ゲーミングGPUの単精度演算器を用いた倍精度行列積の近似計算

大阪府立大学　大学院工学研究科　知能情報工学分野
七井香樹　藤本典幸

研究の背景

- 価格性能比の良さからGPU計算がここ数年様々な研究分野で活用されている
- しかし、倍精度演算性能に関してはデータセンター用の高価なGPUだけが高性能
- ゲームミング用の安価なGPUは倍精度演算性能を大きく制限されている

研究の動機

- ゲーミングGPUで高速計算可能な
  単精度演算器を用いて、倍精度行列積の
  近似解を高速に計算したい

研究の目的

- 尾崎らによって高精度な行列積計算法が
  提案されている
- これは、倍精度行列積計算を
  複数回用いることで倍精度行列積の
  演算結果を高精度化している
- 尾崎らの手法を応用して
  単精度行列積計算を用いて倍精度行列積の
  近似計算をCUDA実装で実現する

基礎知識

- GeForce RTX 2080 SUPERのGPUでは、倍精度の計算は単精度の計算より32倍の計算時間がかかる
- cuBLASは、ベンダーがBLASを元にGPUに最適化した行列計算ライブラリであり、NVIDIA製GPUで高速に計算可能
- 単精度の仮数部は24bit
  倍精度の仮数部は53bit

参考: CUDA Toolkit Documentation.
https://docs.nvidia.com/cuda/cublas/index.html

尾崎らの高精度な行列積計算法

- 行列積C=ABの行列A, Bを仮数部の上位bitの行列$A^{(1)}$, $B^{(1)}$と下位bitの行列$A^{(2)}$, $B^{(2)}$に分割

$$A=A^{(1)}+A^{(2)}$$
$$B=B^{(1)}+B^{(2)}$$

$$\begin{pmatrix} 1.234567 & 2.357111 \\ 3.141592 & 1.357924 \end{pmatrix} = \begin{pmatrix} 1.2 & 2.3 \\ 3.1 & 1.3 \end{pmatrix} + \begin{pmatrix} 0.034567 & 0.057111 \\ 0.041592 & 0.057924 \end{pmatrix}$$

$$C=AB=A^{(1)}B^{(1)}+A^{(1)}B^{(2)}+A^{(2)}B$$

尾崎らの高精度な行列積計算法

$$A=A^{(1)}+A^{(2)}=A^{(1)}+A^{(2)'}+A^{(3)}$$
$$B=B^{(1)}+B^{(2)}=B^{(1)}+B^{(2)'}+B^{(3)}$$

$$\begin{pmatrix}1.234567 & 2.357111\\ 3.141592 & 1.357924\end{pmatrix}=\begin{pmatrix}1.2 & 2.3\\ 3.1 & 1.3\end{pmatrix}+\begin{pmatrix}0.034567 & 0.057111\\ 0.041592 & 0.057924\end{pmatrix}$$
$$=\begin{pmatrix}1.2 & 2.3\\ 3.1 & 1.3\end{pmatrix}+\begin{pmatrix}0.034 & 0.057\\ 0.041 & 0.057\end{pmatrix}+\begin{pmatrix}0.000567 & 0.000111\\ 0.000592 & 0.000924\end{pmatrix}$$

$$C=AB=A^{(1)}B^{(1)}+A^{(1)}B^{(2)'}+A^{(2)'}B^{(1)}$$
$$+A^{(1)}B^{(3)}+A^{(2)'}B^{(2)}+A^{(1)}B^{(2)}$$

尾崎らの行列分割アルゴリズム

$n = \mathrm{size}(A, 2);$ %行列Aの行の大きさ

$\mu_A = \max(\mathrm{abs}(\mathrm{A}), [], 2);$ %各行の絶対値の最大値
$\mu_B = \max(\mathrm{abs}(\mathrm{B}), [], 1);$ %各列の絶対値の最大値

$\alpha = \left\lceil \frac{-\log_2 u + \log_2 n}{2} \right\rceil;$

$t_A = 2.\hat{}(\lceil \log_2 \mu_A + \alpha \rceil);$
$t_B = 2.\hat{}(\lceil \log_2 \mu_B + \alpha \rceil);$

