Using Machine Learning to Improve Shared-Memory General Matrix-Matrix Multiplication

Brandon M. Groth and Glenn R. Luecke
Department of Mathematics, Iowa State University, Ames, IA, USA

Abstract— *General matrix-matrix multiplication (GEMM) is frequently used to train neural networks for machine learning applications. For example, the backpropagation algorithm, which is implemented via matrix multiplication, is used to train feed-forward neural networks. The GEMM algorithm in Intel's Math Kernel Library (MKL) sometimes does not perform well on matrices required by backpropagation. To address this problem, the authors selected six GEMM simple algorithms and compared their performance to Intel's GEMM. This was accomplished by using a multi-class algorithm selector for finding the fastest of these seven algorithms. The algorithm selection was shown to be 93% accurate and the maximum speedup of non-MKL algorithms over MKL ranged from 3.77x to 11.22x. Our library, ML_GEMM, gave a maximum speedup of 23.83x over MKL.*

Keywords: HPC, Matrix Multiplication, GEMM, Algorithm Selection

1. Introduction

The Basic Linear Algebra Subprograms (BLAS) were developed to provide standard building blocks for performing basic vector and matrix operations [1]. These subprograms were designed to be implemented on various hardware to facilitate the fast and portable computation of linear equations solvers, matrix decompositions, eigenvalue solvers, etc. In particular, the subroutine called General Matrix Multiplication (GEMM) is the BLAS routine used for computing a variety of matrix-matrix multiplications. In this paper, the authors consider only single precision matrix-matrix multiplication (SGEMM) to compute $C \leftarrow \alpha AB + \beta C$ with $\alpha, \beta \neq 0$. GEMM is used extensively in scientific computing and more recently in machine learning (e.g. training of neural networks). Computer vendors provide optimized versions of GEMM for their hardware, such as Intel's Math Kernel Library (MKL). Intel's MKL provides both serial and multi-threaded versions of GEMM. Unfortunately, at this time, GEMM in Intel's MKL does not provide good performance for some matrices. This is problematic for some fields in machine learning. For example, deep learning uses highly rectangular matrices in training feed-forward neural networks [2]. The purpose of this work is to provide a new GEMM library call, named ML_GEMM, that overcomes some of these performance problems on shared-memory computers. This is accomplished by comparing seven algorithms of GEMM (including Intel's GEMM) and then use machine learning techniques to decide which algorithm gives the best performance for a given matrix dimension.

2. Methodology

There are many ways to compute GEMM, such as a library call or by manually coding the algorithm. Memory stride, cache awareness, parallelism, and the compiler (including compiler options) can have a major impact on the algorithm's performance. The following are the seven algorithms used in this paper:

- Algorithm 1 - Basic
- Algorithm 2 - Basic Transpose
- Algorithm 3 - Dot
- Algorithm 4 - Dot Transpose
- Algorithm 5 - Matmul (Fortran Intrinsic)
- Algorithm 6 - MKL
- Algorithm 7 - MKL Transpose

See Section 6 for the code used for each algorithm. For the Basic and Dot algorithms, the order of the loop indices's for $(AB)_{ij} = \sum_{l=1}^{k} A_{il}B_{lj}$ was chosen to give stride-1 memory for the columns of B.

The transpose versions of these methods use a double transpose on matrix A to compute $C \leftarrow \alpha(A^T)^T B + \beta C$. These transpose variations are considered because they give stride-1 memory access for the rows of A even though they require additional memory. This allows compilers to use vectorization, which may give a speedup that outweighs the cost for computing a matrix transpose. For this paper, the authors define the group of non-MKL algorithms to be Algorithms 1-5 and Algorithm 7. The MKL Tranpose algorithm is included in the non-MKL group because the transpose modification differentiates it from a typical MKL call.

Due to increasing processor core counts, developers are using multi-threading APIs, such as OpenMP, to parallelize their applications [3]. In this paper, the authors choose to use OpenMP for mutli-threading our GEMM implementations.

The computing resource that we choose to use for our experiments was the Condo HPC Cluster located at Iowa State University. Each node that was used on Condo had two 2.6 GHz 8-Core Intel E5-2640 v3 processors (Haswell) with 128 GB of memory. The operating system used on Condo is

Red Hat Enterprise Linux 7.3 and the compiler used for all Fortran code was Intel 16.0. While Intel 17.x compilers were available, a bug prevented the use of Matmul with more than one OpenMP thread using the `-qopt-matmul` compiler option. The full list of compiler options used are:

```
ifort -parallel -mkl=parallel -qopt-matmul
  -qopenmp -Ofast -xHost -ipo <source>
```

For descriptions of each compiler option, see the Intel Fortran Compiler Developer Guide and Reference.

