連立一次方程式の反復解法ソルバーにおける並列処理と収束精度の問題について

片桐孝洋（東大）、黒田久泰（愛媛大）

共同研究者：
尾崎克久（芝浦工大）、荻田武史（東京女子大）、
大石進一（早大）

第２回CMSI人材育成シンポジウム
開催日　:　2013年12月2日(月)　13:50-14:20
開催場所:　大阪大学 基礎工学研究科G217

話の流れ

- **第１部：**
  連立一次方程式の反復解法ソルバーにおける並列化による誤差
  - Moving Particle Semi-implicit (MPS)法
  - 陰解法としての反復解法（共役勾配法、CG法）における並列化時の誤差
- **第２部：**
  高精度行列-行列積ライブラリの開発
  - 高精度行列-行列積アルゴリズム（尾崎の方法）
  - スレッド並列化手法
  - 疎行列化による高速化とスレッド並列化

第１部
連立一次方程式の
反復解法ソルバーにおける
並列化による誤差

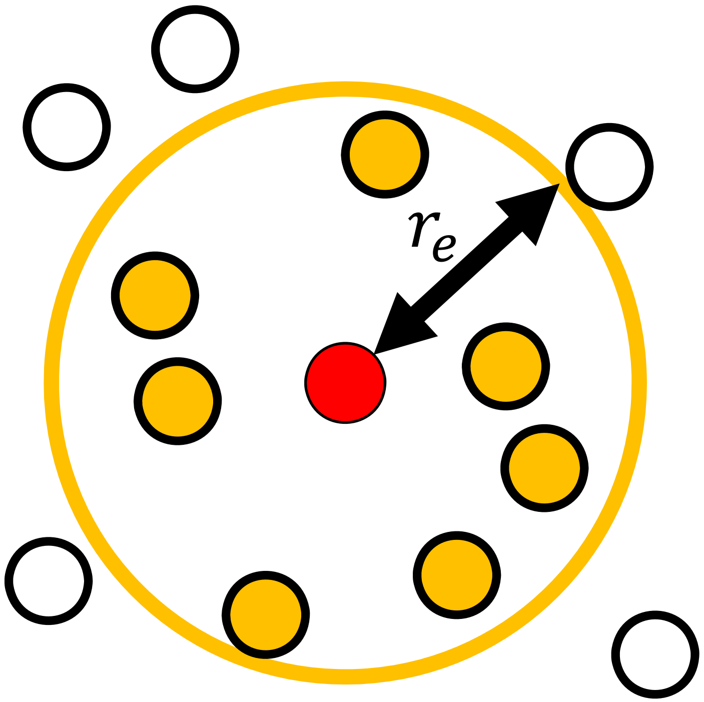
共同研究者:
黒田久泰（愛媛大）

MPS法による数値シミュレーション

- Moving Particle Semi-implicit 法の略
- 1995年に東京大学の越塚誠一教授が開発
- 連続体を多数の粒子で近似する粒子法の一種
- 非圧縮性流体を解析するのに有効
- 粒子の座標が変化し、時間ステップ毎に粒子間の関係式を作って解く
- 微分演算子に対応する粒子間相互作用モデルを用いて連続体の支配方程式を離散化する

粒子間相互作用モデル

$$\omega(r) = \begin{cases} \dfrac{r_e}{r} & 0 \le r \le r_e \\ 0 & r_e \le r \end{cases}$$

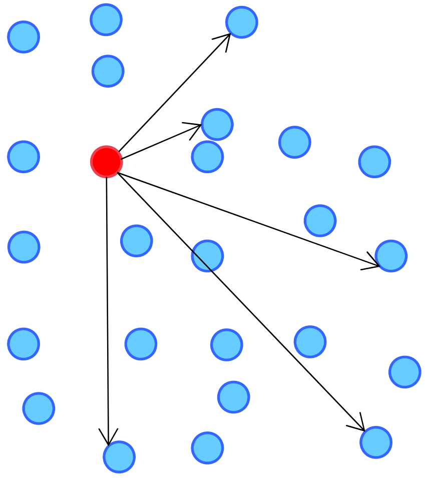


r :粒子間距離
r_e :粒子間相互作用半径

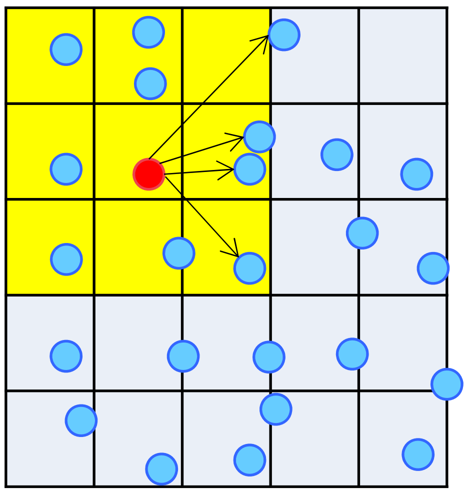
全ての粒子に対して相互半径以内であるかどうかを探索する必要がある(近傍粒子探索と呼ぶ)

近傍粒子探索における領域分割

- 全ての粒子に対して相互作用半径以内か探索
- 領域分割により注目粒子の周りのみを探索



$O(n^2)$



$O(n)$

非圧縮性流れの支配方程式

- 非圧縮性流れの支配方程式はナビエ･ストークス方程式と呼ばれる運動量保存則で表される

$$\frac{D\rho}{Dt} = 0$$

密度一定条件

ある地点での速度変化

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\boldsymbol{u} + \boldsymbol{g}$$

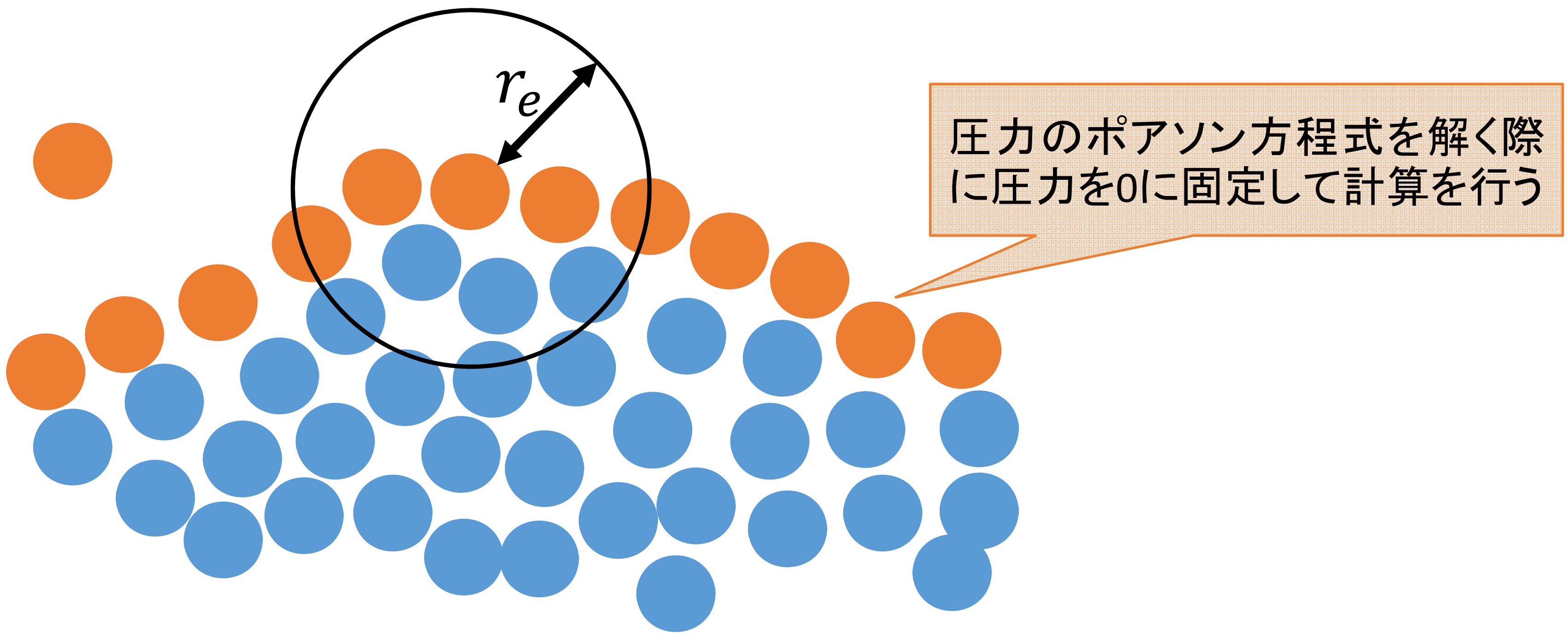
圧力項　粘性項　重力項

- MPS法では圧力項を陰的に、重力項と粘性項を陽的に計算
- 圧力項の計算におけるポアソンの圧力方程式を解くときに共役勾配法（CG法）を利用

境界条件

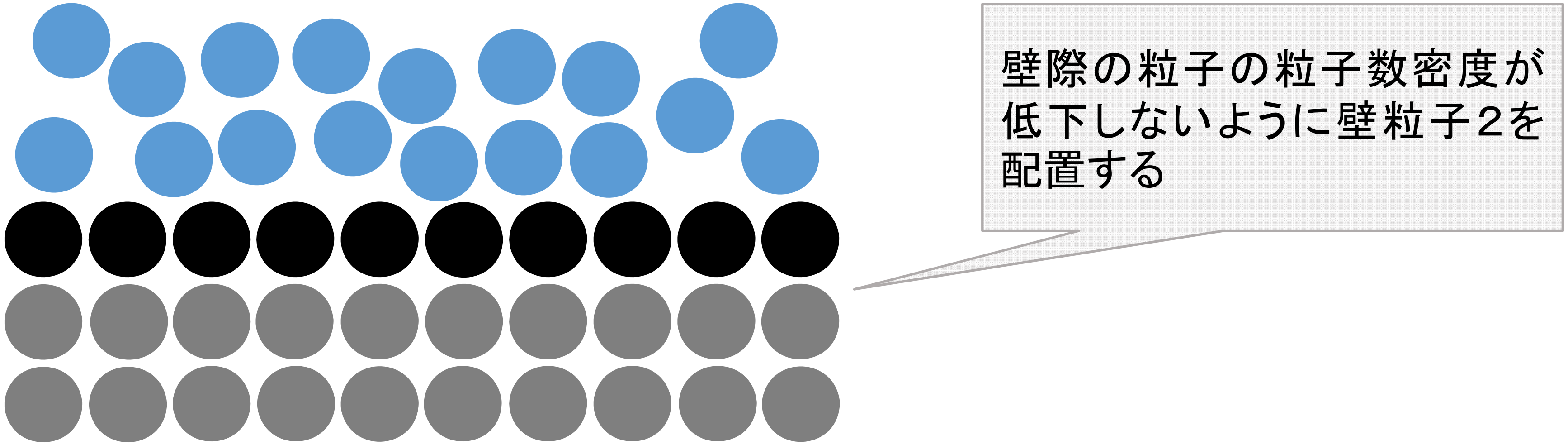
MPS法の自由表面判定には粒子数密度を用いる

- 自由表面にあると判定された粒子：
- 自由表面の内部とみなされた粒子：



粒子の種類

- 流体粒子●
- 壁粒子１（圧力計算あり）●
- 壁粒子２（圧力計算なし）●



共役勾配法

・連立一次方程式$A\boldsymbol{x} = \boldsymbol{b}$を解くための反復解法の一つ

$$r_0 = b - Ax_0;\;\; \beta_0 = \frac{1}{(r_0, r_0)};\;\; p_0 = \beta_0 r_0$$

for $k = 0,1,\cdots$ **until** $\|r_k\| \le \varepsilon\|b\|$ **do**

$$\alpha_k = \frac{1}{(p_k, Ap_k)}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

