Performance Comparison of Accurate Matrix Multiplication

Katsuhisa Ozaki (Shibaura Institute of Technology)
and
Takeshi Ogita (Tokyo Christian Woman's University)

Sep. 25th, 2012

SCAN 2012, Novosibirsk, Russia

Introduction

This talk is concerned with accurate matrix multiplication for floating-point matrices.

Floating-point numbers as defined by IEEE 754 has finite information,

- 24 siginificand bits for binary32
- 53 siginificand bits for binary64

Therefore, rounding error may occur in each arithmetic operation.

Notation

- $\mathbb{F}$: the set of floating-point numbers.
- $A \in \mathbb{F}^{m \times n}, B \in \mathbb{F}^{n \times p}$, we compute the matrix multiplication AB.
- $\mathrm{fl}(\cdots)$ means that an expression is evaluated by fl-pt arithmetic.
- $\mathbf{u}$: unit roundoff (binary64: $\mathbf{u} = 2^{-53}$)

For $\mathrm{fl}(\cdots)$, we assume that neither overflow nor underflow occur.

Introduction

Matrix multiplication consists of dot products:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

For example,

$$a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + \cdots + a_{1n}b_{n1}.$$

Maximally, rounding errors occur $2n - 1$ times.

Introduction

In the worst case, the computed result is inaccurate due to accumulation of rounding errors. From an a priori error analysis, we have the following error bound

$$|\mathrm{fl}(AB) - AB| \le \frac{n\mathbf{u}}{1 - n\mathbf{u}}|A||B|,$$

namely

$$\frac{|\mathrm{fl}(AB) - AB|_{ij}}{|AB|_{ij}} \le \frac{n\mathbf{u}}{1 - n\mathbf{u}}\frac{(|A||B|)_{ij}}{|AB|_{ij}}.$$

Introduction

We develop a new and accurate algorithm for matrix multiplication.

An error bound for a computed result by our algorithm satisfies

$$|AB - \tilde{C}| \le \mathbf{u}|AB|.$$

Overview of our algorithm is

Error-Free Transformation of Matrix Multiplication

+

Accurate Summation Algorithm

Table of Contents

- Naive Approach
- Error-free Transformation of Matrix Multiplication
- Memory reduced Implementation
- Comparison of Computational Performance

Naive Approach

We apply Veltkamp-Dekker's error-free transformation of a product of floating-point number. For $a, b, x, y \in \mathbb{F}$, their algorithm transforms

$$a * b = x + y, \quad x = \mathrm{fl}(a * b), \quad \mathbf{u}|x| \geq |y|.$$

It requires 17 floating-point operations.

Naive Approach

Applying error-free transformation by Veltkamp and Dekker,

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{2n} v_k.$$

S.M.Rump, T. Ogita, S. Oishi:
Accurate floating-point summation part II: Sign, K-fold faithful and rounding to nearest. Siam J. Sci. Comput., 31(2):1269-1302, 2008.

Then

$$|AB - \tilde{C}| \leq \mathbf{u}|AB|.$$

Accurate Matrix Multiplication

We introduce the error-free transformation of the matrix product. Both A and B are divided into an unevaluated sum of k and l floating-point matrices, respectively, i.e.

$$A = A^{(1)} + A^{(2)} + \cdots + A^{(k)}, \quad B = B^{(1)} + B^{(2)} + \cdots + B^{(l)}$$

and for all k and l

$$A^{(k)} \in \mathbb{F}^{m\times n}, \quad B^{(l)} \in \mathbb{F}^{n\times p}, \quad \mathrm{fl}(A^{(k)}B^{(l)}) = A^{(k)}B^{(l)}.$$

$$q = \texttt{size}(A, 2);$$
$$k = 1;$$
$$\beta = \mathrm{fl}(\lceil(-\log_2(\mathbf{u}) + \log 2(q))/2)\rceil);$$
$$A^{(i)} = \texttt{zeros}(\texttt{size}(A));$$
$$\texttt{while}\ (\texttt{norm}(A, inf)\tilde{}= 0)$$
$$\quad \mu = \max(\texttt{abs}(A), [], 2); \quad \%\ \mu(i) = \max_{1\le j\le q} a_{ij}$$
$$\quad \texttt{if}\ (\max(\mu) == 0),\ \texttt{return};\ \texttt{end}$$
$$\quad w = \mathrm{fl}(2.\hat{}(\texttt{ceil}(\log 2(\mu)) + \beta));$$
$$\quad S = \texttt{repmat}(w, 1, q); \quad \%\ w \cdot e^T$$
$$\quad A^{(k)} = \mathrm{fl}((A + S) - S);$$
$$\quad A = \mathrm{fl}(A - A^{(k)});$$
$$\quad k = k + 1;$$
$$\texttt{end}$$

