科学技術計算分科会　選出
SS 研 HPC フォーラム 2014 より

新ベンチマークプログラム：HPCG の概要と「京」における性能

南 一生
(理化学研究所)

サイエンティフィック・システム研究会 2014 年 8 月 26 日
SS 研 HPC フォーラム 2014 資料

新ベンチマークプログラム：
HPCG の概要と「京」における性能

南　一生

独立行政法人　理化学研究所(計算科学研究機構)

[アブストラクト]

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 プロジェクトは、1993 年に発足したが、用いられているベンチマークプログラム：LINPACK が、浮動小数点演算性能の評価に重きを置いているために、近年、実アプリケーションとの乖離が指摘されることもあった。そこで、LINPACK の開発者である J. Dongarra 博士より、実アプリケーションで使われる計算手法の性能評価に重きを置いたベンチマーク・プログラム:HPCG が提案されている。2014年6月に開催された ISC14 において、正式なランキングではないものの、世界のトップクラスのスパコンを含む HPCG のランキングのミニリストが発表され、日本の「京」が2位に位置づけられた。本講演では、HPCG の概要および「京」における性能について報告する。

[キーワード]

スーパーコンピュータ、ベンクマーク、共役勾配法、チューニング、HPCG

1. はじめに

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。独立行政法人　理化学研究所と富士通で共同開発した、スーパーコンピュータ「京」は 2011 年6月の TOP500 で1位を獲得し、その後 2011 年 11 月の SC11 でも1位をキープした。2014 年の現時点では4位の位置につけている。

TOP500 プロジェクトは、1993 年に発足したが、用いられているベンチマークプログラム　:LINPACK は、密行列に対する並列の連立一次方程式の直接解法であり、行列・行列積の計算が主である。したがって LINPACK は、浮動小数点演算性能の評価に重きを置くことになり、近年、実アプリケーションとの乖離が指摘されることもあった。そこで、LINPACK の開発者である J. Dongarra 博士より、実アプリケーションで使われる計算手法の性能評価に重きを置いたベンチマーク・プログラム:HPCG が提案されている。

2. HPCG(High Performance Conjugate Gradient)とは

実アプリケーションで良く使われる連立一次方程式の解法として、疎行列に対する反復解法がある。その中でも前処理付き共役勾配法(PCG 法)は、一般的によく使われる手法である。HPCG は前処理として、ガウス・ザイデル法をベースとしたマルチグリッド法を採用している。

計算科学的な特長から言うと、HPCG は、疎行列・ベクトル積であり、アプリケーションが要求する B/F 値が大きいタイプの計算となる。そのために CPU 単体性能が出しにくいアプリケーションである。また、リストベクトルを使用するため、通常の要求 B/F 値が大きいアプリケーションより、さらに性能が出しにくい。この意味では、要求 B/F 値が小さい LINPACK とは、対極にあるベンチマークプログラムであると言える。

3. HPCG ベンチマークプログラムの概要

HPCG ベンチマークプログラムは非構造格子を扱うことができるプログラムとなっているが、実際のベンチマークデータは、構造格子である。プログラム内で構造格子をリストアクセスすることにより、非構造格子のように処理している。ベンチマークのセットアップ段階で、このようなメッシュの作成をプログラムの内部で行っている。次の段階で、プログラムのチューニングや行列の最適化に問題がないか検証を行う。計測フェーズの最初に、オリジナルの処理とチューニング後の処理の実行時間の差を検証し、計測のためのパラメタを決定している。その後、計測フェーズを実施する。最後に、レポート出力フェーズを実施する。ここで、検証フェーズに問題があった場合は、不合格となる。

4. HPCG のチューニングと性能

まず、HPCG ベンチマークプログラムが大規模並列に対応可能なようにプログラムの修正を実施した。その後デフォルトで設定されている問題を使用し 32000 ノードまでのウィークスケーリング測定とコスト分析を実施した。ウィークスケーリングについては良好であり、また、通信時間は6%程度であった。さらにマルチグリッド前処理と行列・ベクトル積の処理のコストが大きいことが分かった。特にマルチグリッド前処理については、ガウスザイデル法をベースとしているため、オリジナルコードのままではスレッド並列化ができない。そこで今回は節点データをカラーリングにより、リオーダリングすることでスレッド並列に対応した。その結果、CPU 単体性能として、オリジナルプログラムの 19.2 倍、フルノードまでの良好な並列性能を達成した。

上記の結果、2014年6月に開催された ISC14 において、正式なランキングではないものの、世界のトップクラスのスパコンを含む HPCG のランキングのミニリストが発表され、0.427Pflops の性能で日本の「京」が2位に位置づけられた。「京」が達成した性能は、他のスパコンとピーク性能比で比較すると、群を抜いている。これは、「京」の良好な性能バランスとアプリケーションのチューニングの努力によるものである。

[参考文献]

(1) Hasegawa, Y., Iwata, J.I., Tsuji, M., Takahashi, D., Oshiyama, A.,Minami, K., Boku, T., Shoji, F., Uno, A., Kurokawa, M., Inoue, H.,Miyoshi, I., Yokokawa, M.: “First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the K computer.”, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1:1-1:11. SC ’11, ACM, New York, NY, USA (2011)

(2) 黒田 明義, 長谷川 幸弘, 寺井 優晃, 井上 俊介, 市川 真一, 小松 秀実, 大井 憲行, 安藤 琢也, 山崎 隆浩, 大野 隆央, 南 一生. :″ナノ材料第一原理分子動力学プログラム PHASE の京速コンピュータ「京」上の計算性能最適化″, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp.144-152 (2012)

(3) 南 一生, 井上 俊介, 堤 重信, 前田 拓人, 長谷川 幸弘, 黒田 明義, 寺井 優晃, 横川 三津夫.:″「京」コンピュータにおける疎行列とベクトル積の性能チューニングと性能評価″ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp.23-31 (2012)

(4) Kiyoshi Kumahata, Shunsuke Inoue, Kazuo Minami.: Kernel performance improvement for the FEM-based fluid analysis code on the K computer,Procedia Computer Science,Volume 18, 2013, Pages 2496-2499,2013 International Conference on Computational Science

(5) 井上俊介, 堤重信,前田拓人,南一生.:スーパーコンピュータ「京」におけるメモリインテンシブな アプリケーションの評価および高性能化,先進的計算基盤システムシンポジウム SACSIS2013 論文集,pp,123(2013)

(6) Masaaki Terai, Eiji Tomiyama, Hitoshi Murai, Kazuo Minami and Mitsuo Yokokawa.: ″K-scope: a Java-based Fortran Source Code Analyzer with Graphical User Interface for Performance Improvement″, Third International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI2012).

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

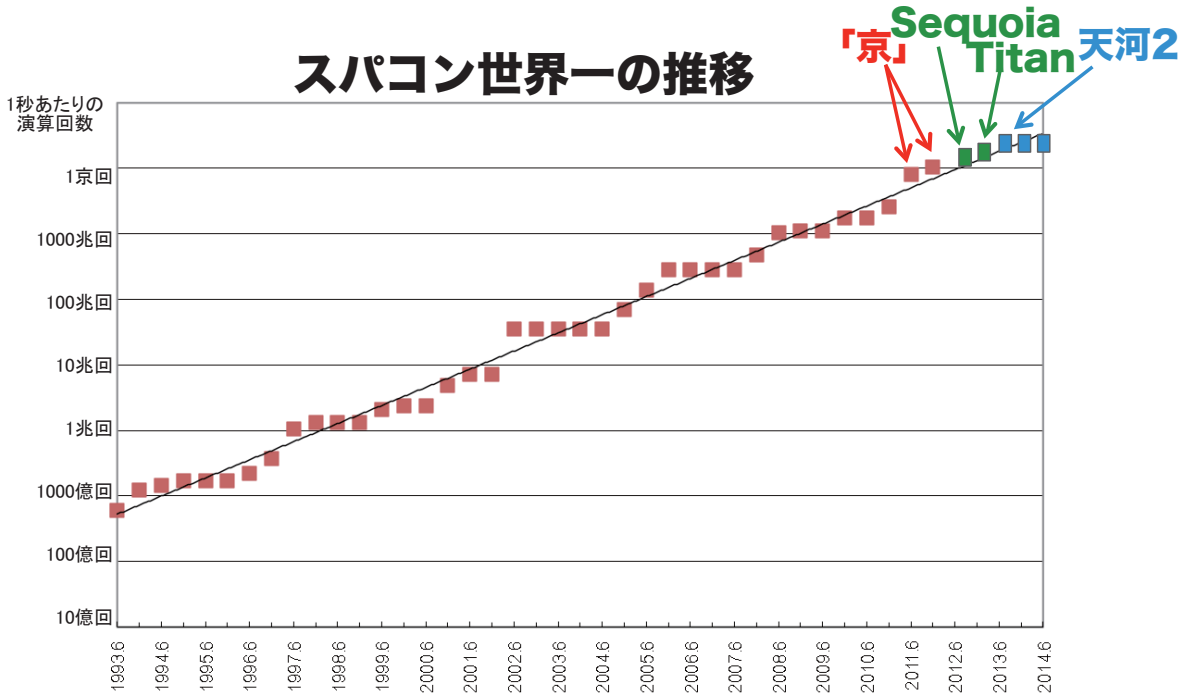
南一生

OUTLINE

- 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

スパコン世界一をめぐる競争



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！

ちなみに

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

≒

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

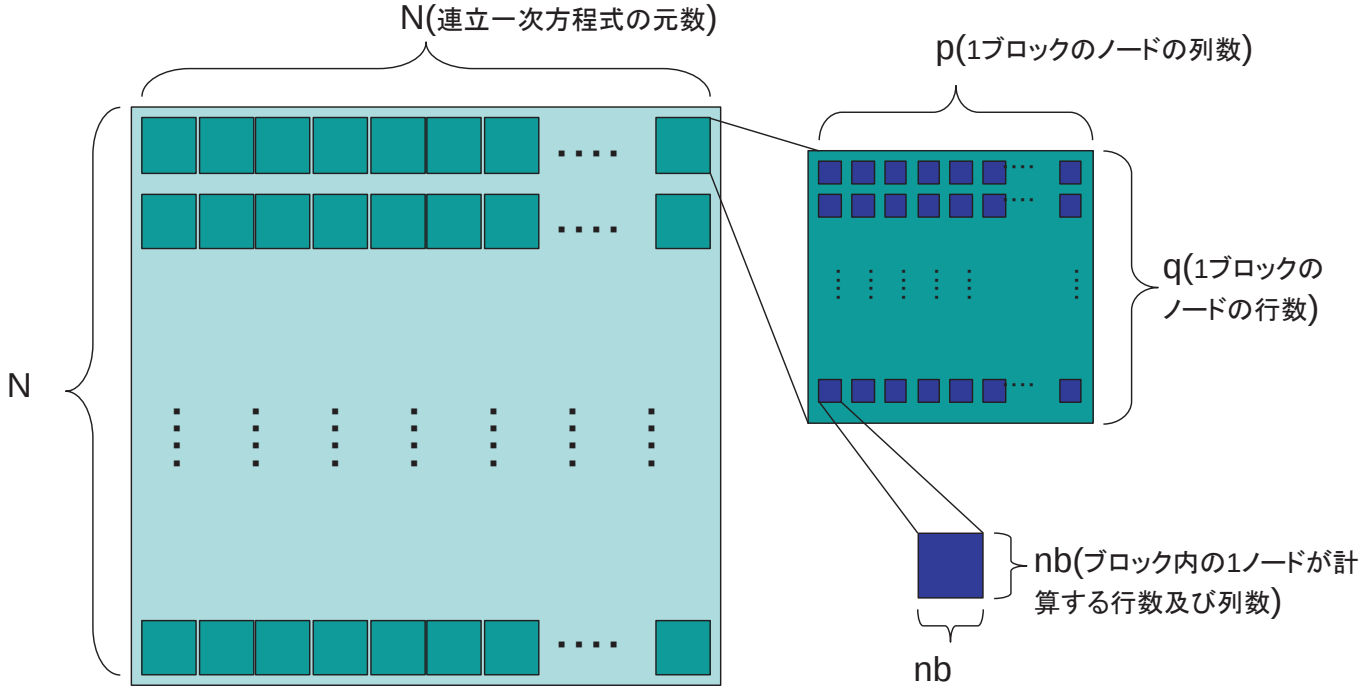
LINPACK

High Performance LINPACK(HPL)

- 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

HPLのプロセス分割

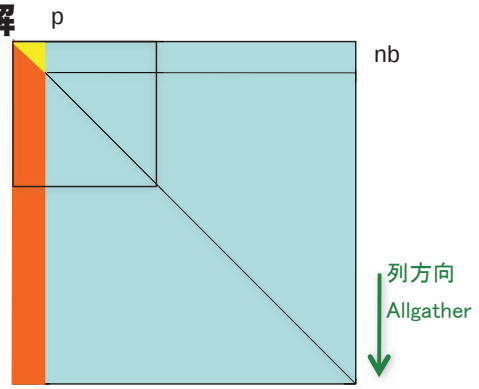
「京」の例



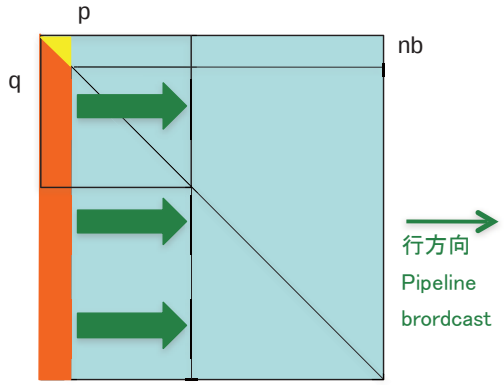
HPLの計算手順

(1)Lパネル分解

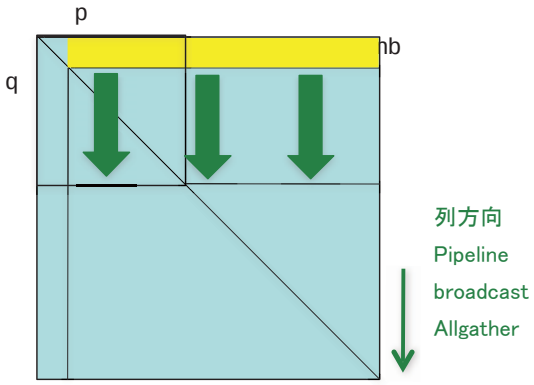
DTRSM
(三角行列計算)



(2) 分解済みLパネル放送

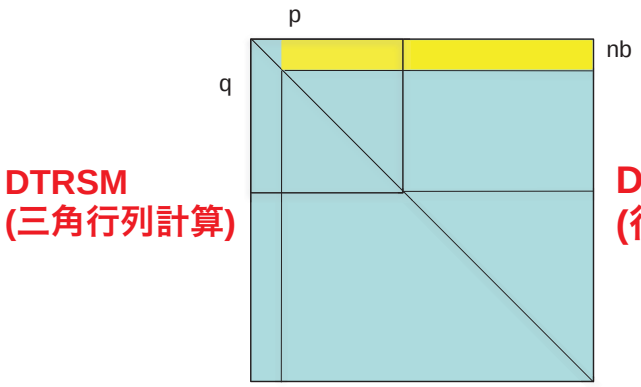


(3)Uブロック行放送とPivot行交換

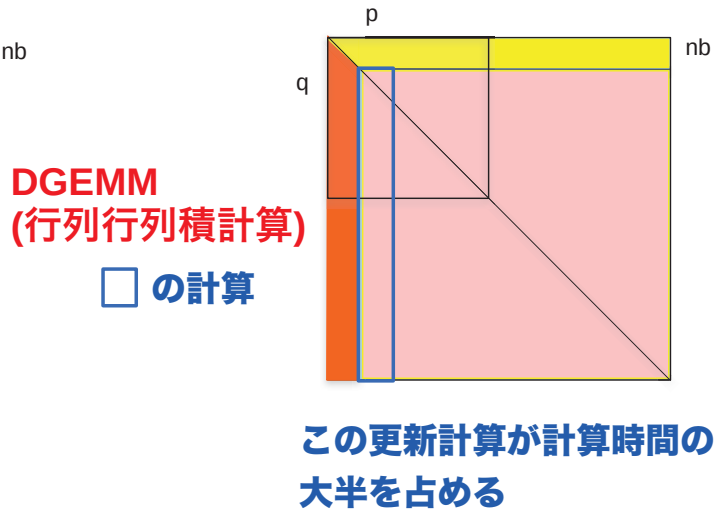


HPLの計算手順

(4) Uブロック分解



(5) 列パネル更新



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

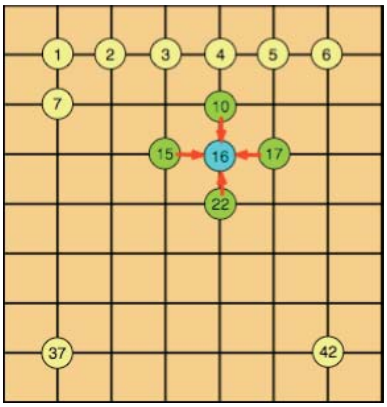
HPCGとは

HPCGベンチマークの提案

(*) 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

連立一次方程式と共役勾配法(CG法)



- 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.

$$\mathbf{Ax} = \mathbf{b}$$

$$f(x) = \frac{1}{2}ax^2 - bx$$

f(x)の最小値問題

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

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

$$f'(x) = ax - b = 0$$

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)

$$r_0 = b - A x_0$$

$$p_0 = r_0$$

for $i = 0, 1, 2, \cdots$

$$\alpha_i = \frac{(r_i, r_i)}{(p_i, A p_i)}$$

$$x_{i+1} = x_i + \alpha_i p_i$$

$$r_{i+1} = r_i - \alpha_i A p_i$$

$$\beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)}$$

$$p_{i+1} = r_{i+1} + \beta_i p_i$$

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

- 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(x)が最小値が得られるようにxiの列を求める.
- ri 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.

前処理付き共役勾配法(CG法)

$$\mathbf{M} = \mathbf{U}^T\mathbf{U}$$

Aに近い正値対称行列:Mを考えコレスキー分解する.

$$\left.\begin{aligned} \mathbf{U}\mathbf{x} &= \tilde{\mathbf{x}} \\ \mathbf{U}^{-T}\mathbf{b} &= \tilde{\mathbf{b}} \\ \mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} &= \tilde{\mathbf{A}} \end{aligned}\right\}$$

このようなx,b,Aの置き換えを行う.

$$\tilde{\mathbf{A}} = \mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} \approx \mathbf{I}$$

置き換えたAの性質は単位行列に近い.
置き換えた行列Aの固有値は1の周りに密集している

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$$

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

前処理付き共役勾配法(CG法)

- 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: $x_i^{k+1} = x_i^k + \alpha^k p_i^k$

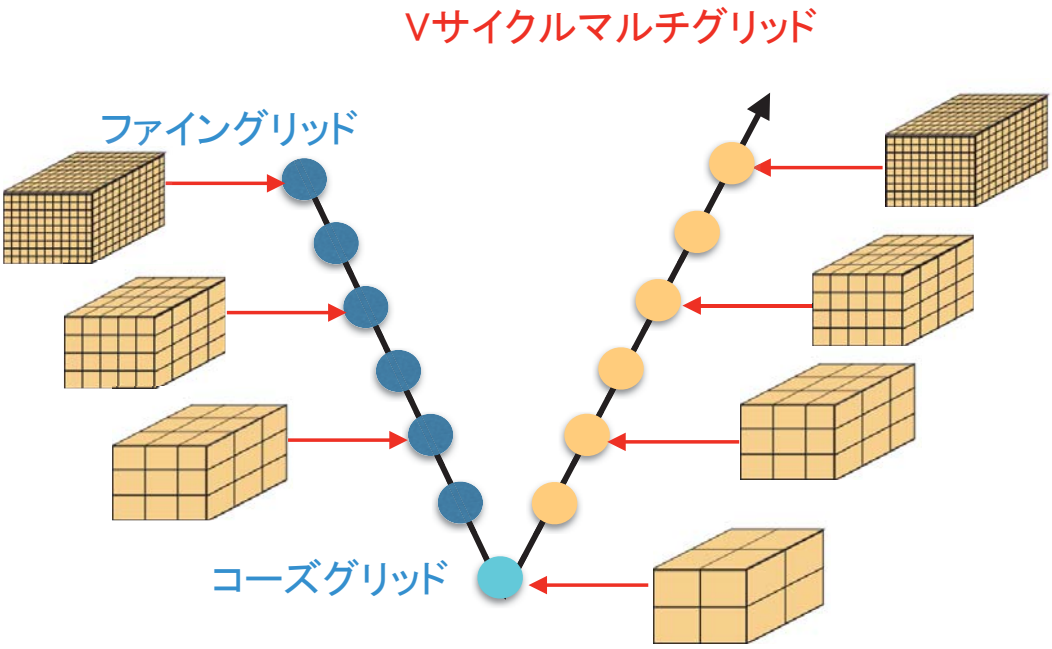
ステップ3: $r_i^{k+1} = r_i^k - \alpha^k Ap_i^k$

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

前処理としてのマルチグリッド法

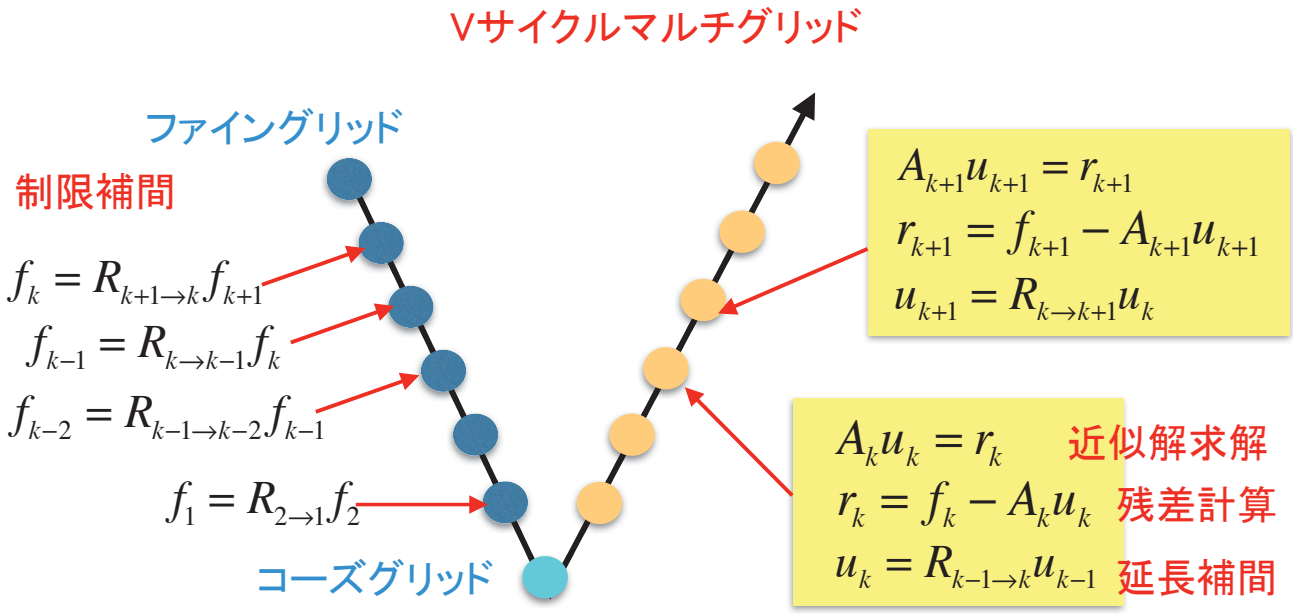
連立一次方程式の解法としてのマルチグリッド法の例



前処理としてのマルチグリッド法

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



LINPACKとHPCGの単体性能比較

High Performance

A Processor Performance

1	行列行列積に書き換え可能
2	キャッシュブロッキング可能
3	要求B/F値が低くループボディがシンプル
4	要求B/F値が低いがループボディが複雑
5	要求B/F値が高い
6	要求B/F値が高くリストアクセスを使用

Low Performance

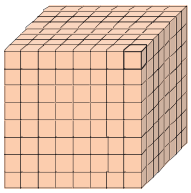
LINPACK:
大規模密行列の連立一次方程式の並列直接解法.
行列行列積の計算が中心.

HPCG:
大規模疎行列の連立一次方程式の並列反復解法.
疎行列ベクトル積が中心.