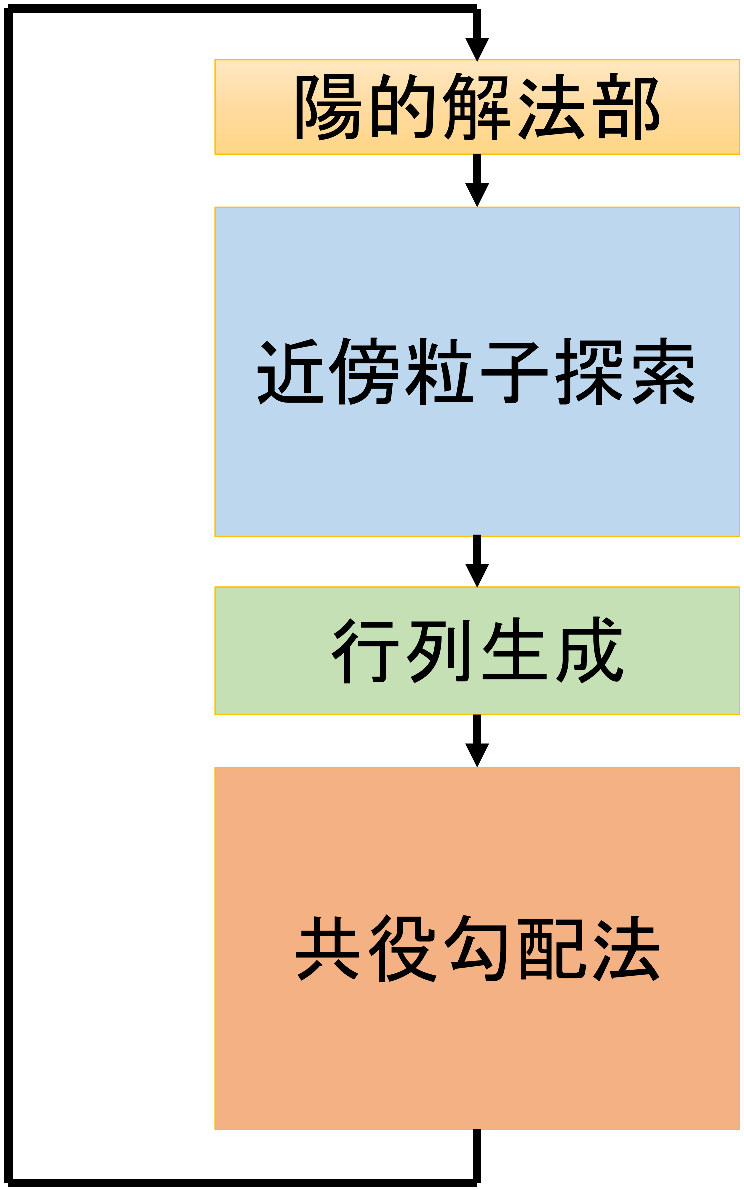
$$\beta_{k+1} = \frac{1}{(r_{k+1}, r_{k+1})}$$