Accurate Matrix Multiplication

Expanding the expression,

$$AB = (A^{(1)} + A^{(2)} + \cdots + A^{(k)})(B^{(1)} + B^{(2)} + \cdots + B^{(l)}),$$

AB is transformed into

$$AB = \sum_{i=1}^{kl} C^{(i)}, \quad C \in \mathbb{F}^{m \times p}.$$

By using Rump-Ogita-Oishi's NearSum algorithm,

$$|AB - \tilde{C}| \leq \mathbf{u}|AB|.$$

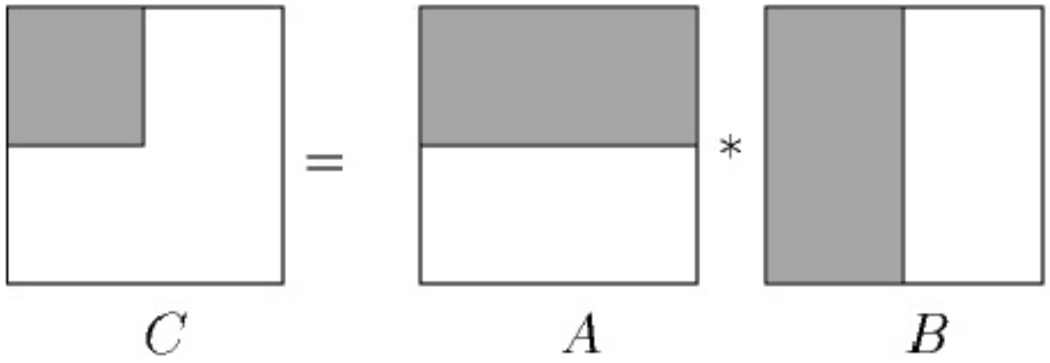
Advantage and Disadvantage

Advantage: Dependence of High Performance Library

Disadvantage: Memory Consumption.

K. Ozaki, T. Ogita, S. Oishi, S. M. Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, Vol. 59:1 (2012), pp. 95-118.

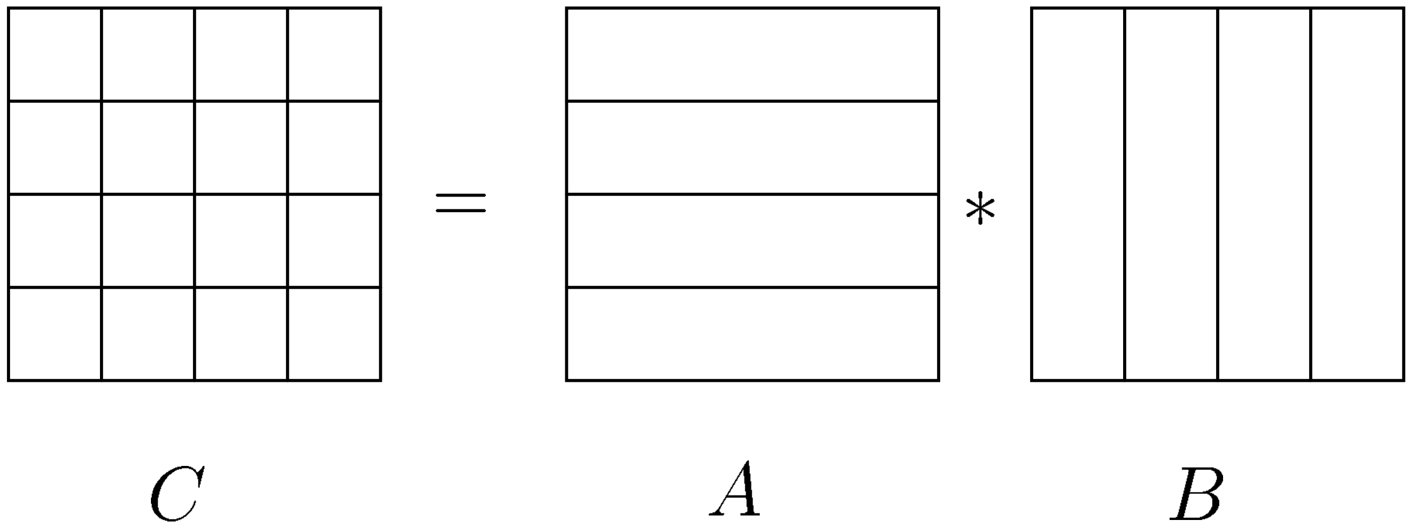
Memory Reduced Implementation



Assume that $A, B \in \mathbb{F}^{n\times n}$ (n is even), and we use MATLAB notation.

$$C(1:n/2, 1:n/2) = A(1:n/2, :) * B(:, 1:n/2)$$

Memory Reduced Implementation



We call this method Type 1.

