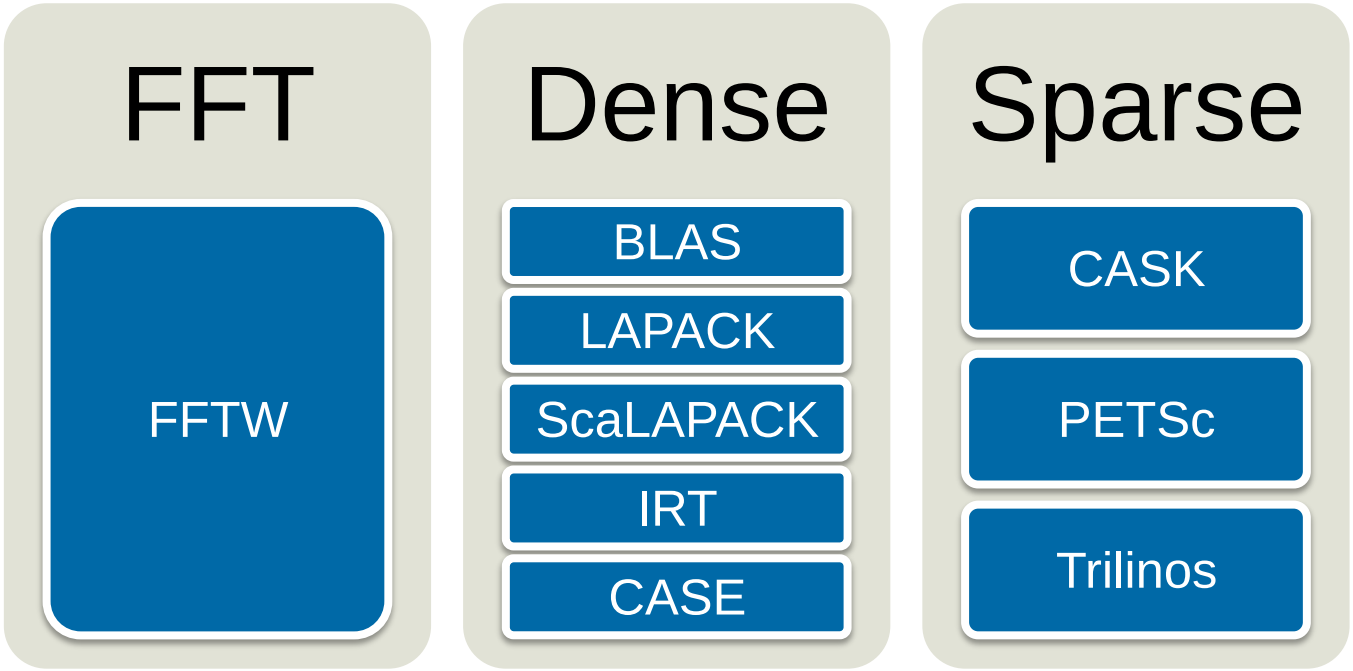
Cray Scientific Libraries

Overview

Please email detailed questions or speak to me later!

Cray Scientific Libraries

- **Large variety of standard libraries available via modules**
  - Optimized for Cray Hardware and also for Haswell processor.



IRT – Iterative Refinement Toolkit
CASK – Cray Adaptive Sparse Kernels
CASE – Cray Adaptive Simplified Eigensolver

What makes Cray libraries special

1. **Node performance**
   - Highly tuned routines at the low-level (ex. BLAS)
2. **Network performance**
   - Optimized for network performance
   - Overlap between communication and computation
   - Use the best available low-level mechanism
   - Use adaptive parallel algorithms
3. **Highly adaptive software**
   - Use auto-tuning and adaptation to give the user the known best (or very good) codes at runtime
4. **Productivity features**
   - Simple interfaces into complex software

Library Usage Overview.