$$p_{k+1} = p_k + \beta_{k+1} r_{k+1}$$

end

高橋秀俊版の共役勾配法

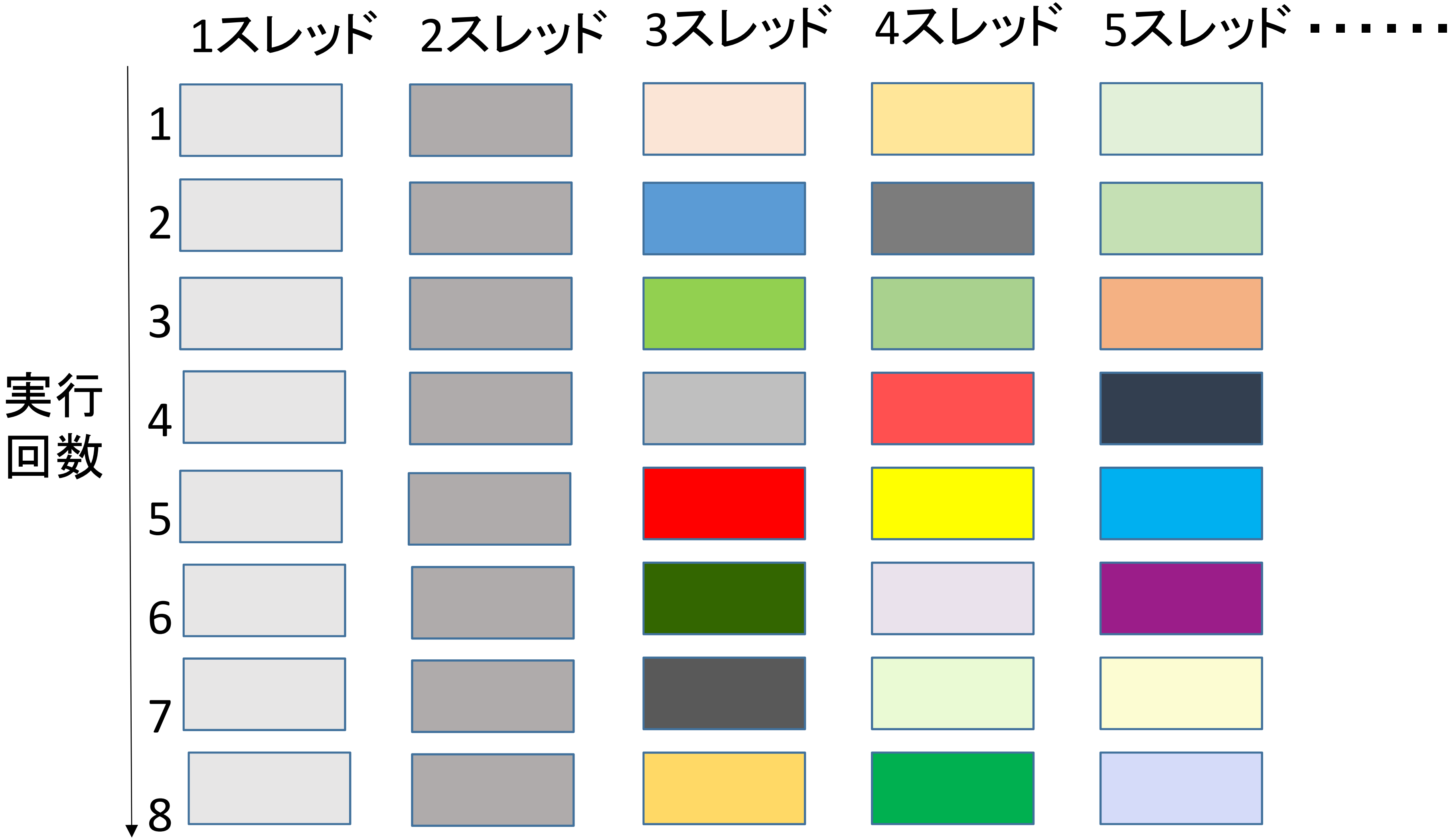
MPS法のシミュレーションの中身



MPSのシミュレーション（逐次実行）

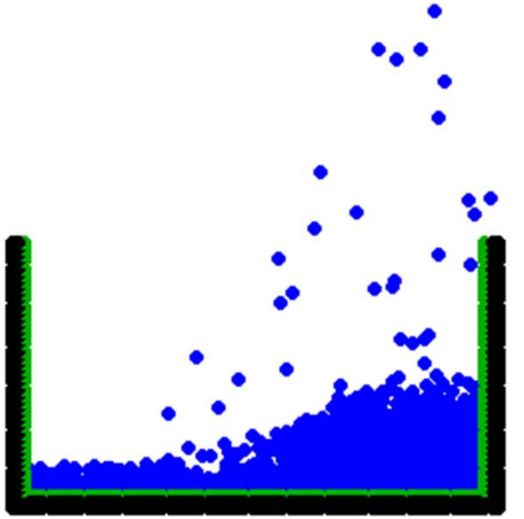
1スレッドで実行すると
毎回同じシミュレーション
結果が得られる

MPSのシミュレーション（並列実行）

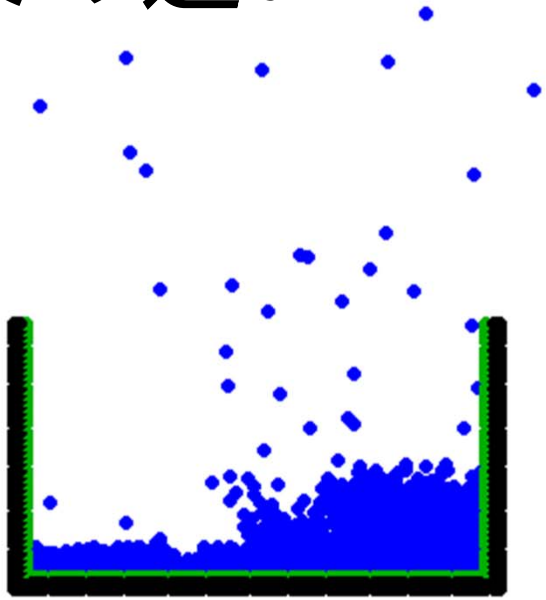


3スレッド以上になるとシミュレーションの度に結果が異なる

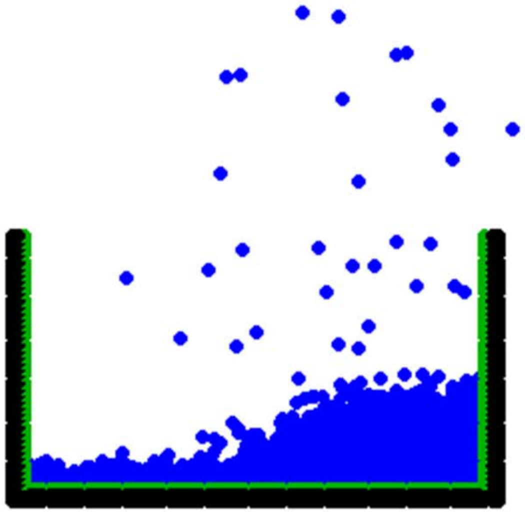
シミュレーション結果の違い

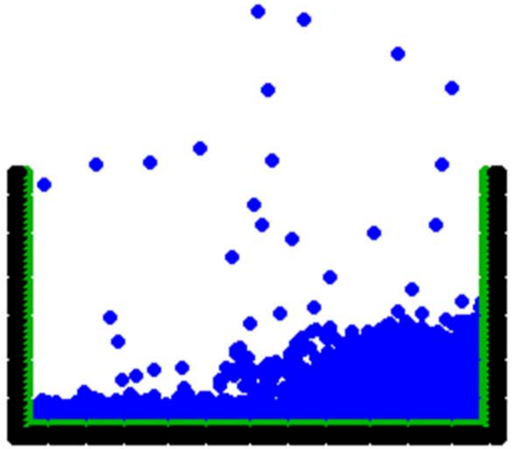


0.790019

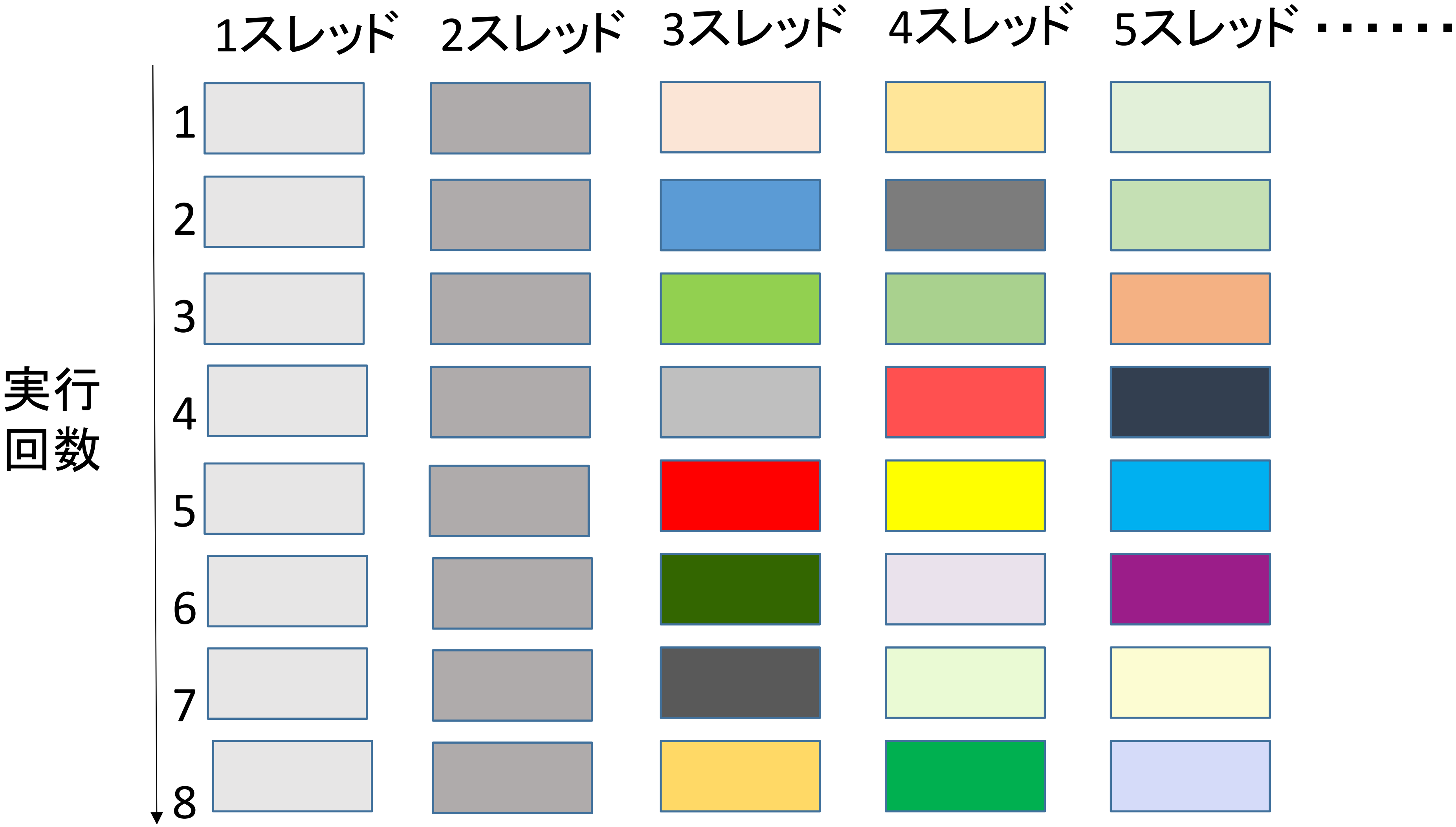


0.790201





共役勾配法（並列実行）のみの計算結果



3スレッド以上になると毎回結果が異なる

共役勾配法 改良①（同じスレッド数なら同じ結果）

$$r_0 = b - Ax_0;\ \ \beta_0 = \frac{1}{(r_0, r_0)};\ \ p_0 = \beta_0 r_0$$

for $k = 0,1,\cdots$ **until** $\|r_k\| \le \varepsilon \|b\|$ **do**

$$a_k = \frac{1}{(p_k, Ap_k)}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_{k+1} = \frac{1}{(r_{k+1}, r_{k+1})}$$

$$p_{k+1} = p_k + \beta_{k+1} r_{k+1}$$

end

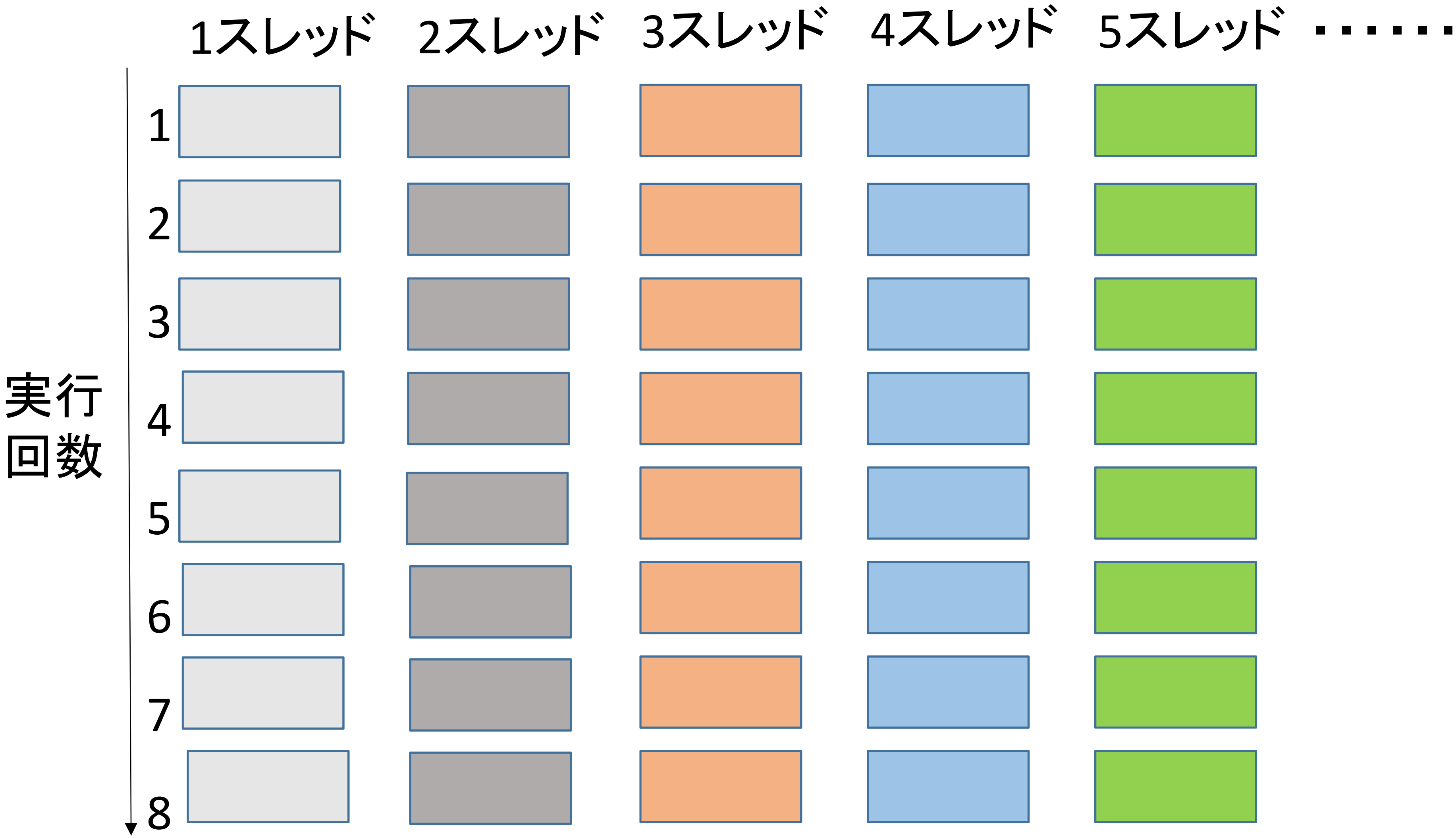
ベクトル同士の内積演算の部分で結果が異なる

各スレッドが計算した内積の値を合計するときに、足す順番を固定化する

sum=スレッド0の値+スレッド1の値+スレッド2の値+スレッド3の値

sum=(スレッド0の値+スレッド1の値)+(スレッド2の値+スレッド3の値)

