



# 新ベンチマークプログラム： HPCGの概要と「京」における性能

理化学研究所 計算科学研究機構

運用技術部門 ソフトウェア技術チーム



南一生



RIKEN ADVANCED INSTITUTE FOR COMPUTATIONAL SCIENCE



## OUTLINE

2

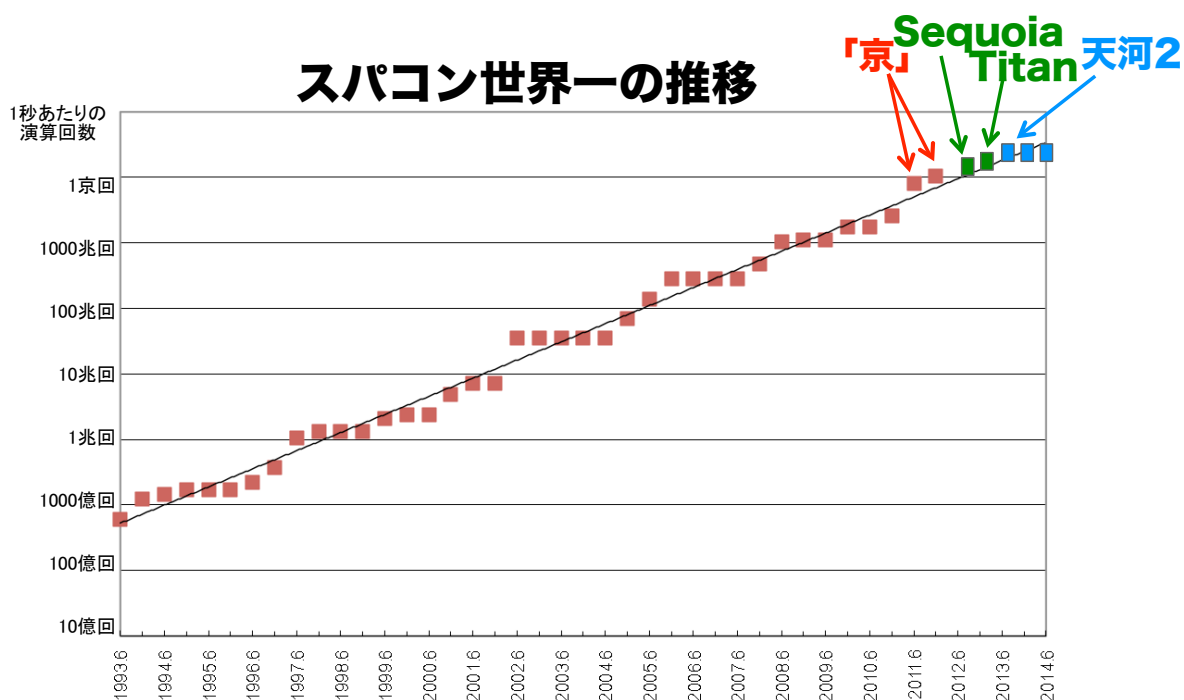
- はじめに
- LINPACK
- HPCGとは
- HPCGベンチマークプログラム
- HPCGのチューニングと性能

# はじめに

2014年8月26日SS研HPCフォーラム

## スパコン世界一をめぐる競争

4



**平均すると1年に約1.9倍の性能向上！**

**「京」は、世界初のスパコンCRAY-1（1976年）の約7000万倍！**

2014年8月26日SS研HPCフォーラム

世界初のスパコンCRAY-1の演算性能は、約160メガフロップス  
一方、iPhone4Sの演算性能は、約140メガフロップス



1976年



2011年



=

2014年8月26日SS研HPCフォーラム

## 最近のスパコンランキングの推移

	Jun. 2011		Nov.2011		Jun. 2012		Nov. 2012		Jun. 2013～ Jun. 2014	
	System (country)	PFLOPS	System (country)	PFLOPS	System (country)	PFLOPS	System (country)	PFLOPS	System (country)	PFLOPS
1	K computer (JPN)	8.16	K computer (JPN)	10.51	Sequoia (USA)	16.32	Titan (USA)	17.59	Tianhe-2 (CHN)	33.86
2	Tianhe1A (CHN)	2.57	Tianhe1A (CHN)	2.57	K computer (JPN)	10.51	Sequoia (USA)	16.32	Titan (USA)	17.59
3	Jaguar (USA)	1.76	Jaguar (USA)	1.76	Mira (USA)	8.16	K computer (JPN)	10.51	Sequoia (USA)	17.17
4	Nebulae (CHN)	1.27	Nebulae (CHN)	1.27	SuperMUC (GER)	2.90	Mira (USA)	8.16	K computer (JPN)	10.51
5	TSUBAME2.0 (JPN)	1.19	TSUBAME2.0 (JPN)	1.19	Tianhe1A (CHN)	2.57	JUQUEEN (GER)	4.14	Mira (USA)	8.59

2014年8月26日SS研HPCフォーラム

# LINPACK

2014年8月26日SS研HPCフォーラム

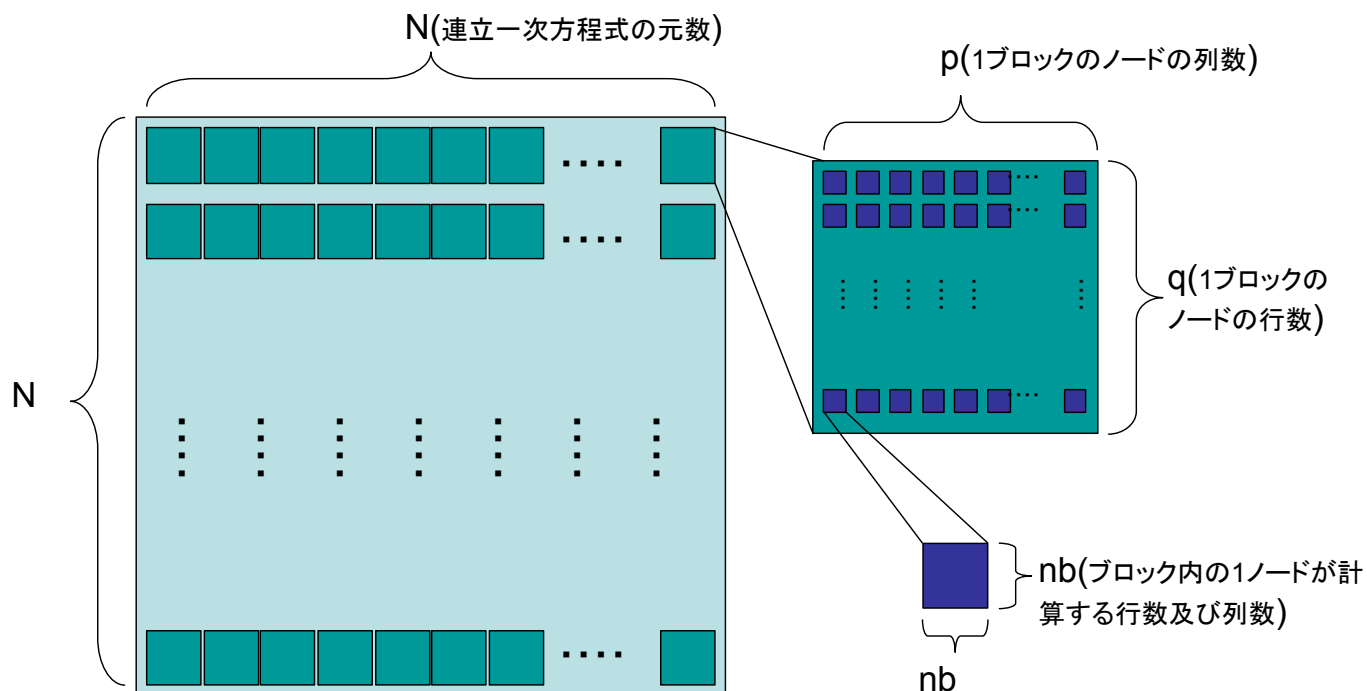


## High Performance LINPACK(HPL) <sup>8</sup>

- 世界で最も**高速なコンピュータシステムの上位500位**までを定期的にランク付けし、評価するプロジェクトとして**TOP500**がある
- TOP500は、米国のテネシー大学の**J. Dongarra博士**によって開発された**密行列計算による連立一次方程式を解く**ベンチマーク・プログラム：LINPACKが用いられている
- LINPACKを用いたTOP500プロジェクトは、**1993年に発足**
- LINPACKが**浮動小数点演算性能**のベンチマーク・テストに重きを置いている

2014年8月26日SS研HPCフォーラム

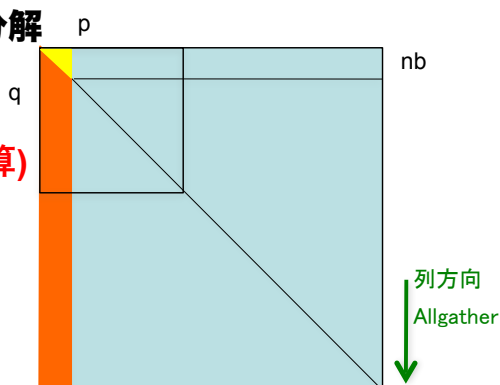
「京」の例



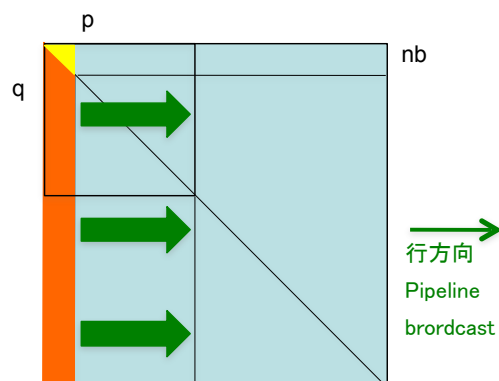
2014年8月26日SS研HPCフォーラム

## (1) Lパネル分解

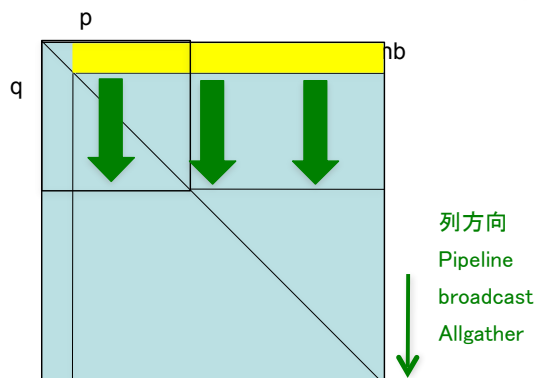
**DTRSM**  
(三角行列計算)