- **LibSci**
  - Includes BLAS, CBLAS, BLACS, LAPACK, ScaLAPACK
  - Module is loaded by default (`man libsci`)
  - Threading used within LibSci (OMP_NUM_THREADS). If you call within a parallel region, single thread used. More info later on.
- **FFTW**
  - `module load fftw` and `man fftw`
- **PETSc**
  - `module load cray-petsc{-complex}` and `man intro_petsc`
- **Trilinos**
  - `module load cray-trilinos` and `man intro_trilinos`
- **Third Party Scientific Libraries**
  - `module load cray-tpsl` (use online documentation)
- **Iterative Refinement Toolkit (IRT) through LibSci.**
  - man intro_irt
- **Cray Adaptive Sparse Kernels (CASK) are used in cray-petsc and cray-trilinos (transparent to the developer).**

Third party Scientific Libraries (cray-tpsl)

- TPSL (Third Party Scientific Libraries) contains a collection of outside mathematical libraries that can be used with PETSc and Trilinos.
- This module will increase the flexibility of PETSc and Trilinos by providing users with multiple options for solving problems in dense and sparse linear algebra.
- The cray-tpsl module is automatically loaded when PETSc or Trilinos is loaded. The libraries included are MUMPs, SuperLU, SuperLU_dist, ParMetis, Hypre, Sundials, and Scotch.

Check you got the right library!

- Add options to the linker to make sure you have the correct library loaded.
- `-Wl` adds a command to the linker from the driver
- You can ask for the linker to tell you where an object was resolved from using the –y option.
  - e.g. `–Wl,-ydgemm_` (notice the '_' at the end of the name)

```
.//main.o: reference to dgemm_
/opt/xt-libsci/11.0.05.2/cray/73/mc12/lib/libsci_cray_mp.a(dgemm.o):
definition  of dgemm_
```

Note: do not explicitly link "-lsci"

Threading for BLAS and LAPACK

- **LibSci is compatible with OpenMP**
  - Control the number of threads to be used in your program using **OMP_NUM_THREADS**
  - e.g., in job script **`export OMP_NUM_THREADS=16`**
  - Then run with **`aprun`** with –d16
- **What behavior you get from the library depends on your code**
  1. No threading in code
     - The BLAS call will use OMP_NUM_THREADS threads
  2. Threaded code, outside parallel regions
     - The BLAS call will use OMP_NUM_THREADS threads
  3. Threaded code, inside parallel regions
     - The BLAS call will use a single thread
  - Threaded LAPACK works exactly the same as threaded BLAS
  - Anywhere LAPACK uses BLAS, those BLAS can be threaded.
  - Some LAPACK routines are threaded at the higher level

Intel MKL

- **The Intel Math Kernel Libraries (MKL) is an alternative to LibSci**
  - Features tuned performance for Intel CPUs as well
- **Using MKL together with the Intel compilers (PrgEnv-intel) is usually straightforward. Simply add `-mkl` to your compile and linker options**
- **Linking can be quite complicated otherwise, but the Intel MKL Link Line Advisor can tell you what to add to your link line (once you have recovered from your faint on seeing what it tells you)**
  - http://software.intel.com/sites/products/mkl/
- **If using the CCE compiler, first do "module unload cray-libsci"**
  - Also compile with –hnopattern to avoid the compiler pattern-matching parts of the code with libsci blas/lapack calls
  - "module load intel" to set $MKLROOT

FFTW

- **Cray's main FFT library is FFTW from MIT**
  - Some additional optimizations for Cray hardware
- **Usage is simple**
  - Load the module
  - In the code, call an FFTW plan
- **Cray's FFTW provides wisdom files for these systems**
  - You can use the wisdom files to skip the plan stage
  - This can be a significant performance boost
- **FFTW 3.3.4.1 includes Cray optimizations for SB, IVB, HSW processors**
- **There are some Cray specific optimizations you can enable by setting FFTW_CRAY_FASTPLAN=1**
  - Currently this help only on Haswell and power of 2 sizes

Cray Adaptive Sparse Kernel (CASK)

- **Sparse matrix operations in PETSc and Trilinos on Cray systems are optimized via CASK**
- **Offline**
  - ATF program builds many thousands of sparse kernels
  - Testing program defines matrix categories based on density, dimension etc
  - Each kernel variant is tested against each matrix class
  - Performance table is built and adaptive library constructed
- **Runtime**
  - Scan matrix at very low cost
  - Map user's calling sequence to nearest table match
  - Assign best kernel to the calling sequence
  - Optimized kernel used in iterative solver execution

Performance Focus and Autotuning

- **Some components of the library are performance critical**
  - For example BLAS and specifically GEMM
- **It is a significant challenge to get best performance across a range of architectures and problem sizes and thread counts**

- **Cray has an autotuning framework to address this:**
  - It uses a general GEMM framework
  - Offline tuning runs are done for a wide range of problem sizes
  - CPU and GPU targets
  - Knowledge gained from offline runs incorporated into the runtime library.