共役勾配法（並列実行）の計算結果 改良①



同じスレッド数であれば同じシミュレーション結果が得られる

共役勾配法 改良②（スレッド数変えても同じ結果）

$r_0 = b - Ax_0;\ \ \beta_0 = \dfrac{1}{(r_0, r_0)};\ \ p_0$

$\quad = \beta_0 r_0$

for $k = 0,1,\cdots$ **until** $\|r_k\| \le \varepsilon \|b\|$ **do**

$\quad a_k = \dfrac{1}{(p_k, Ap_k)}$

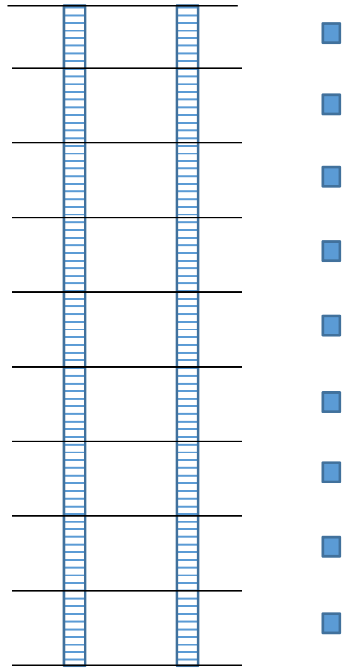
$\quad x_{k+1} = x_k + \alpha_k p_k$

$\quad r_{k+1} = r_k - \alpha_k A p_k$

$\quad \beta_{k+1} = \dfrac{1}{(r_{k+1}, r_{k+1})}$

$\quad p_{k+1} = p_k + \beta_{k+1} r_{k+1}$

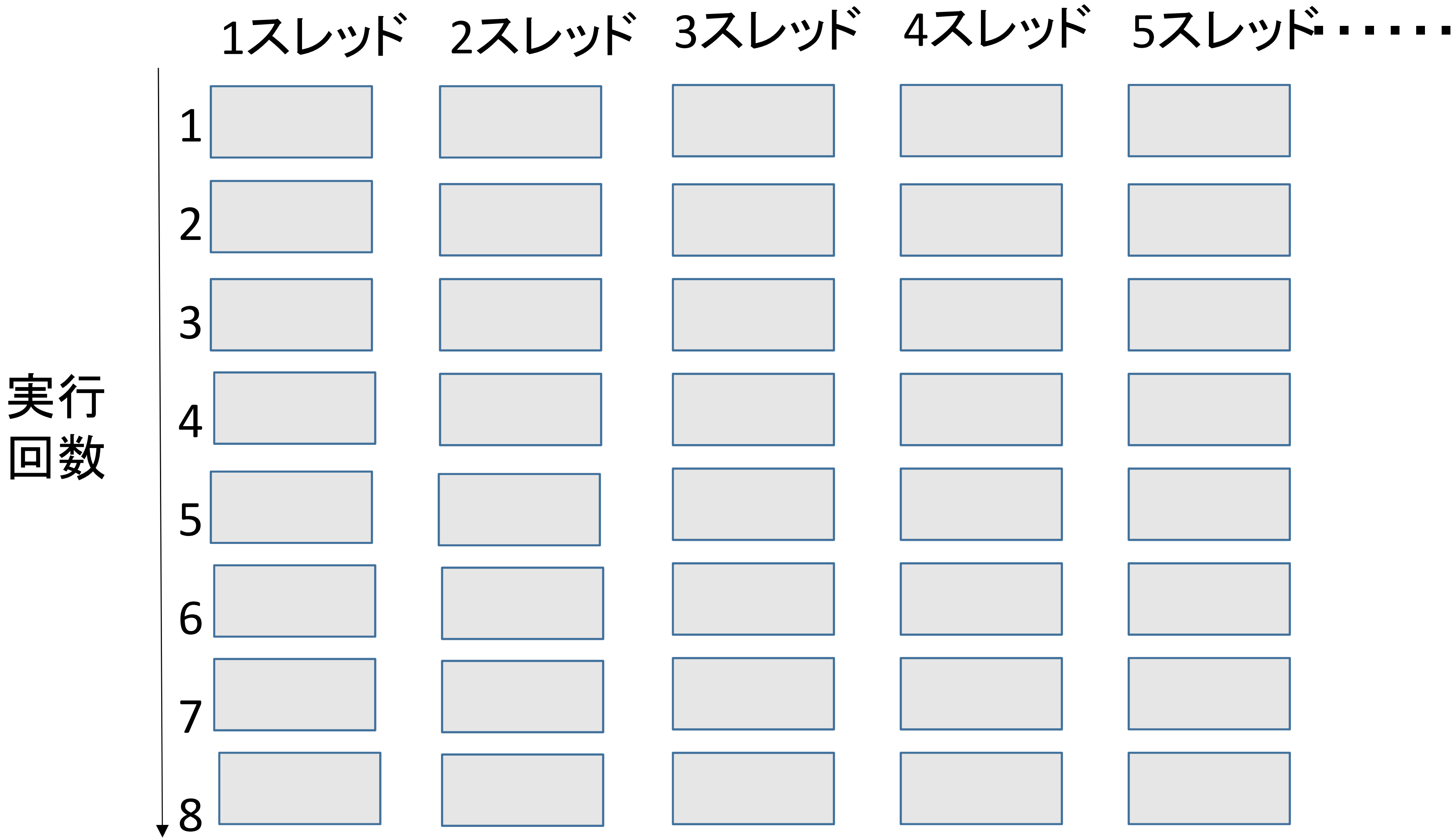
end



1,2,3,4,5,6,7,8のどのスレッドでも同じ結果を出力したい場合は、これらの最小公倍数の840個にベクトルを分割して計算すれば良い

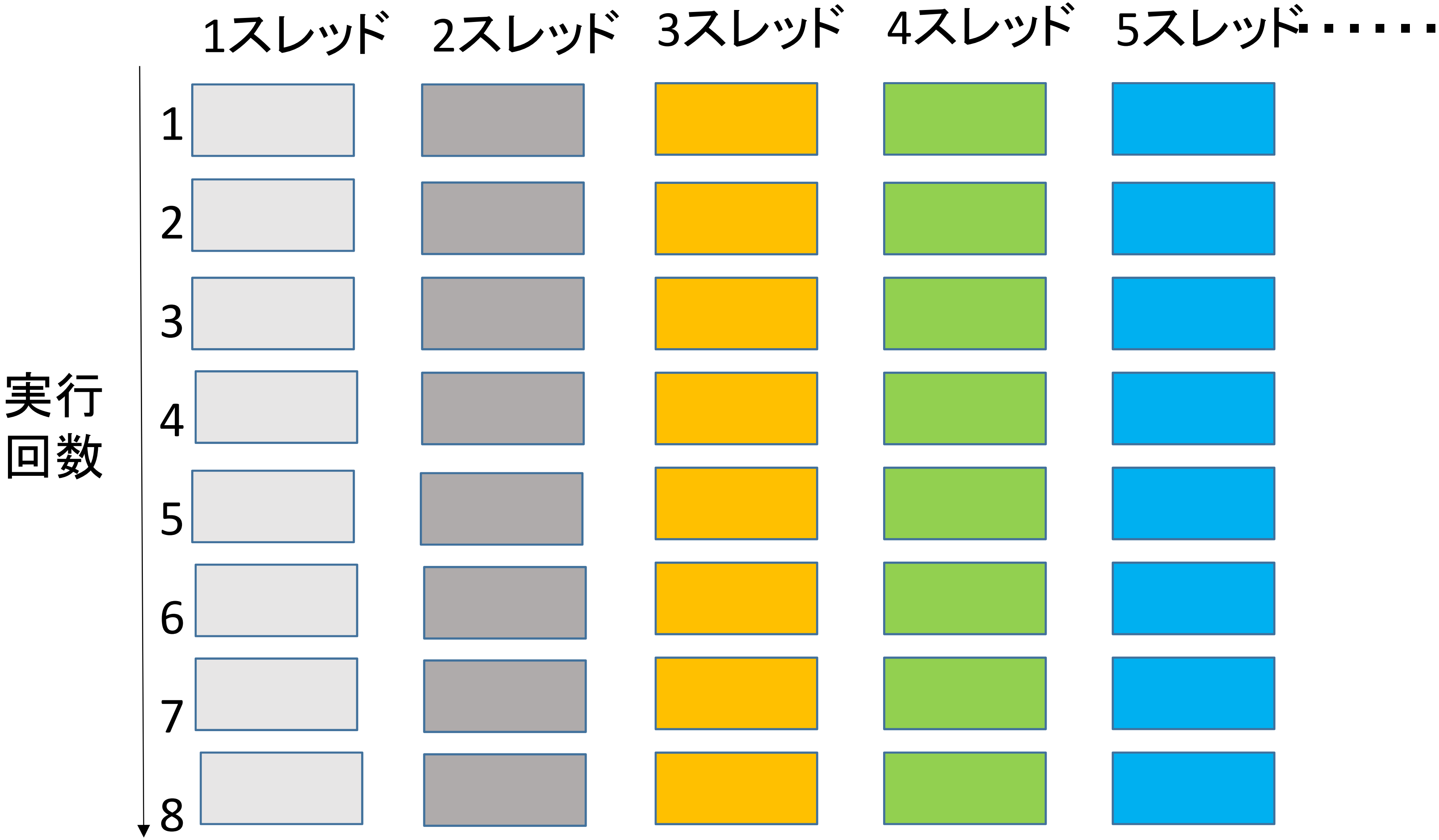
※ 1,2,3,4,6,8で良ければ24分割で済む

共役勾配法（並列実行）の計算結果 改良②



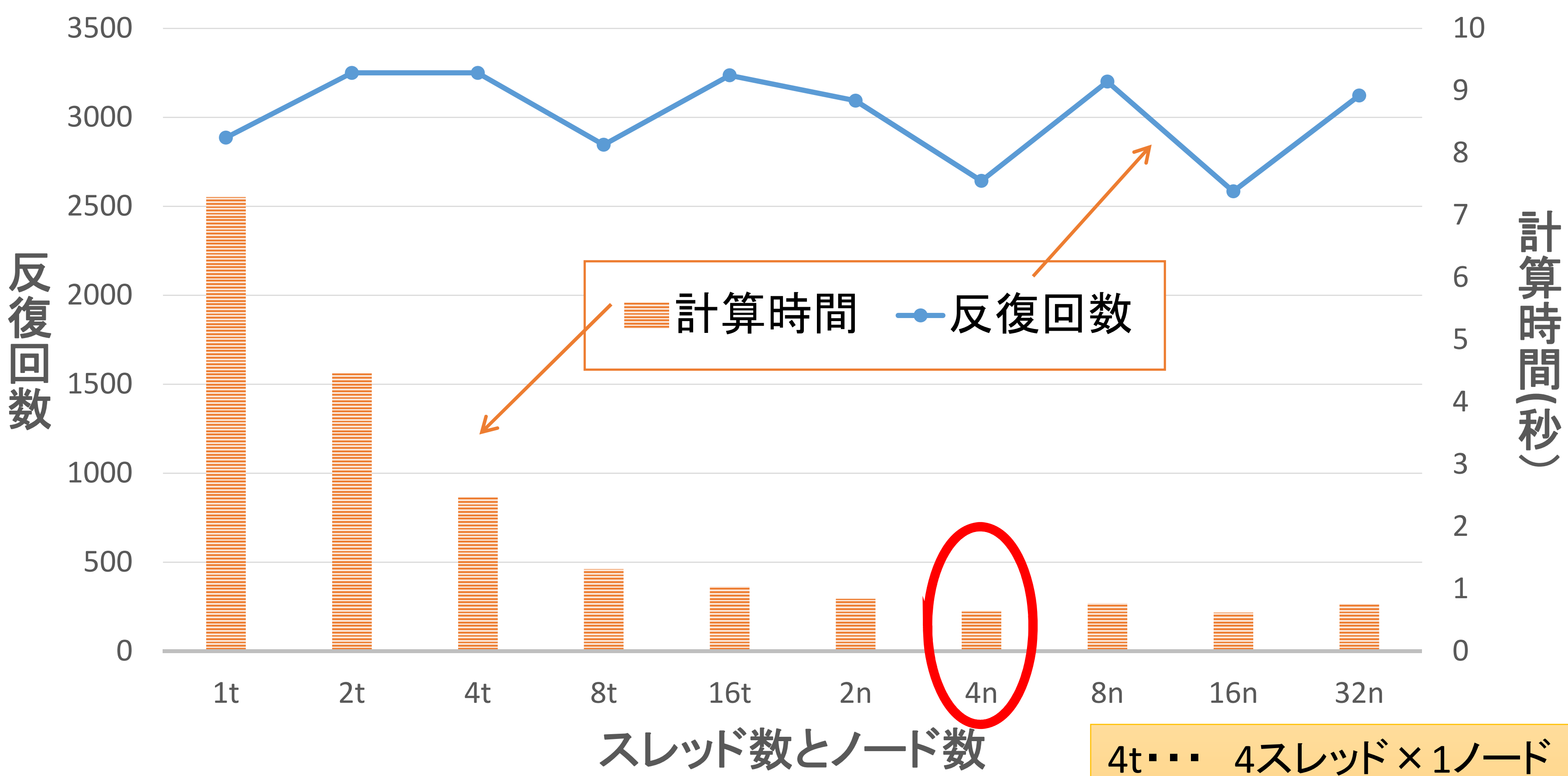
全て同じ結果になる

共役勾配法 改良①を使ったMPSのシミュレーション



同じスレッド数であれば同じシミュレーション結果が得られる

スレッド数やノード数の変化による反復回数と演算時間の影響(行列名(フロリダ行列)：chem_master1)



4t・・・　4スレッド×1ノード
8n・・・16スレッド×8ノード

- スレッド数やノード数を変えるだけで、反復回数は大きく変化する
- 上記の場合4ノードで実行したときが一番速くなる

第２部
高精度行列-行列積
ライブラリの開発

共同研究者:
尾崎克久(芝浦工大)、
荻田武史(東京女子大)、
大石進一(早大)

研究目的

- 行列−行列積に代表される BLAS (Basic Linear Algebra Subprograms) は、多くの線形計算で必須
- BLASを含む従来の数値計算ライブラリは演算速度は考慮しているが、**計算結果の正確性の考慮が不十分**
- BLASを精度保証する研究が、早稲田大学の大石教授のグループにより進められている

背景

- 成分が浮動小数点数で表された
  行列-行列積を〈高精度〉に計算したい
- 応用
  - 行列の残差の計算
  - 行列積の区間演算
    - 行列の正則性の証明に利用
    - 連立一次方程式の数値解に対する精度保証に利用
  - 高精度な行列分解に使用可
  - 疑似多倍長精度への応用
- 高精度行列−行列積の方法（尾崎の方法）
