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

숭실대학교
Soongsil University

Motivation

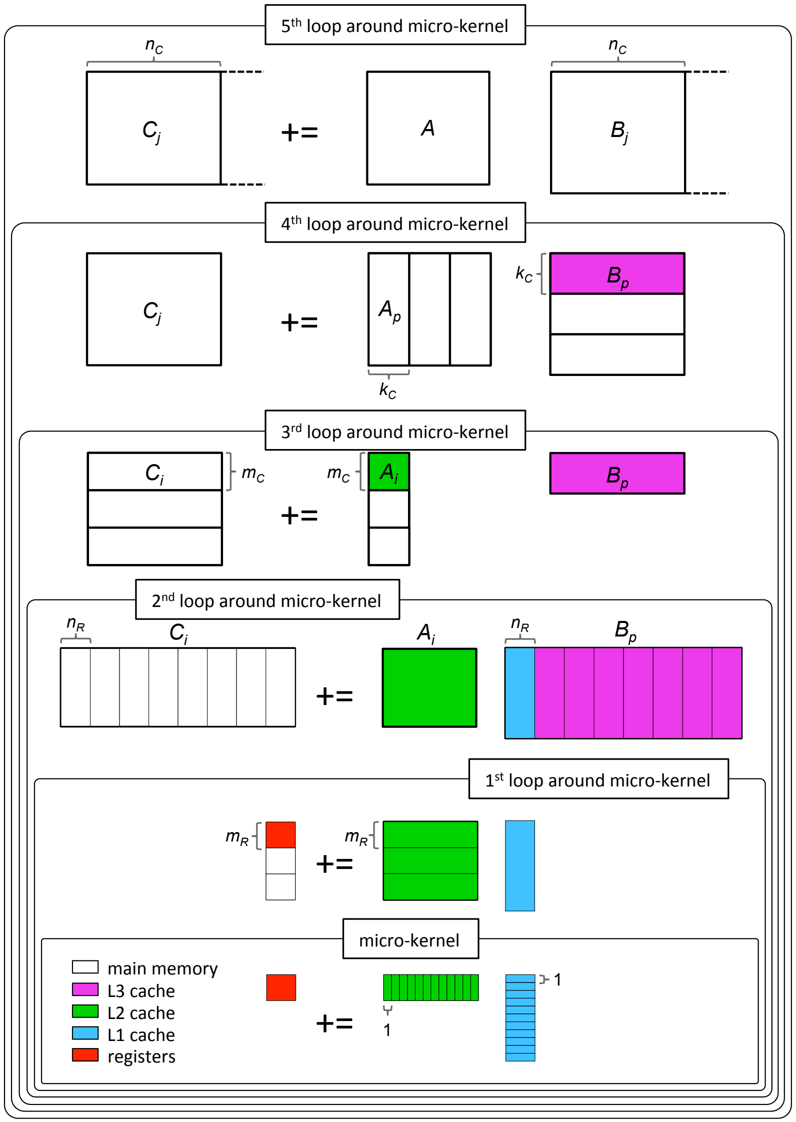
- Studying the architecture of the KNL
  - GEMM is used to illustrate how to attain high performance on a target architecture
  - 2D tile mesh architecture, Intel AVX-512 instructions
- Providing the implementation guidelines of GEMM for general users
  - Method for choosing the sizes of block matrices
  - 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.

