4. 各会合での情報提供資料

本 WG の各会合にて、以下の情報提供を実施した。次ページ以降に資料を掲載する。

No	実施日	タイトル	情報提供者（敬称略,所属は当時）
1	2011/08/23	Japan's Grand Challenge Activities in Life Science	姫野　龍太郎（理化学研究所）
2	2011/10/28	多倍長計算の紹介と K-computer 向け固有値ソルバ「Eigen-K」について	今村　俊幸（理化学研究所計算科学研究機構）
3	2011/10/28	高精度線形代数演算ライブラリ MPACK の紹介	中田　真秀（理化学研究所）
4	2012/03/01	宇宙プラズマでのペタスケール数値計算(惑星磁気圏シミュレーション)	深沢　圭一郎（九州大学）
5	2012/03/01	ペタスケールのアプリケーションへ向けて	町田　昌彦（日本原子力研究開発機構）
6	2012/07/23	JAXA における数値計算ライブラリの利用状況と OPL 適用評価	松尾　裕一（宇宙航空研究開発機構）
7	2012/07/23	並列擬似乱数発生アルゴリズムとペタスケール時代のモンテカルロ法の展望	三浦　謙一（国立情報学研究所）
8	2012/07/23	FFTE について	高橋　大介（筑波大学）
9	2013/02/22	フラグメント分子軌道（FMO）法と並列 FMO プログラム OpenFMO について	稲富　雄一（九州大学）

Japan’s Grand Challenge Activities in Life Science

日本における生命科学分野でのグランドチャレンジ

Computational Science Research Program, RIKEN

理化学研究所、次世代計算科学研究開発プログラム、副プログラムディレクタ

Ryutaro Himeno

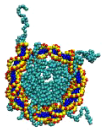
姫野龍太郎

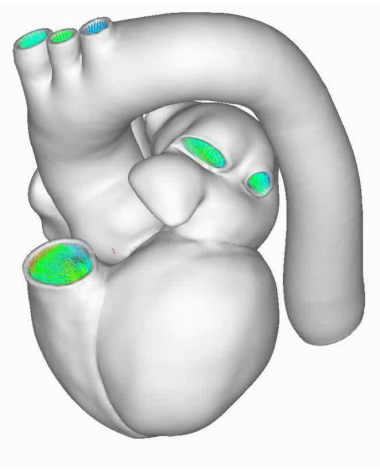
概要

次世代スーパーコンピュータ京を使って、その性能がフルに発揮できるソフトウェアを開発し、それ以前では不可能だった計算を可能にするための生命科学分野でのグランドチャレンジを 2006 年 10 月から理化学研究所を拠点に開始した。このプロジェクトには全国の１３機関から約２００名の研究者が参加して 2013 年３月末まで行われた。この間、分子スケール、細胞スケール、臓器全身スケール、大規模生命化学系データ解析、脳神経系など多岐にわたるアプリケーションプログラムと、大規模可視化ツールや並列化ツールなど、合計３４本を開発していた。これらの開発状況と並列性能を紹介した。これらのソフトのうち、この時点では１１本が京の上で実行できるようになり、十分な並列性能を達成していた。さらに５本は京の上で非常に良い性能を出していた。特に分子動力学プログラム cppmd は京で 1 PFLOPS を超える性能を達成した。

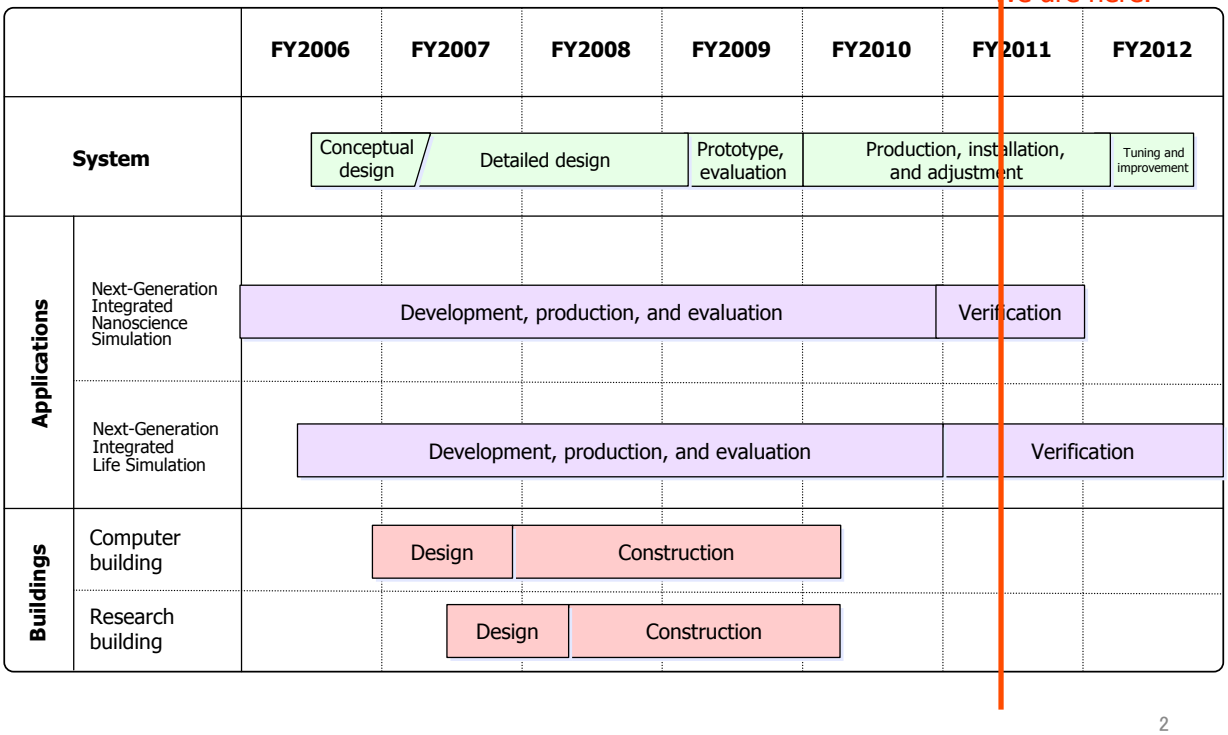
Japan's Grand Challenge Activities in Life Science

Ryutaro Himeno
Next-Generation Integrated Simulation of Living Matter
Computational Science Research Program
RIKEN





Schedule of the Next-generation supercomputer R&D project



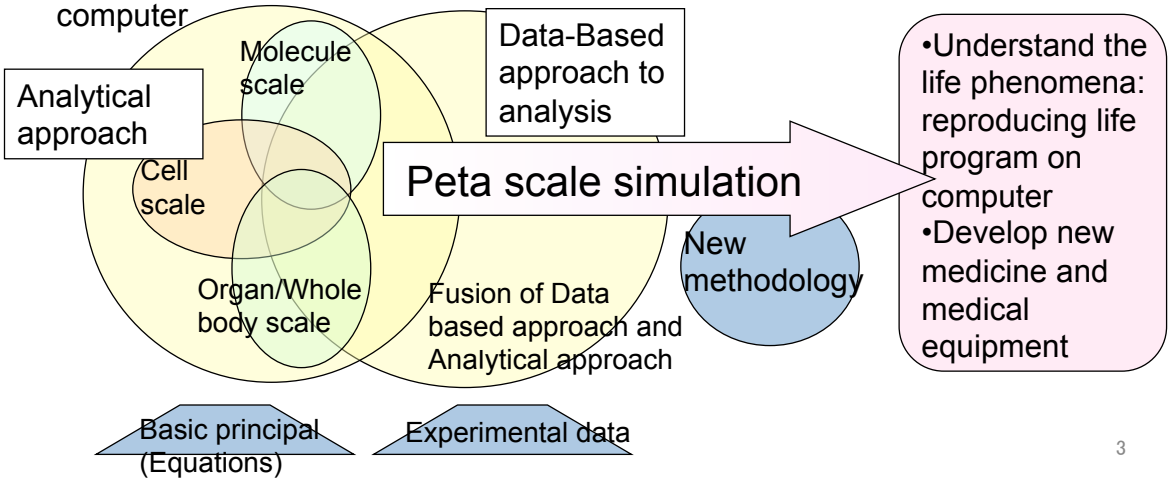
Total Simulation of Living Matter

Goal

Understand the life phenomena: reproducing life program on computer
Develop new medicine and medical equipment

Approach

combination of analytical approach and Data-driven approach on Peta-scale computer



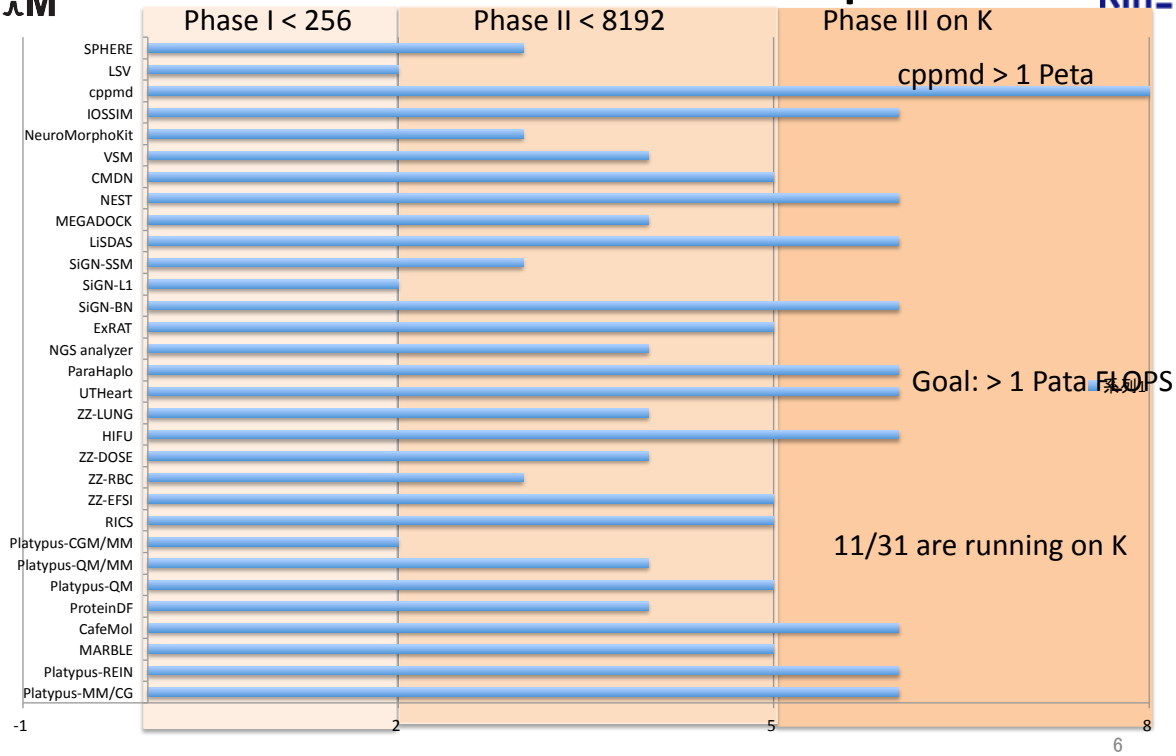
List of developing software 1/2

scale	ID	Software Name	Discription
Molecule scale (9)	M-4	CafeMol	The coarse-grained MD program
	M-3	MARBLE	The all-atom MD program
	M-9	Platypus-CGM/MM	The coupled coarse-grain model (CGM)-MM calculation
	M-1	Platypus-MM/CG	The class library foundation for the development of multi-copy and multi-scale molecular simulations
	M-7	Platypus-QM	The quantum chemistry calculation program.
	M-8	Platypus-QM/MM	The coupled quantum mechanics (QM)-molecular mechanics (MM) calculation.
	M-6	Platypus-QM/MM-FE	The hybrid QM/MM reaction free energy calculation.
	M-2	Platypus-REIN	The replica exchange molecular dynamics (MD) interface.
	M-5	ProteinDF	The all-electron wave function solving program for a protein by the density-functional approach.
Cell scale (1)	C-1	RICS	The cell simulation integrated platform.
Organ scale (6)	O-4	HIFU	The simulation code for HIFU: High Intensity Focused Ultrasound
	O-6	UT-Heart	The multiscale, multiphysics heart simulator.
	O-3	ZZ-DOSE	The voxel Monte Carlo heavy-particle-dose analysis program in a whole body.
	O-1	ZZ-EFSI	Fluid-structure interaction analysis program for whole body voxel simulation.
	O-5	ZZ-LUNG	The simulation program for lung breathing and circulation.
	O-2	ZZ-RBC	The microcirculation simulator with the immersed boundary method.

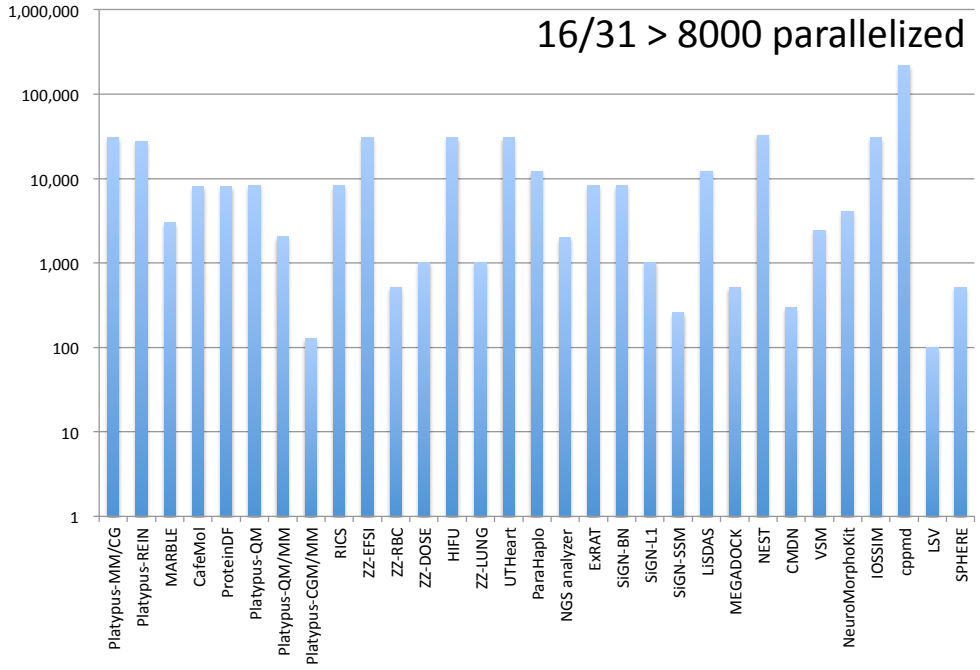
List of developing software 2/2

9	Data-driven	D-3	ExRAT	The whole genome association analysis code for a two SNPs combination calculation using the extended RAT method.
		D-8	LiSDAS	The life science data assimilation systems.
		D-9	MEGADOCK	The exhaustive protein-protein interaction network prediction tools based on 3D conformation data.
		D-2	NGS analyzer	The next-generation genome sequencer analysis program
		D-1	ParaHaplo	The program package for haplotype-based whole-genome association study using parallel computing.
		D-7	SBiP	The data analysis fusion platform.
		D-4, 5, 6	SiGN	The large scale gene network estimation software series.
5	Brain & nerve	B-2	CMDN	The cortical microcircuit simulator developed on NEST.
		B-5	IOSSIM	The whole-brain simulator for an insect's olfactory system.
		B-1	NEST	Neural Simulation Tool
		B-4	NeuroMorphoKit	The neuron configuration simulation kit.
		B-3	VSM	The visual information-processing analysis with a whole-visual-system model: the common platform for the visual-system simulation.
4	HPC	H-1	cppmd	MD core program for large scale parallel computing
		H-2	LSV	The distributed-parallel large-data visualization system.
		H-3	SPHERE.	An application middleware for a next-generation supercomputer
		H-4	VLSVL	The large scale virtual library of syntheticable chemical structures and reaction schemes.

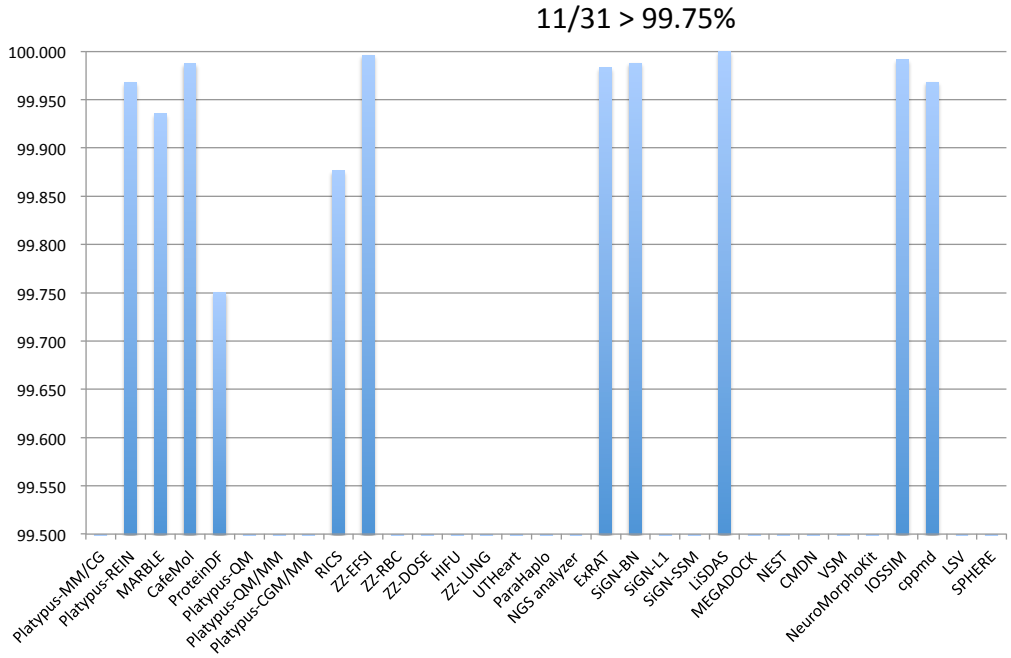
Current status of development



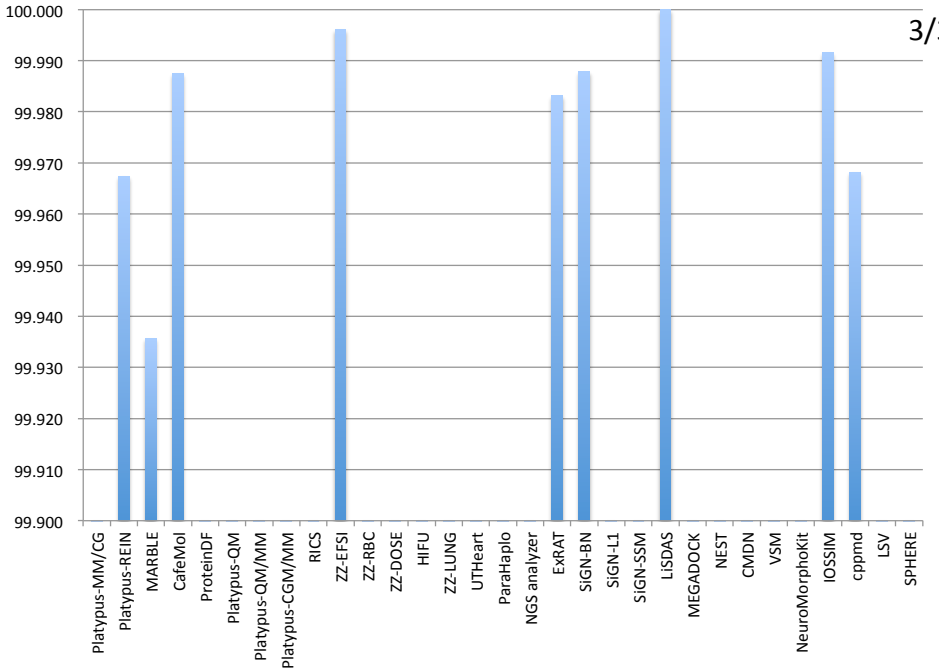
Maximum No. of parallel execution



Parallelized ratio (1)

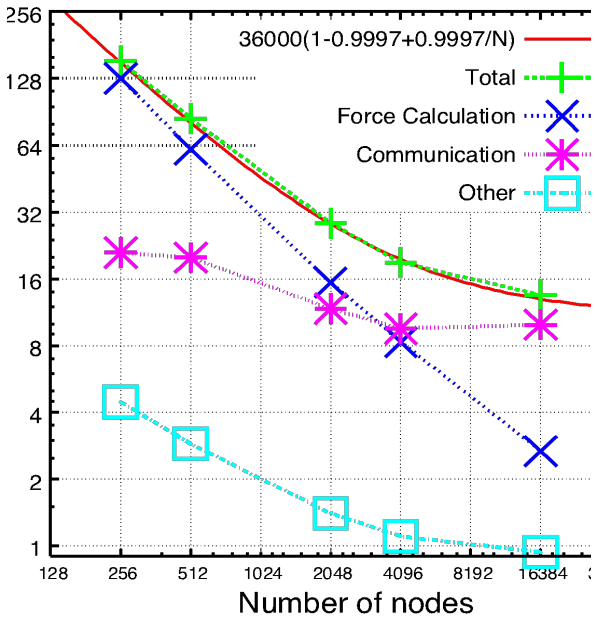


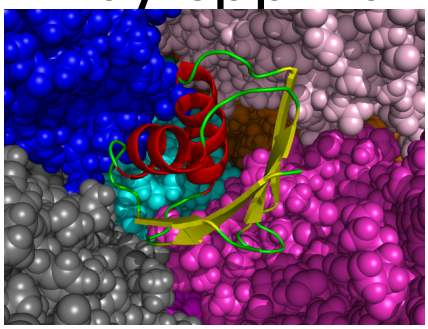
Parallelized ratio (2)



3/31 > 99.99%

Strong scaling shown by cppmd





Model	432 *in vitro*
Number of atoms	180,881,424
Cutoff radius	28 Å + 2 Å margin
Number of pairs per atom	8,835(+ 2,031 in margin)
FLOP counts for 1,000 steps	202,851,990,144 MFLOP
Topology of computation node	24 × 24 × 48
Number of nodes	27,648
Number of cores	221,184
Theoretical peak performance	3.539 PFLOPS
Calculation time for 1,000 step	154.29 sec
Sustained performance	**1.315 PFLOPS**
Efficiency	0.3716

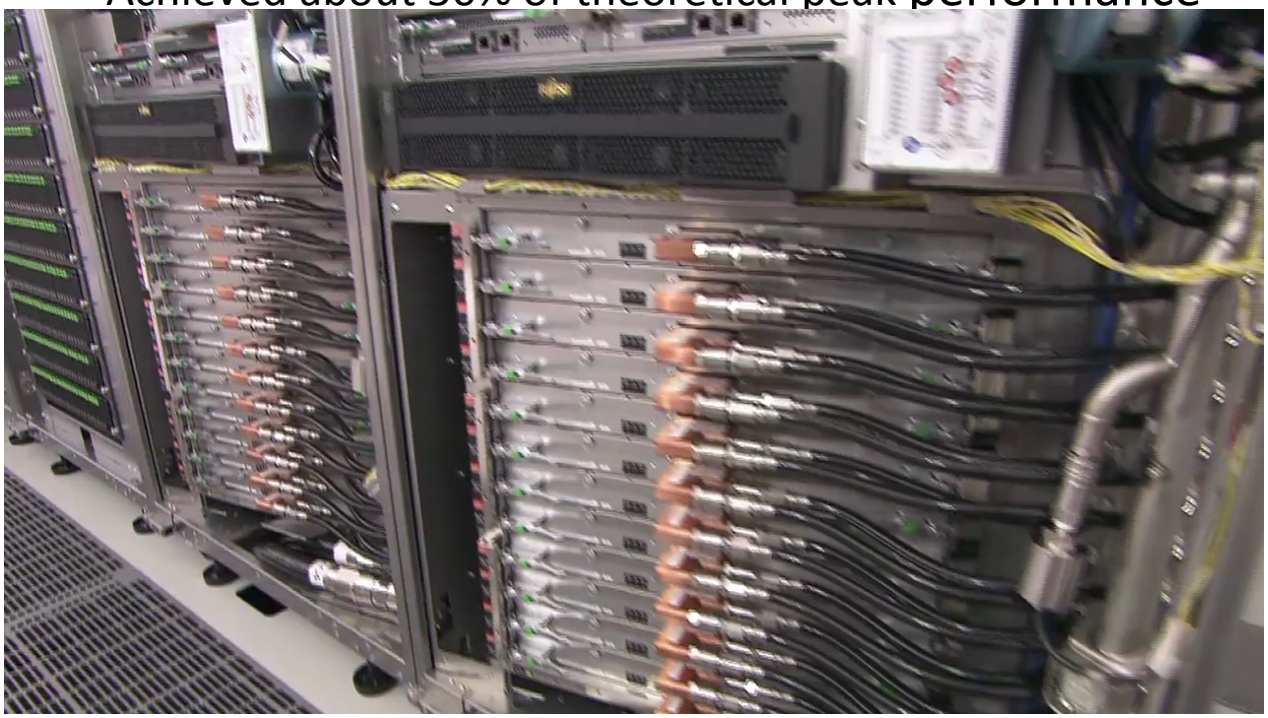
Measured At the end of March, 2011

Library used in our developing software package

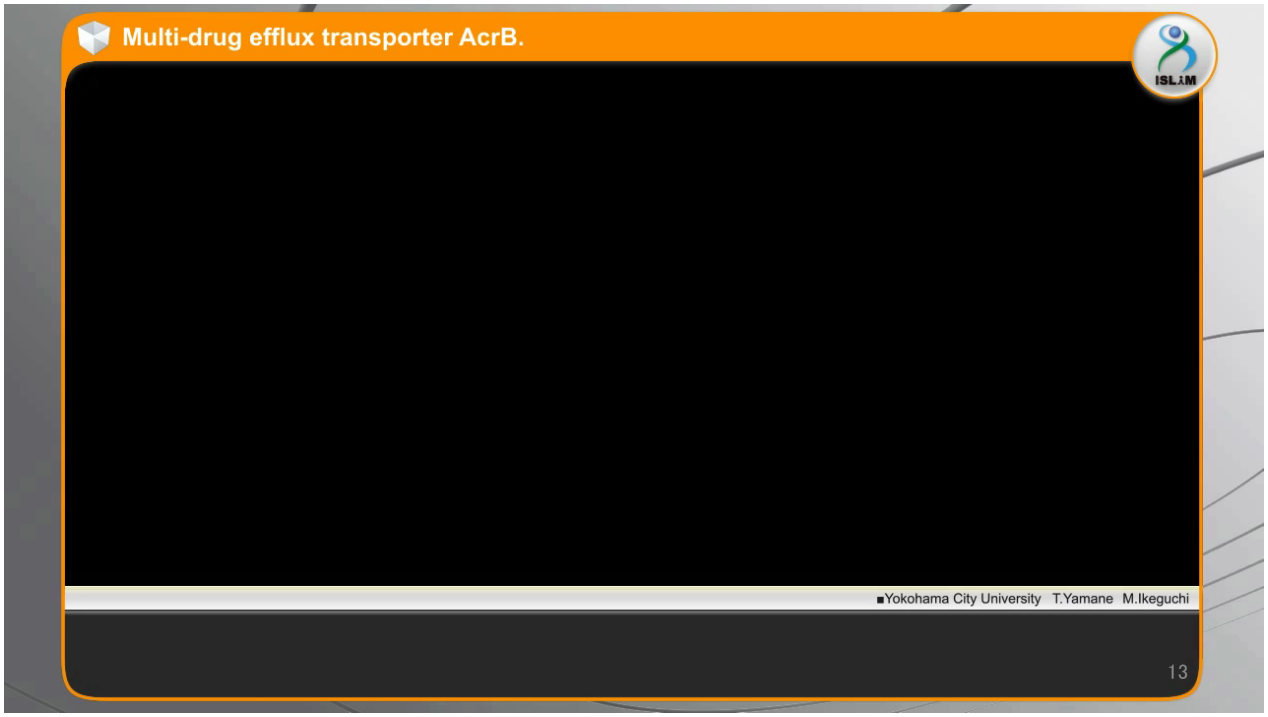
	FFTW	NetCDF	HDF	Lapack	ScaLapack	GSL	others
Marble	o						
Platypus	o	o	o	o	o		
ProteinDF					o		
RICS						o	Boost
HIFU							SHERE
ZZ-EFSI							SHERE
Megadock	o						
CMDN						o	
IOSSIM							Sundail InterView
EuroMorpho Kit		o				o	GD, zlib
VSM		o				o	
LSV							Libxml2, zlib
SHERE							Libxml2, zlib

Molecular Dynamics Simulation Code: cppmd

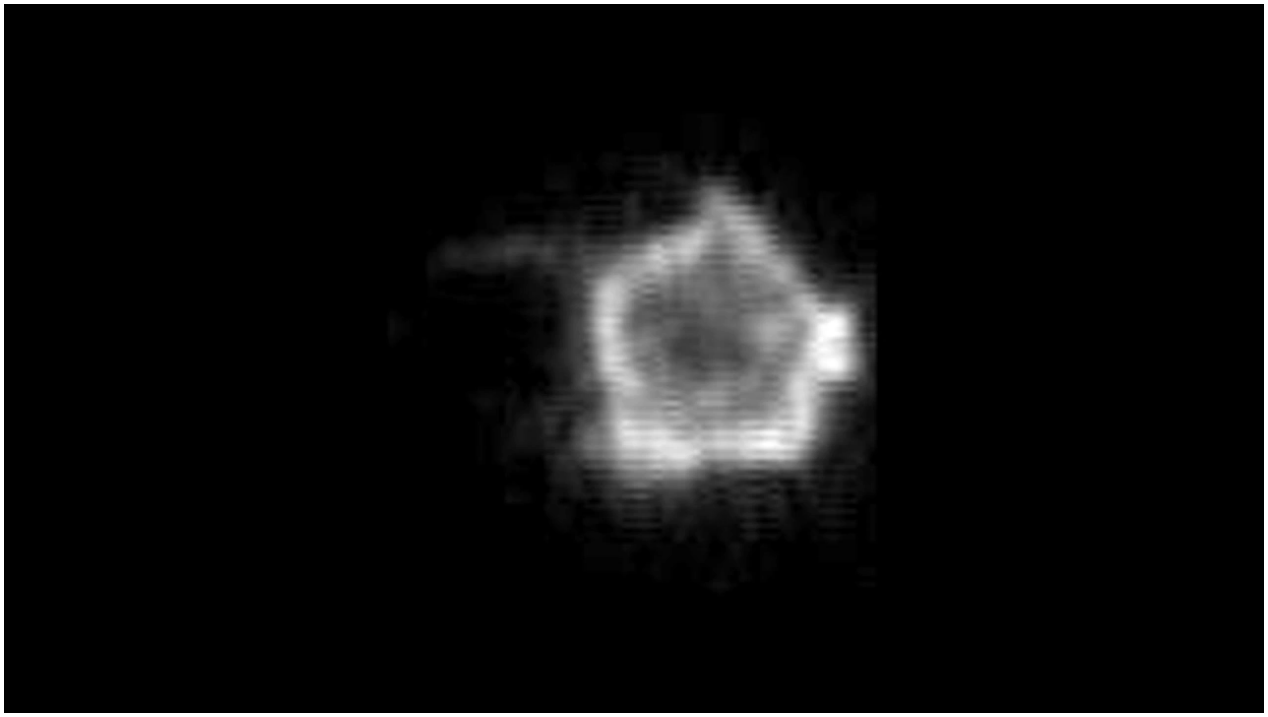
- Designed for K computer
- Achieved about 50% of theoretical peak performance



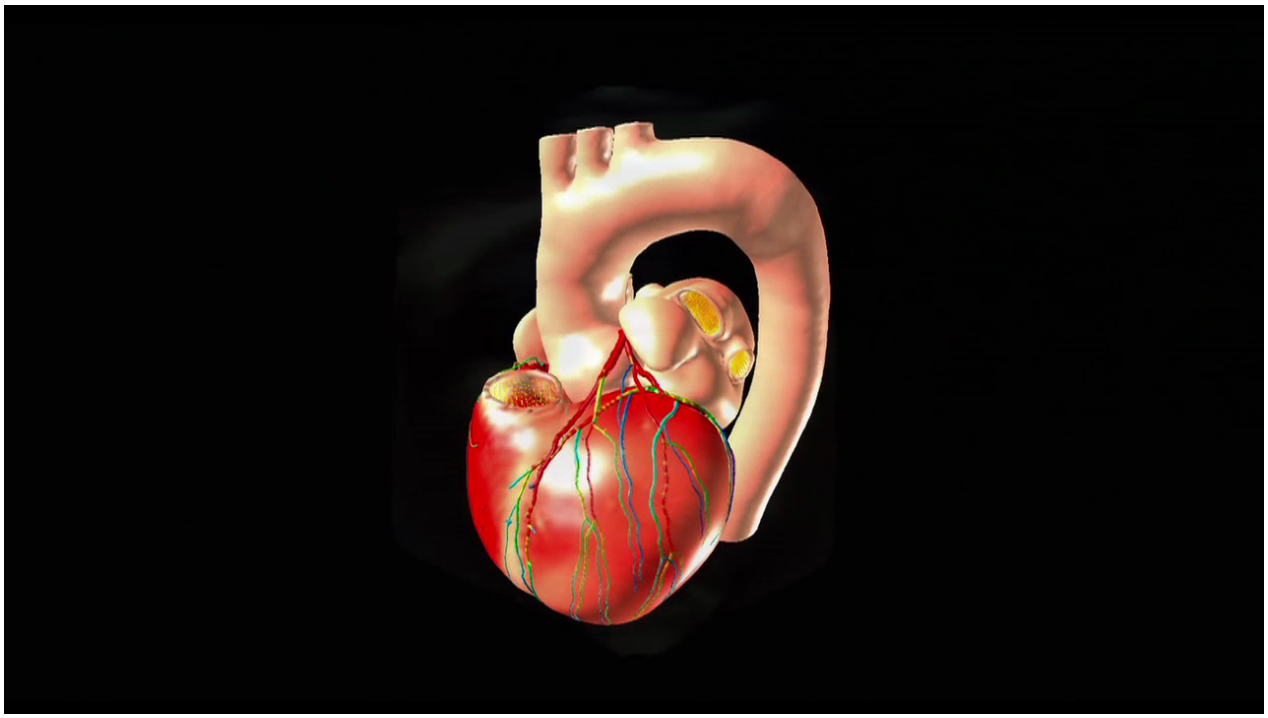
Molecular simulation of multi-drug efflux transporter AcrB



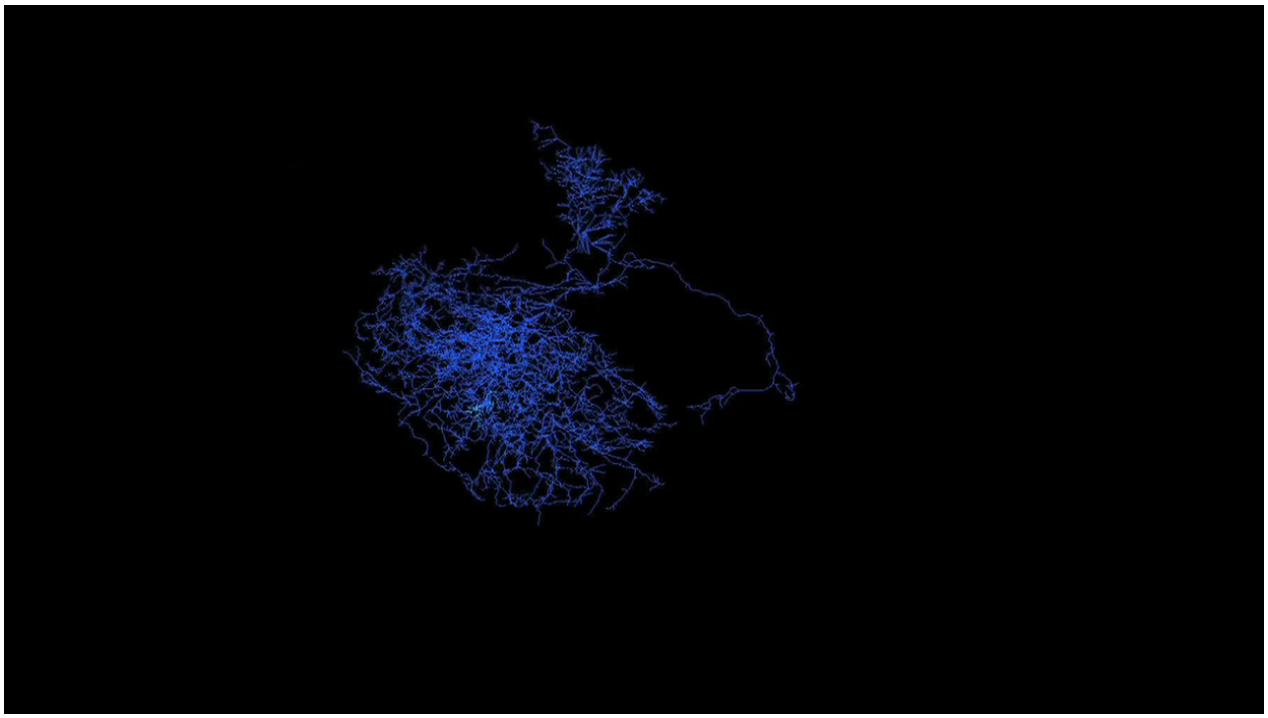
Platelet and Blood clotting simulation



Heart Simulation



Neuron simulation (silkworm)



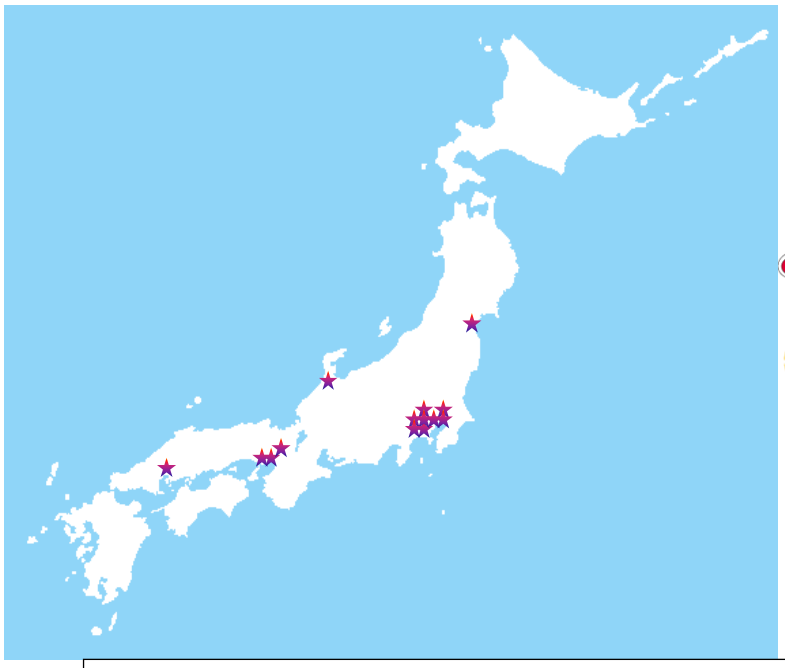
Summary

- We are developing 34 application software for Life Science researchers
- 31 applications that are highly tuned for K will be offered to use on K by the end of FY2012.
- 16 out of 31 have been checked its parallel performance on clusters.
- 5 applications shows good parallel performance
- cppmd: MD simulation core program has already achieved more than 1 Peta FLOPs

Contributors

Workshop in 2008

Workshop in 2009



Organization 13
(RIKEN, U. of Tokyo, Osaka U., Kyoto U. Tohoku U., Tokai U., JAIST, Chiba U., Keio U. Yokohama City U., Kobe U., TiTch, ISM)
About 200 person, including 60 postDoc

多倍長計算の紹介と K-computer 向け固有値ソルバ「Eigen-K」について

報告者：今村俊幸
（電気通信大学大学院 情報理工学研究科，
現理化学研究所計算科学研究機構）
共同研究：山田進，町田昌彦 (JAEA)

１． 多倍長のソフトウェア開発

ペタスケールの計算機を数週間稼働させたとき、その演算回数は 10^21 に達する。理論上、浮動小数点計算の蓄積誤差はその有効精度を超えてしまう。実際には丸め誤差は相当数がキャンセルするため、多くの場合はシビアではない。しかし、ペタスケール計算機で解くべき問題規模においては、問題そのものの表現が倍精度で十分かの議論も登場しつつある。そのような背景の中で、我々は double-double データフォーマットに基づく多倍長数値計算ソフトウェア(multiple-precision numerical software)の研究開発を進めている。Knuth 以来、多倍長計算の基本アルゴリズムは確立しているが、本研究で採用するのは Bailey, Hida らによる double-double である。通常の多倍長計算が integer を多数並べて実現するところを，double によって実現するものである。基本演算が double の演算で構成できるため，マイクロプロセッサの持つ高い浮動週数点演算性能を活かして高速計算が可能という利点を持つ。Double-double の基本演算自身は Bailey, Hida らによって qd ライブラリの一部という形で公開されている。また、数値計算ライブラリも中田により MPACK の一部に収録されている（MPACK 自身は double-double 以外にも多くの多倍長フォーマットをサポートしている）。