Memory Reduced Implementation

k : the number of blocks

`accmul` : usual accurate matrix multiplication

```
d = n/k;
for i = 1 : k
  for j = 1 : k
    C((i − 1)d + 1 : i ∗ d, (j − 1)d + 1 : j ∗ d) =
        accmul(A((i − 1)d + 1 : i ∗ d, :)∗
         B(:, (j − 1)d + 1 : j ∗ d));
  end
end
```

Memory Reduced Implementation

Table 1: Comparison of FLOPS (Core i7-2620M, 2.66GHz, 2 cores).

A	B	FLOPS
$\mathbb{F}^{1200\times1200}$	$\mathbb{F}^{1200\times1200}$	36.83
$\mathbb{F}^{600\times1200}$	$\mathbb{F}^{1200\times600}$	32.85
$\mathbb{F}^{300\times1200}$	$\mathbb{F}^{1200\times300}$	30.10
$\mathbb{F}^{2400\times2400}$	$\mathbb{F}^{2400\times2400}$	40.24
$\mathbb{F}^{1200\times2400}$	$\mathbb{F}^{2400\times1200}$	37.98
$\mathbb{F}^{600\times2400}$	$\mathbb{F}^{2400\times600}$	33.17

Memory Reduced Implementation

Table 2: Comparison of FLOPS (Core i7-2620M, 2.66GHz, 2 cores).

A	B	FLOPS
$\mathbb{F}^{4800\times4800}$	$\mathbb{F}^{4800\times4800}$	33.36
$\mathbb{F}^{2400\times4800}$	$\mathbb{F}^{4800\times2400}$	36.72
$\mathbb{F}^{1200\times4800}$	$\mathbb{F}^{4800\times1200}$	36.20
$\mathbb{F}^{9600\times9600}$	$\mathbb{F}^{9600\times9600}$	39.72
$\mathbb{F}^{4800\times9600}$	$\mathbb{F}^{9600\times4800}$	42.02
$\mathbb{F}^{2400\times9600}$	$\mathbb{F}^{9600\times2400}$	41.86

Memory Reduced Implementation

Table 3: Comparison of FLOPS (Xeon X5550, 2.67GHz, 2 CPU, 8 cores).

A	B	FLOPS
$\mathbb{F}^{1200\times1200}$	$\mathbb{F}^{1200\times1200}$	62.2
$\mathbb{F}^{600\times1200}$	$\mathbb{F}^{1200\times600}$	48.2
$\mathbb{F}^{300\times1200}$	$\mathbb{F}^{1200\times300}$	32.3
$\mathbb{F}^{2400\times2400}$	$\mathbb{F}^{2400\times2400}$	75.1
$\mathbb{F}^{1200\times2400}$	$\mathbb{F}^{2400\times1200}$	70.7
$\mathbb{F}^{600\times2400}$	$\mathbb{F}^{2400\times600}$	66.5

Memory Reduced Implementation

Table 4: Comparison of FLOPS (Xeon X5550, 2.67GHz, 2 CPU, 8 cores).

A	B	FLOPS
$\mathbb{F}^{4800\times 4800}$	$\mathbb{F}^{4800\times 4800}$	77.4
$\mathbb{F}^{2400\times 4800}$	$\mathbb{F}^{4800\times 2400}$	77.4
$\mathbb{F}^{1200\times 4800}$	$\mathbb{F}^{4800\times 1200}$	74.1
$\mathbb{F}^{9600\times 9600}$	$\mathbb{F}^{9600\times 9600}$	77.4
$\mathbb{F}^{4800\times 9600}$	$\mathbb{F}^{9600\times 4800}$	75.1
$\mathbb{F}^{2400\times 9600}$	$\mathbb{F}^{9600\times 2400}$	77.7

Memory Reduced Implementation

Next, we consider an another way (Type 2).