숭실대학교
Soongsil University

GotoBLAS algorithm for GEMM



Loop 5 **for** $j_c = 0 : n-1$ **steps of** n_c

$\mathcal{J}_c = j_c : j_c + n_c - 1$

Loop 4 **for** $p_c = 0 : k-1$ **steps of** k_c

$\mathcal{P}_c = p_c : p_c + k_c - 1$

$B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow \widetilde{B}_p$

Loop 3 **for** $i_c = 0 : m-1$ **steps of** m_c

$\mathcal{I}_c = i_c : i_c + m_c - 1$

$A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow \widetilde{A}_i$

// macro-kernel

Loop 2 **for** $j_r = 0 : n_c - 1$ **steps of** n_r

$\mathcal{J}_r = j_r : j_r + n_r - 1$

Loop 1 **for** $i_r = 0 : m_c - 1$ **steps of** m_r

$\mathcal{I}_r = i_r : i_r + m_r - 1$

//micro-kernel

Loop 0 **for** $p_r = 0 : p_c - 1$ **steps of** 1

$C_c(\mathcal{I}_r, \mathcal{J}_r) \mathrel{+}= \widetilde{A}_i(\mathcal{I}_r, p_r)\, \widetilde{B}_p(p_r, \mathcal{J}_r)$

endfor

endfor

endfor

endfor

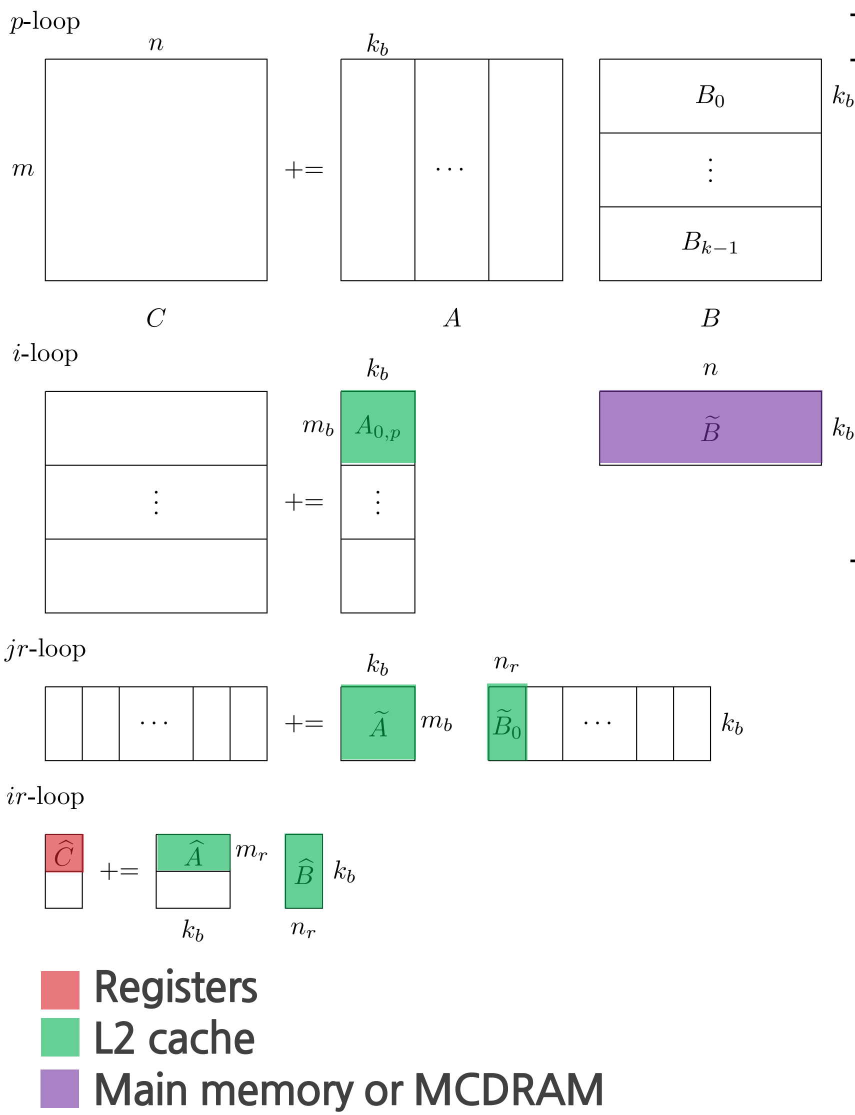
endfor

endfor

*Field G. Van Zee, and Tyler M. Smith. "Implementing high-performance complex matrix multiplication." In *ACM Transactions on Mathematical Software (TOMS)*.

