Just-in-time compilation: Speeding up small linear algebra operations

Sarah Knepper

Intel® Math Kernel Library (Intel® MKL)

25 May 2018, iWAPT 2018

Outline

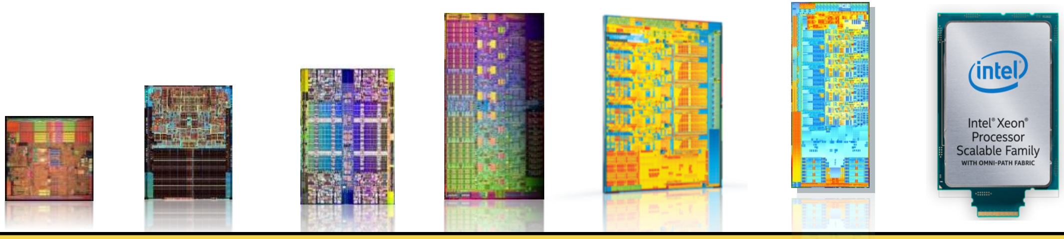
- Motivation
- Problem statement and solutions
- Simple example
- Performance comparison

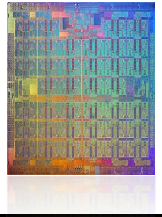
Motivation

- Partial differential equations
- FEM solvers
- Block sparse matrices
- Earthquake simulations
- Astrophysics
- Molecular dynamics
- Tensor contractions
- Machine learning
- Autonomous driving

Processor improvements

More cores → More Threads → Wider vectors





	Intel® Xeon® Processor 64-bit	Intel® Xeon® Processor 5100 series	Intel® Xeon® Processor 5500 series	Intel® Xeon® Processor 5600 series	Intel® Xeon® Processor E5-2600 v2 series	Intel® Xeon® Processor E5-2600 v3 series v4 series	Intel® Xeon® Scalable Processor[1]	Intel® Xeon Phi™ x200 Processor (KNL)
Up to Core(s)	1	2	4	6	12	18-22	28	72
Up to Threads	2	2	8	12	24	36-44	56	288
SIMD Width	128	128	128	128	256	256	512	512
Vector ISA	Intel® SSE3	Intel® SSE3	Intel® SSE4-4.1	Intel® SSE 4.2	Intel® AVX	Intel® AVX2	Intel® AVX-512	Intel® AVX-512

1. Product specification for launched and shipped products available on ark.intel.com.

Optimization Notice

Overheads for small sizes

- Low vectorization
- Low parallelization
- Loop trip count
- Non-local data access for large leading dimensions
- Error checking
- High function call overheads
  - Dispatching to ISA-specific codepath

Methods for improving performance for small sizes

- Specific kernels
- Compile-time optimizations
- Just-in-time (run-time) compilation
- Batching operations together
    - Modifying data layout

JIT compilers

- Xbyak (https://github.com/herumi/xbyak)
  - Used in Intel MKL and Intel MKL-DNN
- libxsmm back-end (https://github.com/hfp/libxsmm)
  - Used in libxsmm
- LLVM (https://llvm.org/docs/index.html)
- Others

Xbyak syntax

- Similar to Intel syntax:
  - mov eax, ebx → mov(eax, ebx);
  - vaddps xmm1, xmm2, xmm3 → vaddps(xmm1, xmm2, xmm3);
- Registers:
  - Xmm xmm0; Zmm zmm31; Reg32 eax; Zmm(i)
- Addressing:
  - mov(eax, ptr [ebx + ecx]); test(byte [esp], 4)
- Labels:
  - L(“L1”); //create label
  - jmp(“L1”); //jump to that label

Simple example: saxpy with unit increments
y = alpha * x + y

```
void saxpy_unit(int n, float alpha, float* x, float* y) {
    for (int i = 0; i < n; i++) {
        y[i] += alpha * x[i];
    }
}
```

