SSI: Scalable Software Infrastructure
大規模シミュレーション向け基盤ソフトウェアの開発

SSI
Scalable Software Infrastructure

本研究では，従来それぞれの分野において別個に進められてきた数値アルゴリズムや並列実装手法に関する知見をもとに，今後の計算環境の高並列化に対応したスケーラブルなソフトウェア基盤を整備することを目指している．基礎分野に関しては

・固有値解法
・連立一次方程式解法
・高速関数変換

の三分野を設定し，各分野の研究者の協力のもとに研究を進めるとともに，積極的にベンダとの共同研究を推進し，今後普及すると思われる計算機環境を想定した開発を行っている．

計算手法

まず，固有値解法，線形解法については，スケーラビリティの観点から共役勾配法・共役残差法系の反復解法を中心に研究を進め，これらの拡張について複数の提案，発表を行った．

また，代数的マルチグリッド前処理法に着目するとともにその並列実装手法に関して研究を進め，大規模計算機環境上で有効性を実証した．また，高速関数変換については，高速ルジャンドル変換の提案，ライブラリ化を行うとともに，高速フーリエ変換の効率的な実装手法について研究を行い，複数のアーキテクチャ上での効率的な実装方式を提案した．これらの成果の一部は論文として発表するとともに，ライブラリの核となる技術として現在実装を進めている．

仕様設計

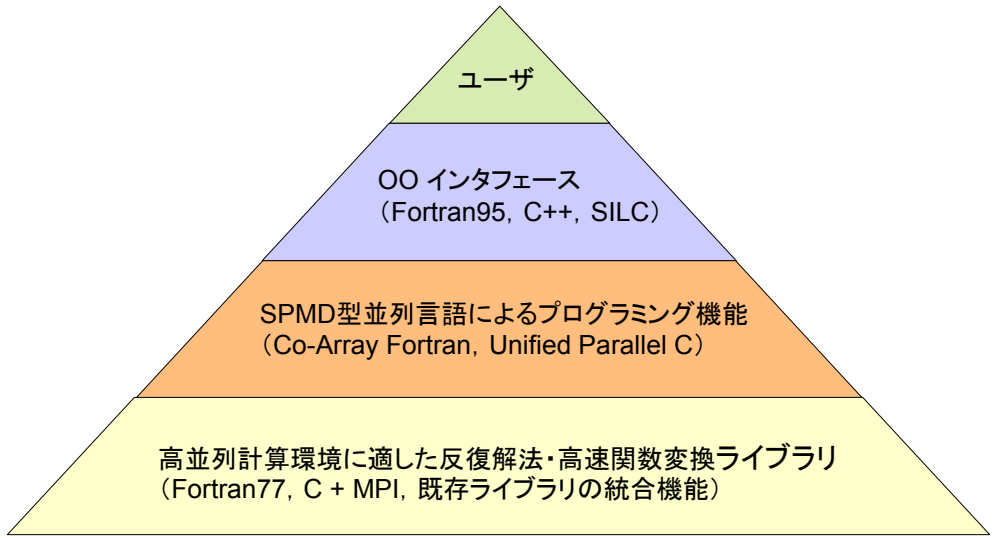
ライブラリの設計においては，可搬性を重視するとともに，利用者が効率的に処理を記述できる必要がある．本研究では，基礎的な数値ライブラリについては標準的なプログラミング言語（C, Fortran77）を用いて記述し，より高級な言語（C++, Fortran95）によりオブジェクト指向に基づいたインタフェースを付加する方針で実装を行っている．

並列処理に関してはMPI を標準的なインタフェースとして採用しているが，これと平行してSPMD 型並列言語の利用技術に関して研究を進めており，Co-Array Fortran に関しては平成17 年度よりスカラアーキテクチャを対象としたコンパイラの移植性向上に関する研究をCray 社と共同で開始する予定である．また，より抽象度の高いインタフェースとして，データの受け渡しと演算処理を分離し，演算処理を文字列で指定することにより計算環境への依存を取り除いたライブラリインタフェースSILC（Simple Interface for Library Collections）に関する研究を進めている．

実装

ハードウェア，システム技術に関しては，今後普及すると思われる利用形態を的確に予測するとともに，これらの上で高い性能を発揮することのできるソフトウェアを設計，開発していく必要がある．現時点では共有メモリ型並列計算機（SGI Altix 3700），ベクトル計算機（NEC SX-6i），及び中規模PC クラスタ（IA32）を導入し，最新の計算機環境のもとで可搬性を備えたライブラリの開発を行っている．