숭실대학교 Soongsil University

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 $\widetilde{B}$;
  for $i = 0, \ldots, m-1$ *in steps of* m_b **do**
    Pack $A_{i,p}$ into $\widetilde{A}$;
    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**
        $\widehat{A} = \widetilde{A}_{ir}$;
        $\widehat{B} = \widetilde{B}_{jr}$;
        $\widehat{C}$ += $\widehat{A} \times \widehat{B}$;
        Update C using $\widehat{C}$;
      end
    end
  end
end

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

GEMM for KNL

```
// _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]);
  _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

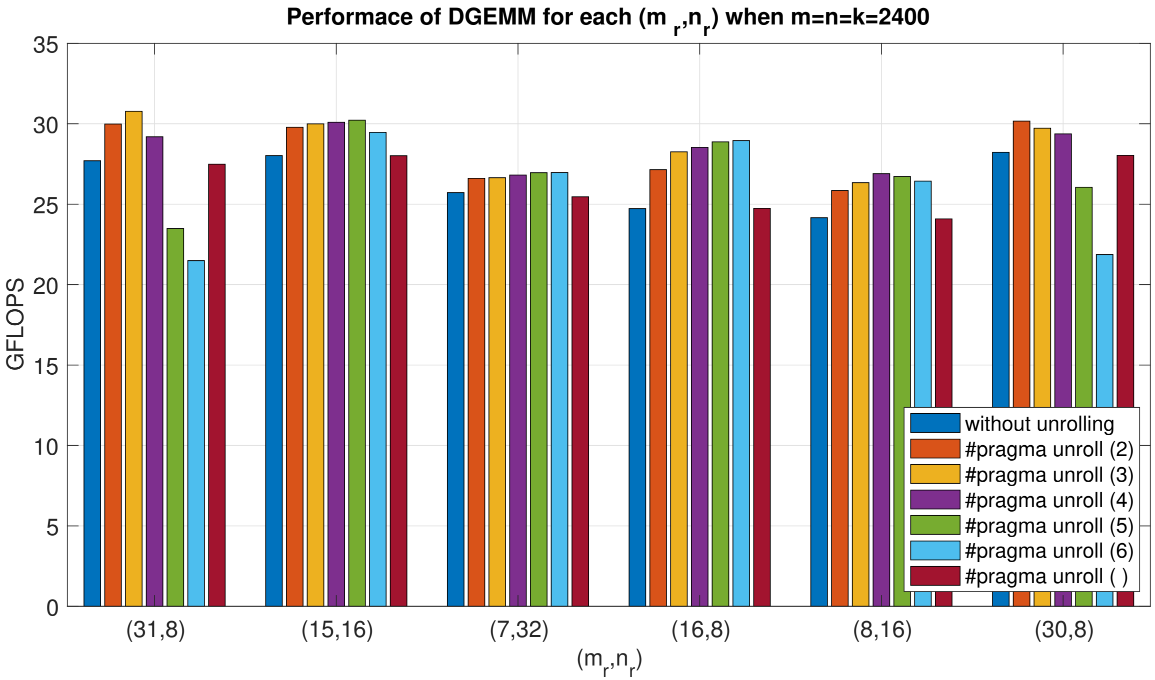
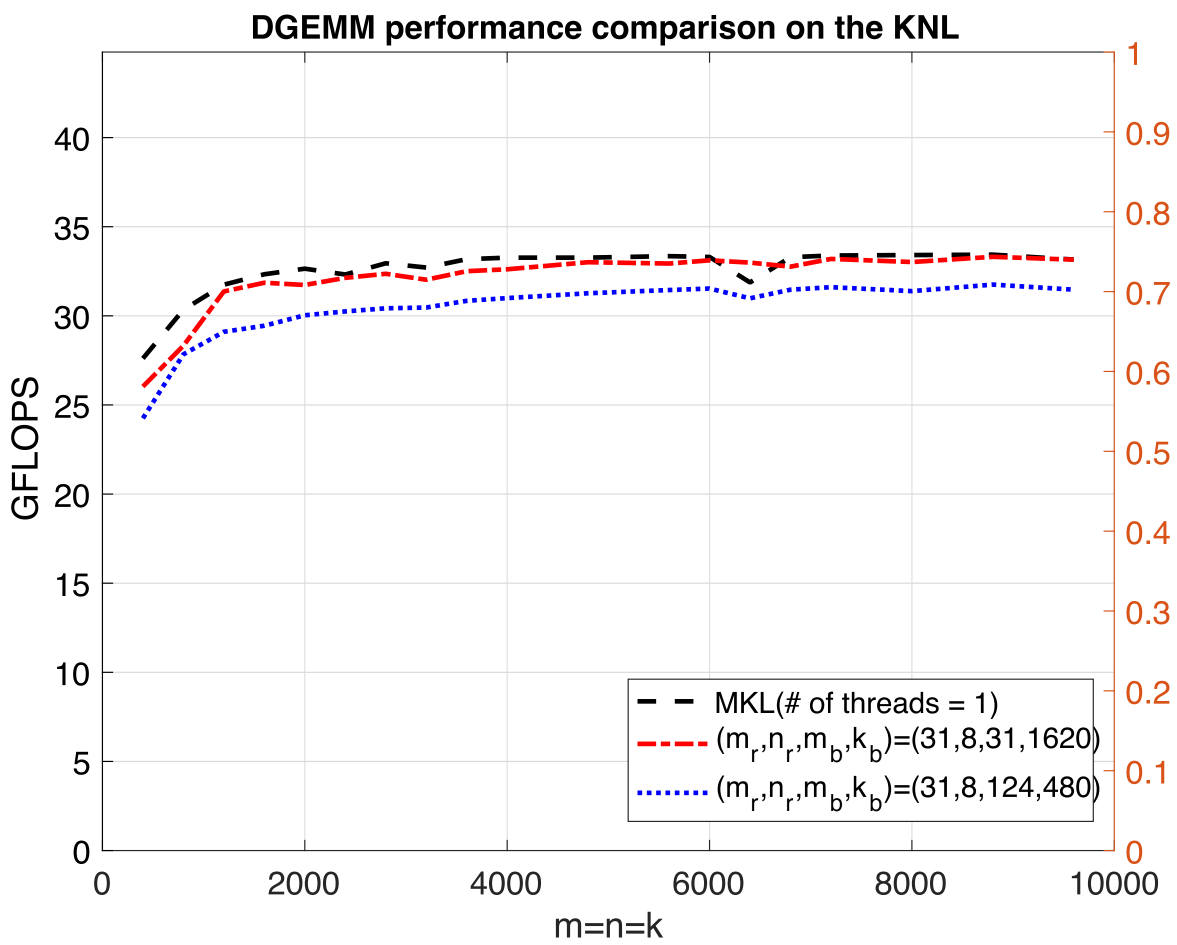


Table 1 k_b, m_b and prefetch distances for DGEMM.

(m_r, n_r)	(31,8)	(15,16)	(7,32)	(16,8)	(8,16)	(30,8)
# of vector registers	32	32	32	17	18	31
k_b	480	400	400	480	480	480
m_b	124	135	126	112	112	120
L1 distance for $\widehat{A}$	$10m_r$	$20m_r$	$20m_r$	$20m_r$	$20m_r$	$10m_r$
L1 distance for $\widehat{B}$	$28n_r$	$16n_r$	$12n_r$	$36n_r$	$24n_r$	$28n_r$
Size of $(\widetilde{A} + \widehat{B} + \widehat{C})$	497 KB	474 KB	496 KB	451 KB	481 KB	481 KB

≤ 512 KB (half of the size of the L2 cache)

GEMM for KNL


DGEMM performance comparison on the KNL
GFLOPS
0
5
10
15
20
25
30
35
40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
2000
4000
6000
8000
10000
m=n=k
MKL(# of threads = 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)

Opportunities for parallelism

ALGORITHM 1: Matrix-matrix multiplication algorithm.

for $p = 0, \ldots, k-1$ *in steps of* k_b **do**
  Pack B_p into $\widetilde{B}$;
  for $i = 0, \ldots, m-1$ *in steps of* m_b **do** ← m/m_b iterations
    Pack $A_{i,p}$ into $\widetilde{A}$;
    for $jr = 0, \ldots, n-1$ *in steps of* n_r **do** ← n/n_r iterations
      for $ir = 0, \ldots, m_b-1$ *in steps of* m_r **do**
        $\widehat{A} = \widetilde{A}_{ir}$;
        $\widehat{B} = \widetilde{B}_{jr}$;
        $\widehat{C} \mathrel{+}= \widehat{A} \times \widehat{B}$;
        Update C using $\widehat{C}$;
      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

