BLUE WATERS

SUSTAINED PETASCALE COMPUTING

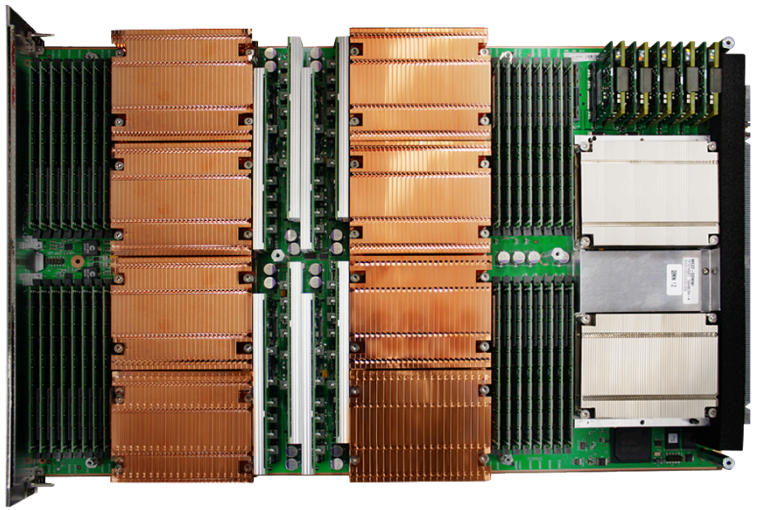
May 22, 2013

Optimizing Applications on Blue Waters

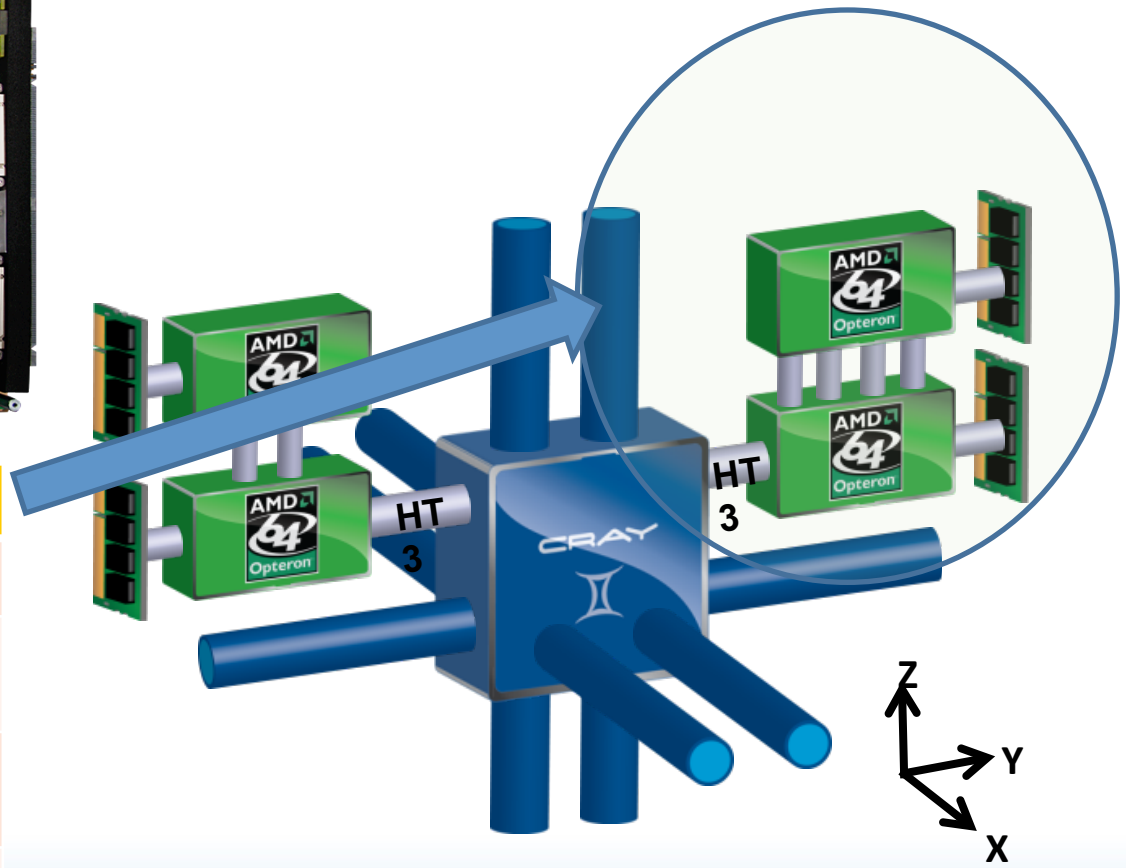
Robert Brunner, Galen Arnold, Victor Anisimov,
Tom Cortese, Manisha Gajbe, Andriy Kot, NCSA