今後の傾向としては，マイクロプロセッサの消費電力上の制約から，より高並列な環境が一般的になってくるものと予想される．このような背景から，本グループでは平成16 年度よりIBM Watson 研究所との間で国内初の Blue Gene/L の利用に関する共同研究契約を締結し，数万プロセッサレベルでの高並列な環境下での数値ライブラリの実装技術について研究を開始している．今後応用分野の研究者との研究協力，設備の相互利用を拡大し，より大規模な環境を用いた評価を進めていく予定である．



担当：西田 晃（東京大学・JST）

SILC: Simple Interface for Library Collections

SSI
Scalable Software Infrastructure

目標

- 数値計算プログラムを計算機環境に依存しない形で記述し、書き換えなしに利用できること

基本アイデア

- データを預けて計算してもらう
- 計算リクエストは文字列（数式）で指示する
- 計算にはユーザのメモリ空間を使用しない

設計と実装

- 設計方針
  - ユーザプログラムとライブラリプログラムの分離
  - データ授受と計算リクエストの分離
- 実装
  - クライアント・サーバ方式
  - 逐次版と並列版

命令記述

- 環境に依存しない命令記述
- 線形代数を中心とする演算子
- 関数（戻り値あり）と手続き（戻り値なし）
- エイリアス（変数の相互干渉）の防止
  - エイリアスの例： f(X) + (X = Y)
  - 代入と手続き呼出しは文
  - 関数の引数は変更不可

連立一次方程式 $Ax = b$ を解くプログラムの例

従来法

```
CALL LU ( N, A, IP, STATUS )
IF ( STATUS.EQ.0 ) THEN
      CALL SOLVE ( N, A, IP, B )
END IF
```

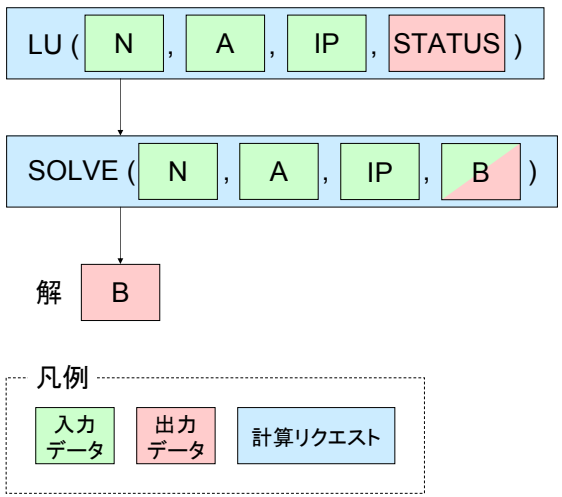
※解 x は配列 B に格納される

SILC

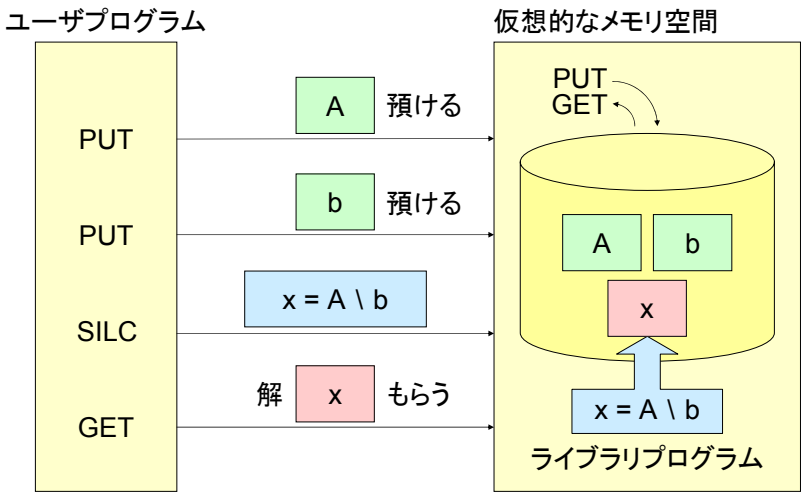
```
CALL PUT ( “A”, A, N )
CALL PUT ( “b”, B, N )
CALL SILC ( “x = A \ b” )
CALL GET ( “x”, X, N )
```

