BLISlab: A Sandbox for Optimizing GEMM

Jianyu Huang Robert A. van de Geijn

Draft
February 7, 2016

Abstract

1 Introduction

Matrix-matrix multiplication (GEMM) is frequently used as a simple example with which to raise awareness of how to optimize modern processors. A reason is that the operation is simple to describe, challenging to fully optimize, and of practical importance. In this paper, we walk the reader through the techniques that underly the currently fastest implementations for CPU architectures.

1.1 A minimal history

Need to mention BLAS3 [1] paper.

The advent of cache-based architectured, high-performance implementation of GEMM necessitated careful attention to the amortization of the cost of data movement between memory layers and computation with that data []. To keep this manageable, it helps to realize that only a "kernel" that performs a matrix-matrix multiplication with relatively small matrices needs to be highly optimized, since computation with larger matrices can be blocked to then use such a kernel without an adverse impact on overall performance. This insight was first explicitly advocated in

> Bo Køagström , Per Ling , Charles Van Loan.
> GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark.
> ACM Transactions on Mathematical Software (TOMS).
> Volume 24 Issue 3, p.268-302, Sept. 1998.

For more than a decade after that paper, the intricacies of high-performance optimization of GEMM was considered to be sufficiently complex that it should be left to to the hardware vendors, yielding IBM's ESSL, Intel's MKL, Cray's ???, and AMD's ACML libraries, or auto-generated as advocated in papers on the Portable High Performance ANSI C (PHiPAC) guidelines for writing high-performance matrix-matrix multiplication in C [] and the Automatically Tuned Linear Algebra Software (ATLAS) [].

Around 2000, Kazushige Goto revolutionized how GEMM is implemented on current CPUs with his techniques that were first published in the paper

> Kazushige Goto, Robert A. van de Geijn.
> Anatomy of high-performance matrix multiplication.
> ACM Transactions on Mathematical Software (TOMS).
> Volume 34 Issue 3, May 2008, Article No. 12.

At the end on this note we will discuss the major insights in this paper.

1.2 The BLIS-like Library Instantiation Software (BLIS)

More recently, the BLAS-like Library Instantiation Software (BLIS) "refactored" the approach pioneered by Goto, exposing additional loops around a *micro-kernel*, as described in

> Field G. Van Zee, Robert A. van de Geijn.
> BLIS: A Framework for Rapidly Instantiating BLAS Functionality.
> ACM Transactions on Mathematical Software (TOMS).
> Volume 41 Issue 3, June 2015, Article No. 14.

One goal of the BLIS paper was to further expose the layering of Goto's approach while simultaneously improving portability by reducing how much code must be written at a low level (e.g., in assembly code).

1.3 You too can optimize like a pro

The purpose of this note is to expose the basic techniques that underlie the best implementations of GEMM so that you too can achieve high-performance for such operations.

2 Step 1: The Basics

2.1 Simple matrix-matrix multiplication

In our discussions, we will consider the computation

$$C := AB + C$$

where A, B, and C are $m \times k$, $k \times n$, $m \times n$ matrices. respectively. Letting

$$A = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{0,k-1} \\ \vdots & & \vdots \\ \alpha_{m-1,0} & \cdots & \alpha_{m-1,k-1} \end{pmatrix}, B = \begin{pmatrix} \beta_{0,0} & \cdots & \beta_{0,n-1} \\ \vdots & & \vdots \\ \beta_{k-1,0} & \cdots & \beta_{k-1,n-1} \end{pmatrix}, \text{ and } C = \begin{pmatrix} \gamma_{0,0} & \cdots & \gamma_{0,n-1} \\ \vdots & & \vdots \\ \gamma_{m-1,0} & \cdots & \gamma_{m-1,n-1} \end{pmatrix}.$$

$C := AB + C$ computes

$$\gamma_{i,j} := \sum_{p=0}^{k-1} \alpha_{i,p}\beta_{p,j} + \gamma_{i,j}.$$

If A, B, and C are stored as floating point numbers in two-dimensional arrays `A`, `B`, and `C`, the following pseudocode computes $C := AB + C$:

```
for i=0:m-1
   for j=0:n-1
      for p=0:k-1
         C( i,j ) := A( i,p ) * B( p,j ) + C( i,j )
      endfor
   endfor
endfor
```

Counting a multiply and an add separately, the computation requires $2mnk$ floating point operations flops.