我々は、DD(double-double)に基づく基本線形計算ルーチンの整備から始め、実機向けに最適化を施し実用レベルまで性能を引き出すことを考えている。次に、MPI 通信ライブラリ、更に高度なソフトウェア群（LAPACK, ScaLAPACK）へと開発を進め、大規模なスパコン環境下での実用的な多倍長数値計算ライブラリの展開を進める計画でいる。

この報告書を作成している現在（平成２５年４月）、我々は BLAS として知られている線形基本演算サブプログラム群の double-double 版 QPBLAS の公開を開始している（http://ccse.jaea.go.jp/ja/download/qpblas.html）。さらに、東大奥田洋司教授の協力を得て GPU 版も公開をしている(http://ccse.jaea.go.jp/ja/download/qpblas_gpu.html)。また、次章で紹介する「京」向け固有値ソルバ EigenK についても double-double による実装 QPEigenK が完成し、その速報が SC2012(Salt Lake City, USA)にてポスター報告されている(*T.Imamura 他, Preliminary Report for a High Precision Distributed Memory Parallel Eigenvalue Solver, SC12, Nov. 2012*)。

２．　京コンピュータ向け固有値ソルバ「Eigen-K」

我々が地球シミュレータ時代から開発を進めてきた固有値ソルバーを、2008 年ごろから「京」コンピュター向け（開発初期は次世代計算機と呼んでいた）に改良をした Eigen-K が昨年６月に JAEA より公開されている(http://ccse.jaea.go.jp/ja/download/eigenk.html)。報告資料にもあるように、MPI/OpenMP ハイブリッドモデルによる２階層の並列処理が可能であり、ベクトル化もしくは SIMD による性能チューニングも実施されている。ベンダー提供の高性能 BLAS を最大限に利用することで、高い性能を発揮できるように設計されている。開発当時は「京」コンピュータ実機が利用できなかったため、東大情報基盤センターの T2K スパコンや FX10 を使った開発体制をとっていたが、平成２４年９月の一般共用開始により「京」上でのチューニングや機能強化が進んでいる（「京」上での機能強化版の公開は近日を予定)。

現在、「京」から「ポストペタ」に向けた固有値ソルバへの開発を指向して、JST CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」領域の中の「ポストペタスケールに対応した階層モデルによる超並列固有値解析エンジンの開発」課題が進行している。我々は当該課題において Eigen-K を基にした Eigen-Exa の開発を進めている。Eigen-Exa は「京」以上に高並列化が進むと予想される次世代計算環境を想定して、メニイコア向けの階層的なアルゴリズム構成、省通信最適化、自動チューニングなど次世代計算機環境向けの研究技術が盛り込まれた数値計算ライブラリになる予定である。

多倍長計算の紹介と K-computer向け固有値ソルバ 「Eigen-K」について

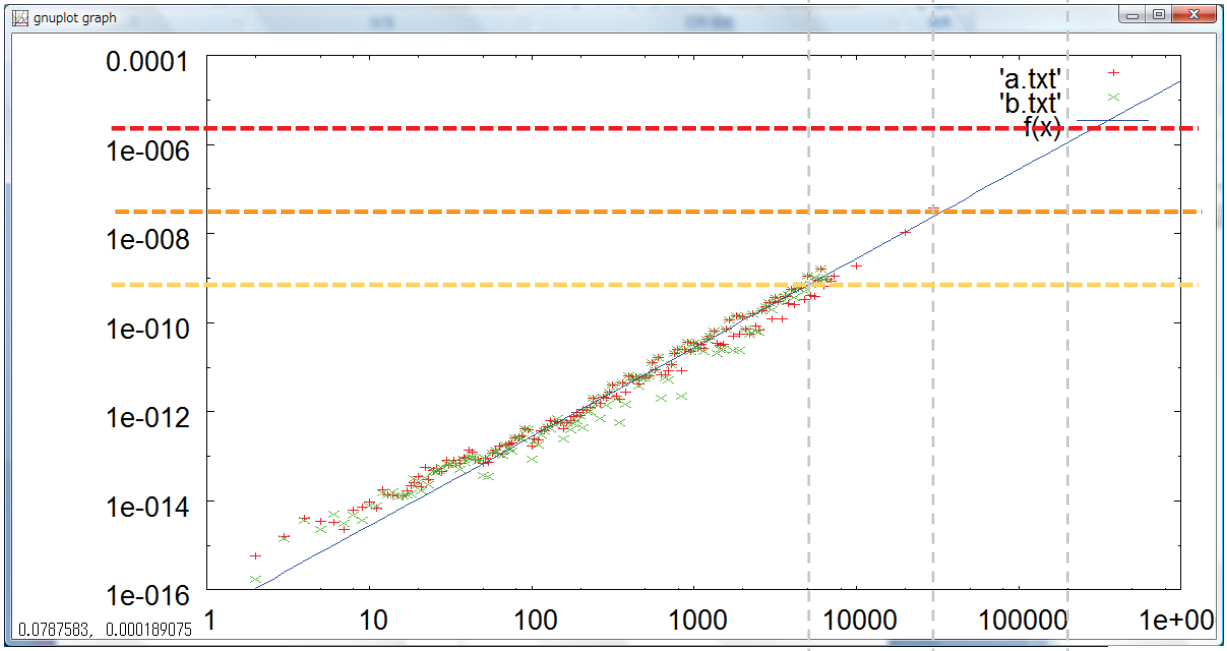
電気通信大学大学院　情報理工学研究科
今村俊幸
共同研究：　山田進, 町田昌彦　(JAEA)

1部：　理論・計算法の詳細は中田氏にお任せします。

多倍長のソフトウェア開発

大規模化による精度への影響

- 倍精度計算の破綻も近い・・・
  - 最大相対誤差：Frank Matrix κ(A)=O(N^2)、2アルゴリズムで



10万超コアへの展望

- 性能以外：精度
  - N=10^5を超えると、条件数がO(N^2)の問題で精度不足が発生する。これは倍精度での表現に基づく本質的な問題（途中の計算法改善で何とかというレベルではない）

  例：Frank matrix $A_{ij} = N + 1 - \max\{i, j\},\ \kappa = O(N^2)$

  - エキサスケールではN=10^6がターゲット
    →高速化も重要だが精度の問題もvisibleに

- コアの有効利用
  - （例えば）幾つかのコアに多倍長計算
  - QD,DDによる安価な疑似8or4倍精度計算
    - 例：XBLASなどの先行研究あり
    - DDをインライン展開できるよう実装すれば、コストは20倍程度の増加でOK, さらに要求Byte/flopは低いので付加的な計算として主の倍精度計算を圧迫しないと予想。

戦略的な多倍長ソフト整備の重要性

- 範囲は広いので、ターゲットを行列計算に絞って。
  - BLASのdouble-double版の完全実装
    - 工数は？単純に10人月でできるか？
  - 最適化はアンローリング程度
    - 対象ルーチンが多数→外注ベースの作業。
    - コーディング担当複数。数値計算の知識がなくても可能な範囲で。
    - 高度なブロック化は第二ステップ
- DD-BLAS [2011]
  - 2011年12月を目標に公開を準備
  - LAPACKなどの上位数値計算アルゴリズムへの展開は，中田氏のMPACKと連携を。

Double-double (Knuth,Dekker,Bailey,Hida)

- 疑似4倍精度： a=(a.hi/a.lo)
  - 倍精度演算2個で構成、(例：DD.MUL)

倍精度の加算 (a+b → sum+err)

$|a| \geq |b|$ を仮定

```
subroutine quick_two_sum( a, b, sum, err)
  real*8 :: a, b, sum, err
  sum=a+b
  err=b-(s-a)
end
```

$|a| \geq |b|$ を仮定しない

```
subroutine two_sum( a, b, sum, err)
  real*8 :: a, b, sum, err, v
  sum=a+b
  v=sum-a
  err=(a-(sum-v))+(b-v)
end
```

4倍精度の加算 (c=a+b)

a=a.hi+a.lo, b=b.hi+b.lo, c=c.hi+c.lo とする
(a(1)=a.hi, a(2)=a.lo, b(1)=b.hi, b(2)=b.lo, c(1)=c.hi, c(2)=c.lo)

```
subroutine add( a, b, c)
  real*8 :: a(2), b(2), c(2), sum, err
  call two_sum(a(1), b(1), sum, err)
  err=err+a(2)+b(2)
  call quick_two_sum(sum, err, c(1), c(2))
end
```

Double-double (Knuth,Dekker,Bailey,Hida)

- (例：DD.MUL, IntelのSSEレジスタを使う場合)

倍精度の乗算 (a*b → prod+err)

```
subroutine split( a, hi, lo )
  real*8 :: a, hi, lo
  hi = IAND( a, 0xfffffffffc000000 )
  lo = a – hi
end
```

```
subroutine two_prod( a, b, prod, err)
  real*8 :: a, b, prod, err
  real*8 :: a_hi,a_lo,b_hi,b_lo
  prod = a * b
  call split( a, a_hi, a_lo )
  call split( b, b_hi, b_lo )
  err = ((((a_hi * b_hi – prod)
  + a_hi * b_lo) + a_lo * b_hi) + a_lo * b_lo)
end
```

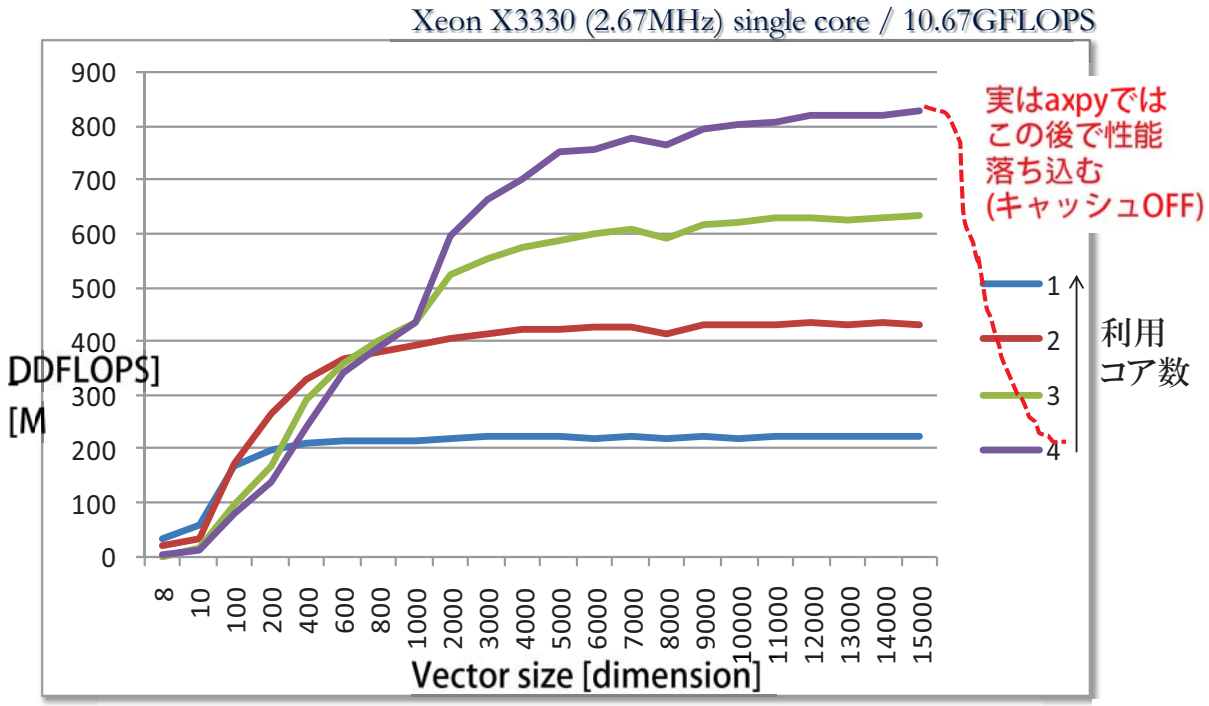
4倍精度の乗算 (c=a*b)

a=a.hi+a.lo, b=b.hi+b.lo, c=c.hi+c.lo とする
(a(1)=a.hi, a(2)=a.lo, b(1)=b.hi, b(2)=b.lo, c(1)=c.hi, c(2)=c.lo)

```
subroutine mul( a, b, c)
  real*8 :: a(2), b(2), c(2), prod, err
  call two_prod( a(1), b(1), prod, err )
  err = err + a(1)*b(2) + a(2)*b(1)
  call quick_two_sum( prod, err, c(1), c(2) )
end
```


4.6, DD-BLAS dd_axpyの性能

- 今村が精一杯チューニングした場合で



戦略的な多倍長ソフト整備の重要性

- 戦略的には
  1. DD-BLASなどの基本ルーチン群を整備
  2. さらに高度な数学ソフトウェア関数群LAPACKやFFTの整備
  3. **Doubl-Doubleを利用したプログラミングの一般化（Hida-Baileyのqdライブラリ）**
     - C++: qdクラスをベースにして、よい性能が期待される
     - Fortran90: moduleを用いることも可能だが性能はよろしくない。コンパイラか高度なプリプロセッサ。自動チューニングの技術が必要。
  4. **並列化**
     - MPI対応は容易。OpenMPはreductionがネック。
     - GPGPUへの展開
       - Doubleがサポートされる以前はdouble-floatもあった。
       - Double-doubleは容易に実装可能かつ、既存研究あり

       椋・高橋、中田、中里ら
  5. **アプリケーションユーザレベルでの必要性の認識**

戦略的な多倍長ソフト整備の重要性

- 部分ごとにであったり、C++ユーザあたり（計算科学でない開発者経由）での開発は今後ありうるだろう。
- HPCの視点がわかるSS研による率先した整備体制が望まれる。
- 「京」やポストペタへの展開
  - Q：Fujitsuは対応するのか？

2部：

密行列版固有値ソルバ EIGENシリーズ

Eigenシリーズ＝分散並列計算機向け密対称行列固有値ソルバーパッケージ :: 開発ヒストリー

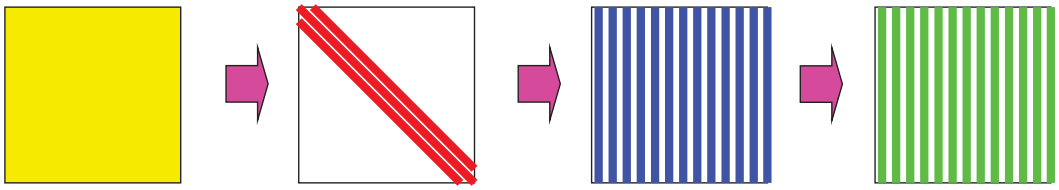
- 2006
  - SuperComputing2006, GBP Finalists
  - Quantum Physics Simulation codeへの応用
  - 地球シミュレータとSGI Altix向け
- 2008
  - T2K運用開始直後から「京」向け固有値ソルバーとして開発開始
- 2010
  - T2Kにおいて256ノード(4096コア)動作確認
  - 他、BX2, FX1, RICCなど国内主要な大規模クラスタでの動作検証
  - GPGPUへの対応も開始(eigen_g)
- 2011
  - T2Kにおいて(8192コア)での動作確認済み(10TFLOPS超、後述）
  - K開発関連グループに提供（例：RSDFT）
  - オープンソースとしての公開予定(eigen_K)
  - Open Petascale Library への参加予定
  - CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」のもとで、ポストペタスケール計算環境向け固有値ソルバー開発開始(eigen-Exa)

Quick Review of the algorithms, Diagonalization, for a dense real symmetric matrix

- Householder-D&C

Method:

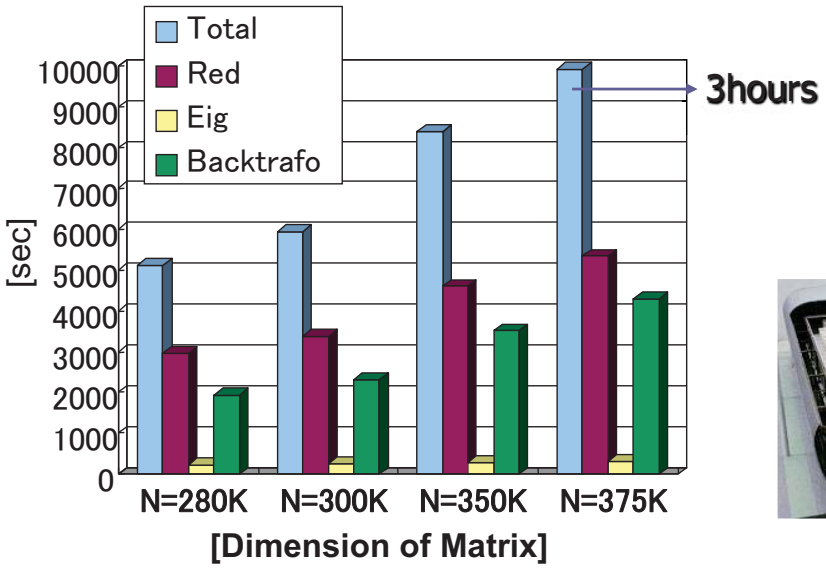
Householder	Divide&Conqure	Back Transform.
$O(N^3)$	$O(N^2)$~$O(N^3)$	$O(N^3)$



Dense matrix → Tridiagonal → Eigenpairs → Eigenpairs

Householder:	Self-development	Block Algorithm Loop unrolling 2-D distribution
Divide & Conqure:	Porting ScaLAPACK	
Back Transform:	Self-development	

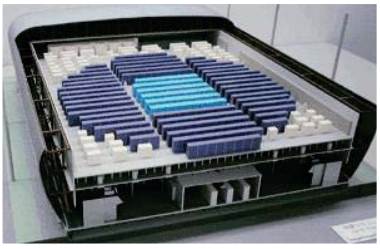
ESでの固有値ソルバーの性能,



Matrix dim.:
280,000~375,000

Earth Simulator:
4,096 VPU's

Memory:
2~3 T Bytes



Reported at SC|06, Tampa.

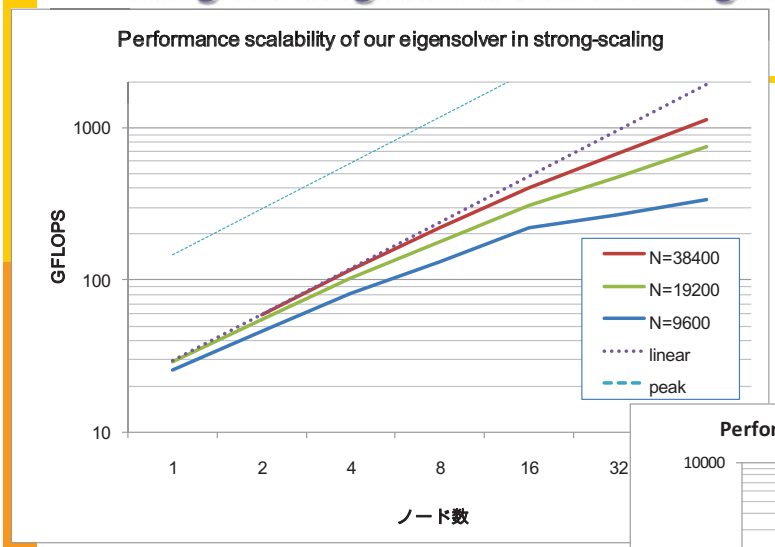
We confirmed stability of the solver up to 375,000 X 375,000

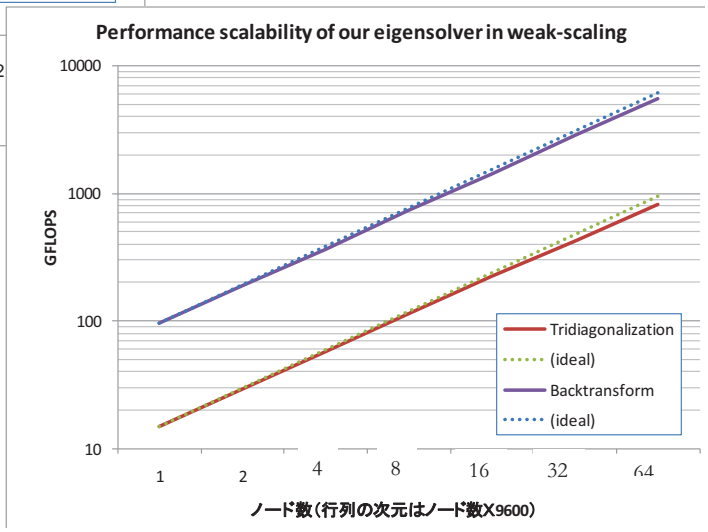
※ Accuracy is pretty excellent up to 300K X 300K (confirmed).

Eigen-s/Eigen-sx

- 次世代計算機向けに開発中の固有値ソルバー
  - Eigen_s (対称行列向けソルバー)
    - 従来開発の固有値ソルバー(SGI Altix, ES)をマルチコア・マルチプロセッサ向けに改良(T2Kほか)、Householder3重対角化＋分割統治法＋逆変換
  - Eigen_sx
    - 帯行列を経由するアルゴリズムを採用。大規模問題で高速。
- 目標機能
  - MPI/OpenMPハイブリッドモデルでの動作
    - 全てのモデルでの高性能達成(ScaLAPACKよりは高速)
      - 高いスケーラビリティ
      - 104コアまではほぼリニアで、105以降ピークの1%を目標
  - チューニングパラメータ
    - ScaLAPACKとは異なり、性能パラメタは分散方式と独立
    - 動的に制御可能
    - BLASも積極的に利用。但し、スレッド呼び出し方式
  - マルチコア向けアルゴリズムの採用
    - Narrow-band-reductionアルゴリズム+帯行列向け分割統治法

Strong scaling and weak scaling for eigen_s/T2K Tokyo





性能最適化・チューニングについて

「問題点」 近代的なマルチコアプロセッサは計算能力に対して、プロセッサーメモリ間のバンド幅つまり、データ供給能力が低い

- 高速化の指針：
  1. コンパイラの最適化
  2. 手動でのソースコードチューニング
     - ループ構造やデータの変更
     - ループアンローリングなど（HPCのテキストにある基本技術）
  3. 高速な実装をできるだけ利用する
     - BLAS (ATLAS, GotoBLAS, MKL, CUBLAS, MAGMA)
     - 数値計算ライブラリ
  4. 上記の「問題点」を解決するようなアルゴリズムの改良
     - 特にブロック化によるLevel3 BLASの活用
  5. 全く異なる計算原理の導入

Eigen_sxの最大性能 (規模: 512node,N=300K, HA8000クラスタシステム 512ノードサービス Feb.2011)

```
 N=     300000 NM=       9392
 NV=       9392
 NUM.OF.PROCESS=       2048 (       32       64 )
 NUM.OF.THREADS=          4
 calc (u,beta)       569.725378274918
 mat-vec (Au)       2382.50363969803
7555.07765028280
 2update (A-uv-vu)   372.795704603195
48283.8181281065
 calc v            327.551038742065
 v=v-(UV+VU)u       149.752300024033
 UV post reduction   1.38964557647705
 COMM_STAT
   BCAST  ::    110.417062759399
   REDUCE ::    685.837212562561
   REDIST ::    618.934033155441
   GATHER ::   0.000000000000000E+000
 TRD-BLK     300000     3831.22618889809
9396.46949175666      GFLOPS
 TRD-BLK-INFO      300000          48
 before PDSTEDC  0.361311912536621
 PDSTEDC   401.591748952866
 COMM_STAT
   BCAST  ::   0.125138998031616
   REDUCE ::   0.000000000000000E+000
   REDIST ::   0.000000000000000E+000
   GATHER ::   0.000000000000000E+000
D&C   408.536870002747      ERRCODE=          0
TRBAK=   1203.30433797836
COMM=   221.230289697647
     44876.4275966327      GFLOPS
     54519.0569849592      GFLOPS
     56566.4621065163      GFLOPS
COMM_STAT
   BCAST  ::    88.3142938613892
   REDUCE ::    126.187926292419
   REDIST ::   0.000000000000000E+000
   GATHER ::   0.000000000000000E+000
TRBAKWY   1203.30767083168
 TRDBAK     300000   1203.34389090538
44874.9512761035      GFLOPS
 Total   5443.10710787773
16534.6736099997      GFLOPS
```

注意：分割統治法のコストを無視しているので、実際のFLOPS値は更に高い

T2K 512ノード(8192cores), 理論ピーク75.4TFLOPS

Eigen-Kを利用予定のソフト

- 第二分野
  - RSDFT： 押山先生、岩田さん（東大）、
  - 今田先生、三沢さん（東大）
  - その他、物性研のユーザなど
- 第四分野
  - PHASE: 黒田さん（理研）

- 利用ユーザ拡大のために
  - エルミート公開版の整備（H23年度）
  - 開発環境の整備

    ←これは固有値ソルバだけの話ではないので、こちらも戦略的にアメリカのドンガラ教授のようにやらないといけない！

まとめ

本日のお話のまとめ

- 多倍長計算プログラムの重要性
  - ペタスケール時代になり、データ表現レベルで精度不足が潜在的にあり、それがvisibleになってきている。
  - 戦略的に多倍長ソフトウェアを整備して、世界をリード＆計算科学シミュレーションの質的変化を

    性能とともに、精度も含めて考えよう。

- K向け固有値ソルバ
  - Eigen-K　PFLOPSが可能なソルバ
  - 公開予定
  - 利用者拡大のための周辺ソフトウェア整備

4.3 高精度線形代数演算ライブラリ MPACK の紹介

高精度線形代数演算ライブラリ MPACK の紹介　　中田真秀

線形代数は有史以来、人類の知的好奇心を満足させるだけでなく、具体的にも応用面で役立ってきた、本質的に重要な分野である。特に計算を行うということに限れば、コンピュータの発展はより大規模な線形代数の問題を、より高速に解くこととパラレルになっているといって良い。

今までのアプローチでは、大規模な線形代数の問題を、高速に解くことにほぼ限られていた。つまり正確に解くということに関してはほとんど光があたって来なかった。私は線形代数の問題をコンピュータ上で正確に解くというパラダイムを提案したい。

コンピュータでの演算には誤差が入るが、これまで演算誤差、結果の正確性についてはあまり注目されて来なかった。IEEE 754 倍精度は 10 進 16 桁で十分だったからだ。

しかし、そもそも倍精度では足りない問題は多くある。例としては、反復法での連立一次方程式を解くことや半正定値計画問題を解くことだろう。ただ、これは氷山の一角であると考えられる。研究者たちは実際に倍精度では足りない問題などは、うまくゆかない問題、運が悪い問題などと諦めることが多く表に出ることが少ないからだ。さらに CPU の処理速度が爆発的に伸び、超巨大な問題を長時間かけて解く現在、精度が不足することは容易に考えられる。

これらを踏まえ、高精度線形代数演算パッケージである MPACK (http://mplapack.sourceforge.net/) を開発してきた。線形代数演算パッケージは、BLAS, LAPACK が有名であるが、これをベースにし、高精度な演算ができるようにした。以下に特徴を挙げる。コンピュータ資源の許す限り高精度な計算や、四倍精度程度だが高速な計算も可能で、様々なニーズに応えられるようになっている。BLAS, LAPACK のように、線形代数演算のビルディングブロック、参照実装、プログラムインターフェースを提供する。C++で書かれており、5 つの高精度演算のための型をサポートしている（GMP, MPFR, QD, DD, 四倍精度)、さまざまなプラットフォーム（Linux, FreeBSD, Windows, Intel Xeon Phi, NVIDIA GPU）をサポートし、オープンソースのライセンス(2-clause BSD ライセンス)で配布している。

高精度線形代数演算ライブラリ MPACK の紹介

中田真秀

理化学研究所, 情報基盤センター 2F

2011/10/28 SS 研ペタスケール数値計算ライブラリ WG

MPACK 0.6.7: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

中田真秀 @ 理研

- MPACK: multiple precision version of BLAS and LAPACK.
- ビルディングブロック、参照実装、API の提供
- Version 0.6.7 (2010/8/20); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- マルチプラットフォーム:Linux/BSD/Mac/Win...
- 5 つの高精度ライブラリをサポート: GMP, MPFR, DD, QD, double
- C++で書いてある: 簡単、高速なプログラミング
- フリーソフトウェア: 2 条項 BSD ライセンス: 改変、再配布が自由にできる。

あらまし

1. ペタ、エクサフロップスに向かってより高精度へ
2. 浮動小数点: コンピュータ上でどうやって計算をするか
3. MPACK0.6.7: BLAS, LAPACK の高精度版
4. GPU での行列-行列積の加速
5. まとめ、これからについて

ペタ、エクサフロップスマシンでは高精度が必要になる

- 数値計算誤差が大きな問題になってくる。
  - エクサマシンは一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- 規模の小さい問題でも精度が出ずお蔵入りになる問題は多数あるようだ。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

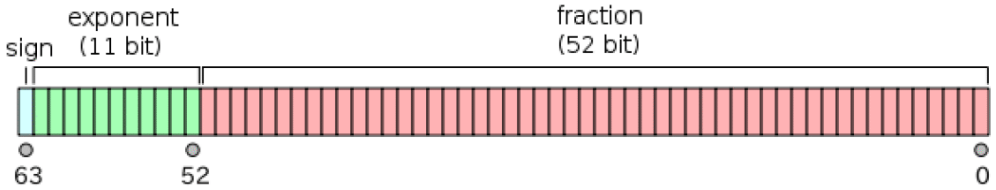
丸め誤差の影響をを高精度計算で解決する

- 浮動小数点: 実数の計算機上での近似的取扱い。
- 丸め誤差が演算ごとに起こりうる
- 誤差への (一つの) 解決法: 高精度計算

高精度計算は丸め誤差に対する brute force な解決方法

IEEE 754: Standard for Binary Floating-Point Arithmetic

- 754-2008 IEEE Standard for Floating-Point Arithmetic
- もっとも使われているし高速 (on Core i7 920: ~40GFlops; RADEON HD5970 ~1TFlops)
- 10 進数 16 桁 16 binary64 (いわゆる倍精度) 64bit



- $a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{52}}{2^{52}}\right) \times 2^e$, $d = 0$ or 1, $e = -1022 \sim 1023$

(Partially taken from Wikipedia: http://en.wikipedia.org/wiki/IEEE_754-2008).

高精度計算とは

- GMP とは? GMP は自由なライブラリで、符号つき整数、有理数、浮動小数点数を扱える。
- 精度は任意に大きく選べる (32, 64 ビットの倍数)

他の高精度演算ライブラリ

他の高精度演算ライブラリ

- 倍倍精度, 四-倍精度 (20 から 200 倍程度遅い): QD, DD ライブラリ
- 丸めモード変更のサポート (100 倍から 1000 倍程度遅い): MPFR, MPC, exflib

デマンドに従って様々な形態のライブラリが存在している。

このプロジェクトでは高精度演算ライブラリの構築は目的としていない

BLAS, LAPACK とはなにか?

- BLAS: ベクトル-ベクトル, 行列-ベクトル, 行列-行列の演算を行う。参照実装や高速な実装 (GotoBLAS2, Intel MKL, ATLAS etc.) もある。
- LAPACK: 線形連立一次方程式、固有値問題, 最小二乗法、特異値分解等を BLAS をブロックとして解く。
- デファクトスタンダードのライブラリ: 気づかずに使っていることも多々ある。
- LAPACK の web のヒット数: 90,340,774(Thu Feb 24 12:54:24 EST 20110)

BLAS, LAPACK は大変重要なライブラリ

MPACK 0.6.7: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

- MPACK: multiple precision version of BLAS and LAPACK.
- ビルディングブロック、参照実装、API の提供
- Version 0.6.7 (2010/8/20); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- マルチプラットフォーム:Linux/BSD/Mac/Win...
- 5 つの高精度ライブラリをサポート: GMP, MPFR, DD, QD, double
- C++で書いてある: 簡単、高速なプログラミング
- フリーソフトウェア: 2 条項 BSD ライセンス: 改変、再配布が自由にできる。

MPACK 0.6.7: 何ができて何ができないか

- Version 0.6.7 (2010/8/20); MBLAS 全 76 が完成, 100 MLAPACK ルーチン完成.
- MLAPACK でできること: 実, 複素対称行列対角化, LU 分解, Cholesky 分解, 条件数推定, 逆行列
- MLAPACK でまだできないこと: 非対称対角化, 固有値分解, 最小二乗法, QR 分解など..

MPACK 紹介:API 提供: 命名規則

プリフィックスの変化
float, double → “R”eal,
complex, double complex → “C”complex.

- daxpy, zaxpy → Raxpy, Caxpy
- dgemm, zgemm → Rgemm, Cgemm
- dsterf, dsyev → Rsterf, Rsyev
- dzabs1, dzasum → RCabs1, RCasum

MBLAS 0.6.7 でサポートされているルーチン

LEVEL1 MBLAS

Crotg	Cscal	Rrotg	Rrot	Rrotm	CRrot	Cswap
Rswap	CRscal	Rscal	Ccopy	Rcopy	Caxpy	Raxpy
Rdot	Cdotc	Cdotu	RCnrm2	Rnrm2	Rasum	iCasum
iRamax	RCabs1	Mlsame	Mxerbla			

LEVEL2 MBLAS

Cgemv	Rgemv	Cgbmv	Rgbmv	Chemv	Chbmv	Chpmv	Rsymv
Rsbmv	Ctrmv	Cgemv	Rgemv	Cgbmv	Rgemv	Chemv	Chbmv
Chpmv	Rsymv	Rsbmv	Rspmv	Ctrmv	Rtrmv	Ctbmv	Ctpmv
Rtpmv	Ctrsv	Rtrsv	Ctbsv	Rtbsv	Ctpsv	Rger	Cgeru
Cgerc	Cher	Chpr	Cher2	Chpr2	Rsyr	Rspr	Rsyr2
Rspr2							

LEVEL3 MBLAS

Cgemm	Rgemm	Csymm	Rsymm	Chemm	Csyrk	Rsyrk	Cherk
Csyr2k	Rsyr2k	Cher2k	Ctrmm	Rtrmm	Ctrsm	Rtrsm	

MLAPACK 0.6.7 でサポートされている 100 ルーチン

Mutils	Rlamch	Rlae2	Rlaev2	Claev2	Rlassq	Classq
Rlanst	Clanht	Rlansy	Clansy	Clanhe	Rlapy2	Rlarfg
Rlapy3	Rladiv	Cladiv	Clarfg	Rlartg	Clartg	Rlaset
Claset	Rlasr	Clasr	Rpotf2	Clacgv	Cpotf2	Rlascl
Clascl	Rlasrt	Rsytd2	Chetd2	Rsteqr	Csteqr	Rsterf
Rlarf	Clarf	Rorg2l	Cung2l	Rorg2r	Cung2r	Rlarft
Clarft	Rlarfb	Clarfb	Rorgqr	Cungqr	Rorgql	Cungql
Rlatrd	Clatrd	Rsytrd	Chetrd	Rorgtr	Cungtr	Rsyev
Cheev	Rpotrf	Cpotrf	Clacrm	Rtrti2	Ctrti2	Rtrtri
Ctrtri	Rgetf2	Cgetf2	Rlaswp	Claswp	Rgetrf	Cgetrf
Rgetri	Cgetri	Rgetrs	Cgetrs	Rgesv	Cgesv	Rtrtrs
Ctrtrs	Rlasyf	Clasyf	Clahef	Clacrt	Claesy	Crot
Cspmv	Cspr	Csymv	Csyr	iCmax1	RCsum1	Rpotrs
Rposv	Rgeequ	Rlatrs	Rlange	Rgecon	Rlauu2	Rlauum
Rpotri	Rpocon					

赤字は 0.6.6 から 0.6.7 に向けての 10 個のルーチン

API の提供: コール方法の違い

違い: call by value / call by reference
MBLAS/MLAPACK:

```
Rgemm("n", "n", n, n, n, alpha, A, n, B, n, beta, C, n);
Rgetrf(n, n, A, n, ipiv, &info);
Rgetri(n, A, n, ipiv, work, lwork, &info);
Rsyev("V", "U", n, A, n, w, work, &lwork, &info);
```

BLAS/LAPACK:

```
dgemm_f77("N", "N", &n, &n, &n, &One, A, &n, A, &n, &Zero, C,
dgetri_f77(&n, A, &n, ipiv, work, &lwork, &info);
```

プログラミングモデル

- 数値の型: INTEGER, REAL, COMPLEX, LOGICAL.
- 高精度ライブラリを “typedef” で入れ替え REAL → mpf_class, qd_real, dd_real etc.
- 初等関数も必要 (log, sin etc); 倍精度程度の精度で充分
- 現在サポートされている高精度計算ライブラリ: GMP, MPFR, QD (DD), double
- 中間的な関数で高精度計算ライブラリの違いを吸収
- ほぼ C++の “double” と同じ感覚でプログラミングできる。

MPACKでなぜいくつも高精度演算ライブラリをサポートするか

- 倍倍精度, 四-倍精度 (一番高速:10 から 200 倍程度しか遅くない): QD, DD ライブラリ
- 丸めモード変更のサポート (100 倍から 1000 倍程度遅い): MPFR, MPC, exflib
- とりあえずの高精度計算: GMP (MPFR, MPC などよりかは少し高速)
- double (普通の倍精度): デバッグ用

計算デマンドに従って様々な形態のライブラリが存在している。

MBLASのコードの例

Caxpy: axpy の複素数版から抜粋

```
void Caxpy(INTEGER n, COMPLEX ca, COMPLEX * cx, INTEGER incx, COMPLEX
{
    REAL Zero = 0.0;
    if (n <= 0)
        return;
    if (RCabs1(ca) == Zero)
        return;
    INTEGER ix = 0;
    INTEGER iy = 0;
    if (incx < 0)
        ix = (-n + 1) * incx;
    if (incy < 0)
        iy = (-n + 1) * incy;
    for (INTEGER i = 0; i < n; i++) {
        cy[iy] = cy[iy] + ca * cx[ix];
        ix = ix + incx;
```

MLAPACK のコード例

Rsyev; 対称行列の対角化から抜粋

```
        Rlascl(uplo, 0, 0, One, sigma, n, n, A, lda, info);
    }
//Call DSYTRD to reduce symmetric matrix to tridiagonal form.
    inde = 1;
    indtau = inde + n;
    indwrk = indtau + n;
    llwork = *lwork - indwrk + 1;
    Rsytrd(uplo, n, &A[0], lda, &w[0], &work[inde - 1], &work[indtau -
        &work[indwrk - 1], llwork, &iinfo);
//For eigenvalues only, call DSTERF.  For eigenvectors, first call
//DORGTR to generate the orthogonal matrix, then call DSTEQR.
    if (!wantz) {
        Rsterf(n, &w[0], &work[inde - 1], info);
    } else {
        Rorgtr(uplo, n, A, lda, &work[indtau - 1], &work[indwrk - 1],
            &iinfo);
        Rsteqr(jobz, n, w, &work[inde - 1], A, lda, &work[indtau - 1],
    }
//If matrix was scaled, then rescale eigenvalues appropriately.
    if (iscale == 1) {
        if (*info == 0) {
```

MPACK (MBLAS/MLAPACK) に関する事実

- Google 検索で ”Multiple precision BLAS” とするとほぼ MPACK についてのみ
- 23000 回の web ページヒット, ダウンロード 1280 回以上 (2011/2/25)

Date (UTC)	Rank	Total Pages	Downloads	Project Web Hits	Tracker opened (closed)	Forum Posts
Dec 2009	N/D	79	11	258	0 (0)	0
Nov 2009	3,901	311	66	825	0 (0)	0
Oct 2009	2,013	325	55	746	0 (0)	0
Sep 2009	18,846	335	27	590	0 (0)	0
Aug 2009	82,634	136	26	436	0 (0)	0
Jul 2009	90,825	63	10	535	0 (0)	0
Jun 2009	68,481	125	19	473	0 (0)	0
May 2009	32,211	136	13	533	0 (0)	0
Apr 2009	10,449	174	24	388	0 (0)	0
Mar 2009	4,249	273	34	461	0 (0)	0
Feb 2009	3,177	647	39	407	0 (0)	0
Jan 2009	5,640	796	17	208	0 (0)	0

Partial data: End of month not yet reached

Calculated as sourceforge.net page views plus sflogo button impressions

MBLAS の品質保証

BLAS は代数処理のみ

- 色々な場合について値を MBLAS, BLAS に代入して比較、差が小さいなら ok
- これでアルゴリズム的なバグが取れる。全ての場合これで ok だった。

```
for (int k = MIN_K; k < MAX_K; k++) {
 for (int n = MIN_N; n < MAX_N; n++) {
  for (int m = MIN_M; m < MAX_M; m++) {
...
   for (int lda = minlda; lda < MAX_LDA; lda++) {
    for (int ldb = minldb; ldb < MAX_LDB; ldb++) {
     for (int ldc = max(1, m); ldc < MAX_LDC; ldc++) {

      Rgemm(transa, transb, m, n, k, alpha, A, lda, B, ldb, beta, C
      dgemm_f77(transa, transb, &m, &n, &k, &alphad, Ad, &lda,
                                            Bd, &ldb, &betad, Cd, &
...
    diff = vec_diff(C, Cd, MAT_A(ldc, n), 1);
    if (fabs(diff) > EPSILON) {
       printf(``#error %lf!!\n'', diff);
       errorflag = TRUE;
```

MLAPACK の品質保証

収束の概念が入るのでとても難しくなる

-
- 収束の概念が LAPACK から入っている
  - 本質的に変わるが、結局 MBLAS のときと同じようなデバッグ方法で大体取れる
- 研究に使うとバグが発見されることがある (脇ら *et al*: seg fault, 無限ループなど。)