	従来法	SILC
プログラムとデータのメモリ空間	共通	独立
データ授受と演算リクエスト	同時	個別

従来法



SILC



担当：梶山民人（JST・東京大学）

反復解法ライブラリ

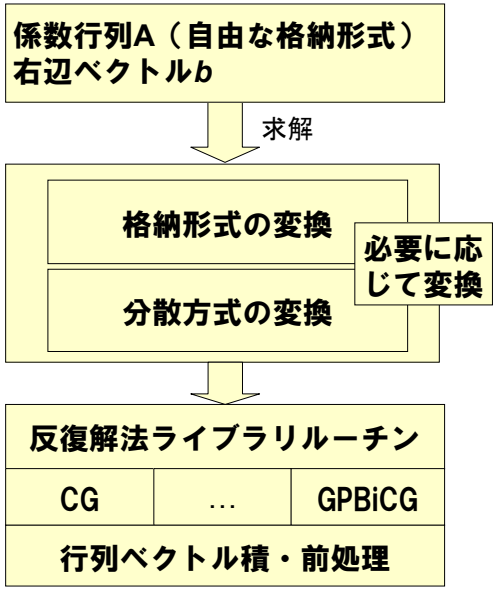
SSI
Scalable Software Infrastructure

特徴

- 様々な格納形式の入力を受け付ける
- 様々な解法と前処理を組み合わせて使える

反復解法

対称	CG
非対称	BiCG
	CGS
	BiCGSTAB
	BiCGSTAB(l)
	GPBiCG
	Orthomin(m)
	GMRES(m)
	QMR
	Jacobi
	Gauss-Seidel
	SOR




```
SSI_MATRIX *A,A2;
SSI_SCALAR *x, *b;
SSI_SCALAR params[SSI_PARAMS_LEN];
int options[SSI_OPTIONS_LEN];

ssi_matrix_inout_init(&minout);
ssi_matrix_input(&A,&b,&x,minout,SSI_MATRIX_TYPE_CRS,path);
ssi_matrix_convert(A,&A2,SSI_MATRIX_TYPE_JDS);
params[SSI_PARAMS_RESID] = 1.0e-8;
options[SSI_OPTIONS_MAXITER] = 500;
options[SSI_OPTIONS_SOLVER] = SSI_SOLVER_CG;
options[SSI_OPTIONS_PRECON] = SSI_PRECON_TYPE_NONE;
ssi_solver(A2,b,x,params,options);
```

CRS形式をJDS形式に変換後CG法で解くスケルトンプログラム

前処理

- Jacobi
- SSOR
- Block Jacobi
- ILU
- Hybrid
  - □反復法を使用
- I+S type
  - □単位行列と係数行列A=(a_{ij})のi+1の部分の符号を逆にしたもの

$$S = \begin{cases} -a_{ij} & j = i+1 \\ 0 & else \end{cases}$$

  - □バリエーション
    - ・I+$S^{(m)}$ ：対角側より非対角要素をm個使用
    - ・I+S_{max} ：各行絶対値最大の要素を1個使用

格納形式

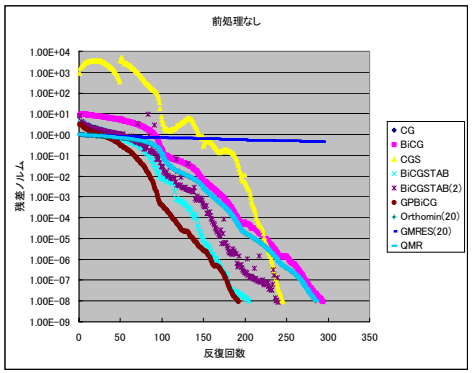
Point	Dense	(DNS)
	Coordinate	(COO)
	Compressed Row Storage	(CRS)
	Compressed Column Storage	(CSS)
	Modified Compressed Sparse Row	(MSR)
	Diagonal	(DIA)
	Ellpack-Itpack generalized diagonal	(ELL)
	Jagged Diagonal	(JDS)
Block	Block Sparse Row	(BSR)
	Block Sparse Column	(SBC)
	Variable Block Row	(VBR)