$S_A = \mathrm{repmat}(t_A, 1, n);$
$S_B = \mathrm{repmat}(t_B, n, 1);$

$A^{(1)} = \mathrm{fl}\left((A + S_A) - S_A\right);$ %浮動小数点演算で行列Aの上位bit
$A^{(2)} = \mathrm{fl}\left(A - A^{(1)}\right);$ %浮動小数点演算で行列Aの下位bit

$B^{(1)} = \mathrm{fl}\left((B + S_B) - S_B\right);$ %浮動小数点演算で行列Bの上位bit
$B^{(2)} = \mathrm{fl}\left(B - B^{(1)}\right);$ %浮動小数点演算で行列Bの上位bit

出典
大石ら, 精度保証付き数値計算の基礎, コロナ社, 2018

尾崎らの高精度な行列積計算法

$$a_{i,k} = a_{i,k}^{(1)} + a_{i,k}^{(2)}$$

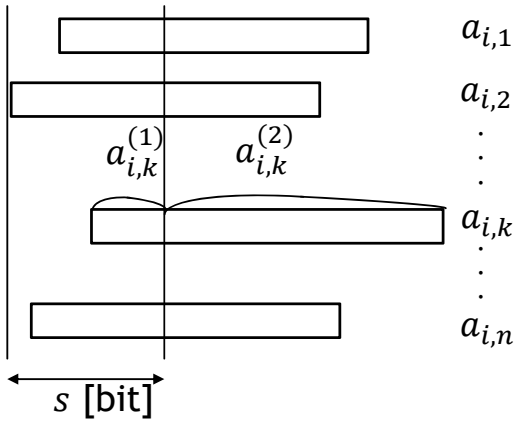

図1 行列Aの要素の分割

$$b_{k,j} = b_{k,j}^{(1)} + b_{k,j}^{(2)}$$

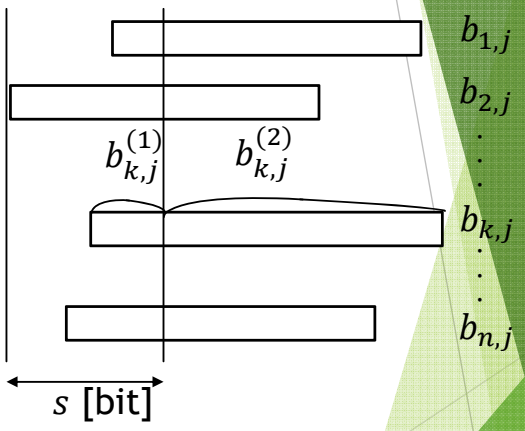

図2 行列Bの要素の分割

尾崎らの高精度な行列積計算法

$$C^{(1)} = A^{(1)}B^{(1)}$$

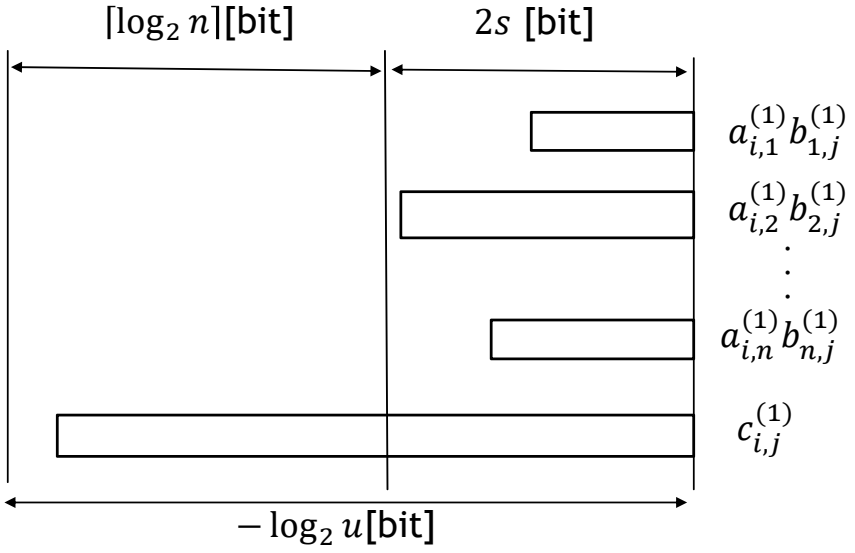

図3 分割した要素の足し合わせ

尾崎らの高精度な行列積計算法

$$\sigma_{a_i} = 2^{\left(\lceil \log_2 \mu_{a_i} \rceil + \alpha\right)}$$

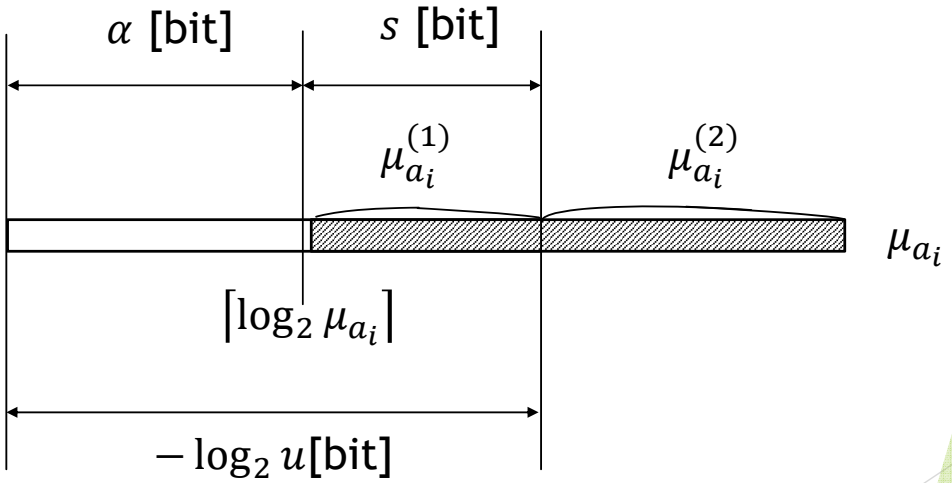