JIT-ing saxpy_unit

- What parameter(s) will the generator take?
- What is the API for the generated kernel?

```
saxpy_unit_jit jc(n);
void (*saxpy_kernel)(float, float*, float*) =
       jc.getCode<void(*)(float, float*, float*)>();
saxpy_kernel(alpha, x, y);
```

Xbyak saxpy_unit - setup

```
#define ELE_IN_REG 16
class saxpy_unit_jit: public Xbyak::CodeGenerator {
    public: saxpy_unit_jit(int n): Xbyak::CodeGenerator()
{
    // convenience register variables
    auto reg_alpha = xmm0; // 1st parameter (alpha)
    auto reg_x = rdi;      // 2nd parameter (pointer to x)
    auto reg_y = rsi;      // 3rd parameter (pointer to y)
    auto alpha = zmm2;     // holds broadcasted alpha

    // broadcast alpha
    vbroadcastss(alpha, reg_alpha);
```

Xbyak saxpy_unit - main loop

```
// loop over full vector registers
for (int i = 0; i < (n / ELE_IN_REG); i++) {
    vmovups(zmm0, ptr[reg_x + sizeof(float)*(i*ELE_IN_REG)]);
    vmovups(zmm1, ptr[reg_y + sizeof(float)*(i*ELE_IN_REG)]);
    vfmadd231ps(Zmm(1), Zmm(0), alpha);
    vmovups(ptr[reg_y + sizeof(float)*(i*ELE_IN_REG)], zmm1);
}
```

Xbyak saxpy_unit - tail

```
 // finish off tail using masks
if ( n % ELE_IN_REG ) {
    auto full_regs = n / ELE_IN_REG;
    auto mask = (1 << (n % ELE_IN_REGISTER)) - 1;
    mov(eax, mask);
    kmovw(k1, eax);
    vmovups(zmm0 | k1, ptr[reg_x + sizeof(float)*(full_regs*ELE_IN_REG)]);
    vmovups(zmm1 | k1, ptr[reg_y + sizeof(float)*(full_regs*ELE_IN_REG)]);
    vfmadd231ps(zmm1, zmm0, alpha);
    vmovups(ptr[reg_y] | k1, zmm1);
}
```

Generated code

n = 32

```
vbroadcastss zmm2, xmm0
vmovups zmm0, zmmword ptr [rdi]
vmovups zmm1, zmmword ptr [rsi]
vfmadd231ps zmm1, zmm0, zmm2
vmovups zmmword ptr [rsi], zmm1
vmovups zmm0, zmmword ptr [rdi+0x40]
vmovups zmm1, zmmword ptr [rsi+0x40]
vfmadd231ps zmm1, zmm0, zmm2
vmovups zmmword ptr [rsi+0x40], zmm1
```

n = 42

```
vbroadcastss zmm2, xmm0
vmovups zmm0, zmmword ptr [rdi]
vmovups zmm1, zmmword ptr [rsi]
vfmadd231ps zmm1, zmm0, zmm2
vmovups zmmword ptr [rsi], zmm1
vmovups zmm0, zmmword ptr [rdi+0x40]
vmovups zmm1, zmmword ptr [rsi+0x40]
vfmadd231ps zmm1, zmm0, zmm2
vmovups zmmword ptr [rsi+0x40], zmm1
mov eax, 0x3ff
kmovw k1, eax
vmovups zmm0{k1}, zmmword ptr [rdi+0x80]
vmovups zmm1{k1}, zmmword ptr [rsi+0x80]
vfmadd231ps zmm1, zmm0, zmm2
vmovups zmmword ptr [rsi+0x80]{k1}, zmm1
```