HPCGベンチマークプログラム

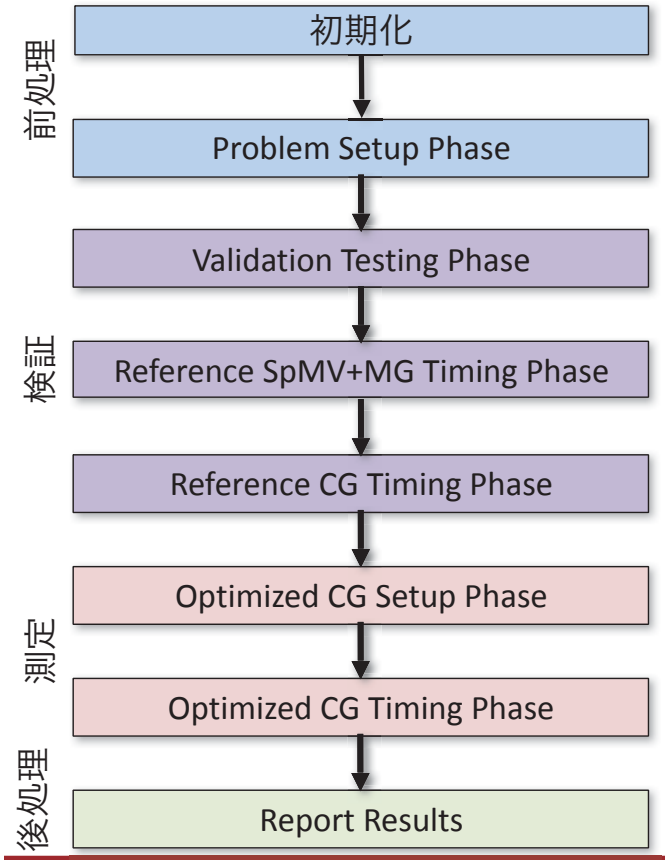
HPCGベンチマーク

- 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

HPCGベンチマークアプリの概要



- 大まかに4つの部分に分けられる

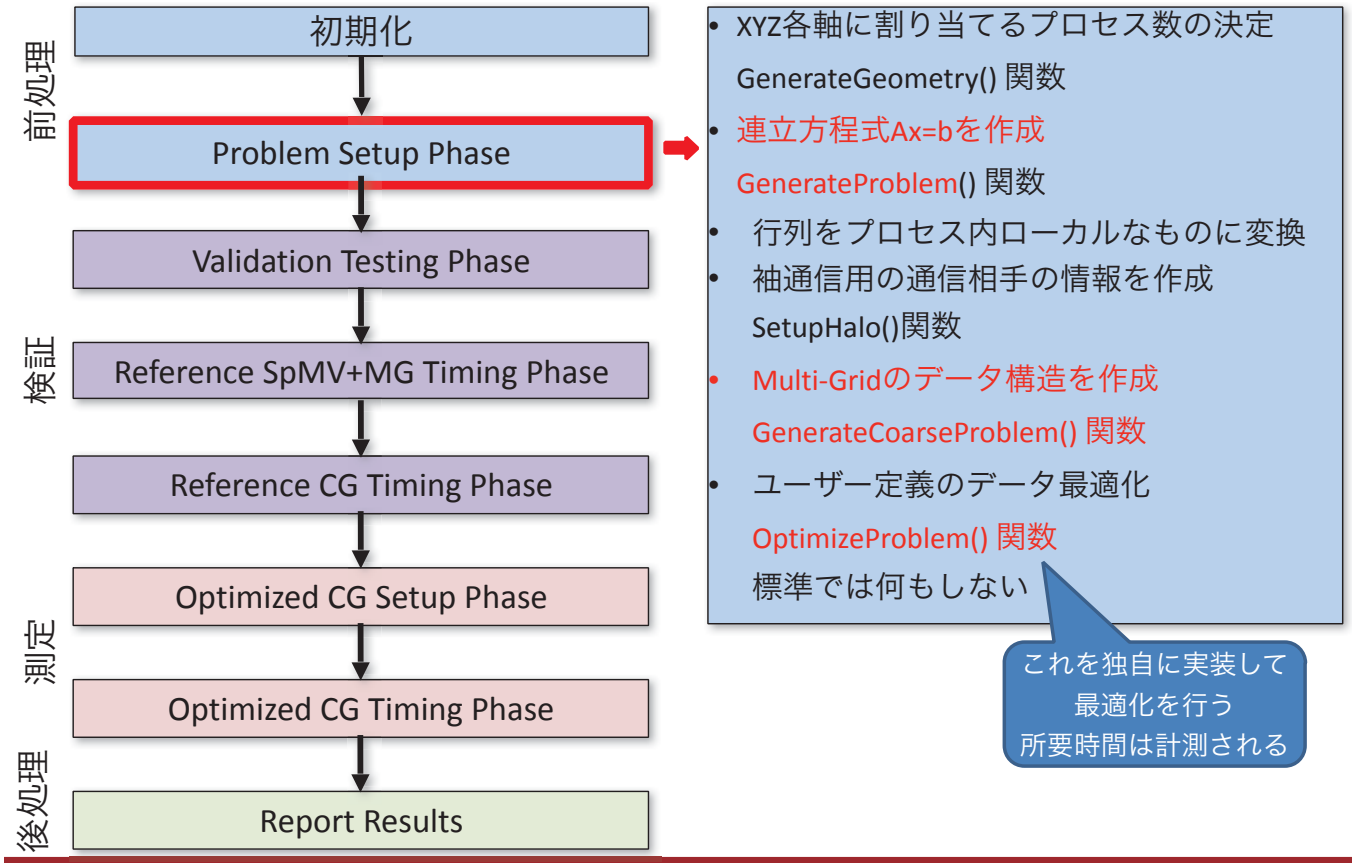
前処理
- メモリ確保などの初期化
- 問題設定,行列の最適化

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)

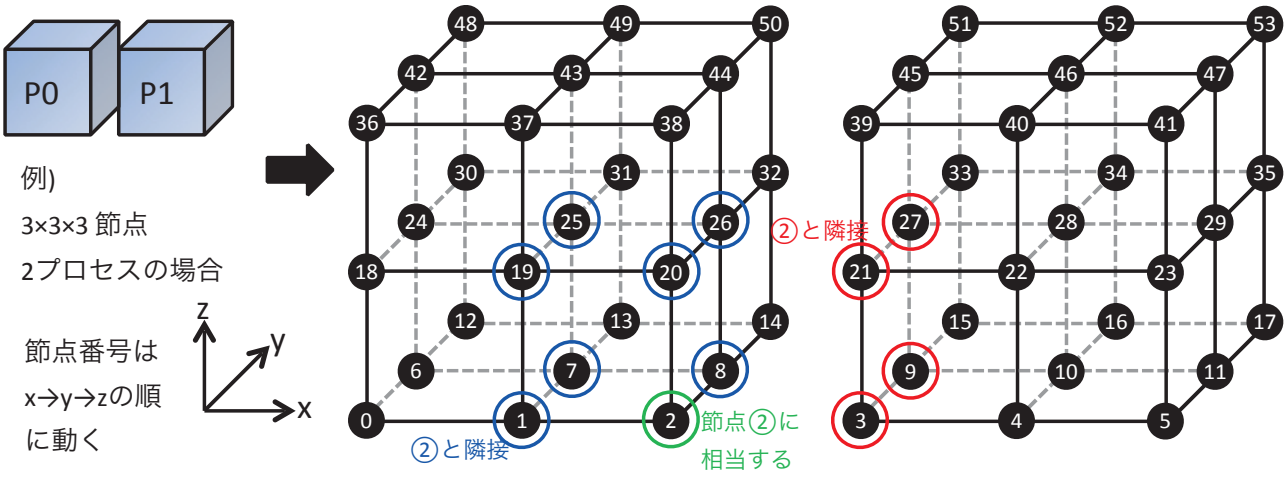
ここでは各部分の主要な処理について紹介

HPCGベンチマークアプリの概要



GenerateProblem 関数の処理

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)



②行の内容

列番号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	···	53
値		−1	26	−1				−1	−1	−1										−1	−1	−1				−1	−1	−1				

赤字は外部節点への参照

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		···		

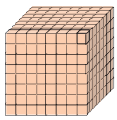
非ゼロ個数は 27個/行 に固定
global_int_t型(実体はint)
doulbe型

GenerateCoarseProblemの処理

MG用の粗いメッシュに対応する係数行列を作成

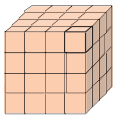
粗い格子は3レベル作成して
合計4レベルの格子

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

① Generate Geometry
↓
② Generate Problem
↓
③ SetupHalo
↓
④ Generate CoarseProblem

行列データの構造体

格子の情報
- XYZの節点数
- XYZのプロセス形状
- etc

行列の情報
- 行数，行毎の非ゼロ数
- 列番号，非ゼロ値
- etc

袖通信の情報
- 隣接数
- 隣接相手
- 送信節点情報

粗い行列データへのポインタ
細←→粗間の節点対応表

① Generate Geometry
↓
② Generate Problem
↓
③ SetupHalo
↓
④ Generate CoarseProblem

行列データの構造体

格子の情報
- XYZの節点数
- XYZのプロセス形状
- etc

行列の情報
- 行数，行毎の非ゼロ数
- 列番号，非ゼロ値
- etc

袖通信の情報
- 隣接数
- 隣接相手
- 送信節点情報

粗い行列データへのポインタ
細←→粗間の節点対応表

HPCGベンチマークアプリの概要

前処理
- 初期化
- Problem Setup Phase

検証
- Validation Testing Phase
- Reference SpMV+MG Timing Phase
- Reference CG Timing Phase

測定
- Optimized CG Setup Phase
- Optimized CG Timing Phase

後処理
- Report Results

演算関数は
2種類存在

基準版 ComputeCG_ref()
↑ 標準では単にラッパー
チューニング版 ComputeCG()

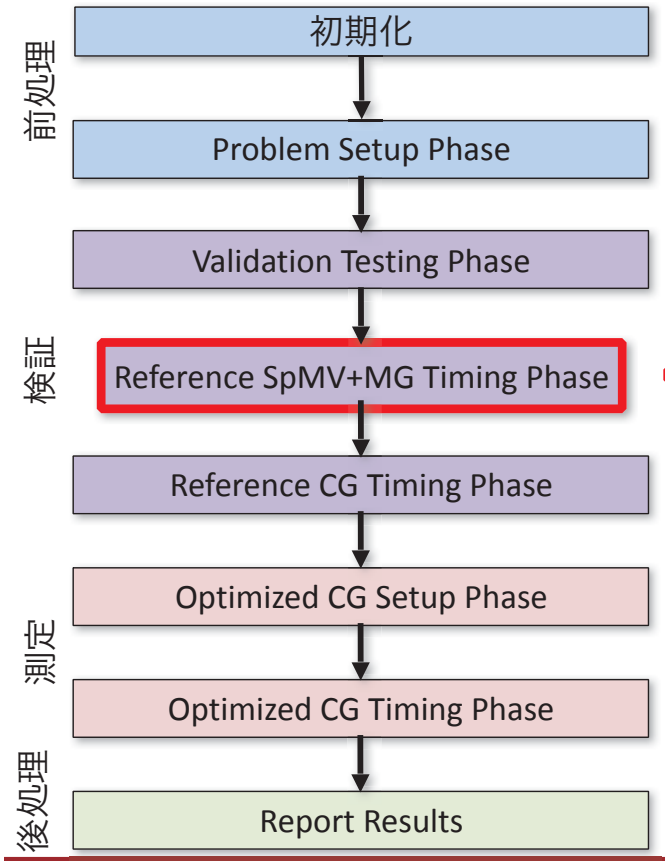
カーネルのチューニングや行列の最適化が正しいかをチェック

TestCG()
- チューニング版CG関数に,強く対角優位な行列を与えて,規程回数で収束するかを判定(収束判定値1.0×10^{-12})
  - 前処理なしなら12回以内
  - 前処理ありなら2回以内

  条件は決め打ち

TestSymmetry()
- OptimizeProblem()で最適化した行列が対称になっているかチェック

HPCGベンチマークアプリの概要



基準版のSPMV(疎行列ベクトル積)とMG(Multi-Grid)関数の実行時間を計測

ComputeSPMV_ref()
- 基準版の疎行列ベクトル積

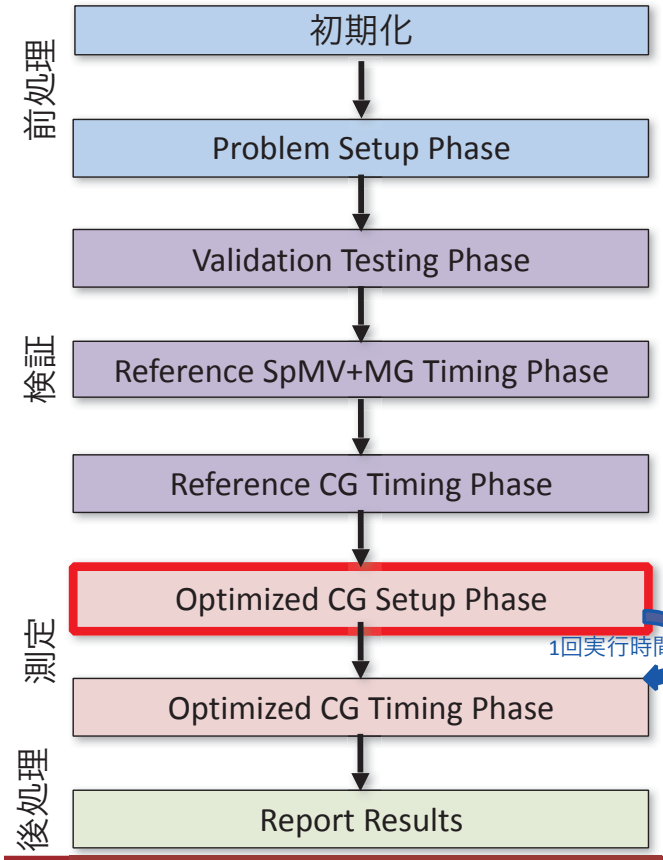
ComputeMG_ref()
- 基準版のMG

10回

この時のSPMVとMGの1回当りの平均時間を求める

Report Result部で
OptimizeProblem()の時間と,この平均時間との比
を出力している

HPCGベンチマークアプリの概要



チューニング版CGを1回実行し,要した時間を測定

CG()
- チューニング版CG
- 反復上限は500回
  収束判定はCG_ref()を実行して求めた
  ref_tolerance(1.0E-3程度)
- 1回だけ実行

この時間を元に,次のフェーズでCG()を何回実行するかを決定

HPCGベンチマークアプリの概要

前処理
- 初期化
- Problem Setup Phase

検証
- Validation Testing Phase
- Reference SpMV+MG Timing Phase
- Reference CG Timing Phase

測定
- Optimized CG Setup Phase
- Optimized CG Timing Phase（1回実行時間）

後処理
- Report Results

本番測定
チューニング版CGを複数回実行し測定

複数回

解は実行の度にゼロクリア

CG()
- チューニング版CG
- 反復上限は先の1回実行の反復数(収束判定は0なので必ず指定回数反復する)
- 合計実行回数は,先に計った1回の実行時間を元に,入力データで指定した時間となるよう決定

提出は1時間以上必要

TestNorms()
- 再現性・安定性のチェック
- CGの実行毎に $\|\vec{r}_{50}\|_2 / \|\vec{r}_0\|_2$ の値を記録しておき,ブレがないかチェック

分散が1.0×10^{-6}以下なら合格

MGの処理

$\vec{z} = \mathrm{MG}(A, \vec{r})$

最上位を含む途中のレベル

$\vec{z} = 0$

Relaxation $A\vec{z} = \vec{r}$ by Gauss-Seidel (Pre-smoothing)
袖通信

$\overrightarrow{Az} = A \cdot \vec{z}$ 袖通信

$\vec{r}_{coarse} = \mathrm{Restrict}(\vec{r} - \overrightarrow{Az})$

細かい格子での値を粗い格子上に補間

$\vec{z}_{coarse} = \mathrm{MG}(A_{coarse}, \vec{r}_{coarse})$ (再帰呼び出し)

$\vec{z} = \vec{z} + \mathrm{Prolongate}(\vec{z}_{coarse})$

粗い格子での値を細かい格子上に補間

Relaxation $A\vec{z} = \vec{r}$ by Gauss-Seidel (Post-smoothing)
袖通信

最も粗いレベル

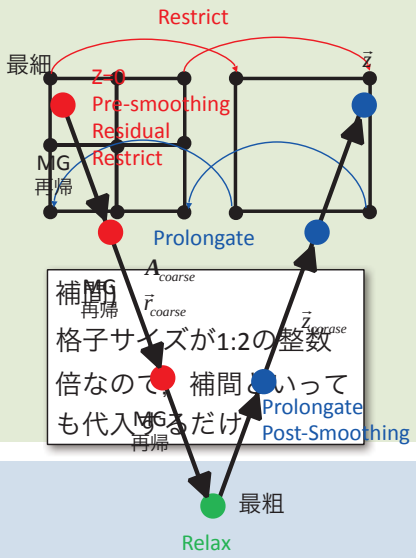
Relaxation $A\vec{z} = \vec{r}$ by Gauss-Seidel
袖通信

前進Sweep $k = 1 \sim N$

$$z_i^{(k+\frac{1}{2})} = \frac{1}{a_{ii}}\left\{ r_i - \sum_{j<i}^{N} a_{ij} z_j^{(k+\frac{1}{2})} - \sum_{j>i}^{N} a_{ij} z_j^{(k)} \right\}$$

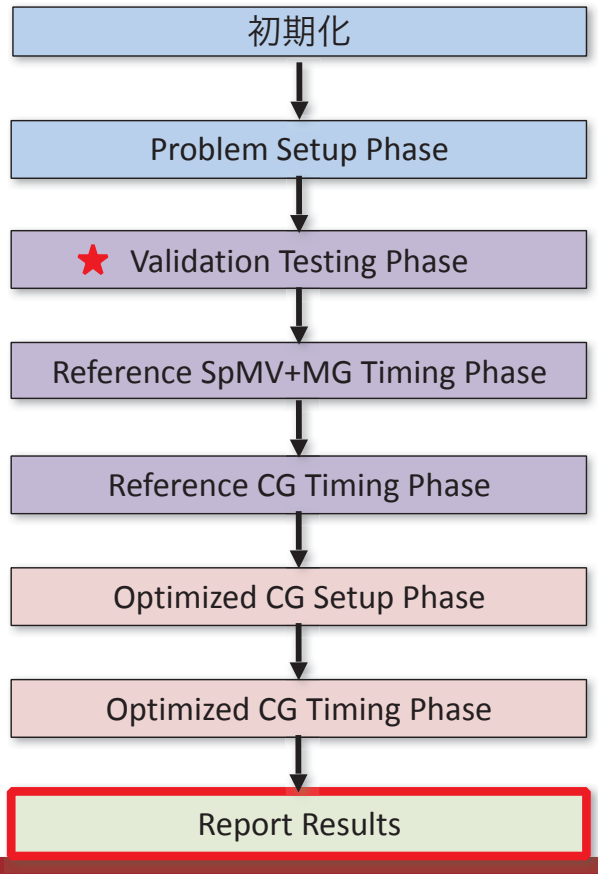
後退Sweep $k=N \sim 1$

$$z_i^{(k+1)} = \frac{1}{a_{ii}}\left\{ r_i - \sum_{j<i}^{N} a_{ij} z_j^{(k+\frac{1}{2})} - \sum_{j>i}^{N} a_{ij} z_j^{(k+1)} \right\}$$



補間
格子サイズが1:2の整数倍なので,補間といっても代入するだけ

HPCGベンチマークアプリの概要



提出する実行結果を出力
(HPCG-Benchmark-2.1-年月日時刻.yaml)

- 並列数・スレッド数
- 全体問題サイズ
- プロセス形状
- 1ノード当りの問題サイズ
- マルチグリッド各階層での行列の行数と非ゼロ数
- チューニングの正当性検証結果
- 再現性チェック結果
- 測定結果(各カーネルと全体)
  - Elapse時間
  - 理論的な浮動小数点演算数
  - GFLOPS値

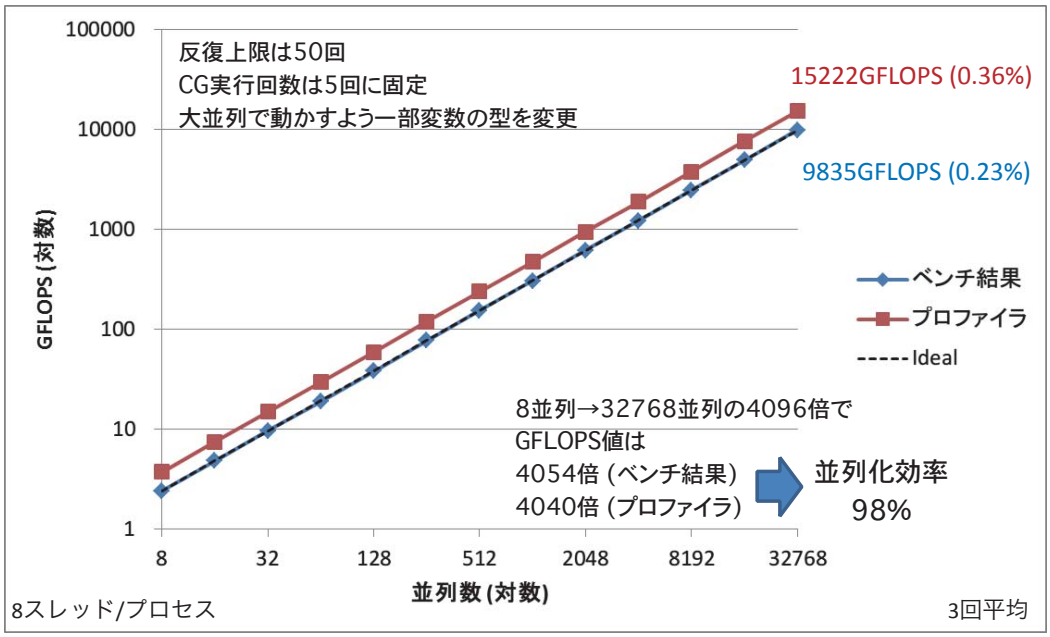
検証フェーズ★で不合格があればFAILED, 正常ならVALID+GFLOPS値が出力される

HPCGのチューニングと性能

HPCG テスト実行 on 京

- 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104×104×104=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

測定値のプロファイラとの違いについて

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

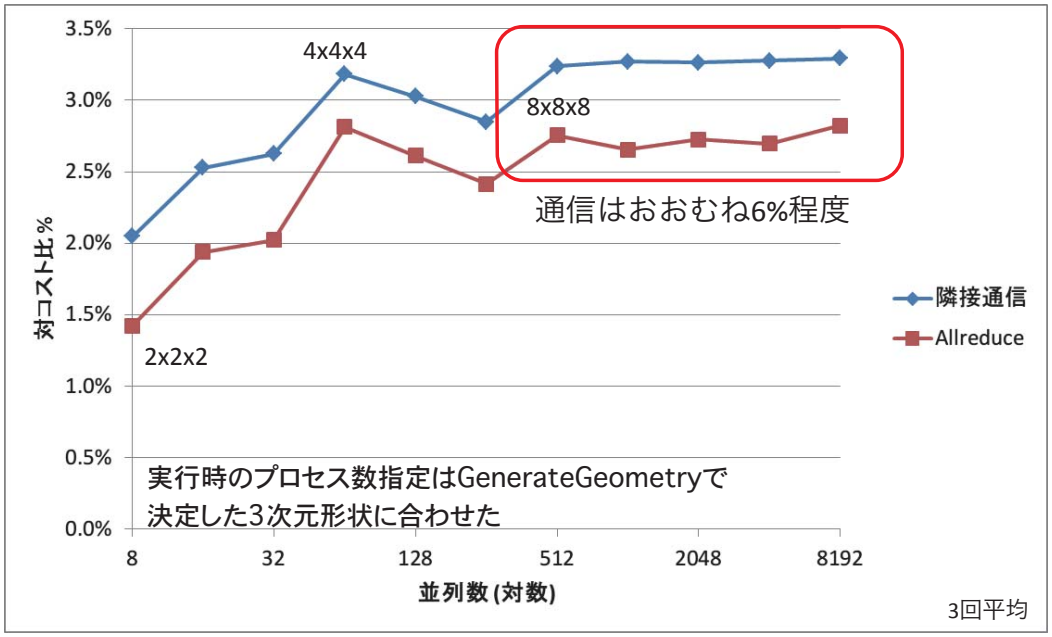
$\vec{x} = \mathrm{CG}(\boldsymbol{A}, \vec{b}, \vec{x}_0, \varepsilon, \max)$

```
SPMV()
WAXPBY()                    反復の前処理の分は
DotProduct                  カウントしない
                            ※V2.4ではカウントするようになった
for(k=1; k<max && !conv; k++){
    MG()
    DotProduct()
    WAXPBY()                反復内の分だけ
    SPMV()                  カウントする
    ...
}
```

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

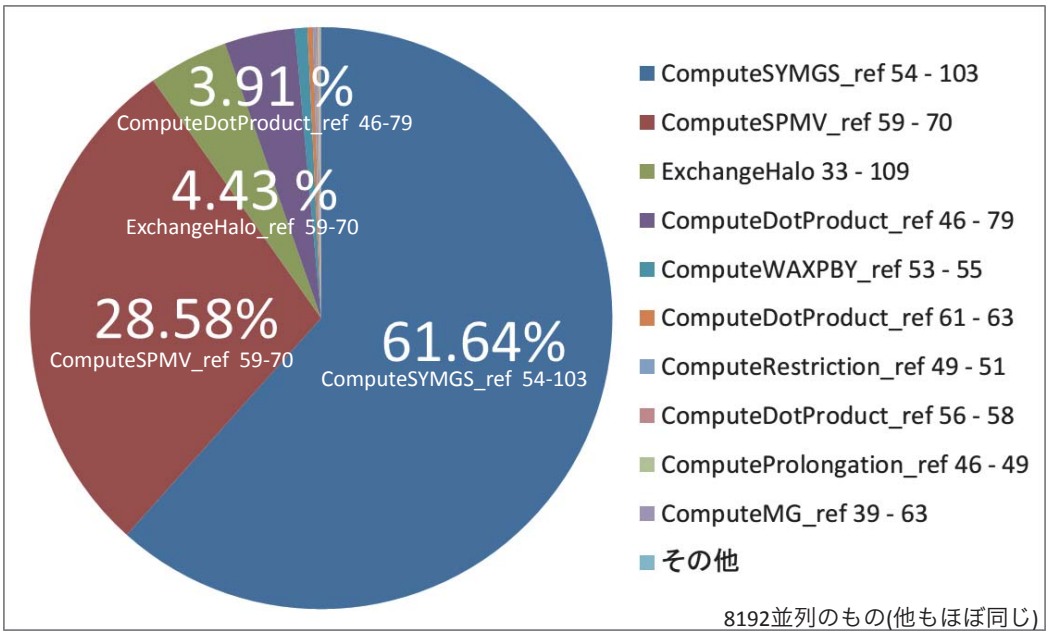
```
for(i=0; i<行数; i++){
    ...
    for(j=0; j<nz[i]; j++){
        col = A.mtxIndL[i][j]
カウントする  sum -= val[j] * x[col];
    }
    sum += x[i] * Diag[i];
    sum = sum / Diag[i];
}   カウントしない
```


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

HPCGコスト分析



- 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, Isend, Irecv, 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