Raxpy のパフォーマンス

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

$$y \leftarrow \alpha x + y$$

`Raxpy` Flops 単位で. 括弧内は OpenMP による加速

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	130(570)M
QD(64)	13.7(67)M
GMP(77)	11.3(45)M
GMP(154)	7.6(32)M
MPFR(154)	3.7(17)M
GotoBLAS(16)	1.5G

Performance of Rgemv

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

$$y \leftarrow \alpha Ax + \beta y$$

`Rgemv` Flops 単位。

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	140M
QD(64)	13M
GMP(77)	11.1M
MPFR(77)	4.7M
GMP(154)	7.1M
MPFR(154)	3.7M
GotoBLAS(16)	3.8G

Performance of Rgemm

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

`Rgemm` Flops 単位。括弧内は OpenMP での高速化

$$C \leftarrow \alpha AB + \beta C$$

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	136 (605)M
QD(64)	13.9 (63)M
GMP(77)	11.5 (44)M
MPFR(77)	4.6 (20)M
GMP(154)	7.2 (28) M
MPFR(154)	3.7 (16) M
GotoBLAS(16)	42.5G

Performance of Rsyev

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

`Rsyev` (300x300 対称行列、固有値固有ベクトルを求めるのにかかる時間)

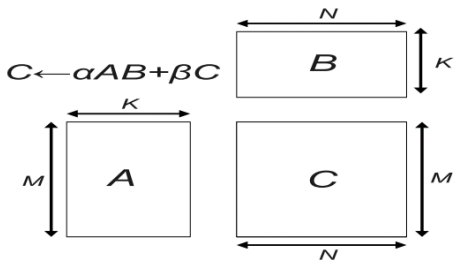
$$AX = diag[\lambda_1, \lambda_2, \cdots \lambda_N]X$$

MP Library(sign. digs.)	seconds
DD(32)	2.4
QD(64)	25.6
GMP(77)	36.9
MPFR(77)	78.9
GMP(154)	64.0
MPFR(154)	111
GotoBLAS(16)	0.1

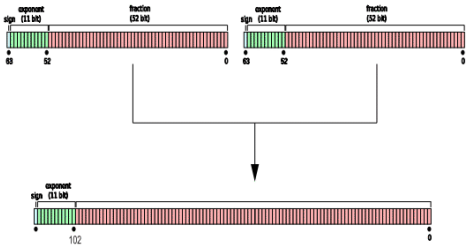
GPU での行列-行列積の加速

GPU での行列-行列積の加速

行列-行列積 Rgemm 完全実装

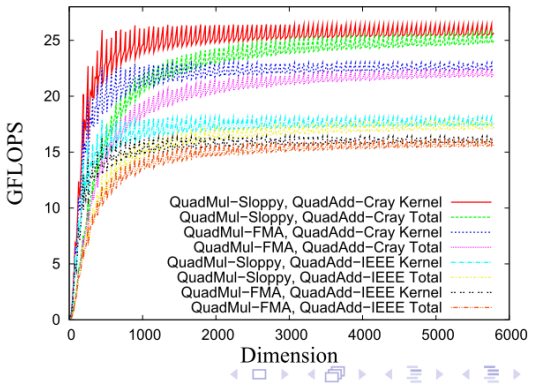


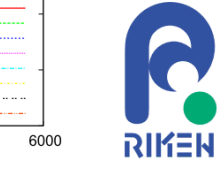
倍々精度:約 4 倍精度 (32 桁)



nVidia C2050, GPU の利用

CPU 比 150 倍, 最高 26GFlops





高精度の重要性

じゃあなぜ高精度計算はあんまりやられてないか?
遅いという思い込みがある。

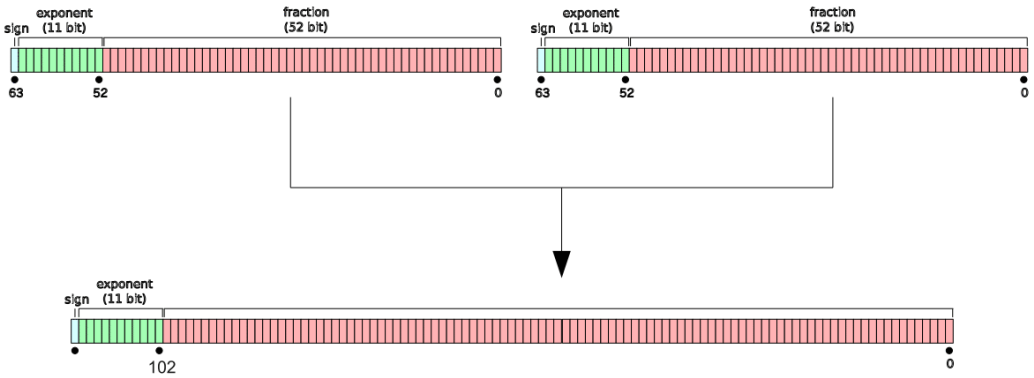
- 高精度計算は 100 倍から 1000 倍遅いという根拠の無い思い込み。
- 倍々精度は 8-20 倍程度で計算できる。10Pflops(倍精度)→1PFl ops(倍々精度)
- プロセッサ最適化が進んでない...reference BLAS と GotoBLAS2 の DGEMM 性能では 20 倍以上パフォーマンスが違う。
- 近年の CPU-Memory の転送は非常に遅い... 演算密度の高い高精度演算は有利
- FMA などハードのサポートが高速化...nVidia, AMD の GPU は持っている。

GPU で 100 倍から 200 倍高速化して CPU とコンパラになれば

やってみようという気になるはず → やってみた

倍々精度:手軽な四倍精度

倍精度二つくっつけて四倍精度



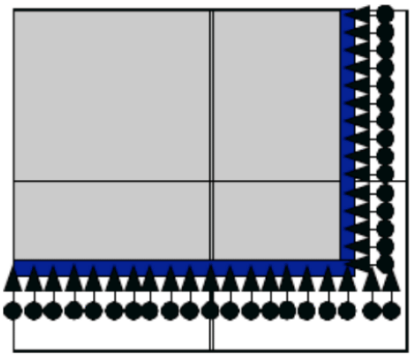
倍々精度型 a は二つの倍精度 a_{hi}, a_{lo} で定義される:

$$a = (a_{hi}, a_{lo}).$$

GPU 上での実装と評価

“Accelerating GPU kernels for dense linear algebra”, Rajib Nath, Stanimire Tomov, and Jack Dongarra による Pointer Redire cting の手法

- 超シンプルにとりあえずブロック化の外に出たらその端っこを返すようにする。

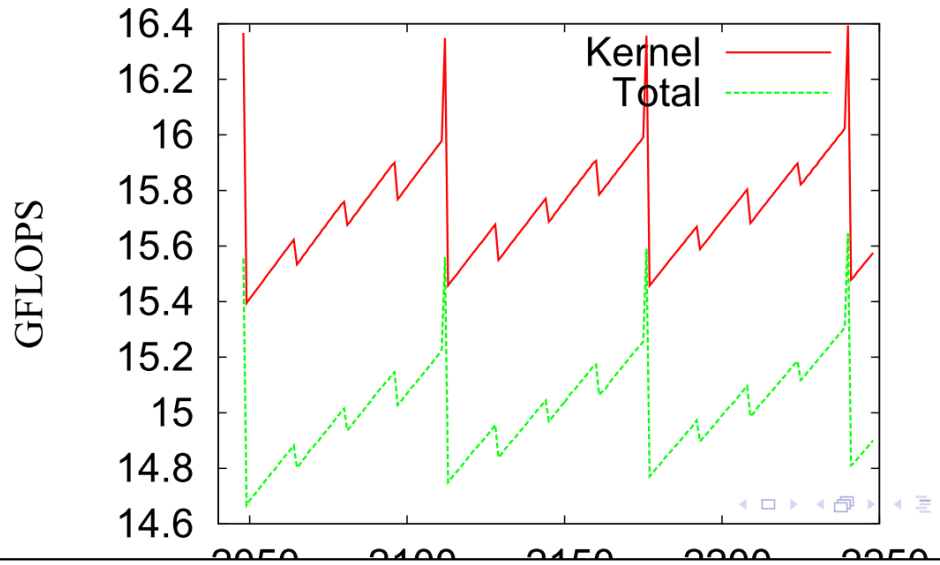


- プログラムは超簡単。
- 若干の計算の無駄がでる。

コロンブスの卵的ブレークスルー

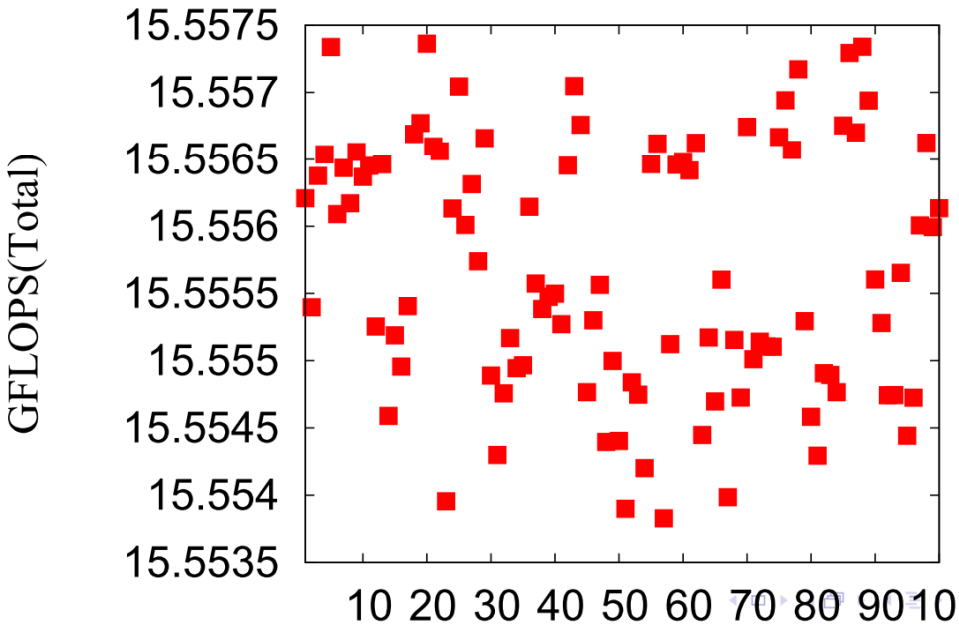
GPU 上での実装と評価

Nath らによる Pointer Redirecting の手法を用いた場合のパフォーマンスロスは 6%程度だった。正方行列で $m = 2048$ から $n = 2248$ まで動かした場合の性能。$n = 64$ の倍数の時, 特にパフォーマンスが良かった.



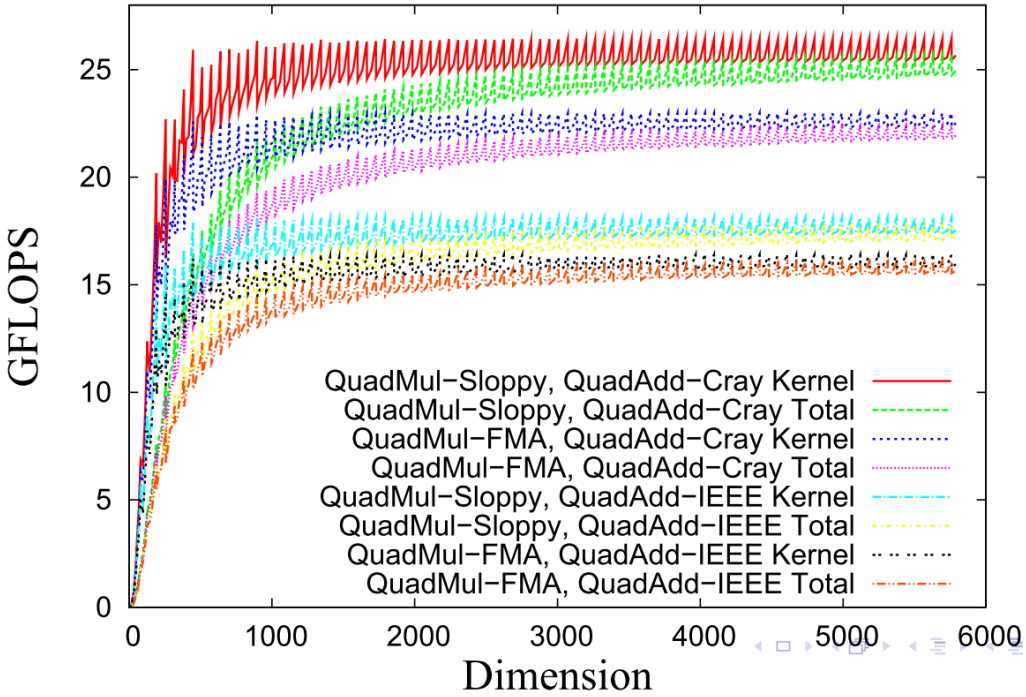
GPU 上での実装と評価

$n = 2048$ のとき 100 回連続して測定したときのパフォーマンス。
性能のぶれは 0.1% 以内だった。



GPU 上での実装と評価

演算を精度を落として高速化し, 最大で 26.4GFlops を得た.



GPU 上での実装と評価

演算を精度を落として高速化した場合の最大性能。CPU は Xeon 3470 + DDR3-1066

アルゴリズム	パフォーマンス
QuadAdd-Cray, QuadMul-Sloppy カーネル	26.4GFlops
QuadAdd-Cray, QuadMul-Sloppy トータル	25.7GFlops
QuadAdd-Cray, QuadMul カーネル	23.0GFlops
QuadAdd-Cray, QuadMul トータル	22.4GFlops
QuadAdd-IEEE, QuadMul-Sloppy カーネル	18.1GFlops
QuadAdd-IEEE, QuadMul-Sloppy トータル	17.8GFlops
QuadAdd-IEEE, QuadMul カーネル	16.4GFlops
QuadAdd-IEEE, QuadMul トータル	16.1GFlops
QuadAdd-IEEE, QuadMul CPU	100MFlops
QuadAdd-IEEE, QuadMul OpenMP CPU	400MFlops

GPU 上での実装と評価

- 16.1GFlops = 92.4% (or 46.2%) の性能 (QuadAdd-IEEE, QuadMul-FMA)
- 平均 Clock の計算:QuadAdd-IEEE 20 演算, QuadMul-FMA 10 演算, Rgemm では和積が半分ずつ

$$(20 + 10 - 1)/2 = 14.5$$

- 理論性能値は...

$$515\text{GFlops}/14.5 = 35.5\text{GFlops}$$

- 最大性能の見積りは大雑把に...nVidia C2050 だと FMA をフルに使って 515GFlops なので、

$$515\text{GFlops}/14.5/2 = 17.75\text{GFlops}$$

となる。

応用計算:半正定値計画問題ソルバ “SDPA-DD” の 10 倍の加速

応用編

応用計算:半正定値計画問題ソルバ “SDPA-DD” の 10 倍の加速

半正定値計画問題 (SDP) とは

$$
\begin{aligned}
\text{主問題}\quad \text{最小化:}\quad & A_0 \bullet X \\
\text{s.t.:}\quad & A_i \bullet X = b_i \qquad (i = 1, 2, \cdots, m) \\
& X \succeq 0 \\
\text{双対問題}\quad \text{最大化:}\quad & \sum_{i=1}^{m} b_i z_i \\
\text{s.t.:}\quad & \sum_{i=1}^{m} A_i z_i + Y = A_0 \\
& Y \succeq 0
\end{aligned}
$$

A_i は $n \times n$ 実対称行列、X $n \times n$ は実対称の変数行列、b_i は m-次元定数ベクトル、Y は $n \times n$ 実対称の変数行列、$X \bullet Y := \sum X_{ij} Y_{ij}$.
$X \succeq 0$ は X が半正定値、つまり固有値がゼロ以上

応用計算:半正定値計画問題ソルバ“SDPA-DD”の 10 倍の加速

最適解の性質と最適解付近で誤差が溜まりやすいことについて

Theorem (相補性定理)

(X^*, Y^*, z^*) の組が内点実行可能解とする。従って SDP の主問題、双対問題の条件を満たすとき、(X^*, Y^*, z^*) が最適解である必要十分条件は

$$X^* \bullet Y^* = \mathbf{0}$$

である。

応用計算:半正定値計画問題ソルバ“SDPA-DD”の 10 倍の加速

最適解で解が特異的になる:$\boldsymbol{X}^*$, $\boldsymbol{Y}^*$ が最適解のとき,

$$X^* \bullet Y^* = \mathbf{0}$$

が成立する。そして線形代数の定理から

$$\mathrm{rank}\boldsymbol{X}^* + \mathrm{rank}\boldsymbol{Y}^* \leq n \qquad (1)$$

つまり

$\boldsymbol{X}^*, \boldsymbol{Y}^*$ はどちらかが少なくとも特異的

大抵は $\boldsymbol{X}^*, \boldsymbol{Y}^*$ 両方とも特異的 → 数値計算誤差がたまりやすい。

応用計算:半正定値計画問題ソルバ“SDPA-DD”の 10 倍の加速

SDPLIB から大きないくつか問題を解いたときのベンチマーク
CPU: Xeon 3470, DDR3 -1066

問題名	CPU(秒)	GPU(秒)	加速率
equalG51	6531.9	573.2	11.4
gpp500-1	902.0	72.2	12.5
gpp500-4	638.0	74.8	8.5
maxG32	36284.4	4373.1	8.3
maxG55	521575.4	53413.1	9.8
mcp500-4	539.1	65.2	8.3
qpG11	16114.7	1408.0	11.4
qpG51	39678.9	3299.2	12.0
ss30	310.7	138.6	2.2
theta5	3250.0	239.8	13.6
theta6	9028.2	623.6	14.5
thetaG51	49161.5	4870.4	10.1

MPACK へのコントリビューション

- MPACK へのコードなどのコントリビューションは大歓迎。今までパッチ提供、バグ報告などは数件あった。
- このルーチンが欲しいという問い合わせ、要望は多数くるが、なかなか応えられてない...
- 2 条項 BSD ライセンスでコードを提供していただけるとありがたい。
- 開発者権限も発行できるといいな...

MPACK 0.6.7: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

中田真秀 @ 理研

- MPACK: multiple precision version of BLAS and LAPACK.
- ビルディングブロック、参照実装、API の提供
- Version 0.6.7 (2010/8/20); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- double-double 精度の Rgemm 作成, リリースした。他の研究 [椋木, 高橋 2010, 2011], [中里 2010])

4.4 宇宙プラズマでのペタスケール数値計算（惑星磁気圏シミュレーション）

宇宙プラズマでのペタスケール数値計算（惑星磁気圏シミュレーション）

九州大学情報基盤研究開発センター　深沢　圭一郎

宇宙プラズマ研究において、我々は主に太陽から吹いてくる磁場を伴ったプラズマの風（太陽風）と地球の磁場が相互作用して起こる様々な現象を研究ターゲットにしている。これらは宇宙空間で起きる現象であるため探査機を打ち上げて観測を行うが、基本的に”その場”の観測しか行えない。そのため、宇宙プラズマ計算機シミュレーションがこの分野の理論の発展、また観測結果の理解の促進に非常に重要な役割を果たしてきている。宇宙プラズマの振る舞いはブラソフ方程式によって正確に記述されるが、特に我々が注目している太陽風と惑星の磁場の相互作用によって形成される磁気圏というグローバルな領域では、ブラソフ方程式を近似した電磁流体（MHD）近似というものがよく成り立ち、MHD 方程式を用いて惑星時磁気圏のシミュレーションをおこなっている。

近年計算機の発達により高精度なシミュレーションが行えるようになってきた。例として、土星磁気圏では今までのシミュレーション結果からは見ることができなかったスモールスケールのプラズマ対流の渦構造が磁気圏内で見えてきている。この渦構造は電流を生成するため、惑星のオーロラ発光と関連づけられることがある。今回の結果は現在土星を観測しているカッシーニ探査機により撮像された土星オーロラとの相関性が見え、土星オーロラの発光に渦構造が関係しているということが強く示唆された。

一方でこのような高精度の計算も、実際に我々が見たい物理現象のスケールからはまだまだ粗く、今後の計算機の発展が期待されている。その際は超大規模並列計算を行うことになるが、そのような状況で効率良く計算できるか把握しておくことは重要である。そこで九州大学の計算機システムの一つである、富士通 PRIMERGY RX200 S6 を用いて MHD シミュレーションコードのベンチマークを行った。5000 並列近くまで利用でき、weak scaling で性能を測定した。その結果実行効率は 30%を超え、非常に良い性能であったが、スケーラビリティが悪く、1000 並列時に比べ、5000 並列近くでは～10%の性能劣化が見えた。今後より多くの CPU を利用し、計算が行われることを考慮すると、この並列性能劣化は非常に大きな問題である。

宇宙プラズマでのペタスケール数値計算（惑星磁気圏シミュレーション）

深沢圭一郎[1, 2, 3]

1. 九州大学情報基盤研究開発センター
2. 九州大学宙空環境研究センター
3. CREST, JST

KYUSHU UNIVERSITY

March 1, 2012

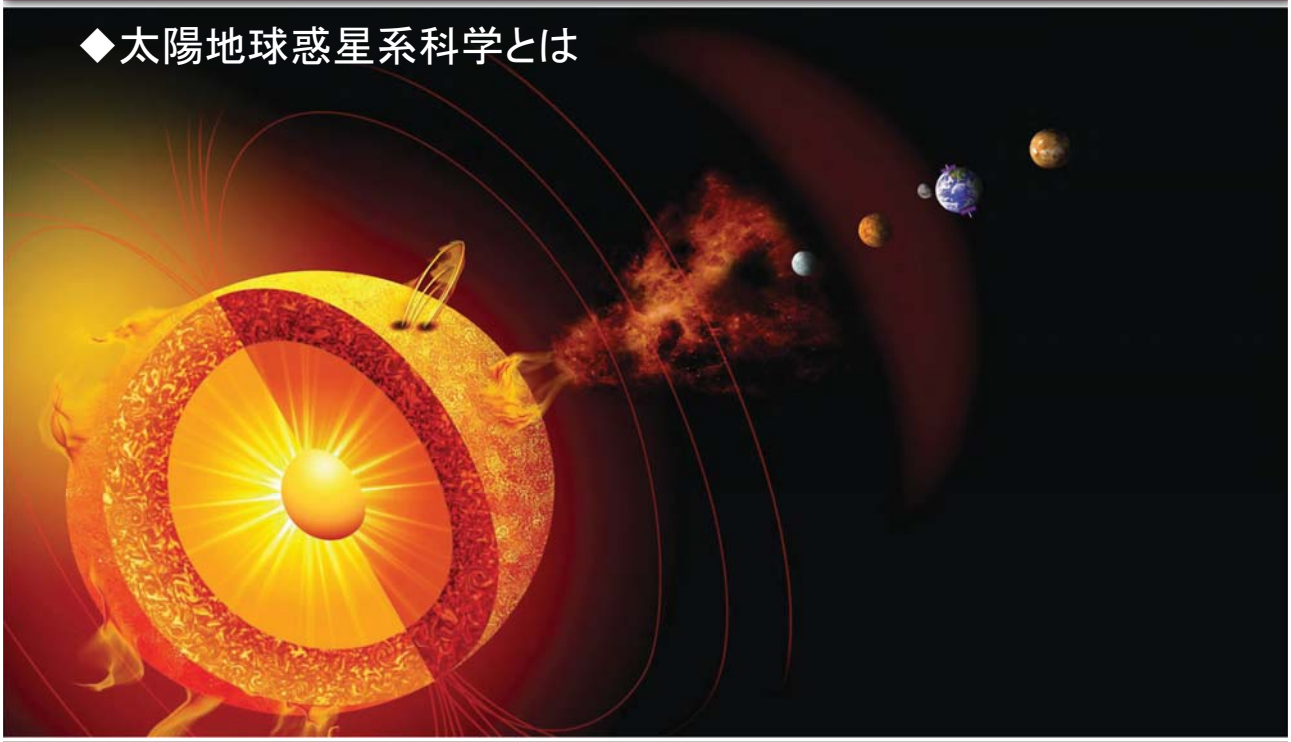
Kyushu University 2011
100th Anniversary

Context

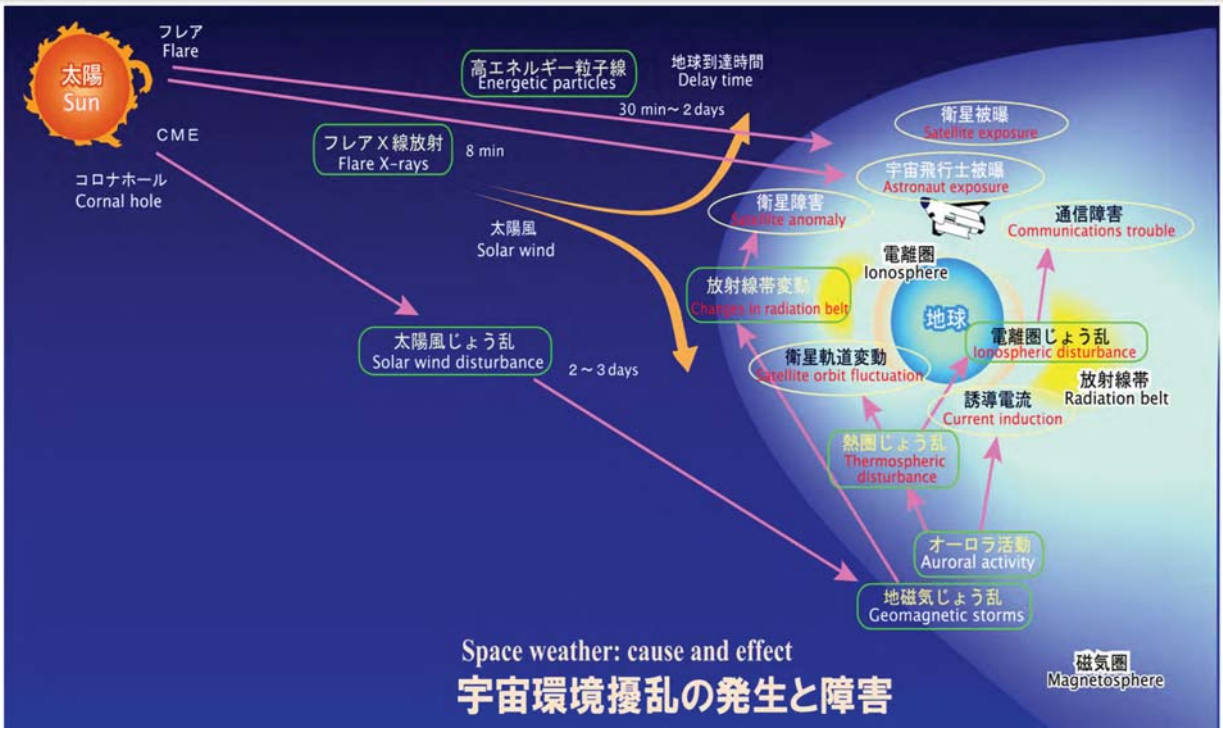
1. Introduction
   - ✓ 太陽地球惑星系科学の紹介
   - ✓ 惑星と特徴と磁気圏の構造
   - ✓ MHD方程式とVlasov方程式
2. 土星磁気圏のシミュレーション
   - ✓ プラズマ渦構造とオーロラの関連
3. MHDコードの性能測定
   - ✓ 様々な計算機システムでの性能
4. まとめ

Introduction -1

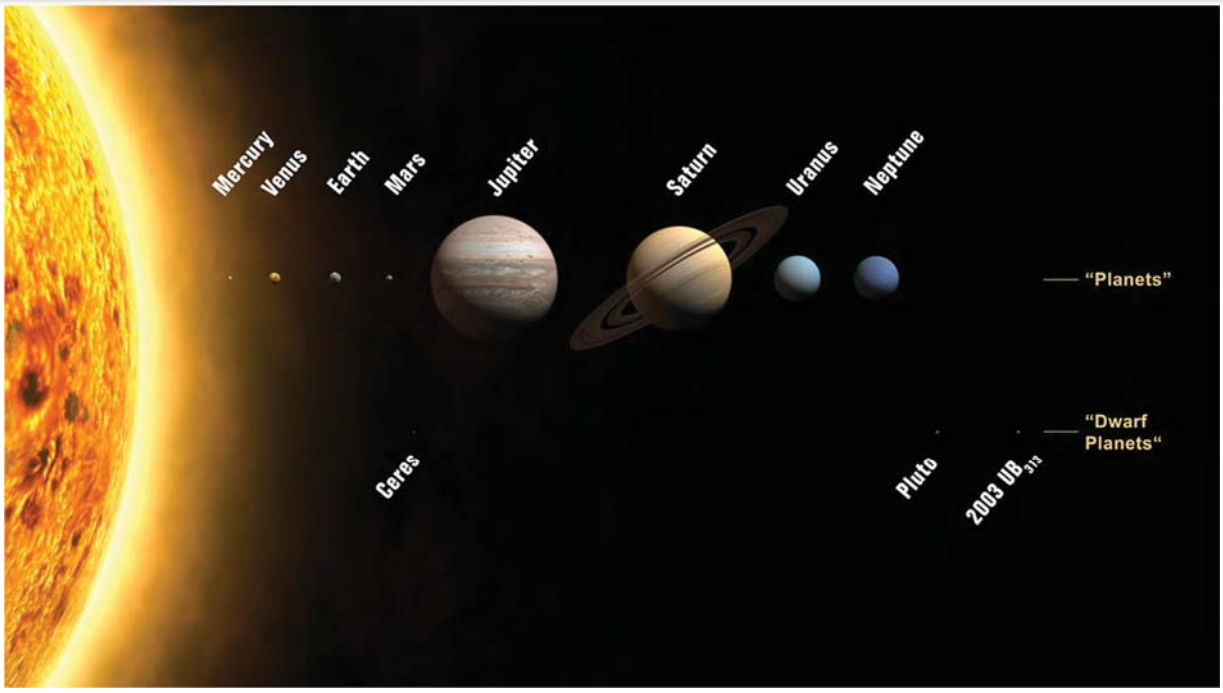
◆太陽地球惑星系科学とは



Introduction -2



Introduction -3



Introduction -4

◆地球、木星、土星における電磁気的特徴

	Jupiter	Saturn	Earth
Magnetic field [nT]	420,000	21,000	31,000
Magnetic polarity	N pole is north	N pole is north	N pole is south
Rotation period [hr]	10	10.65	24
Main plasma source	Io, ionosphere	Enceladus, ionosphere	ionosphere
Equatorial Radius [km]	71,492	60,268	6378
From Sun [A.U.]	5.2	9.55	1

木星は巨大な磁場と豊富なプラズマを持って高速自転する.
土星は豊富なプラズマを持って高速自転する。

Introduction -5

◆地球、土星磁気圏の形

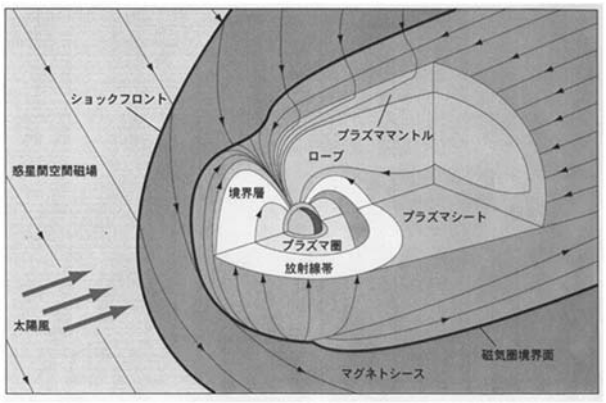


Fig. 1. A schematic of Terrestrial magnetosphere

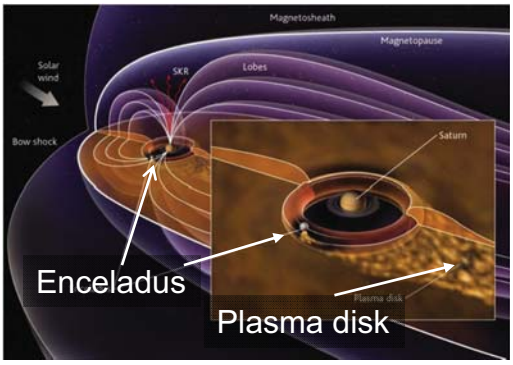


Fig. 2. Schematic of Saturn's magnetosphere [*Kivelson, 2006*]

土星は高速自転しているため、磁気圏内のプラズマ対流が複雑。

Introduction -6

◆**Spacecrafts**

- 8機の探査機が木星を観測
  - Pioneer10 (1973), Pioneer11 (1974), Voyager1 (1979), Voyager 2 (1979), Ulysses (1992), Galileo (1995-2003), Cassini (2000), and New Horizon (2007)
- 4機の探査機が土星を観測
  - Pioneer11 (1979), Voyager1 (1980), Voyager 2 (1981), Cassini (2004 - now),

- Future missions
  - JUNO (launch at 2011, arrive at Jupiter in 2016)
  - EJSM (Jupiter mission: launch after 2020)
  - TANDEM (Saturn mission: launch after 2020)

Motivation

◆巨大で複雑な惑星磁気圏を計算機シミュレーションで調べたい

□なぜ数値シミュレーションか

- 惑星の100倍以上に広がる磁気圏全体を見るには観測では難しく、シミュレーションしか無い。
- 衛星観測ではある時間の空間一点しかわからず、時空間の分離も難しく、物理現象の理解が難しい。
- 磁気圏は10m～10^9mスケールまでの現象を含むマルチスケール環境であり、シミュレーションに最適。

- 最終的には宇宙環境を理解し、予報することで人類が宇宙空間に出て行く際に安全な情報を提供したい。

電磁流体コード -1

◆宇宙プラズマを取り扱う方程式(1)

□Vlasov方程式

- 無衝突Boltzmann方程式とMaxwell方程式から成るプラズマの振る舞いを最も正確に表現できる方程式系

速度分布関数$f(\mathbf{x}, \mathbf{v}, t)$を考えると、

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}} + \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \frac{\partial f}{\partial \mathbf{v}} = 0$$

これとMaxwell方程式を連立して解く。

- しかし、位置(x, y, z)3次元、速度(v_x, v_y, v_z)3次元と時間から成る非線形方程式系で、解くことが困難。
意味のある$f(x, y, z, v_x, v_y, v_z, t)$を計算するには8PB程度のメモリが必要であり、現在の計算機ではメモリ不足。

電磁流体コード -2

◆宇宙プラズマを取り扱う方程式(2)

□MHD (Magnetohydrodynamics)方程式

– Vlasov方程式のn次モーメント取ることで、求められる。
0次(速度空間で積分)、1次(**v**かけて積分)、2次(**v**^{2}かけて積分)より、

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\mathbf{v}\rho)$$

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{1}{\rho}\nabla p + \frac{1}{\rho}\mathbf{J}\times\mathbf{B}$$

$$\frac{\partial p}{\partial t} = -(\mathbf{v}\cdot\nabla)p - \gamma p\nabla\cdot\mathbf{v}$$

を得る。これらと磁場の誘導方程式

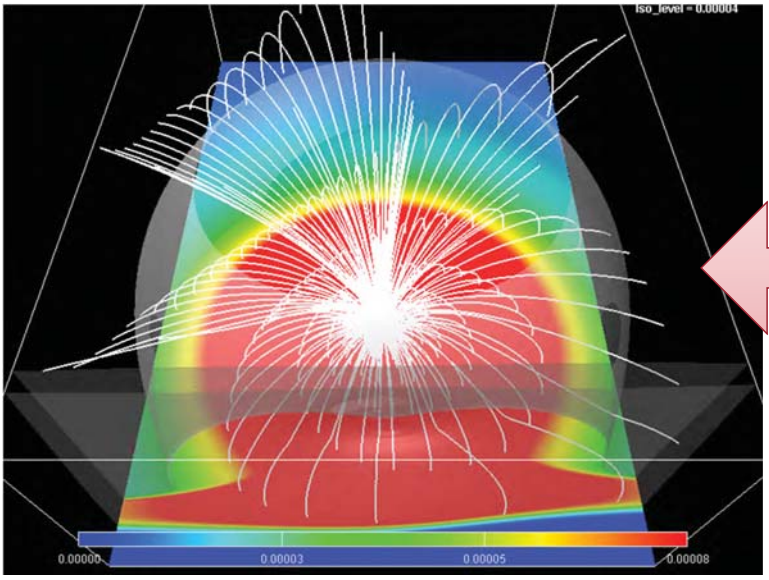
$\frac{\partial \mathbf{B}}{\partial t} = \nabla\times(\mathbf{v}\times\mathbf{B})$　をまとめてMHD方程式という。

この場合次元が$f(x, y, z, 8)$になり、使用メモリが激減する。

土星磁気圏シミュレーション1

◆土星磁気圏大規模シミュレーション

□渦構造とオーロラの関係を調べている。

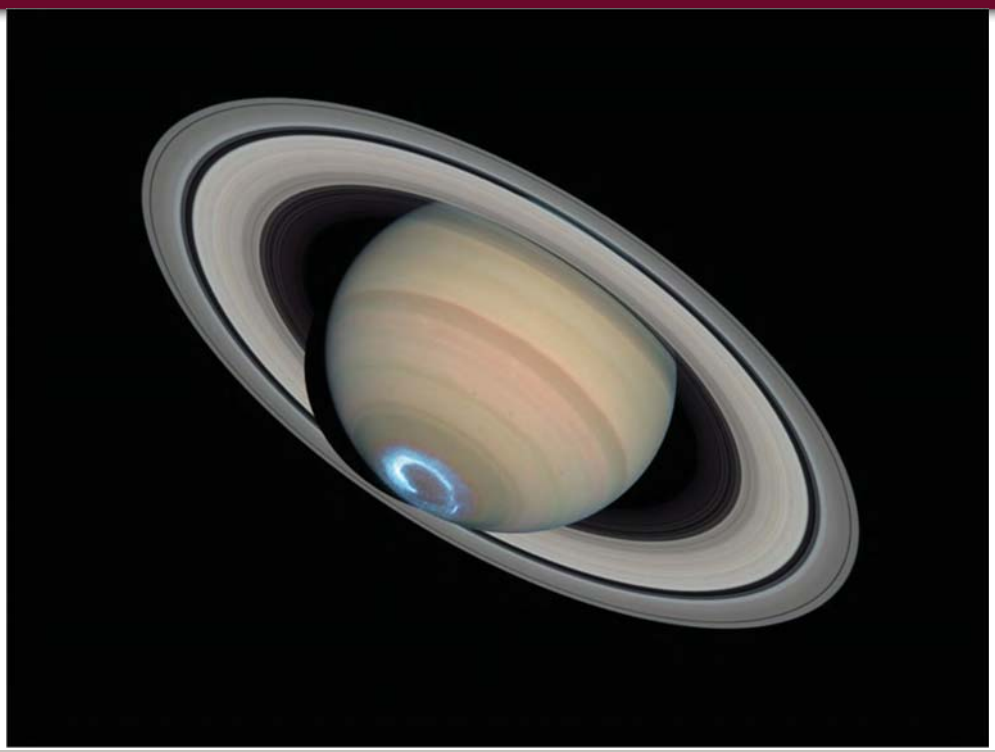


左側に渦構造が見えた後、右側にも渦構造が見える。

観測でも同様の構造が報告され始めている。

Fig. 3.　土星磁気圏赤道面における磁場強度[*Fukazawa et al.*, *JGR*, 2012]

土星にもオーロラがある



土星磁気圏シミュレーション2

◆ 沿磁力線電流と渦構造の関係

□沿磁力線電流の強い箇所から磁力線を伸ばすと. . .

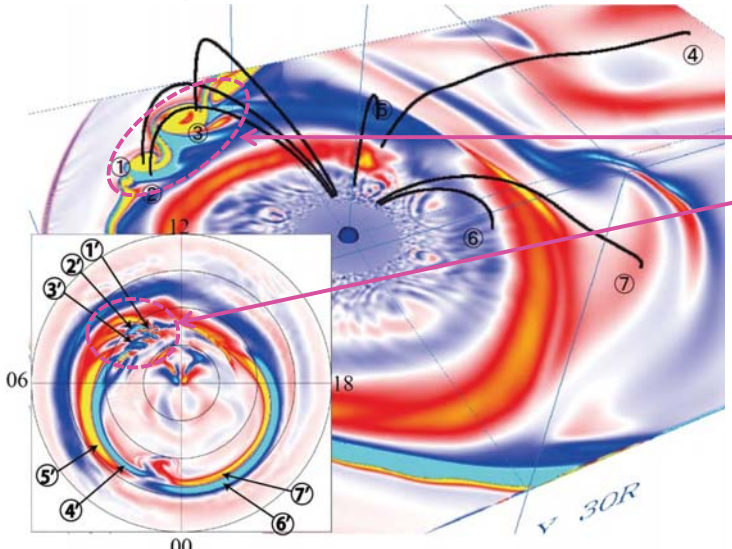


Fig. 4. 土星磁気圏における渦構造と沿磁力線電流の関係 [*Fukazawa et al., JGR*, 2012]

大規模計算による高精度なシミュレーションにより、土星渦構造と斑点オーロラ構造の関連性を初めて示唆。

観測結果に似た構造を再現。

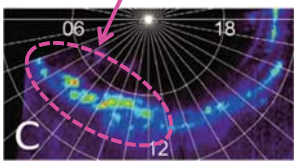


Fig. 5. 土星南極におけるオーロラの輝度[*Grodent et al., 2011*]

さらに高精度の計算へ

◆オーロラのような現象を精度良く解く必要がある

- 今までの計算では0.1 R_S（R_Sは土星半径 = 60,300 km）の格子幅を利用。
- マクロとミクロの遷移領域（MHD近似の限界領域）を計算するためには、0.01 R_Sの格子幅→$15{,}000^3$程度のグリッドが必要。
- CPUの周波数がそれほど上がらないと仮定すると
  ➡ $15{,}000^3$グリッドを現実的な時間で計算するためには200万コアが必要。
- 一方で、HA8000@東大を利用した8192並列の計算では、並列化効率が1024並列と比べ約5%低下している。
  ➡ 超大規模並列は大丈夫か?