- 各解法･前処理に適した格納方式に自動変換

並列化

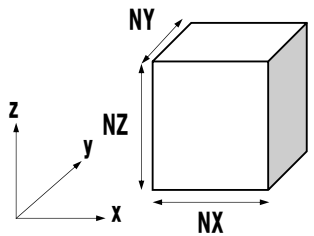
- 各解法･前処理に適したデータ分散方式に自動変換
- 共有メモリ型　OpenMP
- 分散メモリ型　MPI

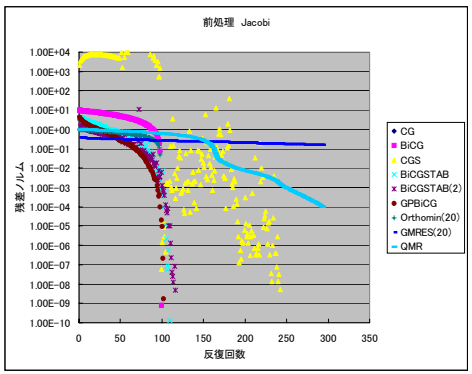
様々な解法を同時に比較できる



数値例

- ポアソン方程式　$-\Delta\phi = 1$
- 境界条件　$\phi = 0,\ Z = Z_{\min}$
- NX=NY=NZ=100
- 格納形式 CRS





担当：小武守恒（JST･東京大学）

AMG (Algebraic Multi-Grid solver)

SSI
Scalable Software Infrastructure

目的

Smoothed Aggregation Algebraic Multi-Grid法（SA-AMG法）を様々な高性能計算環境に実装する。

SA-AMG法の概略

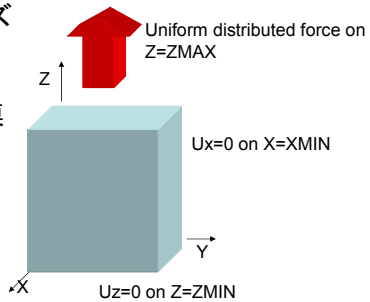
問題行列AからAと似た性質を持つサイズの小さい行列A'を生成する（構築部）。その後、サイズの小さい問題A'x'=b'とAx=bを交互に緩和法を適用しながら効率よく解く（解法部）。

特徴

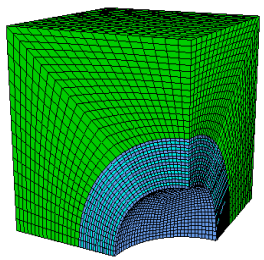
高速解法： 計算時間が問題サイズによらない

領域並列性：領域分割により大規模並列性を実現できる
- ➢ 問題領域の分割や再分散にParMETISライブラリを使用

Pulled Cube：



Hard Sphere in Cube：



ベクトル環境

NEC SX-6i（最大ベクトル性能 8GFLOPS ）上で以下の問題について実験を行った。

Pulled Cube： 立方体の弾性体の上面を上方向に引っ張る問題(GeoFEMのテスト問題)
Hard Sphere in Cube： やわらかい弾性体の中に硬い球面上の弾性体がはいっている。上面から圧力（Adams Mark's Finite element market）

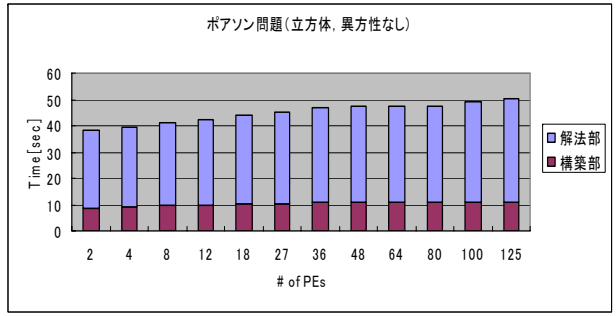
問題名	DOF	反復回数	Total (MFLOPS)	解法部 (MFLOPS)	構築部 (MFLOPS)
Pulled Cube	375k	24	1113	2733	562
Hard Sphere in Cube	150k	317	2318	2601	456

問題行列Aとベクトルの積はどちらの問題でも2.8～3GFLOPSで計算できる。この計算性能がAMG法の上限として考えられる。解法部は良好な性能。ベクトル化不可能な部分のある構築部の性能向上が課題である。