図4 σ_{a_i}とμ_{a_i}の関係（ σ_{a_i}はS_Aのi行目要素）

提案手法

- 倍精度行列積計算であるが
  $-\log_2 u$の値に24を用いる
- 安価なGPUで高速に計算可能な
  単精度で行列積計算を行うことができ、
  近似計算を高速に実現できる

提案する倍精度行列積の近似計算アルゴリズム（2分割の場合）

- $n = \mathrm{size}(A, 2);$
- $\mu_A = \max(\mathrm{abs}(\mathrm{A}), [], 2);$
  $\mu_B = \max(\mathrm{abs}(\mathrm{B}), [], 1);$
- $\alpha = \left\lceil \frac{24 + \log_2 n}{2} \right\rceil;$
- $t_A = 2.\hat{}(\lceil \log_2 \mu_A + \alpha \rceil);$
  $t_B = 2.\hat{}(\lceil \log_2 \mu_B + \alpha \rceil);$
- $S_A = \mathrm{repmat}(t_A, 1, n);$
  $S_B = \mathrm{repmat}(t_B, n, 1);$
- $A^{(1)} = \mathrm{fl}\big((A + S_A) - S_A\big);$
  $A^{(2)} = \mathrm{fl}\big(A - A^{(1)}\big);$
- $B^{(1)} = \mathrm{fl}\big((B + S_B) - S_B\big);$
  $B^{(2)} = \mathrm{fl}\big(B - B^{(1)}\big);$
- $C = A^{(1)}B^{(1)} + A^{(1)}B^{(2)} + A^{(2)}B$