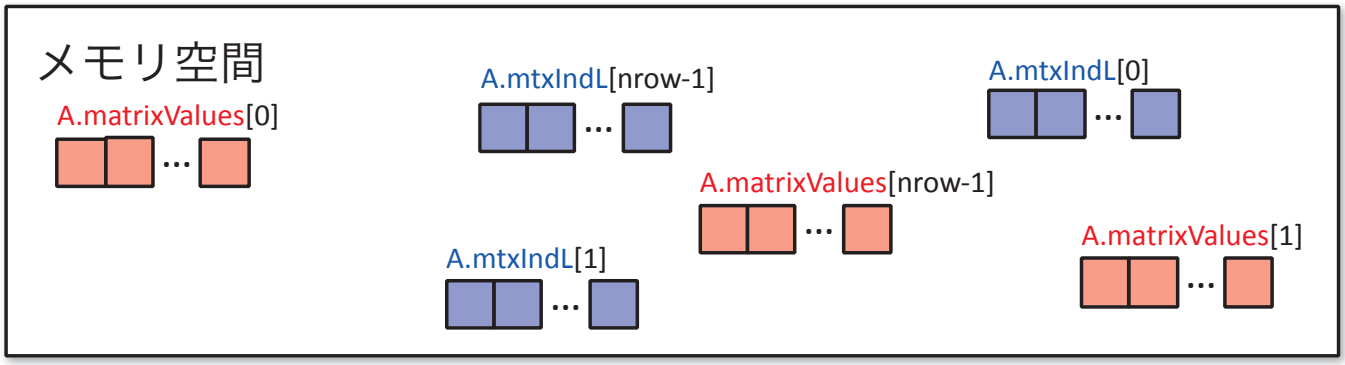
単体性能チューニング

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



単体性能チューニング

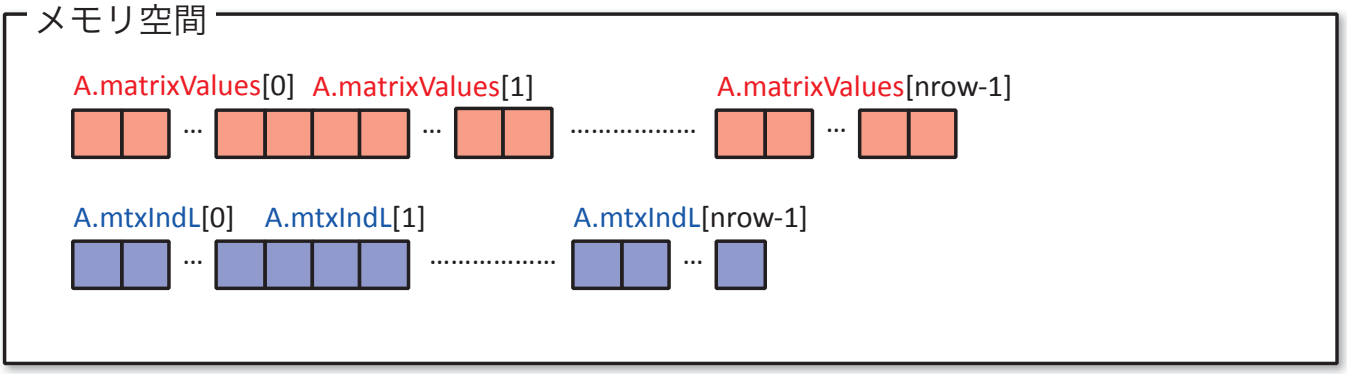
メモリ領域の連続化

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]);
}
```

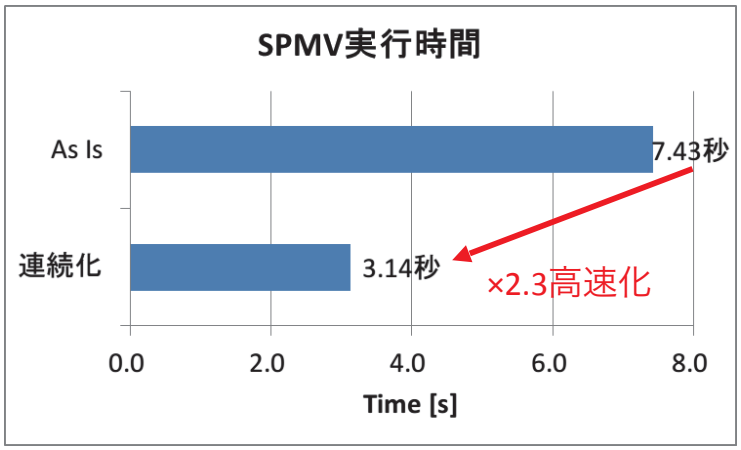
メモリ確保を一括して行い，各行の情報が連続するように変更

ムダな領域もなくなった



単体性能チューニング

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%



8プロセス，8スレッド/プロセス

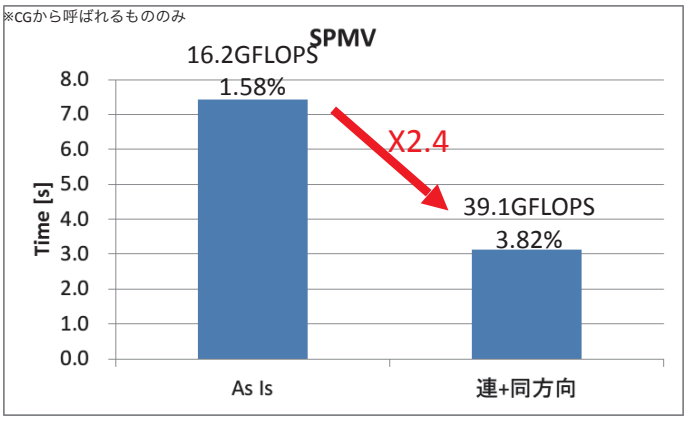
CG()から直接呼ばれるSPMVについて，実行時間と,キャッシュミス率,スループット等を調べた

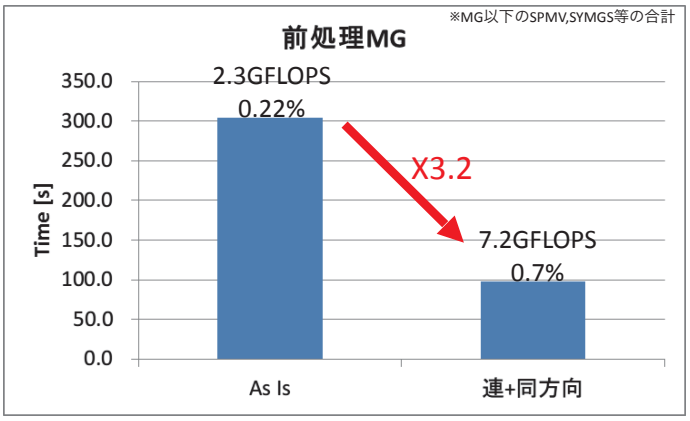
※ MG()から呼ばれる分は除外

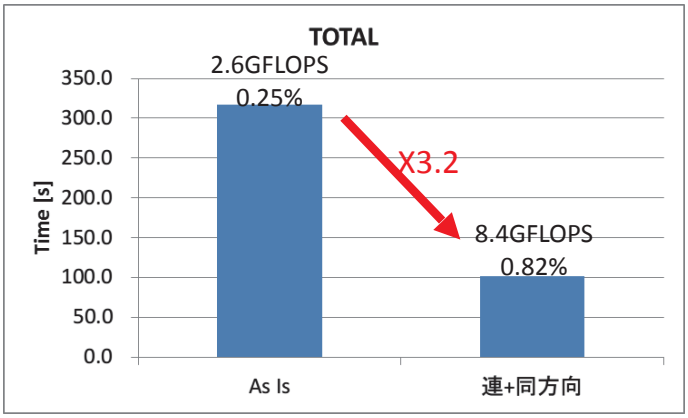
メモリ領域の連続化の効果によりキャッシュミス率減少,スループット向上

実行時間は1/2.3に短縮

その他のチューニングを実施

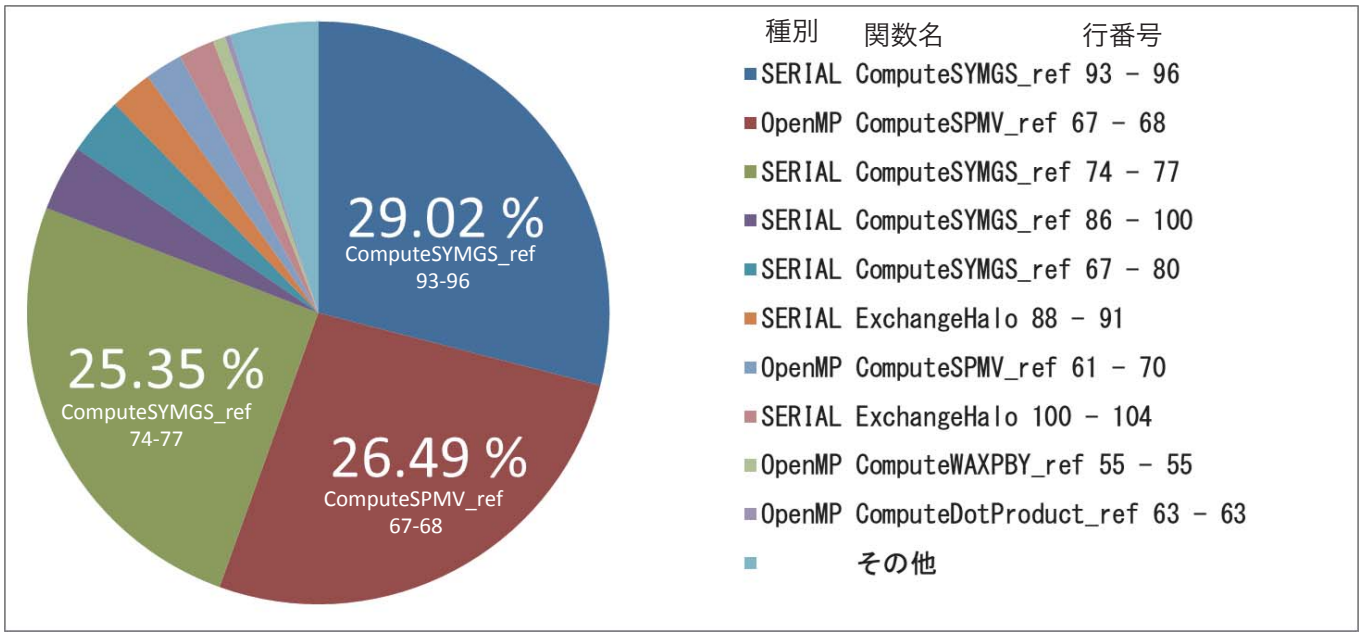






- HPCG自身が測定した提出するデータから抜粋したもの
  - SPMVの時間CG()から呼ばれる分のみの時間
  - 前処理MGはSPMV,SYMGS等々を含む時間
- 2点の簡単なチューニングだけで3.2倍の高速化が得られた
  - 行列のメモリ領域の連続化
  - SYMGSのBAKループでのループ方向の反転

前処理のマルチスレッド化



- 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

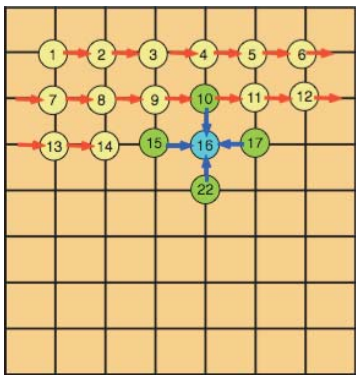
↓

- ここではComputeSYMGSのマルチスレッド化を実施

カラーリング

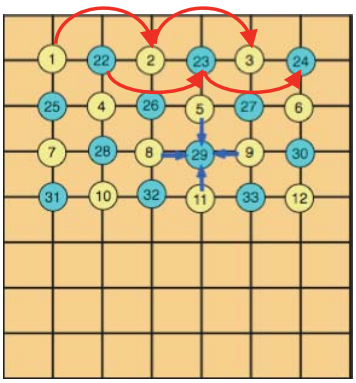
- SYMGSは対称行列についてガウス・ザイデル法を使用したマルチグリッド処理を行う.
- ガウス・ザイデル法 $a_{ii}x_i^{(m+1)} = -\sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=1+1}^{n} a_{ij}x_j^{(m)} + k_j$
- (m+1)世代の解を得るために(m+1)世代の下三角行列を使用して(リカレンスの発生).

(a)オリジナル

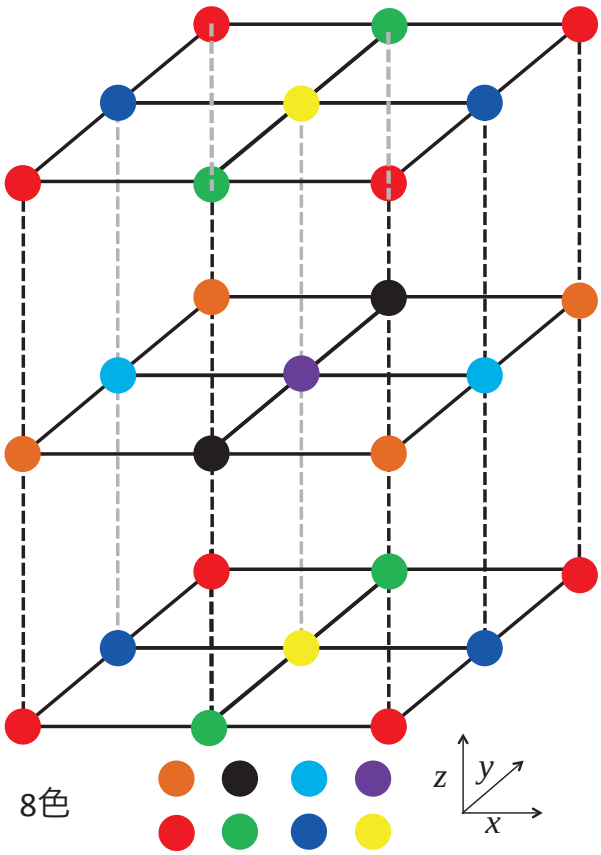


参照関係
計算順序
(スイープ順序)

(c)RED-BLACK２色カラーリング



カラーリング



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
}
```

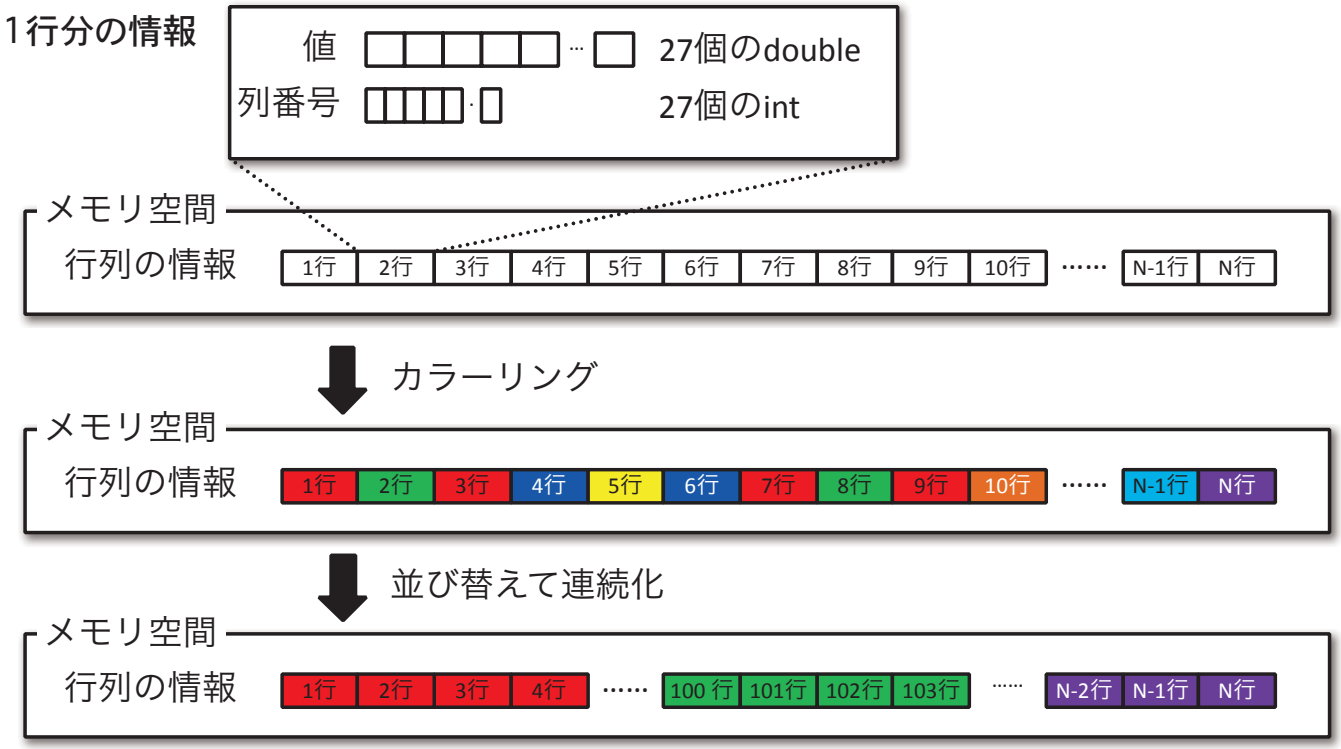
カラーを回す外側ループ追加で3重化

```
for(int ic=0; i<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

    }
}
```

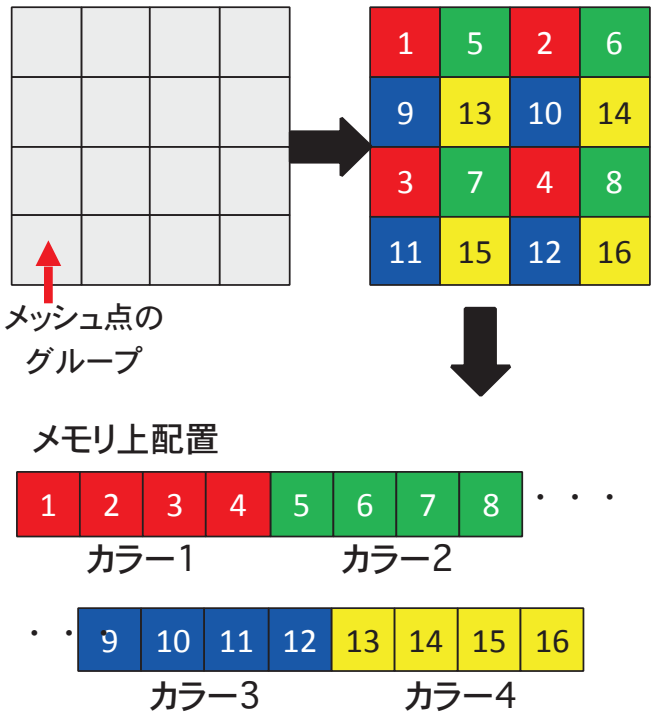
演算順序を変更しカラー毎に処理することで並列化が可能に

カラーリング

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

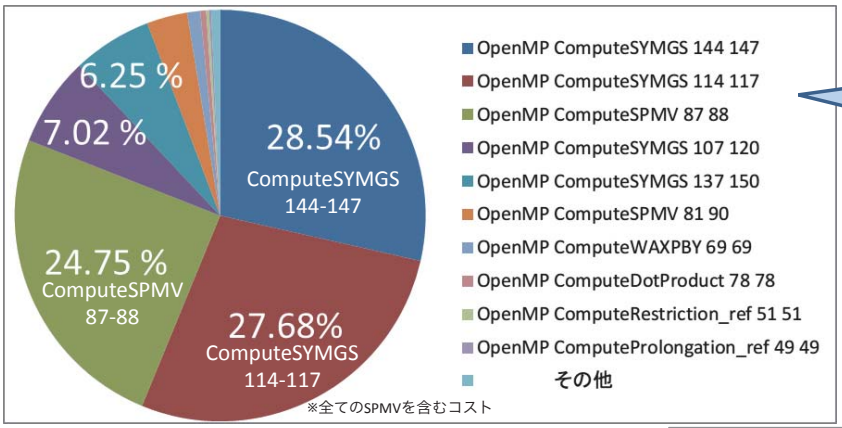


カラーリング



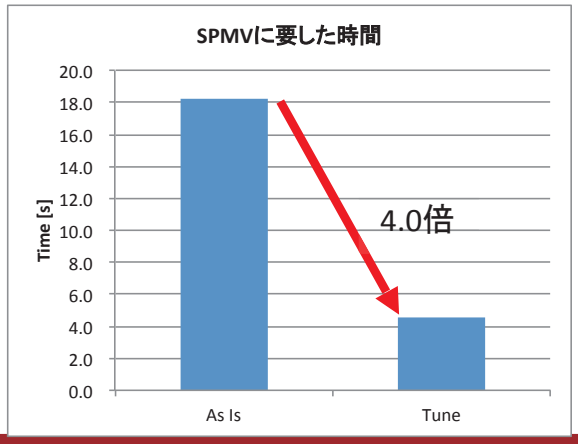
- ここで説明した8色カラーリングは問題の構造的なメッシュ構造を前提としたプログラミングを禁止するレギュレーションに抵触する.
- そのためもっと一般的なカラーリング手法に変更.
- メッシュ点を複数まとめてグループを構成する.
- そのグループ毎にカラーリングを実施.
- カラー内ではグループ間でリカレンスは発生しない.
- そこをスレッド並列に利用.

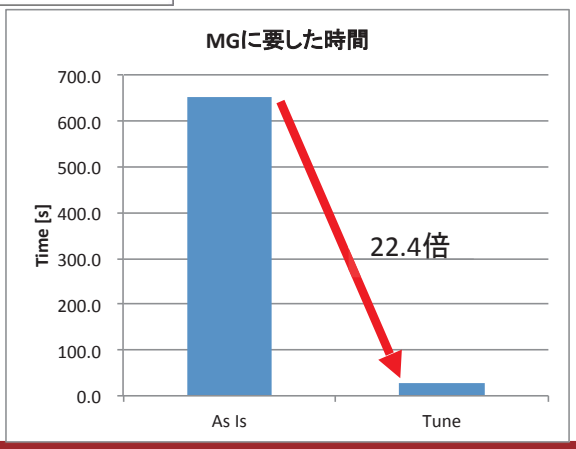
カラーリング



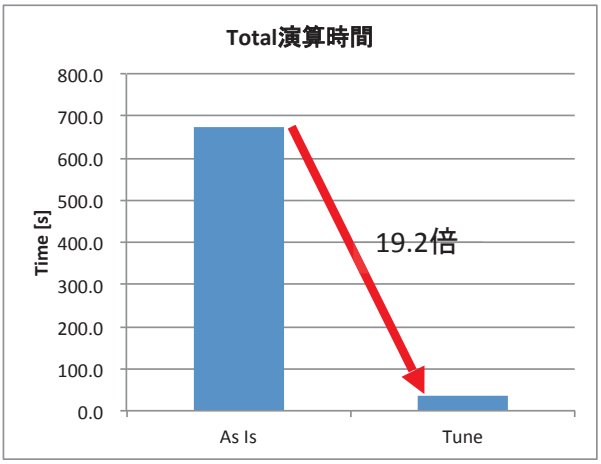
※ データサイズ104^3→112^3に変更

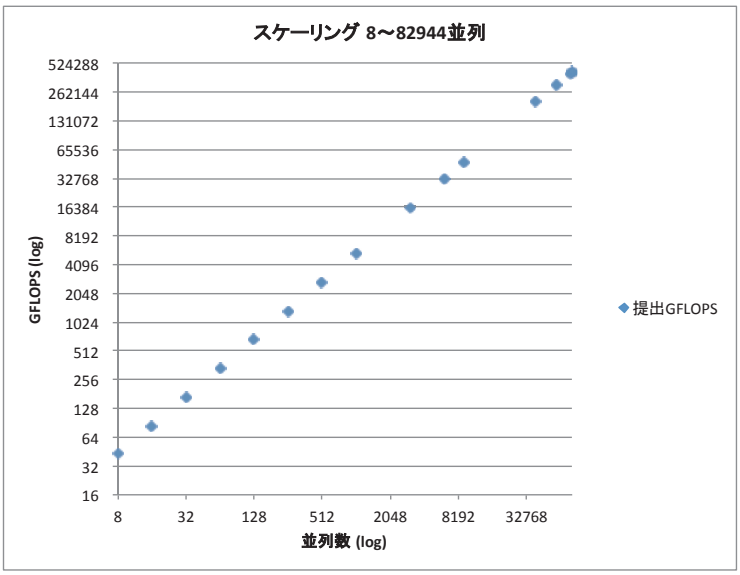
カラー化によりコスト上位のループが全て並列化OK





トータル性能





ISC'14

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 2.2GHz + 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 tine 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 2.4GHz + 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

ISC'14

- 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SC'14で正式のベンチマークとなる予定

まとめ

- 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

科学技術計算分科会　選出
科学技術計算分科会 2014 年度会合 より

スーパーコンピュータを用いた大規模グラフ解析と Graph500 ベンチマーク

藤澤 克樹
(九州大学)

サイエンティフィック・システム研究会
科学技術計算分科会 2014 年度会合　資料

2014 年 10 月 29 日

スーパーコンピュータを用いた大規模グラフ解析と Graph500 ベンチマーク

藤澤 克樹
九州大学　マス・フォア・インダストリ研究所

[アブストラクト]
新しいスーパーコンピュータの応用として大規模なグラフ解析が注目を集めている。新しいスーパーコンピュータ(スパコン)の応用として大規模なグラフ解析やデータ処理が注目を集めている。グラフ解析の応用分野としては大規模災害等での避難誘導計画、社会公共政策や企業経営等のためソーシャル・ネットワーク等の大規模データの有効活用等が想定されているが、非常に計算量やデータ量さらに電力使用量などの規模が大きく従来の手法では処理が困難である。そのためハイパフォーマンスコンピューティング分野の技術を用いて大規模なグラフ解析を行う研究がさかんに行われており、その中から Graph500 ベンチマークとグラフ解析の性能について解説を行う。これまで国内のスパコン（京コンピュータ、東工大 TSUBAME, 東大 FX10 等）を活用して、以下の図 2 のように京コンピュータで世界１位(第８回 Graph500 ベンチマーク)等の優れた成果を挙げた。

[キーワード]　最適化問題, グラフ解析, Graph500 ベンチマーク, HPC, オープンソース

1. はじめに

新しいスーパーコンピュータの応用として大規模なグラフ解析が注目を集めている。グラフは点集合と枝集合から構成される。具体的には図 1 のように様々な応用分野おいて解析の対象とする事象の関係（Relationships）を点と枝で表現していく。例えば道路交通ネットワークでは点は交差点、枝は交差点間の道路に該当する。また Twitter などのソーシャル・ネットワークの解析では、点はユーザ、枝はユーザ間のフォロー関係（あるいはメッセージ送信）などに関連させることが多い。さらに各枝を連結させてグラフを構成して（Step 1)、目的に応じて最短路検索などのグラフ解析を行う(Step 2)。またグラフ解析の結果は元の応用問題の分析や理解のために使用される（Step 3)。実際にカーナビゲーションシステムでは道路ネットワークがグラフデータとして内蔵されていて、ユーザの指示に応じて出発地点と目的地点間の最短路検索を行っている。このように社会における実データをグラフデータに変換して、計算機で高速処理する需要が非常に高まっている。

2. 大規模グラフ探索と Graph500 ベンチマーク

スパコンのベンチマークテストでは Top 500 が有名である。Top 500 では主に数値計算能力が測定される。今日ではスパコンの応用が幅広い分野に及び、交通ネットワーク分析やソーシャル・ネットワーク解析で注目されている大規模グラフ解析などのビッグデータ処理用途においては Top 500 の結果を用いて計算機評価を行うことが難しくなっている。
そのため、スパコンの大規模データ処理性能を計測する Graph 500 （注１） ベンチマークテストが 2010 年より開始された。また、Green Graph500 （注２） は Graph500 と同じくグラフ探索性能を競うベンチマークテストであり、昨今の厳しい電力事情を考慮すると、性能と同時に省電力性も極めて重要であり、今回の省電力グラフ探索の技術の多方面への応用が期待されている。

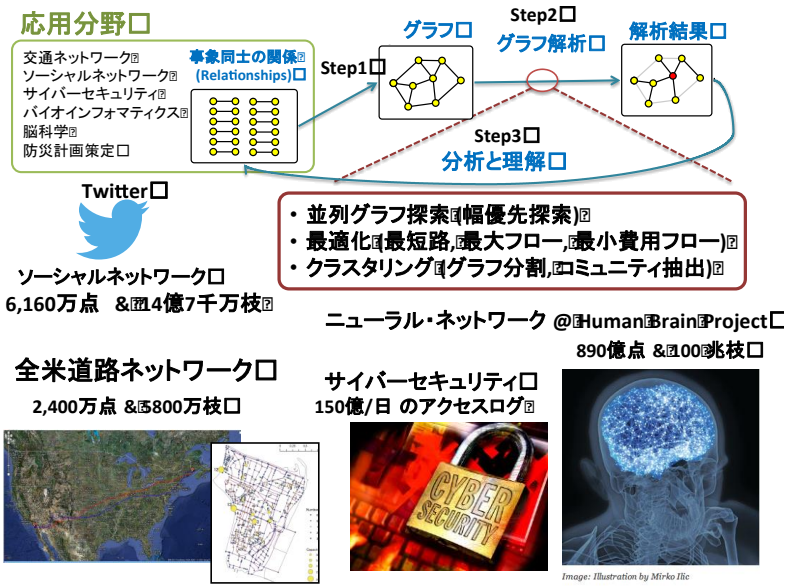


図 1: グラフ解析の利用方法と応用分野

共同研究チームは、次世代のスパコン上で大規模なグラフの高速な探索処理を行うソフトウェアの開発を 2011 年から進めてきた。冗長なグラフ探索を削減するアルゴリズムの利用、数千〜数万台規模が高速なネットワークで接続された超並列計算機上での通信性能の最適化、マルチコアプロセッサ上でのメモリへのアクセス最適化、フラッシュデバイスの利用による大規模グラフ処理と高性能化の両立などの先進的なソフトウェア技術を高度に組み合わせることにより、モバイルデバイスからスパコンまでの様々なコンピュータ上での高速かつ省電力なグラフ処理を可能にしている。最近（国際会議 ISC14, 2014 年 6 月）、独自に開発したソフトウェアを用いて、大規模なグラフを解くことでスパコン上のビッグデータ処理を計測する Graph500、及び、その省電力性を計測する Green Graph500 ベンチマークを様々な研究機関や企業の協力と支援によって実施した結果、両者において世界第 1 位の高成績を達成した。当 CREST チームでは国内のスパコン（京コンピュータ、東工大 TSUBAME, 東大 FX10 等）を活用して、以下の**図 2** のように京コンピュータで世界 1 位(第 8 回 Graph500 ベンチマーク)等の優れた成果を得た。

