OpenMP-based parallel implementation of matrix-matrix multiplication on the Intel Knights Landing

Roktaek Lim, Yeongha Lee, Raehyun Kim, Jaeyoung Choi

Soongsil University

IXPUG Workshop Asia 2018
Motivation

• Studying the architecture of the KNL
  – GEMM is used to illustrate how to attain high performance on a target architecture
  – 2D tile mesh architecture, Intel AVX-512 instructions

• Providing the implementation guidelines of GEMM for general users
  – Method for choosing the sizes of block matrices
  – Coding with AVX-512 instructions
Hardware and Software Descriptions

• The first system is equipped Intel Xeon Phi Processor 7210. The system is configured in the Flat/Quadrant mode. The Intel Parallel Studio XE 2017 update 4 is installed.

• The second system is equipped Intel Xeon Phi Processor 7250. The system is configured in the Flat/Hemisphere mode. The Intel Parallel Studio XE 2017 update 1 is installed.
GotoBLAS algorithm for GEMM

GEMM for KNL

ALGORITHM 1: Matrix-matrix multiplication algorithm.

for \( p = 0, \ldots, k - 1 \) in steps of \( k_b \) do
  Pack \( B_p \) into \( \tilde{B} \);
  for \( i = 0, \ldots, m - 1 \) in steps of \( m_b \) do
    Pack \( A_{i,p} \) into \( \tilde{A} \);
    for \( j_r = 0, \ldots, n - 1 \) in steps of \( n_r \) do
      \( \tilde{A} = A_{ir} \);
      \( \tilde{B} = B_{jr} \);
      \( \tilde{C} \) += \( \tilde{A} \times \tilde{B} \);
      Update \( C \) using \( \tilde{C} \);
    end
  end
  end
end

- Removing Loop 5 (no L3 cache)
- Only blocking for the L2 cache (amortizing the cost of updating \( \tilde{A} \))
- Using one thread per core (due to improved threading technology for the KNL)
GEMM for KNL

```c
// C += A*B
#pragma unroll (N)
for (i = 0; i < k; ++i) {
    _mm_prefetch((const void*) &B[L1_DIST_B], MM_HINT_T0); // L1
    _mm_prefetch((const void*) &A[L1_DIST_A], MM_HINT_T0); // L1
    _mm_prefetch((const void*) &A[L1_DIST_A+8], MM_HINT_T0); // L1
    _mm_prefetch((const void*) &A[L1_DIST_A+16], MM_HINT_T0); // L1
    _mm_prefetch((const void*) &A[L1_DIST_A+24], MM_HINT_T0); // L1
    _B = _mm512_load_pd(&B[0]);
            \text{한 줄 통합 결과}
    _C0 = _mm512_fmadd_pd(_mm512_set1_pd(A[0]), _B, _C0);
    \_C1 = _mm512_fmadd_pd(_mm512_set1_pd(A[1]), _B, \_C1);
    \_C2 = _mm512_fmadd_pd(_mm512_set1_pd(A[2]), _B, \_C2);
    \_C3 = _mm512_fmadd_pd(_mm512_set1_pd(A[3]), _B, \_C3);
    \_C4 = _mm512_fmadd_pd(_mm512_set1_pd(A[4]), _B, \_C4);
    \_C5 = _mm512_fmadd_pd(_mm512_set1_pd(A[5]), _B, \_C5);
    \_C6 = _mm512_fmadd_pd(_mm512_set1_pd(A[6]), _B, \_C6);
    \_C7 = _mm512_fmadd_pd(_mm512_set1_pd(A[7]), _B, \_C7);
    \_C8 = _mm512_fmadd_pd(_mm512_set1_pd(A[8]), _B, \_C8);
    \_C9 = _mm512_fmadd_pd(_mm512_set1_pd(A[9]), _B, \_C9);
    \_CA = _mm512_fmadd_pd(_mm512_set1_pd(A[10]), _B, \_CA);
    \_CB = _mm512_fmadd_pd(_mm512_set1_pd(A[11]), _B, \_CB);
    \_CC = _mm512_fmadd_pd(_mm512_set1_pd(A[12]), _B, \_CC);
    \_CD = _mm512_fmadd_pd(_mm512_set1_pd(A[13]), _B, \_CD);
    \_CE = _mm512_fmadd_pd(_mm512_set1_pd(A[14]), _B, \_CE);
    \_CF = _mm512_fmadd_pd(_mm512_set1_pd(A[15]), _B, \_CF);
    \_C10 = _mm512_fmadd_pd(_mm512_set1_pd(A[16]), _B, \_C10);
```

Manual loop unrolling

L1 prefetching
GEMM for KNL

![Performance of DGEMM for each \((m_r,n_r)\) when \(m=n=k=2400\)](image)