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

Cray XE6 Blade and Node

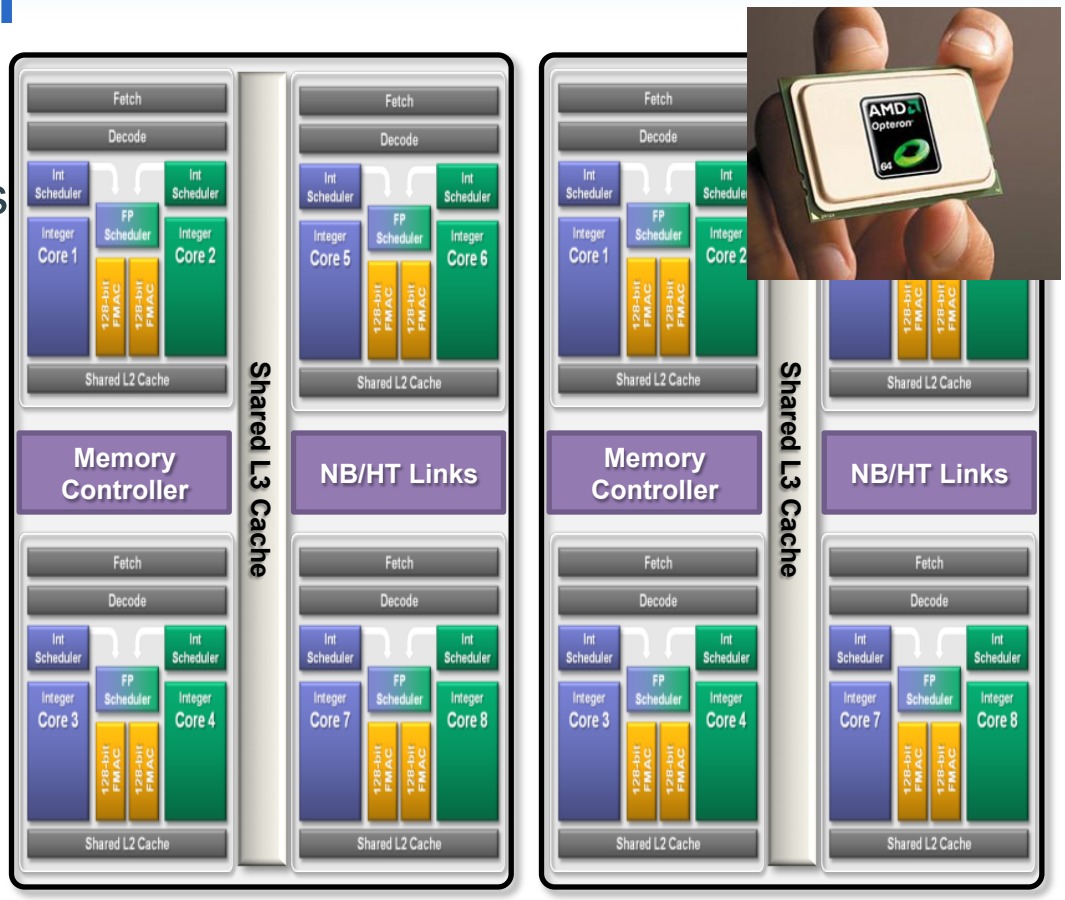


Node Characteristics	
Number of Cores	16 Core modules
Peak Performance	313 Gflops/sec
Memory Size	4 GB per core-mod 64 GB per node
Memory Bandwidth(Peak)	102.4 GB/sec