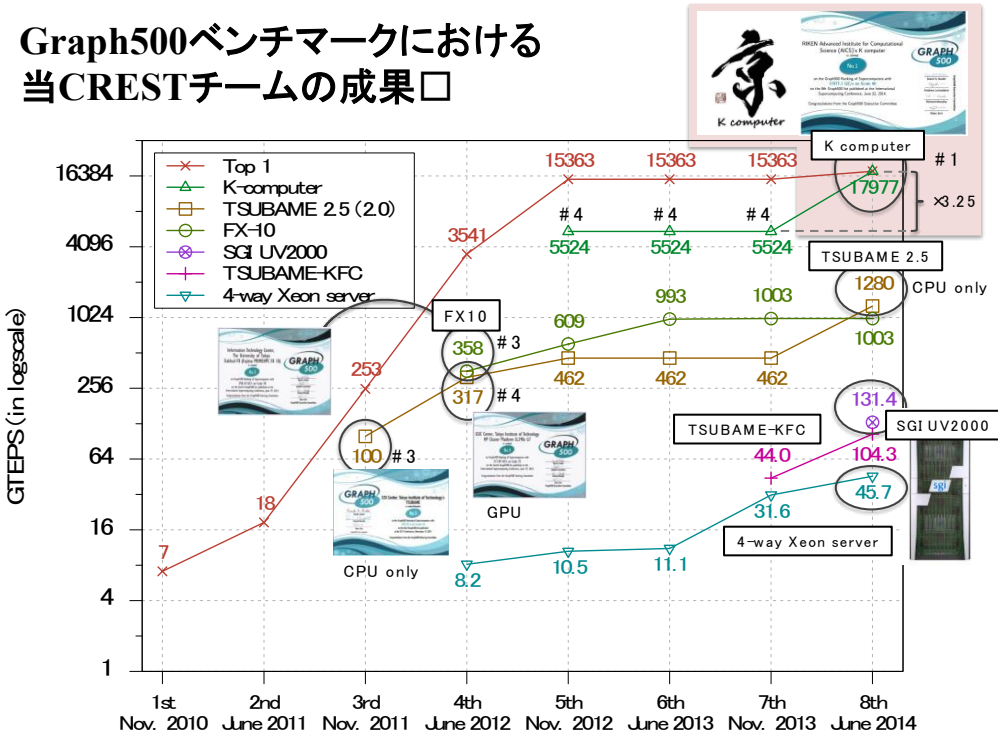


図 2: Graph500 ベンチマークにおける当 CREST チームの成果

（注１） Graph500 ： 幅優先探索によって１秒間に探索したグラフのエッジ （枝） の数が指標として用いられる。具体的には単位指標は TEPS （Traversed Edges Per Second） が用いられ、TEPS 値が高ければ、高速にグラフ探索を行うことができることを意味する。Web サイト：http://www.graph500.org/

（注２） Green Graph500 ： 単位指標は TEPS / W（Traversed Edges Per Second / Watt）が用いられ、TEPS /W 値が高ければ、単位消費電力あたりのグラフ探索性能が高い、つまり省電力性能が高いことを意味する。Green Graph500 では処理する対象となるグラフの規模に応じてビッグデータ部門とスモールデータ部門に分かれている。Web サイト：http://green.graph500.org/

CREST 戦略的創造研究推進事業
Core Research for Evolutional Science and Technology

JST Japan Science and Technology Agency

スーパーコンピュータを用いた大規模グラフ解析とGraph500 ベンチマーク

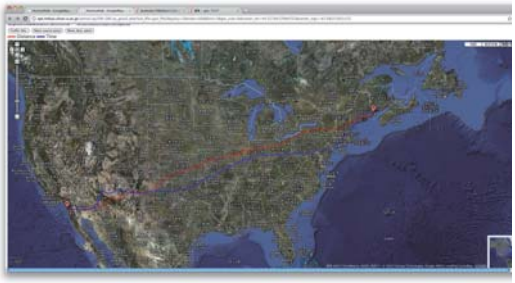
藤澤 克樹

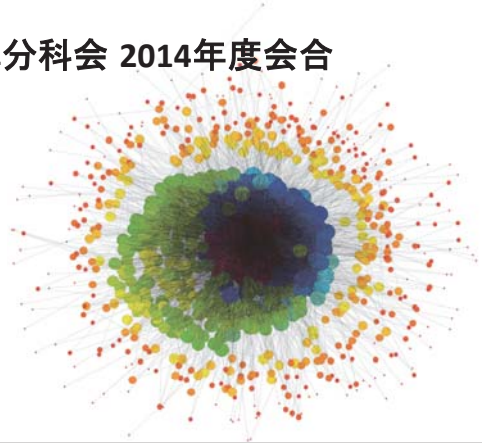
九州大学マス・フォア・インダストリ研究所＆ JST CREST

共同研究者：上野 晃司（東工大), 安井 雄一郎(九大), 鈴村 豊太郎（University College Dublin), 佐藤 仁, 岩渕 圭太, 溝手 竜(東工大)

2014年10月29日

サイエンティフィック・システム研究会 科学技術計算分科会 2014年度会合
次世代HPCを支える技術
ホーテルオークラ神戸

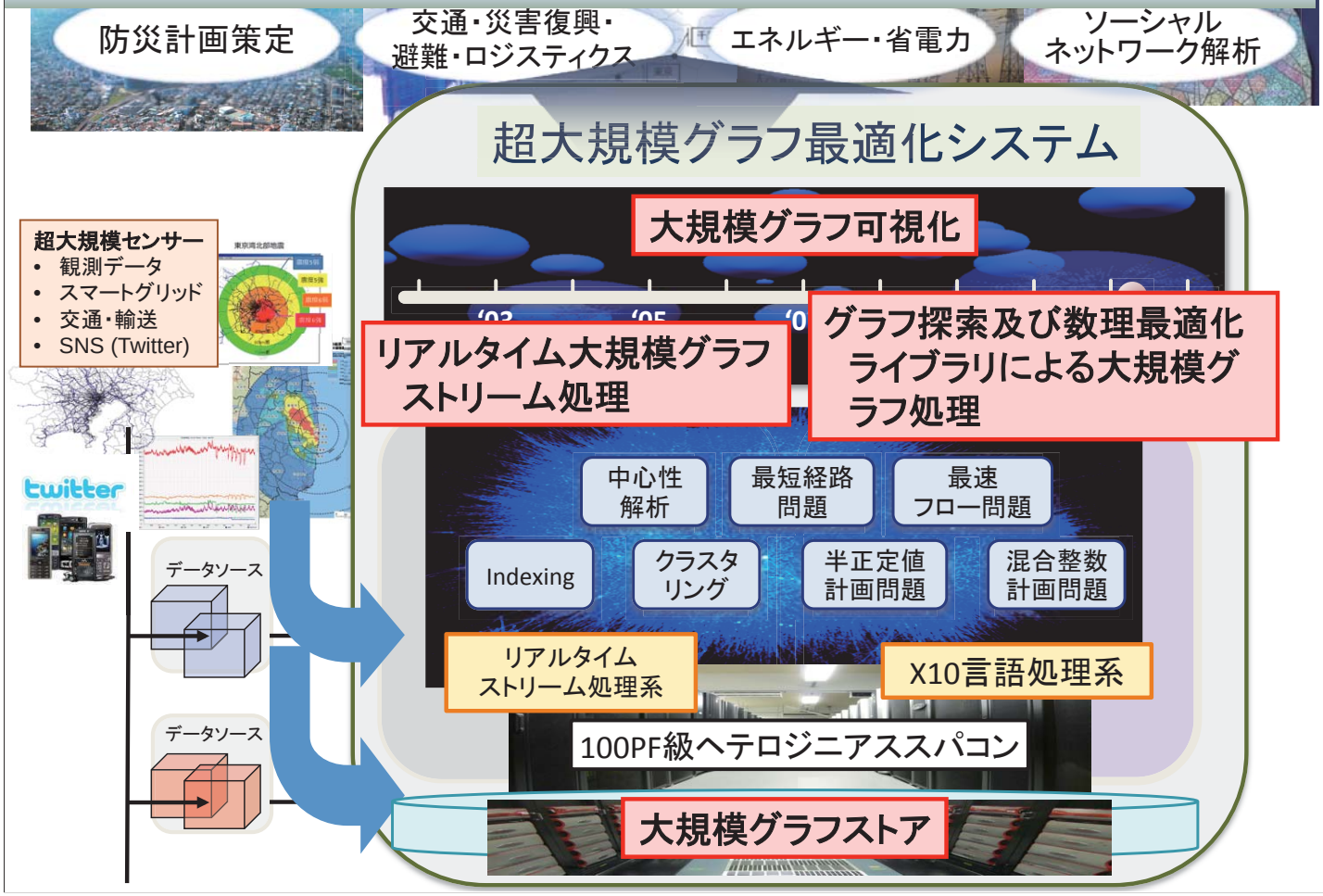




当 CREST の研究計画の概要

- 次世代ポストペタスパコンでの解決すべき課題の特定
  - 並列数の爆発的増大、不均質化、高密度化
  - 記憶装置の多階層化・大容量化
  - アルゴリズム的、システム的に様々な解決すべき課題と困難が存在
- **大規模グラフ解析及び数理最適化システム**
  - 緊急に取り組むべき課題と実社会へのインパクト
  - Graph500(Green Graph500)ベンチマーク (巨大グラフ, BFS)
    - ISC12 : 358GTEPS (世界3位) , 8.15GTEPS (1ノード世界1位)
    - SC12 : 5524GTEPS(世界4位), 10.495GTEPS(1ノード世界1位)
    - ISC13 : 6.119GTEPS/kW (世界1位)
    - SC13 : 6.72GTEPS/kW (BigData 1位), 153.17GTEPS/kW (SmallData 1位)
    - ISC14 : 17977GTEPS(世界1位), 59.12GTEPS/kW (BigData 1位)
  - 数理計画問題（SDP): (世界記録更新：233万制約 ; 1.713PFlops)
    - SC12(Tech. paper) , IPDPS14 など： 疎&密データ計算(24,480CPUコア ＆ 4080GPU) ： 日本オペレーションズリサーチ学会研究賞、NVIDIA GTC Japan 最優秀ポスター発表賞等
- 最適化とHPC系研究者のポストペタスパコン上での **Co-design による解決**
- ポストペタスパコン上での基盤ソフトの整備に貢献すると共に安心安全な社会の実現を目指す

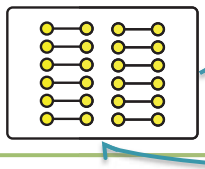
超大規模グラフ最適化システムの提案



応用分野

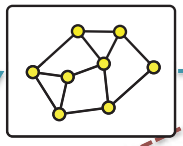
交通ネットワーク
ソーシャルネットワーク
サイバーセキュリティ
バイオインフォマティクス
脳科学
防災計画策定

事象同士の関係 (Relationships)



Step1

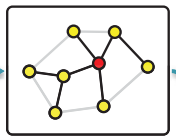
グラフ



Step2

グラフ解析

解析結果



Step3

分析と理解

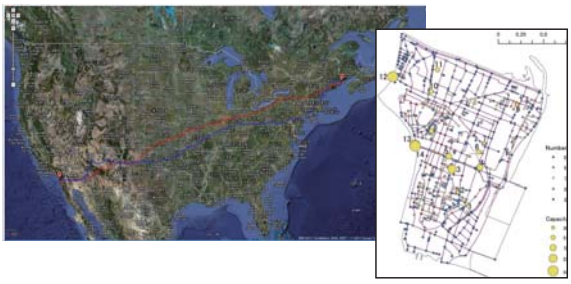
- 並列グラフ探索 (幅優先探索)
- 最適化 (最短路, 最大フロー, 最小費用フロー)
- クラスタリング (グラフ分割, コミュニティ抽出)

Twitter

ソーシャルネットワーク
6,160万点 & 14億7千万枝

ニューラル・ネットワーク @ Human Brain Project
890億点 & 100 兆枝

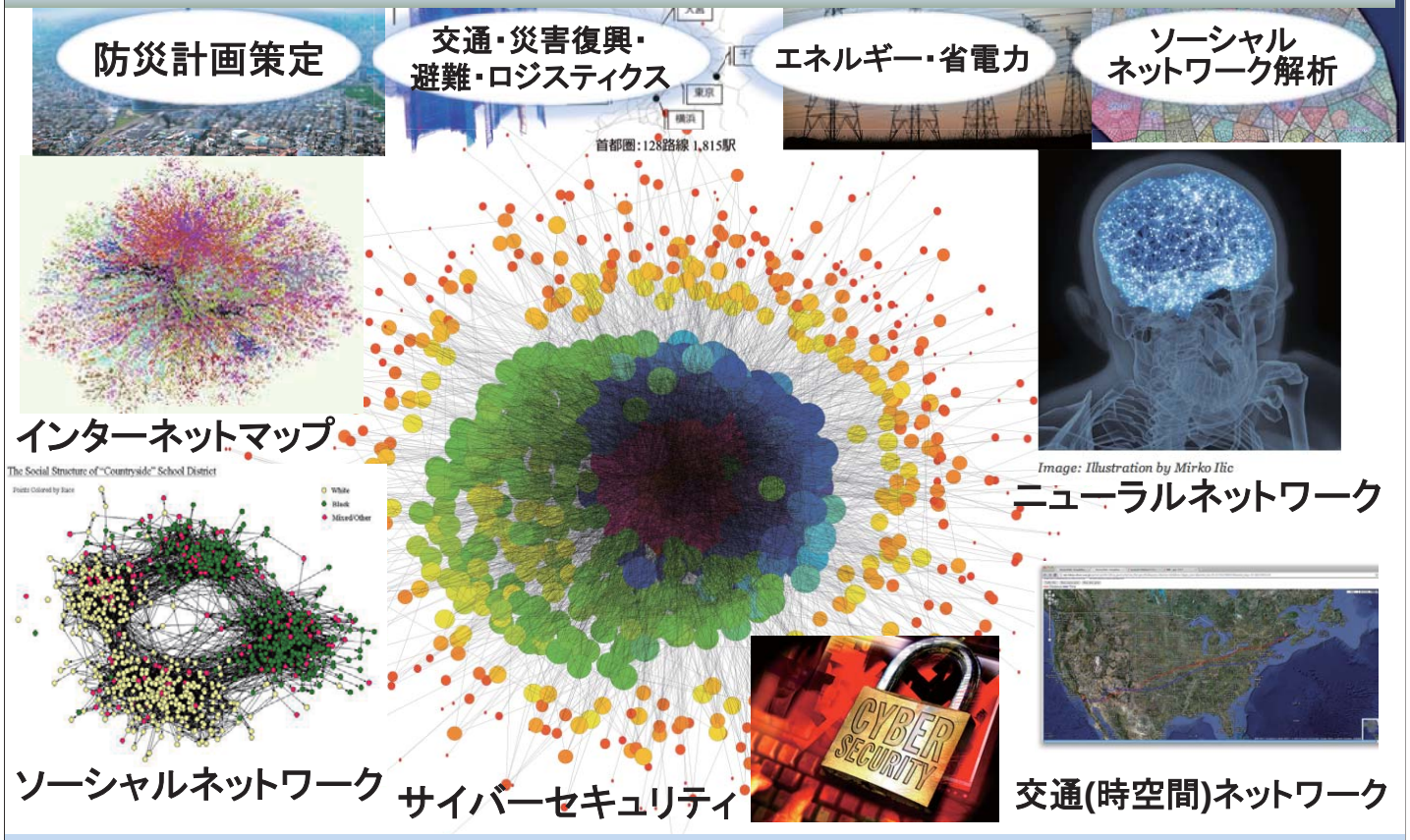
全米道路ネットワーク
2,400万点 & 5800万枝



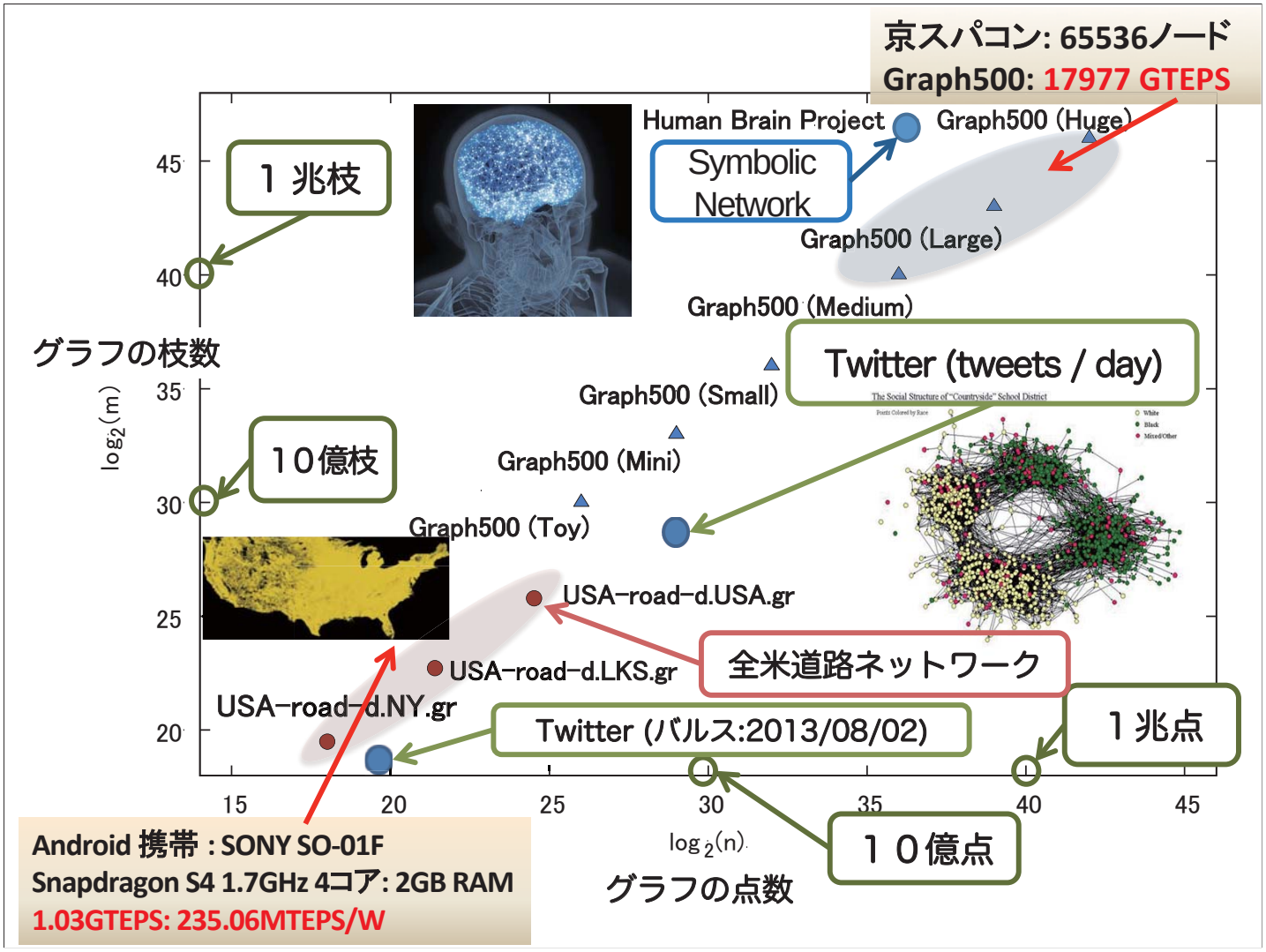
サイバーセキュリティ
150億/日 のアクセスログ

Image: Illustration by Mirko Ilic

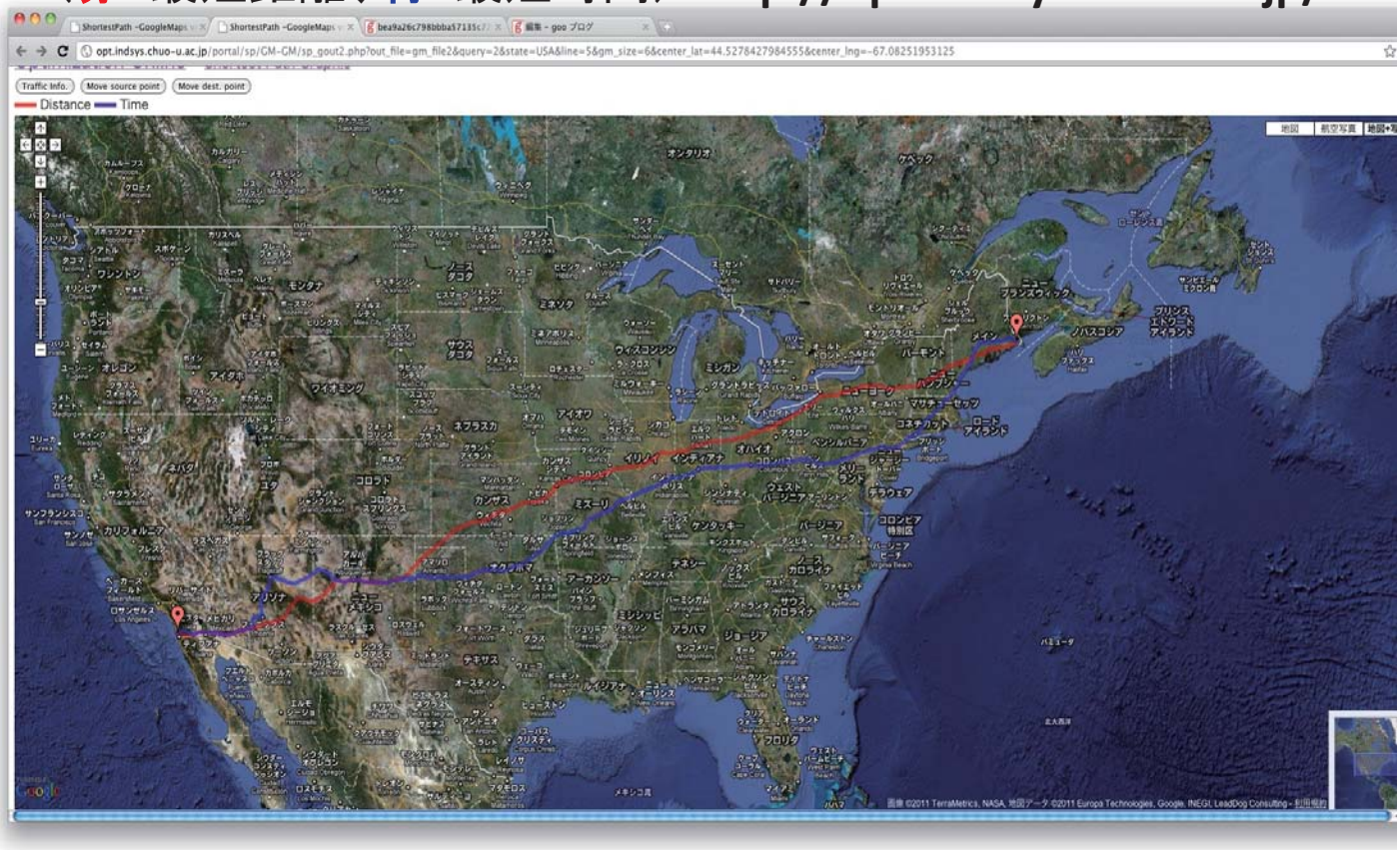
大規模グラフ解析：緊急に取り組むべき課題と実社会へのインパクト



スモールワールドやスケールフリー性などの特性を持つ超巨大グラフ(1兆点以上) 上での高速探索技術（アルゴリズム、実装技術、高速計算）



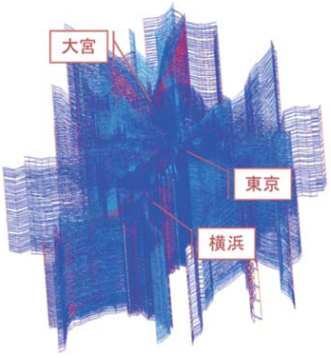
全米横断の最短路: 2400万点, 5800万枝 → 約2秒で計算(3W)
サンディエゴから北東部のメーン州までの最短路
(赤:最短距離、青:最短時間): http://opt.imi.kyushu-u.ac.jp/



超大規模ネットワークにおける高速探索技術の開発
コアとなるアルゴリズムやソフトウェアのカーネル部分の実装と評価

- 超大規模ネットワークデータ（点数１０億以上)が対象
- 最短路(1対全), グラフ探索(深さ優先、幅優先等)
- グラフ探索の局所的な評価では優先キュー(ヒープ木)を用いる⇒実行時間、メモリ消費量が安定的
- 実行時間(1対全最短路計算: CPU : Intel Xeon 5670 2.93GHz)
  - 全米グラフ 3秒（点数 n = 23,947,347、枝数 m = 58,333,344)
  - 超大規模グラフ 870秒 (点数 n = 10億, 枝数 m = 20億)
- 多くの応用(交通ネットワーク, e-コミュニティのネットワーク、ロジスティクスネットワーク)

			スレッド数	クエリ数	消費メモリ MB	実行時間 (秒)	実行時間/クエリ(マイクロ秒)	確認リンク数	usec/ #arcs	コスト
始点	横浜	4822491	1	100	564.54	5.564	55640	428086	0.1300	19108
終点	東京	757995	8	100	1815.59	2.139	21390	428086	0.0500	19108
始点	新宿	1085224	1	100	564.54	35.794	357940	4046406	0.0885	34493
終点	千葉	4174499	8	100	1815.59	7.610	76100	4046406	0.0188	34493
始点	川崎	661699	1	100	564.54	17.954	179540	2072075	0.0866	30527
終点	大宮	1576725	8	100	1815.59	3.905	39050	2072075	0.0188	30527
始点	東京	2321666	1	100	564.54	10.190	101900	1026584	0.0993	19002
終点	ディ	261784	8	100	1815.59	3.160	31600	1026584	0.0308	19002
始点	沖縄	377	1	72	564.54	257.421	3575292	44595172	0.0802	17264925
終点	北海道	9065116	8	72	1815.59	44.871	623208	44595172	0.0140	17264925



点の重要性

- グラフの特徴を示す指標
  - 中心性、PageRank …
- 中心性指標が広く利用されている
  - 近年は中心性が盛んに研究されている
- 中心性指標にもいくつか種類がある
  - 各点に接続されている枝数（Degree Centrality）
  - 各点が最短路に含まれる回数（Betweenness Centrality）
  - 最も遠い点までの最短距離（Graph Centrality）
  - 各点までの最短距離（Closeness Centrality）