## (2) 分解済みLパネル放送

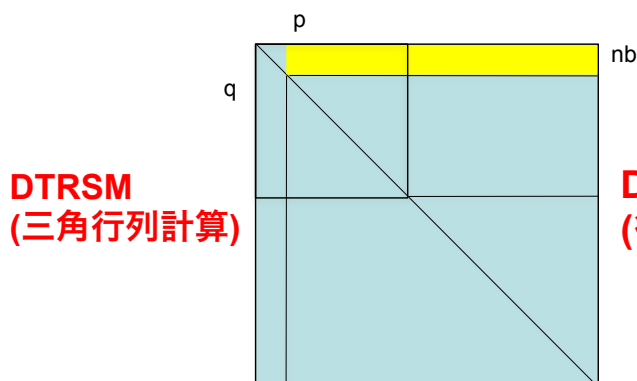


## (3) Uブロック行放送とPivot行交換

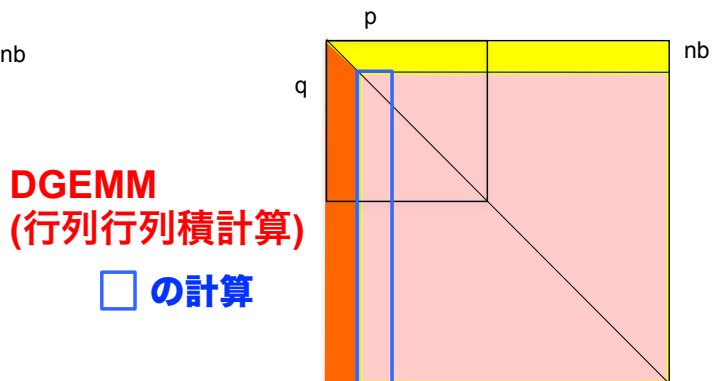


2014年8月26日SS研HPCフォーラム

## (4) Uブロック分解



## (5) 列パネル更新



この更新計算が計算時間の  
大半を占める

# TOP500ベンチマークの成功と失敗

(\*) HPCG WorkShop, 25, March, 2014, Bethesda, MD開催より

- 成功
  - わかりやすさ
  - システム最大性能との線形相関
- 問題点
  - 実問題との乖離
    - 発足して20年以上が経過
    - 実際のアプリケーションで求められる性能要件との乖離
    - Stunt machineの開発を助長する
  - 長すぎる実行時間
    - 途中からの実行など実行時間短縮案を検討中

# HPCGとは

2014年8月26日SS研HPCフォーラム

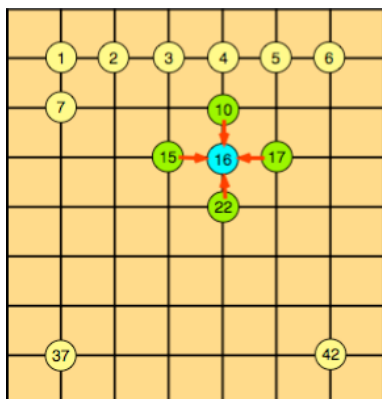
## HPCGベンチマークの提案

14

(\*) HPCG WorkShop, 25, March, 2014, Bethesda, MD開催より

- ベンチマーク自体の紹介
  - HPCG(High Performance Conjugate Gradient)
  - Version 2の紹介
- TOP500との関係
  - 置き換えるものではない
    - TOP500のリストに列を追加してHPCGスコアも表示するなどの案あり
    - WS参加者のなかでは将来的に置き換えるべきという意見もあった

2014年8月26日SS研HPCフォーラム



- 左図のような2次元の領域で微分方程式を解くとする.
- 例として5点差分で差分化.
- 元の微分方程式は,差分化により連立一次方程式を解く事に帰着.

$$Ax = b$$

$$f(x) = \frac{1}{2}ax^2 - bx \quad \mathbf{f(x)の最小値問題}$$

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) - (\mathbf{b}, \mathbf{x})$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{1}{2} \sum_{i=1}^n a_{ik} x_i + \frac{1}{2} \sum_{j=1}^n a_{kj} x_j - b_k$$

$$f'(x) = ax - b = 0 \quad \text{本式を解く事と同値}$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \sum_{i=1}^n a_{ki} x_i - b_k = 0$$

## 共役勾配法(CG法)

$$r_0 = b - Ax_0$$

$$p_0 = r_0$$

$$\begin{aligned} \text{for } & i = 0, 1, 2, \dots \\ & \alpha_i = \frac{(r_i, r_i)}{(p_i, A p_i)} \\ & x_{i+1} = x_i + \alpha_i p_i \\ & r_{i+1} = r_i - \alpha_i A p_i \\ & \beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \\ & p_{i+1} = r_{i+1} + \beta_i p_i \end{aligned}$$

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) - (\mathbf{b}, \mathbf{x})$$

- CG法アルゴリズムの基本形.
- この他にアルゴリズムの派生形あり.
- $f(\mathbf{x})$ が最小値が得られるように $x_i$ の列を求める.
- $r_i$ は残差ベクトルであり互いに直交し一次独立である.
- 残差ベクトルはN元の連立一次方程式にはN個しか存在しない.
- N元の連立一次方程式は高々N回の反復で収束する.
- またAの固有値が縮重しているか密集していれば収束が速くなる性質を持つ.

- $M = U^T U$  ←  $\bullet$  Aに近い正値対称行列:Mを考えコレスキー分解する.
- $$\left. \begin{array}{l} Ux = \tilde{x} \\ U^{-T}b = \tilde{b} \\ U^{-T}AU^{-1} = \tilde{A} \end{array} \right\}$$
 ←  $\bullet$  このようなx,b,Aの置き換えを行う.
- $\tilde{A} = U^{-T}AU^{-1} \approx I$  ←  $\bullet$  置き換えたAの性質は単位行列に近い.  
 $\bullet$  置き換えた行列Aの固有値は1の周りに密集している
- $\tilde{A}\tilde{x} = \tilde{b}$  ←  $\bullet$  置き換えた行列とベクトルに対する連立一次方程式の収束は早いものと期待.

この前処理によりCG法が広く使用されるようになった

2014年8月26日SS研HPCフォーラム

- 不完全コレスキー分解による前処理CG法アルゴリズムの基本形.
- 計算は行列ベクトル積・ベクトルスカラ積・ベクトルの和・内積・除算で構成される.
- 赤線で示した前処理部分は前進代入・後退代入で計算される.

ステップ1:  $\alpha^k = (r_i^k \bullet \underline{(LL^T)^{-1} r_i^k}) / (Ap_i^k \bullet p_i^k)$

ステップ2:  $x_i^{k+1} = x_i^k + \alpha^k p_i^k$

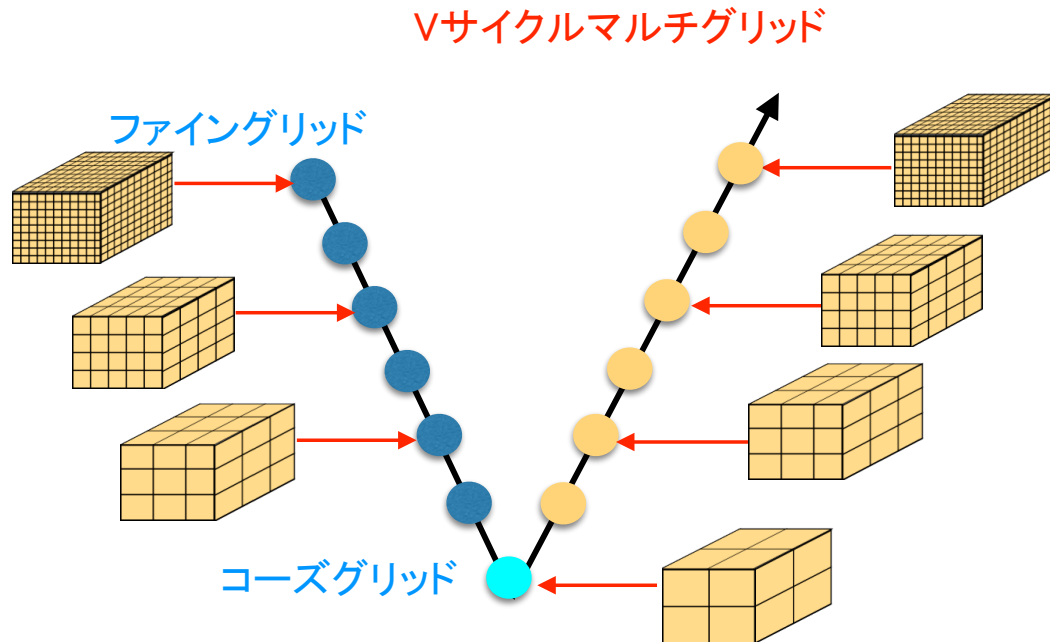
ステップ3:  $r_i^{k+1} = r_i^k - \alpha^k Ap_i^k$

ステップ4:  $\beta^k = (r_i^{k+1} \bullet \underline{(LL^T)^{-1} r_i^{k+1}}) / (r_i^k \bullet \underline{(LL^T)^{-1} r_i^k})$

