 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.

---

**PROPOSED METHODOLOGY**

**Performance testing in seconds, GFLOPS, GFLOPS/sec** (“Giga Floating Operations Per Second”)

1. **Nested-For Loop Matrix Multiplication (MM) - QUARK**
2. **DGEMM - PLASMA, Intel MKL**
3. **Cholesky - Intel MKL**

Both Native and Offload Execution were taken into consideration.

---

**MATRIX MULTIPLICATION RESULTS**

The objective is to test the performance of the Intel Xeon Phi System on Beacon. The general trend for the HOST shows optimal performance at 16 threads. The table below shows the performance of DGEMM on the Intel Xeon Phi System on Beacon.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Threads</th>
<th>GFLOPS/sec</th>
<th>GFLOPS</th>
<th>MAX GFLOPS/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>4</td>
<td>5.52</td>
<td>2.22</td>
<td>3.70</td>
</tr>
<tr>
<td>Native</td>
<td>8</td>
<td>7.88</td>
<td>4.57</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>16</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>32</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>64</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>128</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>256</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>512</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
<tr>
<td>Native</td>
<td>1024</td>
<td>5.52</td>
<td>3.70</td>
<td>5.87</td>
</tr>
</tbody>
</table>

---

**EXPECTED GOALS**

- **Optimize** QUARK implementations (matrix multiplication, DGEMM) with additional OpenMP directives to produce better performance.
- **Incorporate** the OOC Cholesky Factorization into QUARK and implement onto Beacon.
- **Enhance** the OOC Cholesky Factorization.
- Complete the code combining OOC algorithm and general Cholesky factorization.
- Extend to multiple MP processors core.
- Extend to L2 factorization with pivoting and QR factorization.

---

**REFERENCES**

- Brian, Vincent. *Beeon Quizlib Driven at ACC/NSF.*

---

**MODES OF EXECUTION**

Intel Xeon Processor ES-2670
- 2 x 6 cores (16 in total per node)
- 64 cores
- 2.60 GHz Clock Speed
- 10.53 GHz Clock Speed
- 8 GB RAM

Intel Xeon Phi Coprocessor 5110P
- 4 x Intel Xeon Phi Coprocessor 5110P
- 2 x 8 cores (16 in total per node)
- 60 cores
- 2.60 GHz Clock Speed
- 1.03 GHz Clock Speed
- 256 GB RAM

---

**CODE GENERATING DAG & CODE USING QUARK**

- **&OCC**
- **&QUARK**
- **&DGEMM**
- **&SPOTRF**

---

**OUT-OF-CORE ALGORITHM (OOC)**

- **OCC** stores most data on CPU memory and brings small pieces of data into coprocessors for computation, and then write back.
- **CPU vs coprocessors** (GPU, MIC, etc.): CPU is much faster and more energy efficient than CPU but has limited amount of device memory.

---

**CHOLESKY FACTORIZATION**

- **Cholesky Factorization:** $A = LL^T$
- **Cholesky steps on matrix blocks**
  - **Step 1:** $L_{11} \leftarrow \text{cholesky}(A_{11})$, potrf()
  - **Step 2:** $L_{21} \leftarrow L_{11} \cdot A_{12}$, trasn1()
  - **Step 3:** $A_{22} \leftarrow A_{22} - L_{21} \cdot L_{12}$, syrk() and gemm() and
  - **Step 4:** $L_{22} \leftarrow \text{cholesky}(A_{22})$, potrf()

---

**TASK DIRECTED ACYCLIC GRAPH (DAG)**

- Tasks in Cholesky factorization depend on previous tasks if they use the same 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. (Eq. by 4 case)

---

**Pseudocode for DAG:**

```c
for k = 0, \ldots, n - 1
  for i = k, \ldots, n - 1
    if (j > k && (j > k && (j > k && (j > k)))
      // ... compute...
      if ((j > k) && (j > k))
        // ... compute...
      if (j > k)
        // ... compute...
      if (j > k)
        // ... compute...
```

---

**DSOTRF (CHOLESKY FACTORIZATION MKL)**

- **Formula for GFLOPS:**
  - Single Precision Cholesky Factorization was tested on different models of execution.
  - MAX GFLOPS/sec was achieved at 745 within the MIC.
- Given the MIC environment variables, a stress test was implemented to see what were the ideal conditions for getting a similar performance output.
- Best overall performance was attained from using 240 threads and organizing in a compact manner.

---

**REFERENCES**

- Images are provided by Google Images, their respective websites, or generated using software.

---

**TEAM INTO**

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