  は通常の行列積ルーチンを使用するため、従来のHPC技術の適用が可能

尾崎の方法の概要（1/3）

行列積AB

提案手法 →

複数の行列の和
行列は浮動小数点数

エラーフリー変換

$$C = AB = \sum_{q=1}^{r} C_q,$$

$$C_q \in F^{m \times p}$$

28

尾崎の方法の特徴

- 浮動小数点数の表現範囲内で、**極度にばらつく要素値**に対する行列積を、高精度に行える
  - IEEE754の doubleの表現範囲 ：$2.22*10^{-308}$ ～$1.79*10^{308}$
- 以上を演算効率よく行うため、**演算精度を保証した上で**〈double型演算の行列-行列積〉と〈高精度の総和演算〉に変換

尾崎の方法の概要

1. *A*、*B*の行列分解（エラーフリー変換）

$$A \mapsto \sum_{i=1}^{p} A^{(i)},\ B \mapsto \sum_{j=1}^{q} B^{(j)}$$

2. 分解行列に対する行列-行列積

$$C^{(i)} = A^{(i)} B^{(i)},\ i = 1,2,\ldots,pq$$

3. 行列積の結果を高精度総和演算*accsum*で足しこみ、演算結果*C*を得る

$$C = \sum_{i=1}^{pq} accsum\ (C^{(i)})$$

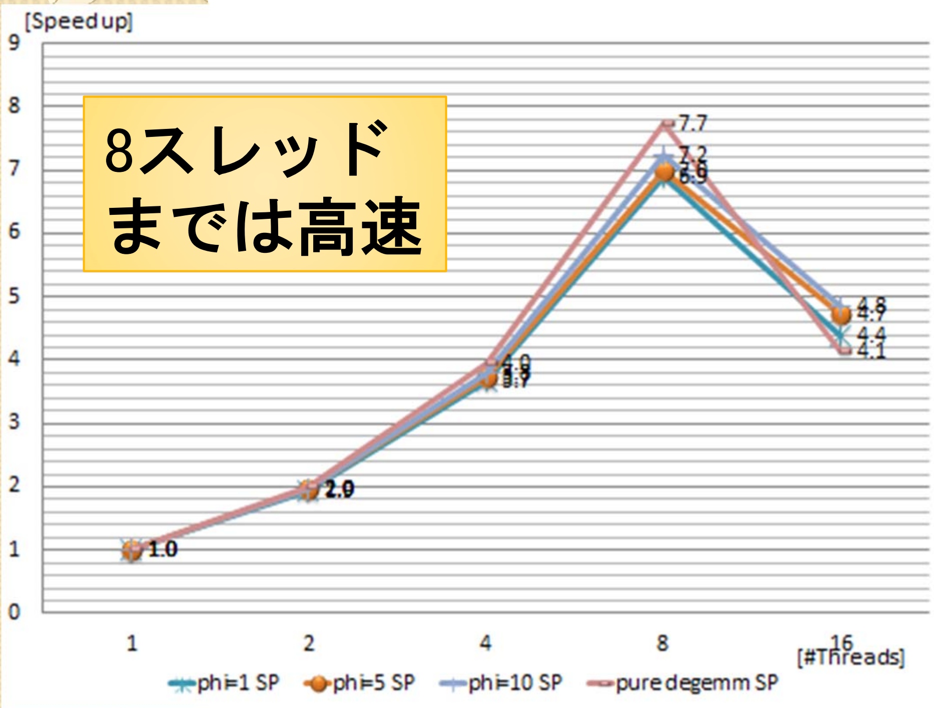
尾崎の方法の行列-行列積のスレッド並列化による高速化

- **以下のスレッド並列化手法の確立**

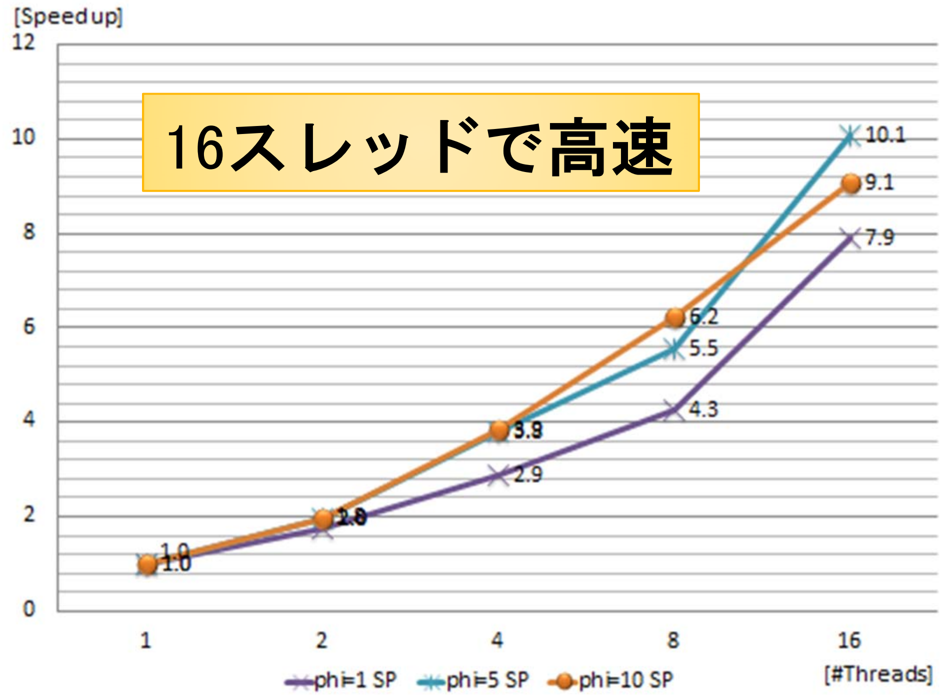
I. 一つの行列-行列積を行うBLAS関数dgemmをスレッド並列化する方法（dgemmスレッド並列化）

II. dgemmは逐次計算を行い演算の並列性を抽出し、スレッド並列化する方法（逐次dgemmで問題レベル並列化）

性能評価結果（T2K, N=8000）



(a)dgemmスレッド並列化



(b)逐次dgemmで
問題レベル並列化

尾崎の方法と通常の行列-行列積の演算精度（相対精度の最大値）

(a)乱数行列の積 **通常の倍精度演算に対し8桁ほど高精度化**

乱数 / phi	1		5		10	
次元	近似	提案	近似	提案	近似	提案
10	8.83E−15	1.00E−16	6.79E−15	1.01E−16	1.97E−14	1.02E−16
100	5.21E−12	1.10E−16	1.08E−12	1.10E−16	1.30E−11	1.11E−16
1000	3.05E−09	1.11E−16	4.77E−09	1.11E−16	4.66E−09	1.11E−16
8000	2.03E−08	1.11E−16	1.96E−08	1.11E−16	2.37E−07	1.11E−16

(b)<乱数行列>と<乱数行列から生成される逆行列>との積

inv / phi	1	
次元	近似	提案
10	3.73E+02	1.10E−16
100	1.54E+03	1.11E−16
1000	6.83E+04	1.11E−16
8000	9.34E+05	1.11E−16