**Table 1** \(k_b, m_b\) and prefetch distances for DGEMM.

<table>
<thead>
<tr>
<th>((m_r,n_r))</th>
<th>(31,8)</th>
<th>(15,16)</th>
<th>(7,32)</th>
<th>(16,8)</th>
<th>(8,16)</th>
<th>(30,8)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of vector registers</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>17</td>
<td>18</td>
<td>31</td>
</tr>
<tr>
<td>(k_b)</td>
<td>480</td>
<td>400</td>
<td>400</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>(m_b)</td>
<td>124</td>
<td>135</td>
<td>126</td>
<td>112</td>
<td>112</td>
<td>120</td>
</tr>
<tr>
<td>L1 distance for (A)</td>
<td>10(m_r)</td>
<td>20(m_r)</td>
<td>20(m_r)</td>
<td>20(m_r)</td>
<td>20(m_r)</td>
<td>10(m_r)</td>
</tr>
<tr>
<td>L1 distance for (B)</td>
<td>28(n_r)</td>
<td>16(n_r)</td>
<td>12(n_r)</td>
<td>36(n_r)</td>
<td>24(n_r)</td>
<td>28(n_r)</td>
</tr>
<tr>
<td>Size of ((A + B + \hat{C}))</td>
<td>497 KB</td>
<td>474 KB</td>
<td>496 KB</td>
<td>451 KB</td>
<td>481 KB</td>
<td>481 KB</td>
</tr>
</tbody>
</table>

\(\leq 512\) KB (half of the size of the L2 cache)
GEMM for KNL

DGEMM performance comparison on the KNL

m=n=k

GFLOPS

0 2000 4000 6000 8000 10000

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

m_r n_r m_b k_b = (31,8,31,1620)

m_r n_r m_b k_b = (31,8,124,480)

- MKL(# of threads = 1)
Opportunities for parallelism

**ALGORITHM 1**: Matrix-matrix multiplication algorithm.

```plaintext
for \( p = 0, \ldots, k - 1 \) in steps of \( k_b \) do
  Pack \( B_p \) into \( \tilde{B} \);
  for \( i = 0, \ldots, m - 1 \) in steps of \( m_b \) do
    Pack \( A_{i,p} \) into \( A \);
    for \( jr = 0, \ldots, n - 1 \) in steps of \( n_r \) do
      for \( ir = 0, \ldots, m_b - 1 \) in steps of \( m_r \) do
        \( \hat{A} = \hat{A}_{ir} \);
        \( \hat{B} = \hat{B}_{jr} \);
        \( \hat{C} += \hat{A} \times \hat{B} \);
        Update \( C \) using \( \hat{C} \);
      end
    end
  end
end
```

*Tyler Smith et al. “Anatomy of High Performance Many-Threaded Matrix Multiplication” In *ACM Transactions on Mathematical Software (TOMS)*.
Opportunities for parallelism

```c
#pragma omp parallel num_threads(17) private(i,mc,_A)
{
    #pragma omp for schedule(dynamic)
    for(i = 0; i < mq; ++i)
    {
        mc = (i != mq-1 || md == 0) ? MB : md;
        packarc(mc,k,&A[k*KB+i*MB*la],la,_A);
        #pragma omp parallel num_threads(4) private(j,nc,pq,pd,p,pc,_C) shared(i,mc,_A)
        {
            // jr-loop
            #pragma omp for
            for(j = 0; j < nq; ++j)
            {
                nc = (j != nq-1 || nd == 0) ? NR : nd;
                pq = (mc+MR-1) / MR;
                pd = mc % MR;
                // ir-loop
                for(p = 0; p < pq; ++p)
            }
        }
    }
}
```
Cache block sizes for KNL

• From performance experiments on a single core,
  \[(m_bk_b + k_bn_r + m_rn_r)\times8 \text{ bytes} \leq \frac{1MB}{2}.\]
• For the parallel implementation,
  \[2\times(m_bk_b + k_bn_r + m_rn_r)\times8 \text{ bytes} \leq 512 KB.\]
• If both cores on the same tile require the same \(\hat{A}\), to compute the inner kernel,
  \[(m_bk_b + 2k_bn_r + 2m_rn_r)\times8 \text{ bytes} \leq 512 KB.\]
Thread affinity

• Parallel implementation of DGEMM on the KNL contains two nested loops, i- and jr-loop.
• HOTTEAM, runtime environment variable should be used to avoid the overhead of creating and destroying threads.

  KMP_HOT_TEAMS_MODE = 1
  KMP_HOT_TEAMS_MAX_LEVEL = 2
Thread affinity

We use only one threads per core.

<table>
<thead>
<tr>
<th>Allocate hardware threads</th>
<th>OpenMP* 4 Affinity</th>
<th>Intel OpenMP Runtime Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OMP_PLACES</td>
<td>KMP_PLACE_THREADS</td>
</tr>
</tbody>
</table>

| Pin OpenMP threads to hardware threads | OMP_PROC_BIND | KMP_AFFINITY |
OpenMP Affinity

OMP_PLACES = cores
OMP_PROC_BIND = spread, spread

\( k_b \) can be determined using the following inequality,

\[
(m_b k_b + 2k_b n_r + 2m_r n_r) \times 8 \text{ bytes} \leq 512 \text{ KB}
\]
Intel OpenMP runtime library

KMP_AFFINITY = scatter

\( i \)-loop

\( jr \)-loop

\( k_b \) can be determined using the following inequality,

\[
2 \times (m_b k_b + k_b n_r + m_r n_r) \times 8 \text{ bytes} \leq 512 \text{ KB}
\]
Xeon Phi 7210/7250

DGEMM performance comparison for the Intel Xeon Phi Processor 7210

DGEMM performance comparison for the Intel Xeon Phi Processor 7250
Summary

• Parallel implementation of DGEMM with OpenMP on the KNL
  – Method for choosing the sizes of block matrices
  – Coding with AVX-512 instructions

• Performance study
  – OpenMP Affinity
  – Intel OpenMP runtime library
Thank you for your attention!