MHDシミュレーションによる性能評価

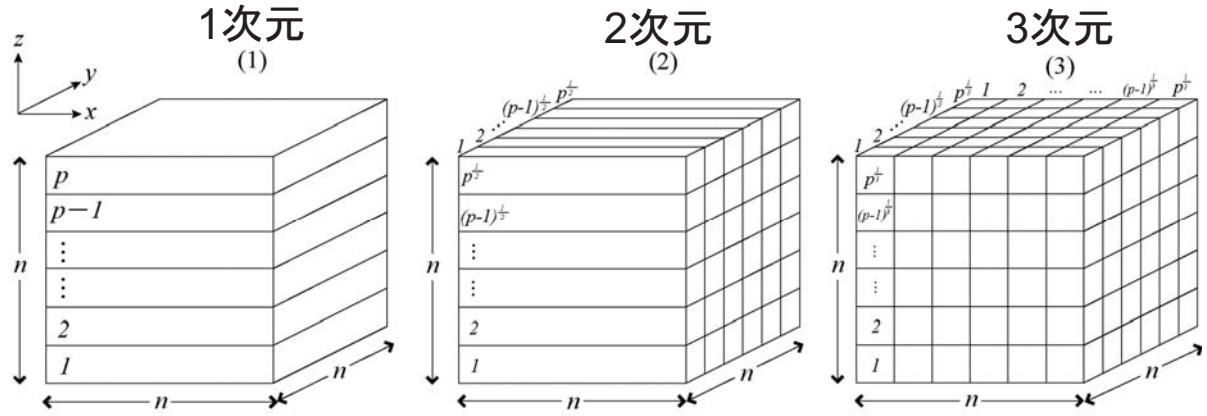
MHDシミュレーションの性能評価条件

◆ベンチマークセッティング

- □解法はLeap-frog+Lax Wendroff法の混合手法
  - 磁場を扱う方程式ではdiv・B=0を保証する手法を用いら無ければ成らず、適用可能手法に限界がある。
  - 今までのベンチマークとの比較、また、simpleな手法を用いることで、計算機の性能が出やすいと考えられる。
- □基本サイズ
  - 1coreあたり$f(x, y, z, 8)$ = 691200grid を設定。
  - 計算上この7倍の配列数を使用する。
- □MPI並列
  - のりしろは前後一つ。OpenMPは使用せず。
- □キャッシュ評価
  - スカラ機ではキャッシュヒット向上を見込んで、配列の並び替えを行うと性能が上がる機種がある（非x86系）ので、その効果も評価する
  - Type A : $f(i, j, k, m)$、Type B : $f(m, i, j, k)$

MHDコードにおける性能評価手法

◆3種類の領域分割法



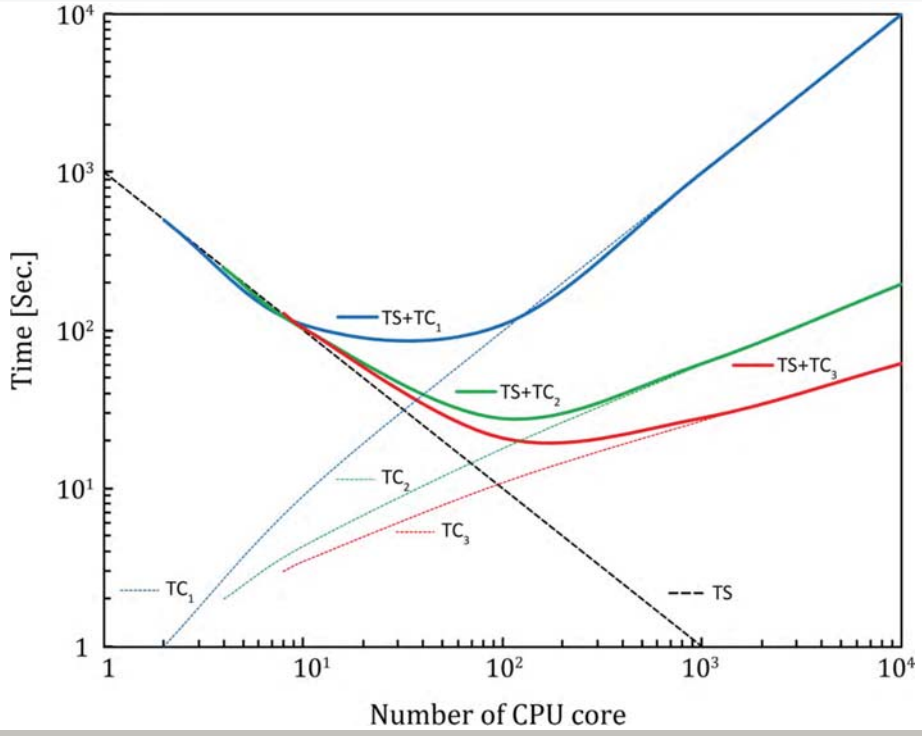
計算時間（T_S）

$$T_{S1} = k_2 n^2 (p-1) \qquad T_{S2} = k_2 n^2 (p-1) \qquad T_{S3} = k_2 n^2 (p-1)$$

通信時間（T_C）

$$T_{C1} = k_2 n^2 (p-1) \qquad T_{C2} = 2k_2 n^2 (p^{\frac{1}{2}} - 1) \qquad T_{C3} = 3k_2 n^2 (p^{\frac{1}{3}} - 1)$$

Estimated Time



PRIMERGY RX200 S6

◆2011年5月に稼働した九州大学の計算機システム

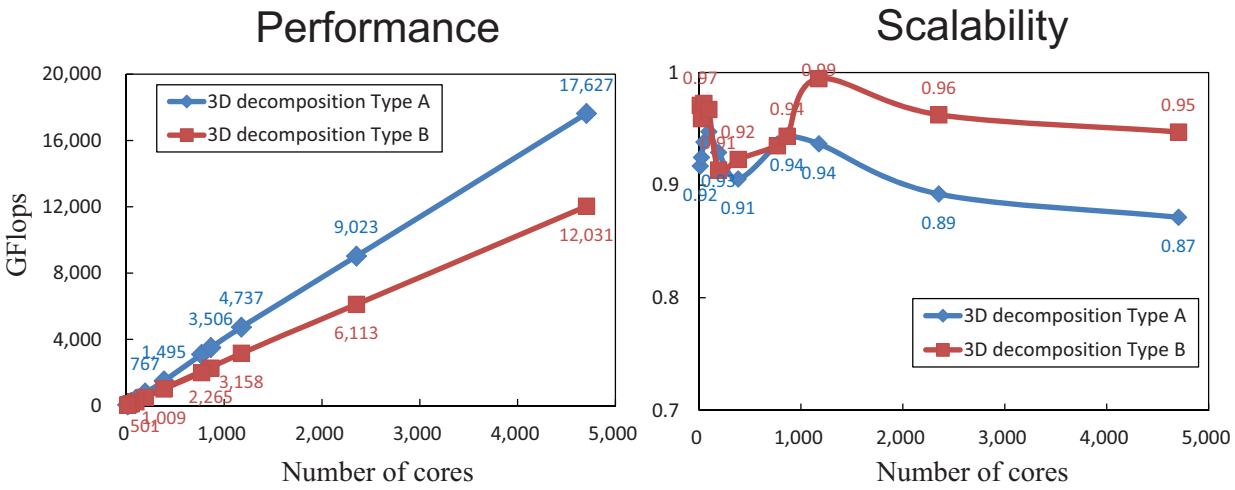
富士通製PCクラスタ型計算機

Westmereコア

		PRIMERGY RX200 S6
CPU	型式	6 core Xeon X5670
	周波数	2.93 GHz (70.32GFlops)
	Cache	L1: 64KB/core L2: 256MB/core L3: 12MB/CPU
Memory	Band幅	32GB/s /CPU
B/F値	32/70.32	0.45
Node	Core数	12
	メモリ	48GB
System	Node数	392 (4704core)
	理論性能	55.13TFlops
	Node間通信	InfiniBand 4x QDR (4GB/s)

MHDコードの性能評価(2)

◆sugoka（PRIMERGY RX200 S6）での測定



- 実効性能はTypeAで4700コア時に最大17TFlops（実行効率33%）出ている。
- スケーラビリティは低並列時に不安定で、緩やかに下がっており、TypeBのほうが10%近く高い。

MHDコードの性能評価(3)

◆現状の各種スパコンとの比較

	Core数/CPU数	実効性能[GFlops]	理論性能[GFlops]	実行効率[%]	領域分割	CPU種類
SX-9	64/64	2188	6553	33	2次元	ベクトル
SX-8R	8/8	80	282	28	1次元	ベクトル
HA8000	8192/1024	10038	75366	13	3次元A	Opteron
HX600	1024/256	2166	10240	21	3次元A	Opteron
FX1	1024/256	2081	10240	21	3次元B	SPARC64VII
SR16000	1344/672	5375	25267	21	3次元B	POWER6
PG S6	4704/144	18267	55130	33	3次元A	Xeon
PG S3	1536/768	2536	18432	14	3次元A	Xeon

MHDコードの性能評価(4)

◆SX-9を1としたときの各種スパコンCPU、coreの性能

	@core	@CPU	CPU種類
SX-9	1.0000	1.0000	ベクトル
SX-8R	0.2925	0.2925	ベクトル
HA8000	0.0358	0.1432	Opteron
HX600	0.0618	0.2474	Opteron
FX1	0.0594	0.2377	SPARC64VII
SR16000	0.1170	0.2340	POWER6
PG S6	0.1187	0.7122	Xeon
PG S3	0.0483	0.0966	Xeon

実アプリではFX1、SR16000やHX600が4CPUでSX-9の1CPUと同等の性能になっている。

PRIMEGY RX200 S6では1CPUでSX-9のCPUの7割近くの性能

Summary

◆宇宙プラズマでのペタスケール数値計算

□ 惑星磁気圏シミュレーション

- 観測では情報が足りないため、数値シミュレーションが力を発揮できる分野。
- 土星磁気圏では渦構造とオーロラの関係が観測から示唆されているが、シミュレーションでも確認出来ている。
- まだまだ計算資源が必要で理論的にはエクサまで対応可能。

□ MHDコードの性能測定

- sugokaでは高い実行効率が出ている一方で、スケーラビリティの低下が問題。
- 計算機システムごとに最適な並列化手法も違うため、ライブラリで対応可能だとうれしい。

ペタスケールのアプリケーションへ向けて

：日本原子力研究開発機構・システム計算科学センター・シミュレーション技術開発室の取り組み

日本原子力研究開発機構
システム計算科学センター
シミュレーション技術開発室
山田進、佐々成正、中村博樹、奥村雅彦、町田昌彦

理化学研究所
今村俊幸

日本原子力機構・システム計算科学センター・シミュレーション技術開発室では、現在、原子力研究開発における喫緊の課題に対し、最先端の大規模シミュレーションを通して、その課題解決を図るため、ペタスケール規模のシミュレーション技術の研究開発を理化学研究所等と連携し進めている。本公開資料では、まず、原子力機構全体の主たる大規模シミュレーションによる取組みを紹介し、原子力研究開発における大規模シミュレーションの役割とその成果について概観した後、当該研究室が取り組んでいるペタスケールライブラリ開発について、その現状と課題について述べる。また、現在、原子力分野において極めて大きな課題となっている福島原発事故後の環境修復の研究課題については、計算科学による研究の現状を紹介することとした。広範囲に拡散した放射性核種による汚染からの環境修復のための研究は、原子力分野がこれまでに経験してきた問題のレベルを遥かに超え、国家的課題となっており、分野を超えて日本（世界も）の英知を結集して取り組むべき課題であることから、本紹介を通して様々な研究者のご意見を伺い、また、可能であれば研究連携を実施して行きたいと考えている。

ペタスケールライブラリ開発において、当該研究室が特に力を入れているのは、行列の対角化に係るソルバー開発である。当該研究室では、原子炉構造材料、原子炉核燃料、そして原子力研究開発に関係する機能材料の第一原理計算を様々なレベル（超並列大規模シミュレーションによる核燃料の高温挙動や、構造材料スクリーニングのためのパラメータサーベイ・シミュレーション）で進めている他、次世代デバイス材料と考えられている強相関電子系の電子構造の高精度計算を実施するため、大規模行列の高速対角化に焦点を当て研究開発を進めてきた。その結果として、当該研究室では Eigen-K と名付けた「京」上にて高速に動作する行列対角化ソルバーの開発に成功し、研究室 HP より現在、公開している。尚、対象とする行列は密行列であり、1 万コア程度までは、ほぼリニアにスケールする並列性能を示し、1 万要素程度以上の行列になると、ScaLAPACK より遥かに十分に高い性能を示すことが分かっている。また、エルミート行列用のライブラリーも準備されてお

り、実対称及びエルミートの大規模密行列を高速に対角化するコードが公開されている。

福島原発事故後、当該研究室では放射性物質セシウムの土壌表面での吸着機構を明らかにするための計算科学研究課題に取り組んできた。土壌には、粘土粒子と称される直径２μｍ以下の細粒成分が多く含まれているが、その原子分子レベルでの吸着化学機構を明らかにするため、第一原理計算手法や分子動力学手法を駆使して研究を行っている。本発表ではその一端を紹介し、物質科学における計算科学研究がどれほど、現在立ちはだかる大きな課題（吸着土壌から脱離させ除染による廃棄土壌を減容化する）の解決に貢献できるかを議論した。物質科学から見ると、土壌のような極めて複雑な系に対し、効果的な議論を進めるためには、できる限り現象の普遍性を捉えるモデル化を進めて、正確な計算手法を用いて解くことと、大規模計算を実施し、非一様性の中での普遍性を逆に確認する必要がある。超並列計算が今後は重要な役割を果たすはずとして、今後の展開をできれば先導したいものと考えている。

ペタスケールのアプリケーションへ向けて
:シミュレーション技術開発室の取り組み

日本原子力研究開発機構
システム計算科学センター
シミュレーション技術開発室
町田昌彦

主たる研究協力者
ライブラリー開発:山田進、佐々成正、今村俊幸(電通大、現・理研)
シミュレーション研究:奥村雅彦、中村博樹

富士通SS研、2012年3月1日

発表内容

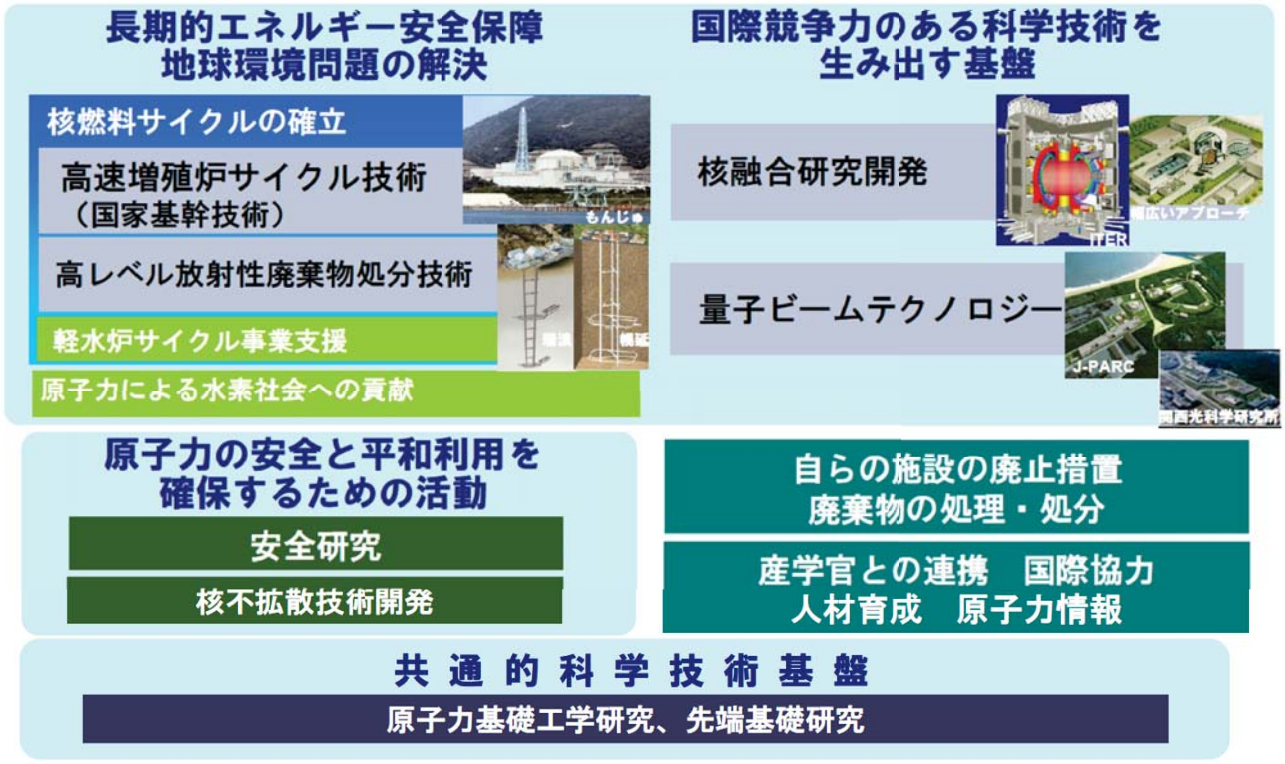
1、原子力機構の主たるアプリケーション

2、(ペタスケール)ライブラリー開発への取り組み

3、福島事故後の計算科学の取り組み

1、原子力機構の主たるアプリケーション

原子力研究開発4大プロジェクト



1、原子力機構の主たるアプリケーション

1. 核融合プラズマ乱流シミュレーション
2. 原子力機器内沸騰二相流解析
3. ナトリウム冷却高速炉における非定常流動現象
4. 第一原理計算による材料劣化メカニズム（粒界脆化）の研究
5. その他計算科学研究事例

1 核融合プラズマ乱流シミュレーション

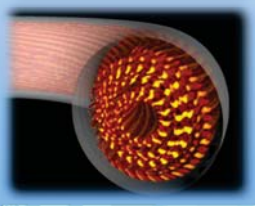
計算コード名：GT5D

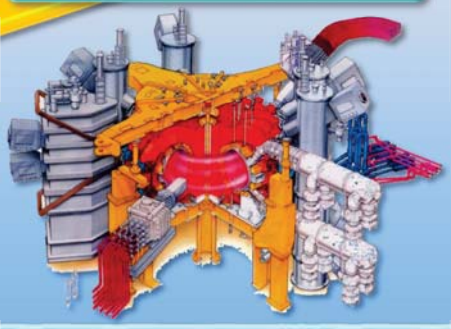
5次元運動論モデルによる核融合炉心全体のプラズマ乱流輸送シミュレーション

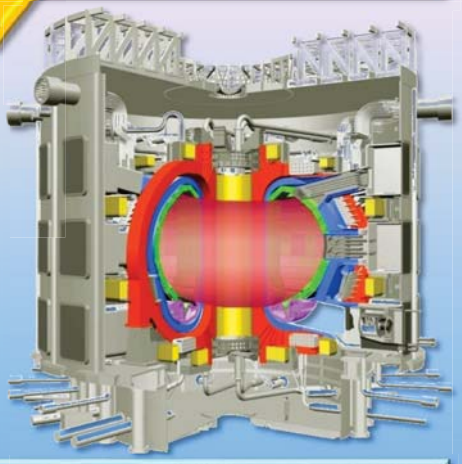
国際熱核融合実験炉 ITER

臨界プラズマ試験装置 JT60U

小型装置モデル（現状サイズ）







	小型装置	JT60U	ITER
炉心体積	$6m^3$	$60m^3$	$800m^3$
格子数	5.24×10^9	1.38×10^{11}	1.23×10^{12}
メモリ量	800GB	14TB	147TB
計算量（実効性能・日）	30TFlops・day	750TFlops・day	7PFlops・day

4 第一原理計算による 材料劣化メカニズム（粒界脆化）の研究

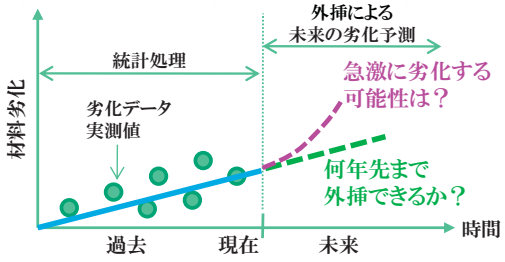
計算コード名：VASP

劣化メカニズムを把握しなければ、科学的・合理的根拠に基づく劣化予測ができない

計算の目的、概要

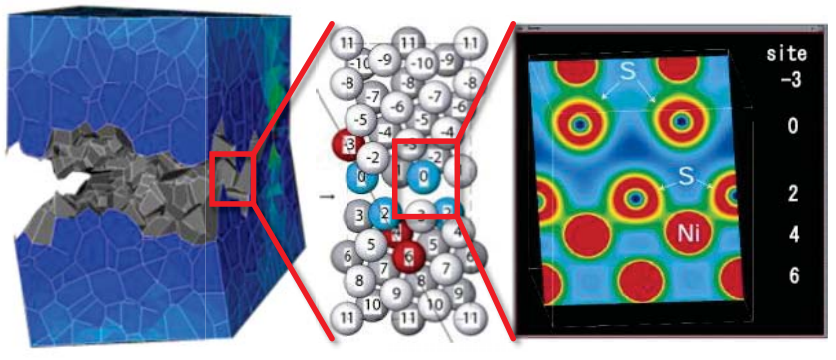
◆原子炉構造材料の劣化メカニズム解明を通して、原子炉の高経年化対策に科学的・合理的根拠を提供すること。

◆本研究では「粒界脆化」という劣化について、原子・電子レベルから、そのメカニズムを明らかにする。



計 算 内 容

◆結晶粒界の原子モデルを作り、第一原理計算によって不純物元素の粒界への集まりやすさ、粒界強度の変化を調べる。



ー主要なコードでのライブラリー

コード名	コード概要	主要ソルバー
GT5D	フォッカー・プランク方程式＋ポアソン方程式	連立一次方程式
TPFIT	Navier-Stokes方程式とポワソン方程式	連立一次方程式
AQUA	単相3次元汎用熱流動解析プログラム 乱流場k-ε モデル	パッケージ （？）
Exact_diag DMRG	ハミルトニアン行列の基底状態を計算	（疎行列＋密行列）対角化
NOPIC	Maxwell方程式と分極方程式	連立一次方程式？
TDDFT	実空間実時間発展第一原理計算	疎or密行列対角化？
VASP	密度汎関数法固体電子構造計算プログラム	パッケージ （FFT＋対角化）

2、（ペタスケール）ライブラリー開発への取り組み

- 密行列対角化⇒シミュレーションで高い需要
  - 複数固有状態を要求するシミュレーションは少なくない

量子力学に基づく物理・化学・生物その他の分野

＊戦略分野2の3課題へ既に提供済み

  - 高性能（高並列）固有値ソルバー開発

高度な（数学的背景＋計算機）知識が必要

計算科学力の証明

2、(ペタスケール)ライブラリー開発への取り組み

次世代計算機向け固有値ソルバー

Eigen_K (実対称) & Eigenh_K (エルミート行列向けソルバー)

これまでの開発実績(Altix/SGI, ES/NEC)
***ES上で最大70%以上の性能 (SC06・GBP Finalist)**
MPI/OpenMPハイブリッドモデルでの動作

目標機能

高いスケーラビリティ
＊1万コアまでほぼリニア
＊10万コア以降・ピークの10%を目指す
GPU／Cell／アクセラレータへの対応も計画中
全てのモデルで高性能達成(ScaLAPACKより高速)

2、(ペタスケール)ライブラリー開発への取り組み

機能

◆性能パラメタはScaLAPACKと異なり分散方式と独立に動的に制御

◆BLASも積極的に利用(但し、スレッド呼び出し方式)

◆マルチコア向けアルゴリズムの採用

Narrow-band-reductionアルゴリズム+帯行列向け分割統治法

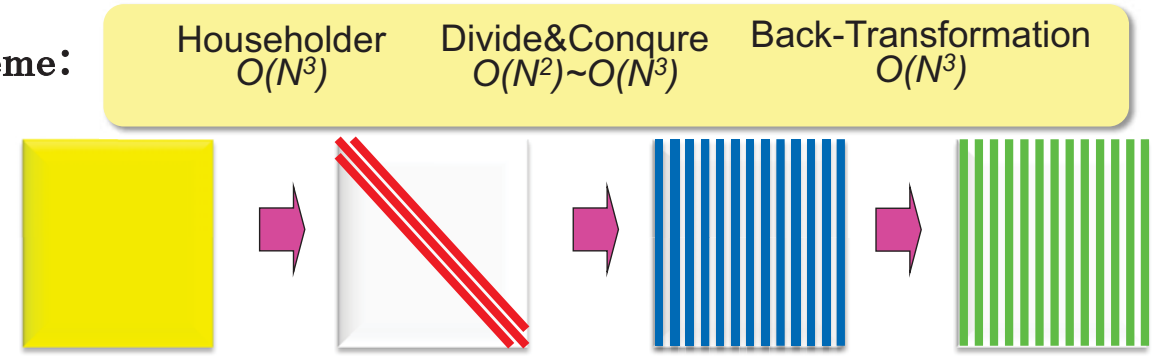
2、(ペタスケール)ライブラリー開発への取り組み

- **Eigen_S:** 標準的な3段階アルゴリズム:
  - ハウスホルダー3重対格化法
  - Cuppenの分割統治法
  - ハウスホルダー逆変換

SC2006・GBP Finalist
最大効率70%/ ES

標準的な方法であり、実装での高速化に注力。
ScaLAPACKのpdsyevdと機能的に同等

Scheme:

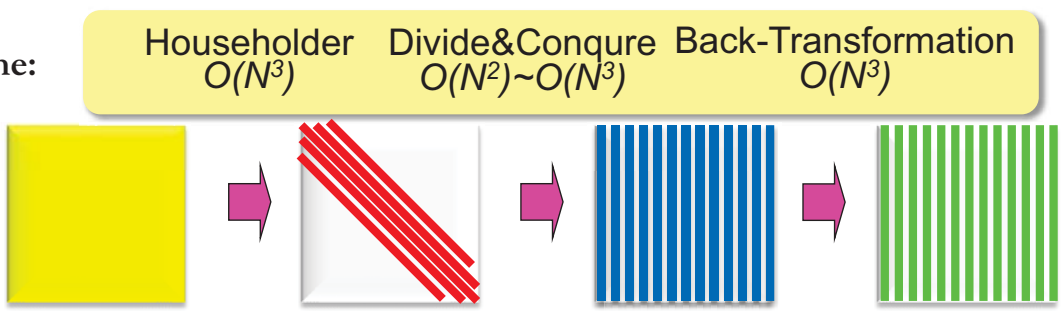


2、(ペタスケール)ライブラリー開発への取り組み

- **Eigen_K**: Eigen_Sを性能上限メモリバンド幅に改良
  - **ハウスホルダー5重対角化法**
  - **帯対称行列向け分割統治法**
  - ハウスホルダー逆変換

2つのアルゴリズムの並列版を新規に開発
Eigen_sと同様の実装と合わせて高速化を実施。
更なる高速化が必要な場面で推奨される

Scheme:



2、(ペタスケール)ライブラリー開発への取り組み

超並列化:

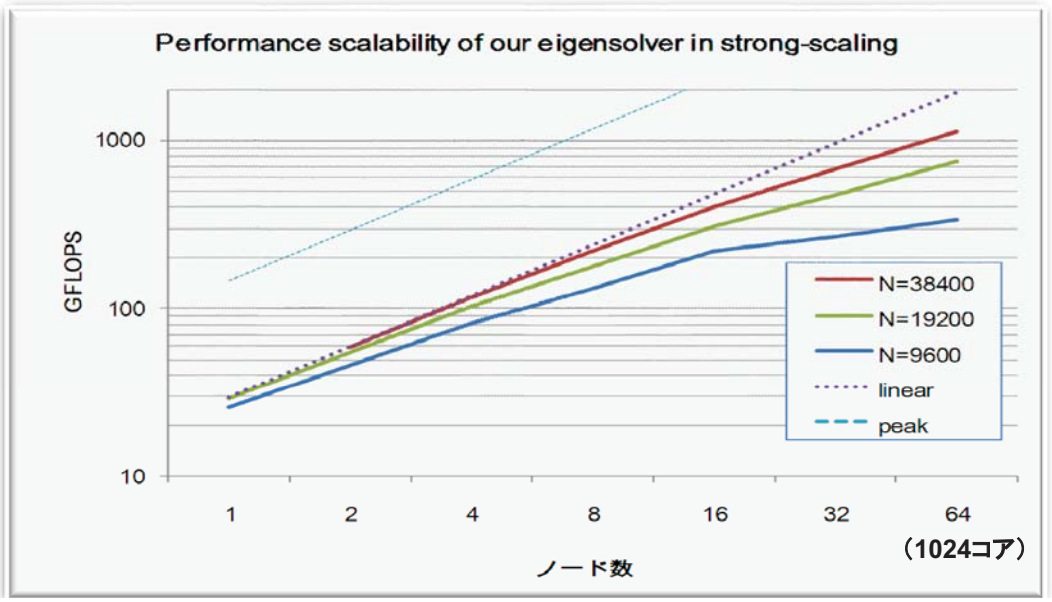
- 「京」で推奨される全ての並列化モデルを採用
  - MPI: 分散メモリ並列処理
  - OpenMP: 共有メモリ型スレッド並列処理
  - SIMD: 機械語レベル並列処理

- Tofuインターコネクトを有効利用
  - ノードを**2次元グループに分け**、**行ー列それぞれで閉じた通信**に限定(1対1通信、集団通信)
  - 通信量ならびに負荷均衡を最小にする、2次元サイクリックデータ分割を採用

2、(ペタスケール)ライブラリー開発への取り組み

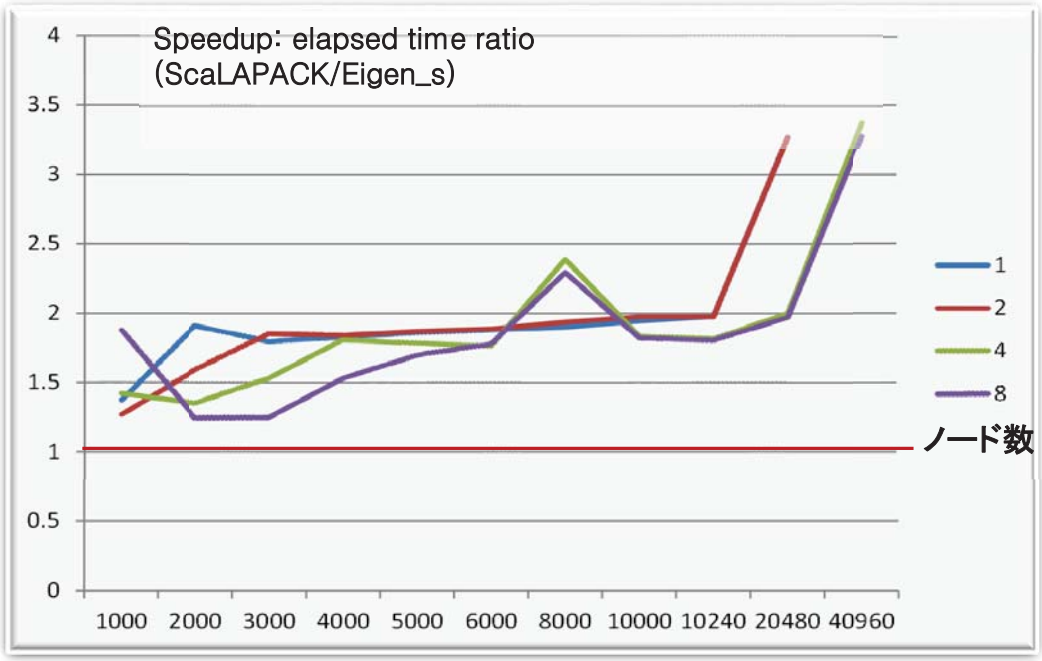
動作実績

- T2Kスパコンでの実測
  最大4096コアまでMPI+OpenMPのハイブリッド並列環境動作()
  (Flat MPIでも**8192コアまで動作検証済み**)



2、(ペタスケール)ライブラリー開発への取り組み

- T2Kスパコンでの実測
  既存ソフトウェアScaLAPACKよりも高速に動作する

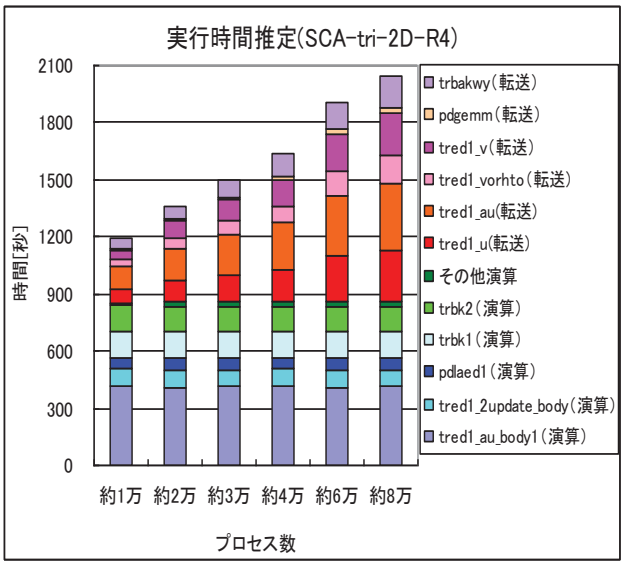


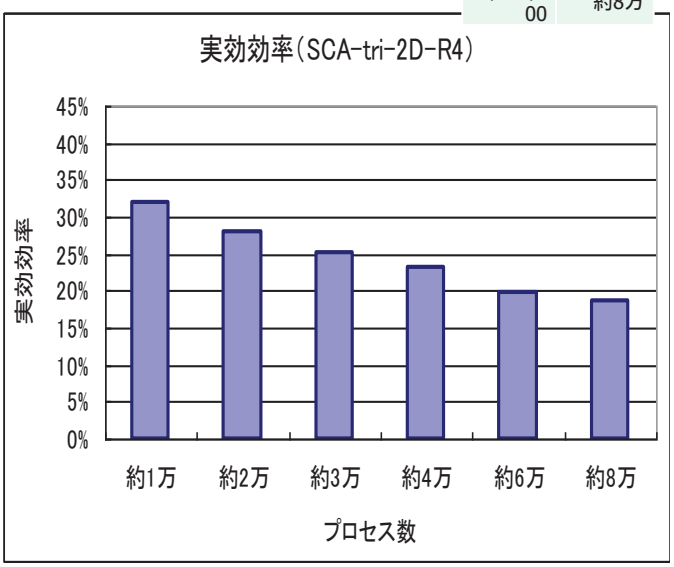
2、(ペタスケール)ライブラリー開発への取り組み

- 性能予測:Weak Scaling (Eigen_S)

  T2Kスパコンでの実測から演算量、通信量(回数)を算出
  「京」コンピュータの想定される性能から推定値を計算

N	プロセス数
437,149	約1万
549,908	約2万
693,452	約3万
794,145	約4万
908,238	約6万
1,000,000	約8万



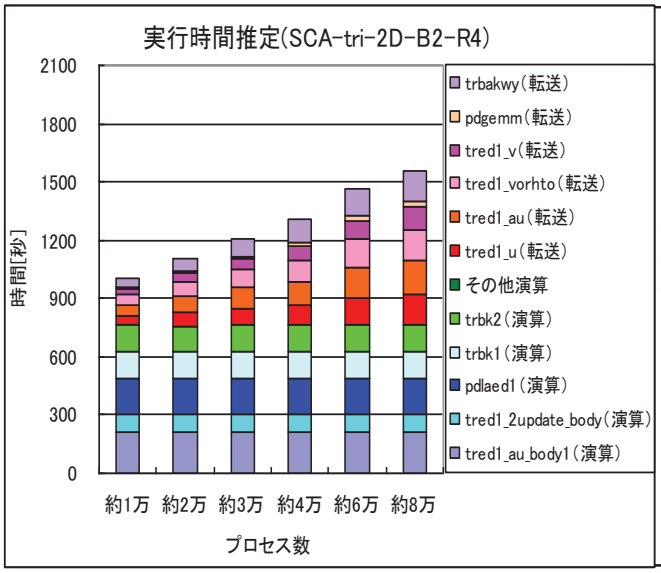


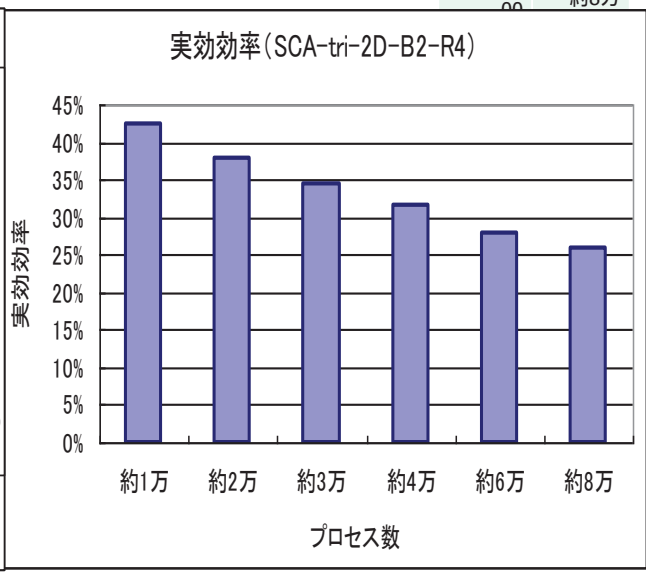
2、(ペタスケール)ライブラリー開発への取り組み

■ 性能予測:Weak scaling (Eigen_K)

T2Kスパコンでの実測から演算量、通信量(回数)を算出
「京」コンピュータの想定される性能から推定値を計算

N	プロセス数
437,149	約1万
549,908	約2万
693,452	約3万
794,145	約4万
908,238	約6万
1,000,000	約8万



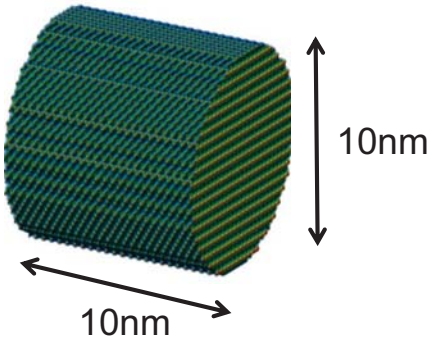


応用例

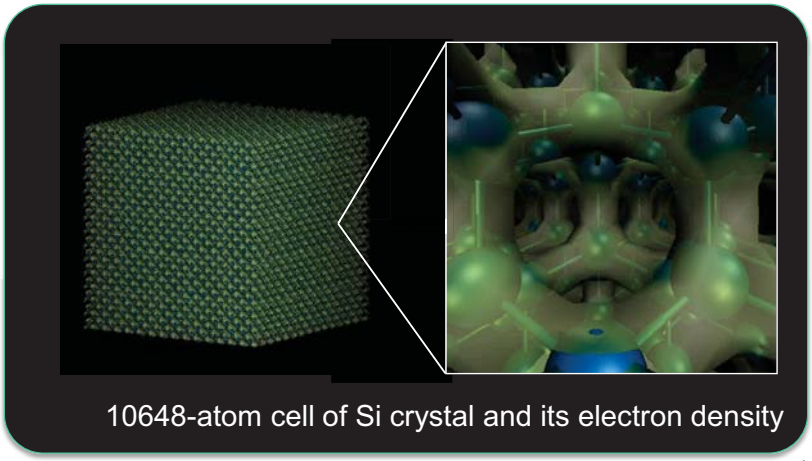
SC2012 Gordon Bell Prize

RSDFT(Real Space Density Functional Theory code)

- First principle electronic structure simulation
- Implemented real space method (not use FFT)
- Main procedure are conjugate-gradient method, ortho-normalization, and subspace diagonalization
- Use MPI + OpenMP
- Parallelized in grid space and orbitals
- Use DGEMM, and eigensolver library (ScaLAPACK ,**EIGEN**)



Silicon nanowire
39,696 atoms



10648-atom cell of Si crystal and its electron density

Performance of challenging simulations

- Si nw more than 100,000 atoms -

Si nw, 107,292 atoms, Grids:576x576x192, Orbitals:229,824

Procedure block	Execution time (sec.)	Computation time (sec.)	Communication time (sec.)				Performance (PFLOPS/%)
			Adjacent /space	Global /space	Global /orbital	Wait /orbital	
SCF	5,456.21	4,417.15	83.18	899.05	15.87	40.93	3.08/ 43.63
DIAG	3,710.01	3,218.72	27.70	458.87	4.70	-	2.72/ 38.52
MatE	1,084.70	717.45	27.70	337.85	4.70	-	3.09/ 43.72
EigenSolve	1,322.16	879.61	-	442.39	-	-	0.04/ 0.61
RotV	1,298.16	1,177.15	-	121.01	-	-	5.18/ 73.25
CG	209.29	57.66	55.48	96.14	0.01	-	0.05/ 0.74
GS	1,536.90	1,140.76	-	344.04	11.16	40.93	4.37/ 61.87

Performance of EIGEN (detail)

Si nw, 107,292 atoms , Grids:576x576x192, Orbitals:229,824

Procedure block	Total (sec.)	Computation(sec.)	Communication(sec.)
TRD1 (Householder)	853.443	141.123	712.320
DC (Divide&Conqure)	72.008	72.006	0.0002
TRKBAKWY (Back Transform)	35.281	13.992	21.360

Communication time of TRD1 is most largest.