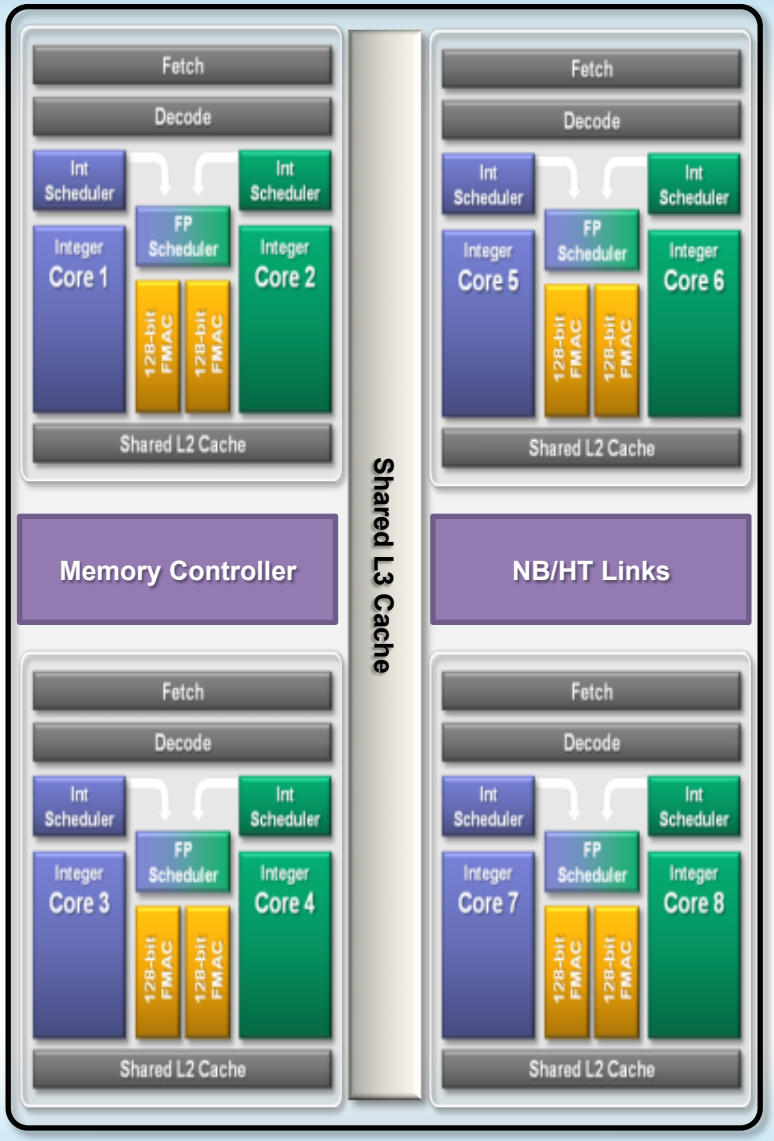
Interlagos Processor

- Each processor die is composed of 4 core modules
- The 4 core modules share a memory controller and 8 MB L3 data cache on one die
- Two die are packaged on a multi-chip module to form a G34-socket Interlagos processor
- Package contains
  - 8 core modules
  - 16 MB L3 Cache
  - 4 DDR3 1600 memory channels