To use OpenMP on the nested matrix loops, the following code was added before the do loops:

```
!$OMP PARALLEL DO SHARED(<A>,B,C)
    PRIVATE(<priv_vars>) SCHEDULE(static)
```

where <A> is A or A^T and <priv_vars> contain loop indices and temporary variables inside the do loops. The OpenMP environment variables used for experiments were `OMP_NUM_THREADS=16`, and `OMP_PROC_BIND=true`. Please see the OpenMP specification for details of these environment variables.

In this project, the execution-time data was generated in Fortran, while data processing was done in R using the Caret package [4]. Caret has support for a large number of machine learning algorithms, and does so with a common template for parameter search, training, testing, etc. The Caret framework allows users to train and test many different models within a single script.

3. Algorithm Selection

The data collected for this experiment were given as `m,n,k,algorithm`, where `m,n,k` are the matrix dimensions and `algorithm` is an integer label corresponding to the GEMM algorithm that gave the minimum average execution time. Let $S = \{2^3, 2^4, \ldots, 2^{16}\}$. In context of supervised learning, the feature space is defined as $\vec{x} = (m, n, k) \in S^3$, and the target output is $y = algorithm \in \{1, 2, \ldots, 7\}$. This formulation defines a multi-class classification problem, where given input $\vec{x}$, we want to predict the target output y. Thus, for some classifier $f : S^3 \rightarrow \{1, 2, \ldots, 7\}$, we want to train f such that for each training example $(\vec{x}_i, y_i)$, $f(\vec{x}_i) = y_i$.

3.1 Data Set Analysis

For each GEMM algorithm, the authors created a $14 \times 14 \times 14$ cube of data for GEMM timings with matrix dimensions in S. At each index of this data cube, the best algorithm is labeled based on the lowest average execution time of the seven GEMM algorithms. The authors were able to successfully run 2640 of the 2744 possible grid points. The 104 missing grid points were due to insufficient memory to run the test.

Table 1 presents a speedup analysis of the seven GEMM algorithms. Each row represents when the denominator algorithm was labeled the fastest, giving a minimum speedup of 1.00x. The authors compared each non-MKL algorithm to MKL (rows 1-6), and MKL was compared to the best non-MKL algorithm found (row 7). The non-MKL algorithms gave maximum speedup ranging from 2.82x-11.22x over MKL. Similarly, MKL gave a maximum speedup of 41.55x over our non-MKL algorithms. From the table, the MKL algorithm was optimal for a majority of cases, but the collection of simple algorithms were superior 12.8% of the time. Furthermore, the speedups of these algorithms over MKL is given in Table 1 when a non-MKL method was selected.

3.2 The C5.0 Classifier

Due to the computation constraints of generating the data set in S^3, the classifier will only see each training example $(\vec{x}_i, y_i)$ once. This is a problem for splitting the data set into distinct training and test sets. Data splitting in this case will lower classification accuracy, especially in regions that classify as different algorithms. For our application, having a high accuracy classifier is more important than other considerations like over-fitting or bias, since picking the wrong GEMM algorithm can lead to poor performance. This will be done by training and testing with the entire data set.

The package we choose to use with Caret was C5.0, which learns a decision tree based on the data given [5]. A decision tree was generated via C5.0 on the data set, which is summarized in Table 2. The most important aspects of this tree is that it has an accuracy of 93% on the entire data set and can return a query in 2.99×10^{-5} seconds on average. Thus, our classifier has a very small overhead. To choose a classifier that was accurately selecting the instances where non-MKL matrix multiplication methods were superior, the authors looked at the corresponding confusion matrix (Table 3). A confusion matrix shows the number correct classifications along the diagonal, while the number of wrong classifications are on off-diagonal entries for each column. From the Confusion matrix, the decision tree does a good job classifying Algorithms 1 through 4, but does poor job classifying Algorithms 5 and 7. While this isn't ideal, other classifiers tested from Carrot were unable to correctly classify non-MKL algorithms. This is because each non-MKL algorithms would be considered a "rare event" compared to MKL in the data set.

3.3 ML_GEMM

Let ML_GEMM be the name of this algorithm selector implementing the C5.0 decision tree in Fortran. See Listing 1 for the pseudo code of ML_GEMM. The authors tested the ML_GEMM against Intel's MKL using only a single time trial. The same grid points were used for this results

Table 1: Speedup Results for GEMM Algorithms. Each row represents when the denominator GEMM algorithm was fastest.