中心性の定義

v-t 間最短路長

- Closeness Centrality

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$$

- Graph Centrality

$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$$

- Stress Centrality

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$

v を通る s-t 間最短路数

- Betweenness Centrality

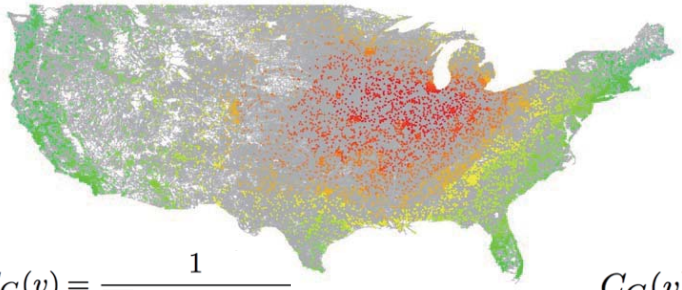
$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

s-t 間最短路数

全米グラフに対する重要性(中心性)の計算

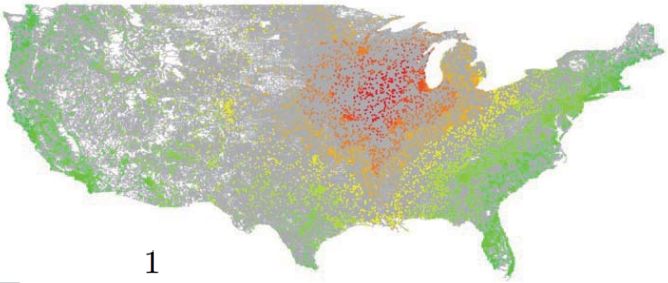
- ネットワーク内での各点の**重要度**を推定
- **各点の周辺、及び広域内における影響**(情報の伝播力)を推定する
- 超大規模ネットワークデータ上での短時間に大量のグラフ探索を行う必要がある

Closeness



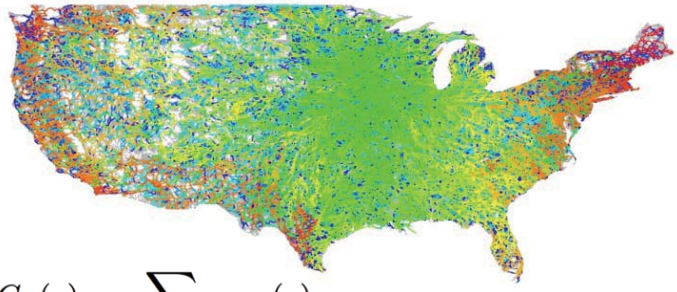
$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$$

Graph



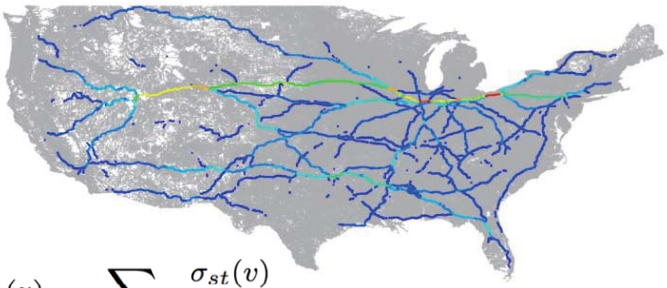
$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$$

Stress(対数)



$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$

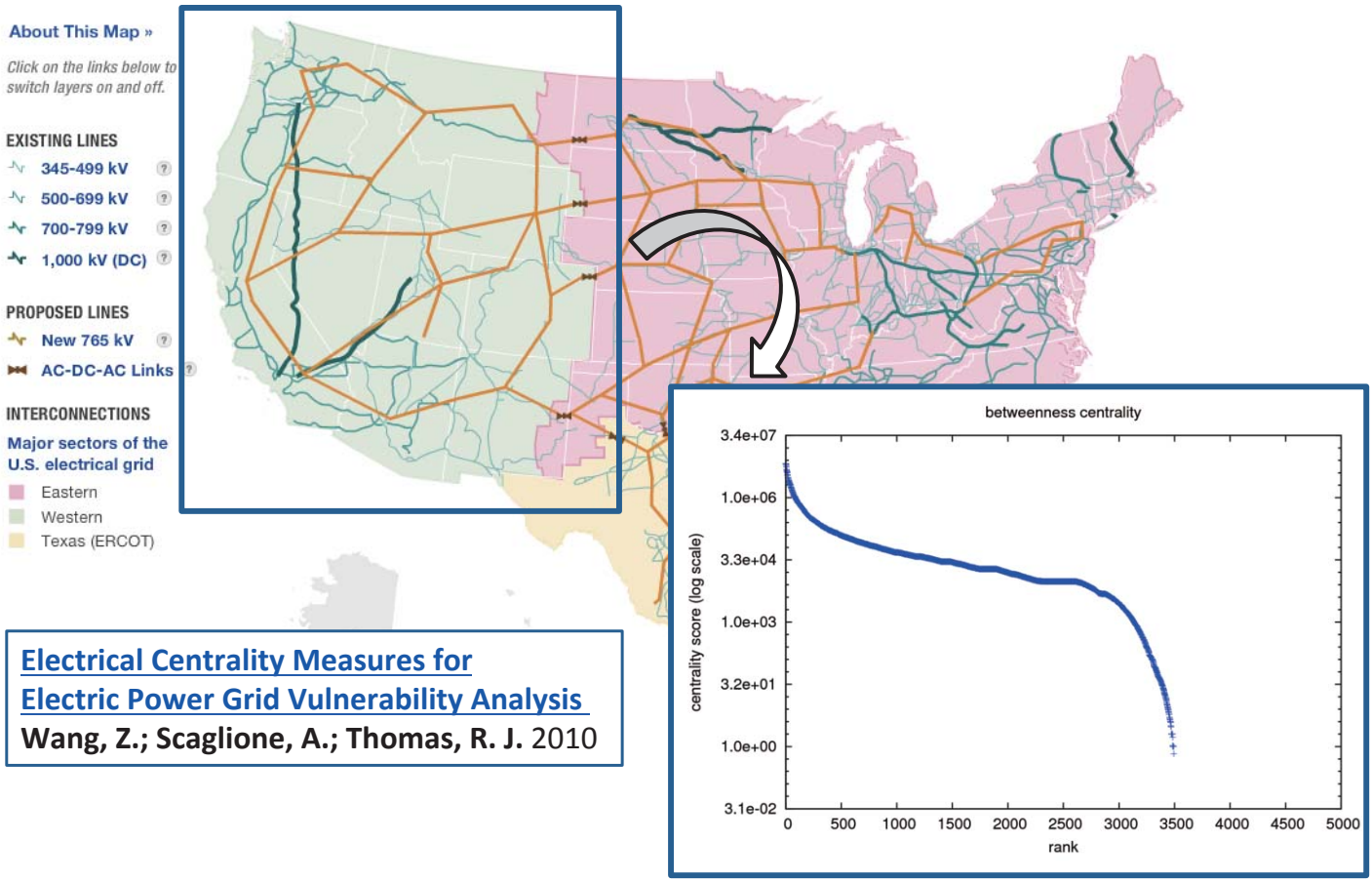
Betweenness



$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

The U.S. Electric Grid

http://www.npr.org/templates/story/story.php?storyId=110997398



Electrical Centrality Measures for Electric Power Grid Vulnerability Analysis
Wang, Z.; Scaglione, A.; Thomas, R. J. 2010

全米道路ネットワークに対する全対全最短路問題

点数 n = 23,947,347, 枝数 m = 58,333,344

4-way Opteron 6174 2.3 GHz (12 cores x 4), 256 GB, GCC-4.6.0

アルゴリズム	全対全最短路	高速化率
Dijkstra's algorithm	512 年	--
Δ-stepping	9.1 年	60 倍
Multi-Level Buckets	4.9 年	110 倍
MSLC + NUMAを考慮した最適化	7.8 日	24,000 倍

各2点間の最短路長を
1.1ns で求めることができる

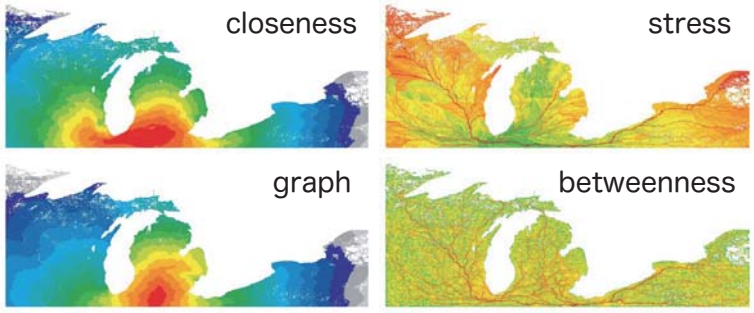
厳密な
全対全最短路長を
約7.8日で計算

五大湖周辺道路ネットワーク最短路を用いた中心性計算

点数 n = 2,758,119, 枝数 m = 6,885,658

4-way Opteron 6174 2.3 GHz (12 cores x 4), 256 GB, GCC-4.6.0

重みを考慮しない中心性：19.35 時間



既存の GraphCT では
BC のみの計算で
20.6 日かかる

NYデータに対する中心性(BC)の計算結果(最短路問題)

○サーバ1 (2CPU x 4 コア = 8 コア)
実行時間 1957.652 秒 : 8スレッド

CPU : AMD Opteron 2356 2.3GHz x 2
メモリ : 32GB
OS : Vine Linux 5.1 for X86_64

○サーバ2 (1CPU x 4 コア = 4 コア)
実行時間 2058.443 秒 : 4スレッド

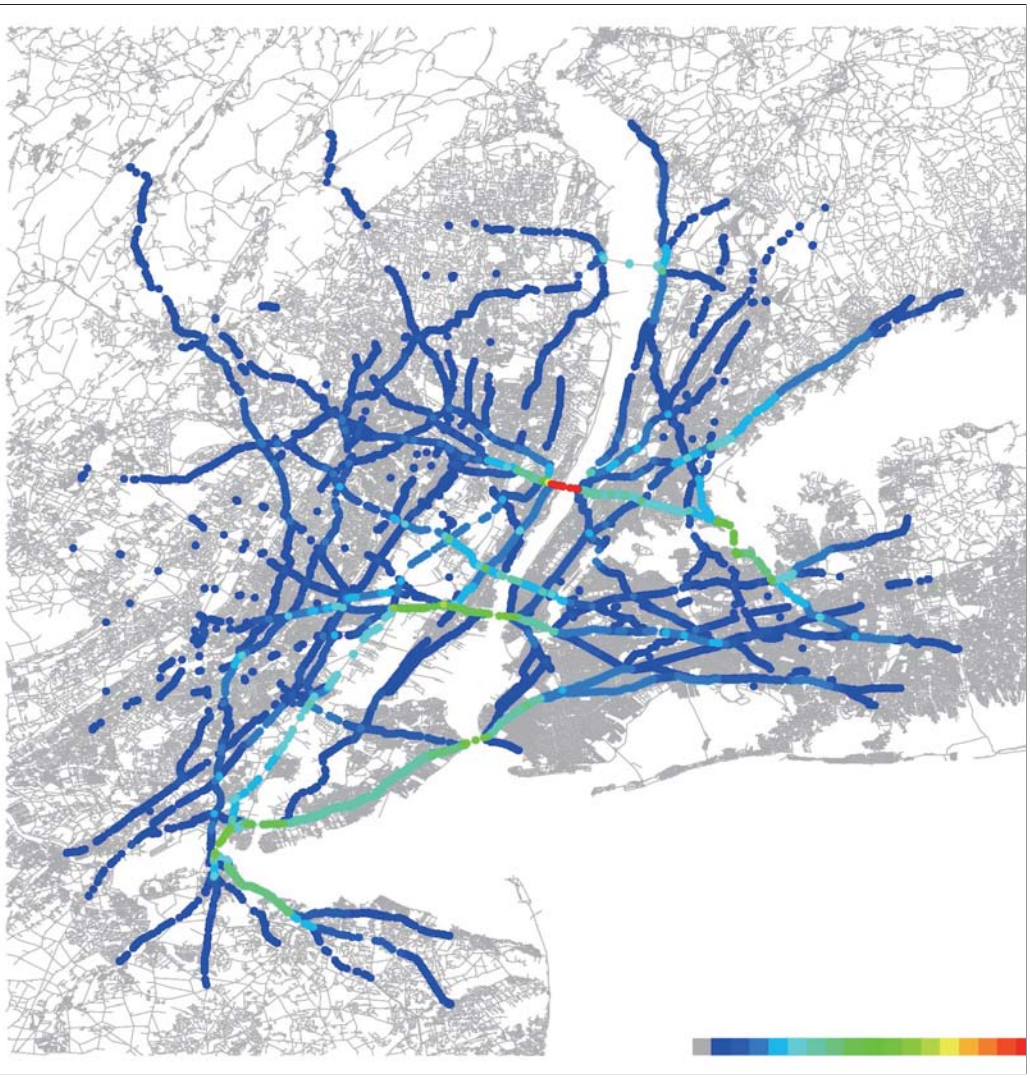
CPU : Intel Core i7 965 3.2GHz x 1
メモリ : 12GB
OS : CentOS 5.5 for x86_64

○サーバ3 (4 CPU x 6 コア = 24 コア)
実行時間 632.859 秒 : 24スレッド

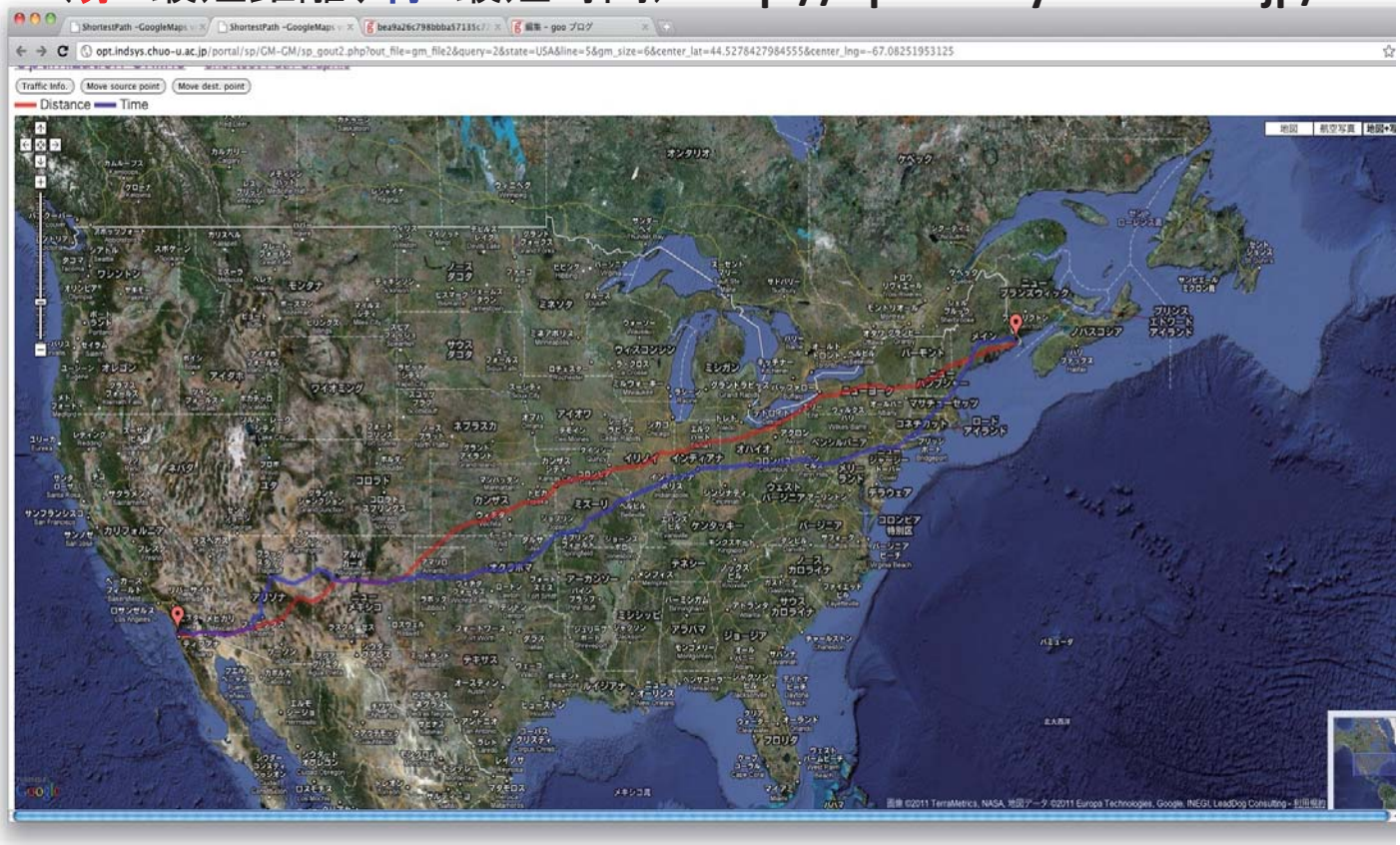
CPU : AMD Opteron 8439 2.80GHz x 4
Memory : 128GB
OS : Fedora 13 for x86_64

○ サーバ4 (2 CPU x 4 コア = 8 コア)
実行時間 1319.054 秒 : 8スレッド

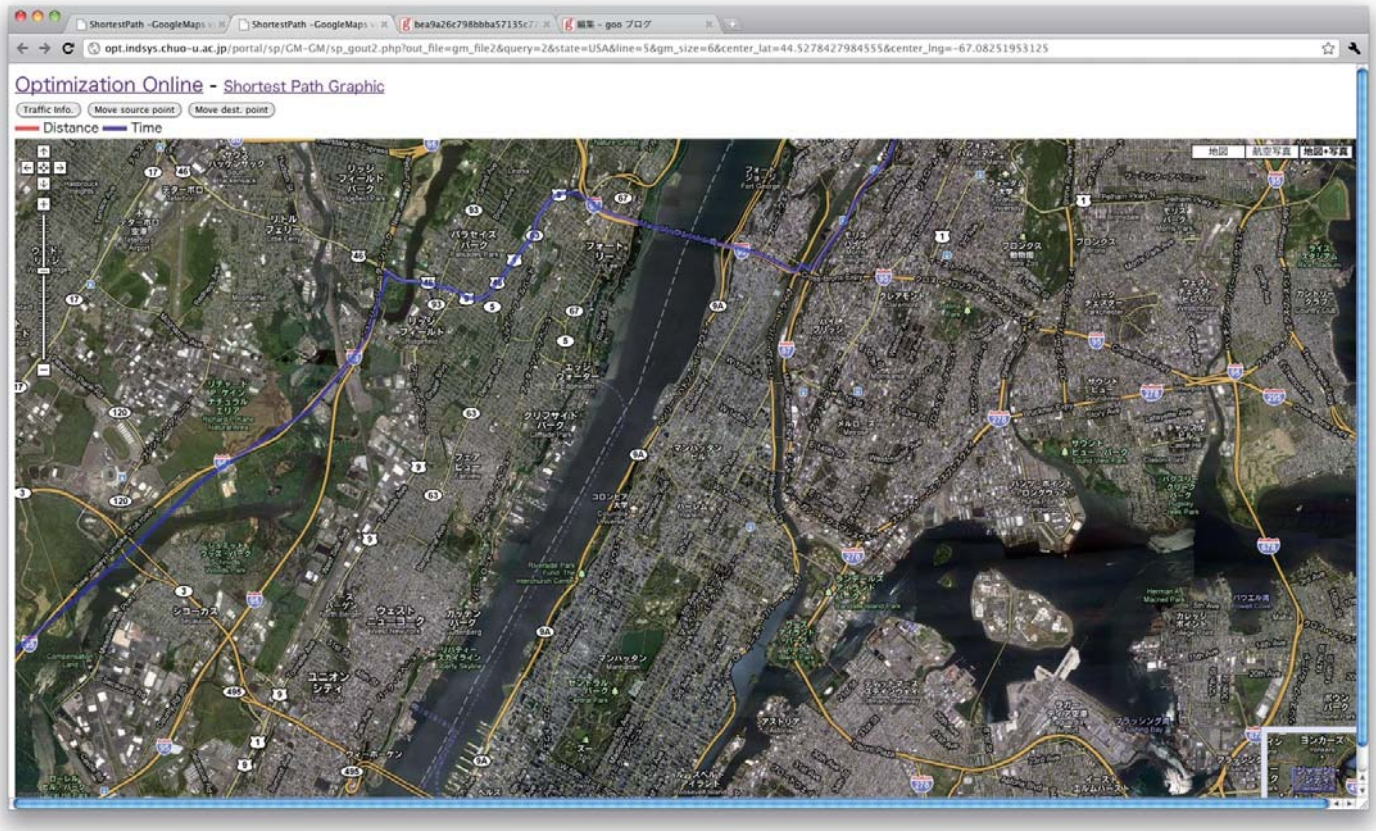
CPU : Intel Xeon 5550 2.66GHz x 2
Memory : 72GB
OS : Fedora 13 for x86_64



全米横断の最短路: **2400万点, 5800万枝 → 約2秒で計算(3W)**
サンディエゴから北東部のメーン州までの最短路
（赤:最短距離、青:最短時間）: **http://opt.imi.kyushu-u.ac.jp/**



ジョージ・ワシントン橋, ニューヨークを
通過する全米横断最短路



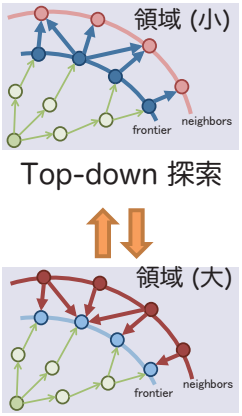
Graph500・Green Graph500

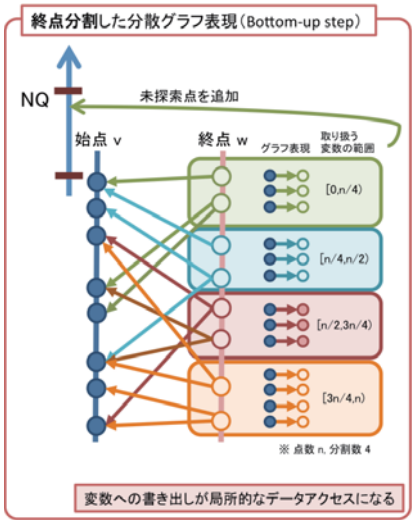
○ Graph500・Green Graph500 ベンチマーク

- パラメータ SCALE と edgefactor (=16) から、点数 $n=2^{SCALE}$, 枝数 m=edgefactor・n となる Kroncekr graph を生成
- 幅優先探索 (BFS) での1秒辺りの探索枝数 TEPS により、Graph500 リストを作成
- 消費電力あたりの TEPS（TEPS/W) により、Green Graph500 リストを作成

○ 高速な幅優先探索の実装 BFS

- 計算機性能を引き出す汎用的な高速化



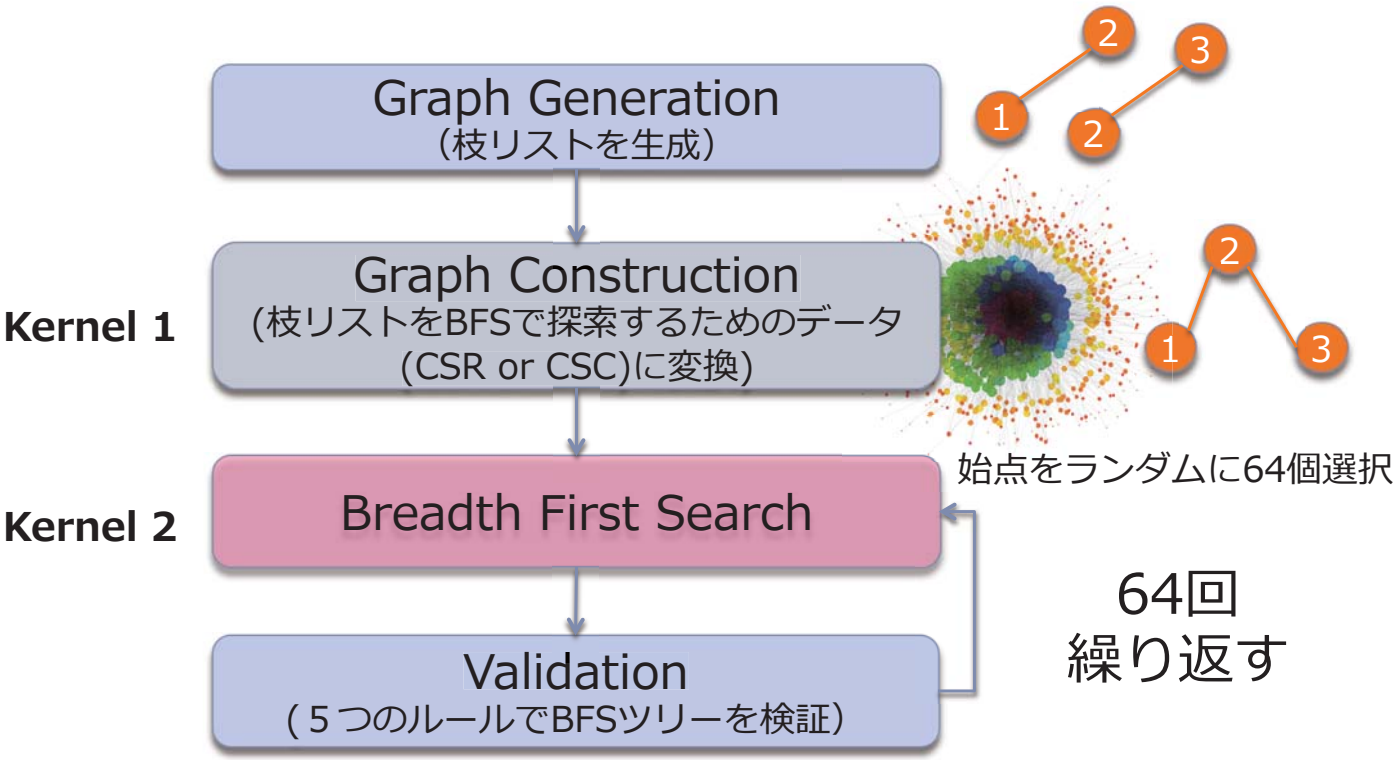


○ Green Graph500 (June 2013)

Rank	MTEPS/W	Site	Machine	G500 rank	Scale	GTEPS	Nodes
1	64.12	Chuo University	GraphCREST-Tegra	132	20	0.153885	1
2	53.82	Chuo University	GraphCREST-Intel-NUC	110	23	1.08175	1
3	53.47	Chuo University	GraphCREST-Mac-mini	90	24	1.94104	1
4	52.02	Chuo University	GraphCREST-MBA13	105	23	1.22761	1
5	51.62	Chuo University	GraphCREST-Retina15	89	24	1.98735	1
6	39.29	Changsha, China	TH-IVB-FEP/C	-	26	9.74402	1
7	32.25	Chuo University	GraphCREST-NEXUS10	133	20	0.118688	1
8	30.41	Tokyo Institute of Technology	MFP2012 Ultra green-ok	78	24	5.12372	1
9	20.68	Chuo University	GraphCREST-Intel-NUC	72	25	6.14248	1
10	18.68	Chuo University	GraphCREST-Sbep2.4	67	25	6.77685	1
11	17.39	Chuo University	GraphCREST-4way58ep2.4	57	26	11.1148	1
12	10.95	Chuo University	GraphCREST-Wex40	58	26	11.061	1
13	2.43	The University of British Columbia	Akindi	106	28	1.20576	1
14	1.89	University of Luxembourg	Viridis HPC@UniLu	-	21	0.389643	32
15	0.20	Swiss National Supercomputing Center	Todi	49	29	15.5983	272