GPUで並列計算

GPUで並列計算

提案手法

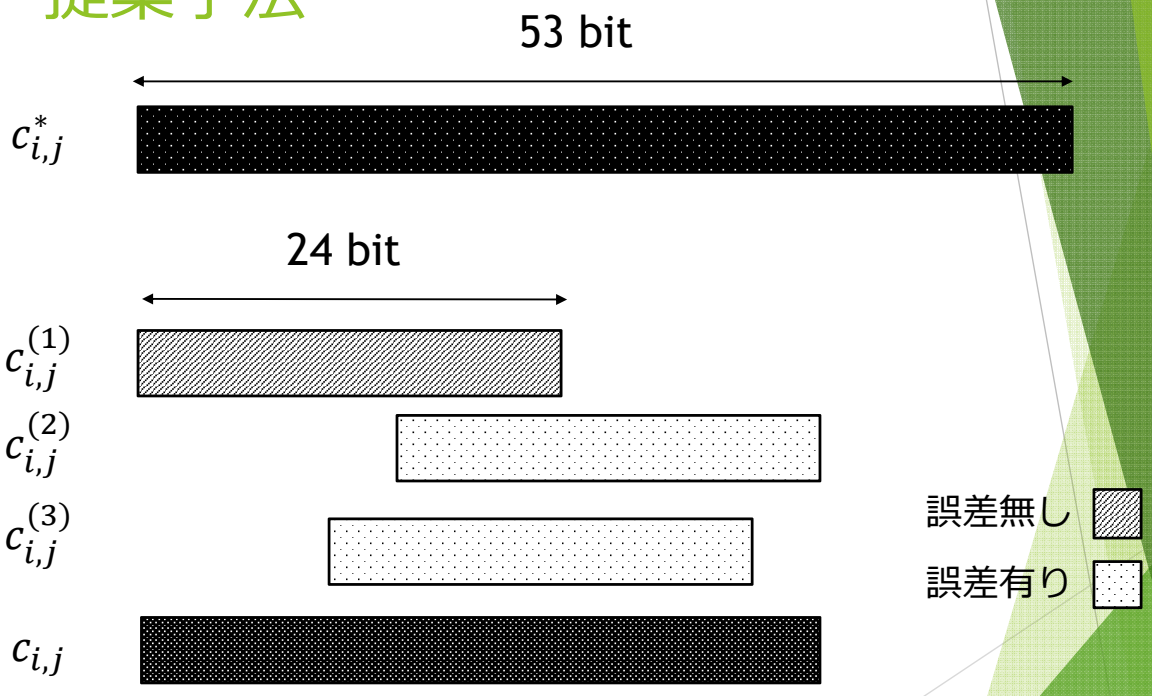

図5 提案手法による精度の改善

提案手法

- μ_A, μ_BはatomicMax関数を用いて計算する
- ただし、floatのatomicMaxはCUDAでは提供されていないため、以下の非負float版のatomicMaxを実装する

```
__device__ float atomicMax(float* addr, float val)
//'val' must be non-negative.
{
    unsigned old = atomicMax((unsigned*)addr,*((unsigned*)&val));

    return *((float*)&old);
}
```


実験条件

- CPU：Intel Core i3-8100
- GPU：NVIDIA GeForce RTX 2080 SUPER
- CUDA：Version 11.0 Update1
- コンパイラ：Visual Studio 2019
- OS：Windows 10 Pro

- 行列積の計算はcuBLASを利用
- 最大値計算はatomicMaxを利用
- 行列の要素は[-5，5]の一様乱数
- CPUからGPUへのコピー時間は考慮しない

実験内容

1. 単精度，2分割，3分割，倍精度の行列積計算の実行時間の比較
2. 倍精度と単精度，2分割，3分割の最大相対誤差と実行速度の関係
3. 提案手法の実行時間内訳

最大相対誤差： $\max_{1 \le i,j \le n} \frac{|c^*_{i,j} - c_{i,j}|}{|c^*_{i,j}|}$

($c^*_{i,j}$は倍精度行列積の結果の要素)

