Look Inside™

LIBXSMM

Library for small matrix multiplications.

Intel High Performance and Throughput Computing (EMEA)
Hans Pabst, March 12th 2015

Abstract

Library for small matrix-matrix multiplications targeting Intel Architecture (x86). The initial version of the library was targeting the Intel Xeon Phi coprocessor (an instance of the Intel Many Integrated Core Architecture "MIC") particularly by using KNC intrinsic functions (called KNCni or IMCI). Today, the library reaches the Many Integrated Core Architecture as well as other hardware which is capable of executing Intel Advanced Vector Extensions 512 (Intel AVX-512).

The library provides a sophisticated three-stage dispatch mechanism which is also targeting other instruction set extensions (beside of the Intrinsic code path), or falling back to an external LAPACK/BLAS library.

Motivation

“Improving Performance for Small Size Problems.”

Make informed tradeoffs and gain performance

Intel Math Kernel Library (Intel MKL)

DIRECT CALL feature: Intel MKL allows partial inlining of a small set of library functions. This feature may improve performance of computations with small matrices for a subset of configurations because error checking and several intermediate function calls can be skipped.

- Works for Intel and non-Intel compilers
- Works for C/C++ and Fortran
- Compile-time decision

Improve Performance for Small Size Problems.

Intel MKL Version 11.2: Tradeoffs and Limitations of the DIRECT CALL feature

BLAS-conformant error checking vs. low overhead

- No error checking or ‘xerbla’ callback

Code dispatch vs. compile-time decision

- AVX, AVX2, no MIC code path

Subset of functions

- xGEMM

Make informed tradeoffs and gain performance.

Intel MKL 11.2 DIRECT CALL Details

- Inline unrolled implementation for below-threshold sizes using non-Intel compilers or for very small problem sizes
- Inline low-level impl. in cases where the key arguments of the MKL function call are compile-time constants (Intel® Compiler)

DIRECT CALL addresses Intel and non-Intel compilers.

LIBXSMM

Interface (C API)

- Simple interface for matrix-matrix multiplications (no full xGEMM)
- Dispatched matrix-matrix multiplication
- Separate non-dispatched code paths
- Amortized dispatch

Details

- Intrinsics based code path for Intel Xeon Phi Coprocessors
- Open Source Software (BSD 3-clause license)*

Plan

- Intel AVX-512 code path, transposes, and higher level code optimizations
- Highly optimized assembly code generation

* https://github.com/hfp/libxsmm

LIBXSMM: Interface (C API)

```
/** If non-zero function pointer is returned, call (*function)(M, N, K). */
libxsmm_smm_function libxsmm_smm_dispatch(int m, int n, int k);
libxsmm_dmm_function libxsmm_dmm_dispatch(int m, int n, int k);

/** Automatically dispatched matrix-matrix multiplication. */
void libxsmm_smm(int m, int n, int k,
                 const float* a, const float* b,
                 float* c);
void libxsmm_dmm(int m, int n, int k,
                 const double* a, const double* b,
                 double* c);

/** Non-dispatched matrix-matrix multiplication using inline code. */
void libxsmm_simm(int m, int n, int k,
                 const float* a, const float* b,
                 float* c);
void libxsmm_dimm(int m, int n, int k,
                 const double* a, const double* b,
                 double* c);

/** Matrix-matrix multiplication using BLAS. */
void libxsmm_sblasmm(int m, int n, int k,
                 const float* a, const float* b,
                 float* c);
void libxsmm_dblasmm(int m, int n, int k,
                 const double* a, const double* b,
                 double* c);
```

LIBXSMM: Generating Code…

Usual mechanics

```
$ make ; make clean
$ make realclean
```

Column-major

```
$ make ROW_MAJOR=0
```

Specialization

```
$ make INDICES_M="2 4" INDICES_N="1" \
       INDICES_K="$(echo $(seq 2 5))“
```

Generates the following index set:

```
(2,1,2), (2,1,3), (2,1,4), (2,1,5),
(4,1,2), (4,1,3), (4,1,4), (4,1,5)
```

LIBXSMM: Getting Started

```
#include <libxsmm.h>

int main()
{
  const int m = 23, n = 23, k = 23;  /* some problem size */
  double a[m*k], b[k*n], c[m*n];     /* initialize later */
  libxsmm_dmm_function xmm = NULL;   /* see below */

  libxsmm_mm(m, n, k, a, b, c);      /* auto-dispatched */
  libxsmm_imm(m, n, k, a, b, c);     /* inlined */
  libxsmm_blasmm(m, n, k, a, b, c);  /* BLAS */
  libxsmm_dmm_23_23_23(a, b, c);     /* specialized */

  xmm = libxsmm_dmm_dispatch(23, 23, 23);
  if (xmm) {
    for (int i = 0; i < some; ++i) {
      xmm(a, b, c);                  /* amortized */
    }
  }
}
```

LIBXSMM: Code Dispatch Details

- Dispatch levels are accessible directly (for customized dispatch)
- Level 2 and 3 may be supplied by Intel MKL DIRECT CALL
- Amortize dispatch cost for mult. calls of same M, N, K

  `libxsmm_?mm_dispatch`
- Specific kernel access e.g.,

  `libxsmm_dmm_4_4_4`

Automatic code dispatch

1. **Specialized routine**
2. **Inlined code**
3. **BLAS call**

Dispatch-threshold for level 3 can be adjusted / tuned:

```
$   make THRESHOLD=$((24 * 24 * 24))
```

LIBXSMM: Intrinsics == Optimal?

Notes

- Code is just using Intrinsics but does not attempt to avoid issues such as register spilling (inappropriate unrolling, etc.)
- A well-optimizing compiler may generate better code compared even based on LIBXSMM's inline code path

Performance

- Intel Xeon Phi Coprocessor: LIBXSMM typically reaches ~4 GFLOPS/s per single core (may vary with actual M, N, K combination)
- Code is by no means "optimal" or "best-performing“
- (Upcoming AVX-512 assembly kernels may get above 90% of peak due to favorable ISA [unaligned LD/ST] and microarchitectural code tuning.)

References

Intel collaterals

Xeon Phi Applications and Solutions Catalog
http://software.intel.com/xeonphicatalog

3rd Party Tools and Libraries
https://software.intel.com/en-us/articles/intel-and-third-party-tools-and-libraries-available-with-support-for-intelr-xeon-phitm

Other

Performance engineering and code tuning (video/slide series)
http://user.cscs.ch/support/tutorials/2014/node_level_performance_engineering_15_16_may_2014/index.html

Optimized matrix transposes
http://research.colfaxinternational.com/post/2013/04/25/Transposition-Xeon-Phi.aspx

LIBXSMM home page
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®