上位独占

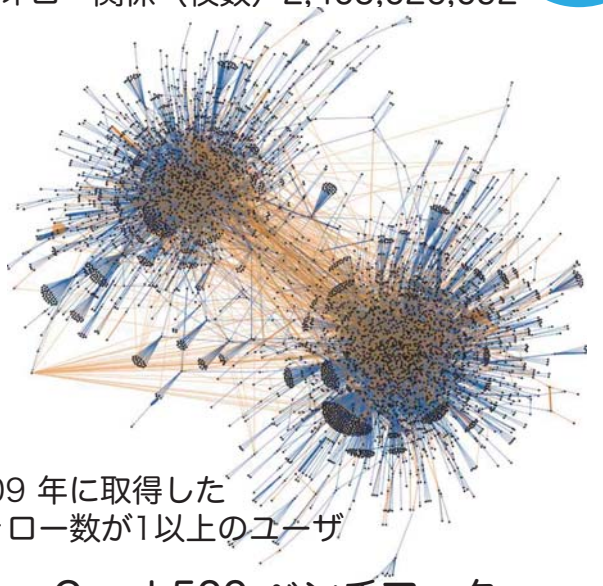
Graph500ベンチマークの実行順



※CSR: Compressed Sparse Row, CSC: Compressed Sparse Column

Graph500技術の応用
Twitter ネットワークの解析

- フォロー・ネットワーク
  - ユーザ数　　（点数）　41,652,230
  - フォロー関係（枝数）2,405,026,092



2009 年に取得した
フォロー数が1以上のユーザ

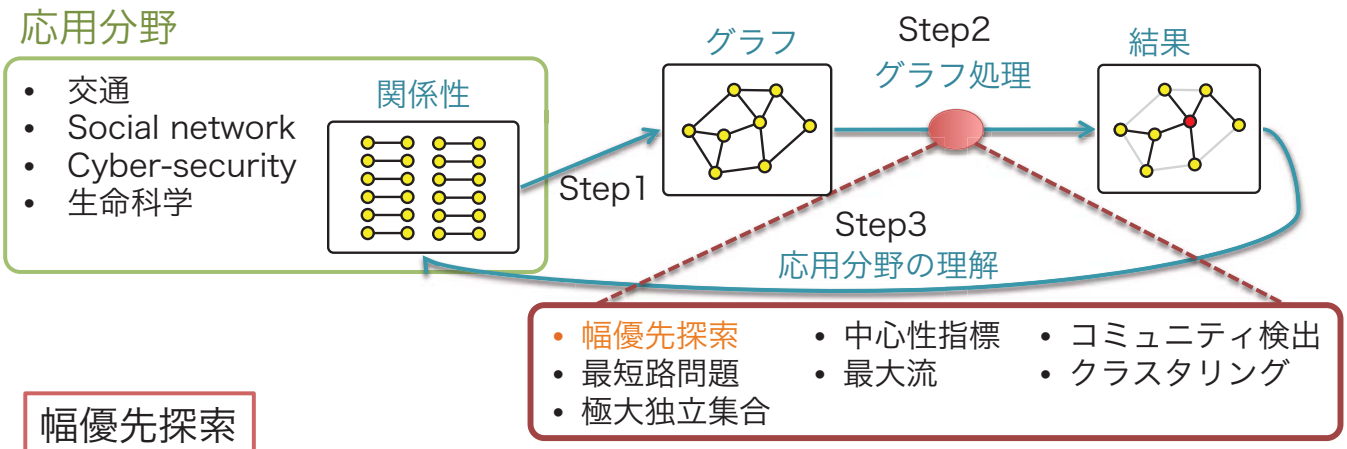
- Graph500 ベンチマーク
  - 幅優先探索の性能「1 秒間に通過した枝数 TEPS」を用いて、コンピュータの性能を比較する

ユーザ 21,804,357 からの幅優先探索の結果

ホップ数	ユーザ数	割合 (%)	累積割合 (%)
0	1	0.00	0.00
1	7	0.00	0.00
2	6,188	0.01	0.01
3	510,515	1.23	1.24
4	29,526,508	70.89	72.13
5	11,314,238	27.16	99.29
6	282,456	0.68	99.97
7	11536	0.03	100.00
8	673	0.00	100.00
9	68	0.00	100.00
10	19	0.00	100.00
11	10	0.00	100.00
12	5	0.00	100.00
13	2	0.00	100.00
14	2	0.00	100.00
15	2	0.00	100.00
合計	41,652,230	100.00	-

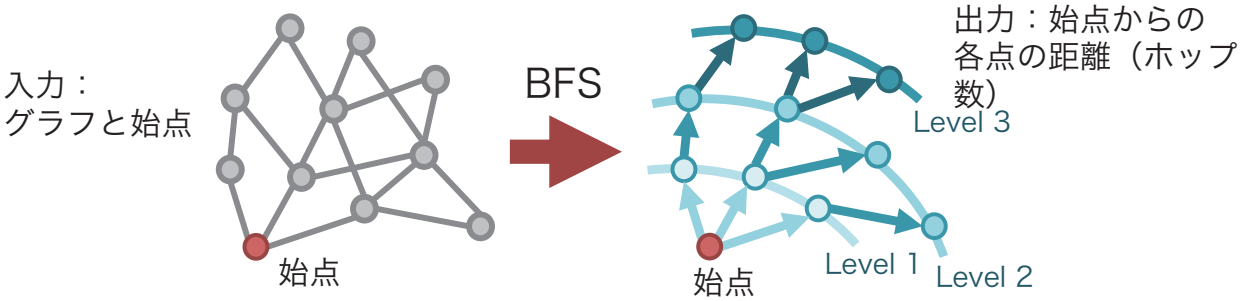
0.069 秒で探索可能
⇒ 21.28 GTEPS（10^9 TEPS）

グラフ処理と幅優先探索



幅優先探索

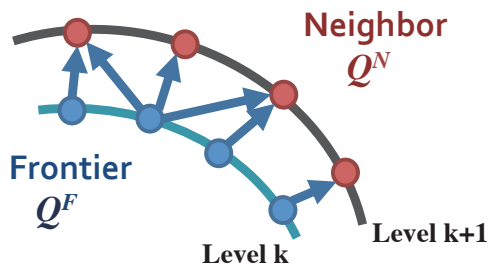
- 最も重要かつ基本的なグラフ処理の一つ
- 他のアルゴリズムのサブルーチン
- 演算量に比べて、不連続なメモリアクセスが多いことが問題



Level-synchronized parallel BFS (**Top-down**)

- Started from source vertex and executes following two phases for each level

Traversal ⋯ finds **unvisited adjacency vertices** from current frontier Q^F and append to neighbor Q^N



Swap ⋯ swaps the frontier Q^F and the neighbor Q^N for next level

Algorithm 1: Level-synchronized Parallel BFS.

Input : $G = (V, A)$: unweighted directed graph.
s : source vertex.
Variables: Q^F : *frontier* queue.
Q^N : *neighbor* queue.
visited : vertices already visited.
Output : $\pi(v)$: predecessor map of BFS tree.

$\pi(v) \leftarrow -1, \forall v \in V$
$\pi(s) \leftarrow s$
$visited \leftarrow \{s\}$
$Q^F \leftarrow \{s\}$
$Q^N \leftarrow \emptyset$
while $Q^F \neq \emptyset$ **do**
  for $v \in Q^F$ **in parallel do**
    for $w \in A(v)$ **do**
      if $w \notin visited$ **atomic then**
        $\pi(w) \leftarrow v$
        $visited \leftarrow visited \cup \{w\}$
        $Q^N \leftarrow Q^N \cup \{w\}$
  $Q^F \leftarrow Q^N$
  $Q^N \leftarrow \emptyset$

Traversal

Swap

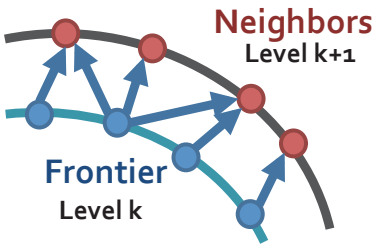
Hybrid-BFS (Direction-optimizing BFS)

Chooses one from **Top-down** or **Bottom-up** for frontier size at each level

Beamer2012

Top-down algorithm

- Efficient for **small**-frontier
- Uses **out-going** edges



Input : *Directed graph* $G = (V, A^{\mathcal{F}})$, *Queue* Q^F
Data : *Queue* Q^N, *visited*, *Tree* $\pi(v)$

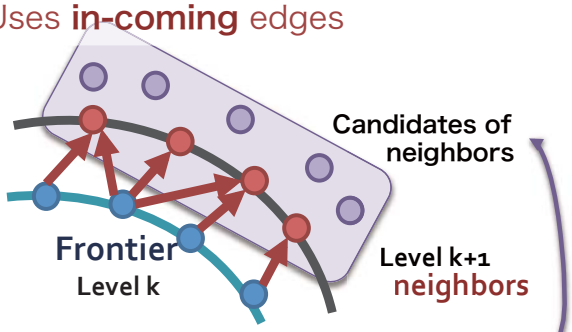
$Q^N \leftarrow \emptyset$
for $v \in Q^F$ **in parallel do**
  for $w \in A^{\mathcal{F}}(v)$ **do**
    if $w \notin visited$ **atomic then**
      $\pi(w) \leftarrow v$
      $visited \leftarrow visited \cup \{w\}$
      $Q^N \leftarrow Q^N \cup \{w\}$
$Q^F \leftarrow Q^N$

Current frontier

Unvisited neighbors

Bottom-up algorithm

- Efficient for **large**-frontier
- Uses **in-coming** edges



Input : *Directed graph* $G = (V, A^{\mathcal{B}})$, *Queue* Q^F
Data : *Queue* Q^N, *visited*, *Tree* $\pi(v)$

$Q^N \leftarrow \emptyset$
for $w \in V \setminus visited$ **in parallel do**
  for $v \in A^{\mathcal{B}}(w)$ **do**
    if $v \in Q^F$ **then**
      $\pi(w) \leftarrow v$
      $visited \leftarrow visited \cup \{w\}$
      $Q^N \leftarrow Q^N \cup \{w\}$
      break
$Q^F \leftarrow Q^N$

Candidates of neighbors

Current frontier

Skips unnecessary edge traversal

Hybrid-BFS (Direction-optimizing BFS)

Chooses one from **Top-down** or **Bottom-up**
for a number of traversed edges at each level

Beamer2012

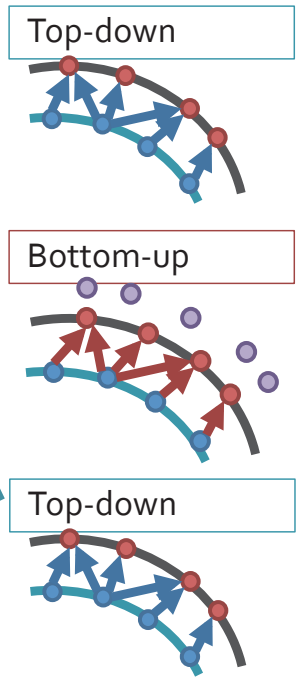
Number of traversal edges of Kronecker graph with SCALE 26

$|V| = 2^{26}, |E| = 2^{30}$

Distance from source

Level	Top-down	Bottom-up	Hybrid
0	2	2,103,840,895	2
1	66,206	1,766,587,029	66,206
2	346,918,235	52,677,691	52,677,691
3	1,727,195,615	12,820,854	12,820,854
4	29,557,400	103,184	103,184
5	82,357	21,467	21,467
6	221	21,240	227
Total	2,103,820,036	3,936,072,360	65,689,631
Ratio	100.00%	187.09%	3.12%

$= |E|$



Hybrid-BFS reduces
unnecessary edge traversals

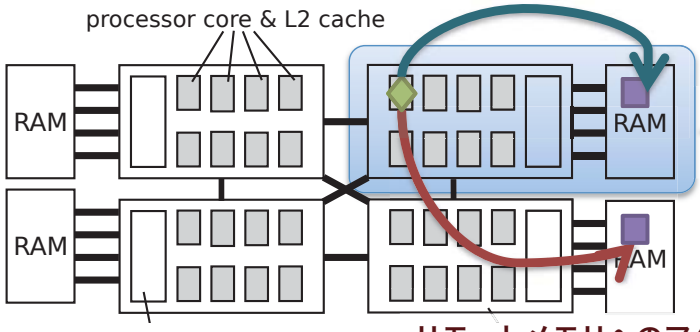
NUMA アーキテクチャの構成

- 4-way Intel Xeon E5-4640 (Sandybridge-EP)
  - 4 (CPU ソケット数)
  - 8 (CPU ソケットあたりの物理コア数)
  - 2 (物理コアあたりのスレッド数)

最大
4 x 8 x 2 = 64 スレッド
（ハイパースレッディング）

NUMAノード

ローカルメモリへのアクセス（アクセスコスト小）



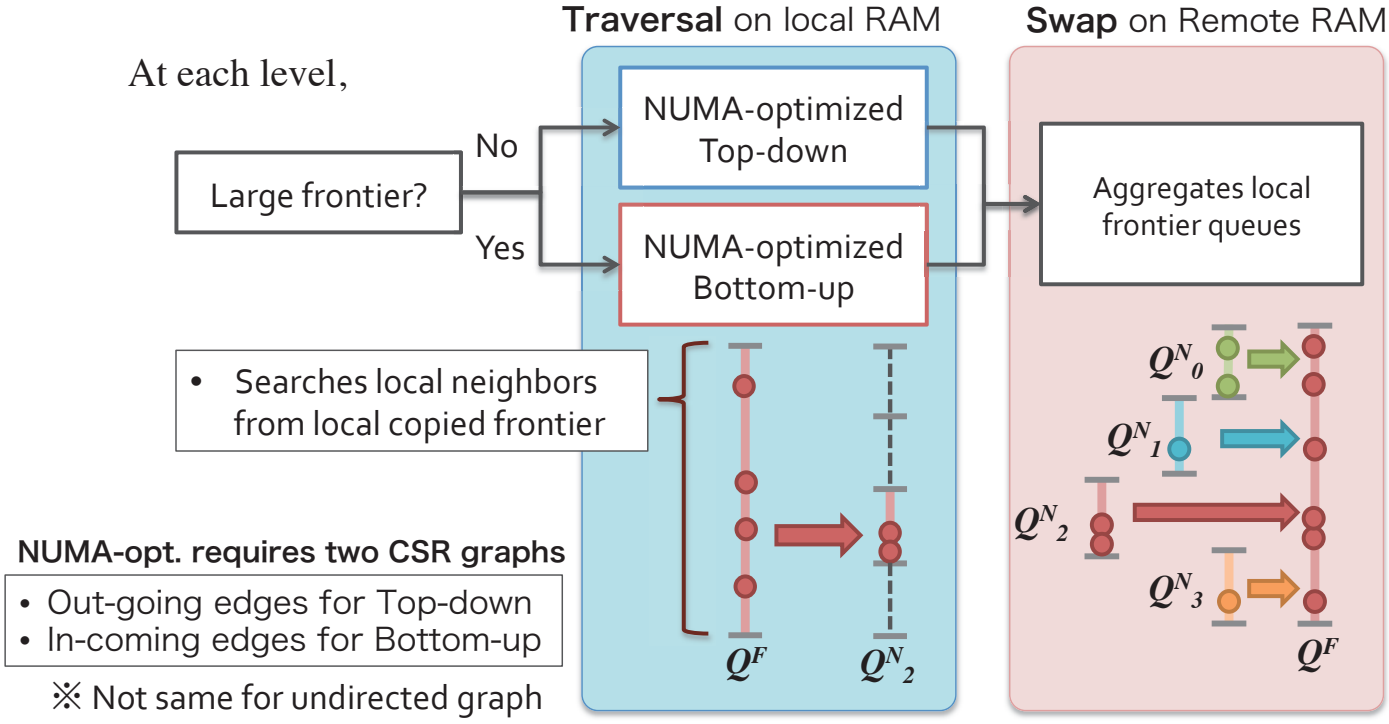
NUMAノード
CPUソケット（16 論理コア）
ローカルメモリ

リモートメモリへのアクセス（アクセスコスト大）

データアクセスが不均一であるため
- 並列時の偏りが生じるため、並列効率を向上することが難しい
- 本来の性能を予想することが難しい

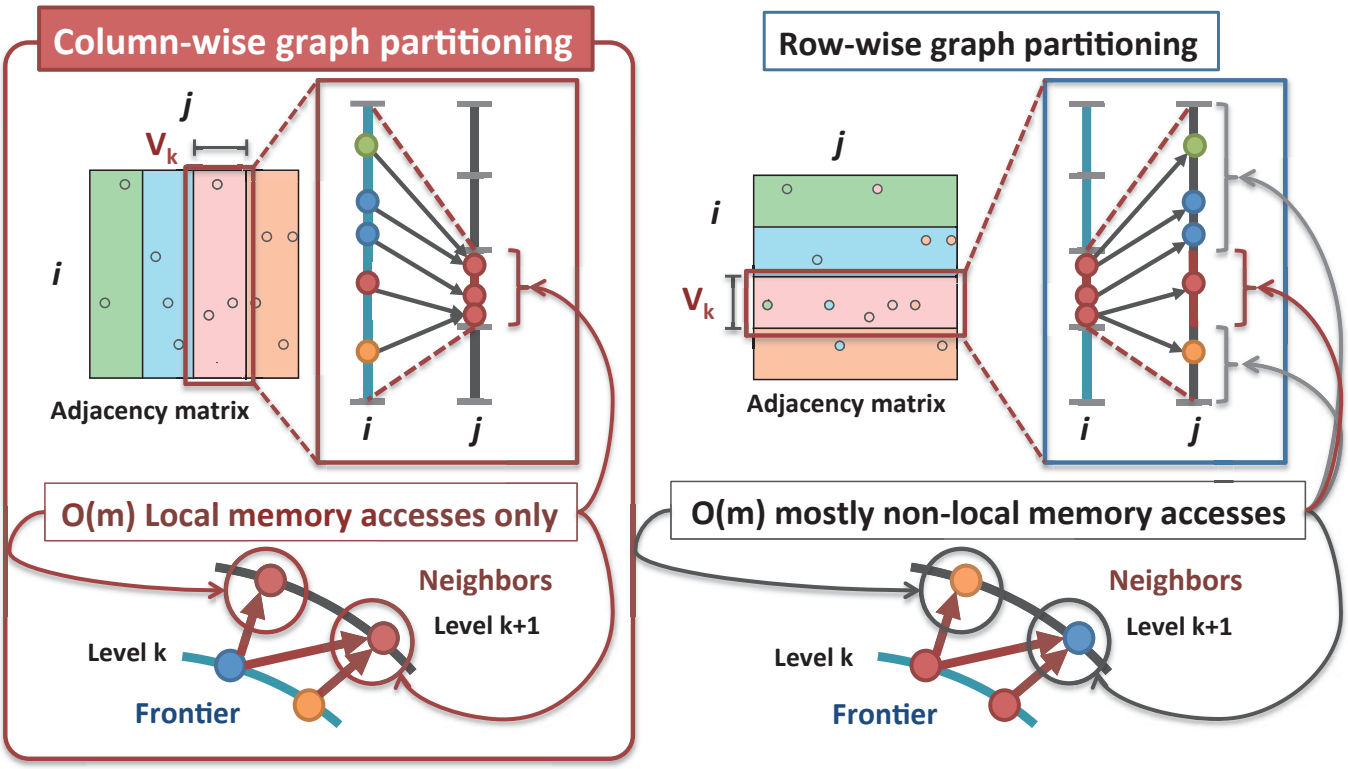
NUMA-optimized BFS

- Clearly separated to accessing for local and remote memory
  - Edge traversal on Local RAM
  - All-gather of local queues and bitmaps for Remote RAM



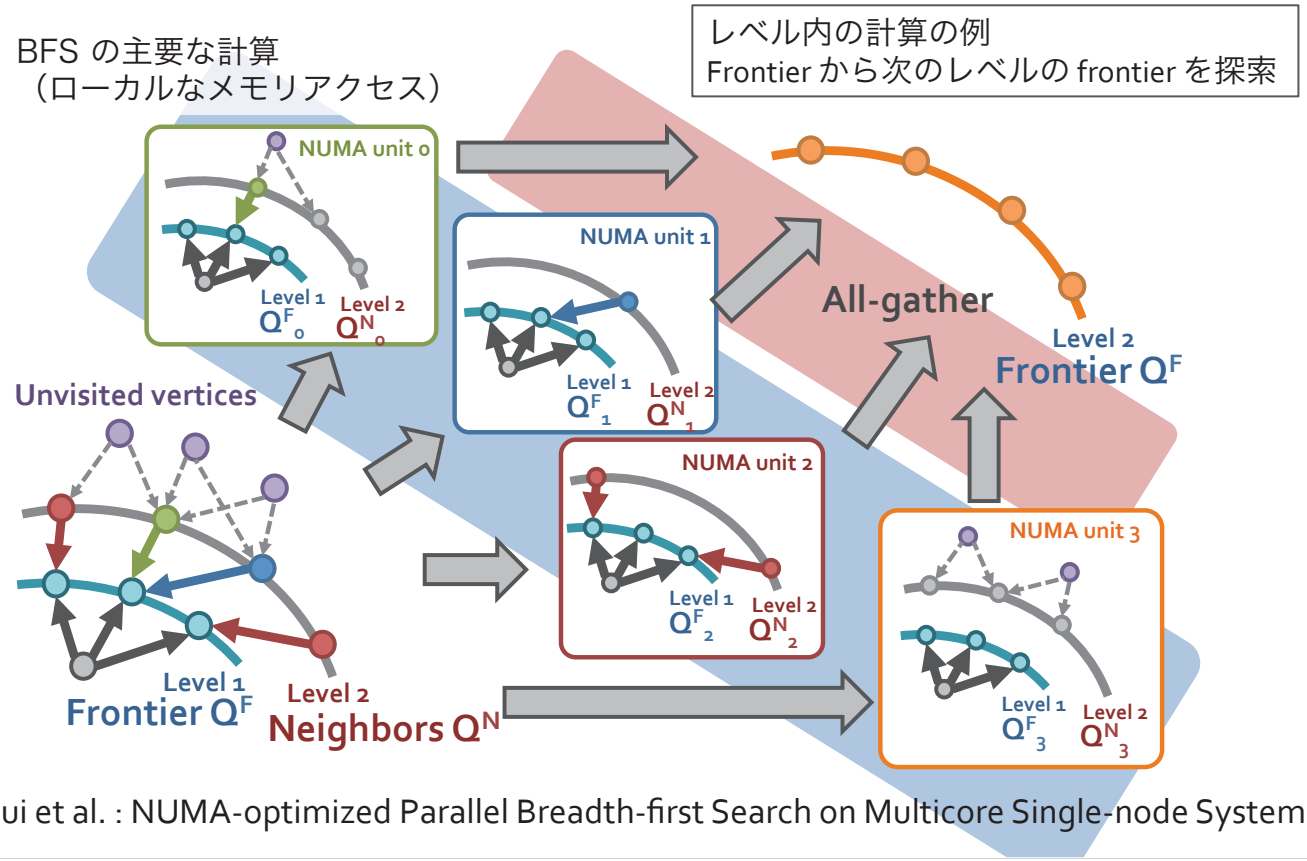
NUMA-opt. Column-wise Graph Partitioning

- divides **G=(V,A)** into partial **G_k=(V_k, A_k)** and binds **local RAM k**.
  - **A_k** is a set of adjacency list that holds **incoming edges** to **V_k**.



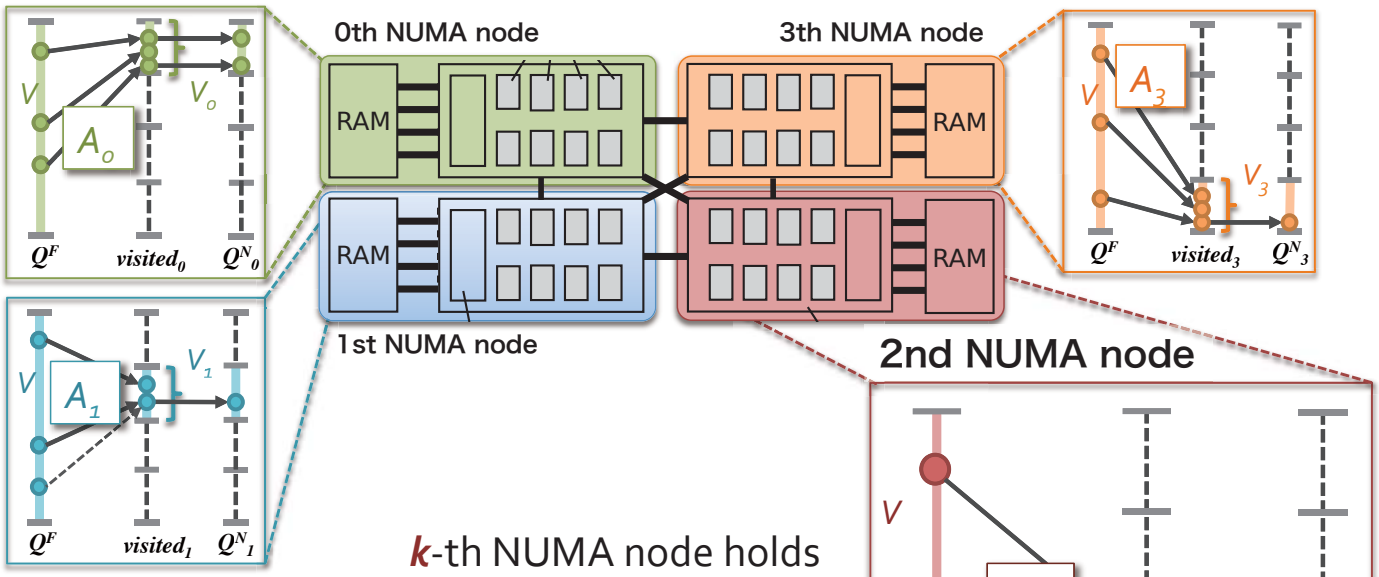
NUMA-opt. parallel hybrid BFS (Graph500 1ノード最速)

- 計算の大部分をローカルなメモリアクセス上で実行



Y. Yasui et al. : NUMA-optimized Parallel Breadth-first Search on Multicore Single-node System

Local Edge Traversal



k-th NUMA node holds

- Local copied frontier Q^F
- partial vertices V_k

$$V_k = \left\{ v_j \in V \mid j \in \left[\frac{kn}{\ell}, \frac{(k+1)n}{\ell} \right) \right\},$$

- partial edges (v_i, v_j), ($v_i \in V$ and $v_j \in V_k$)

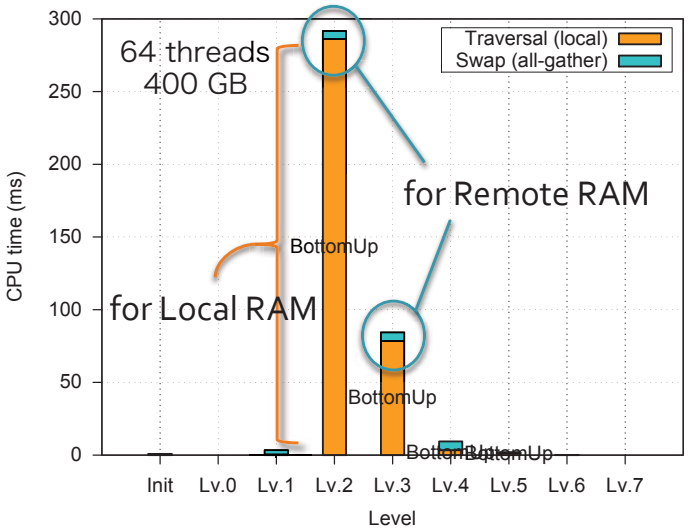
Intel Xeon v.s. SGI Altix UV1000

- Graph500 (SCALE29)
  - 536.9 million vertices, 8.59 billion edges

Median TEPS: 21.81 GE/s

Intel Xeon (Sandybridge-EP arch.)