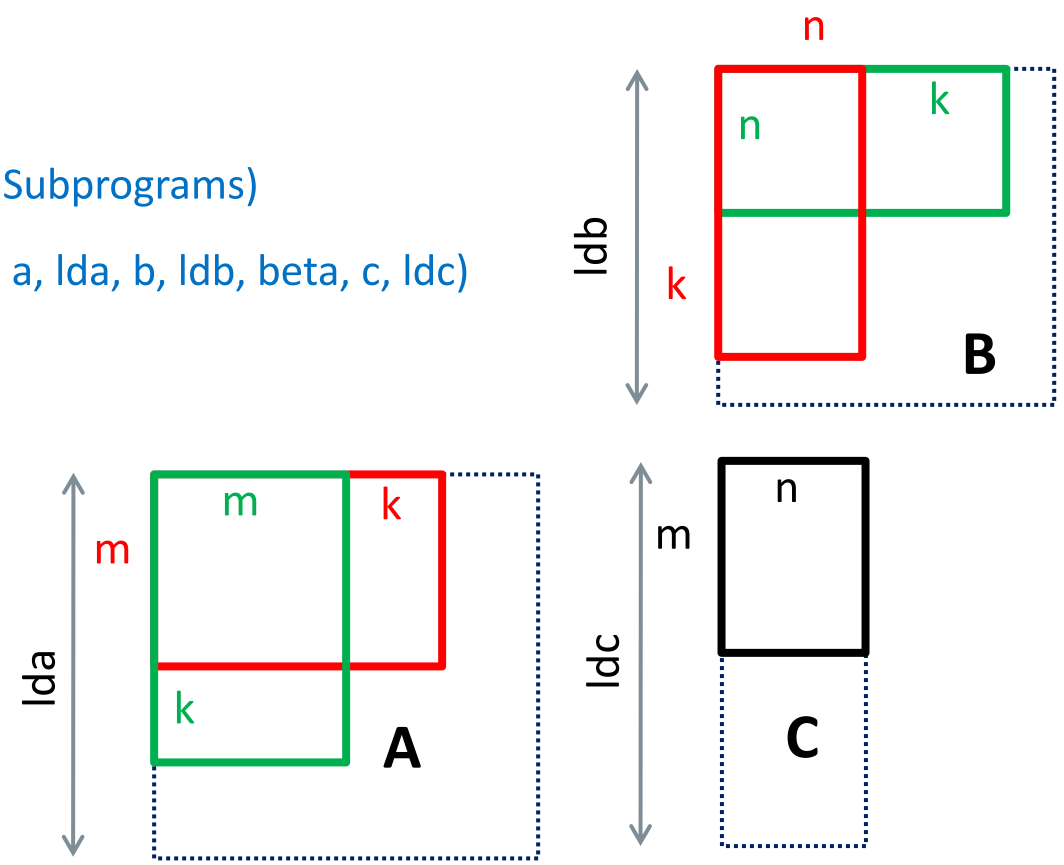
Matrix-matrix multiply

- Part of the BLAS (Basic Linear Algebra Subprograms)
- *GEMM(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

C = *alpha* op(**A**) * op(**B**) + *beta* **C**

op(**X**) = **X** or $\mathbf{X}^T$

```
C = beta*C
DO i=1,M
 DO j=1,N
  DO kk=1,K
   C(i,j) += alpha*A(i,kk)*B(kk,j)
  END DO
 END DO
END DO
```



Direct call compilation flags for Intel MKL

Define the preprocessor macro MKL_DIRECT_CALL or MKL_DIRECT_CALL_SEQ

- Instead of calling a library function, a C implementation may be used
- Starting from Intel MKL 2018.1, compiler intrinsics may be used for some kernels