```
//
omp_set_nested(1);
//
#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[ki*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)
```

i-loop

jr-loop

Cache block sizes for KNL

- From performance experiments on a single core,

$$(m_b k_b + k_b n_r + m_r n_r)\times 8\ bytes \le \frac{1MB}{2}.$$

- For the parallel implementation,

$$2\times(m_b k_b + k_b n_r + m_r n_r)\times 8\ bytes \le 512\ KB.$$

- If both cores on the same tile require the same $\hat{A}$, to compute the inner kernel,

$$(m_b k_b + 2k_b n_r + 2m_r n_r)\times 8\ bytes \le 512\ KB.$$

Thread affinity

- Parallel implementation of DGEMM on the KNL contains two nested loops, i- and jr-loop.
- HOT_TEAM, 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
```

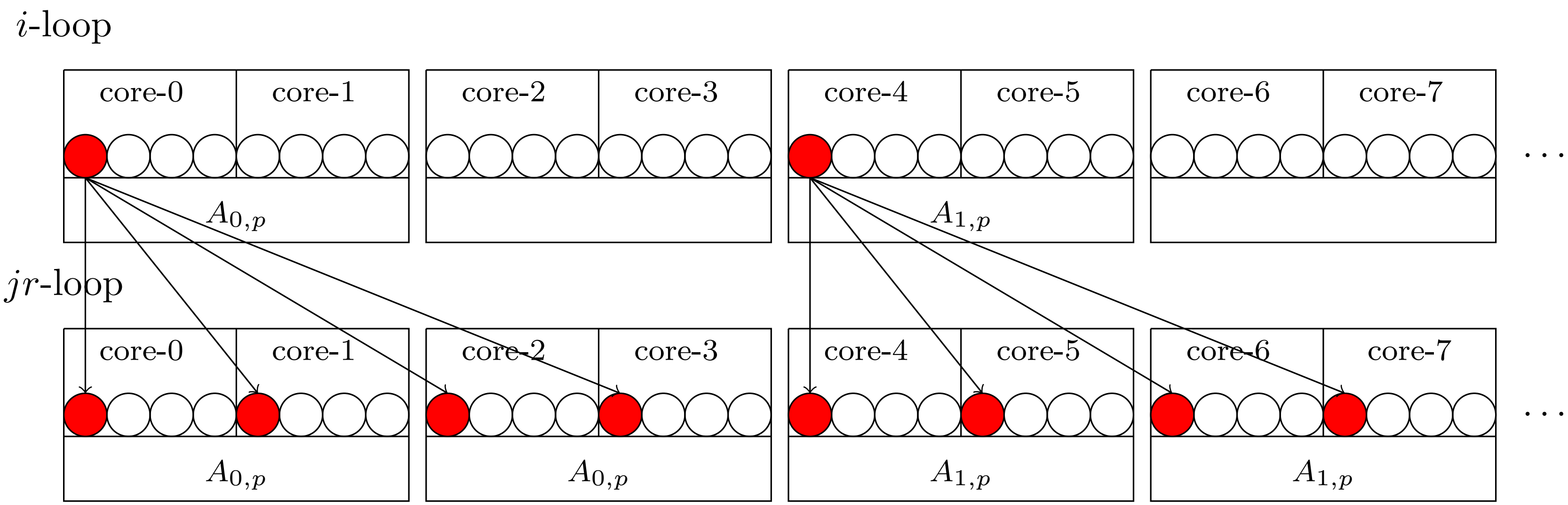
Thread affinity

We use only one threads per core.