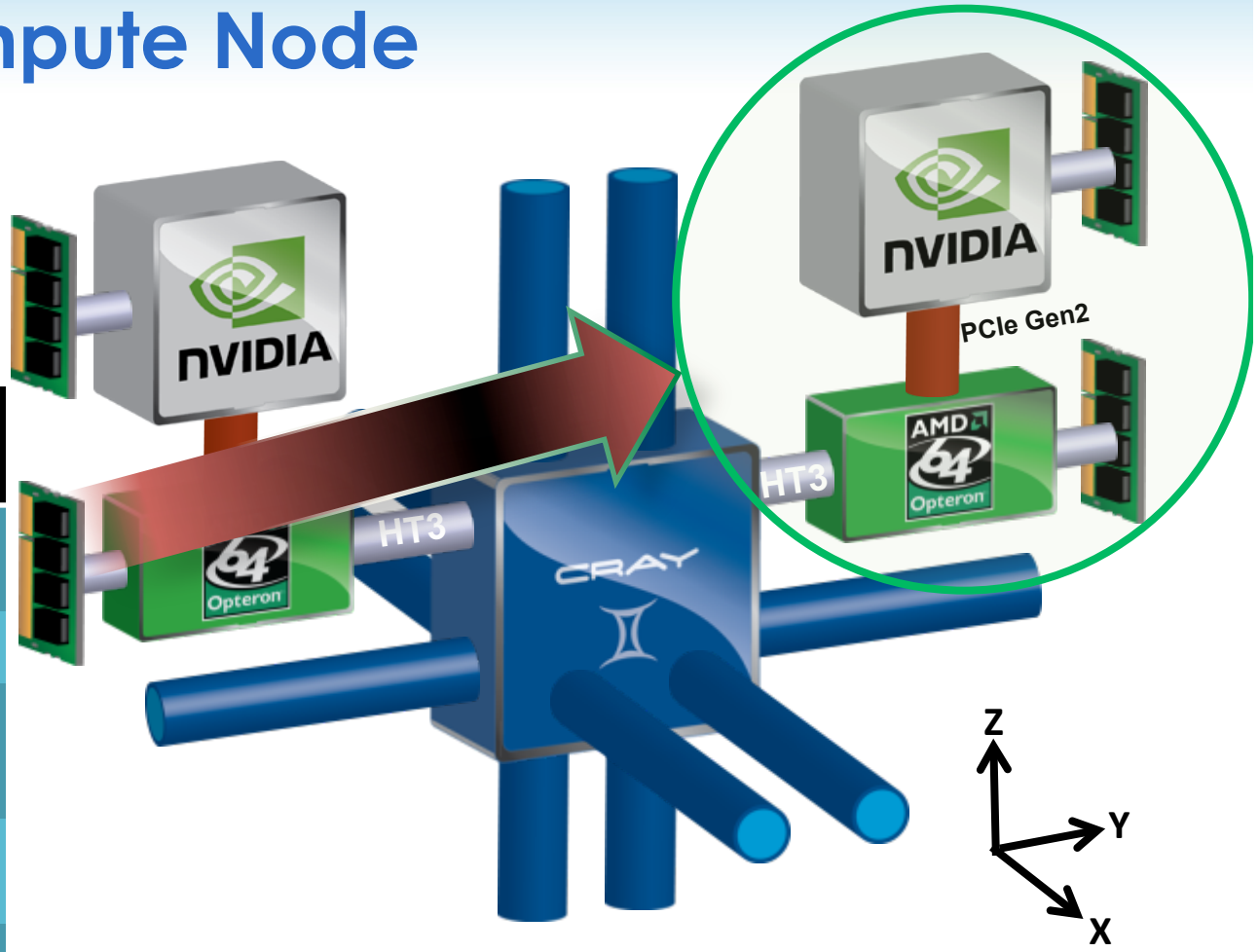
Interlagos Processor

- Four Core Modules per die
- Two Integer cores and one FP core per Core Module
- OS treats each Interlagos as 16 cores (i.e. 32 per XE6 node)
- Each die shares L3 cache