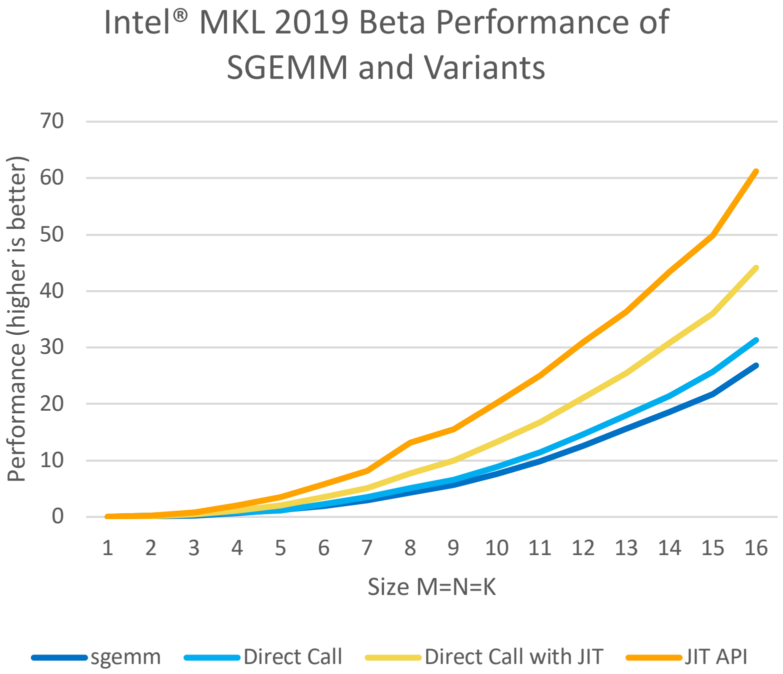
Starting from Intel MKL 2019 Beta: MKL_DIRECT_CALL_JIT or MKL_DIRECT_CALL_SEQ_JIT

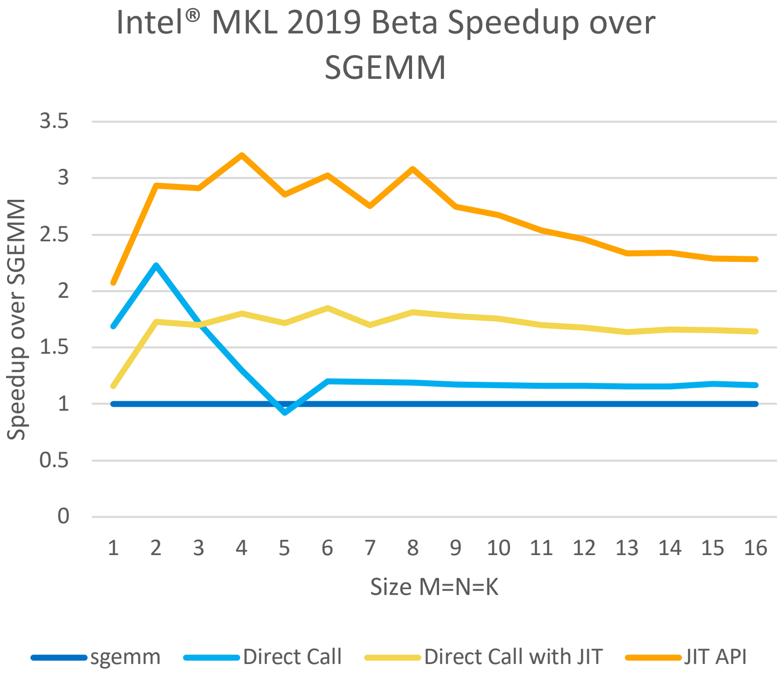
- A JIT-ted kernel may be used

```
// compile with: icc -DMKL_DIRECT_CALL …
#include <mkl.h>
void main(void) {
    dgemm(…);
}
```

```
! compile with: ifort -DMKL_DIRECT_CALL -fpp …
#     include “mkl_direct_call.fi”
      program DGEMM_MAIN
      DGEMM(…)
```