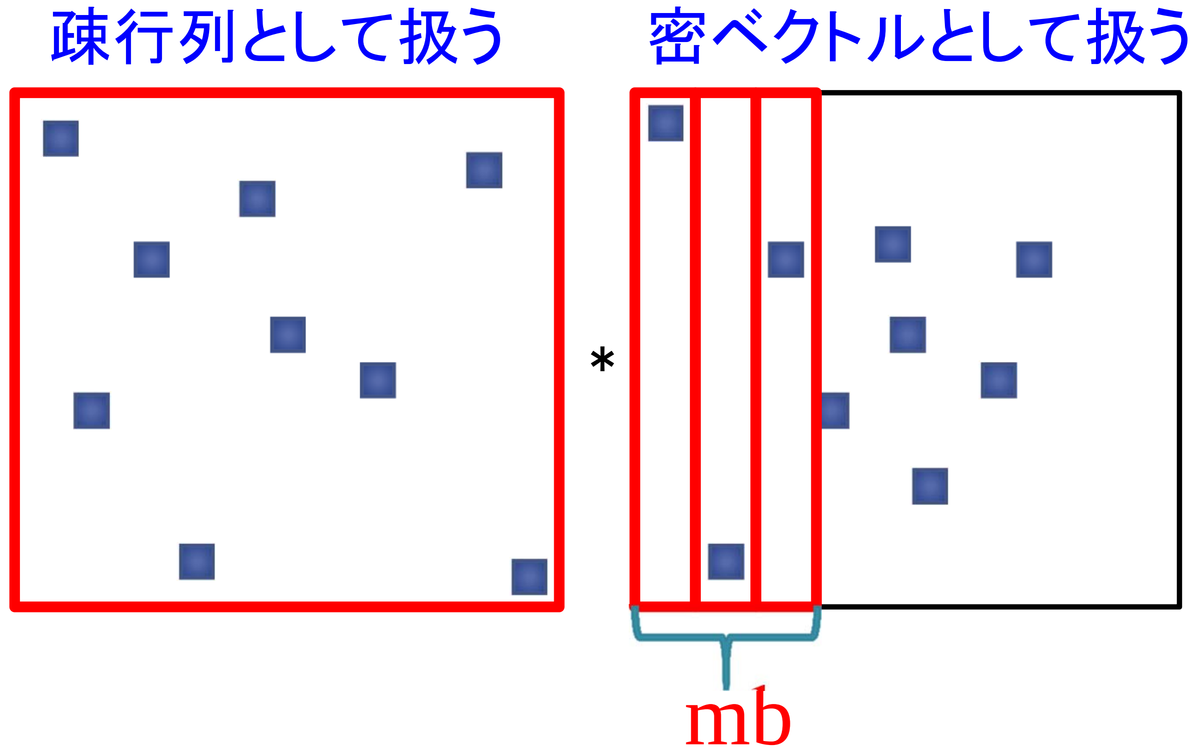
打消しが激しく起きるケースでも精度の破たんなし

尾崎の方法の疎行列演算化による高速化

- 尾崎の方法による行列-行列積演算の中身は、入力行列の数値に依存し、分解の最終段階では密行列から疎行列化していく
- そこで、ある疎度以上になる場合、以下を実行し、演算量の削減を行う
  1. 疎行列データ形式に変換
  2. 疎行列-ベクトル積(SpMV)の実行
- 開発方針
  - 疎行列格納形式：CRS(Compressed Row Storage)
  - 複数右辺のSpMVによる高速化
  - 疎行列データ形式への変換時間を考慮