1．実行時間の比較

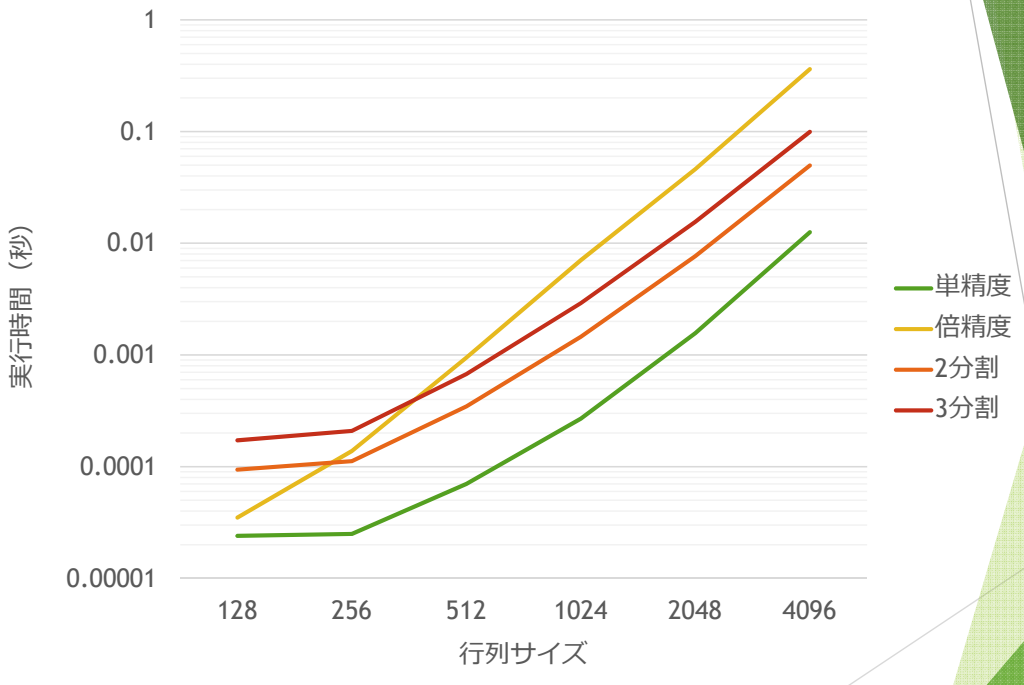

図6 手法別実行時間

2．実行速度と最大相対誤差

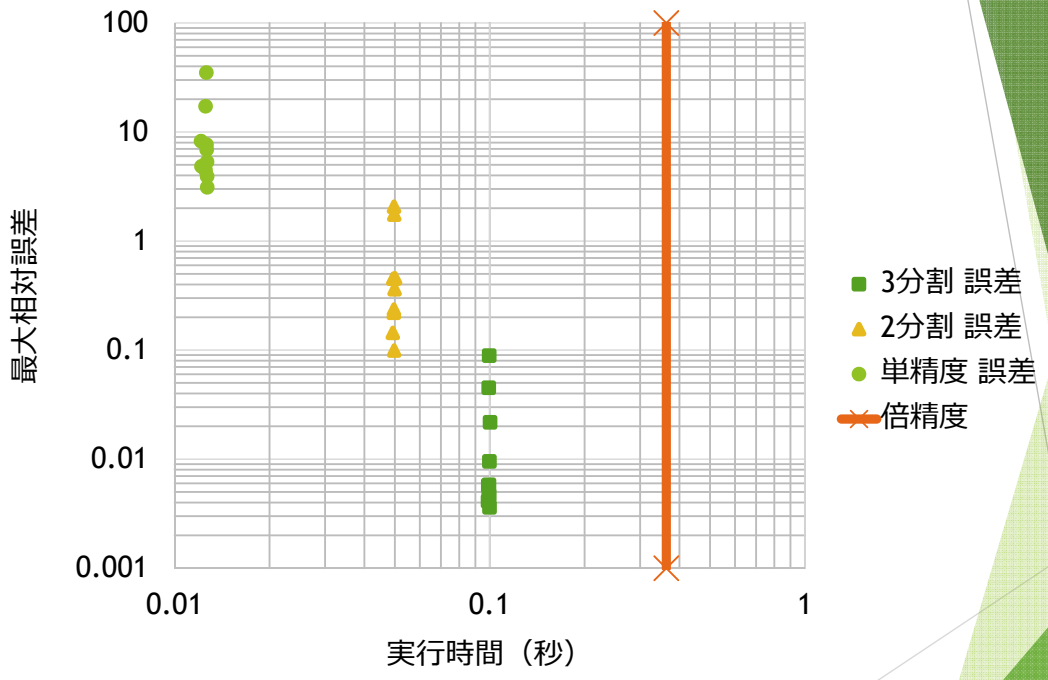

図7 行列サイズ4096×4096の最大相対誤差-速度比較

実行時間内訳

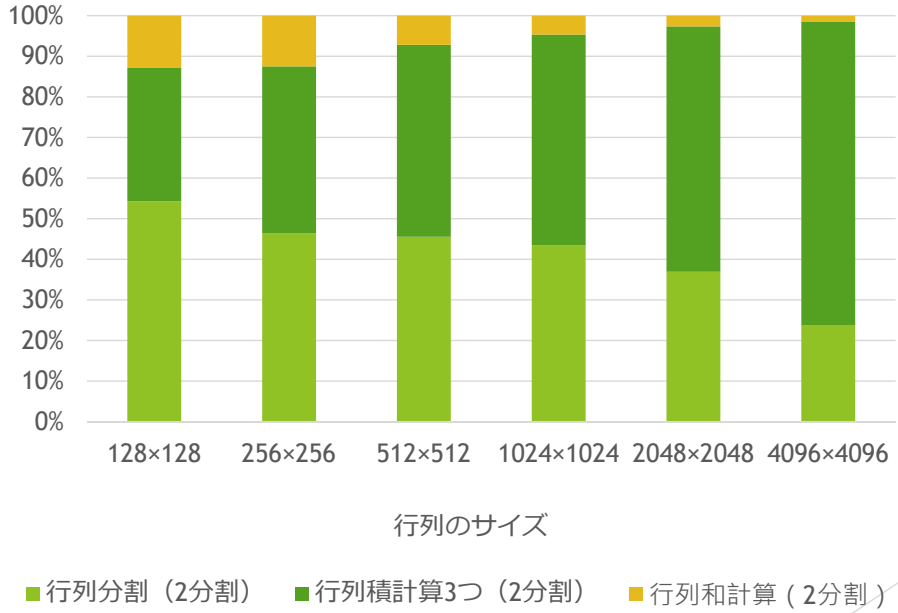

図8 2分割の処理別実行時間割合

実行時間内訳