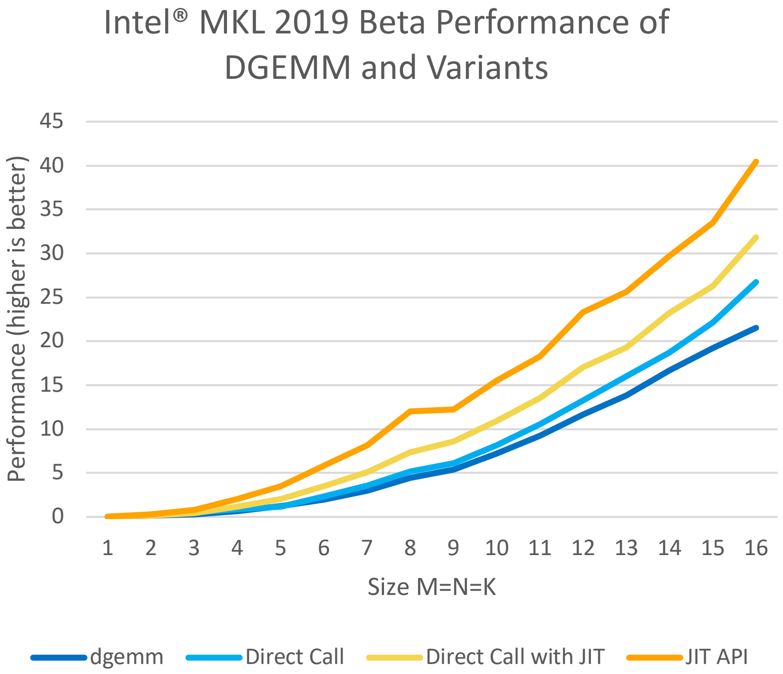
Performance of SGEMM on Intel® Xeon® Platinum

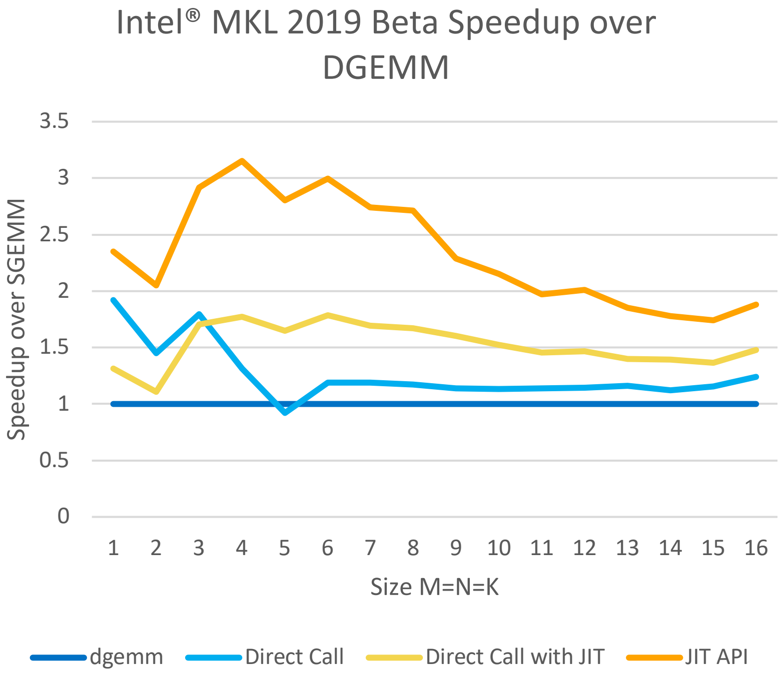




Configuration: Intel® Xeon® Platinum 8180, 2x28 cores, 2.5 GHz, 192 GB RAM, OS RHEL 7.2; Intel® MKL 2019 Beta. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Benchmark source: Intel Corporation. **Optimization Notice:** Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Performance of DGEMM on Intel® Xeon® Platinum





Configuration: Intel® Xeon® Platinum 8180, 2x28 cores, 2.5 GHz, 192 GB RAM, OS RHEL 7.2; Intel® MKL 2019 Beta. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Benchmark source: Intel Corporation. **Optimization Notice:** Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Summary

- Small linear algebra problems:
  - Are ubiquitous
  - Suffer performance overheads
- Just-in-time compilation can help:
  - Generator can create customized kernels for any parameters
- Performance gains can be significant

Resources

- Intel MKL Developer Reference: https://software.intel.com/en-us/articles/mkl-reference-manual
- Intel MKL Forum: https://software.intel.com/en-us/forums/intel-math-kernel-library
- No cost option for Intel MKL: https://software.intel.com/en-us/articles/free-mkl
- Intel MKL-DNN: https://github.com/01org/mkl-dnn
- Xbyak: https://github.com/herumi/xbyak
- libxsmm: https://github.com/hfp/libxsmm

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

intel®