Cray XK7 Compute Node

XK7 Compute Node Characteristics
AMD Series 6200 (Interlagos) Core Module
NVIDIA Kepler
Host Memory - 32GB 1600 MHz DDR3
NVIDIA Memory 6GB GDDR5 capacity
Gemini High Speed Interconnect
Upgradeable to future GPUs



XK7 Characteristics

- GPU: NVIDIA K20X
  - 2688 processor cores
  - Processor core clock: 732 MHz
  - Memory clock: 2.6 GHz
  - Memory bandwidth: 250 GB/sec
  - 6 GB ECC RAM GPU Memory
  - Compute Capability 3.5
  - GPUDirect not supported yet, CUDA_PROXY mode
- CPU: AMD Interlagos
  - 8 Core Modules, 32 GB RAM
  - 156 GFLOPS

Compiler Options - Topics

- Available (Supported) Compilers
- Where to Start
- Compiler Choices – Relative Strength
- Compiler Options focused on
  - Optimization
  - Debugging

Available Compilers

- Cray Compilers (Cray Compiling Environment (CCE))
  - Provided additonal support for fortran 2003, Co-arrays, UPC, PGAS
- GNU Compiler Collection (GCC)
- Portland Group Inc (PGI) Compilers
- All provide Fortran, C, C++, OpenMP support
- UPC, PGAS, (limited) OpenACC support (Cray, PGI)
- So which compiler do I choose?
  - Experiment with various compilers
  - Work with your BW POC
  - Mixing libraries created by different compilers may cause issues

Where to Start

- Unless you have a very good reason, always use compiler wrappers
  - Additional libraries are automatically linked in
  - Optimization targets automatically set
- For most applications, using default settings work very well

Compilers Where to Start

- Load the proper architecture
  - For BW default : xtpe-interlagos (automatic)
  - If the module is not loaded and no arch is specified in the compiler options, the compilers default to the node type on which the compiler is running, which may not be same as the compute nodes. On BW, they are the same
  - The OpenMP threaded BLAS/LAPACK libraries are used
    - The serial version is used if “OMP_NUM_THREADS” is not set or set to 1

Use the Best Compiler

- The best compiler may not be the same for every application.
- Work with your BW POC to compare compilers

Compiler Choices – Relative Strength

- CCE – Outstanding fortran, Very good C and okay C++
  - Very good vectorization
  - Very good fortran language support; only real choice for coarrays
  - C support is very good, with UPC support
  - Very good scalar optimization and automatic parallelization
  - Clean implementation of OpenMP 3.0 with tasks
  - Cleanest integration with other Cray tools (Performance tools, debuggers, upcoming productivity tools)
  - No inline assembly support
  - Excellent support from Cray (bugs, issues, performance etc)

Compiler Choices – Relative Strength

- PGI – Very good fortran, okay C and C++
  - Good vectorization
  - Good functional correctness with optimization enabled
  - Good manual and automatic prefetch capabilities
  - Company focused on HPC market
  - Excellent working relationship with Cray, good bug responsiveness

Compiler Choices – Relative Strength

- GNU – so-so-fortran, outstanding C and C++ (If you ignore vectorization)
  - Obviously, the best gcc compatibility
  - Scalable optimizer was recently rewritten and is very good
  - Vectiorization capabilities focus mostly on inline assembly
  - Few releases have been incompatible with each other and require recompilation of modules (4.3, 4.4, 4.5)