2.2 Set up

To let you efficiently learn about how to efficiently compute, you start your project with much of the infrastructure in place. We have structured the subdirectory, `step1`, somewhat like a project that implements a real library might. This may be overkill for our purposes, but how to structure a software project is a useful skill to learn.

Consider Figure 1, which illustrates the directory structure for subdirectory `step1`:

```
step1
├── README
├── makefile
├── sourceme.sh
├── dgemm
│   ├── bl_dgemm.c
│   ├── bl_dgemm_ref.c
│   └── bl_dgemm_util.c
├── include
│   ├── bl_dgemm.h
│   ├── bl_dgemm_ref.h
│   └── bl_config.h
├── lib
├── makefile.in.files
│   ├── make.intel.inc
│   ├── make.gnu.inc
│   └── make.inc
└── test
    ├── makefile
    ├── test_bl_dgemm.c
    ├── run_bl_dgemm.sh
    ├── test_bl_dgemm.x
    └── tacc_run_bl_dgemm.sh
```

Figure 1: Structure of directory `step1`.

`README` Is a file that describes the contents of the directory.

`dgemm` Is the subdirectory routines that implement GEMM can be found. In it

- `bl_dgemm_ref` contains the routine `dgemm_ref` that is a simple implementation of GEMM that you will use to check the correctness of your implementations.
- `my_dgemm` contains the routine `dgemm` that that initially is a simple implementation of GEMM and that you will optimize as part of the first step on your way to mastering how to optimize GEMM.
- `bl_dgemm_util` contains utility routines that will come in handy later.

`include` This directory contains include files with various macro definitions and other header information.

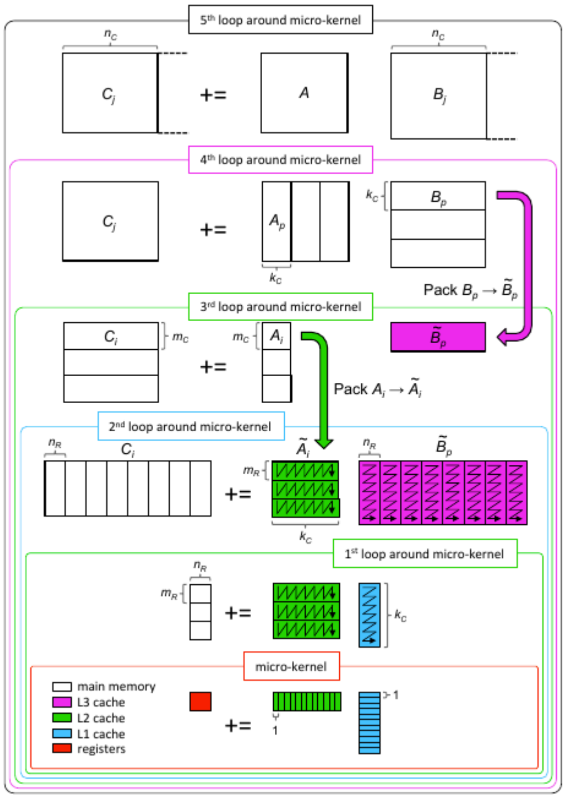
`lib` This directory will hold libraries that you will install so you can compare your performance with existing high-performance solutions.

`test` This directory contains "test drivers" for the various implementations.