ステップ5:  $p_i^{k+1} = \underline{(LL^T)^{-1} r_i^{k+1}} + \beta^k p_i^k$

2014年8月26日SS研HPCフォーラム

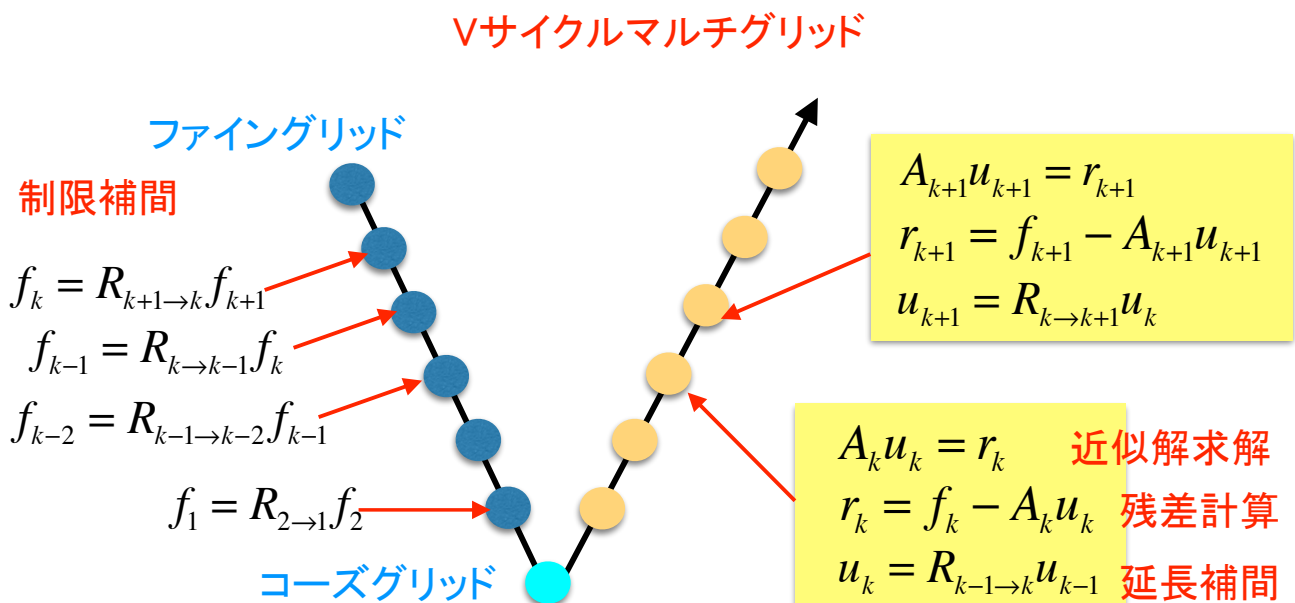
## 連立一次方程式の解法としてのマルチグリッド法の例



2014年8月26日SS研HPCフォーラム

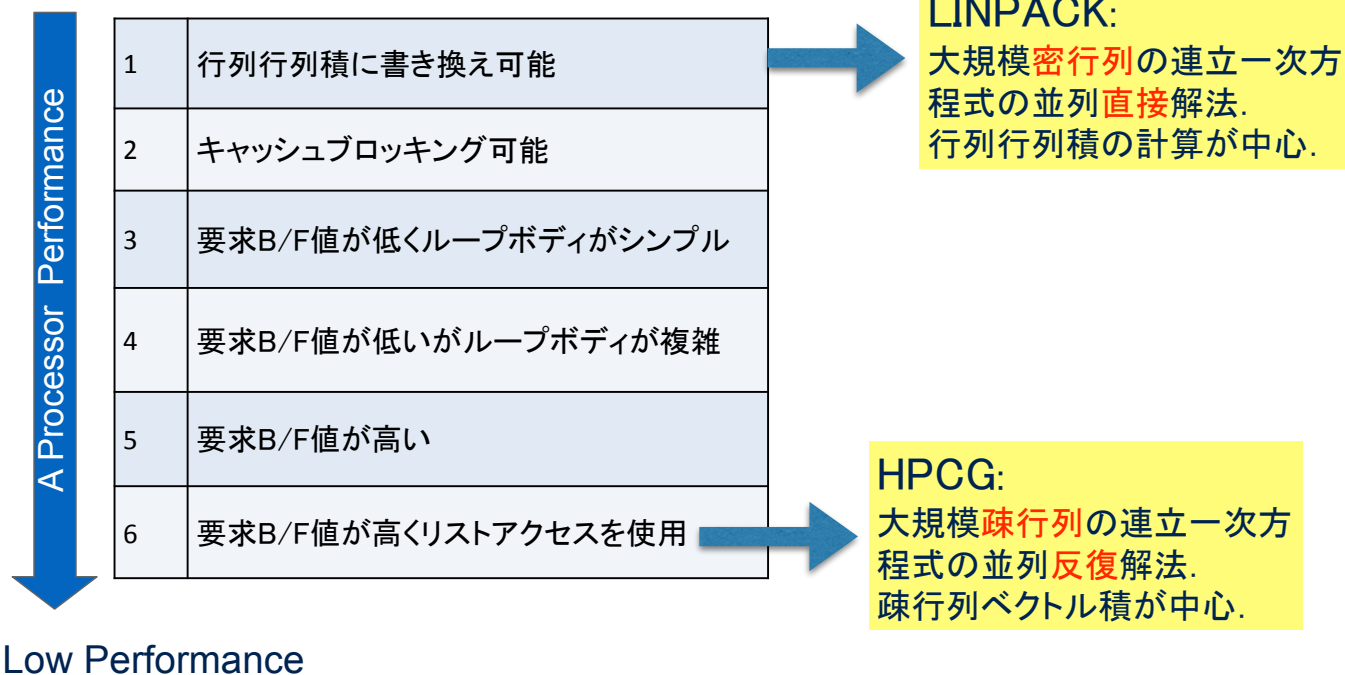
## 連立一次方程式の解法としてのマルチグリッド法の例(NPB MG)

- マルチグリッドも行列ベクトル積の計算が中心



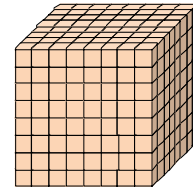
2014年8月26日SS研HPCフォーラム

## High Performance

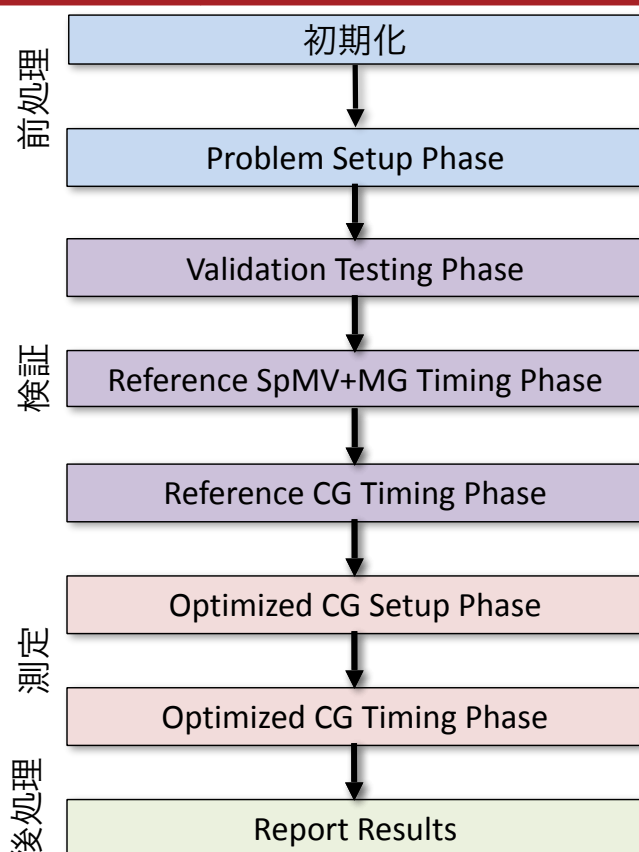


## HPCGベンチマークプログラム

- 連立一次方程式  $\mathbf{Ax} = \mathbf{b}$  を前処理付CG法で解く際の性能をベンチマーク
  - 前処理はVer.2.0以降はMulti-Grid法を採用 (Ver1.1まではGauss-Seidel法)
- 係数行列Aは有限要素法の行列
  - 疎で対称
  - 具体的には直方体領域を規則的に分割したもの
- 特徴
  - LINPACKと比して通信・メモリアクセスの比重が高い
  - メモリアクセスはリストアクセスであり,より高いメモリ性能が必要
  - C++で書かれているが便利なCとしてしか使っていない



2014年8月26日SS研HPCフォーラム



- 大まかに4つの部分に分けられる

### 前処理

- メモリ確保などの初期化
- 問題設定,行列の最適化

### 検証

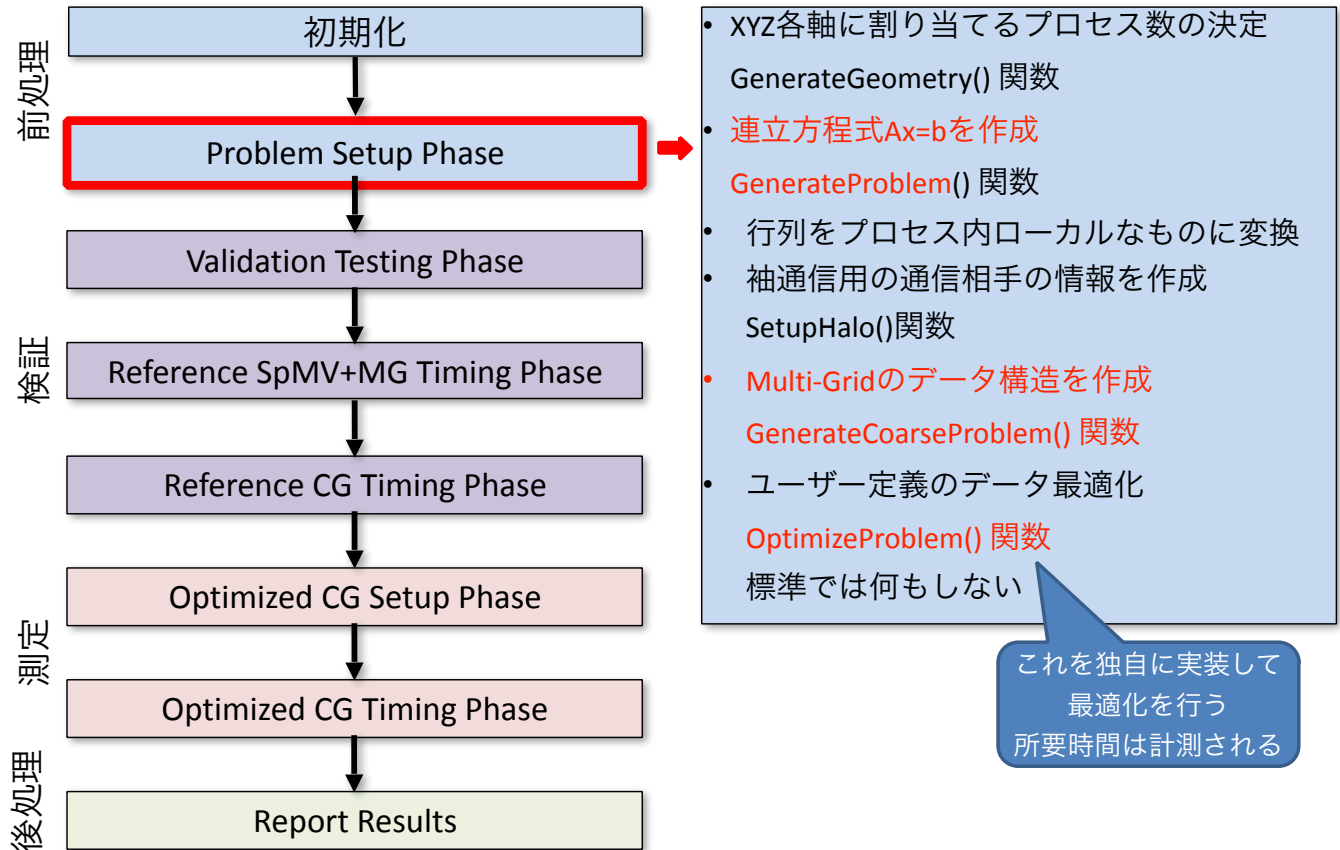
- ユーザーチューニングの正当性検証
- 雑多なデータ測定

### 測定

- 実行時間の予測
- 本番測定 (1時間)

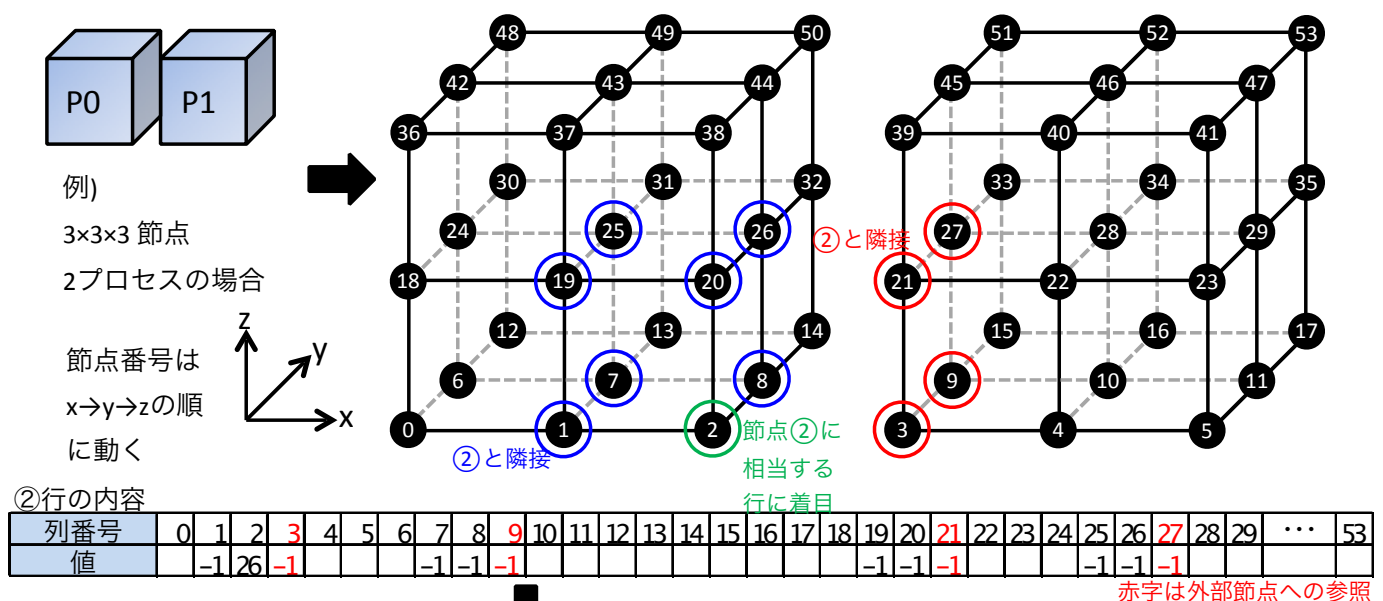
ここでは各部分の主要な処理について紹介

2014年8月26日SS研HPCフォーラム



2014年8月26日SS研HPCフォーラム

連立一次方程式 $Ax=b$ を設定する (この時点ではグローバルな節点番号で作成)



実際の格納形式 ELL形式

非ゼロ番号	0	1	2	3	4	5	6	7	8	9	10	11	12	...	26	27
列番号	1	2	3	7	8	9	19	20	21	25	26	27		...		
値	-1	26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1		...		