Tuning requests – you can help!

- **CrayBLAS is an auto-tuned library**
  - Generally, excellent performance is possible for all shapes and sizes
- **However, the adaptive CrayBLAS can be improved by tuning for exact sizes and shapes**
- **Optimizations for your particular calling parameters can be added to incremental enhancements of the CrayBLAS performance model**
- **Send your specific tuning requirements to crayblas@cray.com**
  - Old advice: Send the routine name and the list of calling sequences
  - Newer advice: Also run your code with:
    - CRAYBLAS_PROFILING=1 ./your_code

ScaLAPACK and IRT

- **ScaLAPACK in LibSci is optimized for Aries interconnect**
  - New collective communication procedures are added
  - Default topologies are changed to use the new optimizations
  - Much better strong scaling
- **It also benefits from the optimizations in CrayBLAS**
- **Iterative Refinement Toolkit (IRT) can provide further improvements**
  - Uses mixed precision
  - Linear solvers that use 32-bit factorization but mixed precision iterative refinement to give solution accurate to double precision
  - For some targets (CPU vector instructions and GPUs) single-precision can be much faster
  - Used for serial and parallel LU, Cholesky and QR
  - Either set IRT_USE_SOLVERS to 1 (good for first-time users, don't need source code changes) or use the advanced API (for experts!).

libsci for accelerators

Introduction

See the directory at $LIBSCI_ACC_EXAMPLES_DIR for numerous libsci_acc examples

LibSci for Accelerators: libsci_acc

- **Provide basic libraries for accelerators, tuned for Cray**
- **Must be independent to OpenACC, but fully compatible**
- **Multiple use case support**
  - Get the base use of accelerators with no code change
  - Get extreme performance of GPU with or without code change
  - Extra tools for support of complex code
- **Incorporate the existing GPU libraries into libsci**
  - CUBLAS
  - Magma
  - Cray Implementation BLAS/LAPACK
- **Provide additional performance and usability**
  - OpenACC support
  - CUDA support
- **Maintain the Standard APIs where possible!**

Why libsci_acc ?

- **Code modification is required to use existing GPU libraries!**
- **Several scientific library packages are already there**
  - CUBLAS, CUFFT, CUSPARSE (NVIDIA), MAGMA (U Tennessee), CULA (EM Photonics)
- **No Compatibility to Legacy APIs**
  - cublasDgemm(….)
  - magma_dgetrf(…)
  - culaDgetrf(…)
  - Why not dgemm(), dgetrf()?
- **Not focused on Fortran API (C/C++)**
  - Require CUDA data types, primitives and functions in order to call them
- **Performance**

Autotuning

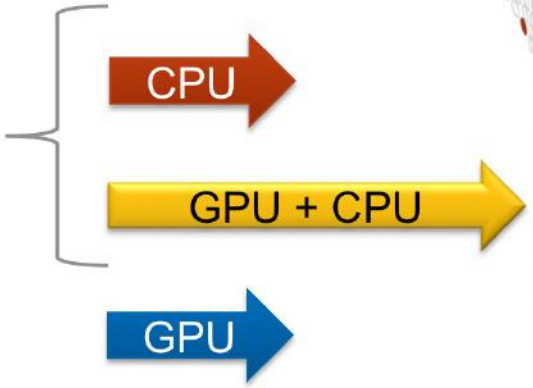
- **Cray Autotuning framework has been built to tune all BLAS for accelerators**
  - GPU kernel codes are built using code generator
  - Enormous offline autotuning is used to build a map of performance to input
  - An adaptive library is built from the results of the autotuning
  - At run-time, your code is mapped to training set of input
  - Best kernel for your problem is used