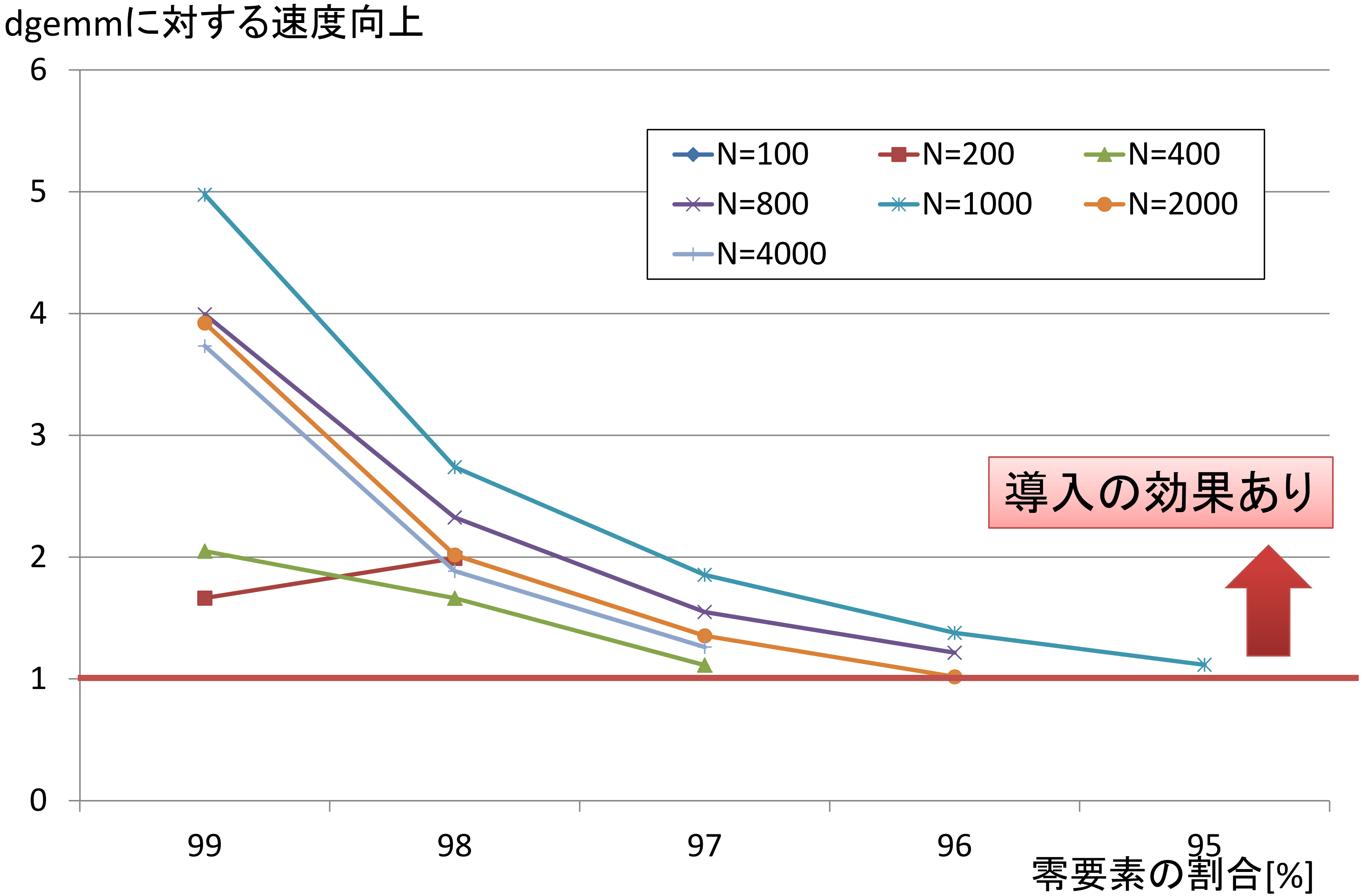
適用した疎行列演算

- 疎行列-疎行列積を、疎行列-ベクトル積(SpMV)を用いて実装
  - 右辺bは実際は疎だが、密行列として扱う
  - 複数の右辺をもつSpMVになる

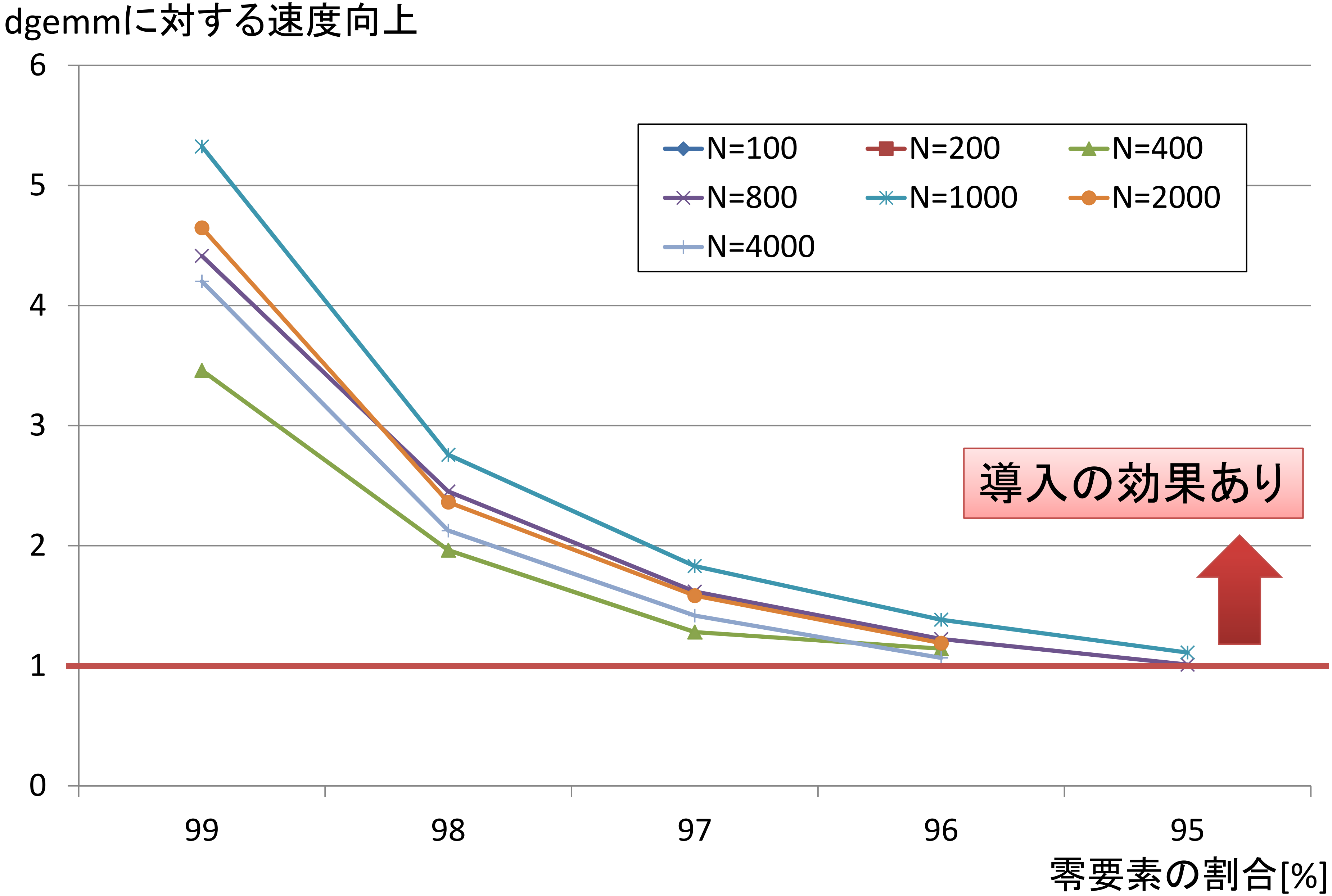


東京大学情報基盤センター
FX10スーパーコンピュータシステム
における疎行列化の効果の調査

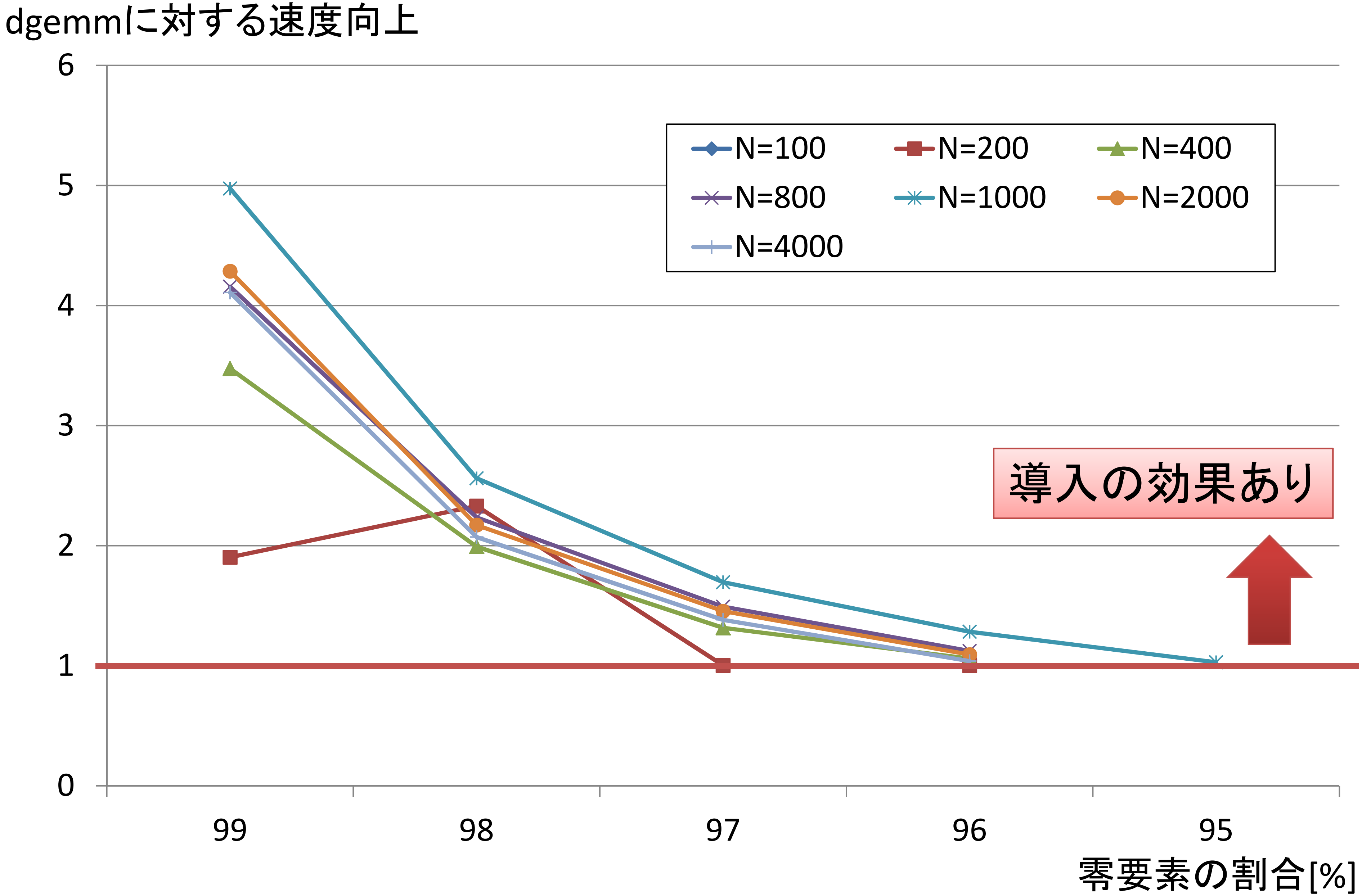
dgemmに対する速度向上（FX10、#thread=16）



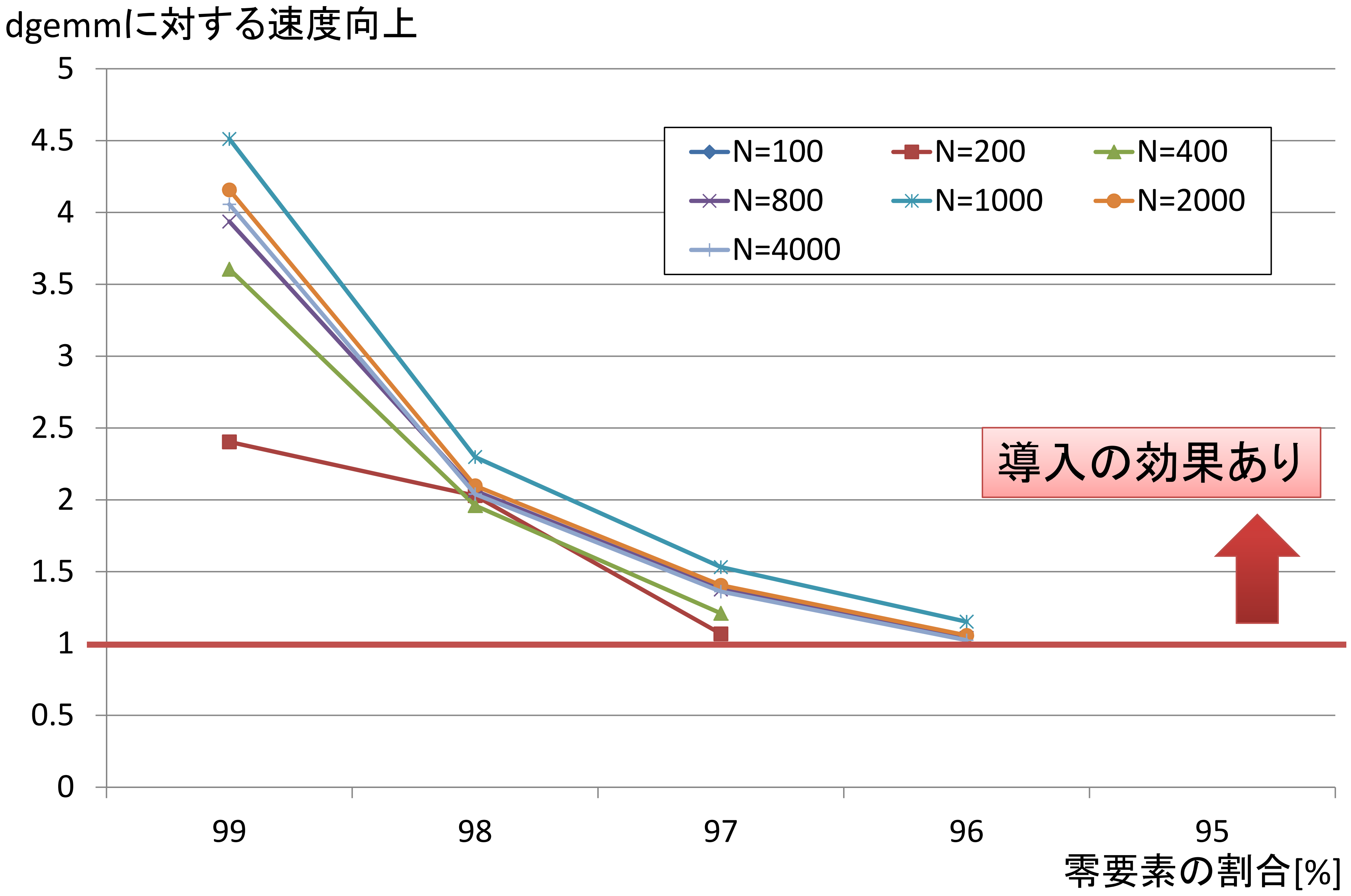
dgemmに対する速度向上（FX10、#thread=8）



dgemmに対する速度向上（FX10、#thread=4）



dgemmに対する速度向上（FX10、#thread=1）



疎行列化の性能調査のまとめ（FX10）

- 零要素の占める割合が約97%以上ないと効果なし
- N=100以下の行列では効果なし
- 1回の行列-行列積演算あたり、
  最大で約5倍の速度向上が期待できる
- N=1000の時、効果が大きい