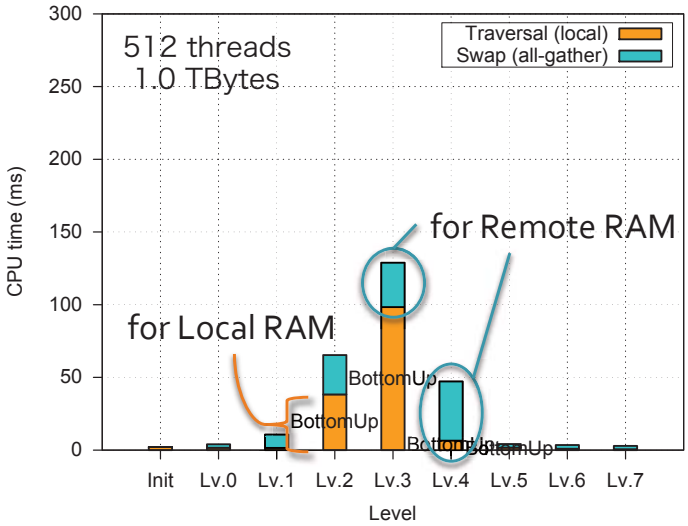
Intel Xeon E5-4650 @ 2.70GHz
64-threads = 8-cores x 2-threads x 4-nodes
512 GBytes RAM



Median TEPS: 31.81 GE/s

SGI Altix UV1000 (Westmere-EX arch.)

Intel Xeon E7-8837 @ 2.67GHz
512-threads = 8-cores x 64-nodes
4.0 TBytes RAM



Intel Xeon v.s. SGI Altix UV1000

- Graph500 (SCALE30)
  - 1.07 billion vertices, 17.18 billion edges

Rank.50
Fastest of single-node
on current list

Intel Xeon (Sandybridge-EP arch.)

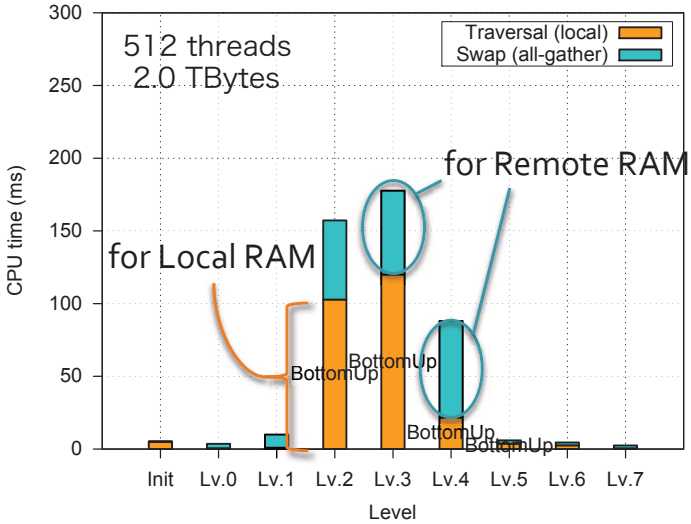
Intel Xeon E5-4650 @ 2.70GHz
64-threads = 8-cores x 2-threads x 4-nodes
512 GBytes RAM

Out of memory

Median TEPS: 37.70 GE/s

SGI Altix UV1000 (Westmere-EX arch.)

Intel Xeon E7-8837 @ 2.67GHz
512-threads = 8-cores x 64-nodes
4.0 TBytes RAM



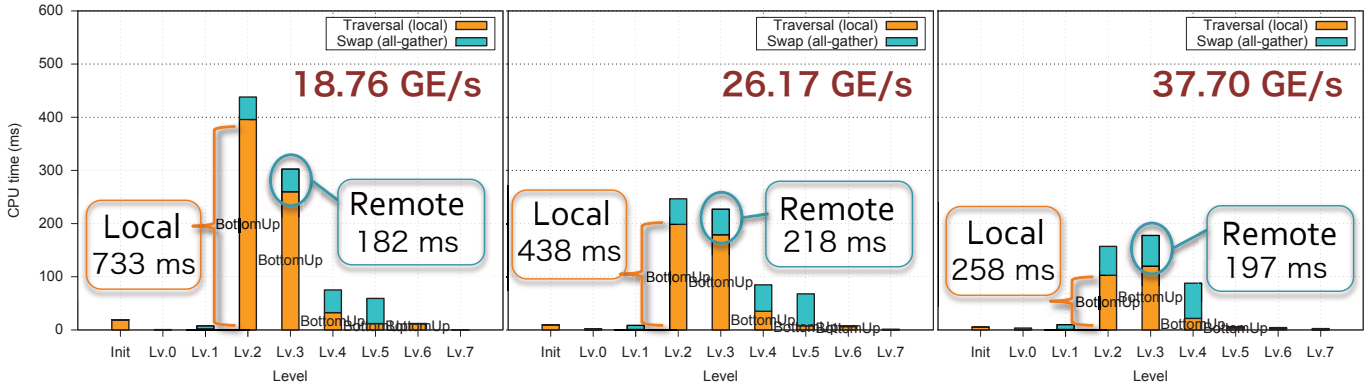
Strong scaling on SGI Altix UV1000

- **Graph500 (SCALE30)**
  - 1.07 billion vertices, 17.18 billion edges

Rank.50
Fastest of single-node
on current list

- **As the number of threads increases,**
  - local traversal on Local memory : **Improvement**
  - swap operation on Remote memory : **Keep**

L : R = 80% : 20%　　L : R = 67% : 33%　　L : R = 57% : 43%

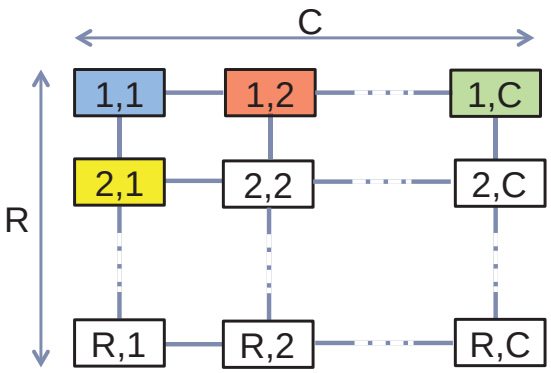


128 threads　　256 threads　　512 threads (one-rack)

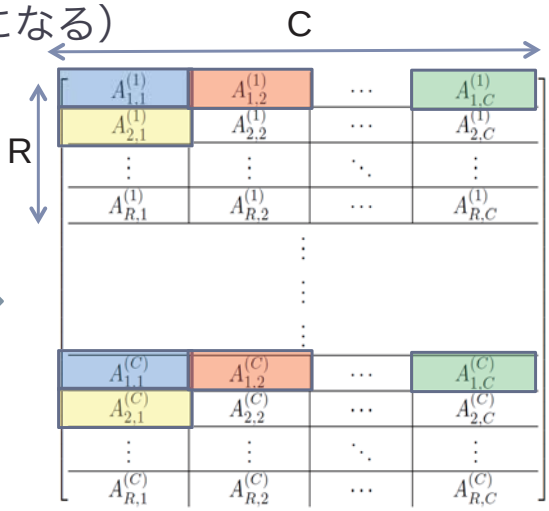
隣接行列の２次元分割※

※Andy Yoo, et. all, *A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L.* SC05.

- 2次元分割BFSでは深さ1レベルの計算で2つの通信フェーズがある
  - Expand: Fronter頂点を**行方向**で共有
  - Fold: Neighbors情報を**列方向**で通信
  - （Bottom-upでは行と列が逆になる）



プロセッサの２次元配置



隣接行列の２次元分割

頂点濃度に応じたデータ形式選択による通信データ量削減

- BFSで通信するデータはFrontierの頂点や未訪問頂点
- 主に２種類のデータ形式が考えられる

頂点セット 1, 4, 6 を表すデータ形式

1,4,6	0100101
Sparse vector	Bitmap

- Frontierや未訪問頂点数が多い＝頂点濃度が高い
  - Bitmapの方が有利
- Frontierや未訪問頂点数が少ない＝頂点濃度が低い
  - Sparse vectorの方が有利

頂点濃度に応じたデータ形式選択による通信データ量削減

- トップダウンやボトムアップフェーズにこだわらず、頂点濃度によってBitmap と Sparse vectorから最適なデータ形式を選択

Graph500グラフにおける典型的なBFS探索におけるデータ形式選択

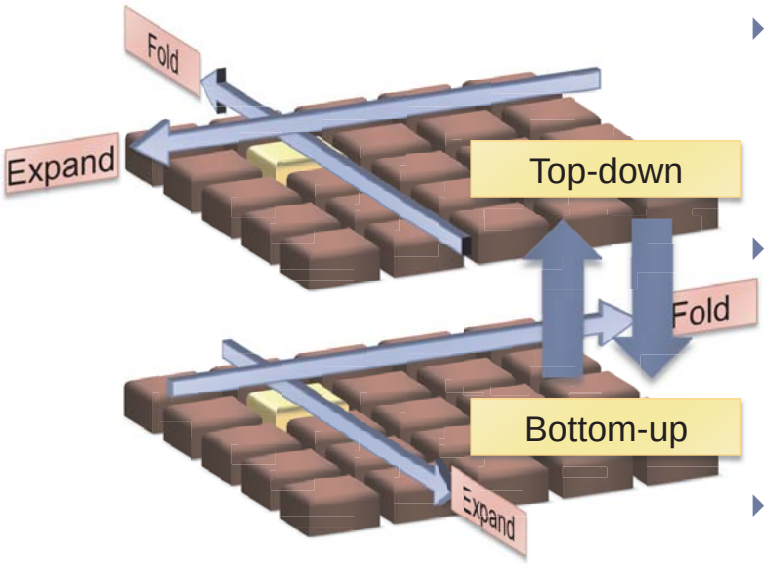
Level	Expand	Fold	Direction
1	Sparse vector	Sparse vector	top-down
2	Sparse vector	Sparse vector	top-down
3	Sparse vector	Bitmap	bottom-up
4	Bitmap	Bitmap	bottom-up
5	Bitmap	Sparse vector	bottom-up
6	Sparse vector	Sparse vector	top-down
7	Sparse vector	Sparse vector	top-down
8	Sparse vector	Sparse vector	top-down

Level5では、フロンティア頂点が全頂点の84.5%なので、bottom-upが有利だが、未訪問の頂点はわずか2.4%なので、Sparse vectorで処理したほうがいい

Level 5における通信データ量比較

Bitmap	Sparse vector
32 MB	**4.5 MB**

Efficient Hybrid Search with 2D Partitioning



: Graph data owned by a single processor

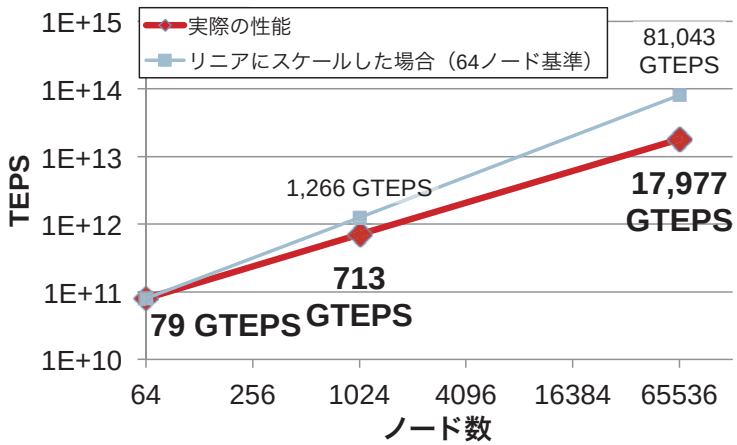
Sharing the same graph data between two direction search.

- Our approach is based on the Top-down and bottom-up hybrid search BFS. [Beamer2011, 2012]
- We realized the hybrid search without any increase of memory footprint.
  - Graph data is shared between two search directions.
- Overlapped communication with computation.
  - Both top-down and bottom-up utilize overlapping communication.

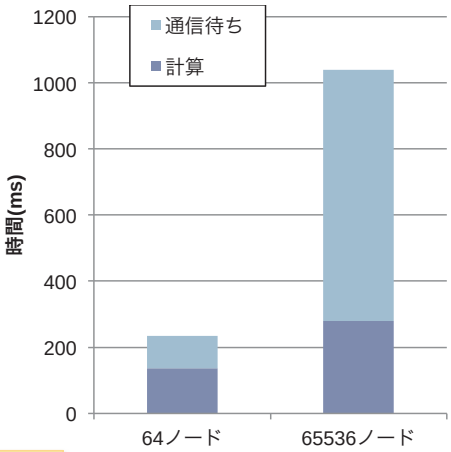
「京」における性能評価

- Graph500ベンチマークで用いるグラフを使用
- 65,536ノードで17,997GTEPSを達成
  - ただし、通信時間の増加はまだ最適化の余地あり
- 各最適化の効果の検証は今後の課題とする

ウィークスケーリング



ブレークダウン



問題サイズは65536ノードでScale40

The Graph500

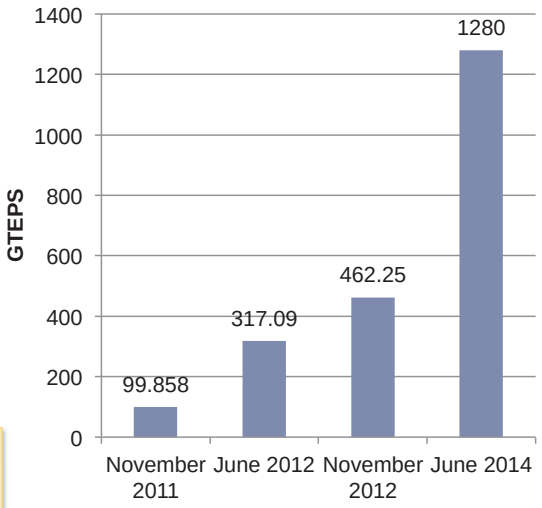
K Computer and TSUBAME 2.0 & 2.5

Graph500 ranking history for TSUBAME2.0 and 2.5

List	Rank	GTEPS	Implementation
November 2011	4	99.858	Top-down only
November 2012	20	462.25	GPU
June 2014	12	1280	Efficient hybrid
November 2014	??	1345	Efficient hybrid

*Every score is obtained using TSUBAME2.0 1366 nodes or TSUBAME 2.5 1024 nodes

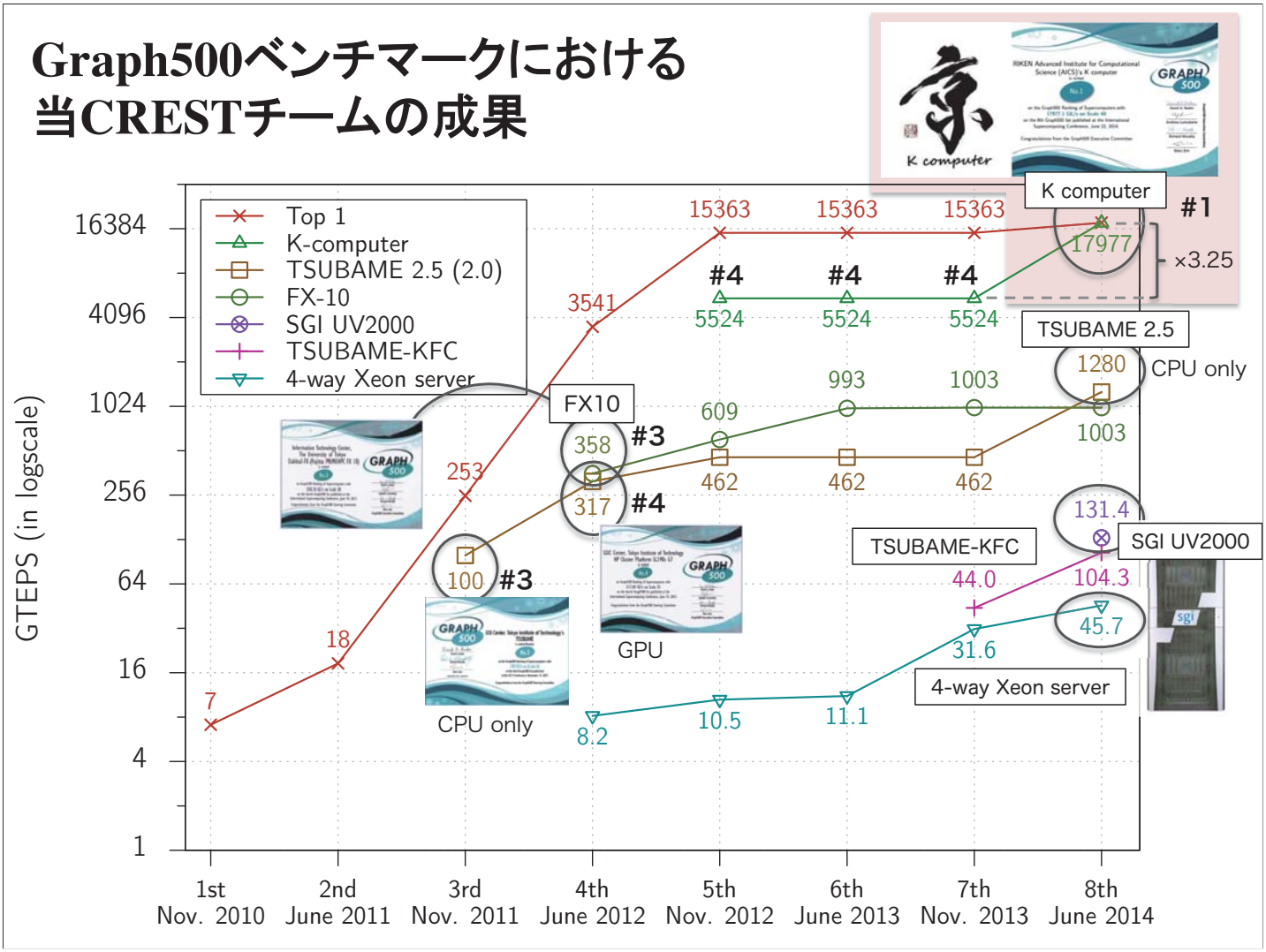
BFS performance on TSUBAME2.0 and 2.5

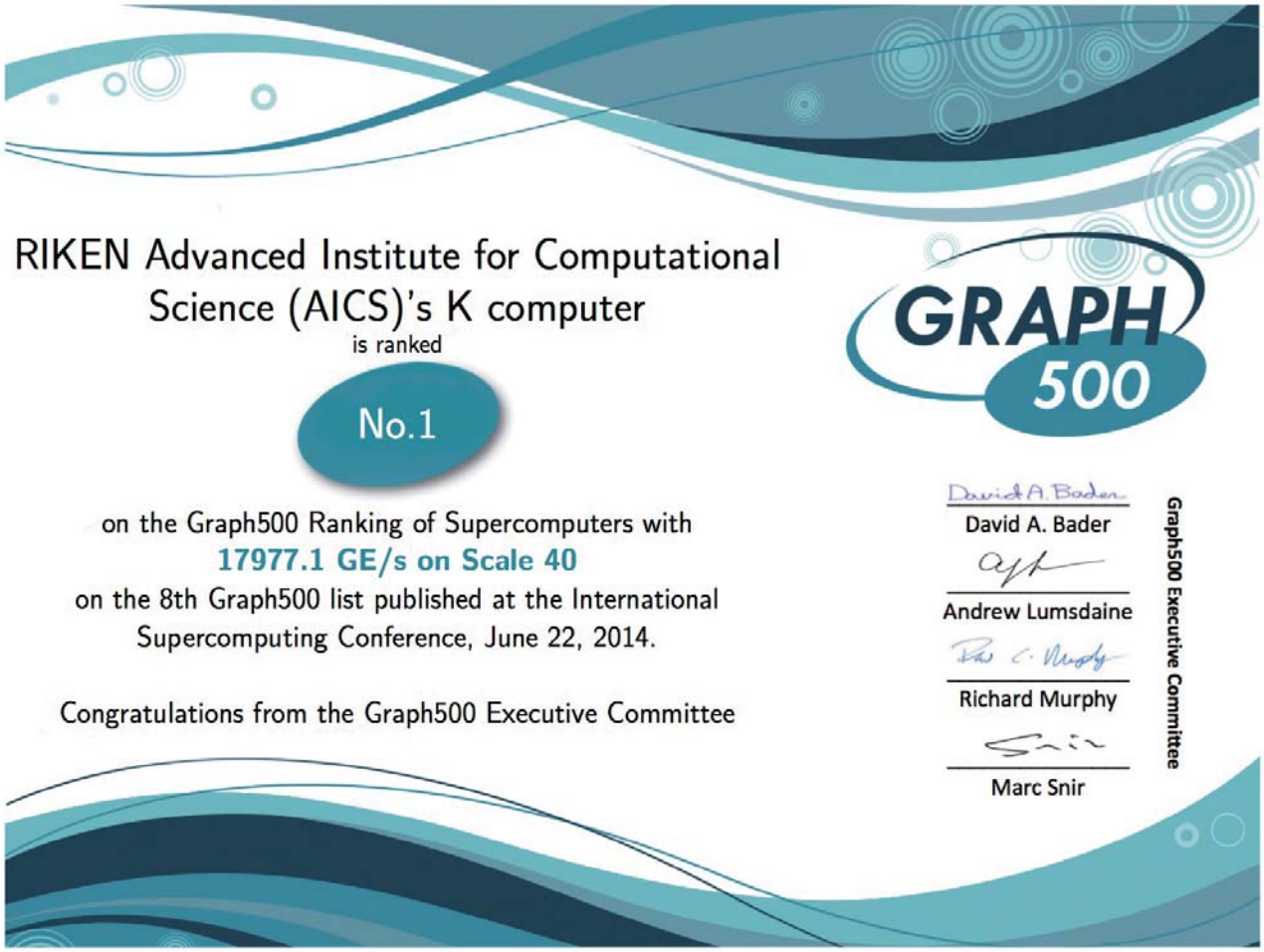


Graph500 ranking history for K Computer

List	Rank	GTEPS	Implementation
November 2013	4	5524.12	Top-down only
June 2014	1	17977.05	Efficient hybrid

Graph500ベンチマークにおける当CRESTチームの成果





The Graph500 List in June 2014

http://www.graph500.org

- Measures performance using **TEPS** (# of Traversed edges per second) in graph traversal such as **BFS**

	No.	Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
Distributed Memory	1	1	K computer (Fujitsu - Custom supercomputer)	RIKEN Advanced Institute for Computational Science (AICS)	65536	524288	40	17977.1
Distributed Memory	12	12	TSUBAME 2.5 (HP - Cluster Platform SL390s G7)	Global Scientific Information and Computing Center, Tokyo Institute of Technology	1024	12288	36	1280.43
Shared Memory	42	42	ismuv2k2 (SGI - SGI UV 2000)	The Institute of Statistical Mathematics	1	640	32	131.427
Distributed Memory	46	46	TSUBAME-KFC (NEC - self-made)	Tokyo Institute of Technology	32	384	32	104.31
Shared Memory	52	52	GraphCREST-Sandybridge-EP-2.7GHz (HPCTECH Corporation - Intel(R) Xeon(R) CPU E5-4650 @ 2.70GHz (4 sockets))	Kyushu University	1	32	27	45.71

Fastest of multi-node

Fastest of single-node

Fastest of single-server

アフィニティ設定毎の GTEPS と MTEPS/W

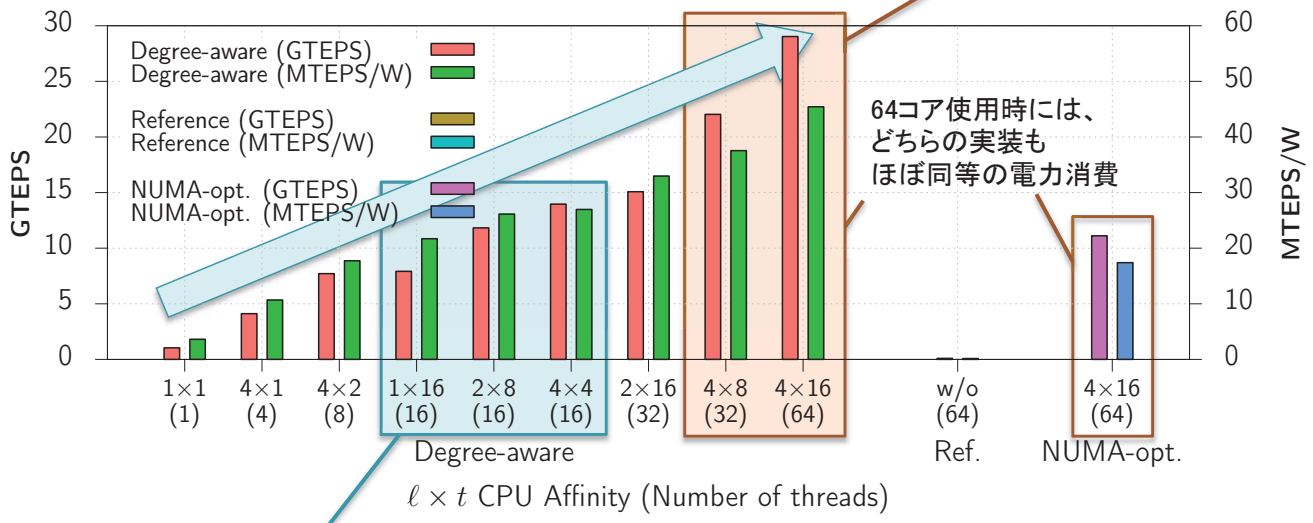
BFS 性能　BFS 電力性能

4-way Intel Xeon E5-4640 2.4GHz 上での数値実験

- BFS 性能 GTEPS と BFS 電力性能 MTEPS/W に共通する性質
  - 同じ**スレッド数**ならば、**ソケット数**は多いほうが良い
  - 同じ**ソケット数**ならば、**スレッド数**は多いほうが良い

Kronecker graph (SCALE27; 1.34 億頂点, 21.48 億枝)

HTコアの使用は
電力性能に効果がある



64コア使用時には、
どちらの実装も
ほぼ同等の電力消費

使用するソケット数を増やすと電力消費も増えるが、
それ以上の TEPS の向上により TEPS/W も改善する

64 スレッド (4-socket x 16-threads) で
29.0 GTEPS と **45.43 MTEPS/W** を達成

Android OS 上での数値実験

- Android NDK (Native Development Kit) の利用
  - オーバヘッドの少ない C/C++ で実装可能(Java は速度的に不向き)
  - GCC をベースとしたクロス・コンパイラの提供
  - OpenMP によるスレッド並列計算 &高速な GCC 拡張のアトミック関数
- Android Developer Tools によるログイン・転送

スマートフォン
SONY Xperia-A-SO-04E
CPU : 4-core Snapdragon
RAM : 2 GB

Step1. Android NDK による
クロスコンパイル
CC=arm-linux-androideabi-gcc

Step2. バイナリの転送
"adb push" コマンド

Step3. Android Developer Tools
によるログイン、Shell による操作
"adb shell" コマンド

Step4. 実行結果の取得
"adb pull" コマンド

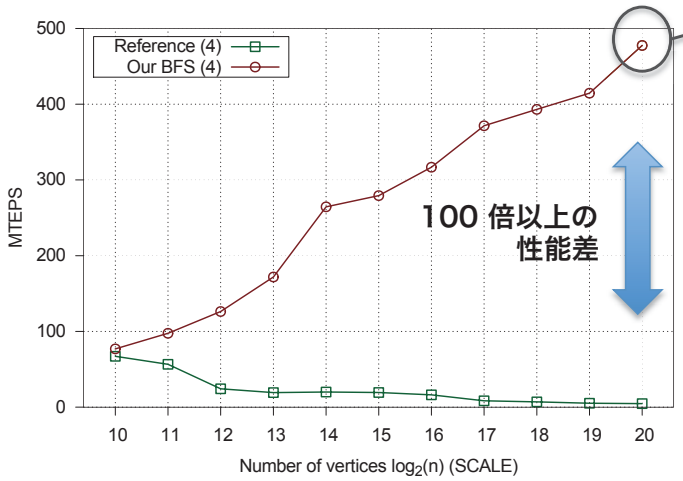
Android Developers: http://developer.android.com.