Three interfaces for three use cases

- **Simple interface**

```
dgetrf(M, N, A, lda, ipiv, &info)
```

```
dgetrf(M, N, d_A, lda, ipiv, &info)
```



- **Device interface**

```
dgetrf_acc(M, N, d_A, lda, ipiv, &info)
```

- **CPU interface**

```
dgetrf_cpu(M, N, A, lda, ipiv, &info)
```

Simple interface

- **You can pass either host pointers or device pointers to simple interface**
- **Host memory pointer**
  - Performs hybrid operation on GPU
  - If problem is too small, performs host operation
- **Device memory pointer**
  - Performs operation on GPU
- **BLAS 1 and 2 perform computation local to the data location**
  - CPU-GPU data transfer is too expensive to exploit hybrid execution

Device interface

- **Device interface gives higher degrees of control**
- **Requires that you have already copied your data to the device memory**
- **API**
  - Every routine in libsci has a version with _acc suffix
  - E.g. dgetrf_acc
  - This resembles standard API except for the suffix and the device pointers

CPU interface

- **Sometimes apps may want to force ops on the CPU**
  - Need to preserve GPU memory
  - Want to perform something in parallel
  - Don't want to incur transfer cost for a small op
- **Can force any operation to occur on CPU with _cpu version**
- **Every routine has a _cpu entry-point**
- **API is exactly standard otherwise**

Usage - basics

- **Supports Cray and GNU compilers.**
- **Fortran and C interfaces (column-major assumed)**
  - Load the module craype-accel-nvidia35.
  - Compile as normal (dynamic libraries used)
- **To enable threading in the CPU library, set OMP_NUM_THREADS**
  - E.g. export OMP_NUM_THREADS=16
- **Assign 1 single MPI process per node**
  - Multiple processes cannot share the single GPU
- **Execute your code as normal**

libsci_acc DGEMM example

- Starting with a code that relies on dgemm.
- The library will check the parameters at runtime.
- If the size of the matrix multiply is large enough, the library will run it on the GPU, handling all data movement behind the scenes.
- NOTE: Input and Output data are in CPU memory.

```
call dgemm('n','n',m,n,k,alpha,&

a,lda,b,ldb,beta,c,ldc)
```

libsci_acc interaction with OpenACC

- If the rest of the code uses OpenACC, it's possible to use the library with directives.
- All data management performed by OpenACC.
- Calls the device version of dgemm.
- All data is in CPU memory before and after data region.

```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$acc end parallel

!$acc host_data use_device(a,b,c)

call dgemm_acc('n','n',m,n,k,&
               alpha,a,lda,&
               b,ldb,beta,c,ldc)

!$acc end host_data
!$acc end data
```

libsci_acc interaction with OpenACC

- libsci_acc is a bit smarter that this.
- Since ‘a,’ ‘b’, and ‘c’ are device arrays, the library knows it should run on the device.
- So just dgemm is sufficient.

```
!$acc data copy(a,b,c)

!$acc parallel
!Do Something
!$acc end parallel

!$acc host_data use_device(a,b,c)

call dgemm     ('n','n',m,n,k,&
                alpha,a,lda,&
                b,ldb,beta,c,ldc)

!$acc end host_data
!$acc end data
```

Advanced controls

- The communication avoidance (CA) version of DGETRF/ ZGETRF can be enabled by setting the environment variable LIBSCI_ACC_DLU = CALU / LIBSCI_ACC_ZLU = CALU

- Change Split Ratio of Hybrid GEMM routines
  - LIBSCI_SGEMM_SPLIT=0.9
  - LIBSCI_DGEMM_SPLIT=0.8
  - LIBSCI_CGEMM_SPLIT=0.9
  - LIBSCI_ZGEMM_SPLIT=0.8

- Force simple API to always call CPU routine
  - CRAY_LIBSCI_ACC_MODE=2

Summary

- **Access to libsci_acc routines simple**
  - No need to explicitly link - Programming Environment drivers (cc, ftn, CC) do this for you
  - Just target the GPU by loading module
- **Automatically take advantage of threading on CPU**
  - Just sent OMP_NUM_THREADS and run
- **Simple interface to enable hybrid, CPU or GPU execution of a routine depending on where memory pointers reside and problem size**
- **Interface for advanced control also available**