Speedup Computation	Speedup Bins				Max Speedup
	1.0-1.5	1.5-2.0	2.0-3.0	>3.0	
MKL / Basic	16	12	1	6	11.22
MKL / Basic Transpose	57	8	2	2	4.06
MKL / Dot	47	21	7	1	3.77
MKL / Dot Transpose	59	14	3	0	2.82
MKL / Matmul	13	1	0	1	4.25
MKL / MKL Transpose	60	4	3	1	7.23
Best non-MKL / MKL	1217	239	201	644	41.55

Table 2: C5.0 Decision Tree Properties

Tree Properties	Value
If Statements	44
Min Depth	2
Max Depth	8
Avg Depth	5.32
Accuracy	92.99%
Avg Query Time	2.99e-5 s

	y_i						
$f(\vec{x}_i)$	1	2	3	4	5	6	7
1	15	0	4	0	0	0	0
2	1	42	0	19	0	1	0
3	10	0	53	1	0	2	0
4	0	18	0	43	0	9	4
5	0	0	1	0	4	0	0
6	9	7	17	13	11	2287	53
7	0	2	1	0	0	2	11

Table 3: C5.0 Confusion Matrix. Rows indicate class predictions of the classifier $f(\vec{x}_i)$ and columns represent the true class values y_i.

phase. The results are given in Table 4. Since ML_GEMM can classify as MKL, it was important to know how many instances had near equal execution times. This corresponds to the the 1.00-1.025x speedup bin for each method. Our library ML_GEMM was noticeably superior (>1.025x speedup) 10.6% of the time. The maximum speedups observed were 23.83x for ML_GEMM and 43.41x for MKL.

Listing 1: ML_GEMM Pseudo Code

```
SUBROUTINE ML_GEMM(m,n,k,alpha,beta,A,B,C)
  Compute mPow,nPow,kPow
  algorithm=gemmClassifier(mPow,nPow,kPow)
  SELECT CASE (algorithm)
    cases 1-7: Compute GEMM via algorithm
    default: Compute GEMM via MKL
  END SELECT
END SUBROUTINE ML_GEMM
```

4. Related Work

Algorithm selection was a technique developed in 1976 to help define when a particular algorithm "works best" [6]. In the past, developing a good selector required domain expertise to select subsets of the feature space for each algorithm used. Recently, developers are employing machine learning techniques to create algorithm selectors for various problems (such as the SAT problem [7]).

Parallel matrix-matrix multiplication algorithm development has been ongoing for decades. The SUMMA algorithm presented by van de Geijn and Watts is a well-known distributed algorithm that rivaled ScaLAPACK in 1995 [8]. More recent work uses recursion and minimal communication, such as the CARMA algorithm created by Demmel et al [9]. CARMA uses divide-and-conquer techniques on blocks of the matrices to perform the matrix multiplication as opposed to traditional loops. This strategy was shown to have better performance than Intel's MKL in 2013.

To the author's knowledge, the first paper to use machine learning to aid matrix multiplication was by Spillinger et al. [10]. Spillenger et al used the support vector machine (SVM) algorithm to choose a library to perform a dense matrix multiplication the fastest. The libraries that the authors choose to compare were Intel MKL and the CARMA algorithm.

Next, Shaohuai Shi et al. examined computing $C = AB^T$ on a GPU [11]. Instead of choosing between two different libraries, the authors choose between using two versions of the cuBLAS GEMM algorithm. The first cuBLAS version transposes matrix B inside GEMM, while the other version pre-transposed the matrix B, then used a standard GEMM matrix multiply. They then embedded their matrix multiply classifier within the Caffe deep learning framework to improve performance of image classification.

5. Conclusion

In this work, the authors have employed an algorithm selector developed via machine learning techniques to improve matrix-matrix multiplication. A multi-class classifier framework is provided to compute the General Matrix-matrix Multiplication (GEMM) in parallel on shared-memory computers. As a proof-of-concept, the authors selected six simple algorithms and compared them with Intel's MKL GEMM. These simple algorithms collectively outperformed MKL 12.8% of the time for matrix dimensions $m, n, k \in \{2^3, 2^4, \ldots, 2^{16}\}$. When non-MKL algorithms were selected, they exhibited an maximum speedup of 2.82-11.22x

Table 4: ML_GEMM vs MKL Speedup Analysis. Each row represents when the denominator algorithm was selected as fastest.

Speedup Computation	Speedup Bins					Max Speedup
	1.0-1.025	1.025-1.5	1.5-2.0	2.0-3.0	>3.0	
MKL / ML_GEMM	265	225	32	14	7	23.83
ML_GEMM / MKL	710	1316	117	28	18	43.41

over MKL. Our algorithm selector, named ML_GEMM, was created with the C5.0 decision tree classification model to select the fastest algorithm for a given m, n, k. This algorithm selection was shown to be 93% accurate on the data set. ML_GEMM was shown to be superior to strictly using MKL 10.6% of the time, with a maximum speedup over MKL of 23.83x.