	OpenMP* 4 Affinity	Intel OpenMP Runtime Extensions
Allocate hardware threads	OMP_PLACES	KMP_PLACE_THREADS
Pin OpenMP threads to hardware threads	OMP_PROC_BIND	KMP_AFFINITY

OpenMP Affinity

```
OMP_PLACES = cores
OMP_PROC_BIND = spread, spread
```

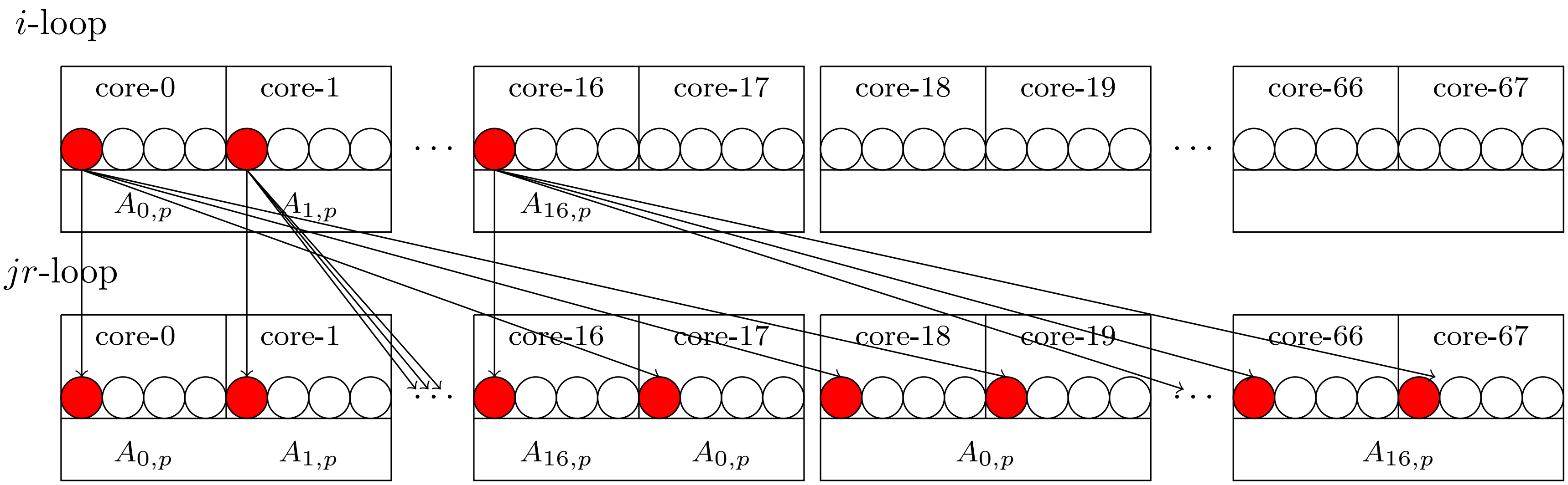


k_b can be determined using the follow inequality,

$$(m_b k_b + 2k_b n_r + 2m_r n_r) \times 8\ bytes \leq 512\ KB$$