$$
\begin{array}{rrl}
A^{(1)} + \underline{A}^{(2)}, & B^{(1)} + \underline{B}^{(2)} \Longrightarrow & A^{(1)}B^{(1)} \\
A^{(1)} + \underline{A}^{(2)}, & \cancel{B^{(1)}} + B^{(2)} + \underline{B}^{(3)} \Longrightarrow & A^{(1)}B^{(2)} \\
A^{(1)} + \underline{A}^{(2)}, & \cancel{B^{(2)}} + B^{(3)} + \underline{B}^{(4)} \Longrightarrow & A^{(1)}B^{(3)} \\
 & \vdots & \\
\cancel{A^{(1)}} + A^{(2)} + \underline{A}^{(3)}, & B^{(1)} + \underline{B}^{(2)} \Longrightarrow & A^{(2)}B^{(1)} \\
\cancel{A^{(1)}} + A^{(2)} + \underline{A}^{(3)}, & \cancel{B^{(1)}} + B^{(2)} + \underline{B}^{(3)} \Longrightarrow & A^{(2)}B^{(2)}
\end{array}
$$

Let μ be space for n-by-n matrix. Pure implementation requires

$$(n_A + n_B + n_A n_B)\mu.$$

Type 1 with k blocks requires

$$(n_A + n_B)\mu/k + n_A n_B \mu/k^2$$

Type 2 requires

$$4\mu + n_A n_B \mu$$

Combination fo Type 1 and Type 2 requires

$$4\mu/k + n_A n_B \mu/k^2.$$

Memory Reduced Implementation

Let $A(1 : n/2, :)$ be A_1.

If A is divided into

$$A = A^{(1)} + A^{(2)} + A^{(3)} + A^{(4)}.$$

The following may happen:

$$A_1 = A_1^{(1)} + A_1^{(2)} + A_1^{(3)}.$$

The number of matrix products may be reduced by block computations.

Numerical Results

We compare computing times for

- M1: Naive Approach for rounding to nearest.
- M2 ($k = 1$): EFT + rounding to nearest
- M2 ($k > 1$): EFT + rounding to nearest with block computations

Computational environments:
Core i7-2620M, MATLAB2011b, Intel C++ Compiler 12.0.

Numerical Results

Table 5: Comparison of computing times and ratio.

Method \ n	1200	2400	4800
M1	15.8 (131.3)	136.1 (194.4)	1362 (203.6)
M2 (k=1)	1.58 (13.1)	17.0 (24.3)	132.0 (19.7)
M2 (k=2)	1.76 (14.6)	17.8 (25.5)	132.9 (19.8)
M2 (k=3)	1.76 (14.6)	18.4 (26.4)	135.0 (20.1)
M2 (k=4)	1.74 (14.3)	19.2 (27.4)	139.7 (20.8)
M2 (k=5)	1.80 (14.9)	19.8 (28.3)	142.0 (21.2)

A and B are generated as `randn`(n).

Numerical Results

Table 6: Comparison of ratio with various ϕ ($n = 1200$).

Method \ ϕ	0	1	4	7	10
M1	149.6	146.7	134.7	143.3	81.1
M2 (k=1)	16.1	17.2	29.7	46.0	42.1
M2 (k=2)	17.5	19.7	31.3	49.8	43.8
M2 (k=4)	17.7	18.7	30.9	57.5	47.4

A and B are generated as $(\mathrm{rand}(n) - 0.5).*\exp(\phi * \mathrm{randn}(n))$.
If ϕ is large, there is big difference in the order of magnitude.

Numerical Results

Table 7: Comparison of ratio with various ϕ ($n = 2400$).

Method \ ϕ	0	1	4	7	10
M1	171.1	170.3	174.4	171.3	169.0
M2 (k=1)	16.2	19.3	34.3	54.9	84.2
M2 (k=2)	16.9	18.5	35.5	55.4	85.1
M2 (k=4)	17.6	19.6	39.6	61.0	92.3

A and B are generated as $(\mathrm{rand}(n) - 0.5). * \exp(\phi * \mathrm{randn}(n))$.
If ϕ is large, there is big difference in the order of magnitude.

Numerical Results

Table 8: Comparison of ratio with various ϕ ($n = 4800$).

Method \ ϕ	0	1	4	7	10
M1	204.5	170.3	204.7	203.8	205.9
M2 (k=1)	15.3	18.7	32.6	70.1	169.3
M2 (k=2)	15.6	19.2	33.3	59.4	89.4
M2 (k=4)	16.2	19.9	34.2	61.6	92.7

A and B are generated as $(\mathrm{rand}(n) - 0.5). * \exp(\phi * \mathrm{randn}(n))$.
If ϕ is large, there is big difference in the order of magnitude.

Conclusion

- EFT of matrix multiplication efficiently helps accurate computing in terms of computational performance
- Block computations reduce the amount of working memory.
- Block computations don't significantly slow computational performance down (sometimes work faster than original one).

Thank you very much for your kind attention!