The next step in this research is to develop and replace the author's simple algorithms with better algorithms. Additional experiments are planned to be used on the Intel Xeon Phi Knights Landing (KNL) nodes.

6. Appendix

Algorithm 1: Basic

```
!$OMP PARALLEL DO SHARED(A,B,C) PRIVATE(i,j,l,sum)
     SCHEDULE(static)
do j=1,n
  do i=1,m
    sum=0.0
    do l=1,k
      sum= sum+A(i,l)*B(l,j)
    enddo
    C(i,j)=alpha*sum+beta*C(i,j)
  end do
end do
```

Algorithm 2: Basic Transpose

```
ALLOCATE(Atr(k,m))
Atr = TRANSPOSE(A)
!$OMP PARALLEL DO SHARED(Atr,B,C) PRIVATE(i,j,l,
    sum) SCHEDULE(static)
do j=1,n
  do i=1,m
    sum=0.0
    do l=1,k
      sum=sum+Atr(l,i)*B(l,j)
    end do
    C(i,j)=alpha*sum+beta*C(i,j)
  end do
end do
DEALLOCATE(Atr)
```

Algorithm 3: Dot

```
!$OMP PARALLEL DO SHARED(A,B,C)
   PRIVATE(i,j) SCHEDULE(static)
do j=1,n
  do i=1,m
    C(i,j)=alpha*DOT_PRODUCT(
           A(i,1:k),B(1:k,j))
           +beta*C(i,j)
  end do
end do
```

Algorithm 4: Dot Transpose

```
ALLOCATE(Atr(k,m))
Atr=TRANSPOSE(A)
!$OMP PARALLEL DO SHARED(Atr,B,C) PRIVATE(i,j)
    SCHEDULE(static)
do j=1,n
  do i=1,m
    C(i,j)=alpha*DOT_PRODUCT(
           Atr(1:k,i),B(1:k,j))
           +beta*C(i,j)
  end do
end do
DEALLOCATE(Atr)
```

Algorithm 5: Matmul

```
! Uses -qopt-matmul and -parallel -qopenmp
    compiler options
C=alpha*MATMUL(A,B)+beta*C
```

Algorithm 6: SGEMM

```
! Uses -mkl=parallel compiler option
CALL SGEMM('N','N',m,n,k,alpha,A,m,B,k,beta,C,m)
```

Algorithm 7: SGEMM Transpose

```
ALLOCATE(Atr(k,m))
Atr=TRANSPOSE(A)
! Uses -mkl=parallel compiler option
CALL SGEMM('T','N',m,n,k,alpha,Atr,k,B,k,beta,C,m)
DEALLOCATE(Atr)
```

Acknowledgments

The research reported in this paper is partially supported by the HPC@ISU equipment at Iowa State University, some of which has been purchased through funding provided by NSF under MRI grant number CNS 1229081 and CRI grant number 1205413.

References

[1] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Trans. Math. Softw.*, vol. 5, no. 3, pp. 308–323, Sept. 1979. [Online]. Available: http://doi.acm.org/10.1145/355841.355847

[2] J. Keuper and F.-J. Preundt, "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability," in *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, ser. MLHPC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 19–26. [Online]. Available: https://doi.org/10.1109/MLHPC.2016.6

[3] C. T. Ruud van der Pas, Eric Stotzer, *Using OpenMP - The Next Step*. The MIT Press, 2017.

[4] M. K. et al., *caret: Classification and Regression Training*, 2012, r package version 5.15-044. [Online]. Available: http://CRAN.R-project.org/package=caret
[5] ——, *C50: C5.0 Decision Trees and Rule-Based Models*, 2015, r package version 0.1.0-24. [Online]. Available: https://CRAN.R-project.org/package=C50
[6] J. R. Rice, "The algorithm selection problem," ser. Advances in Computers, M. Rubinoff and M. C. Yovits, Eds. Elsevier, 1976, vol. 15, pp. 65 – 118. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0065245808605203
[7] M. T. L. et al., "Autofolio: An automatically configured algorithm selector," *J. Artif. Intell. Res.*, vol. 53, pp. 745–778, 2015.
[8] van de Geijn et al., "Summa: Scalable universal matrix multiplication algorithm," Austin, TX, USA, Tech. Rep., 1995.
[9] D. et al., "Communication-optimal parallel recursive rectangular matrix multiplication," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 261–272. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.80
[10] S. et al., "Matrix multiplication algorithm selection with support vector machines," Master's thesis, EECS Department, University of California, Berkeley, May 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-29.html
[11] S. S. et al., "Improving the performance of fully connected neural networks by out-of-place matrix transpose," *CoRR*, vol. abs/1702.03192, 2017. [Online]. Available: http://arxiv.org/abs/1702.03192