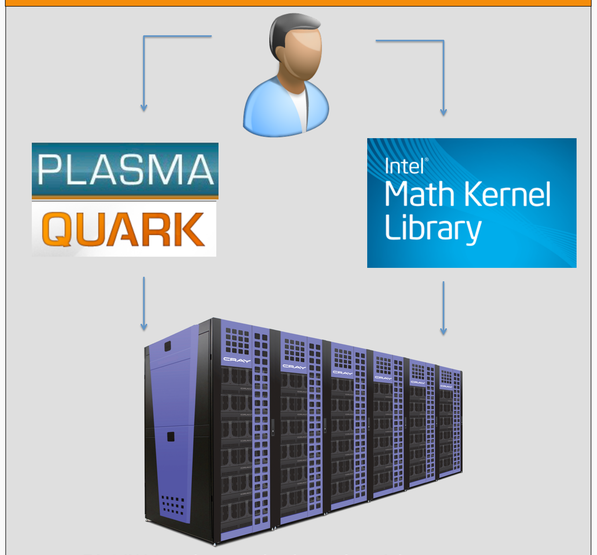
Runtime Systems and Out-of-Core Cholesky Factorization on the Intel Xeon Phi System

Authors: Allan Richmond Razon Morales (The George Washington University), Tian Chong (The Chinese University of Hong Kong) **Mentors:** Dr. Kwai Wong (UTK), Dr. Eduardo D'Azevedo (ORNL)

OBJECTIVE

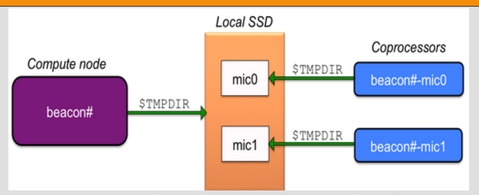
We will explore how different runtime systems can be implemented on the Intel Xeon Phi System on Beacon. This coprocessor does have its own Intel MKL library that implements BLAS and LAPACK functionality. For this research, we will first explore how to utilize PLASMA for handling dense linear algebra computations and QUARK for task management and added parallelism to figure out the dependencies between the tasks and the scheduler. Once accomplished, these algorithms will be rigorously tested on the Beacon's MIC card for performance analysis and comparison with the standard Intel MKL implementation. Another goal is to implement a hybrid Out-of-Core algorithm for Cholesky factorization that can be used in conjunction with the PLASMA/QUARK implementation to see if its performance is efficient and scalable.

VISUAL OF THE OVERVIEW



- **PLASMA** - dense algebra algorithms
- **QUARK** - multithreading and task management
- **Intel MKL Library** - optimized math library for comparison with PLASMA

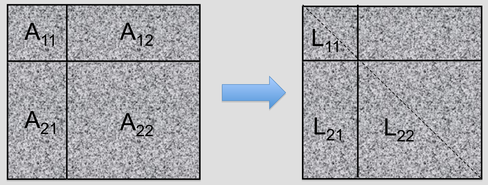
MODES OF EXECUTION



There are two modes of execution within Beacon: Native and Offload. The former relies on programming directly into the co-processor (MIC card). The goal for further optimization is using Offload Mod e, which will run on the host processor and "offload" the dense calculations to the co-processor.

CHOLESKY FACTORIZATION

Cholesky Factorization: $A=LL^T$.
A is SPD (symmetric, positive definite).

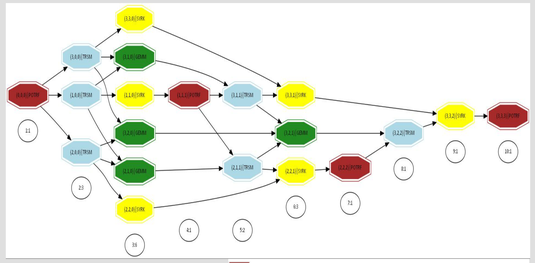


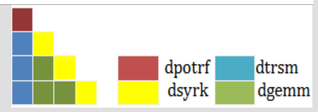
Recursive Cholesky steps on matrix blocks

- Step 1: L_{11} <-- cholesky(A_{11}) ,potrf()
- Step 2: L_{21} <-- A_{21} / L_{11}^T, trsm()
- Step 3: A_{22} <-- A_{22} - L_{21} * L_{21}^T, syrk() and gemm()
- Step 4: L_{22} <-- cholesky(A_{22}), potrf()

TASK DIRECTED ACYCLIC GRAPH (DAG)

- Tasks in Cholesky factorization depend on previous tasks if they use the same tiles of data. If we use a node to represent an operation on a tile and use an edge to represent a data dependency, then a DAG is formed.
- Once the DAG is produced and fed into the QUARK runtime system, tasks can be scheduled asynchronously and independently as long as the dependencies are not violated. Here a 4*4 Cholesky example is shown.(i,j,k): operation in the k-th iteration on the (i,j)-th tile.