Recommended CCE Compilation Options

- Use default optimization levels
  - It's the equivalent of most other compilers –O3 or –fast
- Use –O3, fp3 (or –O3 –hfp3 or some variation)
  - -O3 gives slightly more than –O2
  - -hfp3 gives a lot more floating point optimizations, esp 32 bit
- If an application is intolerant of floating point reassociation, try lower hfp number, try hfp1 first, only hfp0 if absolutely necessary
  - Might be needed for tests that require strict IEEE conformance
  - Or applications that have validated results from diffferent compiler
- Do not suggest using -Oipa5, -Oaggress and so on; higher numbers are not always correlated with better performance
- Compiler feedback : -rm (fortran), -hlist=m (C)
- If don't want OpenMP : -xomp or –Othread0 or –hnoomp
- Manpages : crayftn, craycc, crayCC

Loopmark : Compiler Feedback (CCE)

- Compiler can generate an filename.lst file
- Contains annotated listing of your source code with letter indicating important optimizations
- Loopmark legend

Primary Loop Type	Modifiers
A - Pattern matched	a - atomic memory operation
	b – blocked
C – Collapsed	c - conditional and/or computed
D – Deleted	
E – Cloned	f – fused
G – Accelerated	g – partitioned
I - Inlined	i – interchanged
M - Multithreaded	m – partitioned
V – Vectorized	n - non-blocking remote transfer
	p – partial
	r – unrolled
	s – shortloop
	w - unwound

Starting Point for PGI Compilers

- Suggested Option : -fast
- Interprocedural analysis allows the compiler to perform whole program optimizations : –Mipa=fast(,safe)
- If you can be flexible with precision, also try –Mfprelaxed
- Option –Msmartalloc, calls the subroutine mallopt in the main routine, can have a dramatic impact on the performance of program that uses dynamic allocation of memory
- Compiler feedback : -Minfo=all, -Mneginfo
- Manpages : pgf90, pgcc, pgCC

PGI Compiler Flags

- -default64 : Fortran driver option for –i8 and –r8
- -i8, -r8 : Treats INTEGER and REAL variables in Fortran as 8 bytes (use ftn –default64 option to link the right libraries
- -byteswapio : Reads big endian files in fortran
- -Mnomain : Uses ftn driver to link programs with the main program (written in C or C++) and one or more subroutines (written in fortran)

PGI Compiler Flags

- It is possible to disable optimizations included with –fast, for example –fast –Mnolre enables –fast and then disables loop redundant optimizations
- -Mconcur, -mprof=mpi, -Mmpi and –Mscalapack are no more supported
- Fortran interfaces can be called from C program by inserting an underscore to the respective name
- Pass argument by reference rather than by value
- For example to call dgetrf()
- Dgetrf_(&uplo, &M, &n, ……);
- To debug an optimized code, the –opt flag will insert debugging information without disabling optimizations

PGI Compiler Flags

- Some compiler options mat affect both performance and accuracy
- Lower accuracy is often higher performance, but it also able to enforce accuracy
  - -Kieee : all floating point (FP) math strictly conforms to IEEE , off by default
  - -Ktrap : Turns processor trapping of FP exceptions
  - -Mdaz : Treat all denormalized numbers as zeros
  - Mflushz : Set SSE to flush-to-zero (on with –fast)
  - -Mfprelaxed : allow to use relaxed (reduced) precision to speed up some floating point optimizations
    - Some compilers turn this on by default, PGI chooses to favor accuracy to speed, by default

Starting Point for GNU Compilers

- -O3 –ffast-math –funroll-loops
- Compiler feedback : -ftree-vectorizer-verbose=2
- Manpages : gfortran, gcc, g++

Numerical Libraries Overview

- Many commonly-used packages are available on Blue Waters
- Typically can link with most or all combinations of compiler, language, and parallel programming model
- Use the "`module`" command to select a particular version
- Will try to accommodate special installation requests (*can't install "Everything under the Sun" due to scalibility and other considerations*)

Cray Scientific Library (libsci)

- Contains optimized versions of several popular scientific software routines
- Available by default; can change versions with "`module avail`" and "`module load xt-libsci[/version]`"
  - BLAS, BLACS
  - LAPACK, ScaLAPACK
  - FFT, FFTW
- Unique to Cray (*affects portability*)
  - CRAFFT, CASE, IRT