3 Step 2: Blocking

4 The Goto Approach to Implementing GEMM

[2] [6] [5] [3] [4]



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
$\mathcal{P}_c = p_c : p_c + k_c - 1$
$B(\mathcal{P}_c, \mathcal{J}_c) \to B_c$ // Pack into B_c
Loop 3 **for** $i_c = 0 : m-1$ **steps of** m_c
$\mathcal{I}_c = i_c : i_c + m_c - 1$
$A(\mathcal{I}_c, \mathcal{P}_c) \to A_c$ // Pack into A_c
// 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** $k_r = 0 : k_c - 1$
$C_c(\mathcal{I}_r, \mathcal{J}_r)$
$\mathrel{+}= A_c(\mathcal{I}_r, k_r) \;\; B_c(k_r, \mathcal{J}_r)$
endfor
endfor
endfor
endfor
endfor

Figure 2: Left: The Goto algorithm for matrix-matrix multiplication as refactored in BLIS. Right: the same algorithm, but expressed as loops.

```
Users
└─ rvdg
   └─ blislab ........ The main directory.
      ├─ docs ........ The directory where this document and its LaTeX source exist.
      └─ src ......... Directory where the source for the programming exercises exists.
         ├─ step1 ......... Source for step one and two of the exercise.
         ├─ step3 ......... Source for step three of the exercise.
         └─ step_final .... Source for final step of the exercise.
```

Figure 3: Directory structure for the materials. In this example, I placed `BLISF1.zip` in the home directory `Users/rvdg` before unzipping. You may want to place it on your account's "Desktop" instead.

5 Organization of the Project

One goal of this exercise is to teach the reader how a software library project is often organized into directories. We acknowledge that the structure is probably overkill for this relatively simple situation, but hope that it has value nonetheless.

6 Step-by-step approach to the peak performance

Architecture: Ivy Bridge and Haswell.

6.1 Naive Approach: Three loops

6.2 Cache Blocking: 6 loops

refer to GOTO paper: How to permuate to get the best loop order var2, var1, var3

Performance Graph

6.3 Add Packing

6.4 Micro-kernel Tricks

1. Butterfly or Broadcasting?
2. Double buffering

6.5 Parameter Tuning

6.6 Parallelization

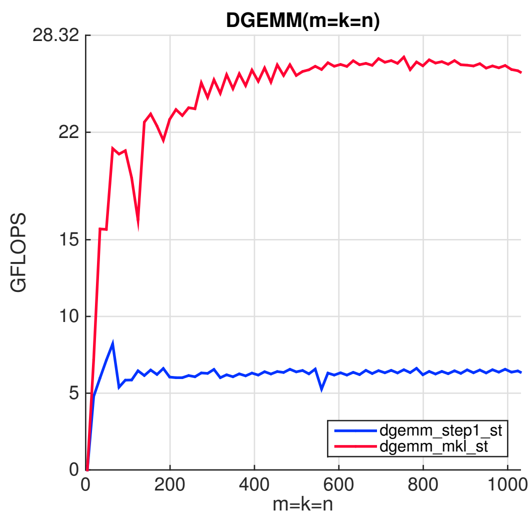


Figure 4: Step1 performance

7 Conclusion

Conclusion.

Additional information

For additional information on FLAME visit

`http://www.cs.utexas.edu/users/flame/`.

Acknowledgements

We thank the other members of the FLAME team for their support. This research was partially sponsored by NSF grant CCF–***.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