Pseudocode for DAG:

```
for k=0...n-1
   for j=k...n-1
      for i=j...n-1 {
         if (i=j=k)  potrf (A(i,j,k-1)^r, A(i,j,k )^w)
         if (i>j=k)  trsm (A(i,j,k-1)^r, A(k,k,k)^r, A(i,j,k)^w)
         if (i=j>k)  syrk (A(i,j,k-1)^r, A(i,k,k)^r, A(i,j,k)^w)
         if (i>j>k)  gemm (A(i,j,k-1)^r,
                           A(i,k,k)^r,A(j,k,k)^r,A(i,j,k)^w)}
```

OUT-OF-CORE ALGORITHM (OOC)

- OOC stores most data on CPU memory and brings small pieces of data into coprocessors for computation,and then write them back. It takes advantage of the computational efficiency of hardware accelerators without limiting the size of the matrix problem.
- **CPU vs coprocessors(GPU,MIC,etc.)**:GPU is much faster and more energy efficient than CPU but has limited amount of device memory.

OOC STRUCTURE

- The **out-of-core** part loads parts of the matrix. For example, matrix panels,to device memory,and applies the "left-looking" update from the parts already factorized and written back.
- The **In-core** part factorizes the parts residing on device memory in which "right-looking" update is involved.

PROPOSED METHODOLOGY

- Matrix Multiplication and other BLAS routines (QUARK, Intel MKL)
- Hello World Multithreading (QUARK, Intel MKL)
- Performance Testing in seconds and GFLOPS - Giga Floating Operations Per Second (PLASMA, Intel MKL)

CURRENT PROGRESS

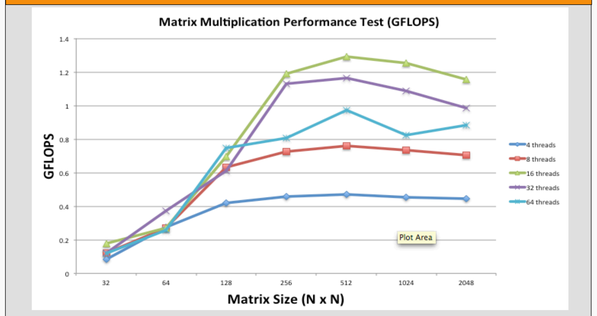
- I have modified example code from Dr. Asim YarKhan, one of the main developers of QUARK, for a matrix multiplication driver that will measure the performance of serial implementation and QUARK multi-threading.

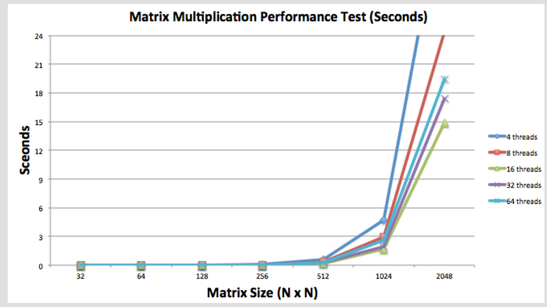
$$\begin{bmatrix}1&2&3&4\\5&6&7&8\\8&7&6&5\\4&3&2&1\end{bmatrix}\times\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}1&2&3&4\\5&6&7&8\\8&7&6&5\\4&3&2&1\end{bmatrix}$$

- The data will be printed in a user-friendly manner and measure the data in seconds and GFLOPS.
- To generate GFLOPS/sec, under the assumption for matrix multiplication C = A * B that A,B,C are symmetric matrices (n by n), then the general formula would be:

$$\frac{2n^3}{10^9 * \text{sec}}$$

PRELIMINARY RESULTS





EXPECTED GOALS

- A driver optimized to implement QUARK and MKL threading to perform matrix multiplication and eventually other BLAS/ LaPACK routines. The data measurements should be organized and readable for both new and experienced users.
- Given the current DAG, optimize the Cholesky DAG and eventually replicate the DAG on QUARK.

REFERENCES

- Betro, Vincent. *Beacon Quickstart Guide at AACE/NICS*
- Betro, Vincent. *Beacon Training: Using the Intel Many Integrate Core (MIC) Architecture: Native Mode and Intel MPI*. March 2013
- D'Azevedo, Eduardo, Shiquan Su, and Kwai Wong. *A Performance Study of Solving a Large Dense Matrix for Radiation Heat Transfer*.
- YarKhan, Asim. *Dynamic Task Execution on Shared and Distributed Memory Architectures*. Dec. 2012.
- YarKhan, Asim, Jakub Kurzak, and Jack Dongarra. *QUARK Users' Guide*. April 2011

TEAM INFO

- **Authors:** Allan Richmond Razon Morales and Tian Chong
- **Mentors:** Dr. Kwai Wong and Dr. Eduardo D'Azevedo
- **Collaborators:** Dr. Shiquan Su, Dr. Asim YarKhan, and Ben Chan