Is this performance too low?

- Matrix element is too small compare to total parallel number.
- It is not suitable to map to Tofu network topology.
  - ✓ EIGEN: 2 dimension V.S OTHERS: 4 dimension

Benchmark of EIGEN

Nearly Weak scaling

Element of matrix	Number of process	Execution time(sec.)	Breakdown			Performance / core	
			TRD (sec.)	DC (sec.)	TRKBAK (sec.)	GFLOPS	Efficiency (%)
151,448	288 (16x18)	1,085.6	694.8	84.3	306.3	37.0	29.0
191,232	576 (24x24)	1,167.1	790.1	66.3	311.1	34.6	27.1
240,768	1,152 (32x36)	1,287.1	846.9	116.6	323.6	31.3	24.5
302,592	2,304 (48x48)	1,355.1	949.5	76.3	329.2	29.5	23.1
382,464	**4,608** (64x72)	1,713.7	1,164.3	184.5	364.8	23.6	**18.4**

3、福島事故後の計算科学の取り組み

計算物性物理による原子力災害への取り組み :土壌汚染の理解及びその他

日本原子力研究開発機構・システム計算科学センター・シミュレーション技術開発室

奥村雅彦*　中村博樹　町田昌彦

*福島技術本部　福島環境安全センター

謝辞:河村雄行(岡山大)、吉田善行・大貫敏彦・山岸功・斎藤公明(原子力機構)

放射性物質の土壌への吸着の物理化学形態

放射性物質（セシウム）の土壌汚染の実態

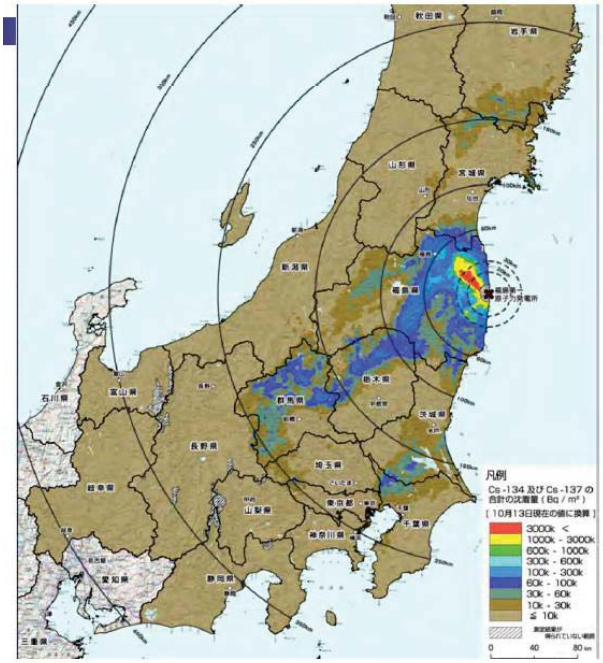
問題：　土壌の表層にのみ強く固定
（表層からの移動が少ない）

・放射能の減衰が遅い

高い放射能汚染地域が残存

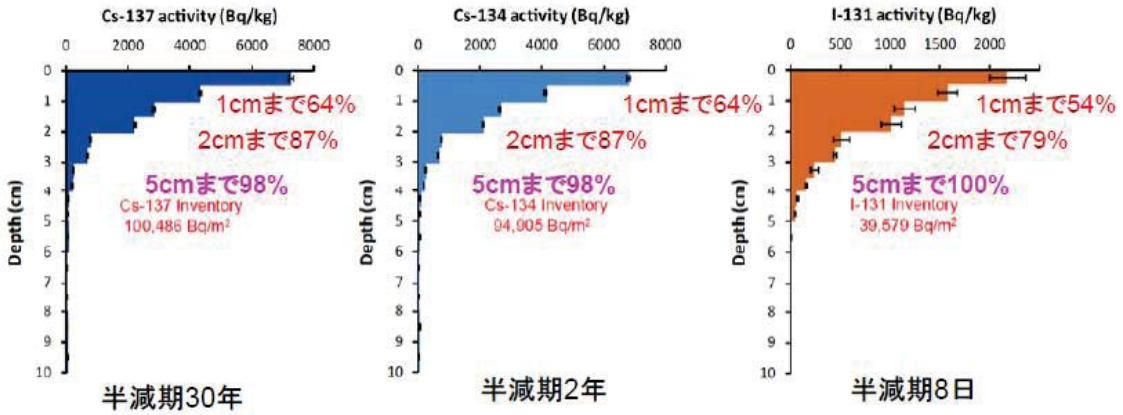
・地下水への移行やその他の移行による被害が少ない

汚染の拡大が抑制される



放射性物質の土壌への吸着の物理化学形態

放射性物質（セシウム）の土壌汚染の実態 → 表層に留まる汚染



Kato, H., et al., Depth distribution of 137Cs, 134Cs, and 131I in soil profile after Fukushima Daiichi Nuclear Power Plant Accident, Journal of Environmental Radioactivity (2011), doi:10.1016/j.jenvrad.2011.10.003

筑波大・恩田教授によるスライドから

放射性物質の土壌への吸着の物理化学形態

放射性物質（セシウム）の土壌汚染の実態 → 表層に留まる汚染

Storage and Migration of Fallout Strontium-90 and Cesium-137 for Over 40 Years in the Surface Soil of Nagasaki

Yasunori Mahara*

- Vertical migration of 90-Sr and 137-Cs was investigated in an unsaturated soil layer in the Nishiyama area of Nagasaki.
- The in situ migration rates of 90-Sr and 137- Cs were estimated to be 4.2 mm/yr and 1.0 mm/yr
- Fallout of 137-Cs and 90-Sr have remained in the surface soil for a long period of time
- More than 95% of ^{137}Cs was to a depth of 0.1 m, no ^{137}Cs was detected in groundwater.
- ^{90}Sr was more mobile.

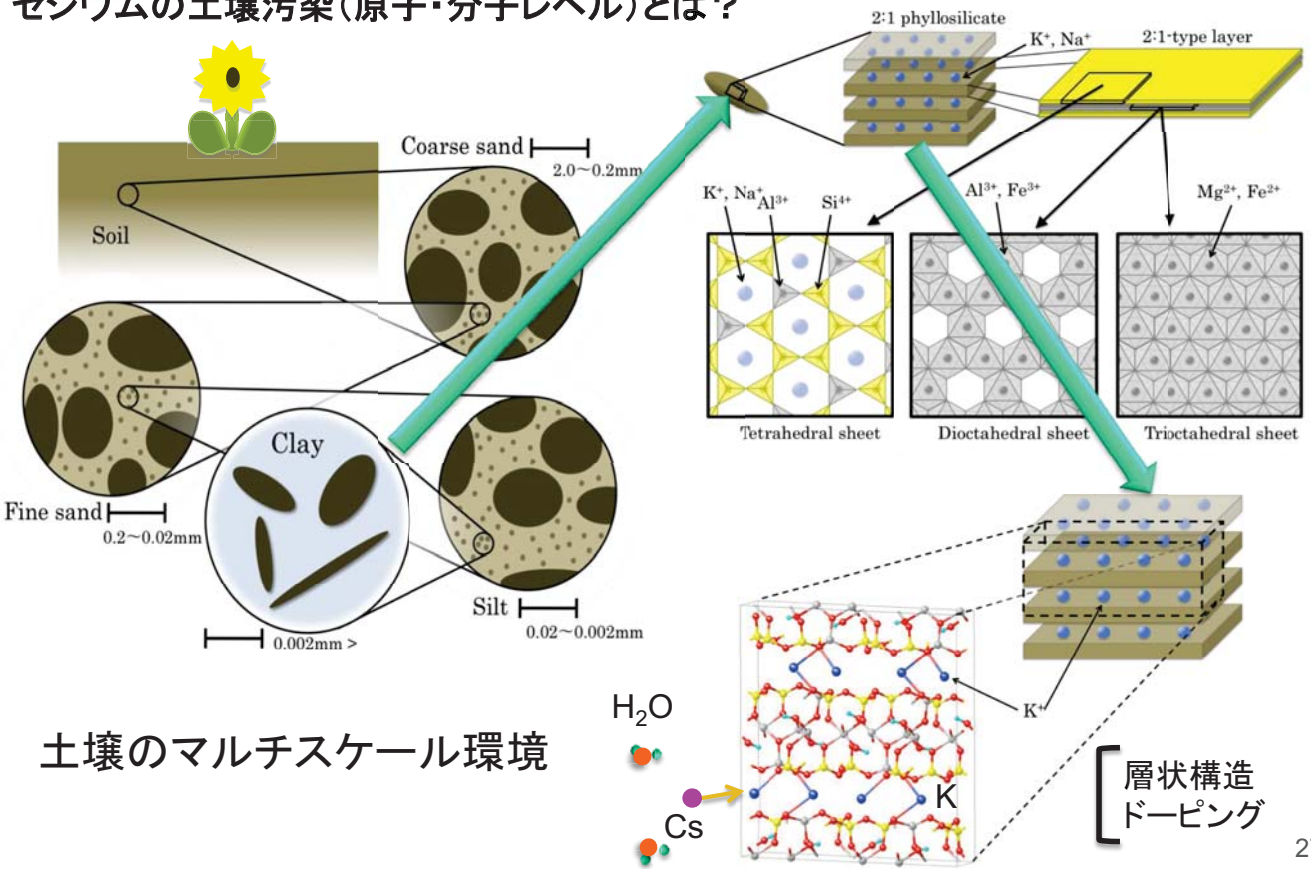
J. Environ. Qual. 22:722–730 (1993).

固着したセシウム脱離は困難！

筑波大・恩田教授によるスライドから

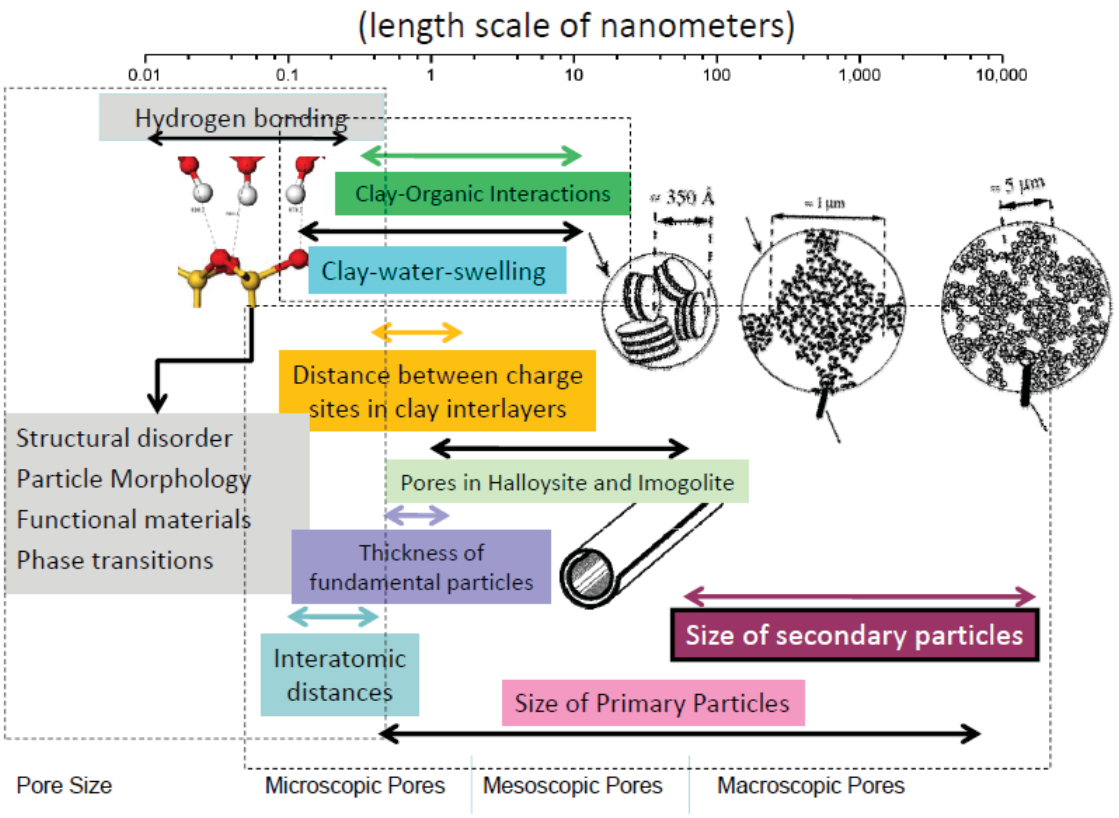
放射性物質の土壌への吸着の物理化学形態

セシウムの土壌汚染（原子・分子レベル）とは？



土壌のマルチスケール環境

層状構造
ドーピング



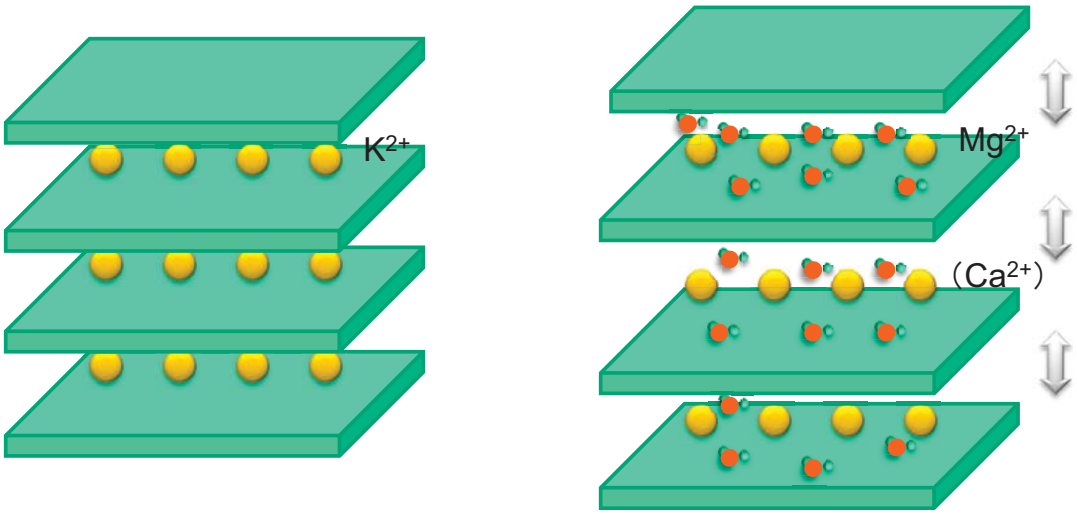
http://www.iai.ga.a.u-tokyo.ac.jp/mizo/seminar/cliffslide.pdf

放射性物質の土壌への吸着の物理化学形態

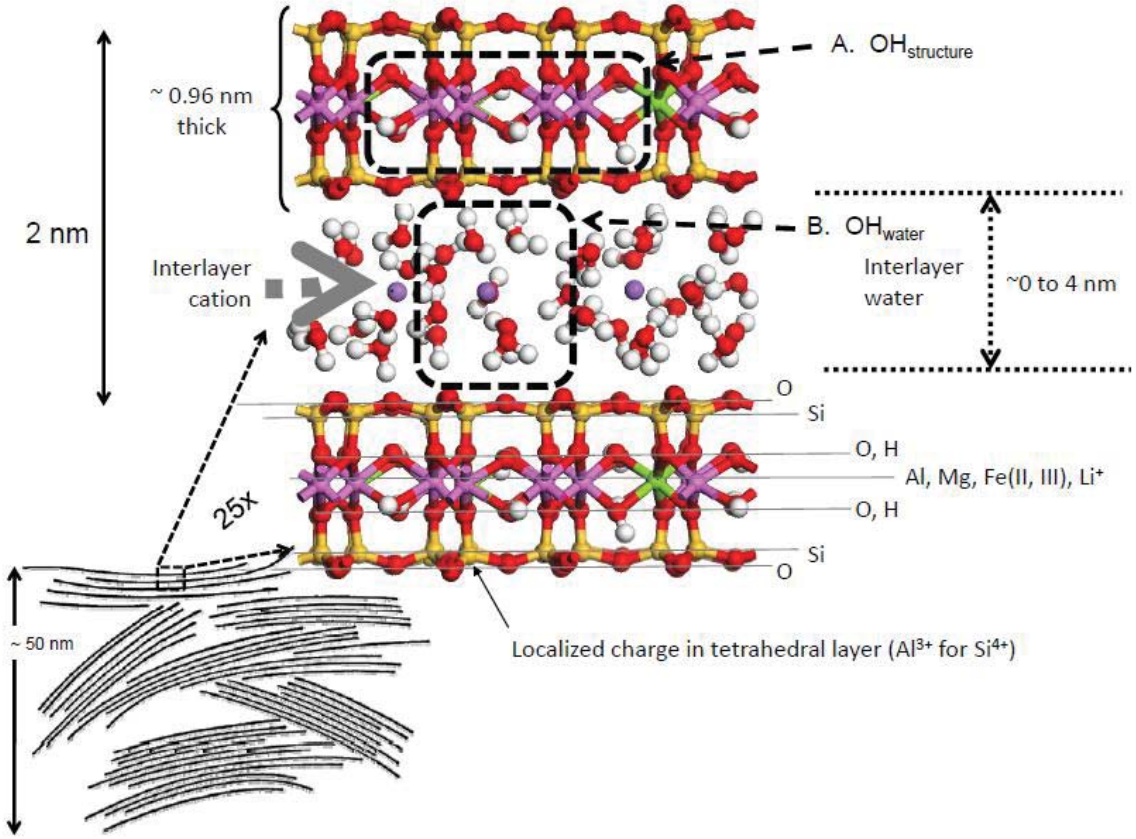
風化土壌

比較的新しい土壌

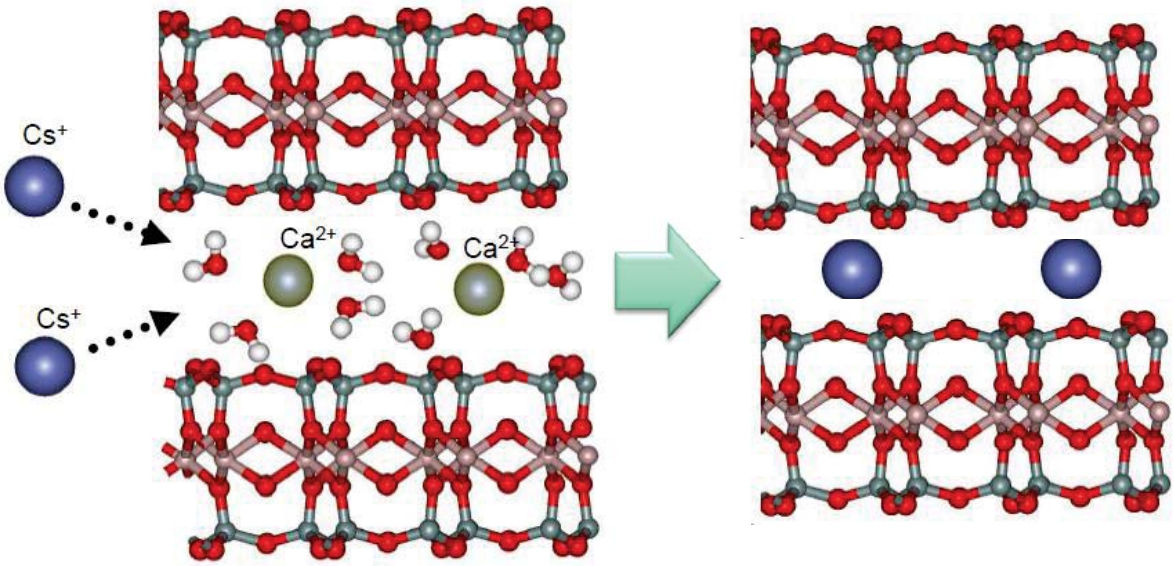
福島の土壌に近い？



水分が挟まり膨潤な粘土



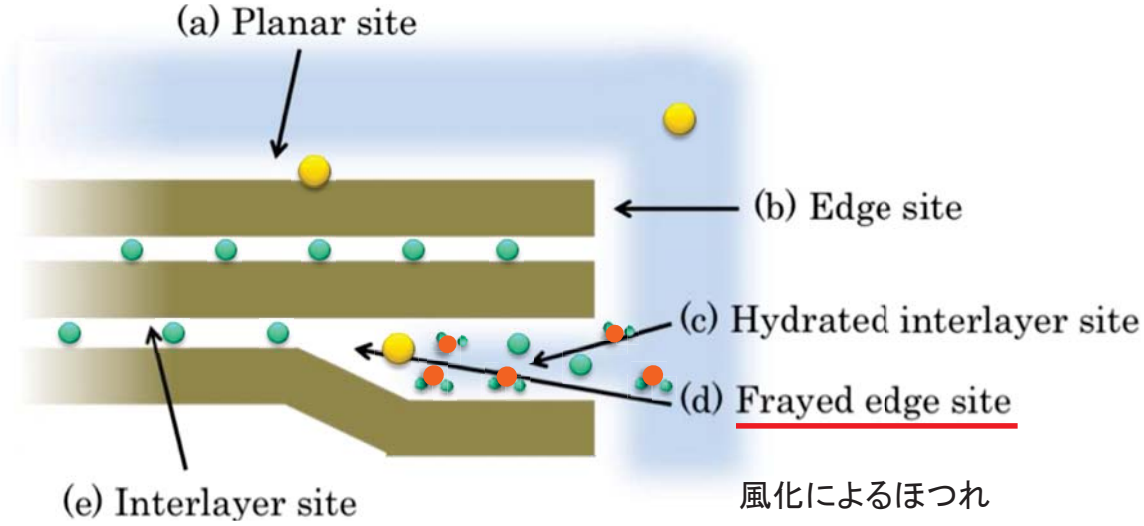
放射性セシウムの吸着シナリオ（新しい土壌粘土鉱物）



Johnston et al., Langmuir 17(12) 3712-3718

NHK・サイエンスゼロ（2/17）

放射性セシウムの吸着シナリオ（古い土壌粘土鉱物）



風化によるほつれ

放射性物質の土壌への吸着の物理化学形態

計算科学の研究対象

1、土壌粘土鉱物の第一原理計算（基本的計算）

セシウムの安定性を実際に確認する

2、シナリオを確かめる（第一原理MD or QM/MM)

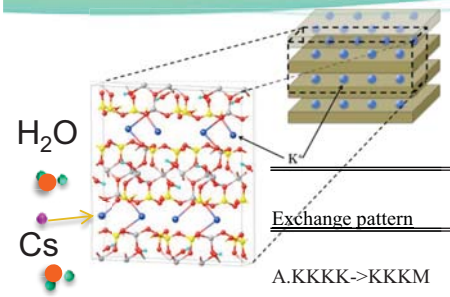
吸着前後のエネルギー変化の評価

3、セシウム脱離のダイナミクス

物理的アプローチ（マイクロ波やレーザー照射、加熱による揮発）

化学的アプローチ（イオン交換その他）

得られた研究成果（奥村・中村・町田）



各種雲母様粘土鉱物

↓

イオン交換エネルギー表（5種鉱物）

↓

但しバルク状態での交換

Csの化学的安定性

- 水中の陽イオンの水和エネルギー
- 陽イオンの置換による結晶歪みエネルギー

Exchange pattern	Li	Na	K	Rb	Cs	NH4
A.KKKK->KKKM	72.08	30.61		3.87	22.82	-187.74
B.KKKCs->KKMCs	86.65	39.37		-2.11	7.62	-189.1
C.KKKCs->KMKCs	82.7	36.18		-0.45	11.48	-192.23
D.KKCsCs->KMCsCs	90.33	39.75		-4.53	1.61	-196.91
E.KCsCsCs->MCsCsCs	100.36	47.04		-8.28	-8.05	-194.58
F.KKKCs->KKKM	49.26	7.79	-22.82	-18.95		-210.56
G.KKCsCs->KKMCs	79.03	31.75	-7.62	-9.73		-196.72
H.KCsKCs->KMKCs	71.22	24.7	-11.48	-11.93		-203.71
I.KCsCsCs->KMCsCs	88.72	38.14	-1.61	-6.14		-198.52
J.CsCsCsCs->MCsCsCs	108.41	55.09	8.05	-0.23		-186.53

アンモニアがイオン交換体としてあらゆる雲母様粘土鉱物に対し有効

まとめと今後の課題

- **原子力研究開発における計算科学でのライブラリー**
  - **連立一次方程式、行列固有値（密＆疎）、FFT等が使われている。**
  - **各ソルバーの移植と性能評価は可能。**
  - **現在は密行列固有値ソルバー(「京」)開発。**
  - **核融合反復法ソルバー開発（「京」）＋4倍精度BLAS等の開発**

- **福島事故以後に大きく変わった計算科学**
  - **土壌汚染の原子・分子レベルでの理解⇒効率的除染手法の開発**
  - **放射性物質の生体内輸送の原子・分子レベルでの理解**
  - **完成度の高い計算コード（第一原理計算、分子軌道計算）が必要！**

JAXA における数値計算ライブラリの利用状況と OPL 適用評価

松尾裕一（JAXA）

本報告では、宇宙航空研究開発機構（JAXA)における数値計算ライブラリ（以下、ライブラリ）の利用動向調査と幾つかのアプリケーション（以下、アプリ）に対する Open Petascale Library(OPL)の適用評価の結果、さらにそれらを踏まえた分析と考察、最後に今後の当該分野におけるライブラリ利用に関する私見を述べる。

JAXA のスパコンシステムは、富士通製 FX1 の 3,008 ノード（SPARC64 VII [2.5GHz、4 コア]、120TFLOPS、94TB メモリ、DDR IB によるファットツリーIC）を中核とするメインシステム（「JSS-M」と呼ばれる）と、大規模共有メモリやベクトルから成るサブシステムから成る。アプリの 9 割は計算流体力学（CFD）であり、計算コードの大半は内製[1]、学術系の大規模ジョブより工学系のパラメータスタディ用ジョブ（並列度は 256 以下）が多い特徴がある。この JSS-M システムにおけるライブラリの利用動向を 2012 年の 1 月から 2 月にかけて調査[2]したところ、延べ 350 人程度のシステム利用者のうち、ライブラリ利用者は、（コンパイラが自動で呼び出している場合も含め）5 割以上あるものの、直接ライブラリを呼びだしている利用者はそれほど多くはなく（述べ 50 人程度）、用途も画一的限定的であり、プロセス並列版の利用といった高度利用は行われていないということがわかった。（調査期間がやや短いが）このような結果は、ベクトルの時代から「メモリを節約するためにコードを全部自分で書く」という分野特有の慣習や、コードの長寿命性（一旦作ったコードは 10 年程度は使う）ゆえ、最初からある程度予想されていたとはいえ、そういった流れとは異なる新しい利用も特には進んでいなかった。調査結果を分析する傍ら、ライブラリを利用している 2 件の実アプリを抽出し、FFT と複素連立一次方程式につき OPL の適用を試みた。もともとは SSLII のライブラリを呼び出している部分（dcftm,dlcx）に、OPL の FFTE（zfft1d）と PLASMA（PLASMA_zgesv_incpiv）を適用し、OPL の方が SSLII より若干速いという結果が得られた。適用性は SSLII とほぼ変わらないが、初期化等の用法が全く同じではないので置き換えに伴うデバッグが難しいとか適当なマニュアルがないといった課題も見出された。また、連立一次等の利用者が自分で書いている部分を OPL に置き替えようと試みたが、複雑すぎて短時間では無理と判断した。このあたりにもライブラリを使う難しさの一端が垣間見られた。少ない事例から何ともいえない面もあるが、SSLII も相当に成熟しているライブラリであることを勘案すると、OPL を使うメリットはまずはその高度利用にあると思われる。次に、ライブラリが使われない（使わない）理由について実際に利用者へ無作為にヒアリングしてみたところ、多い順に、①移植性がなくなるから、②使うのが結構面倒だから、③呼び出しの時間的なオーバーヘッドが大きいから、④使い方・メリットがわからないから（「労多くして益少なし」の印象）、⑤余分なメモリを食うから、という回答が得られた。（ただし、これは「利用者がそう思っている」ということであって、実際にそうかどうかは別であることに注意する。）工学利用は納期が重要なので、様々なプラットフォームへの対応は重要であり、移植性がトップに来たのは意外ではない。面倒さとかオーバーヘッドについては、「オープンソースや市販コード、もっと単純なアルゴリズムを使います」という追加コメントが返って来そうであり、一理ある考え方である。また、流体解析では行列をきっちり解かなくても何とかなることもあり、ライブラリを使わなければならない理由・局面に乏しいことも利用が進まない原因の一つと考えられる。

このような JAXA におけるライブラリの利用状況は、航空宇宙や流体アプリ特有の事情が少なからず関与していることは確かだが、今後、システムが複雑化し、並列度が高まって行く中では、ライブラリ利用の重要性や必要性は増して行くのではないかと予想される。理由の一つは、ハイブリッド並列等の並列処理の複雑化、GPU 等によるシステム利用の複雑化に対して、利用者がこれら全てに自分でコードを書くという対応はどこかで破綻するのではないかと思われるからである。また、数値解析法の成熟とともに、従来の連立一次、固有値、フーリエ、乱数、補間といった処理に加えて、マルチグリッド、解適合（AMR）、クリロフ反復、領域分割といった複雑な処理が定番化していることも、これらのライブラリ化がうまく進めばライブラ

[1] FORTRAN77/90 ベースで書かれ、並列化は MPI または XPF による。最近は市販コードの利用が増えつつある

[2] 2012/1/4-2/23 間に JSS-M でコンパイルを行ったユーザを対象に、コンパイル時のログ情報を解析

リ利用を加速させる（もっと言えば必須となる）可能性がある。OPL においてこのような複雑処理のライブラリ化が行われることを期待したい。ただし、ライブラリだけ作ってどこかに置いておくだけではだめで、利用者とライブラリを近づける部分を担う人材の育成や、ライブラリの有効利用（特に高度利用）に関するノウハウの蓄積、啓蒙活動が同時に必要であろう。また、単にライブラリ利用ということではなく、もう少し広い意味でのフレームワークやミドルウェアといった考え方を取り入れた方が高度利用にはしっくり来るのではないかとも思われ、ちょっと大袈裟かもしれないが、旧来の数値解析プロセスの見直しや欧米のような分業を取り入れる時に来ているような気もするのだが如何。

SS研ペタスケールライブラリWG

JAXAにおける数値計算ライブラリの利用状況とOPL適用評価

松尾裕一

宇宙航空研究開発機構
研究開発本部数値解析グループ
2012年7月23日

報告内容

- JAXAスパコン（JSS）の構成
- JSS-Mにおけるライブラリの利用動向
- 実アプリにおける適用評価事例
- 分析、考察、今後の対応

JAXAスパコン（JSS）のシステム構成

計算エンジン部

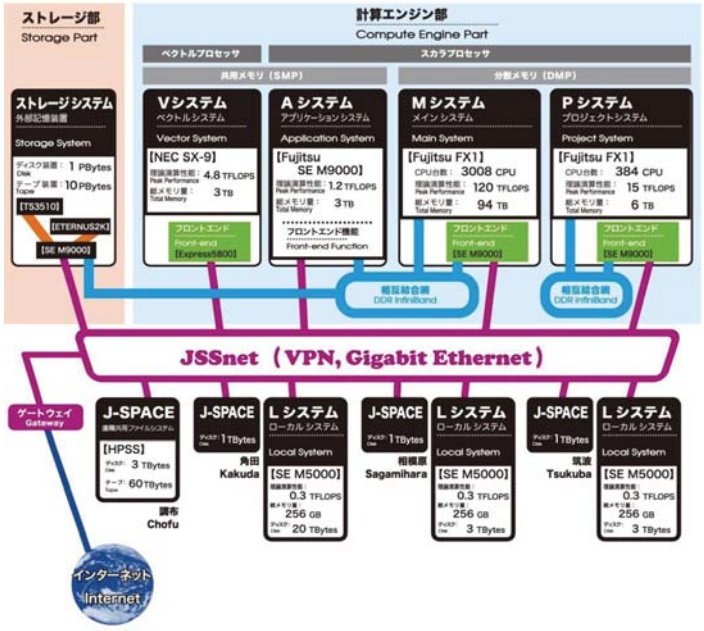
大規模並列計算機システム（M,P）
共有メモリ計算機システム（A,V）

ストレージ部

ストレージシステム

分散環境統合部

遠隔利用システム
分散データ共有システム
高速ネットワーク



計算エンジン部構成

システム名称	Mシステム	Pシステム	Aシステム	Vシステム
CPUタイプ	スカラー			ベクトル
システムタイプ	MPP		SMP	
ノード数	3,008	384	1	3
CPU数/ノード	1	1	32	16
コア数/CPU （全コア数）	4 （12,032）	4 （1,536）	4 （128）	1 （48）
ピーク性能[TFLOPS] （ノードあたり[GFLOPS]）	120 （40）	15 （40）	1.2 （1,200）	4.8 （1,600）
メモリ容量[TB] （ノードあたり [GB]）	94 （32）	6 （16）	1 （1,000）	3 （1,000）
製品名	富士通 FX1		富士通 SEM9000	NEC SX-9

JSS-Mにおけるライブラリの利用動向

- 調査
  - 対象範囲：2012/1/4-2012/2/23の間にMシステムでコンパイルを行った一般ユーザ（管理ユーザおよび実行だけの一般ユーザは対象外）
  - 集計方法：システムのコンパイル時ログ情報を解析し、条件に該当するユーザを人数単位でカウント（アプリの9割はCFD）
- 集計結果

条件	人数	割合
コンパイル利用者全体	348	100.0%
逐次利用	31	8.9%
並列利用	317	91.1%
プロセス並列のみ利用	10	2.9%
スレッド並列のみ利用	74	21.3%
ハイブリッド並列で利用	233	67.0%
SSLII, BLAS等を利用	192	55.2%

平均並列数：50～200

コンパイラが自動でBLASルーチンに変換するような場合も含む（例えばDAXPYなど上位5ルーチンを延べ50人が利用）

JSS-Mにおけるライブラリの利用動向（2）

- 集計結果（つづき）
  - SSLII関数毎の利用人数から上位10個を抽出した結果（ユーザが明示的に呼び出しているもの）

Rank	関数名	人数	割合	機能
1	SPLV(R8)	11	3.2%	3次spline補間式による補間・数値微分
2	RANU2(R4)	8	2.3%	一様乱数(0,1)の生成
3	LCX(C16)	5	1.4%	複素行列の連立一次方程式（クラウト法）
3	ALU(R8)	5	1.4%	実行列のLU分解
5	CEIG2(C16)	4	1.1%	複素行列の固有値固有ベクトル（QR法）
5	CFTM(C16)	4	1.1%	多次元離散型複素フーリエ変換（混合基底FFT）
5	EIG1(R8)	4	1.1%	実行列の固有値固有ベクトル（2段QR法）
5	LUIV(R8)	4	1.1%	LU分解された実行列の逆行列
5	SMLE2(R8)	4	1.1%	最小二乗近似多項式による平滑化（不等間隔離散点）
10	VCFM1(C16)	3	0.9%	1次元離散型複素フーリエ変換（2,3,5および7の混合基底）

同じ人かもしれない

JSS-Mにおけるライブラリの利用動向(3)

- 直接ライブラリを呼出しているユーザはあまり多くない。用途も限られ、高度利用は見られない。
  - 連立一次、固有値、フーリエ、乱数・近似・補間、等に限定
  - プロセス並列版の利用はなかった(スレッド版or逐次版のみ)
- ユーザのライブラリ利用に関する動向の分析、考察は後述
- ユーザが自分で書いている部分(連立一次、等)をライブラリに置き換えようとしたが、複雑すぎて無理と判断
  - 例:

$$\Delta Q^*_i = D_i^{-1}\left[R_i - \frac{1}{2}\sum_{j\in L(i)}\left[\left(F\left(Q_j + \Delta Q^*_j\right) - F\left(Q_j\right) - \rho_A \Delta \vec{Q}^*_j\right)S_{ij}\right]\right] \quad forward\ sweep$$

$$\Delta Q_i = \Delta Q^*_i - D_i^{-1}\frac{1}{2}\sum_{j\in U(i)}\left[\left(F\left(Q_j + \Delta Q_j\right) - F\left(Q_j\right) - \rho_A \Delta Q_j\right)S_{ij}\right] \quad backward\ sweep$$

- ライブラリを使用していて、運用側で実行環境を保有している2件の実アプリに関して、OPLの適用評価を実施

OPLの適用評価ケース

アプリ名	置換え対象SSLII関数		対応するオープンペタスケールライブラリ	
	名前	機能	名前	機能
CHANL	dcftm (SSL2基本)	多次元離散型複素フーリエ変換(混合基底)	zfft1d (FFTE)	1次元複素フーリエ変換(混合基底)
Crux	dlcx (SSL2基本)	複素行列の連立1次方程式(クラウト法)	PLASMA_ zgesv_incpiv (PLASMA)	複素行列の連立1次方程式

注) Crux中では固有値も呼んでいる(DCEIG2)が、OPL側の事情で今回の適用評価には間に合わず

CHANLの場合

- CHANLコード概要
  - 3D非圧縮性ナビエストークス方程式による直接シミュレーション
  - 時間積分：2次クランクニコルソン+3次ルンゲクッタ
  - 空間離散化：4時精度中心差分+2時精度中心差分
  - 圧力ポアソンソルバ：FFT
  - プロセス並列：XPFortran，スレッド並列：VISIMPACT+OMP
  - プログラム量：20,000行程度
- FFTにOPL（FFTE）を適用
- OPL FFTEの概要
  - V4.1（現在はV5.0）、Fortran77
  - OMP、OMP+MPI
  - 複素数/複素数、基底=2,3,5、次元=1,2,3、倍精度

CHANLの場合（2）

- SSLIIとFFTEの比較
  - 機能

	SSLII（DCFTM）	FFTE（zfft1d）
初期化	不要	必要
正規化	しない	する
バッファ	不要	必要
エラーコード	あり	なし

  - 引数

	SSLII（DCFTM）	FFTE（zfft1d）
第1引数	入力：配列の実部 出力：変換後配列の実部（上書き）	入力：複素配列 出力：変換後の複素配列（上書き）
第2引数	入力：配列の虚部 出力：変換後配列の虚部（上書き）	配列の要素数
第3引数	配列の要素数	0：初期化、-1；正変換、1：逆変換
第4引数	次元（本事例では1）	バッファ用複素配列 変換する配列の2倍の要素数を指定
第5引数	1：正変換、-1：逆変換	
第6引数	エラーコード	

CHANLの場合(3)

- 書き換え例(正変換)

適用前	適用後
!XOCL SPREAD DO /ISE DO 100 J = 1, JG ・・・ call gettod(ts1) !$omp do DO 30 I=1,IG DO 31 K=1,KG WAZ1(K)=WB(I,K) WAZ2(K)=0.0D0 31 CONTINUE C CALL DCFTM(WAZ1,WAZ2,NE,1,1,ICON) C DO 32 K=1,KG/2+1 WB1(I,K)=WAZ1(K) WB2(I,K)=WAZ2(K) 32 CONTINUE 30 CONTINUE !$omp end do call gettod(ts2) ・・・ !XOCL END SPREAD	!XOCL SPREAD DO /ISE DO 100 J = 1, JG ・・・ call gettod(ts1) !$omp do DO 30 I=1,IG DO 31 K=1,KG WAZ1(K)=WB(I,K) 31 CONTINUE C CALL zfft1d(WAZ1,NE(1),0,WAZ2) 初期化 CALL zfft1d(WAZ1,NE(1),-1,WAZ2) 1D_FFT C DO 32 K=1,KG/2+1 WB1(I,K)=dreal(WAZ1(K)) WB2(I,K)=dimag(WAZ1(K)) 32 CONTINUE 30 CONTINUE !$omp end do call gettod(ts2) ・・・ !XOCL END SPREAD

CHANLの場合(4)

- 書き換え例(逆変換)

適用前	適用後
!XOCL SPREAD DO /ISE DO 100 J = 1, JG ・・・ call gettod(ts1) !$omp do DO 300 I=1,IG DO 310 K=1,KG/2+1 WAZ1(K)=WB1(I,K) WAZ2(K)=WB2(I,K) 310 CONTINUE DO 311 K=2,KG/2 WAZ1(KG+2-K)=WB1(I,K) WAZ2(KG+2-K)=-WB2(I,K) 311 CONTINUE C CALL DCFTM(WAZ1,WAZ2,NE,1,-1,ICON) C DO 320 K=1,KG WB1(I,K)=WAZ1(K) 320 CONTINUE 300 CONTINUE !$omp end do call gettod(ts2) ・・・ !XOCL END SPREAD	!XOCL SPREAD DO /ISE DO 100 J = 1, JG ・・・ call gettod(ts1) !$omp do DO 300 I=1,IG DO 310 K=1,KG/2+1 WAZ1(K)=dcmplx(WB1(I,K),WB2(I,K)) 310 CONTINUE DO 320 K=2,KG/2 WAZ1(KG+2-K)=dcmplx(WB1(I,K),-WB2(I,K)) 310 CONTINUE C CALL zfft1d(WAZ1,NE(1),1,WAZ2) 1D_FFT C DO 320 K=1,KG WB(I,K)=NE(1)*dreal(WAZ1(K)) 320 CONTINUE 300 CONTINUE !$omp end do call gettod(ts2) ・・・ !XOCL END SPREAD