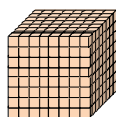
非ゼロ個数は 27個/行 に固定  
global\_int\_t型(実体はint)  
double型

2014年8月26日SS研HPCフォーラム

MG用の粗いメッシュに対応する係数行列を作成

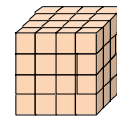
粗い格子は3レベル作成して  
合計4レベルの格子

初期メッシュ

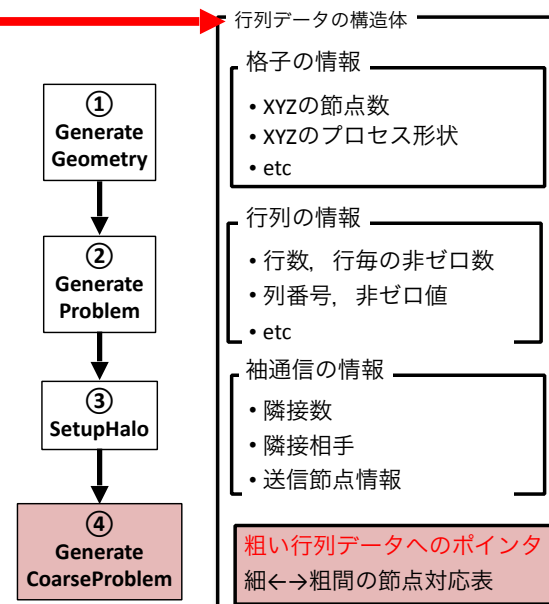
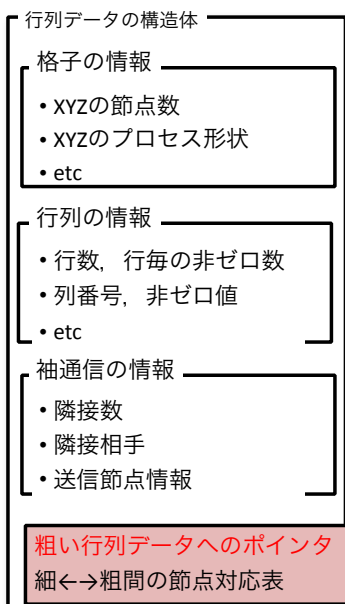
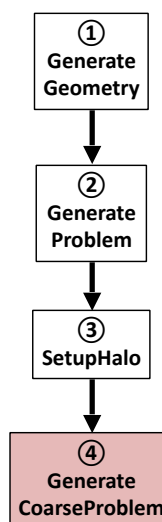