PETSc (Argonne National Laboratory)

- Programmable, Extensible Toolkit for Scientific Computing
- Widely-used collection of many different types of linear and non-linear solvers
- Actively under development; very responsive team
- Can also interface with numerous optional external packages (*e.g.*, SLEPC, HYPRE, ParMETIS, …)
- Optimized version installed by Cray, along with many external packages
- Use "`module load petsc[/version]`"

Other Numerical Libraries

- ACML (AMD Core Math Library)
  - BLAS, LAPACK, FFT, Random Number Generators
- Trilinos (from Sandia National Laboratories)
  - Somewhat similar to PETSc, interfaces to a large collection of preconditioners, solvers, and other computational tools
- GSL (GNU Scientific Library)
  - Collection of numerous computational solvers and tools for C and C++ programs
- All available using "`module load`"

Optimization options

- Hybrid programming model (MPI+OpenMP, *et al*) is usually better
- Try 1, 2, 4, 16, 32 tasks per node

  For 1024 nodes:

  32 tasks+threads/node:

```
aprun –n 4096 –N 4 –d 8 ./myprog
```

  16 tasks+threads/node:

```
aprun –n 4096 –N 4 –d 4 \
  –cc 0,2,4,6:8,10,12,14:16,18,20,22:24,26,28,30 \
 ./myprog
```

- Try using –r 1 to reserve a core for the OS

```
aprun –n 4096 –N 4 –d 7 –r 1 \
  –cc 0-6:8-14:16-22:24-30 ./myprog
```

- Test different compilers, flags
- Use accelerators

OpenACC compiler support

Cray

Module load PrgEnv-cray craype-accel-nvidia35

- Fortran
  - -h acc, noomp # openmp is enabled by default, be careful mixing
  - -fpic -dynamic
  - -rm # include a .lst listing file to show the loop markup
  - -G2 # -g has been observed to break Cray OpenACC code
- C
  - -h pragma=acc -h nopragma=omp
  - -fpic -dynamic
  - -h msgs # show loop markup in stdout/stderr
  - -Gp # bonus points to the person who synchronizes Cray compiler flags between fortran and c...

Cray -rm # loop mark

```
arnoldg@h2ologin2:~/Mori/pic2.0-acc-f> ftn -h acc -rm -c push2.f

!$acc parallel num_gangs(1) vector_length(3072)
ftn-7271 crayftn: WARNING GPUSH2L, File = push2.f, Line = 145
  Unsupported OpenACC vector_length expression: Converting 3072 to 1024.

arnoldg@h2ologin2:~/Mori/pic2.0-acc-f> grep --after-context=5 '!$acc parallel num_gangs(1) vector_length(3072)' push2.lst
  145.  + G----------< !$acc parallel num_gangs(1) vector_length(3072)
ftn-7271 ftn: WARNING File = push2.f, Line = 145
  Unsupported OpenACC vector_length expression: Converting 3072 to 1024.

  146.    G           !!$acc kernels
  147.    G           !!data copy(part),copyin(fxy),create(nn,mm,dxp,dyp,np,mp,dx,dy,vx,vy)
arnoldg@h2ologin2:~/Mori/pic2.0-acc-f> grep 'line 145 ' push2.lst
  A region starting at line 145 and ending at line 240 was placed on the accelerator.

arnoldg@h2ologin2:~/Mori/pic2.0-acc-f>
```

OpenACC compiler support

PGI

Module load PrgEnv-pgi cudatoolkit

- Cudatoolkit is required, PGI is creating CUDA code as intermediate
  - -ta=nvidia,keepgpu,keepptx
- Fortran , C # nice
  - -acc -ta=nvidia
  - -mcmodel=medium
  - -Minfo=accel

GNU

- Don't touch that dial!