CHANLの場合(5)

- 実行条件
  - 計算規模　512(スレッド)×320(XPF)×256(FFT)
  - プロセス数　32
  - スレッド数　4
- 適用結果

サブルーチン	適用前	適用後
COSFFT2DF(正変換)	0.033057秒	0.028954秒
COSFFT2DR(逆変換)	0.031945秒	0.006723秒

  - 逆変換が速いのは，初期化が必要ないため
  - スレッド並列化はされている

Cruxの場合

- Cruxコードの概要
  - 超音速の円錐まわりの軸対象流、超音速の主翼上面の流れ
  - NS方程式を並行流と線形性の近似を施し、e^N法で解く
  - 固有値計算:コロケーション法
  - プロセス並列:なし，スレッド並列:VISIMPACT+OMP
  - プログラム量:1,000行程度
- 複素連立一次方程式にOPL(PLASMA)を適用
- OPL PLASMAの概要
  - V2.4.2(現在はV2.4.5)、Fortran77/C
  - pthread
  - 複素数/複素数、倍精度

Cruxの場合(2)

- PLASMAとSSLIIの比較
  - 機能

	SSLII(DLCX)	PLASMA(PLASMA_ZGESV_INCPIV)
ｱﾙｺﾞﾘｽﾞﾑ	クラウト法	LU分解
初期化	不要	必要
右辺	ベクトル(1次元)	配列(2次元)
バッファ	必要(問題規模に合わせた配列を指定)	必要(専用ライブラリで確保した配列を指定)

  - 引数

引数	SSLII(DLCX)	PLASMA(PLASMA_ZGESV_INCPIV)
第1引数	係数行列A(複素数)	係数行列Aの次数
第2引数	係数行列Aの整合寸法(≧N)	右辺の配列bの次数(≧0)
第3引数	係数行列Aの次数n	入力:係数行列A(複素数) 出力:LU分解結果
第4引数	入力:定数ベクトルb(複素数) 出力:解ベクトルx	係数行列Aの整合寸法(≧N)
第5引数	ピボットの相対零判定値(≧0.0)	出力:行列式を求めるための情報
第6引数	1:1組目の方程式を解く。 2:2組目以降の方程式を解く。	出力:ピボット情報
第7引数	出力:行列Aの行列式を求めるための情報	右辺の配列b
第8引数	作業領域(大きさnの複素数型1次元配列)	右辺の配列bの整合寸法
第9引数	作業領域(大きさnの一次元配列)	—
第10引数	エラーコード	—

Cruxの場合(3)

- 書き換え例

適用前

```
isw  = 1
do j=1,nn
 do i=1,nn
  ze(i)  = b(i,j)
 enddo !i
 call timer_sta(10)
 call dlcx(a,nn,nn,ze,epsz,isw,is,zvw,ivw,icon)
 call timer_end(10)
 do i=1,nn
  zc(i,j) = ze(i)
 enddo !i
 isw = 2
enddo !j
```

適用後

```
include "plasmaf.h"
 integer :: core,info
 parameter (core=1)
 integer*4 :: hl(2), hpiv(2)
 external PLASMA_init, PLASMA_ALLOC_WORKSPACE_ZGESV_INCPIV,
PLASMA_ZGESV_INCPIV
 external PLASMA_DEALLOC_HANDLE, PLASMA_FINALIZE

 call PLASMA_INIT(core,info)
 call PLASMA_ALLOC_WORKSPACE_ZGESV_INCPIV(nn,hl,hpiv,info)

 call timer_sta(10)
 call PLASMA_zgesv_incpiv(nn,nn,a,nn,hl,hpiv,b,nn,info)
 call timer_end(10)
 do j=1,nn
   do i=1,nn
     zc(i,j) = b(i,j)
   end do
 end do
・・・
 call PLASMA_DEALLOC_HANDLE(hl,info)
 call PLASMA_DEALLOC_HANDLE(hpiv,info)
 call PLASMA_finalize(info)
```

Cruxの場合(4)

- 実行条件
  - 計算規模　　118×61×84
  - プロセス数　　なし
  - スレッド数　　4
- 適用結果

適用前	適用後
184.07秒	104.40秒

  - 高速化は、呼び出しが一回に減ったため
  - 対象部分はスレッド並列化されていない
  - 条件により誤差発生
  - PLASMA内部でエラー検出した場合のデバッグが困難
  - 類似の関数が複数あり、どれが適切か判りにくい

ここまでのまとめ

- OPLを適用するにあたり、ユーザアプリにおけるライブラリの利用状況を調査
  - ライブラリがあまり使われていないことが判明(そんなもの?)
  - プロセス並列を含むような大域的なライブラリの利用はない
- 使われているものから、FFTと複素連立一次方程式につきOPLを適用評価
  - 適用性(使い勝手等)につき、大きな問題なし
    - 結果が違う場合の対応に困窮，マニュアルもない
  - 高速化はイマイチ(まあ、こんなもの?)
- 以下で、ライブラリが使われない理由や今後の対応につき考察

使われない理由について考えてみた

- メモリが少なかったベクトル時代からの名残で、少しでもメモリを効率良く使うために自分で書く癖（習慣）が残っている
- 未だに余分なメモリを食う（と思っている）ので使いたくない
  - ワーク配列がメモリを食う
- 呼び出しをする時間的オーバーヘッドが大きい（と思っている）
  - 初期化、関数の引数配列へのコピー、関数呼び出し、結果のコピーなど
- 使うのが結構面倒
  - マニュアル（日本語）がない、引数の意味の理解に時間がかかる
  - 使わなければならないほど複雑な処理（orホットスポット）がない
- 使い方・メリットを知らない（「労多くし益少なし」的印象）
  - 本当に速いのか？　本当に使いやすいのか？
- 使うと移植性がなくなる
- 開発側、センター側に使ってもらうための啓蒙活動が不足
- ブラックボックス的になるのを危惧、答えが正しいか不安

使われない理由について聞いてみた

- メモリが少なかったベクトル時代からの名残で、少しでもメモリを効率(0)良く使うために自分で書く癖（習慣）が残っている
- 未だに余分なメモリを食う（と思っている）ので使いたくない　(1)
  - ワーク配列がメモリを食う
  - 解きたい変数配列に対して直接 差分式を書き下して緩和演算をかけてしまっています。これだと前処理をかけがたいので、本来、収束が悪くなってどうか、ということもあるのですが、幸い前処理しなくとも、数十回程度で必要な精度まで収束するので、よい、ということにしています。
- 呼び出しをする時間的オーバーヘッドが大きい（と思っている）　(2)
  - 初期化、関数の引数配列へのコピー、関数呼び出し、結果のコピーなど
- 使うのが結構面倒　(3)
  - マニュアル（日本語）がない、引数の意味の理解に時間がかかる
  - 使わなければならないほど複雑な処理（orホットスポット）がない
  - 食わず嫌いだけかもしれません

使われない理由について聞いてみた(2)

- 使い方・メリットを知らない(「労多くし益少なし」的印象) (2)
  - 本当に速いのか？　本当に使いやすいのか？
- 使うと移植性がなくなる (4)
  - SSLIIの代わりにフリーのツールを使っている。SSLIIの日本語のマニュアルが理解不能で、一方、フリーのツールの英語のドキュメントは解説が丁寧で判りやすかった
  - 今の最大の理由は移植性です。Fortranの場合は手元のPCだとGnu か Intel で、Linux用富士通Fortran は32bit版は終わり、64bit版もv1のまま長らく新しくなってませんからね。昔、NWTができたころに、並列ライブラリがばっちり整備されていたら、できるだけそっちを使う、ということになっていたかも知れませんが、自分のソースプログラムを移植することで燃え尽きました。これからだと全部C++やJAVAで書くと実効性能が出ない分をライブラリで カバーできる、なんて風になるといいのでしょうね
  - 今は、パソコンでできる計算もあります。さらにPCクラスタ、センターマシンと複数のプラットホームを使うことはあたりまえの時代なので、移植性が保証されていないと使い難いと思います。代表的なプラットホームを網羅したライブラリもありますが、実は特定のプラットホーム向けにしかチューニングされていない、という場合も多いようです。
  - インターフェースは世の中に多く出回っているものと同じで、実装は計算機固有の機能を駆使して格段に速いものを作る、という方向で努力して頂ければと思います。

使われない理由について聞いてみた(3)

- その他 (2)
  - ライブラリで一番効果はあるのは、行列を解いたり、固有値を求めたりという部分ですが、流体計算は基本的に間接法なので、直接法のライブラリを そのまま使うのは少ないと思います。間接法はそこまで コーディングも難しくないですし。固有値も流体ではあまり使わないと思います。構造解析のFEMコードだったら、絶対ライブラリを使うと思います。
  - CFDの場合、線形代数に代表されるプリミティブなライブラリではなく、代表的な移流スキームや乱流モデルのライブラリがあれば、意外と需要があるような気がします。線形代数(等)のライブラリの場合、Navier-Stokes方程式が5行5列のブロック行列を要素とすることから、大規模な疎行列を扱うライブラリではなく、小規模な行列を一度に大量に効率よく扱えるライブラリの方が、需要があるような気がします。特に、メニイコアやGPGPUも視野に入れると、例えば大規模疎行列の逆行列を直接法により求めることは現実的ではなくても、小規模行列の逆行列を大量に求めることは意外と可能になりそうな気がします。(コア数nと計算すべき小行列の数Nが同程度のオーダーであれば問題なく計算可能です。今後は、コアあたりの計算粒度がどんどん小さくなると考えられるので、検討に値する時期になりつつあるのではないかと思います。)

分析、考察

- 使われない理由をまとめると
  - 移植性＞面倒＞オーバーヘッド＞メリット不明
  - ただし、若い人の反応は違うかもしれない
- 分析
  - 工学は納期が重要⇒様々なプラットフォームへの対応⇒移植性
  - オーバーヘッド大、面倒⇒一理ある
    - ライブラリを使う前にオープンソース、市販コード、単純なアルゴリズム化
  - 使わなければならない理由・場面に乏しい⇒分野特有?
    - マトリクスをきっちり解かなくても良い
  - メリットが不明確、説明不足
  - アルゴリズムよりB/F比の世界、ベクトル機の存在
  - CFD専用のライブラリ?

分析、考察(2)

- しかし、それで良いのか
  - 個人的には…ライブラリは重要、今後のHPCには必須
    - 並列処理(ｽﾚｯﾄﾞ、ﾌﾟﾛｾｽ、ﾊｲﾌﾞﾘｯﾄﾞ)、メモリ階層の複雑化
      ⇒ 性能を出すのが難しくなる
    - マルチグリッド、AMR、クリロフ反復といった(複雑)定番処理の増加
  - 航空宇宙は、以前はHPCの牽引役だったが今は?
    - 成功体験の罠、イノベーションが必要
    - 若手が育っていない
- 今後の対応とか
  - 使われない理由の調査・分析を継続
  - アプローチを変えてみる
    - 如何に大変か/大変でないかを実際にやってみる
    - 成功・失敗事例の蓄積、経験者の積
    - 啓蒙活動

（参　考）

JAXAアプリ（CFD）の類型化

Capability系	Capacity系	構造化プログラミング系	New Face
学術利用	工学利用		学術，工学特殊
格子点法			粒子法
ループ並列	領域分割並列		領域分割並列
単一領域	複数領域		複数領域
構造格子	構造/非構造/直交格子		
F77, F90	F77, F90	F90, C, C++ ポインタ，構造体利用	F90
規則的多量通信 （転置，AlltoAll, バリア）	不規則少量通信 （表面のみ，大域通信）	不規則少量通信 （表面のみ，大域通信）	ロードインバランス
XPF,（MPI）	MPI	MPI	MPI
		計算と通信の分離	
大並列	中～小並列	中～小並列	大並列
小頻度多量I/O	多頻度少量I/O	多頻度少量I/O	多頻度少量I/O
少数ジョブ	多数ジョブ	多数ジョブ	少数ジョブ

CFDの数値解法

- 基礎式：ナビエストークス

$$\frac{\partial Q}{\partial t} + \frac{\partial F_j}{\partial x_j} = 0$$

- 解法（典型）

$$\frac{\Delta Q}{\Delta t} + \left(\frac{\partial F_j}{\partial x_j}\right)^p = 0, \quad \Delta Q = Q^{n+1} - Q^n$$

$p = n$：陽解法，　$p=n+1$：陰解法

$$F_j^{n+1} = F_j^n + (\partial F_j / \partial Q)\Delta Q$$

$$\underbrace{\left[1 + \Delta t \, \frac{\partial}{\partial x_j}\left(\frac{\partial F_j}{\partial Q}\right)\right]}_{\text{連立一次（疎行列）}} \Delta Q = \underbrace{-\Delta t \, \left(\frac{\partial F_j}{\partial x_j}\right)^n}_{\text{MG、AMR}}$$

CFDの数値解法（2）

- 解法（つづき）
  - 非圧縮性、MHD⇒ポアソン⇒クリロフ反復、FFT、AMG
  - 安定性問題　⇒固有値
  - 粒子系　⇒乱数
- その他に使われているもの
  - METIS,Zoltan,SCOTCH　非構造格子用領域分割ツール
  - OpenFOAM　オープンソースCFDツール（C++）
  - Paraview ,Visit　可視化ツール
  - CHEMKIN　化学反応テーブル

4.7 並列擬似乱数発生アルゴリズムとペタスケール時代のモンテカルロ法の展望

並列擬似乱数発生アルゴリズムとペタスケール時代のモンテカルロ法の展望

発表者：国立情報学研究所　　特任教授　三浦謙一

ＨＰＣシステムのコア数は増大する傾向にあり、京コンピュータで６５Ｋコア、ローレンスリバモア研究所の Sequoia(IBM BlueGene/Q)ではすでに１　Ｍコアを超えている。このように並列度が高くなるとこれまでの数値解法アルゴリズムの中には十分にスケールせず性能が飽和あるいは低下するものもあり、数値計算アルゴリズムそのものの再評価が必要となってきている。そこでモンテカルロ法に期待が寄せられる。モンテカルロ法の利点としては

① Embarrasingly Parallel でコア間の通信量が少なく、スケーラビリティが良い。
② コアあたりの所要メモリー量が少ない。
③ 統計的な計算のため、ある程度の Fault Resilience が期待できる。

したがってミリオンコア時代のアルゴリズムとしてモンテカルロ法をこれまで以上にいろいろな応用分野に適用することを考えてみる価値がある。実際コンピュータの黎明期には線形代数、楕円型偏微分方程式の境界値問題（ディリクレ問題）等に対してモンテカルロ法の適用が研究されたこともあり、後者についてはその後も研究が続けられており、新しいアルゴリズムが提案されている。

モンテカルロ法による計算を数 10 万から 100 万規模のコア数に対して行うためには質の良い擬似乱数の発生アルゴリズムが不可欠である。具体的な要件としては以下が挙げられる。

① 周期が十分長い。
② 統計的な質が良く、TESTU01 のような検定プログラムをパスする。
③ 擬似乱数の発生が高速である。
④ 多数の CPU コアに対して擬似乱数の初期値を高速に計算し配布するアルゴリズムが存在する。（これが特に重要である。）

本報告者が提唱する、Multiple Recursive Algorithm(MRG)に基いた発生法である MRG8 はこれらの要件をすべて満足するものであり、今後の展開が期待される。
なお本アルゴリズムの実装（逐次版）は SS 研ユーティにも登録されている。

SS研ペタスケール数値計算ライブラリWG

並列擬似乱数発生アルゴリズムとペタスケール時代のモンテカルロ法の展望

2012年7月23日

国立情報学研究所
特任教授
三浦　謙一

モンテカルロ！

アウトライン

- スケーラブルアルゴリズムとしてのモンテカルロ法
- 乱数に求められる性質
- 並列化の容易性
- モンテカルロ法の利用分野の拡大
- Open Petascale Libraries Project

Million Core時代の並列アルゴリズム
～モンテカルロ法のメリット～

- 数多くの伝統的な数値計算アルゴリズムが、並列性の観点から見直しが必要。
- モンテカルロ法はCPUコア間の通信が少ない。
  →“embarrassingly parallel” で“ highly scalable”である。
- CPUコア当たりの所要メモリ量が少なくて済む。
- 統計的処理のため、ある程度”Fault Resilient” である。

コンピュータの初期（1950年代）にはいろいろ考えられたが、
その後あまり注目されていなかった分野での応用が期待される。

- 偏微分方程式（境界値問題）の解法
  - △u =0 (Laplace Equation)
  - △u =f (Poisson’s Equation)
  - △u + u=0

  } with u=g on the boundary (Dirichlet 境界条件.)

- 連立方程式の解法
- 固有値問題の解法

擬似乱数アルゴリズムとして必要な条件

- 周期が長い

  各コア内xCPU数の独立なシーケンス

- 良い統計的な性質（逐次と並列）

  TestU01のような検定プログラムが有用

- 高速に発生できる

  除算、分岐を含まない

- 超並列に適した初期値分配アルゴリズムを持つ

  Jump-aheadの容易性

Random Number Generation Algorithms

(1) Linear Congruential Method: $X_n = (a X_{n-1} + c) \bmod M$

Multiplicative： c=0. → Period $= 2^{j-2}$

Mixed： c .ne. 0. → Period $= 2^{j}$

where $M=2^{j}$（usually Machine Word Size)

(2) Binary M-sequence with Primitive Trinomial:

$X_n = (X_{n-m} .eor. X_{n-k}) \bmod 2$ → Period $= 2^{k}-1$ (m<k)

(3) Generalized Fibonacci Method with Primitive Trinomial:

$X_n = (X_{n-m}$ op. $X_{n-k}) \bmod M$ → Period $=(2^{k}-1)2^{j-1} \sim 2^{k+j-1}$

where op. is {+, -, *}.

(4) Generalized Recurrence Method With Large Prime Modulus

(Multiple Recursive Generator or MRG)

→ Period $=p^{k}-1 \sim 2^{j*k}$,where $p \sim 2^{j}$ is a Prime Number (e.g., $2^{31} - 1$)

Characteristics (and Advantages) of MRG with 8th-order Full Primitive Polynomials

From here on, consider $p=2^{31}-1$ and $k = 8$.

$$f(x) = (x^8 - a_1x^7 - a_2x^6 - a_3x^5 - a_4x^4 - a_5x^3 - a_6x^2 - a_7x - a_8) \ mod\ (p)$$

(1) Better chance of finding a generator which possesses good Lattice Structure with non-constrained full coefficients

$d_t \geq 1/\mathrm{sqrt}(1 + a_1^2 + a_2^2 + a_3^2 + a_4^2 + a_5^2 + a_6^2 + a_7^2 + a_8^2)$

(*Ref*. P.L'Ecuyer , INFORMS Journal on Computing, Vol.9, No.1,pp.57-60, 1997)

(2) Reasonably long period: $(2^{31}-1)^8 - 1 \sim 4.5*10^{74}$

(3) Ease of applying to vector/parallel processing, due to small dimension of the states

(4) Many primitive polynomials to choose from:

$\phi(p^k - 1)/k/(p^k - 1) = 2.2\%$

(5) Simple and straight-forward implementation is possible (and hopefully, good performance too!)

A Sample Polynomial with Good Lattice Structure (LatMRG)

Ref: Prof. Pierre L'Ecuyer, Univ. Montreal

$$x_n = a_1x_{n-1} + a_2x_{n-2} + a_3x_{n-3} + a_4x_{n-4} + a_5x_{n-5} + a_6x_{n-6} + a_7x_{n-7} + a_8x_{n-8} \ mod(p)$$

where

$a_1 = 1089656042$ $\quad a_5 = 189748160$

$a_2 = 1906537547$ $\quad a_6 = 1984088114$

$a_3 = 1764115693$ $\quad a_7 = 626062218$

$a_4 = 1304127872$ $\quad a_8 = 1927846343$

$p = 2^{31} - 1 = 2147483647$

Figure of Merit: M32=.62729

Out of 345597 tested polynomials with a_8 as primitive roots, 31843 are primitive polynomials (9.2%).

Other Reported Full Polynomials with Good Lattice Structure

3rd Order: $x_n = a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3} \mod p$, where $p = 2^{31} - 1 = 2147483647$

Grube-Dieter	Dieter	L'Ecuyer
$a_1 = 518175991$	$a_1 = 388425559$	$a_1 = 2021422057$
$a_2 = 510332243$	$a_2 = 227651891$	$a_2 = 1826992351$
$a_3 = 71324449$	$a_3 = 5412951$	$a_3 = 1977753457$

4th Order: $x_n = a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3} + a_4 x_{n-4} \mod p$, where $p = 2^{31} - 1 = 2147483647$

L'Ecuyer

$a_1 = 2001982722$
$a_2 = 1412284257$
$a_3 = 1155380217$
$a_4 = 1668339922$

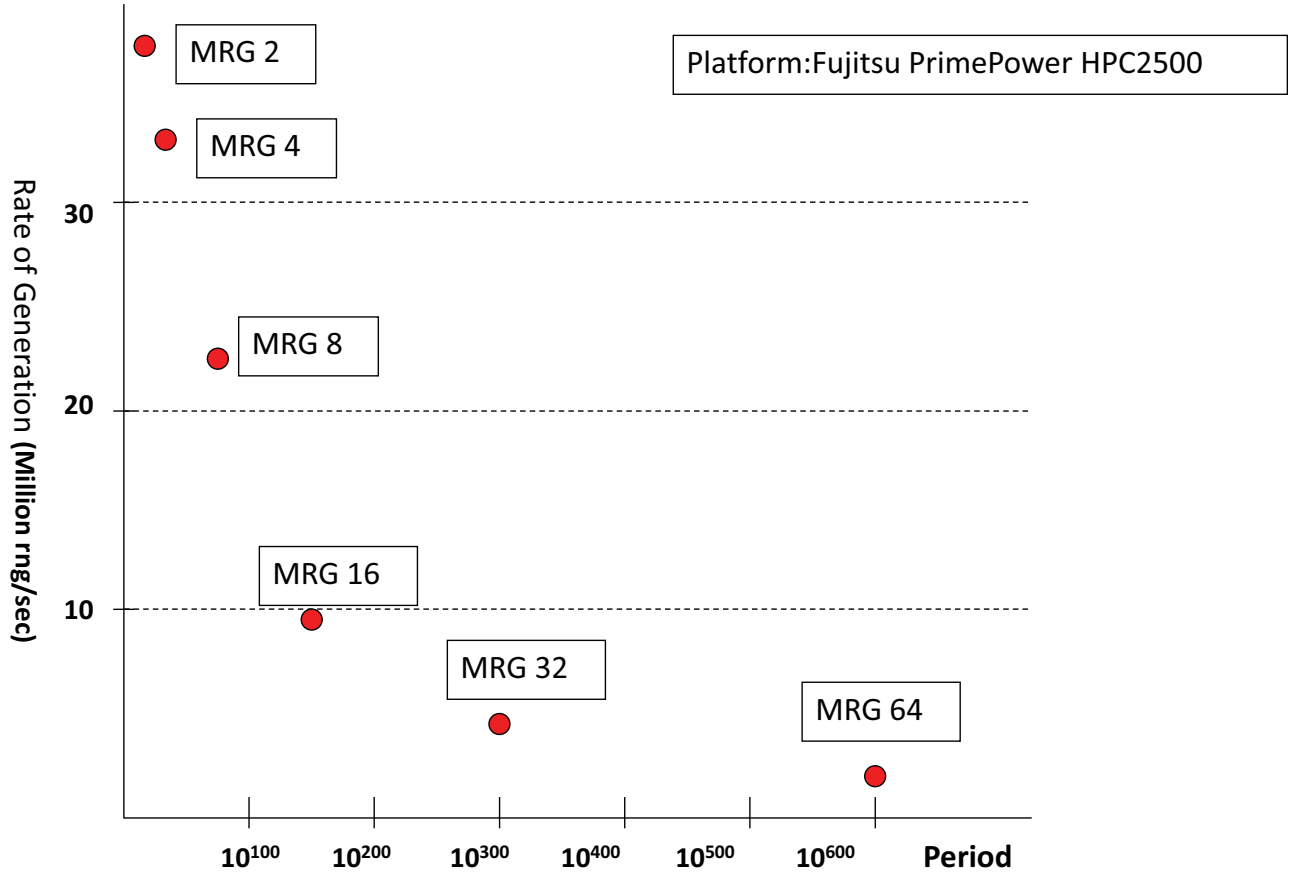
Ref. http://crypto.mat.sbg.ac.at/results/karl/server/node7.html

Performance Measurement of MRG8 on Machines with Different Architectures

As of 8/10/06
Rev. 06/14/11

System	Architecture	Clock (GHz)	Peak Performance (Gflops)	Rate of Generation (10^6 dp rng/sec)
Fujitsu VPP5000	Vector (Proprietary)	.3	9.6	201.8
NEC SX-6	Vector (Proprietary)	.5	8.0	224.3
Fujitsu PrimePower HPC2500	Scalar (Sparc64V)	1.3	5.2	23.2
IBM p-series 690	Scalar (Power4)	1.3	5.2	11.4
SGI Altix3700	Scalar (Itanium 2)	1.3	5.2	39.7
AMD Opteron	Scalar	2.0	4.0	34.0
Intel Woodcrest Xeon	Scalar (1 core)	2.66	10.6	108.1
Fujitsu Primergy RX200S2	Scalar (Xeon EM64T)	3.6	7.2	79.9
Fujitsu PrimeQuest480	Scalar (Itanium 2)	1.5	6.0	80.3
Fujitsu PrimeQuest RXI300	Scalar (Itanium2)	1.6	6.4	83.5
RIKEN/Fujitsu "K"	Scalar (SPARC64VIIIfx) (1 core)	2.0	16.0	34.5 (2 Integer Ops./clock)

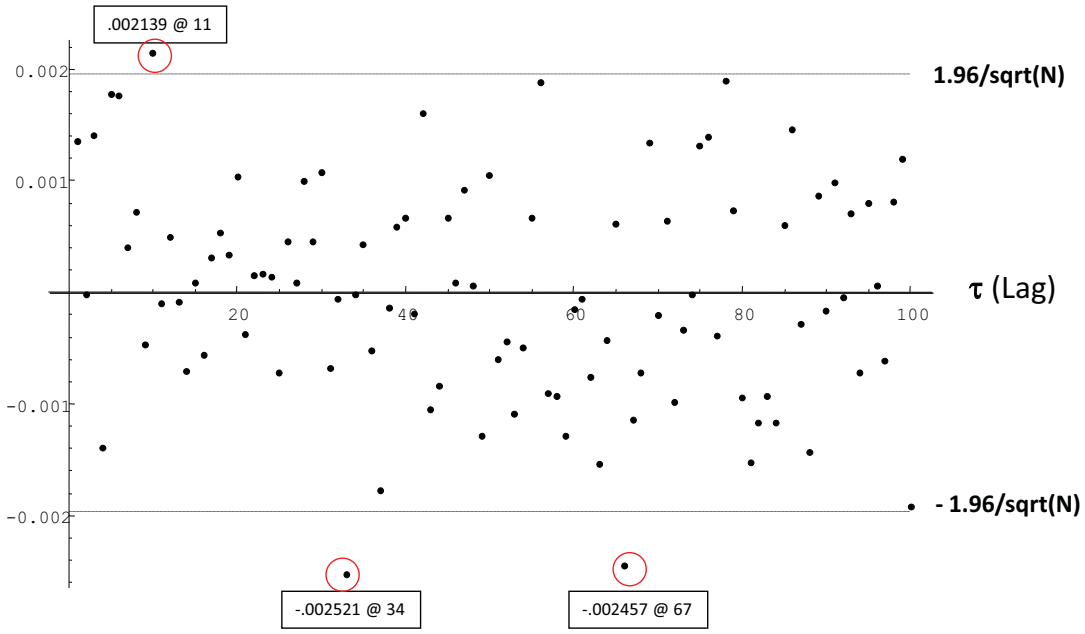
Order of Recurrence vs Rate of Generation



1D and 2D Uniformity Tests of MRG8

File Name (95% Chisq.)	n	Mean .50000	S.D. .28868	1D(100) (123.3)	1D(1000) (1073.6)	2D(10^2) (123.3)	2D(32^2) (1098.6)
RNG5a	10^6	.50007	.28871	83.82	1008.07	99.89	1026.83
RNG5b	10^6	.49977	.28858	96.65	1024.52	111.40	1033.65
RNG5c	10^6	.49953	.28883	114.35	1042.98	95.19	943.46
RNG51	10^6	.50015	.28869	83.99	963.66	106.71	1027.24
RNG55	10^6	.49988	.28851	79.81	1074.59	83.61	1032.41
RNG56	10^6	.49929	.28885	117.17	1013.80	101.32	1022.11
RNG57	10^6	.50033	.28865	95.40	1021.67	97.12	1001.07
RNG58	10^6	.50010	.28876	99.61	979.15	102.52	965.89
RNG59	10^6	.49989	.28842	84.101	999.03	119.95	1044.58

A Sample Autocorrelation Function of MRG8



N=10^6, File: RNG59

Parallelization（1）
~ Random Initialization～

Same as “Birthday Problem” (e.g., Feller)

- Period of Random Numbers : N
  Usage of Random Numbers in each process/thread : n
  Total number of chunks which should not overlap: N/n = m
  Total Number of Cores : p

- Probability of no collision = 1・(1-1/m)・(1-2/m)・・・・(1-(p-1)/m)
  ~= exp[-(p-1)p/(2m)] ~= 1 – (p-1)p/(2 m)

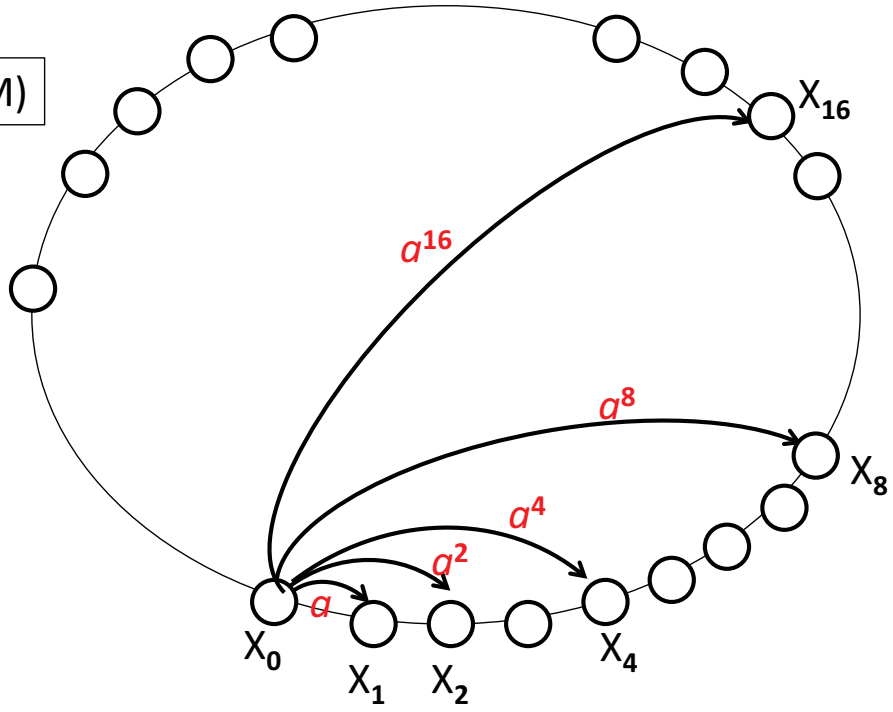
Probability that at least one collision occurs ~= （p-1)p/(2m)

Example : N=10^74, n=10^15, m=10^59, p=10^6 → **~ 5・10^(-48)**

Parallelization（2）

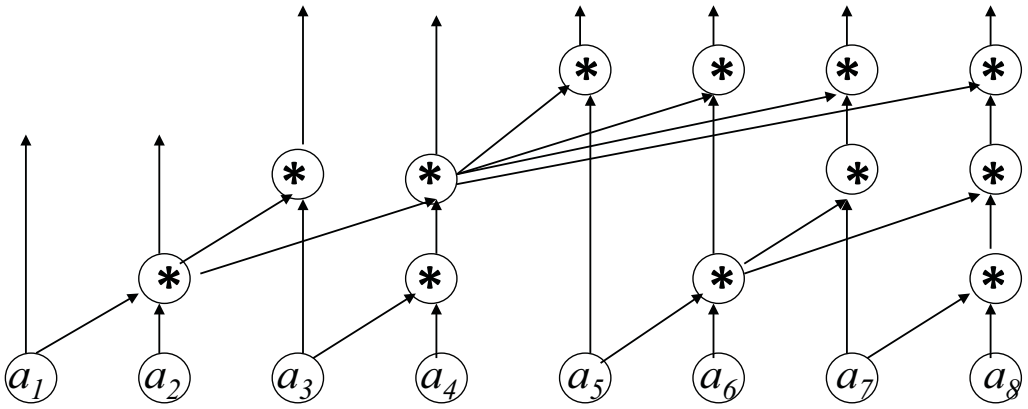
~ Jump-Ahead (First Order Recurrence) ~

$X_{n+1} = a\, X_n \mod(M)$



Parallelization of First Order Recurrence by Recursive Doubling

$x_i = a_i x_{i-1} \Rightarrow x_i = a_i\, a_{i-1} \ldots\ldots\ a_3 a_2 a_1 x_0$



Parallelization（2）
~ Jump-Ahead (the 8th Order Recurrence) ~

$$x_n = a_1x_{n-1} + a_2x_{n-2} + a_3x_{n-3} + a_4x_{n-4} + a_5x_{n-5} + a_6x_{n-6} + a_7x_{n-7} + a_8x_{n-8} \quad mod(p)$$

Define a Transfer Matrix A:

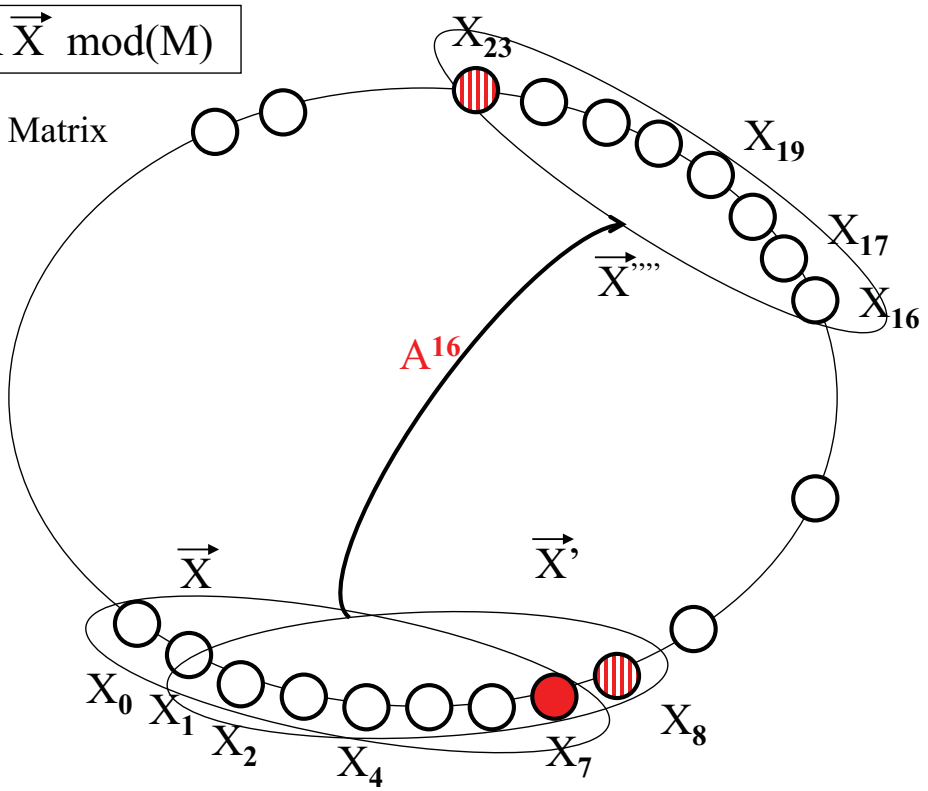
$$A = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_{n-1} \\ x_{n-2} \\ x_{n-3} \\ x_{n-4} \\ x_{n-5} \\ x_{n-6} \\ x_{n-7} \\ x_{n-8} \end{pmatrix}, \quad \boldsymbol{x'} = \begin{pmatrix} x_n \\ x_{n-1} \\ x_{n-2} \\ x_{n-3} \\ x_{n-4} \\ x_{n-5} \\ x_{n-6} \\ x_{n-7} \end{pmatrix}$$

Then $x' = A\,X \bmod(p)$.

Jumping Ahead the Sequence of MRG

$$\vec{X'} = A\,\vec{X} \bmod(M)$$

A: Transfer Matrix



Jump-Ahead for Arbitrary Distance

In order to compute $x_n = A^n x_0 \ mod(p)$ for an arbitrary n:

(1) Compute and store $A_j = A^{2^j} \ mod(p)$ (j=0,1,2,3,4,……).

(2) Represent "n" in the binary form,

e.g., $(b_{m-1},\ldots\ldots,b_2,b_1,b_0)$.

(3) Multiply $A_j \ mod(p)$'s together only when $b_j=1$ (j=0,….,m-1), to obtain A^n.

Note: The same strategy works with the polynomial formulation with fewer arithmetic operations (Knuth).

Random Number Testing on Quantum Diffusion Monte Carlo Application

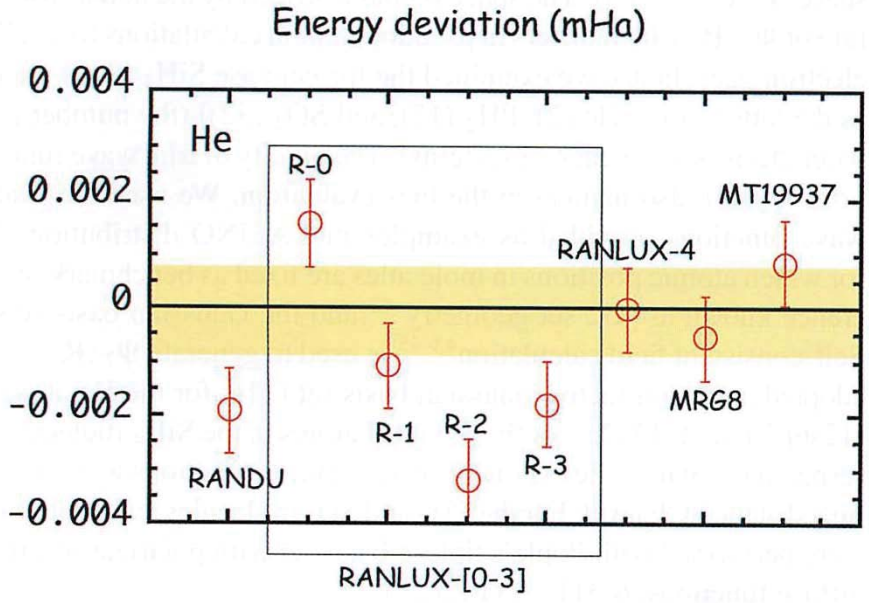


Figure 1. The ground state energies of He atom evaluated by several RNGs in VMC. Data is shown as the deviation from that by RANLUX-4.

Ref. K.Hongo, R.Maezono and K.Miura, J. Comp. Chem. **31** ,pp.2186-2194 2010

Architecture/Software Issues

- Language
  - Fortran
  - C

Parallel Programming Model
  - shared memory model (Open MP)
  - Distributed memory model (MPI)
  - Hybrid model
- Portability
- Multicore CPUs
- Fast Integer Arithmetic (64 bits mandatory)

モンテカルロ法の再認識

コンピュータの初期（1950年代）にはいろいろ考えられたが、その後あまり注目されていなかった分野での応用が期待される。例えば

-偏微分方程式（境界値問題）の解法

$\triangle u = 0$ (Laplace Equation)

$\triangle u = f$ (Poisson's Equation)

$\triangle u + u = 0$

with u=g on the boundary (Dirichlet B.C.)