XYZの節点数を1/2にして  
データ作成プロセスを先頭  
から再度呼ぶ

粗いメッシュ

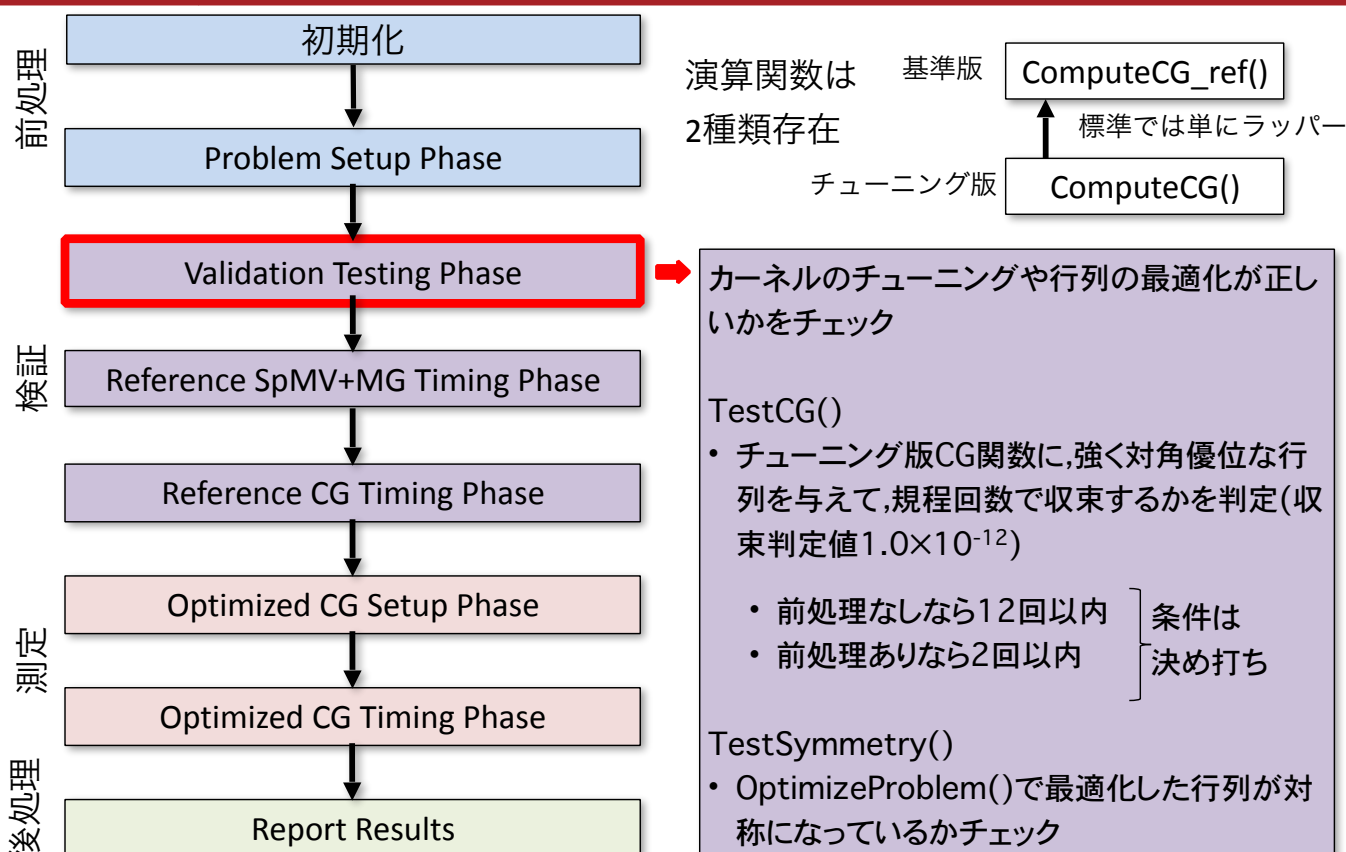


データ作成  
プロセス

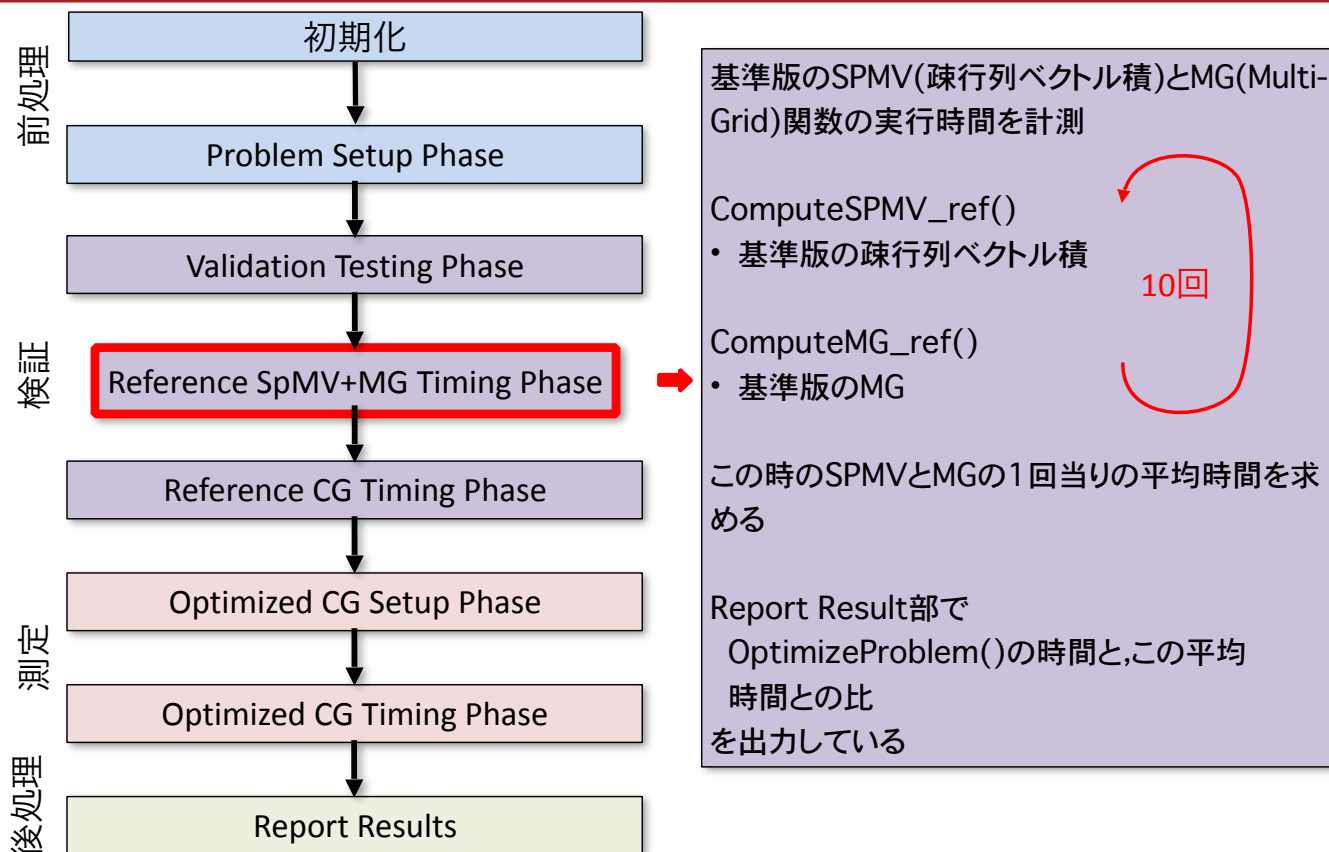


2014年8月26日SS研HPCフォーラム

## HPCGベンチマークアプリの概要

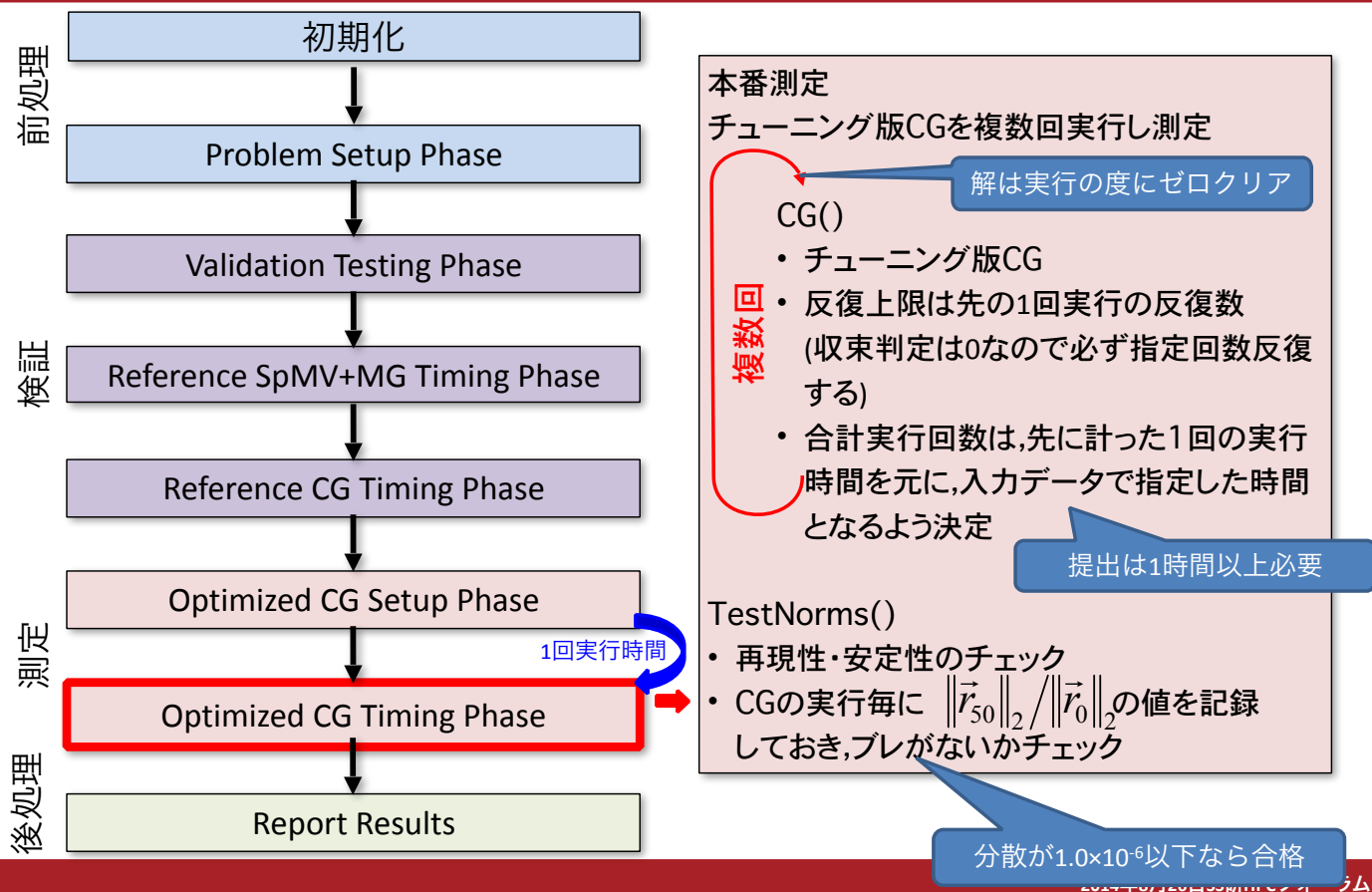


2014年8月26日SS研HPCフォーラム

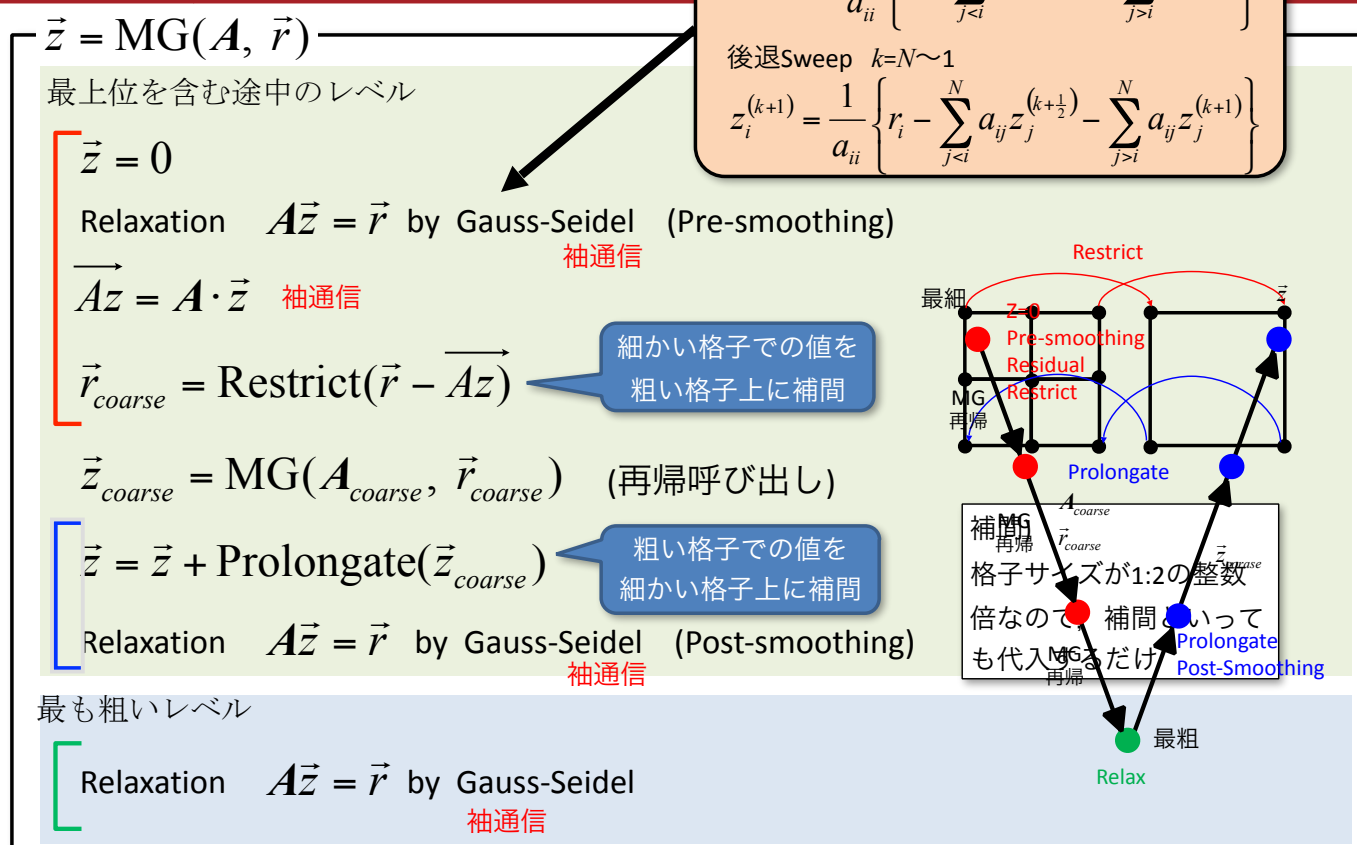


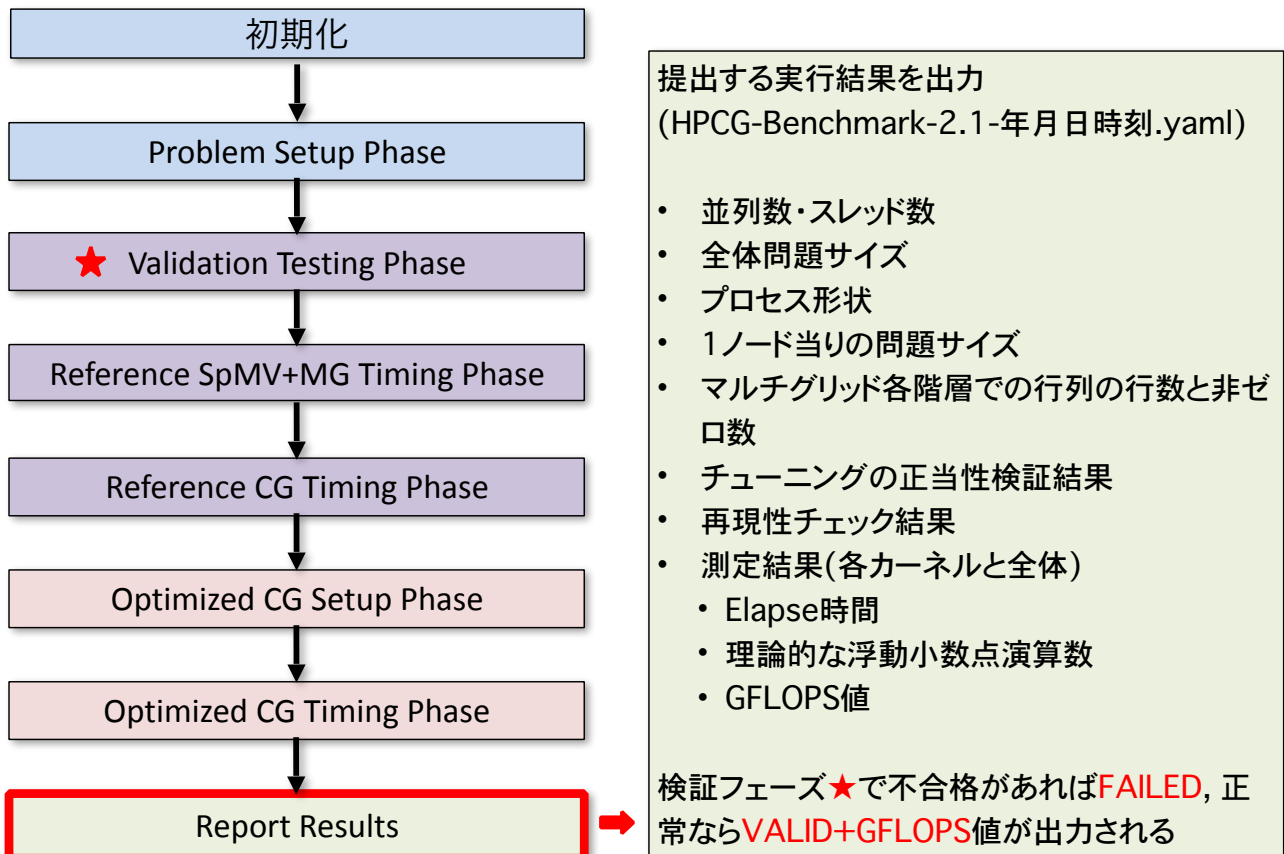
2014年8月26日SS研HPCフォーラム





## MGの処理





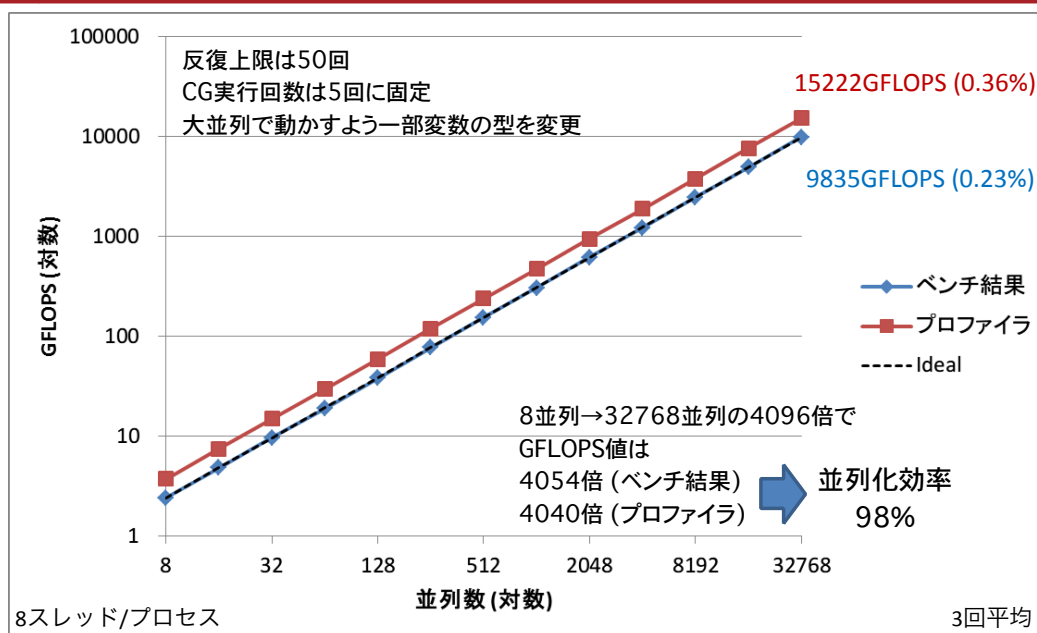
2014年8月26日SS研HPCフォーラム

## HPCGのチューニングと性能

2014年8月26日SS研HPCフォーラム

- コードはAs Isそのまま
  - ただし大並列への対応として一部の変数の型を変更
  - Geometry.hpp中のtypedefされている型をintからlong longに変更
- Make方法
  - QUICKSTART.TXTに従って実行
- テスト問題
  - デフォルトの $104 \times 104 \times 104 = 112$ 万節点/プロセスを利用
  - 測定時のCG繰り返し回数は5回 (本来は1時間分で約50回)