Xperia-A-SO-04E 上での BFS 電力性能

- **高速 (高性能) な実装が電力消費も良い**

Android OS 端末でもある程度の規模まで扱える

1,048,576 (= 2^{20}) 点
16,777,216 (= 2^{24}) 枝



Implementation	SCALE	MTEPS	watt	MTEPS/W
Reference ($p = 1$)	20	3.25	3.15	1.03
Reference ($p = 4$)	20	4.58	3.22	1.42
This study ($p = 1$)	20	136.29	3.23	42.25
This study ($p = 2$)	20	248.08	2.99	82.92
This study ($p = 4$)	20	477.63	3.12	153.17

使用スレッド数と
電力消費の依存性は低い

The Green Graph500 List in June 2014

http://green.graph500.org

- Measures power-efficiency using TEPS/W ratio

Intel Xeon E5-4640
59.12 MTEPS/W (28.48 GTEPS)

Big Data category

Rank	MTEPS/W	Site	Machine	G500 rank	Scale	GTEPS	Nodes
1	59.12	Kyushu University	GraphCREST-SandybridgeEP-2.4GHz		30	28.48	1
2	48.29	Kyushu University	GraphCREST-Sandybridge-EP-2.7GHz		30	31.95	1
3	35.21	Tokyo Institute of Technology	GraphCREST-Custom #1		31	13.8	1
4	28.88	Tokyo Institute of Technology	MEM-CREST Node #2		30	7.98	1
5	17.24	Kyushu University	GraphCREST-Bulldozer		31	13.63	1
6	14.06	Tokyo Institute of Technology	TSUBAME-KFC		32	104.31	32
7	12.48	The Institute of Statistical Mathematics	ismuv2k2		32	131.43	1

SONY Xperia-Z1-SO-01F
235 MTEPS/W (1.03 GTEPS)

NVIDIA JetsonTK1
204 MTEPS/W (1.63 GTEPS)

Small Data category

Rank	MTEPS/W	Site	Machine	G500 rank	Scale	GTEPS	Nodes
2	235.15	Kyushu University	GraphCREST-Xperia-Z1-SO-01F		20	1.03	1
3	230.41	Kyushu University	GraphCREST-Xperia-A-SO-04E		20	0.74	1
4	204.38	Tokyo Institute of Technology	EBD-GoldenBox-Prototype		21	1.64	1
5	180.76	Kyushu University	GraphCREST-Xperia-A-SO-04E		21	0.59	1
6	171.77	Kyushu University	GraphCREST-Xperia-Z1-SO-01F		21	0.91	1
7	153.17	Chuo University	GraphCREST-Xperia-A-SO-04E	143	20	0.478	1

グラフ解析と最適化の今後

九州大学
「センター・オブ・イノベーション(COI)プログラム」

共進化社会システム創成拠点

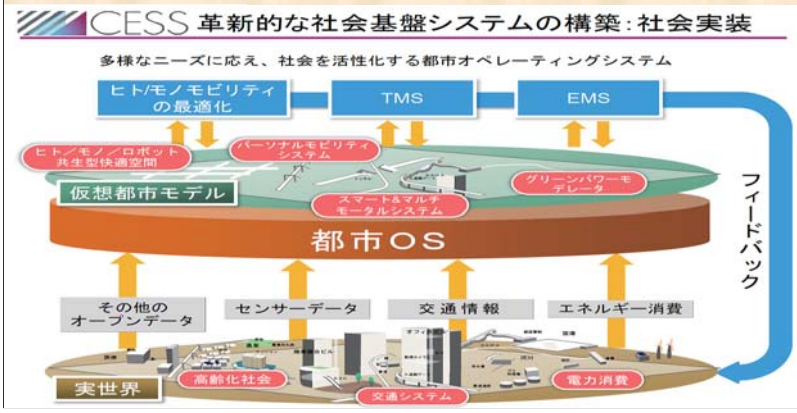
CESS Center for Co-Evolutional Social System
共進化社会システム創成拠点

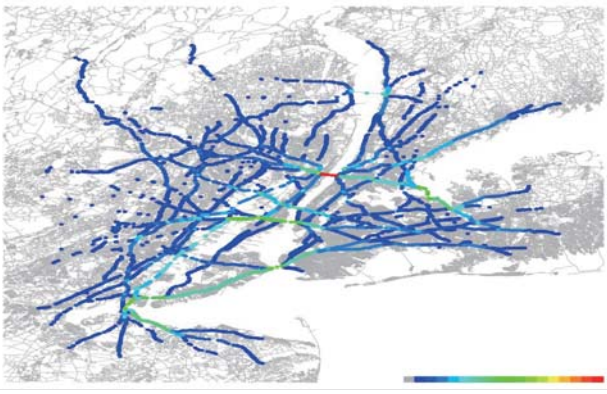
KYUSHU UNIVERSITY

COI STREAM

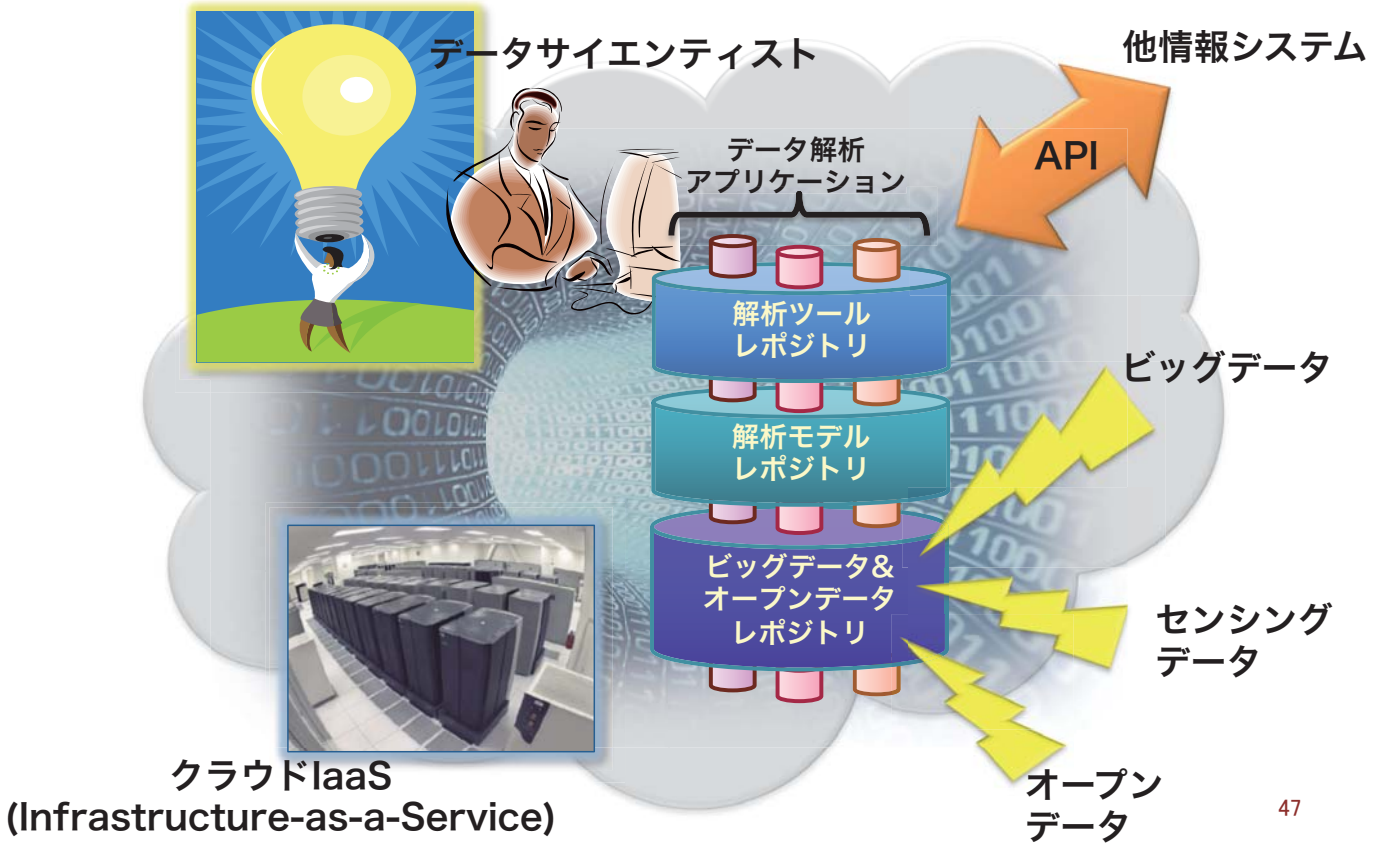
実問題に対する大規模グラフ解析

- 超大規模ネットワークに対する探索アルゴリズムとクラスタリングアルゴリズムの高速実装
- 数百万頂点～数兆頂点、数億枝～数百兆枝からなる超大規模なグラフ解析
  - 数百万人の被災者の避難経路の計算では数千万頂点のグラフ(１スレッドあたりのメモリ要求量1Gbytes)に対して、同時に数百万スレッド単位で各被災者毎の最短路計算と各点の重要度判定
- 新しい産業応用の開拓 → １：ヒト/モノ　２：エネルギー　３：情報のモビリティ→ 数理モデルとスパコンを用いた最適化 (整数計画問題に対する並列ソルバー等)
  - 交通データに対する経路探索動的に変化する交通量等から高速な重要度判定を行い、交通管制等に活用
  - ソーシャルネットワークデータに対する動的な重要度、影響度の判定各点の周辺、および広域内における影響（情報の伝播力）を推定





BODIC.org
(BigData & OpenData in the Cloud)



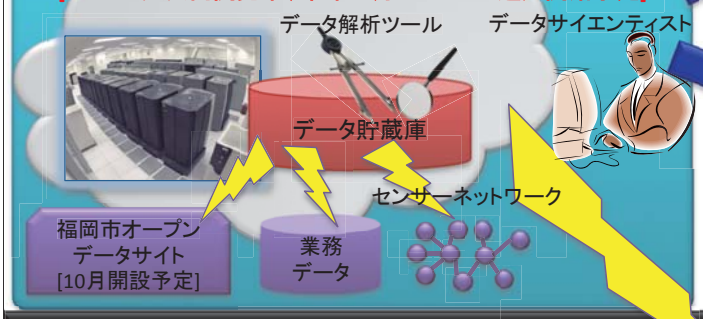
「ビッグデータ&オープンデータを活用した創業」支援策 [20140729PR]

1. 福岡市が有するビッグデータ&オープンデータ活用基盤

BODIK (ビッグデータ&オープンデータ研究会 in 九州)
[福岡市、URC、ISITが共同で昨年12月に発足]

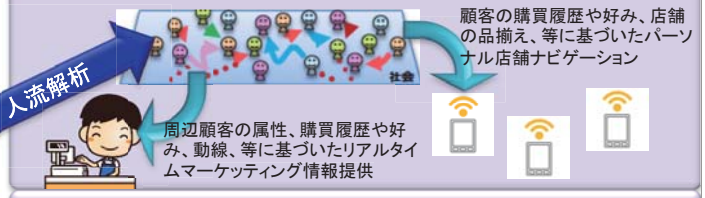
DSS4K (データサイエンススクール for 九州)
[ISITが企画中]

BODIC.org (BigData & OpenData in the Cloud)
[ISIT&九大共同開発中、本年10月からISITが運用開始予定]



3. 人流センシングデータ活用で可能となる多様な創業

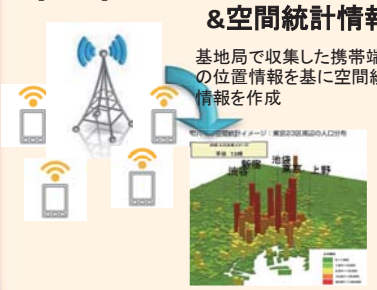
[例1] 店舗向けリアルタイムマーケッティングサービス&顧客向け店舗ナビゲーションサービスビジネス



[例2] スマーターモバイル"XXX"サービスビジネス

2. 規制緩和でもっと[illegible]になる「人流センシング」

[現状] 携帯電話キャリア各社による位置情報収集&空間統計情報提供

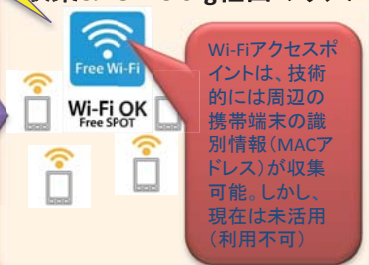


例) NTTドコモ モバイル空間統計

3つの課題
1. キャリアが情報独占
2. リアルタイムでの情報提供は未実施
3. 元データへのアクセス不可(統計情報のみ利用可)

規制緩和

[規制緩和後] Wi-Fiアクセスポイントを用いた位置情報収集&BODIC.org経由のリアルタイムな元データ提供

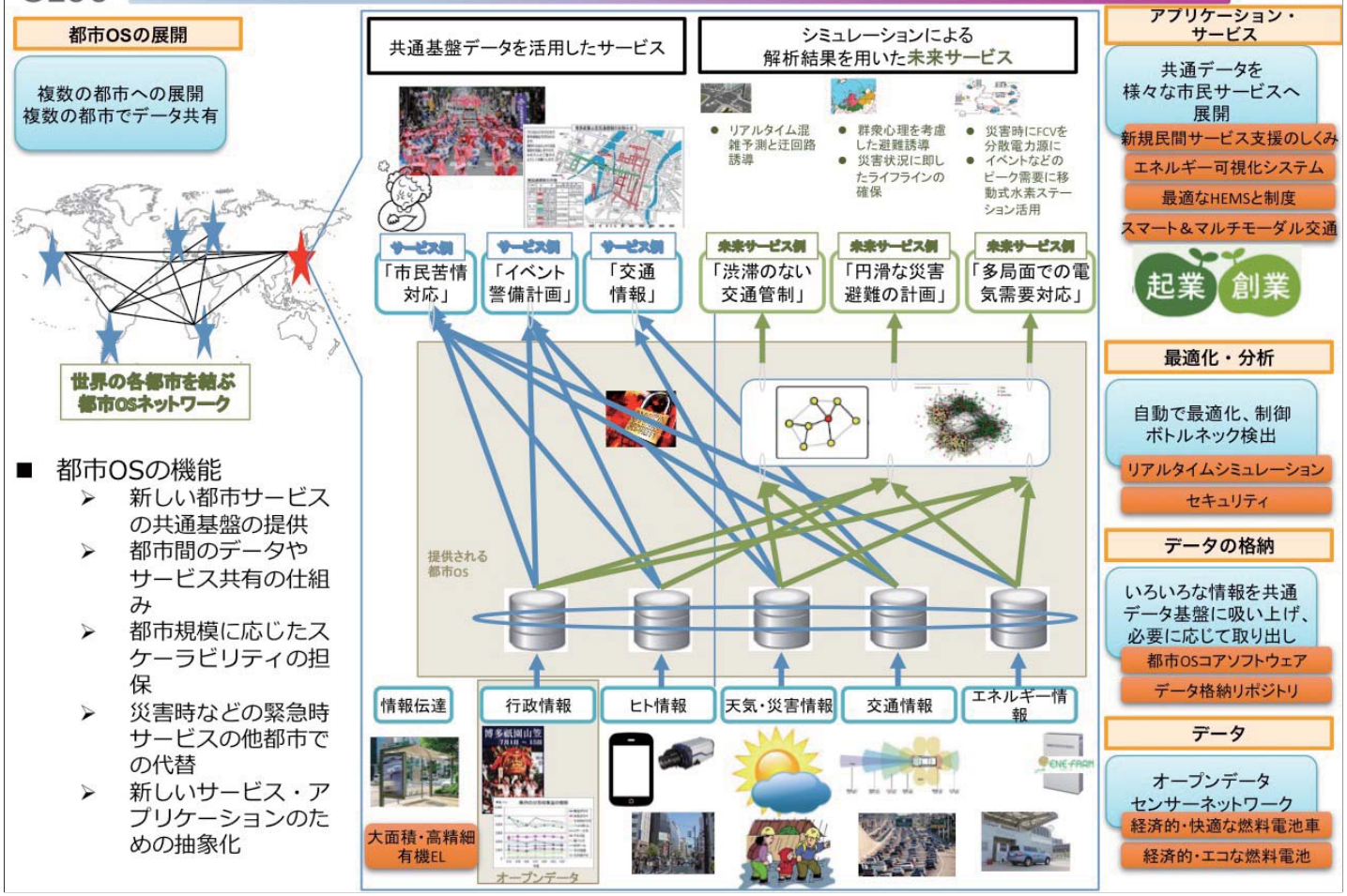


Wi-Fiアクセスポイントは、技術的には周辺の携帯端末の識別情報(MACアドレス)が収集可能。しかし、現在は未活用(利用不可)

2つの規制緩和案
1. Wi-Fiアクセスポイントでの携帯端末識別情報収集を許可
2. 携帯端末所有者の合意の下、収集した識別情報の元データの2次利用を可能に

※九大伊都で実証実験中

都市OSの機能



■ 都市OSの機能
- 新しい都市サービスの共通基盤の提供
- 都市間のデータやサービス共有の仕組み
- 都市規模に応じたスケーラビリティの担保
- 災害時などの緊急時サービスの他都市での代替
- 新しいサービス・アプリケーションのための抽象化

グラフ探索及び数理最適化ライブラリによる大規模グラフ処理

大 ↕ 小　計算量

小 ↕ 大　データ量

上層：NP困難な最適化アルゴリズムの実行

1. MIP(混合整数計画問題)の場合：0-1整数変数の数＝nで計算量はおよそ $O(2^n)$
2. 前処理による変数の削除と並列計算の適用（CPUコア中心の大規模スレッド並列：分枝カット法の適用）
3. データ量は 10^5 以下（整数変数の数）
4. 施設配置問題、集合被覆（分割）問題、スケジューリング問題などの最適化問題

中層：多項式時間最適化アルゴリズムの実行

1. SDP(半正定値計画問題)の場合：n＝行列の大きさ, m＝制約式の数で計算量はおよそ $O(n^3+m^3)$
2. 疎性の追求と前処理さらに並列計算の適用（CPUコア中心の大規模スレッド並列が中心だが、今後はCPU＋GPUによる高速化）
3. データ量は 10^8 以下(非負変数の数)：10^6 以下(制約条件数)
4. グラフ分割、センサーネットワーク、サポートベクターマシンなどの最適化問題

下層：グラフ解析アルゴリズムの実行

1. ダイクストラ法(1対全最短路問題：ヒープ付き)の場合：n＝点数, m＝枝数で計算量はおよそ $O((n+m)\log n)$
2. グラフ探索の局所的な評価では優先キュー(ヒープ木)を用いる⇒実行時間、メモリ消費量が安定的（CPUコア中心の大規模スレッド並列が中心だが、高速ストレージ技術による超大規模グラフ処理）
3. データ量は $10^{12\sim14}$ 以下（グラフの点数と枝数）
4. 最短路計算、ネットワーク内での各点の重要度を推定。各点の周辺、及び広域内における影響（情報の伝播力）を計算

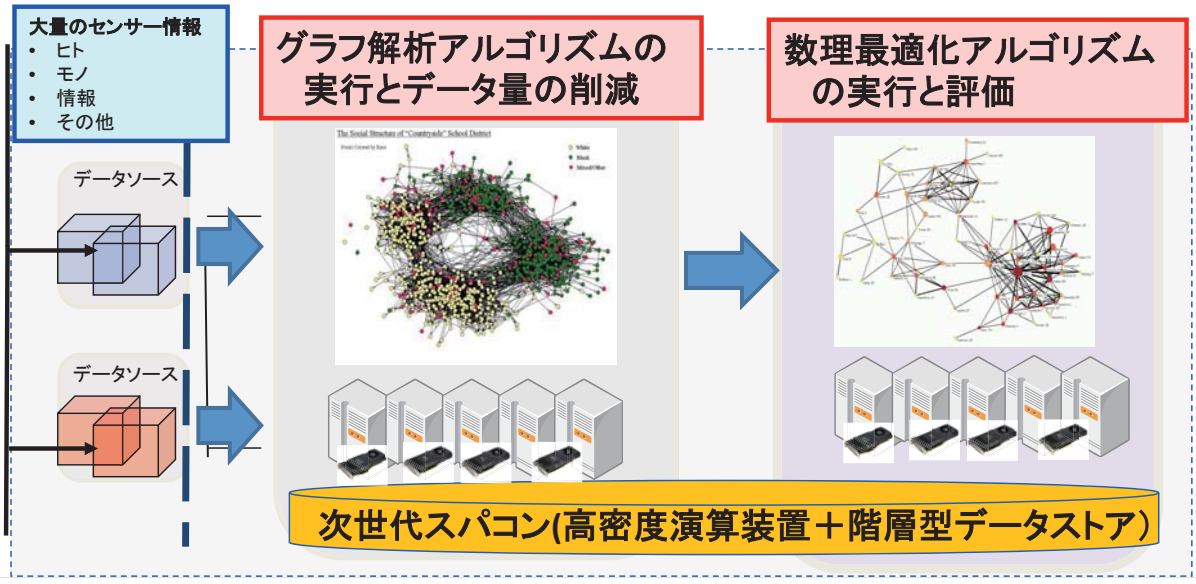
下層：グラフ解析アルゴリズムの実行

1. ダイクストラ法(1対全最短路問題：ヒープ付き)の場合：n＝点数, m＝枝数で計算量 O((n＋m)log n)
2. グラフ探索の局所的な評価では優先キュー(ヒープ木)を用いる⇒実行時間、メモリ消費量が安定的（CPUコア中心の大規模スレッド並列が中心＆高速ストレージ技術による超大規模グラフ処理)
3. データ量は $10^{12\sim14}$ 以下（グラフの点数と枝数)
4. 最短路計算、ネットワーク内での各点の重要度を推定。各点の周辺、及び広域内における影響（情報の伝播力)を計算

- BFS(幅優先探索) → Graph500 : 京コンピュータ上で Scale 40 (2^{40}点, 2^{44} 枝）のグラフに対して 18 TeraTEPS の性能 (Top-down & Bottom-up & Level Synchronized BFS)
- Shortest Path (最短路) → IBM BG/Q 上で Scale 38 (2^{38}点, 2^{42}枝）のグラフに対して 3.1TeraTEPSの性能(Delta Stepping & Bellman Ford のハイブリッド) → 静的なグラフに対してはHighway hierarchies などの前処理が有効
- Maximum Flow (最大フロー) → Push Relabel 法の実装 → 最速フローに適用 →約 4,588万点, 18,537万枝のグラフに対して最大フローの 4～5分の実行時間
- Centrality (中心性) → 前処理や並列計算等も有効
  - 各点に接続されている枝数（Degree Centrality）
  - 各点が最短路に含まれる回数（Betweenness Centrality）
  - 最も遠い点までの最短距離（Graph Centrality）
  - 各点までの最短距離（Closeness Centrality）
- その他のアルゴリズムとツールも多数存在
  - Minimum spanning tree ： 全域木
  - Strongly connected component ： 強連結成分
  - Spectral clustering ： グラフの行列表現(Laplacian Matrix) → 固有値計算 → クラスタリング
  - Hyper ANF : Neighborhood 関数の近似 ： 点 x から t hop で到達する点 y → (x,y) の計算

グラフ探索及び数理最適化ライブラリによる大規模グラフ処理基盤

- 大規模センサーから到着するストリーミングデータに対して精緻な解析を実現する大規模グラフ処理基盤を開発する
- 大規模グラフ処理基盤は以下の処理系から構成される
  - グラフ解析アルゴリズムの実行： 最短路計算、ネットワーク内での各点の重要度を推定。各点の周辺、及び広域内における影響（情報の伝播力)を計算 → 重要な要素を失うことなくデータ量を削減する
  - 数理最適化アルゴリズムの実行： 施設配置問題、集合被覆（分割）問題、スケジューリング、配送計画問題などの数理最適化問題 → モビリティに関する改善提案
  - 大規模なセンサーデータを高速かつ重要性を失うことなく縮約することよって、精緻な解析を実現
- 計算及びデータ蓄積の基盤となる次世代スパコン
  - 並列数の爆発的増大、不均質化、高密度化 ＆ 記憶装置の多階層化・大容量化



数理計画問題（最適化問題）と2015年予想（目標）

- **非常に応用が広範（企業、社会、公共政策）**→ 高性能なソルバーを作ること自体が最適化問題
- センサーデータによる最適化問題の複雑＆巨大化
- **半正定計画問題(SDP)**と**混合整数計画問題(MIP)**が2大注目数理計画問題
- **汎用ソルバーの必要性**（個別の問題に対する仮定やチューニングは効果が低い）
- 入力は**疎データ**と**密データ**の混合
  - **疎データ**：多数のCPUコアによる処理（浮動小数点演算性能に非依存：密データへ変換）→ **GPU 系による処理**
  - **密データ：GPU 系による処理**
  - 理論的性能限界等からボトルネック箇所を特定
  - 数値演算能力とメモリバンド等のトレードオフ関係を把握
  - 計算量とデータ移動量の正確な推定、疎性やサイズなどのデータ特性と性能値の見極め
- 計算量とポストペタスパコンでの想定：
  - **計算量$O(n^3)$ → 数百万程度, $O(n \log n)$ → 100億以上, $O(n)$ → 100兆以上**