Intel OpenMP runtime library

`KMP_AFFINITY = scatter`

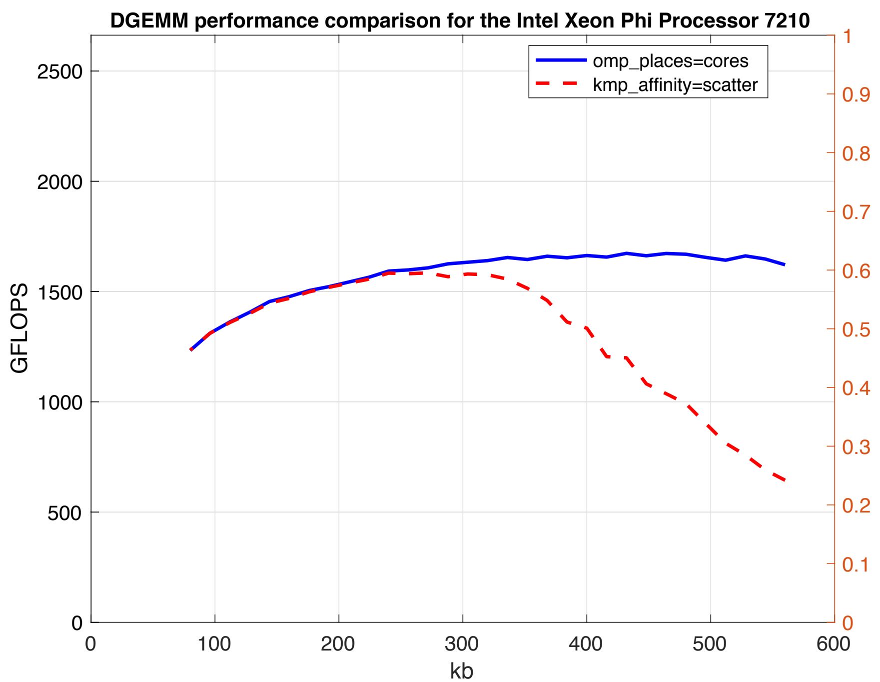


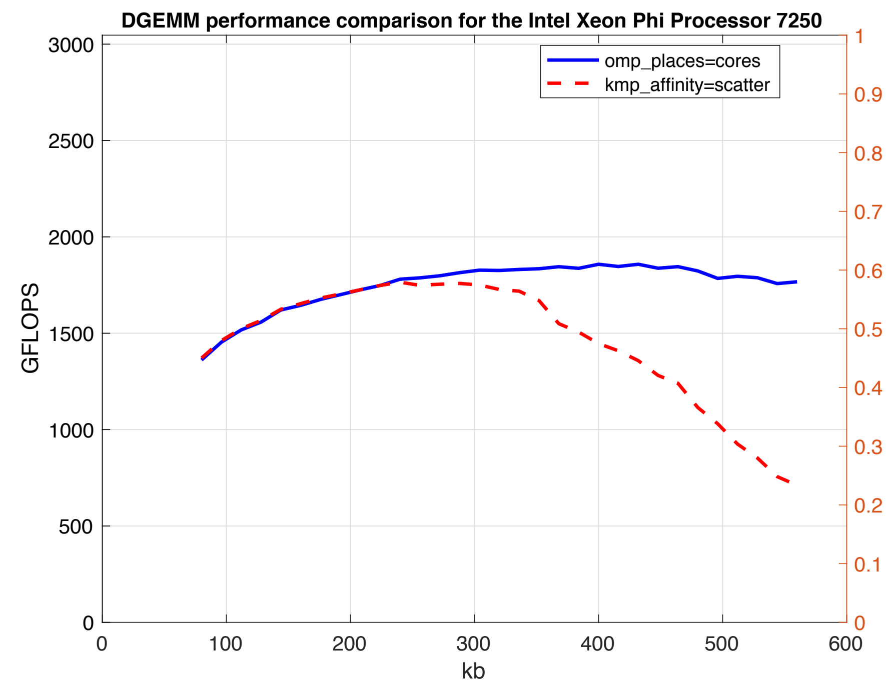
k_b can be determined using the follow inequality,

$$2\times(m_b k_b + k_b n_r + m_r n_r)\times 8\ bytes \leq 512\ KB$$

숭실대학교 Soongsil University

Xeon Phi 7210/7250





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

숭실대학교
Soongsil University

Thank you for your attention!