprocedural Analysis
  - -Mscalarsse: Generate scalar sse code with xmm registers; implies –Mflushz (processor-level flush xmm to zero)
  - -Munroll: Enable loop unrolling
  - -Minline: Inline all functions that were extracted

If you must ./configure...

- Most configure scripts are confused by cross-compilers
- First thing to try:
  - `./configure CC=pgcc F90=pgf90 MPICC=cc MPIF90=ftn`
  - May also support `–with-cc=pgcc –with-mpicc=cc`
  - If you want to use CrayPAT, you'll need to edit the Makefile to the cray wrappers
- Next try
  - `./configure CC=pgcc F90=pgf90`
  - Edit macros.make, if available, or the makefiles to change the compilers to cc/CC/ftn
    - Sed is your friend if you must do this.
  - Works well when there's no MPI tests
- Neither of these will report an error when checking a libc call not supported by catamount
- You may also be able to use a `--host=x86_64-unknown-linux-gnu` flag to force cross-compilation
  - Many configure scripts die with an error when you do this.
- Some configure scripts can be edited to launch tests with and application launcher, but this is the best case, but extremely rare.

Running your job

- The local XT system does not have any interactive nodes, you must use PBS
- Remember that for dual-core `-lsize=(N/2)` where N is the total number of cores you wish to use.
- The XT application launcher is `yod`
  - `-sz` – The number of MPI processes
  - `-VN` – States that you want to use dual core mode
  - `-SN` – States that you want to use single core mode
    - All of the node memory is available to you from one core
    - You are charged by socket, not by core
  - `-small_pages` – Use small memory pages, rather than large
    - Can significantly improve TLB misses
    - Almost every code will benefit from using this flag
    - Almost no code will be hurt by this

Program Execution: Useful Commands

- qsub: submit a job to pbs
- qalter: Change the parameters of a submitted job
- qstat: Show status of pbs batch jobs
- qdel: Delete your job from the queue
- xtshowmesh/xtshowcabs: Shows information about compute and service partition processors and the jobs running in each partition
- showq: Like qstat, but with more information from the scheduler
- showstart: Estimates when your job will run
  - Try ~larkin/scripts/myss.sh
- showbf: Shows current backfill
- checkjob: Get more information about a specific job
  - This is attempt to tell you why your job isn't running.

Useful MPI Environment Variables

- MPICH_RANK_REORDER_METHOD
  - Default: (0,4)(1,5)(2,6)(3,7)
  - Set to 1: (0,1)(2,3)(4,5)(6,7) **Most apps seem to like this**
  - Set to 2: (0,7)(1,6)(2,5)(3,4)
  - Set to 3: User defined mapping (see man mpi)
- MPI_COLL_OPT_ON
  - Some MPI collectives could run faster
- MPICH_FAST_MEMCPY
  - Improves the performance of memcpy between 2 nodes on the same socket, especially for messages larger that 256K.
- The MPI man page lists other more advanced variables to tweak the performance.

Debugging Tools: totalview

- Totalview is available on Jaguar
- To use: module load totalview
- Core file analysis
  - `$ tv7 a.out core`
- Launch from within interactive session
  - `$ tv7 <tvopts> yod -a <yodopts> a.out <opts>`
  - Note: -a is actually a totalview option, not a yod option, which signifies that all options following should be passed to yod.
- Attach to running processes
  - Launch totalview as above
  - From within totalview, attach to your yod process.
- Totalview Users' Guide: http://www.etnus.com/Documentation/index.php

Nodeinfo

- If you don't specify that you want just XT3 or XT4 you may get a mix
- At the moment, there's no way to figure out your node breakdown after a run
- Before your run, you can use nodeinfo to get critical information about your nodes
- Add the following to your PBS script
  - `yod -VN ~larkin/xt3/nodeinfo/nodeinfo`

CrayPAT for Dummies
(A Reference for the Rest of Us)

Jeff Larkin

Cray Supercomputing Center of Excellence

larkin@cray.com

CRAY
THE SUPERCOMPUTER COMPANY

The Basics

1. module load craypat
2. make clean
3. make
   - You must rebuild in order to ensure that the proper symbols are retained during the build
4. pat_build <-w –u –g <group>> exe <exe+pat>
   - Right now tracing is the only option on XT (sampling coming very soon)
   - -u will trace all user functions
   - -g : mpi, io, and heap are useful choices (blas & lapack coming soon)
   - -T: more advanced, trace/don't trace individual entry points
   - Original .o files must remain
5. yod exe+pat
   - Run just like your normally do
6. pat_report whatever.xf
   - See man pat_report for details on options

Pat_Report

- -O
  - ca+src : callers w/ source numbers
  - ct+src : calltree w/ source numbers
  - loadbalance: load balancing information
- -f
  - txt : default, text report
  - ap2 : apprentice 2 input file (can be archived)
  - xml : XML, can also be archived
- Default options are shown at the top of the report and provide a good starting place
- See man pat_report for more options

Gathering Hardware counters

- Build and instrument like before
- Set the PAT_RT_HWPC environment variable
  - 1-9 groups available or you can define specific counters
    1. FP, LS, L1 Misses, & TLB Misses
    2. L1 & L2 Data Accesses & Misses
    3. L1 Accesses, Misses, & bandwidth
    4. Floating Point Mix
    5. Floating Point Mix (2)
    6. Total cycles stalled
    7. Total cycles stalled (2)
    8. Instructions & Branches
    9. Instruction cache
  - See man hwpc for the complete list
- Rerun your executable
- Run pat_report as before

The PAT API

- You can define specific regions within your code that you would like to instrument
- Regions work just like functions
- Include pat header file
  - C: #include <pat_api.h>
  - FORTRAN: #include <pat_apif.h>
- pat_region_begin(1,”Loop1”,ierr)
  - Don’t forget ierr if you’re in FORTRAN, ierr is function return value in C
  - Region numbers must be unique, numbers can start at 1
- pat_region_end(1,ierr)
- You can also turn profiling off/on with pat_profiling_state()
- See man pat_build for API information

Apprentice 2 (When Pictures speak louder than Words)

- Build, instrument, and run as before
- pat_report –f ap2 –o whatever.ap2 whatever.xf
- module load apprentice2
- app2 whatever.ap2

- Note: Apprentice2 input files are also a useful way to archive your profile data and can be used as input to pat_report.