## ウィークスケール測定結果



- 8並列から32768並列まで並列数を換え測定
  - **ベンチ結果** HPCGコードが自身で算出した(提出する) GFLOPS値
  - **プロファイラ** プロファイラを利用して算出した GFLOPS値
- 32768並列までの範囲ではウィークスケール性能は良好

ベンチ結果とプロファイラとで1.5倍程度の差がある  
↓  
浮動小数点演算数カウント方式の違いが原因

## プロファイラでの浮動小数点演算数

- 測定区間で発生する浮動小数点演算全てがカウント対象
- 除算等は複数の浮動小数点演算としてカウント

## HPCGでの浮動小数点演算数

- 理論的に必要な演算のうち,主要なもののみをカウントしており,無視しているものもある

 $\vec{x} = \text{CG}(\vec{A}, \vec{b}, \vec{x}_0, \epsilon, \text{max})$ 

```
SPMV ()
WAXPY ()
DotProduct
for(k=1; k<max && !conv; k++) {
    MG ()
    DotProduct ()
    WAXPY ()
    SPMV ()
    ...
}
```

反復の前処理の分は  
カウントしない

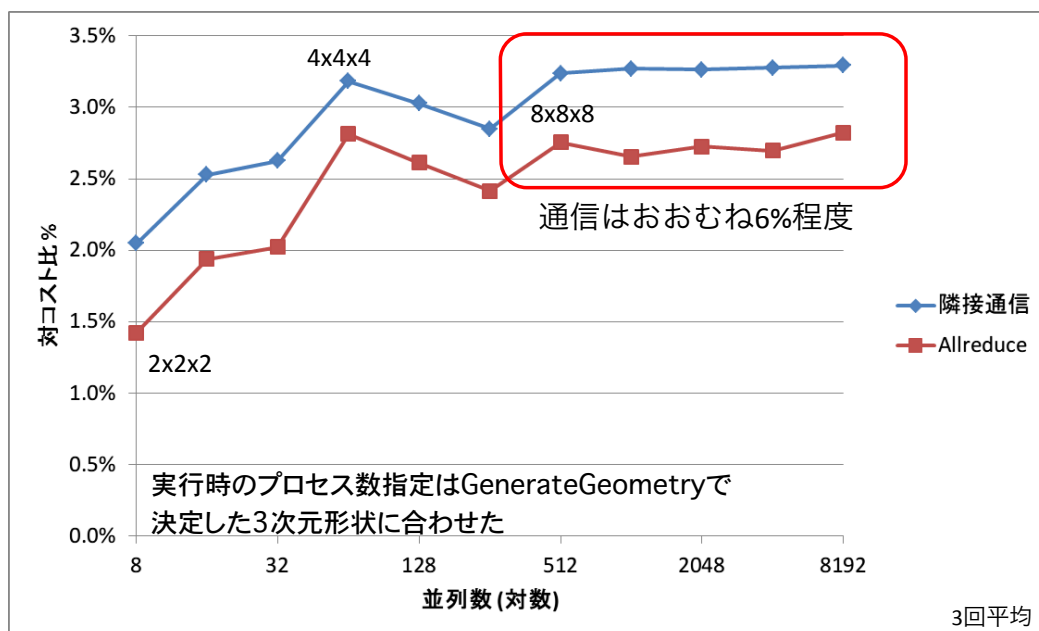
※V2.4ではカウントするようになった

反復内の分だけ  
カウントする

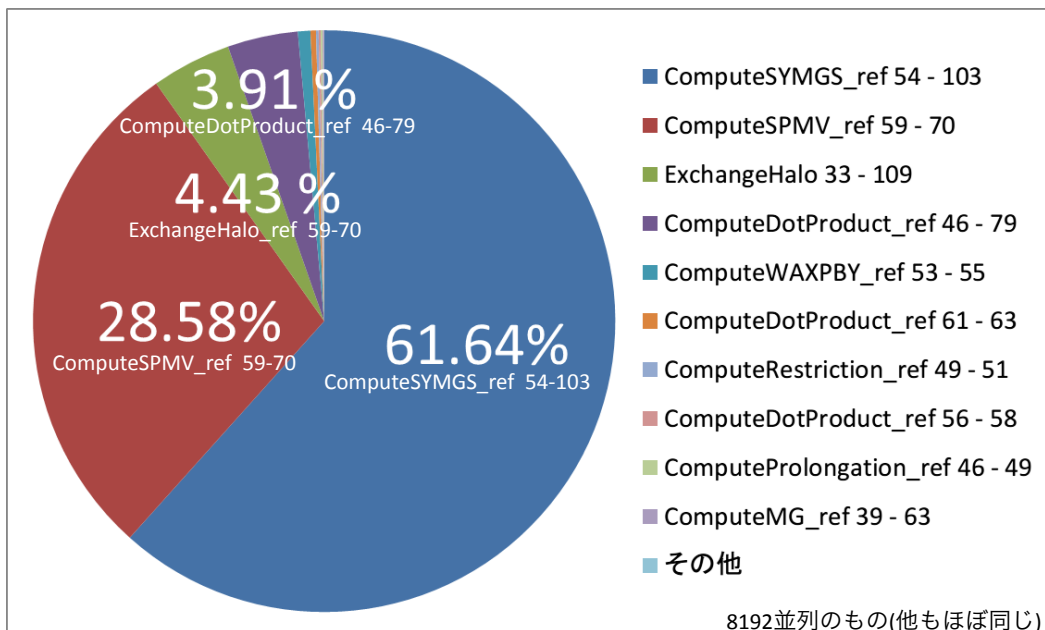
 $\vec{x} = \text{SYMGS}(\vec{A}, \vec{b}, \vec{x})$ の前進ループ

```
for(i=0; i<行数; i++) {
    ...
    for(j=0; j<nz[i]; j++) {
        col = A.mtxIndL[i][j]
        カウントする sum -= val[j] * x[col];
    }
    sum += x[i] * Diag[i];
    sum = sum / Diag[i];
    カウントしない
}
```

# 通信の比率変化



- プロファイラで計測したMPIのコストの,対全コスト比率
- 通信のコストの割合は,512並列以上ではサチってきており,合計6%程度



- ・ プロファイラで採取したCG内で呼ぶ手続きのコスト分布
- ・ 上位4つで98%以上を占める(Allreduce, lsend, lrecv, Waitを含む)
- ・ チューニングはまず単体性能の向上  
ComputeSYMGSとComputeSPMVが主眼に

2014年8月26日SS研HPCフォーラム

## メモリ領域の連続化

### メモリ確保部ソース

```
for(int i=0; i<nrow; ++i){
    matrixValues[i] = new double[27];
    mtxIndL[i] = new int[27];
}
```

行列の各行の情報を保持する配列は  
行毎にnewされているので、メモリ上  
で不連続になっている

### メモリ空間

A.matrixValues[0]  
[ ] [ ] ... [ ]

A.mtxIndL[nrow-1]  
[ ] [ ] ... [ ]

A.mtxIndL[0]  
[ ] [ ] ... [ ]

A.mtxIndL[1]  
[ ] [ ] ... [ ]

A.matrixValues[nrow-1]  
[ ] [ ] ... [ ]

A.matrixValues[1]  
[ ] [ ] ... [ ]

2014年8月26日SS研HPCフォーラム

## メモリ領域の連続化

### メモリ確保部ソース

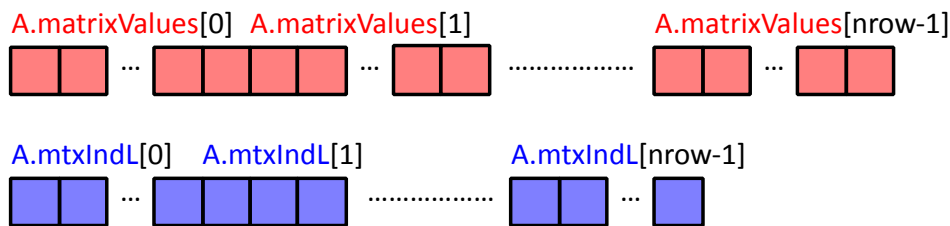
```
double* tmp1 = new double[nrow * 27];
double* tmp2 = new      int[nrow * 27];

for(int i=0; i<nrow; ++i){
    matrixValues[i] = &(tmp1[i*27]);
    mtxIndL[i] = &(tmp2[i*27]);
}
```

メモリ確保を一括して行い、各行の情報が連続するように変更

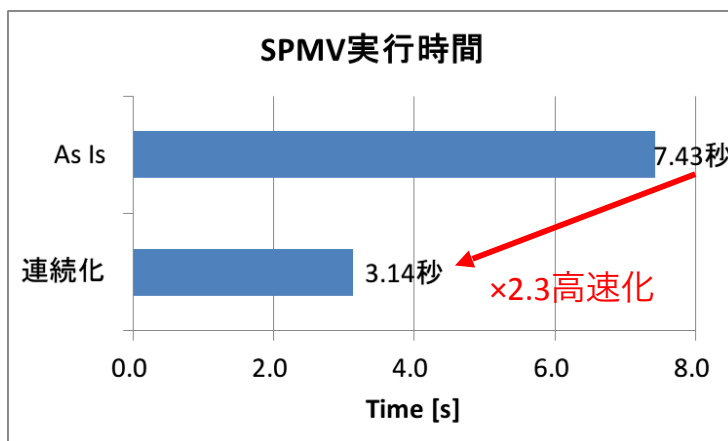
ムダな領域もなくなった

### メモリ空間



2014年8月26日SS研HPCフォーラム

ケース	SPMV 実行時間 (sec)	メモリスループ ット (GB/sec)	L2 スループ ット (GB/sec)	L1D ミス率 (/ロード・ストア 数)	L2 ミス率 (/ロード・ストア 数)
As Is	7.429	25.99	25.72	5.59%	5.56%
連続化	3.137	48.28	55.88	4.88%	4.13%