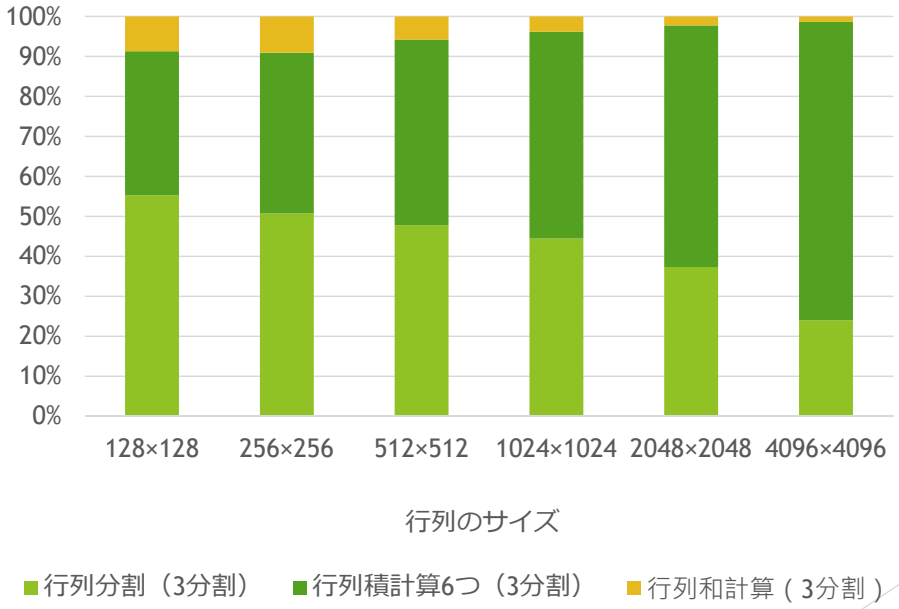

図9 3分割の処理別実行時間割合

まとめ

- 2分割すると単精度より約4倍～5倍、3分割では約8倍～11倍遅くなるが、32倍遅くなる倍精度で1度行列積を計算するよりも、高速に計算できた
- 近似解の精度は、単精度と比べ、2分割では約15倍～40倍、3分割では約500倍～2000倍改善

今後の課題

- 入力行列が値の大きな要素と
  小さな要素といった極端な行列で
  実装を考える必要がある
- 入力行列が疎行列の場合の実装を
  考える必要がある