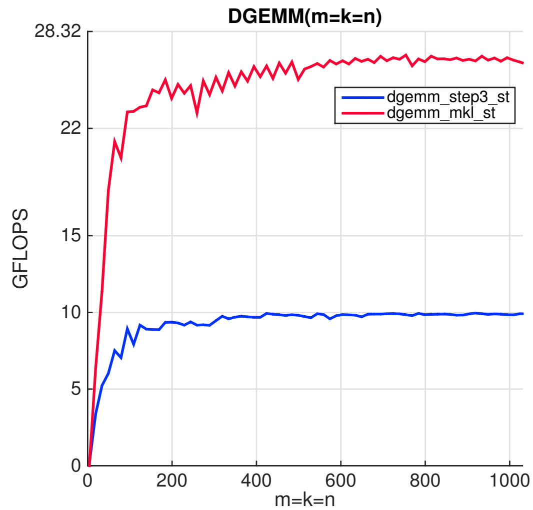


Figure 5: Step3 performance

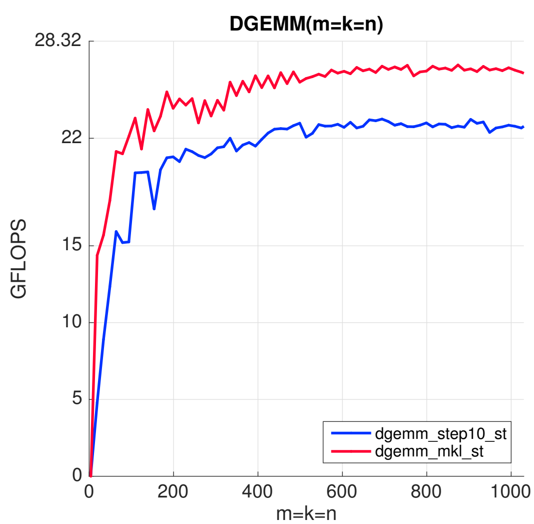


Figure 6: Step10 performance

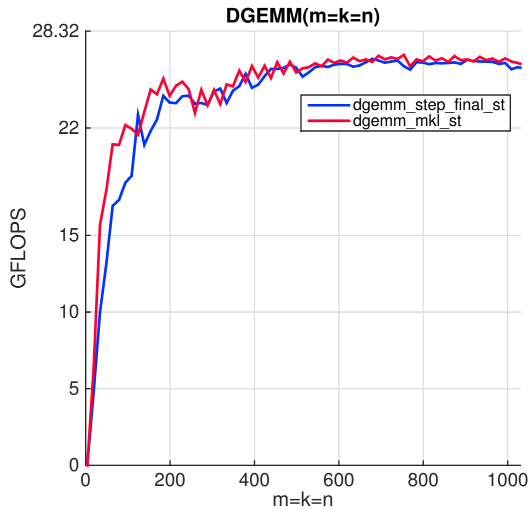


Figure 7: Step Final performance

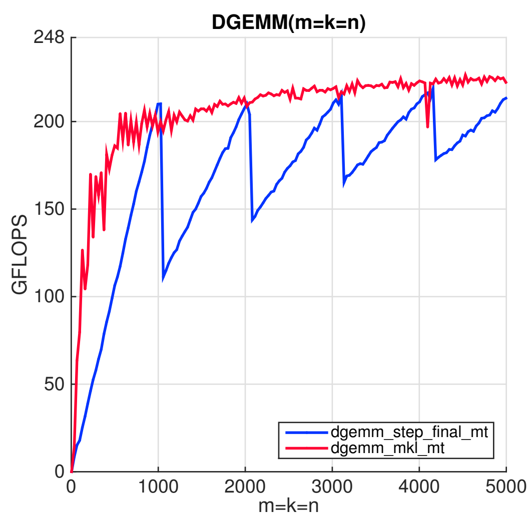


Figure 8: Step Final performance (multi-thread)

References

[1] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[2] Kazushige Goto and Robert A. van de Geijn. Anatomy of a high-performance matrix multiplication. *ACM Trans. Math. Soft.*, 34(3):12, May 2008. Article 12, 25 pages.

[3] Tyler M. Smith, Robert van de Geijn, Mikhail Smelyanskiy, Jeff R. Hammond, , and Field G. Van Zee. Anatomy of high-performance many-threaded matrix multiplication. In *International Parallel and Distributed Processing Symposium 2014*, 2014.

[4] Tze Meng Low, Francisco D. Igual, Tyler M. Smith, and Enrique S. Quintana-Ortí. Analytical modeling is enough for high performance blis. *ACM Transactions on Mathematical Software*. in review.

[5] Field G. Van Zee, Tyler Smith, Bryan Marker, Tze Meng Low, Robert A. van de Geijn, Francisco D. Igual, Mikhail Smelyanskiy, Xianyi Zhang, Michael Kistler, Vernon Austel, John Gunnels, and Lee Killough. The blis framework: Experiments in portability. *ACM Transactions on Mathematical Software*. to appear.

[6] Field G. Van Zee and Robert A. van de Geijn. BLIS: A framework for rapidly instantiating blas functionality (replicated computational results certified). *ACM Trans. Math. Soft.*, 41(3):14:1–14:33, June 2015.