CG()から直接呼ばれるSPMVについて、実行時間と、キャッシュミス率、スループット等を調べた

※ MG()から呼ばれる分は除外

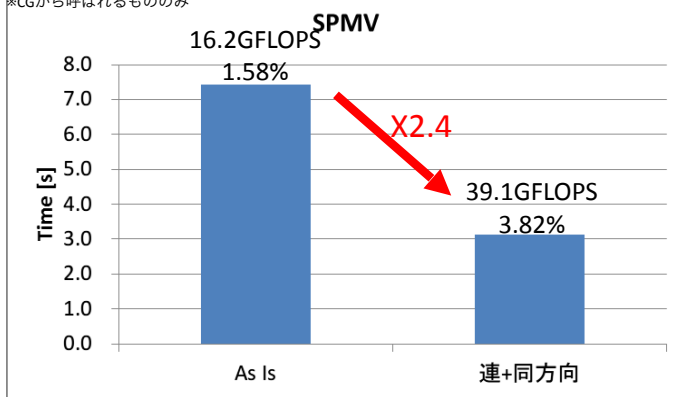
メモリ領域の連続化の効果によりキャッシュミス率減少、スループット向上

実行時間は1/2.3に短縮

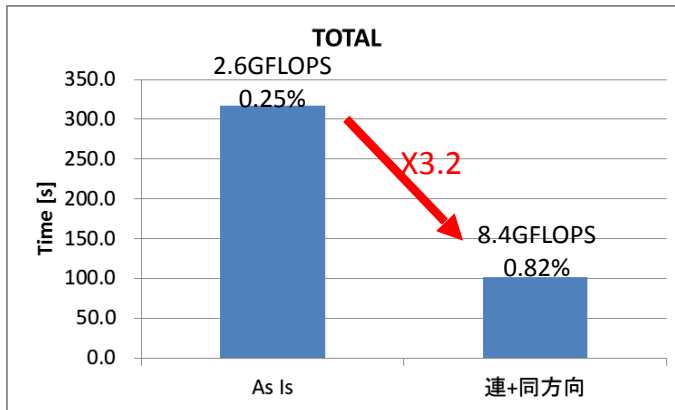
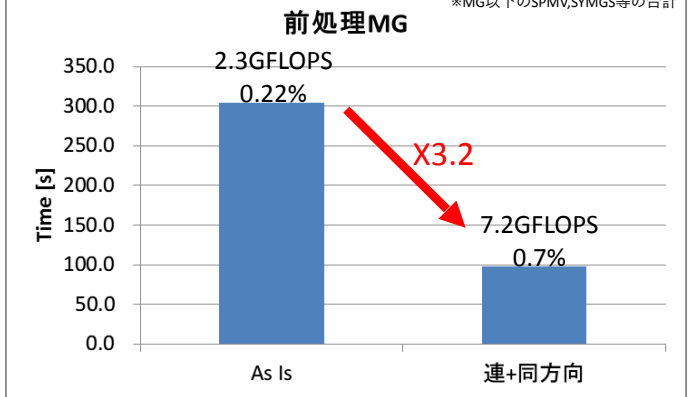
8プロセス、8スレッド/プロセス

2014年8月26日SS研HPCフォーラム

※CGから呼ばれるもののみ

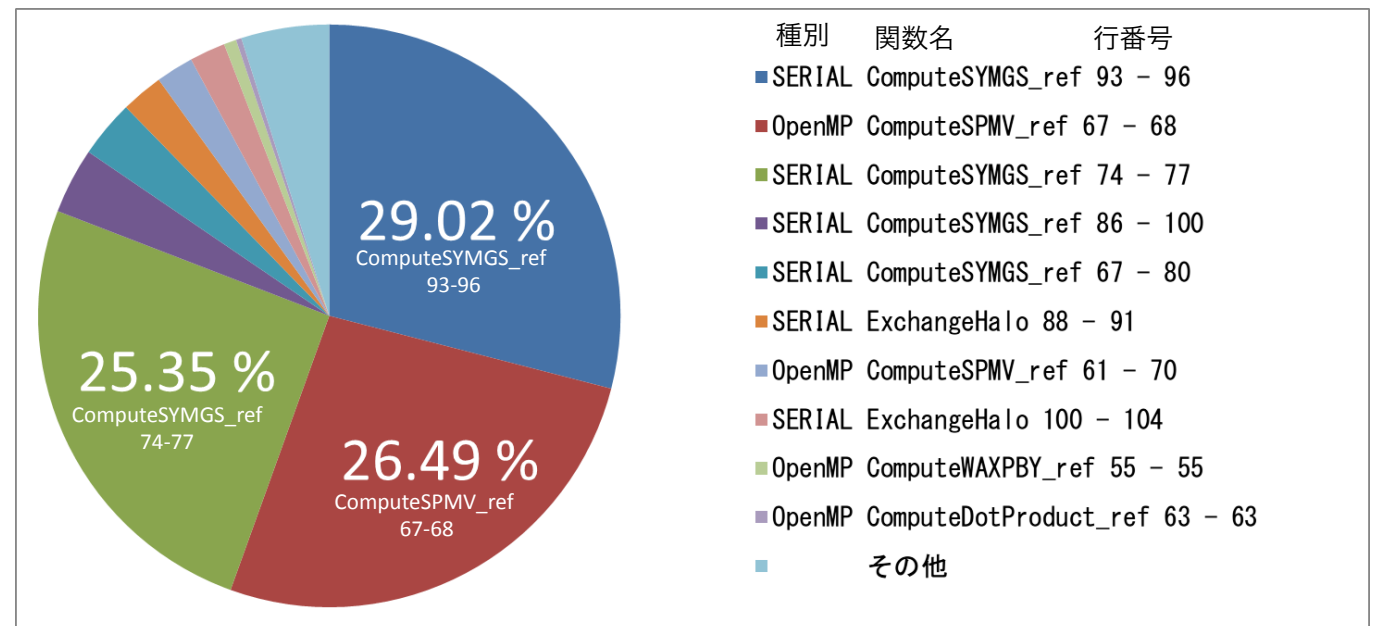


※MG以下のSPMV, SYMGS等の合計



- HPCG自身が測定した提出するデータから抜粋したもの
  - SPMVの時間CG()から呼ばれる分のみの時間
  - 前処理MGはSPMV, SYMGS等々を含む時間
- 2点の簡単なチューニングだけで3.2倍の高速化が得られた
  - 行列のメモリ領域の連続化
  - SYMGSのBAKループでのループ方向の反転

2014年8月26日SS研HPCフォーラム

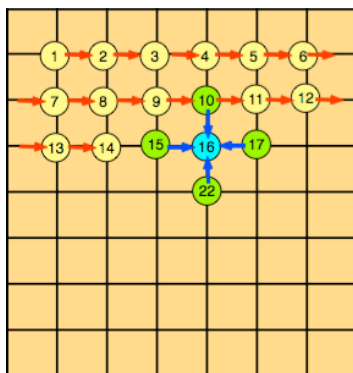


- プロファイラで採取した各手続きのループ単位のコスト比率
  - ComputeSYMGSのループはマルチスレッド化されておらず逐次で動作
- ↓
- ここではComputeSYMGSのマルチスレッド化を実施

2014年8月26日SS研HPCフォーラム

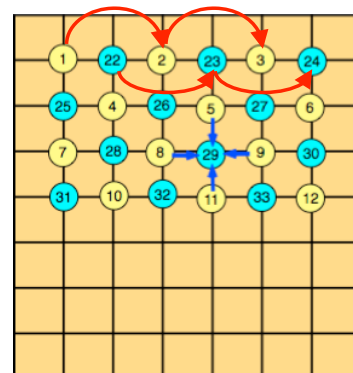
- SYMGSは対称行列についてガウス・ザイデル法を使用したマルチグリッド処理を行う。
- ガウス・ザイデル法 
$$a_{ii}x_i^{(m+1)} = -\sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(m)} + k_i$$
- (m+1)世代の解を得るために(m+1)世代の下三角行列を使用して(リカレンスの発生)。

(a)オリジナル

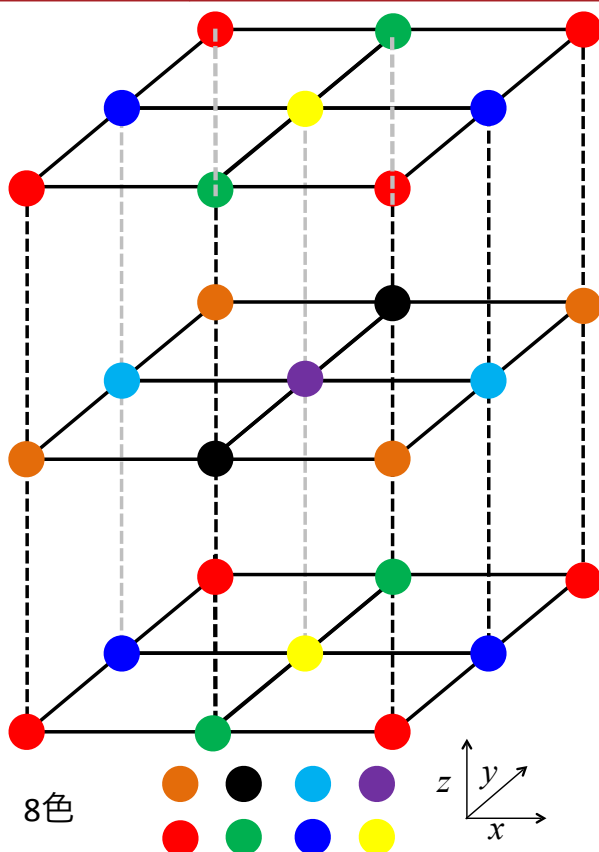


参照関係  
計算順序  
(スイープ順序)

(c)RED-BLACK 2色カラーリング



2014年8月26日SS研HPCフォーラム



XYZ各方向の隣りと斜めを同時に参照する27点  
ステンシルなので、8色必要

```
for(int i=0; i<nrow; i++){
    最内ループ
    SYMGSの外側ループ
}
```

カラーを回す外側ループ追加で3重化  
for(int ic=0; ic<8; ic++){

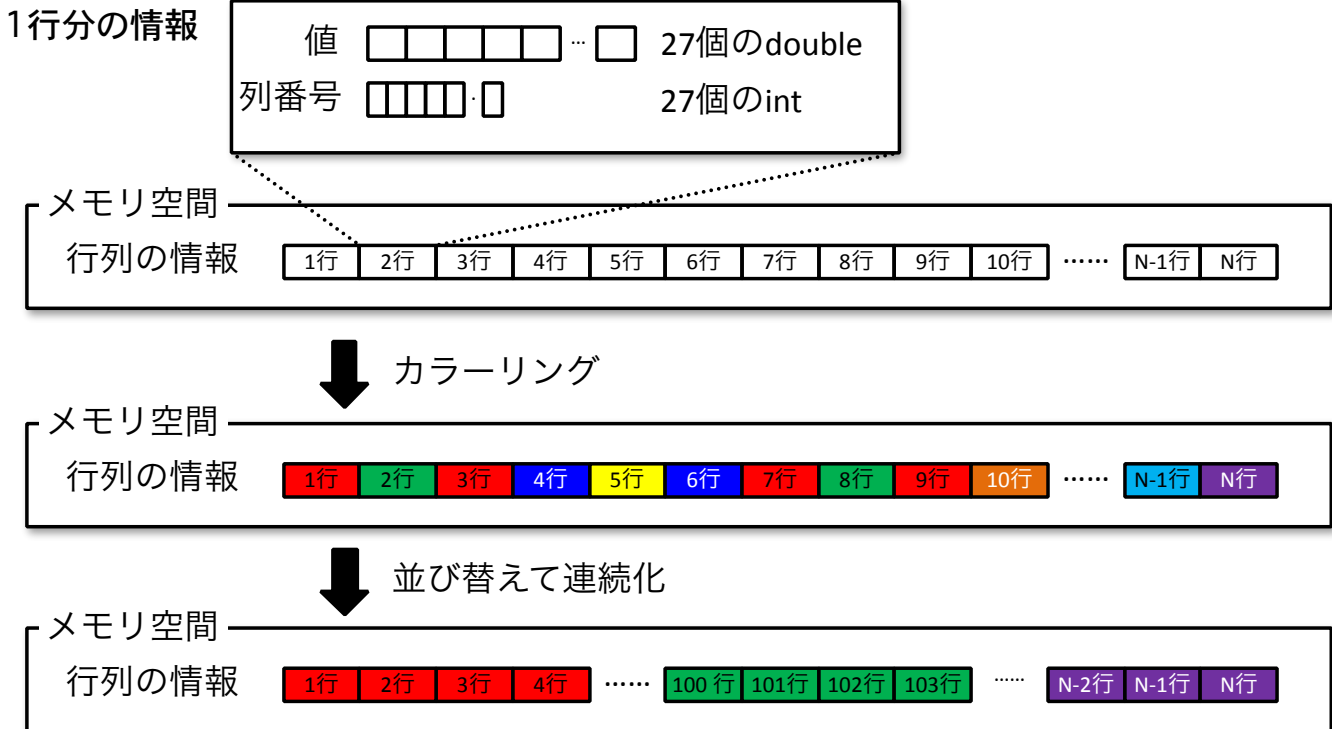
リカレンスがないのでディレクティブ  
挿入で並列化

```
#pragma omp parallel for
for(int i=st; i<=ed ++){
    最内ループ
}
```