-連立方程式の解法

-固有値問題の解法

例:ラプラス方程式の解法

電磁場、力場の計算に不可欠

$$\Delta\Psi = 0 \ \text{in}\ \Gamma$$
$$\Psi = g \ \text{on}\ \gamma$$

1. 差分法/有限要素法
2. 境界要素法
3. モンテカルロ法
   - Walk on Spheres Method
   - Walk on Rectangles Method
   - Walk on Boundary Method

解法の比較

- 有限要素法→すべての格子点での値を計算する必要あり
- 境界要素法→境界条件が変わるごとに境界方程式を求め直す必要あり
- モンテカルロ法→
  - 必要な格子点での計算だけで良い
  - 境界の値が時間的に変わっても即時求解が可能

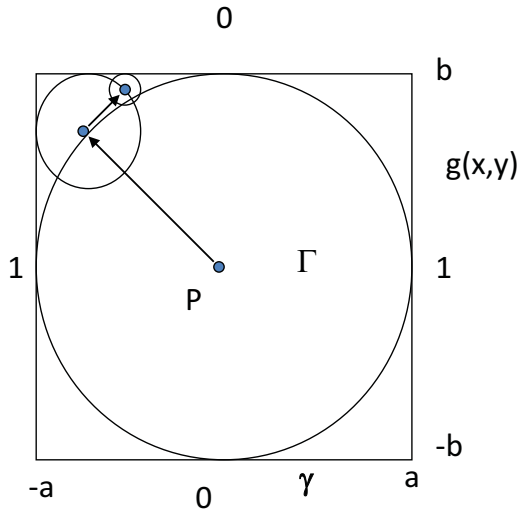
→ 電場内での荷電粒子シミュレーションのように、
粒子が存在する場所の値のみが必要な場合に有効か。

例:ラプラス方程式の解法

各格子点でのランダムウォークを行う必要は無い。
→Walk on Spheres (WOS) Method

$$\Delta\Psi = 0 \ \text{in}\ \Gamma$$
$$\Psi = g \ \text{on}\ \gamma$$

①一番近くの境界に接するような
円(3次元なら球)を描く。
②円周上でランダムに点をとる。
③その点を中心に更に円を描く。
④点が境界からε以内になったら
境界点でのg(x,y)を累算する。



計算例

N=10,000

X=0 y=0 ε = 0.0001
No. of walks = 10000 Sum of hits = 5098
Solution at (x,y) =0.5098
Last radius of sphere = 0.000013661
Generated random numbers = 121292

N=40,000

X=0 y=0 ε = 0.0001
No. of walks = 40000 Sum of hits = 20106
Solution at (x,y) =0.50265
Last radius of sphere = 0.0000106012
Generated random numbers = 484677

N=1,000,000

X=0 y=0 ε = 0.0001
No. of walks = 1000000 Sum of hits = 499734
Solution at (x,y) =0.499734
Last radius of sphere = 1.71 x 10^-6
Generated random numbers = 12083215

注:厳密解は0.50000である。

WOSの考察と課題

- $= 10^{-4}$の場合円が境界に到達するまで平均12回のWalkが必要 （理論上は log( ) に比例）

- 他の計算法との定量的な比較、特に境界要素法との精度 vs 計算時間の比較が必要

- 他のPDEへの拡張

まとめ

- マルチペタスケール時代に入り、コア数もO(10^6)に突入 → エクサではどうなる？
- いくつかの計算アルゴリズムの見直しが必要
- モンテカルロ法はますます有望に
  → 適用分野の拡大（例：PDE)
- そのためには良い並列（擬似）乱数が必要
  → MRGの提案
- 既存のライブラリにある乱数ルーティンは要注意
- Open Petascale Libraries（OPL) Projectでも乱数は重要な位置付けである

Acknowledgement

The author would like to thank

- Prof. P.L'Ecuyer : Sample Primitive Polynomials
- Dr. C.Chen at Fujitsu Computer System
  :Measurement on VPP5000, Primergy, PrimeQuest
- Dr. M.Kurokawa, RIKEN
  :Measurement on SX-6, Woodcrest
- Mr. Hayakawa, AMD Japan
  :Measurement on AMD Opteron
- Mr. Hotta, Fujitsu Limited (Makuhari)
  :Measurement on Sparc64VIIIfx

FFTE について

筑波大学システム情報系（計算科学研究センター）
高橋大介

高速フーリエ変換（Fast Fourier Transform，以下 FFT）は，科学技術計算において今日広く用いられているアルゴリズムである．

本発表では FFT を計算するためのライブラリである FFTE ライブラリの概要および二次元分割を用いた並列三次元 FFT アルゴリズムについて述べている．

FFTE は 2002 年 5 月にバージョン 1.0 を公開し，最新のバージョンは 2011 年 11 月に公開したバージョン 5.0 である．シングルプロセッサ，共有メモリ型並列（OpenMP），分散メモリ型並列（MPI），ハイブリッドプログラミングモデル（OpenMP+MPI）をそれぞれサポートしており，http://www.ffte.jp/よりソースコードを入手可能である．

FFTE ライブラリの設計方針としては，キャッシュブロッキングを行うことでキャッシュミスをできるだけ少なくすると共に，ノード内およびノード間のスケーラビリティの確保を目指している．

2012 年 6 月の Top500 において，2 システムが 10PFlops の大台を突破しているが，今後出現すると予想される 100PFlops 級マシンでは，ほぼ確実に 100 万コアを超える規模のものになると予想される．この場合，MPI と OpenMP のハイブリッド実行を行ったとしても，MPI プロセス数が 10 万個以上になる可能性がある．

これまで，並列三次元 FFT においては三次元（x，y，z 方向）のうちの一次元（例えば z 方向）のみを分割して配列を格納することが多かったが，MPI プロセスが 1 万個の場合，z 方向のデータ数が 1 万個以上でなければならず，三次元 FFT の問題サイズに制約があるという問題があった．この問題を解決するために，FFTE の並列三次元 FFT では一次元分割に加えて y，z 方向の二次元で分割する，二次元分割もサポートしている．

二次元分割では一次元分割に比べて，全体の通信量は 2 倍となるが，MPI プロセス数が大きく，かつ全対全通信のレイテンシが大きい場合には，二次元分割の方が一次元分割に比べて通信時間が短くなる場合がある．

T2K 筑波システムの 4096 コア（256 ノード）までを用いた性能評価の結果，並列三次元 FFT において，二次元分割により通信時間を削減することで，MPI プロセス数が多い場合に一次元分割に比べて性能が改善されることを示した．

二次元分割を用いた並列三次元 FFT は，MPI プロセス数が 1 万個を超えるようなペタスケール環境において，より効果的であると考えられる．

FFTEについて

高橋大介
筑波大学システム情報系
計算科学研究センター

発表内容

- FFTEライブラリの概要
- 二次元分割を用いた並列三次元FFTアルゴリズム

FFTEライブラリの概要

- FFTEは高速フーリエ変換(Fast Fourier Transform, 以下FFT)を計算するためのライブラリである.
- 2002年5月にバージョン1.0を公開し, 現在のバージョンは2011年11月に公開したバージョン5.0.
- 逐次, 共有メモリ(OpenMP), 分散メモリ(MPI), ハイブリッド(OpenMP+MPI)をそれぞれサポート.
  - 二次元および三次元の実数FFT
  - 一次元, 二次元および三次元の複素数FFT
  - それぞれ, 2, 3, 5のべき乗からなるデータ点数に対応している.
  - 三次元の並列複素数FFTでは二次元分割にも対応.
- http://www.ffte.jp/ よりソースコードを入手可能.

FFTEライブラリの設計方針

- 性能
  - キャッシュミスをできるだけ少なくする.
    - 明示的なキャッシュブロッキングを行っている.
  - スケーラビリティの確保(ノード内, ノード間)
- API
  - 引数を少なくして使いやすさを重視.
  - SGI SCSLやAMD ACMLに類似している.
- ポータビリティ
  - ノード間通信:MPI
  - 演算部分:Fortran + OpenMP

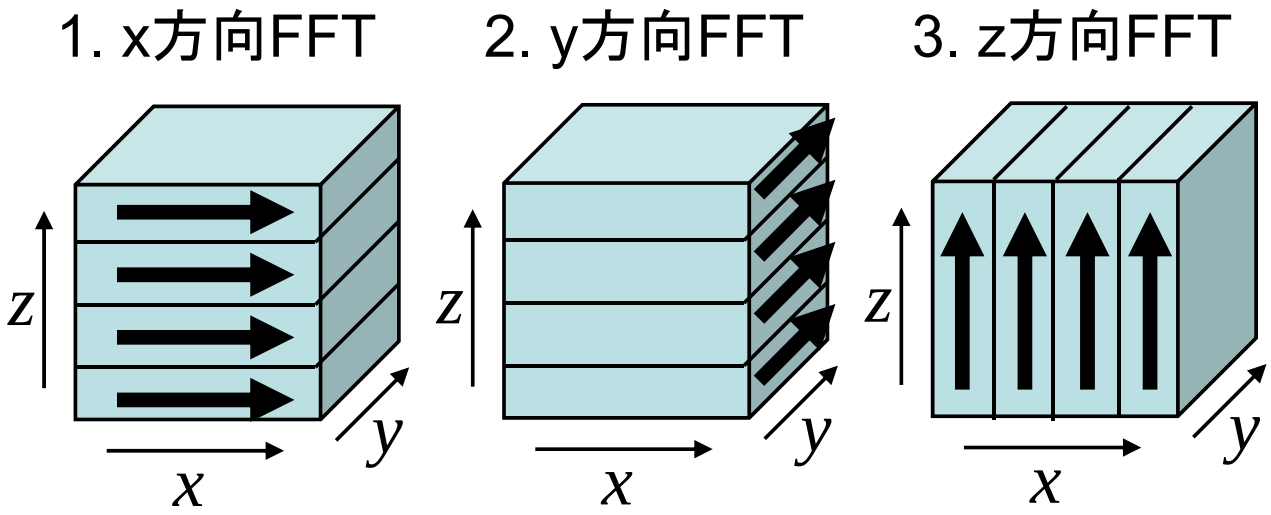
背景

- 2012年6月のTop500において，2システムが10PFlopsの大台を突破している.
  - Sequoia: 20.133 PFlops (1,572,864 Cores)
  - K computer: 10.510 PFlops (705,024 Cores)
- 今後出現すると予想される，100PFlops級マシンは，ほぼ確実に100万コアを超える規模のものになると予想される.
  - MPIとOpenMPのハイブリッド実行を行ったとしても，MPIプロセス数が10万個以上になる可能性がある.

目的

- 並列三次元FFTにおける典型的な配列の分散方法
  - 三次元（x，y，z方向）のうちの一次元のみ（例えばz方向）のみを分割して配列を格納.
  - MPIプロセスが1万個の場合，z方向のデータ数が1万点以上でなければならず，三次元FFTの問題サイズに制約.
- x，y，z方向に三次元分割する方法が提案されている[Eleftheriou et al. ’05, Fang et al. ’07].
  - 各方向のFFTを行う都度，全対全通信が必要.
- 二次元分割を行うことで全対全通信の回数を減らしつつ，比較的少ないデータ数でも高いスケーラビリティを得ることを目的とする.

z方向に一次元ブロック分割した場合の並列三次元FFT



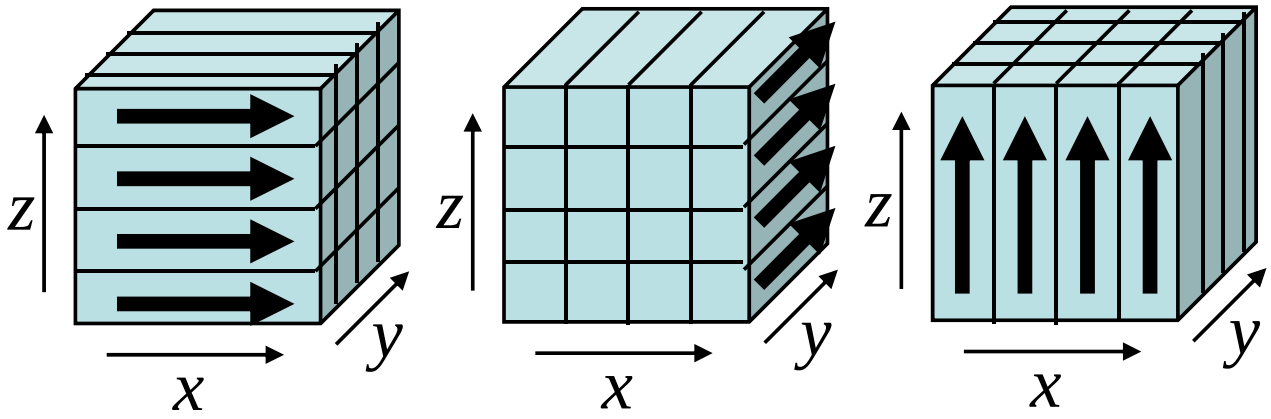
各プロセッサでslab形状に分割

三次元FFTの超並列化

- 並列アプリケーションプログラムのいくつかにおいては，三次元FFTが律速になっている.
- x，y，zのうちz方向のみに一次元分割した場合，超並列化は不可能.
  - 1,024×1,024×1,024点FFTを2,048プロセスで分割できない（1,024プロセスまでは分割可能）
- y，zの二次元分割で対応する.
  - 1,024×1,024×1,024点FFTが1,048,576（=1,024×1,024）プロセスまで分割可能になる.

y, z方向に二次元ブロック分割した場合の並列三次元FFT

1. x方向FFT　2. y方向FFT　3. z方向FFT



各プロセッサで直方体形状に分割

二次元分割による並列三次元FFT

- 二次元分割した場合, $P \times Q$ 個のプロセッサにおいて,
  - P 個のプロセッサ間で全対全通信を Q 組
  - Q 個のプロセッサ間で全対全通信を P 組

  行う必要がある.
- MPI_Comm_Split()を用いてMPI_COMM_WORLDをy方向（P プロセッサ）とz方向（Q プロセッサ）でコミュニケータを分割した.
  - 各コミュニケータ内でMPI_Alltoall()を行う.
- 入力データがy, z方向に, 出力データはx, y方向に二次元ブロック分割されている.
  - 全対全通信はy方向で1回, z方向で1回の合計2回で済む.

一次元分割の場合の通信時間

- 全データ数を N, プロセッサ数を $P \times Q$, プロセッサ間通信性能を W(Byte/s), 通信レイテンシを L (sec)とする.
- 各プロセッサは $N/(PQ)^2$個の倍精度複素数データを自分以外の $PQ-1$ 個のプロセッサに送ることになる.
- 一次元分割の場合の通信時間は

$$T_{1\mathrm{dim}} = (PQ-1)\left(L + \frac{16N}{(PQ)^2 \cdot W}\right)$$

$$\approx PQ \cdot L + \frac{16N}{PQ \cdot W} \quad (\mathrm{sec})$$

二次元分割の場合の通信時間

- y方向の P 個のプロセッサ間で全対全通信を Q 組行う.
  - y方向の各プロセッサは $N/(P^2Q)$ 個の倍精度複素数データを, y方向の $P-1$ 個のプロセッサに送る.
- z方向の Q 個のプロセッサ間で全対全通信を P 組行う.
  - z方向の各プロセッサは $N/(PQ^2)$個の倍精度複素数データを, z方向の $Q-1$ 個のプロセッサに送る.
- 二次元分割の場合の通信時間は

$$T_{2\mathrm{dim}} = (P-1)\left(L + \frac{16N}{P^2Q \cdot W}\right) + (Q-1)\left(L + \frac{16N}{PQ^2 \cdot W}\right)$$

$$\approx (P+Q) \cdot L + \frac{32N}{PQ \cdot W} \quad (\mathrm{sec})$$

一次元分割と二次元分割の場合の通信時間の比較(1/2)

- 一次元分割の通信時間

$$T_{1\mathrm{dim}} \approx PQ \cdot L + \frac{16N}{PQ \cdot W}$$

- 二次元分割の通信時間

$$T_{2\mathrm{dim}} \approx (P+Q) \cdot L + \frac{32N}{PQ \cdot W}$$

- 二つの式を比較すると，全プロセッサ数 $P \times Q$ が大きく，かつレイテンシ L が大きい場合には，二次元分割の方が通信時間が短くなることが分かる.

一次元分割と二次元分割の場合の通信時間の比較(2/2)

- 二次元分割の通信時間が一次元分割の通信時間よりも少なくなる条件を求める.

$$(P+Q) \cdot L + \frac{32N}{PQ \cdot W} < PQ \cdot L + \frac{16N}{PQ \cdot W}$$

を解くと，

$$N < \frac{(LW \cdot PQ)(PQ - P - Q)}{16}$$

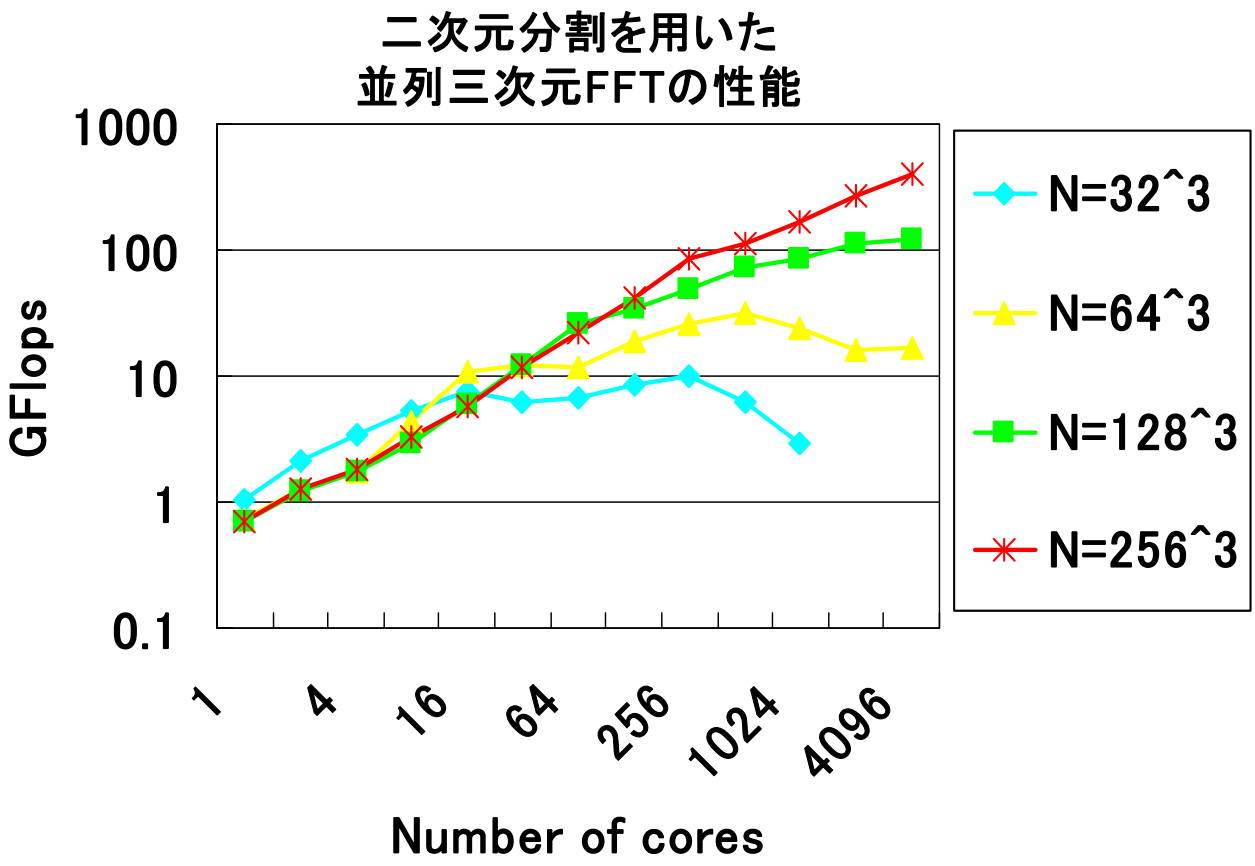
- 例えば，$L = 10^{-5}$ (sec)，$W = 10^{9}$ (Byte/s)，$P = Q = 64$ を上の式に代入すると，$N < 10^{10}$ の範囲では二次元分割の通信時間が一次元分割に比べて少なくなる.

性能評価

- 性能評価にあたっては，二次元分割を行った並列三次元FFTと，一次元分割を行った並列三次元FFTの性能比較を行った.
- Strong Scalingとして$N = 32^3, 64^3, 128^3, 256^3$点の順方向FFTを1～4,096MPIプロセスで連続10回実行し，その平均の経過時間を測定した.
- 評価環境
  - T2K筑波システムの256ノード（4,096コア）を使用
  - flat MPI（1core当たり1MPIプロセス）
  - MPIライブラリ：MVAPICH 1.2.0
  - Intel Fortran Compiler 10.1
  - コンパイルオプション：”ifort –O3 –xO”（SSE3ベクトル命令）

T2K筑波システムの概要

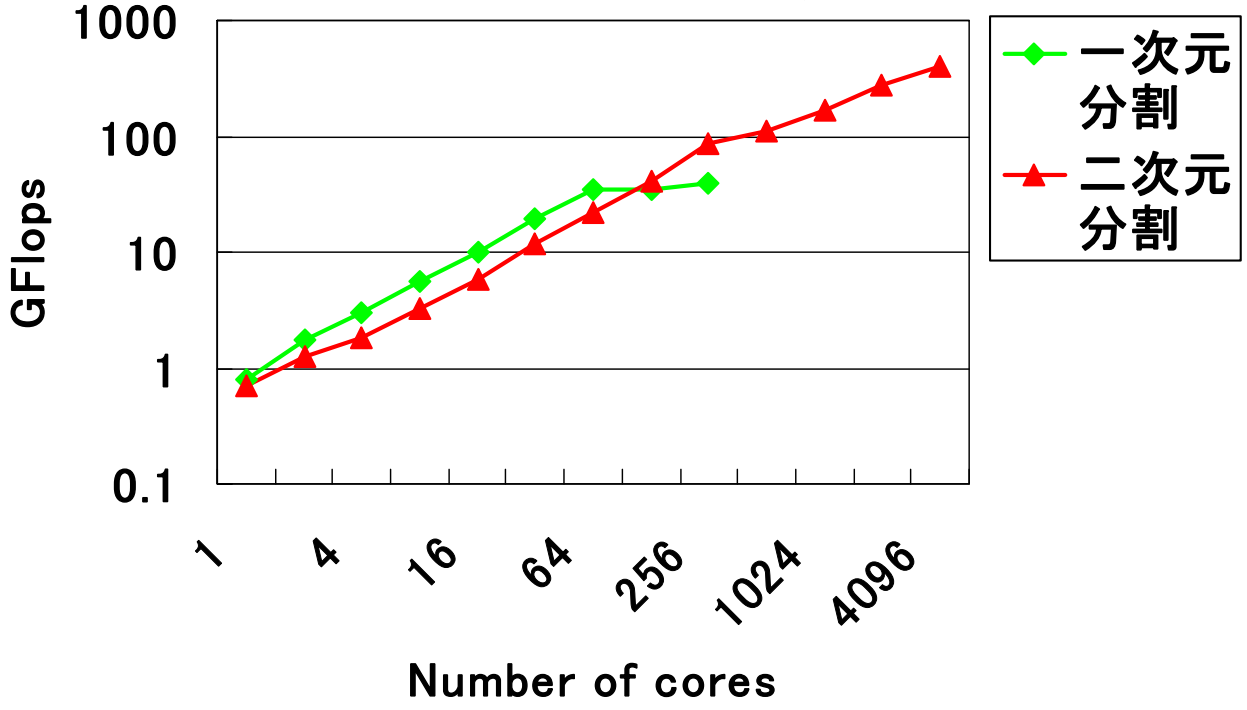
- T2K筑波システムの諸元
  - ノード台数：648（Appro Xtreme-X3 Server）
  - ノード構成：4ソケット／ノード
  - CPU：Quad-Core AMD Opteron 8356（Barcelona 2.3GHz）
  - 総メモリ容量：20TB
  - ネットワークインターフェース：DDR InfiniBand Mellanox ConnectX HCA×4
  - ネットワークトポロジ：Fat Tree
  - Full-bisectionバンド幅：5.18TB/s



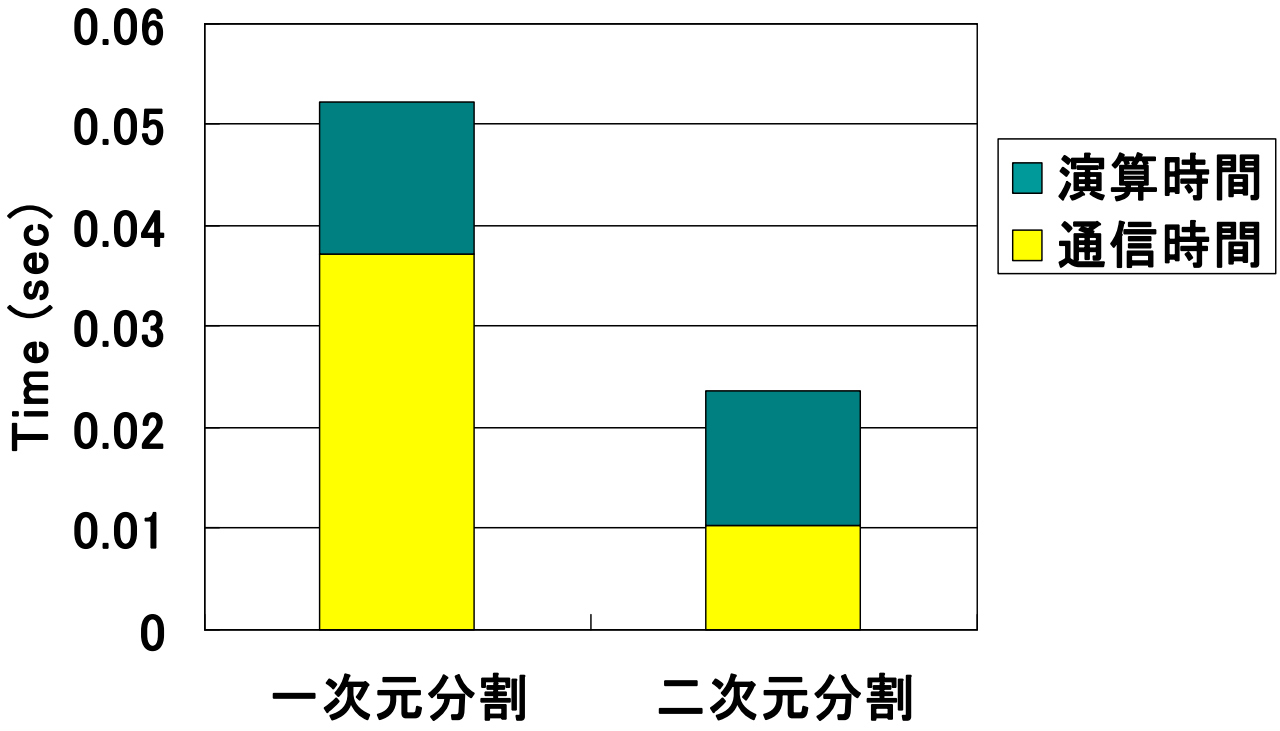
考察(1/2)

- $N = 32^3$ 点FFTでは良好なスケーラビリティが得られていない.
- これは問題サイズが小さい(データサイズ:1MB)ことから, 全対全通信が全実行時間のほとんどを占めているからであると考えられる.
- それに対して, $N = 256^3$ 点FFT(データサイズ:512MB)では4,096コアまで性能が向上していることが分かる.
  - 4,096コアにおける性能は約401.3 GFlops (理論ピーク性能の約1.1%)
  - 全対全通信を除いたカーネル部分の性能は約10.07 TFlops (理論ピーク性能の約26.7%)

256^3点FFTにおける一次元分割と二次元分割の性能比較



並列三次元FFTの実行時間の内訳 (256cores, 256^3点FFT)



考察(2/2)

- 64コア以下の場合には，通信量の少ない一次元分割が二次元分割よりも性能が高くなっている.
- 128コア以上では通信時間を少なくできる二次元分割が一次元分割よりも性能が高くなっていることが分かる.
- 二次元分割を行った場合でも，4,096コアにおいては96%以上が通信時間に費やされている.
  - 全対全通信において各プロセッサが一度に送る通信量がわずか1KBとなるため，通信時間においてレイテンシが支配的になるためであると考えられる.
- 全対全通信にMPI_Alltoall関数を使わずに，より低レベルな通信関数を用いて，レイテンシを削減する工夫が必要.

二次元分割の場合のFFTの限界性能(1/2)

- 以下のようなシステムを仮定し，$N = 1024^3$ 点FFTを計算
  - プロセッサ数: $P \times Q = 256 \times 256 = 65{,}536$
  - プロセッサ当たりの理論ピーク演算性能:「∞」 Flops
  - プロセッサ間通信性能: $W =$「∞」 Byte/s
  - 通信レイテンシ: $L = 1\mu\text{sec}$
- N 点FFTの演算量は $5N \log_2 N$ とする.
- N 点のデータ(倍精度複素数)の通信量: $16N$ バイト
- 全対全通信に要する時間は，

$$T_{2\text{dim}} \approx (P+Q)\cdot L + \frac{32N}{PQ\cdot W} = 5.12\times 10^{-4}\ \text{sec}$$

- $N = 1024^3$ 点FFTの限界性能は，

$$(5N \log_2 N)/T_{2\text{dim}} \approx 314.57\ \text{TFlops}$$

二次元分割の場合のFFTの限界性能(2/2)

- 全対全通信にshuffle-exchangeアルゴリズムを使った場合の通信時間は,

$$T_{2\dim} \approx (\log_2 P)\left(L + \frac{16N}{P^2 Q \cdot W}\right) + (\log_2 Q)\left(L + \frac{16N}{PQ^2 \cdot W}\right)$$

$$= 1.6 \times 10^{-5} \text{ sec}$$

- $N = 1024^3$ 点FFTの限界性能は,

$$(5N \log_2 N) / T_{2\dim} \approx 10.07 \text{ PFlops}$$

まとめ

- FFTEライブラリの概要および二次元分割を用いた並列三次元FFTアルゴリズムについて述べた.
- 並列三次元FFTにおいて, 二次元分割により通信時間を削減することで, MPIプロセス数が多い場合に性能が改善されることを示した.
- 二次元分割を用いた並列三次元FFTは, プロセッサ数が1万個を超えるようなペタスケール環境において, より効果的であると考えられる.

4.9 フラグメント分子軌道（FMO）法と並列 FMO プログラム OpenFMO について

フラグメント分子軌道（FMO）法と並列 FMO プログラム OpenFMO について

九州大学情報基盤研究開発センター　稲富　雄一

分子軌道（Molecular Orbital, MO）法は、原子や分子の中にある電子の振る舞い（分布）を求めるための量子力学に基づいた計算手法である。MO 法では、分子の安定構造や化学反応における反応エネルギーや活性化エネルギーなどの有用な情報が得られるため、理論化学の研究者だけでなく、実験化学分野の研究者にも幅広く利用されている。しかしながら、経験的なパラメタを一切用いない非経験的な MO 法では、最も計算量が小さい Hartree-Fock（HF）法と呼ばれる近似計算手法を用いた場合でも、$O(N^4)$の計算量であり、MO 法の応用が期待される分野の一つであるタンパクや DNA などの巨大生体分子に対する計算を行うことができない。

北浦和夫現神戸大学特命教授によって開発されたフラグメント分子軌道（FMO）法は、従来困難であった大規模分子に対する MO 計算を行うために開発された計算手法である。FMO 法では、計算対象となる巨大な分子を 20～40 原子程度の小さなフラグメントに分割して、各フラグメント（モノマー）、および、フラグメントペア（ダイマー）に対する小規模な MO 計算を行うことで、巨大分子全体の状態を近似する。複数のモノマーやダイマーに対する MO 計算を並列に処理できること、さらに、各小規模 MO 計算自身も並列処理が可能であることなどから、大規模並列処理向きの計算手法である。そこで我々は、京コンピュータをはじめとした超並列計算機を用いて、効率よく並列 FMO 計算を行うための最適化を並列 FMO プログラム OpenFMO に対して行った。OpenFMO は九州大学と九州先端科学技術研究所でスクラッチから開発された C 言語で記述されている並列 FMO プログラムであり、HF 法に基づいた FMO 計算に特化しているためソースコードが 5 万行程度と小規模であるため、最適化が行い易い。

我々がこれまでに行った主な最適化は、MPI/OpenMP ハイブリッド並列化、グローバルカウンタを用いた動的負荷分散による負荷均等化、および、FMO 計算で用いる中間データの保存方法の改良、の３つである。これらの改良を行った OpenFMO プログラムを用いて２万並列までのベンチマークを行った結果、非常に高い並列性能を示すことが分かった。また、２万並列時に１万６千原子を超える分子に対する FMO 計算が 30 分弱で終わることが分かった。このことから、より大規模な並列計算を行うことで、大規模分子に対する電子状態計算を 10 分未満で行うことが近い将来に可能であり、創薬等の分野での応用が、より簡単になる可能性が見えてきた。

一方で、小規模 MO 計算部分においては、99%以上の計算量である分子積分計算は、並列化が容易なために、並列処理により計算時間が大幅に短縮されたが、その代わりに、一般化固有値問題求値（対角化）の時間がノード間並列化が困難な部分として、並列化効率を低下させる要因になっていることが分かった。今後は、対角化などのノード間並列が困難な部分の処理時間を隠蔽するなどの工夫をすることで、更なる並列処理速度の改善をする必要がある。

フラグメント分子軌道（FMO）法と並列FMOプログラムOpenFMOについて

九州大学情報基盤研究開発センター
稲富　雄一

2013年2月22日

九州大学

目次

1. 目的
2. 量子化学計算
3. フラグメント分子軌道（FMO）法
4. OpenFMOの最適化と性能評価
   FMO計算における行列計算
5. まとめ

目標

大規模生体分子に対する第一原理（量子力学）に基づいた電子状態計算を高速に行うことができるようにしたい！

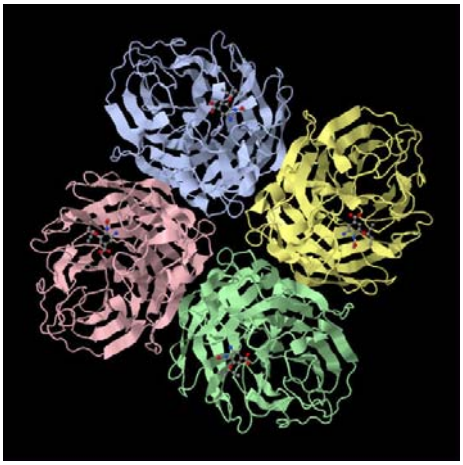
大規模分子軌道計算の応用が期待される分野

➢創薬

- 薬の候補分子と生体分子（タンパク質、核酸、糖鎖、ウイルスなど）との結び付きやすさが分かるため、生理活性の推定や候補物質の絞り込み（スクリーニング）に使える

右図：H5N1型鳥インフルエンザウイルスのノイラミニダーゼ（PDB ID=2HU4, 23,856原子）

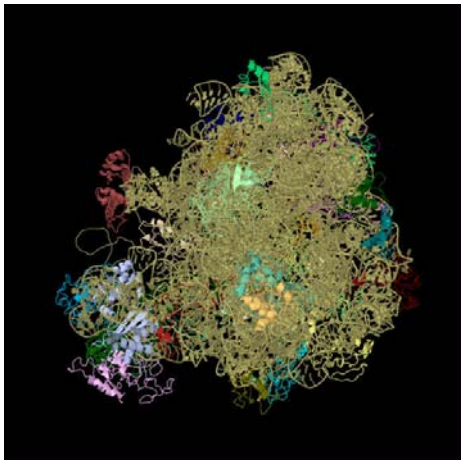
感染した細胞からウイルスを分離させるために必要。このたんぱくの働きが阻害されると、細胞内で合成されたウイルスが他の細胞に移動できなくなるので、増殖できない（抗インフルエンザ薬のタミフル、リレンザはノイラミニダーゼ阻害剤）



計算機シミュレーションによるスクリーニングを行うためには、多数回の計算を短時間で行う必要がある！

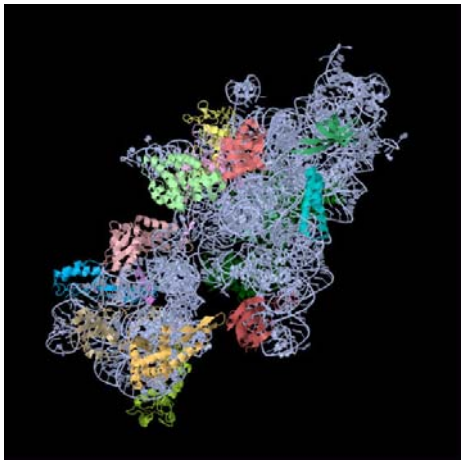
ついでにもうひとつ（リボゾーム）

60Sユニット（PDB ID=1S1I）



タンパク質合成（ペプチド結合形成）を担う

40Sユニット（PDB ID=1S1H）



m-RNAの暗号解読を担う

➢細菌とヒトではリボゾームの構造が違うので、細菌のリボゾームの機能を選択的に阻害する薬剤（リボゾーム阻害剤）は細菌性感染症の薬になる
（ストレプトマイシン、テトラサイクリンなどの抗生物質はリボゾーム阻害剤）

➢タンパクの折りたたみの機構の解明につながる（かも）

主に、Hartree-Fock（HF）法について

量子化学計算（≒分子軌道法）

量子化学計算とは？

量子力学に基づいた第一原理計算

分子中の電子に対するSchrödinger方程式を解いて、分子内での電子の運動（分布）を求める

$$\hat{H}_{\mathrm{elec}}\Psi = E_{\mathrm{elec}}\Psi$$

$$\hat{H}_{\mathrm{elec}} = \sum_{i}^{n}\left(-\frac{1}{2}\nabla_i^2\right) + \sum_{i}^{n}\sum_{A}^{N}\left(-\frac{Z_A}{|\mathbf{r}_i - \mathbf{R}_A|}\right) + \sum_{i}^{n}\sum_{j<i}^{n}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

n = 電子数

N = 原子数

Z_A = 原子Aの原子番号

得られた波動関数 Ψ が電子状態を表している

一般には3粒子以上が関係するので、近似的に解く必要がある

分子軌道（Molecular Orbital, MO）法とは？

- 電子のSchrödinger方程式の近似解法の1つ
- 電子が分子軌道（MO）と呼ばれる軌道（のようなもの＝1電子波動関数）を運動している、という化学者に分かりやすい描像を与える
- 実験値を利用する（半）経験的方法から、実験値を用いない非経験的な方法、近似の粗い方法や高精度な方法など、たくさんの方法が存在

量子化学計算≒分子軌道計算

分子軌道法のなかま（これでも、まだ一部・・・）

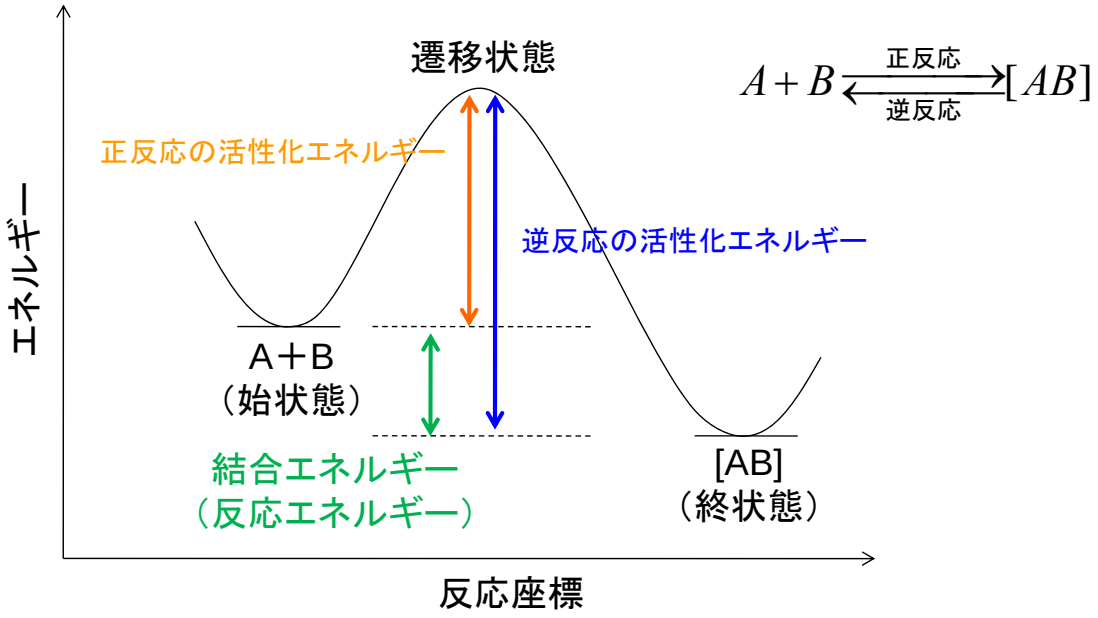
- 半経験的方法
  - （拡張）Huckel法、PPP法、CNDO法、PM5、AM3, DFTB, … etc.
- 非経験的（ab initio）分子軌道法
  - Hartree-Fock（HF）法
    - RHF法、UHF法、ROHF法
  - Post Hartree-Fock法
    - （Møller-Plesset,MP）法（摂動法）
      - MP2、MP3、MP4 …
    - 配置間相互作用（Configuration Interaction, CI）法
      - CID、CISD、CISDT、full CI …
    - クラスター展開（Coupled Cluster, CC）法
      - CCSD, CCSD(T), CCSDT、…
    - 多配置参照（Multi Reference, MR）法
      - MCSCF、MR-MP、MR-CI、MR-CC …

