An Exercise in Optimizing Matrix-Matrix Multiplication

Jianyu Huang Robert A. van de Geijn

Draft
January 31, 2016

Abstract

Abstract.

1 Introduction

In this exercise, we are going to optimize GEMM (General Matrix-Matrix Multiplication) on modern computer architectures. Specifically, we consider the simple case:

$$C := C + A * B$$

where A, B, and C are $m \times k$, $k \times n$, $m \times n$ matrices, respectively. GEMM can be performed using $2mkn$ floating point operations, as illustrated in the following pseudocode:

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

2 The Goto Approach to Implementing GEMM

[1] [5] [4] [2] [3]

3 Step-by-step approach to the peak performance

Architecture: Ivy Bridge and Haswell.

3.1 Naive Approach: Three loops

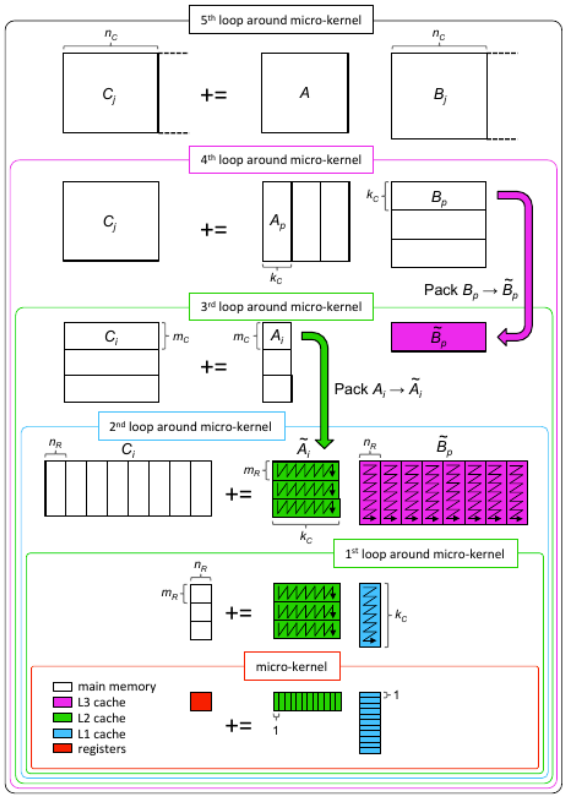
3.2 Cache Blocking: 6 loops

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

3.3 Add Packing

3.4 Micro-kernel Tricks

1. Butterfly or Broadcasting?
2. Double buffering



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

Figure 1: Left: The Goto algorithm for matrix-matrix multiplication as refactored in BLIS. Right: the same algorithm, but expressed as loops.

3.5 Parameter Tuning

3.6 Parallelization

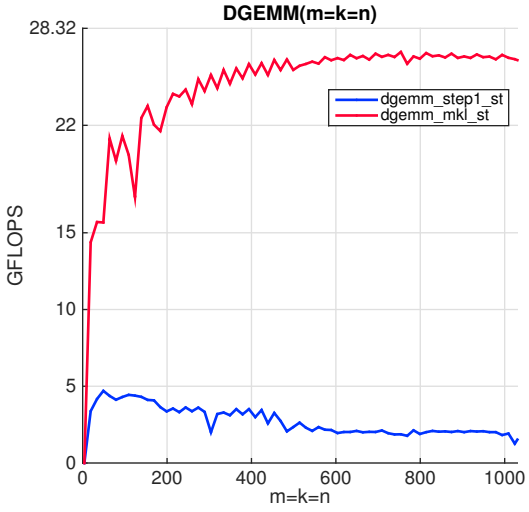


Figure 2: Step1 performance

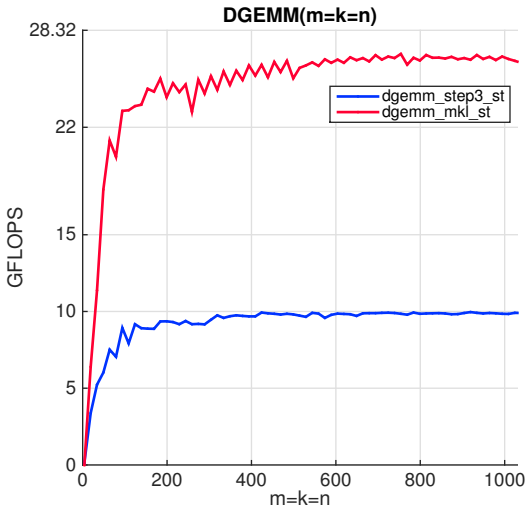


Figure 3: Step3 performance

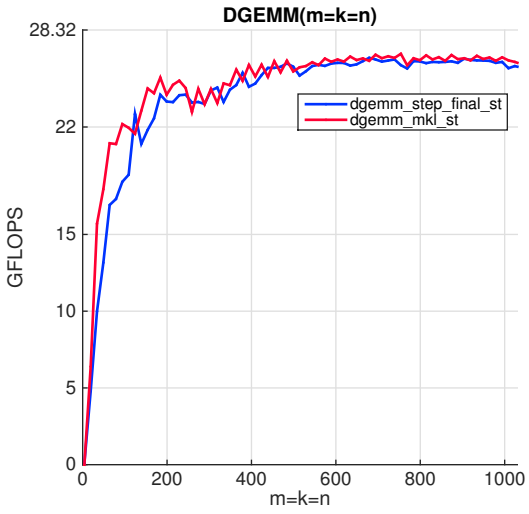


Figure 4: Step Final performance

4 Conclusion

Conclusion.

Additional information

For additional information on FLAME visit

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

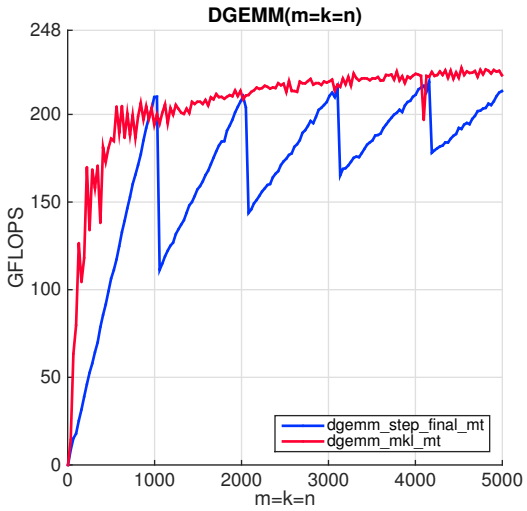


Figure 5: Step Final performance (multi-thread)

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).

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.