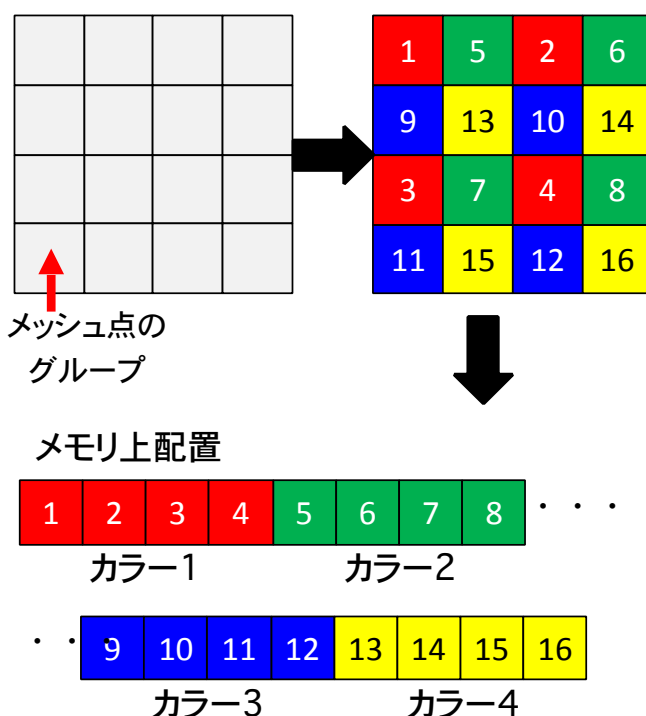
演算順序を変更しカラー毎に処理することで  
並列化が可能に

2014年8月26日SS研HPCフォーラム

演算順序の変更に伴い、メモリ上での再配置も行い、メモリアクセスを連続化

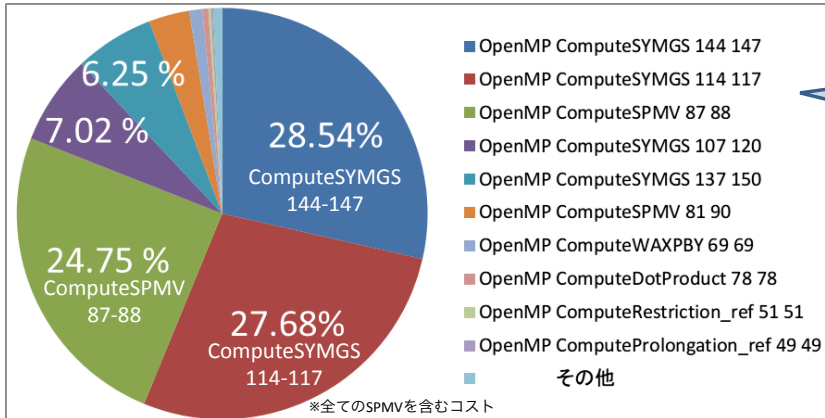


2014年8月26日SS研HPCフォーラム



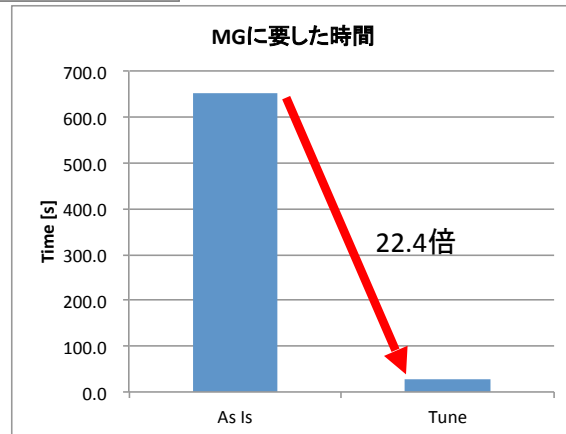
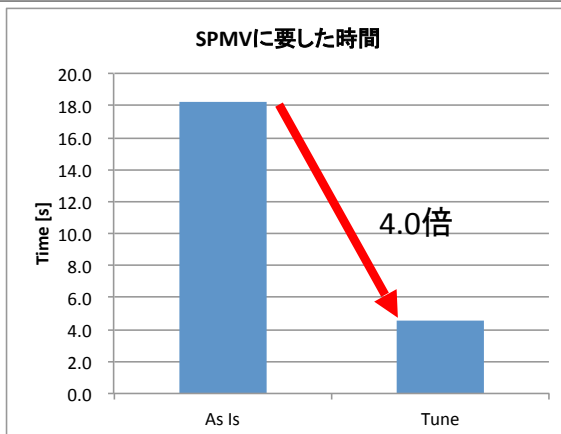
- ここで説明した8色カラーリングは問題の構造的なメッシュ構造を前提としたプログラミングを禁止するレギュレーションに抵触する.
- そのためもっと一般的なカラーリング手法に変更.
- メッシュ点を複数まとめてグループを構成する.
- そのグループ毎にカラーリングを実施.
- カラー内ではグループ間でリカレンスは発生しない.
- そこをスレッド並列に利用.

2014年8月26日SS研HPCフォーラム

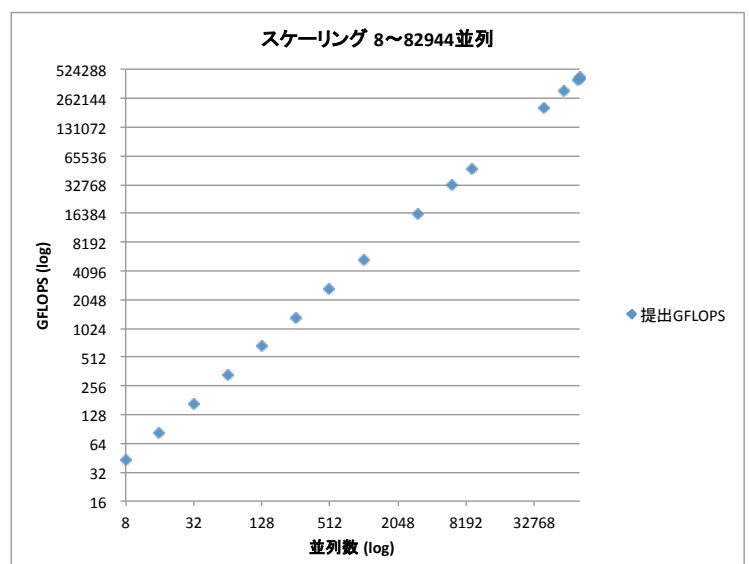
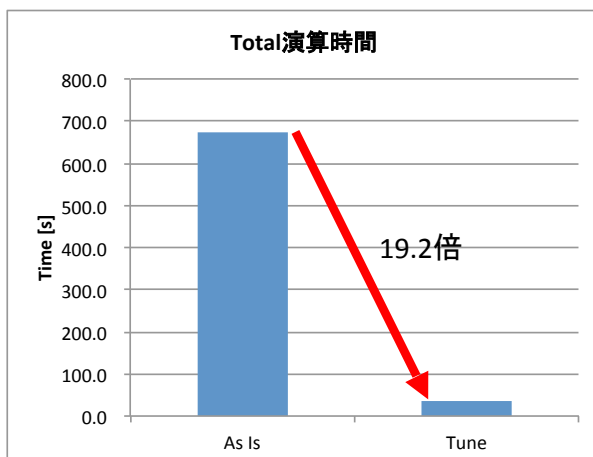


※ データサイズ104<sup>3</sup>→112<sup>3</sup>に変更

カラー化によりコスト上位の  
ループが全て並列化OK



2014年8月26日SS研HPCフォーラム



2014年8月26日SS研HPCフォーラム

天河2

「京」

Titan

Mira

Site	Computer	Cores	HPL Rmax (Pflops)	HPL Rank	HPCG (Pflops)	HPCG/ HPL
NSCC / Guangzhou	Tianhe-2 NUDT, Xeon 12C 2.26GHz + Intel Xeon Phi 57C + Custom	3,120,000	33.9	1	.580	1.7%
RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx 8C + Custom	705,024	10.5	4	.427	4.1%
DOE/OS Oak Ridge Nat Lab	Titan, Cray XK7 AMD 16C + Nvidia Kepler GPU 14C + Custom	560,640	17.6	2	.322	1.8%
DOE/OS Argonne Nat Lab	Mira BlueGene/Q, Power BQC 16C 1.60GHz + Custom	786,432	8.59	5	.101#	1.2%
Swiss CSCS	Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler 14C + Custom	115,984	6.27	6	.099	1.6%
Leibniz Rechenzentrum	SuperMUC, Intel 8C + IB	147,456	2.90	12	.0833	2.9%
CEA/TGCC-GENCI	Curie tme nodes Bullx B510 Intel Xeon 8C 2.7 GHz + IB	79,504	1.36	26	.0491	3.6%
Exploration and Production Eni S.p.A.	HPC2, Intel Xeon 10C 2.8 GHz + Nvidia Kepler 14C + IB	62,640	3.00	11	.0489	1.6%
DOE/OS L Berkeley Nat Lab	Edison Cray XC30, Intel Xeon 12C 2.46GHz + Custom	132,840	1.65	18	.0439 #	2.7%
Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	78,848	.881*	7	.0161	1.8%
Meteo France	Beaufix Bullx B710 Intel Xeon 12C 2.7 GHz + IB	24,192	.469 (.467*)	79	.0110	2.4%
Meteo France	Prolix Bullx B710 Intel Xeon 2.7 GHz 12C + IB	23,760	.464 (.415*)	80	.00998	2.4%
U of Toulouse	CALMIP Bullx DLC Intel Xeon 10C 2.8 GHz + IB	12,240	.255	184	.00725	2.8%
Cambridge U	Wilkes, Intel Xeon 6C 2.6 GHz + Nvidia Kepler 14C + IB	3584	.240	201	.00385	1.6%
TiTech	TUSBAME-KFC Intel Xeon 6C 2.1 GHz + IB	2720	.150	436	.00370	2.5%

HPL

HPCG

\* scaled to reflect the sam  
number of cores  
# unoptimized implement

2014年8月26日SS研HPCフォーラム

- ISC'14では正式のベンチマークではなく参考のミニリストとして発表
- 京はこのミニリストで2位を獲得
- 1位の天河2はピーク性能で京の5倍程度
- HPCGの性能差は1.3倍
- HPCG/HPL性能比: 4.1%は群を抜いている
- HPCGはISC'14で正式のベンチマークとなる予定

2014年8月26日SS研HPCフォーラム

- はじめに
- LINPACK
- HPCGとは
- HPCGベンチマークプログラム
- HPCGのチューニングと性能