高精度（計算量大）

非経験的分子軌道法は、ほぼすべて、HF法の解を0次近似として用いている

MO法で分かること（1）分子のエネルギー

化学反応の起こりやすさ、結合の強さ、吸収波長などが分かる

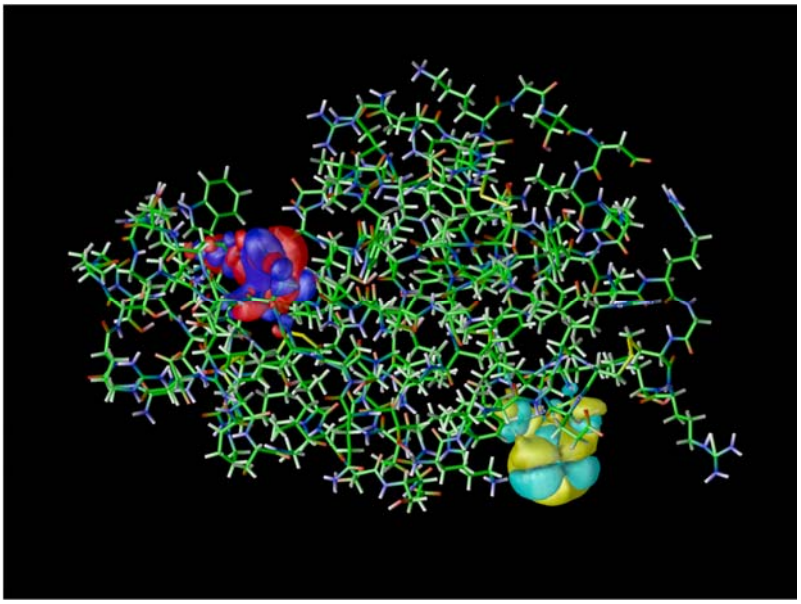


MO法で分かること（2）分子軌道

化学反応が起こる場所（活性部位）が分かる

Lysozyme分子のフロンティア軌道 (1,961原子、10,969関数)

赤-青＝Highest Occupied Molecular Orbital (**HOMO**)
黄色-水色＝Lowest Unoccupied Molecular Orbital (**LUMO**)



Hartree-Fock法で解くべき基本方程式

Schrödinger方程式からHartree-Fock-Roothaan方程式まで

$$\hat{H}_{\text{elec}}\Psi = E_{\text{elec}}\Psi$$

$$\hat{H}_{\text{elec}} = -\frac{1}{2}\sum_{i}^{n}\nabla_i^2 - \sum_{i}^{n}\sum_{A}^{N}\frac{Z_A}{|\mathbf{r}_i - \mathbf{R}_A|} + \sum_{i}^{n}\sum_{j<i}^{n}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

エネルギー期待値を最小化する

$$E_{\text{elec}} = \frac{\langle \Psi | \hat{H}_{\text{elec}} | \Psi \rangle}{\langle \Psi | \Psi \rangle}$$

制限付きスピン軌道

$$\chi_{2k}(\mathbf{x}) = \psi_k(\mathbf{r})\alpha(\sigma)$$
$$\chi_{2k+1}(\mathbf{x}) = \psi_k(\mathbf{r})\beta(\sigma)$$
$$k = 1, 2, \ldots$$

Hartree-Fock近似

$$\Psi \approx \tilde{\Psi}^{\text{HF}} = \frac{1}{\sqrt{n!}}\begin{vmatrix} \chi_1(\mathbf{x}_1) & \chi_2(\mathbf{x}_1) & \cdots & \chi_n(\mathbf{x}_1) \\ \chi_1(\mathbf{x}_2) & \chi_2(\mathbf{x}_2) & \cdots & \chi_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(\mathbf{x}_n) & \chi_2(\mathbf{x}_n) & \cdots & \chi_n(\mathbf{x}_n) \end{vmatrix}$$

基底関数展開 $\psi_i(\mathbf{r}) = \sum_{\mu=1}^{N_{\text{AO}}} c_\mu^i \varphi_\mu(\mathbf{r}; \mathbf{n}_\mu, \mathbf{R}_\mu)$

Hartree-Fock-Roothaan方程式

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$$

Hartree-Fock-Roothaan（HFR）方程式

Schrödinger方程式に各種近似を適用して得られる代数方程式

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$$

$\mathbf{F}$ = Fock行列（対称行列）

$\mathbf{S}$ = 重なり行列（正定値対称行列）

$\mathbf{C}$ = MO係数行列（固有ベクトル）$\equiv \{\mathbf{c}^1, \mathbf{c}^2, \ldots, \mathbf{c}^{N_{\text{AO}}}\}$

$\boldsymbol{\varepsilon}$ = MOエネルギー（固有値）$\equiv \text{diag}\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{N_{\text{AO}}}\}$

エネルギー期待値を極小化する（スピン）軌道を求める問題が、分子軌道を基底関数で展開する線形結合係数（MO係数）を求める問題に置き換えられている

$$\psi_i(\mathbf{r}) = \sum_{\mu=1}^{N_{\text{AO}}} c_\mu^i \varphi_\mu(\mathbf{r}; \mathbf{n}_\mu, \mathbf{R}_\mu)$$

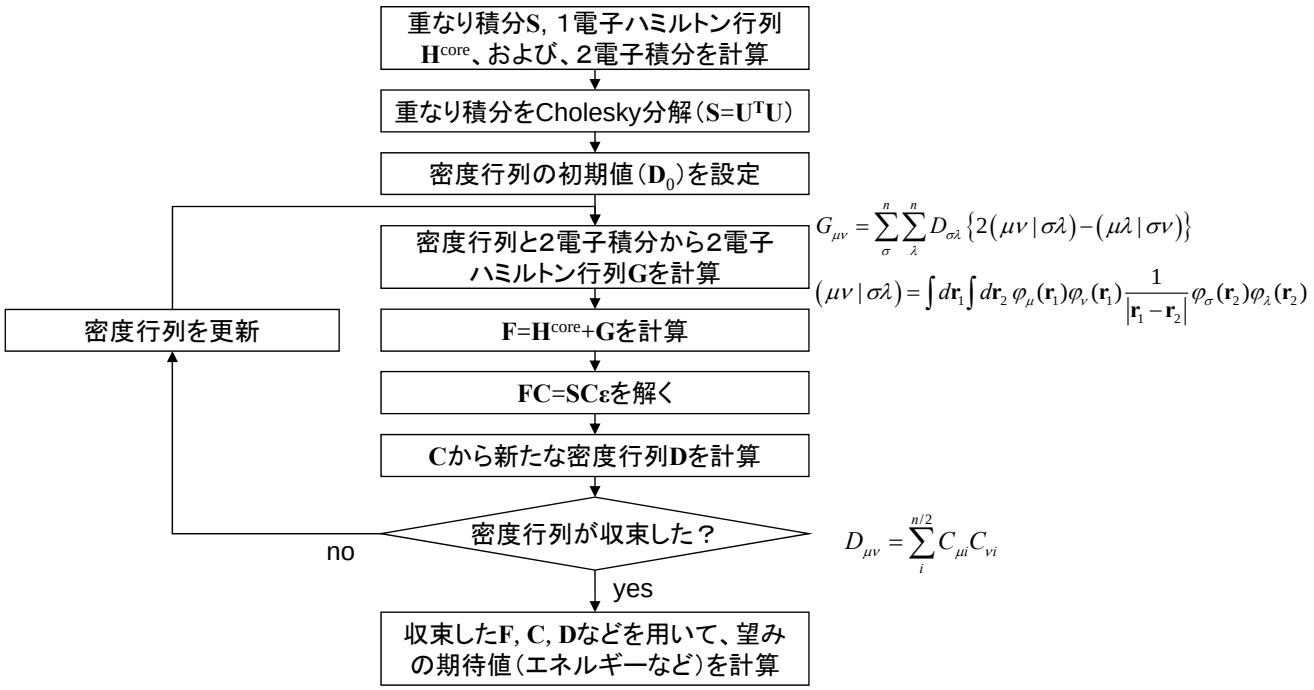
HFR方程式の特徴

$$\mathbf{FC} = \mathbf{SC}\varepsilon$$

Fock行列	$\mathbf{F} = \mathbf{H}^{\text{core}} + \mathbf{G}$
1電子ハミルトン行列	$\mathbf{H}^{\text{core}} = \mathbf{T} + \mathbf{V}$
運動エネルギー積分	$T_{\mu\nu} = \int d\mathbf{r}_1 \varphi_\mu(\mathbf{r}_1)\left(-\frac{1}{2}\nabla_1^2\right)\varphi_\nu(\mathbf{r}_1)$
核引力積分	$V_{\mu\nu} = \int d\mathbf{r}_1 \varphi_\mu(\mathbf{r}_1)\left(\sum_A^N \frac{-Z_A}{\lvert\mathbf{r}_1 - \mathbf{R}_A\rvert}\right)\varphi_\nu(\mathbf{r}_1)$
2電子ハミルトン行列	$G_{\mu\nu} = \sum_\sigma^n \sum_\lambda^n D_{\sigma\lambda}\{2(\mu\nu\mid\sigma\lambda) - (\mu\lambda\mid\sigma\nu)\}$
2電子積分	$(\mu\nu\mid\sigma\lambda) = \int d\mathbf{r}_1 \int d\mathbf{r}_2 \varphi_\mu(\mathbf{r}_1)\varphi_\nu(\mathbf{r}_1)\frac{1}{\lvert\mathbf{r}_1 - \mathbf{r}_2\rvert}\varphi_\sigma(\mathbf{r}_2)\varphi_\lambda(\mathbf{r}_2)$
密度行列	$D_{\mu\nu} = \sum_i^{n/2} C_{\mu i} C_{\nu i}$

- Fock行列**F**と重なり行列**S**を入力とした一般化固有値問題である
- 与えるべきFock行列**F**の計算には、この方程式の解であるMO係数行列**C**が必要であるため、この方程式は非線型方程式であり、反復解法（Self Consistent Field, SCF法）によって解かなければならない
- 計算量は$O(N^4)$（2電子積分の計算量に依存）

HFR方程式を解く手順



そのままでは大規模計算ができない・・・

計算量が$O(N^4)$かそれ以上である

- コンピュータが1000倍速くなっても、入力データの大きさは約5.6倍（=1000の4乗根）にしかできない

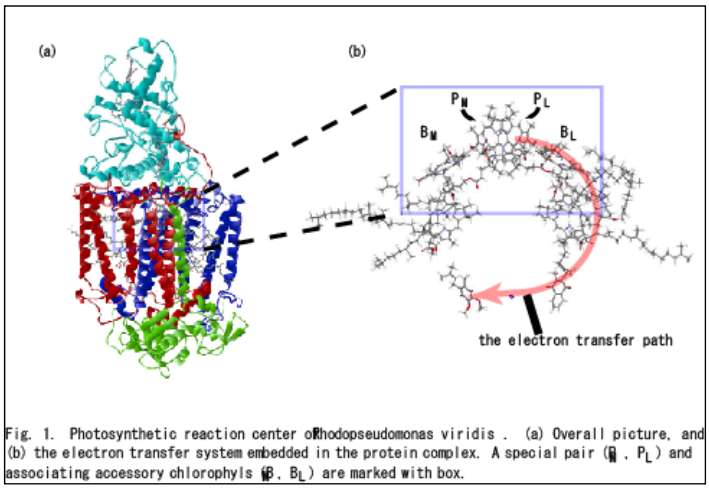


Fig. 1. Photosynthetic reaction center of Rhodopseudomonas viridis . (a) Overall picture, and (b) the electron transfer system embedded in the protein complex. A special pair (P_M, P_L) and associating accessory chlorophyls (B_M, B_L) are marked with box.

参考：SC|05, T. Ikegami et al., technical paper

20,581原子
77,754電子
164,442関数
1,398フラグメント

従来分子軌道法(Hartree-Fock法）で計算した場合の予想計算時間＝およそ20万年！

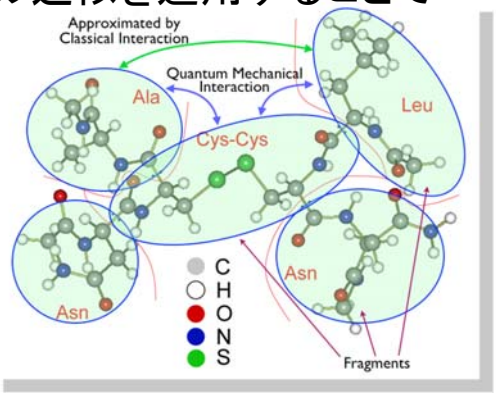
FMOでの計算時間＝72.5時間（Opteron (model 246), 600プロセッサ使用時）

大規模分子に対する電子状態計算を行うための計算手法

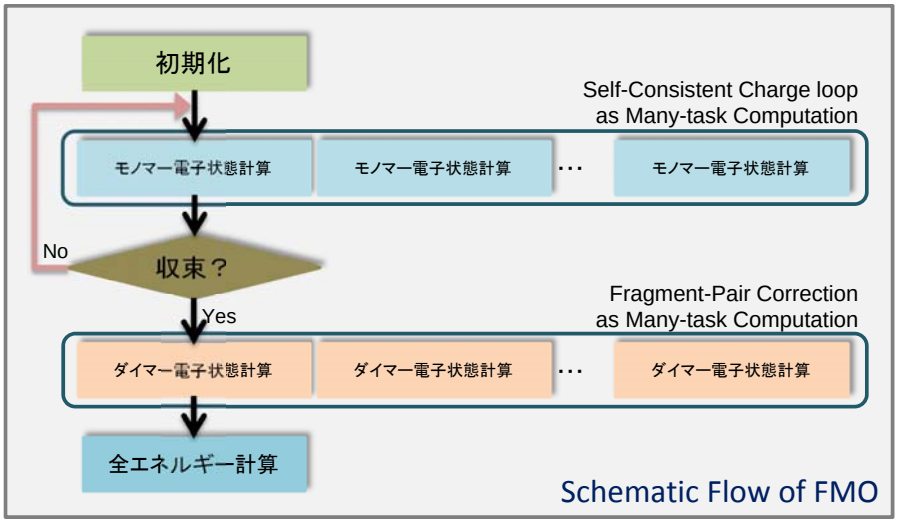
フラグメント分子軌道（FMO）法

フラグメント分子軌道（FMO）法とは？

- 大規模生体分子に対するMO計算を行うために北浦和夫神戸大学特命教授により開発された並列処理向きの計算手法
- 分子を20～40原子程度のフラグメントに分割して、各フラグメント（モノマー）とフラグメントペア（ダイマー）に対する電子状態で、分子全体の電子状態を近似
- フラグメント間ではクーロン相互作用のみを考慮
- フラグメント間距離に応じて、いくつかの近似を適用することで計算時間を大幅に削減

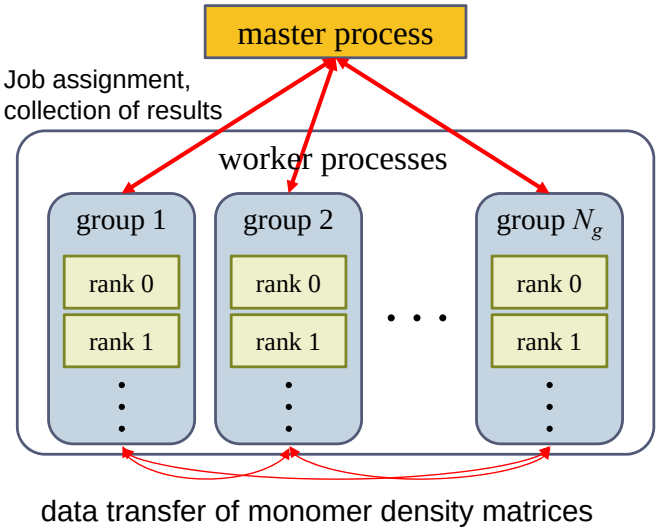


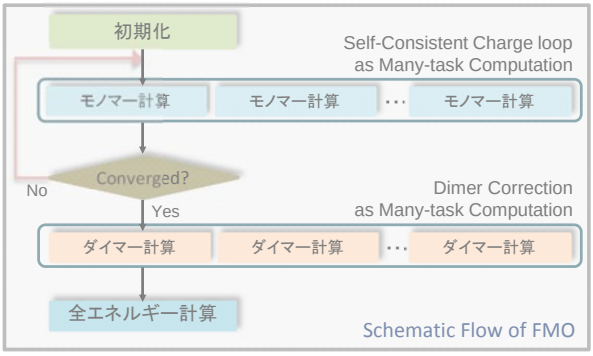
FMO計算のフローチャート



FMO計算のほとんどはモノマーやダイマーに対する小規模電子状態計算（フラグメント計算）

並列FMOプログラムの構造





- いくつかのgroupに分割して、各groupに小規模電子状態計算を割り当てる（**粗粒度並列処理**）
- group内プロセスで小規模電子状態計算を並列処理（**細粒度並列処理**）する
- Master-Worker型（ただしワーカー（グループ）間通信が存在する・・・）

超並列化に向けた最適化

OpenFMOの最適化と性能評価

OpenFMOとは？

- 九州大学、九州先端科学技術研究所で開発された並列FMOプログラム
- Hartree-Fock法に基づいた計算に特化しており、コードがコンパクト（C言語、約5万行）

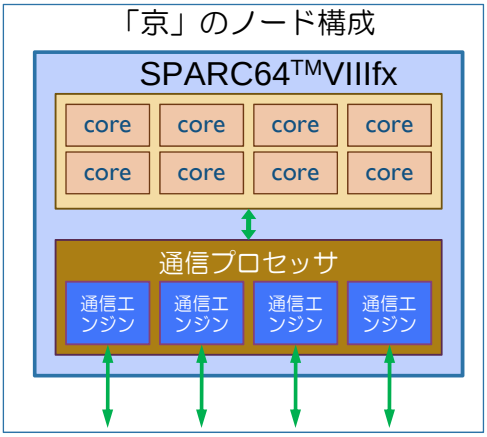
  さまざまな最適化の適用が容易！
- 中間データ（フラグメント密度行列データ）の保存方法を工夫することで、大規模入力にも対応
- 並列化における標準API（MPIやOpenMP）を用いる
- フリーのツール（MPICH2, OpenMPI, GCCなど）でも使えるようにする
- ターゲットはSMPクラスタ型並列計算機

これまでに行ってきた最適化

- MPI/OpenMPハイブリッド並列化
- 中間データの分散保存、アクセス方法の改良
- グローバルカウンタを用いた動的負荷分散による負荷均等化

MPI/OpenMP hybrid並列化

- SMPクラスタ型並列計算機を構成する各ノードには、プロセッサ（コア）数よりも少ないNICしか搭載されていないケースが多いので、Flat MPIプログラムでは、通信資源の競合による性能低下を引き起こす恐れがある
- MPIライブラリが使用するメモリ量や通信量の観点から、起動MPIプロセス数が少ない方が望ましい



OpenMPによるノード内スレッド並列とMPIによるノード間プロセス並列を組み合わせたhybrid並列化を適用した

ただし、分子積分部分にのみ手を加えており、行列計算に関連する部分は、ライブラリ任せ

モノマー密度行列の保存・アクセス方法

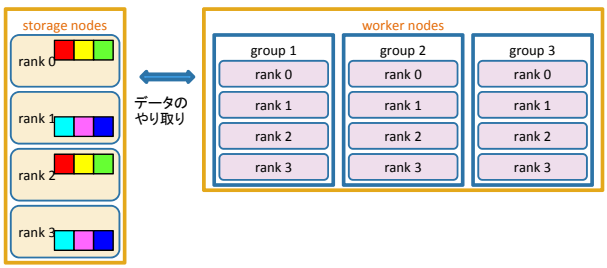
数万～10万フラグメントの超巨大分子に対するFMO計算を行う場合を想定して、2種類のモノマー密度行列の保存・アクセス方法を検討した

方法1：全体で1（複数）セットのデータを持つ

参照処理	ターゲットプロセスからGet+計算グループ内Bcast
更新処理	ターゲットプロセスへのPut
同期処理	記憶グループ間の対応するプロセス間でのBcast（または、Allreduce）

- （MPI-2の）片側通信が必要となる
- データ保存のための特別なプロセスが必要ない（コードがシンプル）

方法2：データ保存専用プロセスの利用



参照処理	ターゲットプロセスからのRecv＋計算グループ内Bcast
更新処理	ターゲットプロセスへのSend
同期処理	記憶グループ間の対応するプロセス間でのBcast（または、Allreduce）

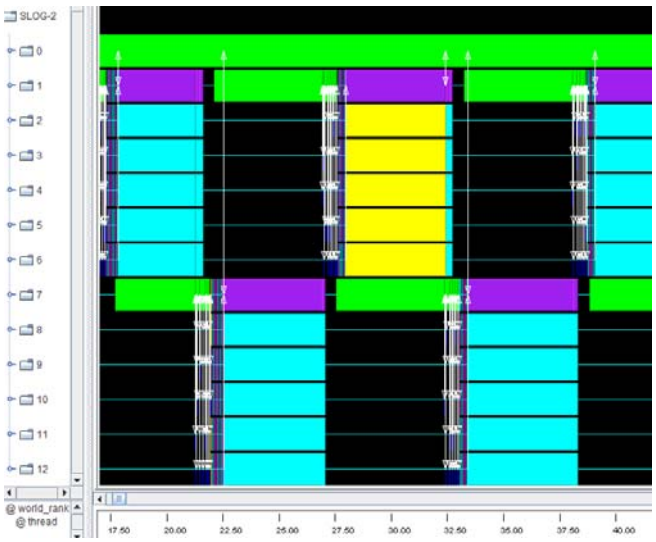
- MPI-1規格でも記述できる
- 記憶グループ用と計算グループ用の少なくとも2種類のコードが必要（コードが複雑）

アクセス方法による性能の違い

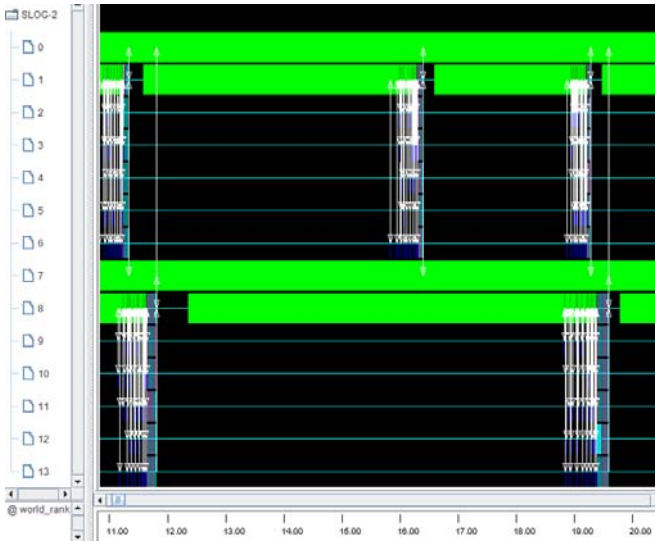
モノマー電子状態計算部分のプロファイル

入力＝アクアポリン（PDB ID=2F2B，492フラグメント、基底関数＝STO-3G）
計算機＝PowerEdge M610(quad core Xeon×2)×5, interconnect=10GbE, MPICH2-1.3.1, intel C++ compiler (12.0.0)

MPI-2の片側通信を用いた場合



中間データ保存専用プロセスを利用した場合

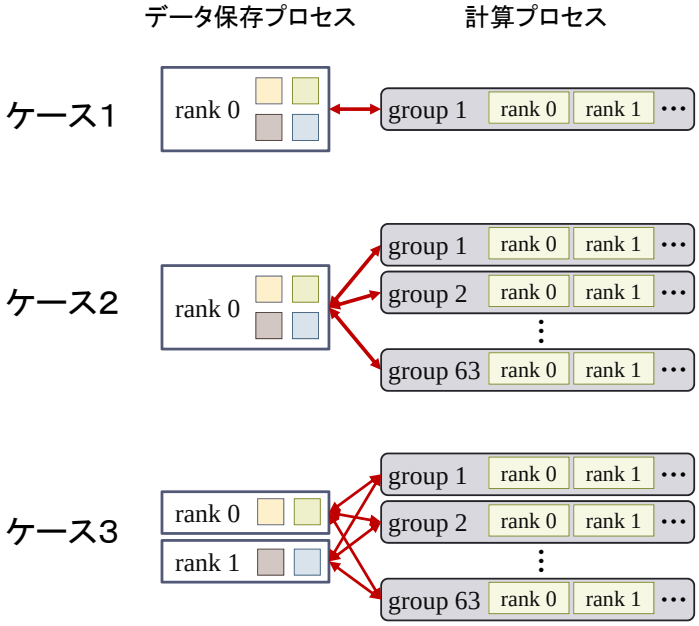


- MPI-2の片側通信の実装によっては、密度行列データへのアクセス性能が著しく低下
- データ保存専用プロセス用いた場合には密度行列データへの効率的なアクセスができた

方法2の大規模実行時の通信性能

データ保存プロセス数、計算プロセス（グループ）数とモノマー密度行列データ取得時間

測定環境: Riken Integrated Cluster of Clusters (RICC)，63 groups, 32 processes/group, 4threads/process (128 threads/group), OpenMPI(version 1.4.3), Intel C++ compiler (version 11.0)
入力データ:アデノウィルスKNOBドメイン(PDB ID=1nob, 576 fragments)，基底関数 6-31G*



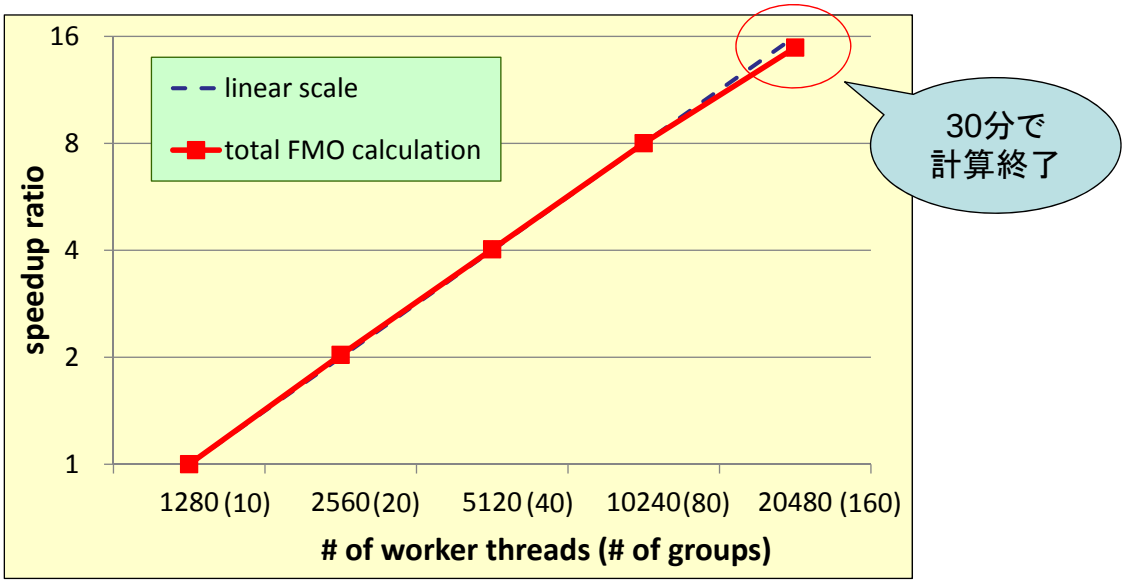
	データ取得時間*	総実行時間
ケース1	37.8 (37.8)	10773
ケース2	44.2 (0.70)	177.6
ケース3	37.3 (0.59)	177.8

* モノマー密度行列データを取得するために要した時間の全グループでの合計（かっこ内はグループあたり）

大規模並列時でも効率よく密度行列データにアクセスできる

粗粒度並列部分の並列化効率

Machine:FUJITSU PRIMERGY(CX400) in Kyushu University(16 processes x 8 threads/group (=128 threads/group), # of groups=10-160) **Compiler:** FUJISTU MPI + FUJITSU C compiler（version 1.2.0)
Input : Adenovirus KNOB domain (PDB ID=1nob, 576 fragments, 16,764 atoms), basis set= 6-31G* (142,974 functions)



2万並列時でも高い並列化効率を保っている！

小規模電子状態計算部分の模擬コード

```
calculation_SCF_energy(int ifrag) {
  for (id=0; id<(N_ifc4c+1); id++ ) make_cutoff_table(id);      // カットオフテーブル計算
  calc_twoint ( myrank, nprocs, ERI_buffer);                    // 2電子積分
  V=0.0;
  for ( i=0; i<N_ifc4c; i++ ) V+=calc_ifc4c( myrank, nprocs, i);   // 4中心クーロン積分
  for ( i=0; i<N_ifc3c; i++ ) V+=calc_ifc3c( myrank, nprocs, i);   // 3中心クーロン積分
  for ( i=0; i<N_frag; i++ ) V+=calc_ifc2c( myrank, nprocs, i);    // 2中心クーロン積分
  if ( myrank == 0 ) {
    calc_oneint( S, H );                                        // 1電子積分
    calc_projection( P );                                       // 射影演算子
  }
  MPI_Allreduce( V, comm );
  SCF_procedure( S, H, P, V, ERI_bufferi, D_ifrag ,comm );      // SCF計算
}
```

計算内容	1回あたりの計算時間	小規模電子状態1回あたりの呼び出し回数
カットオフテーブル計算	0.01～0.2秒	10～30
2電子積分	数10～数100秒	1*
4中心クーロン積分	数10～数100秒	10～30
3中心クーロン積分	数秒	10～20
1電子積分、射影演算子	0.05～0.5秒	1
2中心クーロン積分	0.05～0.5秒	フラグメント数（～10万）

計算時間の多くを占める各種分子積分計算の負荷均等化が並列性能向上の上で重要！

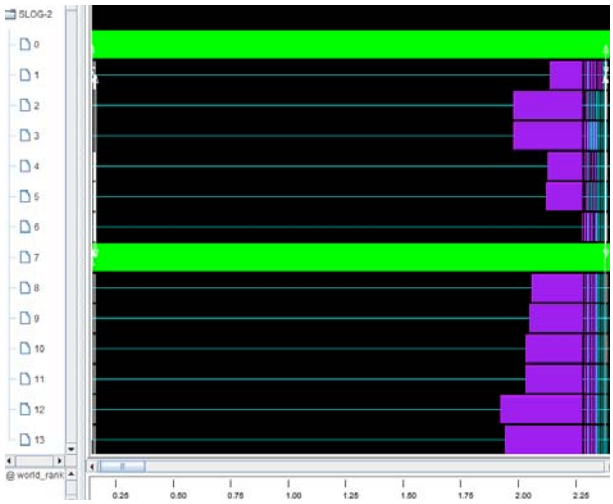
*; 大量の2電子積分（10^6～10^9個）が、すべてがメモリに保存できた場合

グローバルカウンタを用いた負荷均等化の効果

小規模電子状態計算実行プロファイル

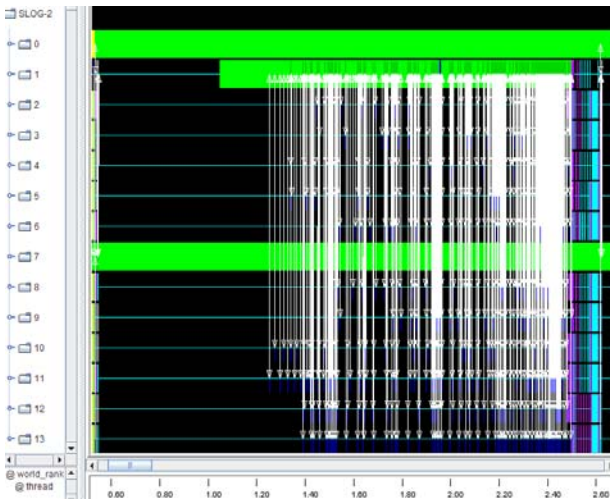
入力＝アデノウィルスのKNOBドメイン（PDB ID=1NOB，576フラグメント、基底関数＝STO-3G）
計算機＝PowerEdge M610(quad core Xeon×2)×5, interconnect=10GbE, MPICH2-1.3.1, intel C++ compiler (12.0.0)

グローバルカウンタによる負荷均等化なし



分子積分部分に負荷不均衡が見られる

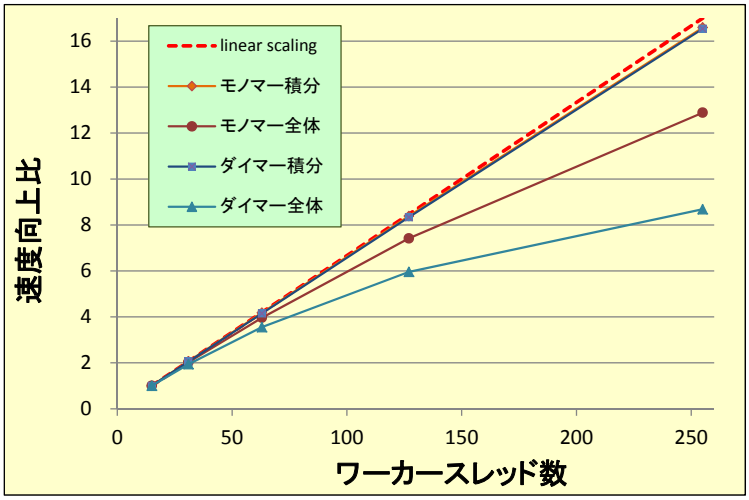
グローバルカウンタによる負荷均等化あり



分子積分計算の負荷均等化が図られている

フラグメント計算部分の速度向上比

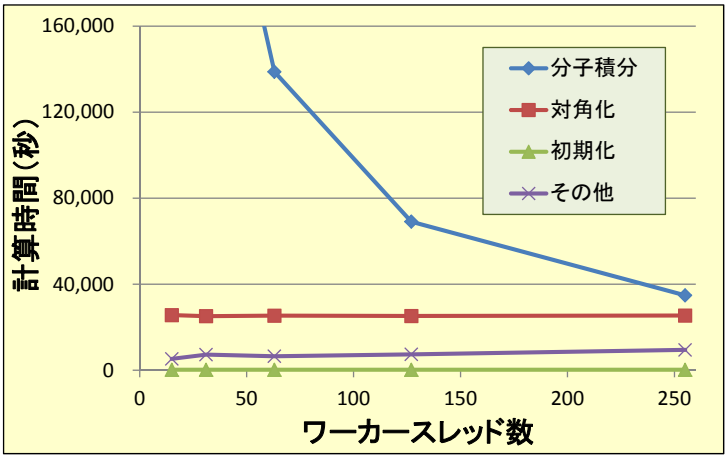
Input＝Adenovirus KNOB domain（PDB ID=1NOB, 576 monomers, basis set＝6-31G*）
Machine＝FUJITSU PRIERGY CX400 (8-core Xeon×2/node, 1476 nodes, Interconnect=InfiniBand FDR)
Compiler=Fujisu C compiler, MPI=Fujitsu MPI library, SSL II



分子積分計算部分だけなら、256並列時で約97%の高い並列化効率を達成しているが、全体では並列化効率が悪い・・・（特に、ダイマー計算）

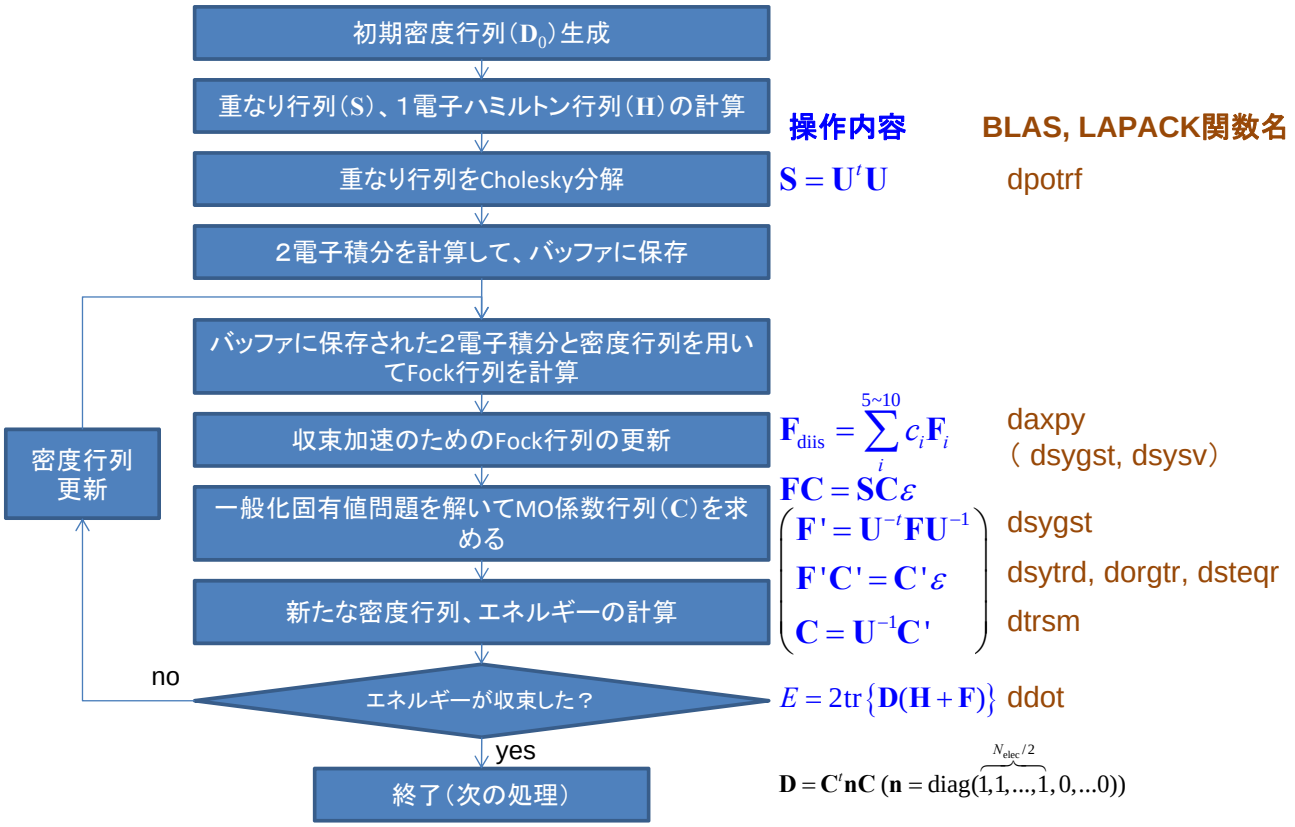
ダイマー計算における各部分の計算時間

Input＝Adenovirus KNOB domain（PDB ID=1NOB, 576 monomers, basis set＝6-31G*） Machine＝FUJITSU PRIERGY CX400 (8-core Xeon×2/node, 1476 nodes, Interconnect=InfiniBand FDR)
Compiler=Fujisu C compiler, MPI=Fujitsu MPI library, SSL II, 8 threads/process

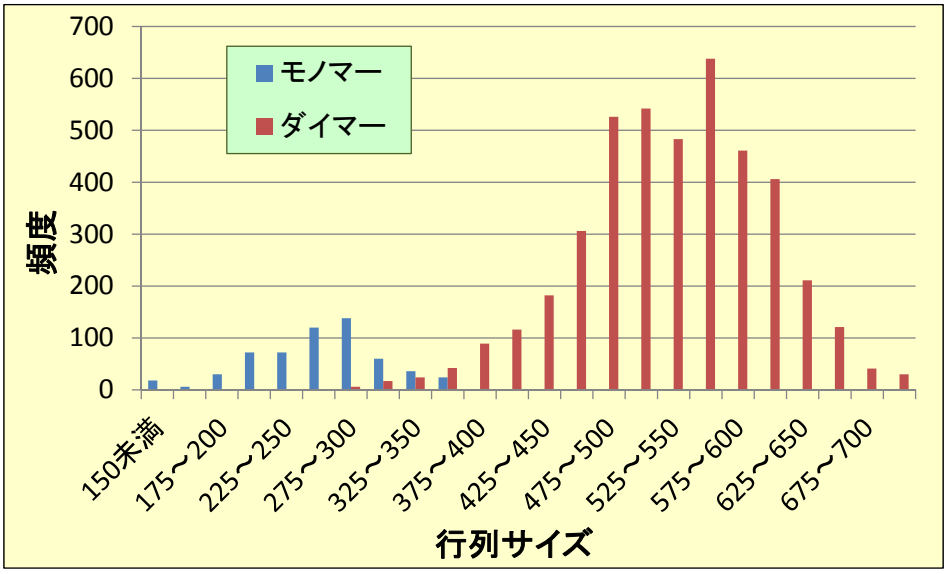


- 分子積分計算時間はスレッド数の増加とともに減少
- 対角化部分の計算時間は、ノード間並列していないため、ワーカースレッド数に依存せず一定

SCF計算で用いている行列演算関数



フラグメント計算部分で扱う行列のサイズ



一般化固有値問題の特徴

- 密行列である
- 1/2～1/3の固有値と対応する固有ベクトルが必要

まとめ

- FMO計算は、解くべき問題を分割して解くことで、これまで困難であった大規模生体分子に対する電子状態計算を可能にした
- FMO計算の粗粒度並列部分の並列化効率は非常に高い
- 細粒度並列部分（フラグメント計算の並列処理）では、計算量の最も大きな分子積分計算は高並列が可能であるが、非並列部分の存在などによる並列性能低下が大きい
- FMO計算で扱う行列サイズは小さいため、行列計算部分の並列化による性能向上は困難かも

  更なる並列性能向上のためには、対角化と他の処理（他のフラグメントの積分計算など）をオーバーラップさせるなどの対策が必要！

謝辞

以下のみなさま（敬称略）の協力を得ています

- 眞木淳（九州先端研）
- 本田宏明（九州大学）
- 高見利也（九州大学）
- 小林泰三（九州大学）
- 青柳睦（九州大学）

本研究の一部は理化学研究所と九州大学との共同研究で行っている。また、科学研究費補助金基盤研究（C）「超並列フラグメント分子軌道法プログラムライブラリの開発」（課題番号22550015）、および、JST,CRESTの研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の支援を受けている。さらに、学際大規模情報基盤共同利用・共同研究拠点の公募型共同研究平成23年度採択課題「超並列フラグメント分子軌道法プログラム OpenFMOの 性能評価と高性能化」による計算機利用を行っている。

ご静聴ありがとうございました