疎行列化が効果がある実例

疎行列化をして効果がある例（FX10）

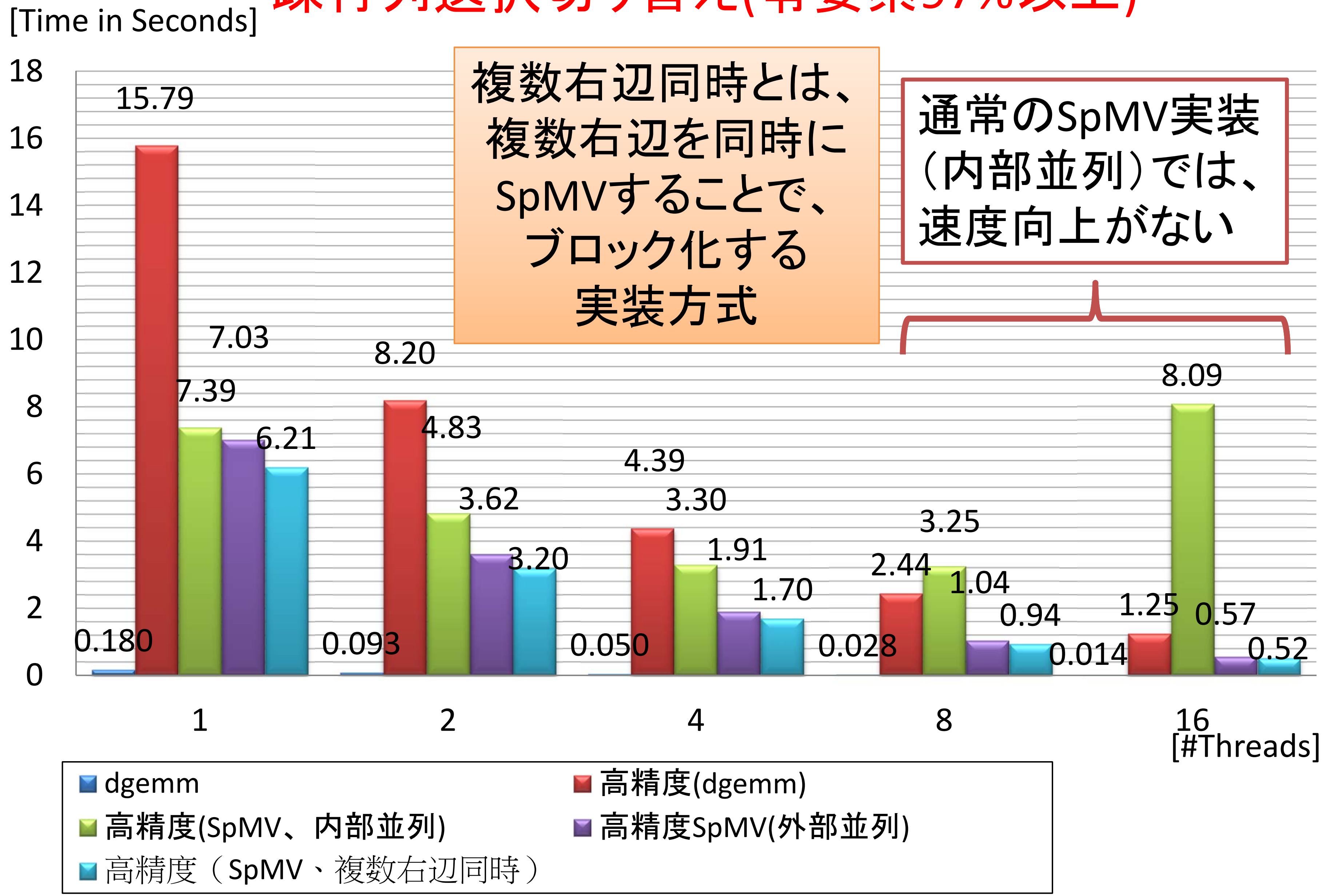
- N=1000
- 通常の行列要素値はpow(10,rand()%1)で生成
- 全要素に対し1%のデータに対して、以下の「特別に」大きい値を設定
  - pow(10,rand()%200)
  - 各行に少なくとも1要素は、上記の値を挿入する。
  - 各行とも、ほぼ均等の個数、挿入する。
  - 挿入場所（列の位置）は、乱数で決定。
  - つまり、99%は小さい値になるので、分離後の行列のゼロ要素が占める値が99%程度になることを想定

疎行列化をして効果がある例（FX10）

- 尾崎の方法による、エラーフリー変換における分解後の行列は29個
（行列Aに対しての分解行列の個数）
- **零要素が占める割合が97%以上の行列は以下の20個（零行列を除く）**
  - i:0 / SpNum:998605 / 99.860500 % / iAsp:1
  - i:1 / SpNum:998210 / 99.821000 % / iAsp:1
  - i:2 / SpNum:997864 / 99.786400 % / iAsp:1
  - i:3 / SpNum:998769 / 99.876900 % / iAsp:1
  - i:4 / SpNum:998825 / 99.882500 % / iAsp:1
  - i:5 / SpNum:997839 / 99.783900 % / iAsp:1
  - i:6 / SpNum:998869 / 99.886900 % / iAsp:1
  - i:7 / SpNum:998813 / 99.881300 % / iAsp:1
  - i:8 / SpNum:998821 / 99.882100 % / iAsp:1
  - i:9 / SpNum:997792 / 99.779200 % / iAsp:1
  - i:10 / SpNum:996847 / 99.684700 % / iAsp:1
  - i:11 / SpNum:995873 / 99.587300 % / iAsp:1
  - i:12 / SpNum:997948 / 99.794800 % / iAsp:1
  - i:13 / SpNum:995953 / 99.595300 % / iAsp:1
  - i:14 / SpNum:996957 / 99.695700 % / iAsp:1
  - i:15 / SpNum:994959 / 99.495900 % / iAsp:1
  - i:16 / SpNum:989986 / 98.998600 % / iAsp:1
  - i:17 / SpNum:989908 / 98.990800 % / iAsp:1
  - i:18 / SpNum:985982 / 98.598200 % / iAsp:1
  - i:19 / SpNum:972135 / 97.213500 % / iAsp:1
  - i:20 / SpNum:959328 / 95.932800 % / iAsp:0
  - i:21 / SpNum:936590 / 93.659000 % / iAsp:0
  - i:22 / SpNum:934748 / 93.474800 % / iAsp:0
  - i:23 / SpNum:867517 / 86.751700 % / iAsp:0
  - i:24 / SpNum:836006 / 83.600600 % / iAsp:0
  - i:25 / SpNum:726573 / 72.657300 % / iAsp:0
  - i:26 / SpNum:818917 / 81.891700 % / iAsp:0
  - i:27 / SpNum:1000000 / 100.000000 % / iAsp:2
  - i:28 / SpNum:1000000 / 100.000000 % / iAsp:2

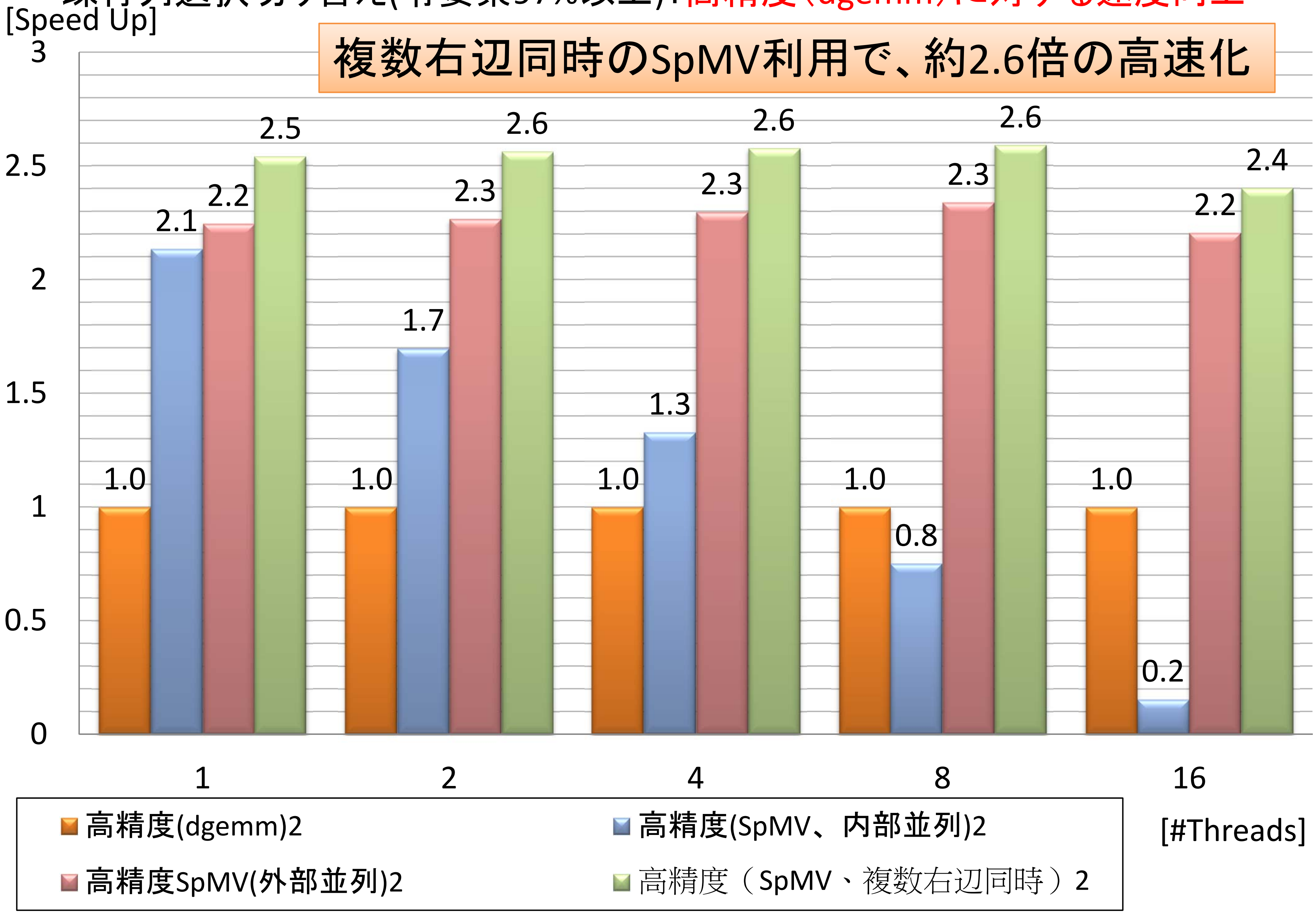
N=1000（例題） FX10(1ノード, 16コア）

～疎行列選択切り替え(零要素97%以上)～



N=1000（例題）FX10(1ノード, 16コア）

～疎行列選択切り替え(零要素97%以上)：高精度（dgemm）に対する速度向上～



まとめ（第１部）

- 連立一次方程式の反復解法の並列化では、**内積計算の加算順序が収束精度に影響を与える**
- 通常、逐次処理と並列処理では、演算結果は一致しない
- 逐次処理と演算結果を一致させるために、**並列加算の足しこみ順序を逐次と同一にする**必要あり
  - **富士通社のMPIライブラリには専用オプションあり**
    - ただし、足しこみ時の実行時間を犠牲にする
    - 精度と実行時間のトレードオフ
  - **演算順序が変わるかは、ハードウェアやコンパイラの実装依存**
    - gcc + intel CPUでは変わるが、富士通コンパイラ+Sarc64 IX-fxでは変わらないことがある
- **足しこみ順序の固定化（≠逐次と同じ順序化）**をすると、同一のスレッド数では計算結果が一致
  - 実行時間と収束精度の観点から実用性が高い

まとめ（第２部）

- 倍精度の行列-行列積演算において、高精度計算を行う**尾崎の方法**がある
- エラーフリー変換と高精度和により、倍精度の範囲内で、高精度演算を実現
- エラーフリー変換で生じる多数の行列-行列積演算をスレッド並列化するには、**各スレッド実行において逐次BLASを呼び出す実装が良い**
- エラーフリー変換では疎行列演算になるため、**疎行列演算を用いると高速化**
- **今後、密行列ライブラリ(LAPACK、ScaLAPACK)や、疎行列反復解法への適用が期待される**

ご清聴ありがとうございます