PGI -Minfo=accel

```
arnoldg@h2ologin2:~/Mori/pic2.0-acc-f> ftn -acc -ta=nvidia -Minfo=accel -c push2.f
gpush2l:
    145, Accelerator kernel generated
        145, CC 1.3 : 18 registers; 112 shared, 32 constant, 0 local memory bytes
             CC 2.0 : 26 registers; 0 shared, 132 constant, 0 local memory bytes
        148, !$acc loop vector(3072) ! blockidx%x threadidx%x
        169, Sum reduction generated for sum1
    145, Generating present_or_copy(part(:4,:nop))
         Generating present_or_copyin(fxy(:,:,:))
         Generating compute capability 1.3 binary
         Generating compute capability 2.0 binary
    148, Loop is parallelizable
```

CUDA is a parallel computing platform

- NVIDIA Kepler K20X accelerators
- XK nodes support CUDA compute capability 3.5
- CUDA C code should be compiled with nvcc
- Cray provides cc and CC wrappers for C/C++ that include support for MPI and OpenMP (use cc and CC instead of mpicc)
- Dynamic linking (static linking is not supported)

Tips for NVIDIA Kepler K20x GPUs

- CRAY_CUDA_PROXY=[0|1] default=1 (On) - multiple MPI tasks accessing same GPU on the node; turn to off (0) - single MPI task accessing GPU
- LD_LIBRARY_PATH= $CRAY_LD_LIBRARY_PATH: $LD_LIBRARY_PATH
- MPICH_RDMA_ENABLED_CUDA=[0|1]

  Allows the MPI application to pass GPU pointers directly to point to-point and collective communication functions. If the send or receive buffer for a point-to-point or collective communication is on the GPU, the network transfer and the transfer between the host CPU and the GPU are pipelined to improve performance.

Building CUDA Applications on Cray

- Setup CUDA programming environment

  module load cudatoolkit

  module show cudatoolkit

- Build CUDA code using PGI compiler

  module load PrgEnv-pgi

NVIDIA example: simpleMPI.cpp simpleMPI.cu simpleMPI.h

nvcc -c -gencode arch=compute_35,code=compute_35 –o simpleMPIcuda.o simpleMPI.cu

CC -o simpleMPI.x simpleMPI.cpp simpleMPIcuda.o

Rule: keep all MPI staff in C++ files and CUDA kernels in CU-files

Building GPU-to-GPU Application

OSU micro-benchmarks: **osu_latency_39.c**

Download link http://mvapich.cse.ohio-state.edu/benchmarks/

```
module swap PrgEnv-cray PrgEnv-gnu
module swap cray-mpich2 cray-mpich2/5.6.4
module load cudatoolkit
export LD_LIBRARY_PATH=$CRAY_LD_LIBRARY_PATH:
                                        $LD_LIBRARY_PATH
export MPICH_RDMA_ENABLED_CUDA=1
export CRAY_CUDA_PROXY=1
cc -D_ENABLE_CUDA_ -o osu_latency39.x osu_latency_39.c –lcudart
aprun -n 2 -N 1 ./osu_latency39.x D D > job39.out
```

PGI CUDA Fortran

Extension of F90 standard by CUDA language constructs

CUDA Fortran file has extension .CUF (compare to .F90)

Building CUDA Fortran application on Cray

wget http://www.pgroup.com/lit/samples/matmul.CUF

module swap PrgEnv-cray PrgEnv-pgi

module add cudatoolkit

pgfortran -ta=nvidia,kepler matmul.CUF -L/opt/cray/nvidia/default/lib64

#PBS -l nodes=1:ppn=16:xk

aprun -n 1 -N 1 ./matmul.x > job.out

OpenCL

- Limited support from Cray (not documented)
- Included with CUDA

```
module load PrgEnv-gnu cudatoolkit
cc -c -I$CUDATOOLKIT_HOME/include hello.c
cc hello.o -L/opt/cray/nvidia/default/lib64 -lOpenCL -o hello
```

The End