並列環境

実験環境：
Sun Blade1000（UltraSPARC III 750MHz×2, メモリ1GB）×128, Myrinet2000により接続

問題は立方体のポアソン方程式（異方性があるものとないもの）と立方体の弾性問題（Pulled Cube）を扱った。

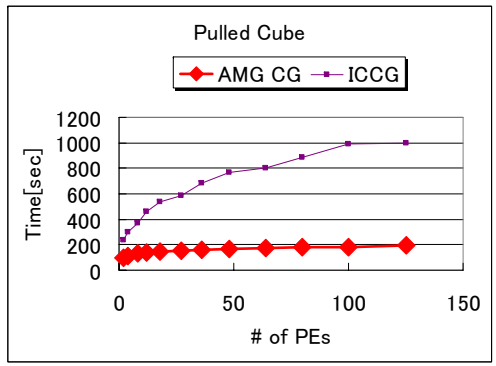


立方体のポアソン方程式 解法部と構築部の時間の割合：
2PE:25万DOF～125PE:1562万DOF

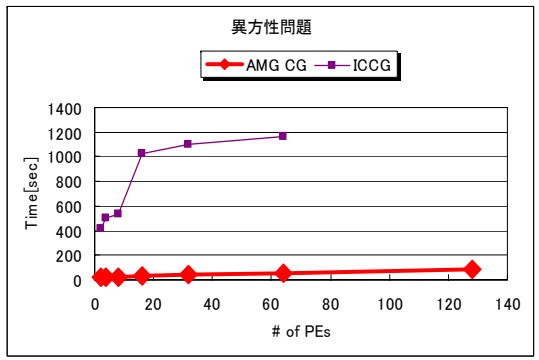
右上図： 簡単なポアソン問題（立方体、異方性なし）を解いたときの解法部と構築部の時間の割合である。サイズが大きくなっても構築部はほとんど時間が増えていない。この場合、構築部はだいたい5回から6回分の反復時間（V-cycle）で終わる。
⇒ 構築部の割合は全体の2割程度で、領域並列性がある

左下図： PE 数に比例させてサイズを大きくした Pulled Cube 問題では、SA-AMG CG法の実行時間は ICCG 法の実行時間の 1/5 になる（問題サイズ 1607万DOF）。
⇒ SA-AMG法は高速解法でかつ並列化性能が良い

右下図： 異方性問題を取り扱った例である。異方性問題では条件数は大きくなり、異方性の方向を意識した反復解法でないとなかなか収束しない。
⇒ SA-AMG法は異方性問題に対しても有効



Pulled Cube：
2PE：25万DOF～125PE：1607万DOF



異方性問題：立方体のポアソン方程式 10000倍の異方性：
2PE：12万DOF～128PE：819万DOF

今後の研究

計算性能の向上： ベクトル性能や並列アグリゲート生成戦略の改善
負荷分散： 状況に応じた負荷分散の実現
実用化： 様々な問題に適用し、評価、改善

担当：藤井昭宏（工学院大学・JST）

SSI
Scalable Software Infrastructure

Jacobi-Davidson 法

大規模疎行列向きの固有値解法

◆必要な固有値は数個
◆大規模な問題では全固有値を計算する QR 法等は使用できない
◆Lanczos / Arnoldi 法
- 射影部分空間上での小規模な固有値問題に帰着
- 固有値が近接している場合に計算が難しい

◆Davidson 法
- 量子化学で対角化の計算に利用
- 残差について修正方程式を解く
- 精度に問題

◆Jacobi-Davidson 法
- Davidson 法を修正
- 修正ベクトル成分を近似固有ベクトルの直交補空間から選ぶ

Jacobi-Davidson 法の特性

◆アルゴリズム
- 反復解法
- 第 k ステップにおいて, 固有値問題 $Ax = \lambda x$ に関する次元 k の部分空間
  K= (v_1, …, v_k) 上で固有対 (θ _k, u_k) を計算
- 残差 r = A u_k – θ _k u_k に関する修正方程式
  M_k t = r
  を計算, t を K と直交化して v_{k+1} を得る
- V_{k+1} = [v_1,…,v_{k+1}] から
  H_{k+1} = V*_{k+1} A V_{k+1}
  を計算, H_{k+1} から新しい Ritz 対 (θ _{k+1}, u_k) をQR 法で求める

修正方程式

◆固有値計算部の残差 r = A u_k – θ _k u_k に関して修正方程式
M_k t = r
を解き, 解 t により固有ベクトルを修正
- Lanczos / Arnoldi 法
  - M_k = I （修正を行なわない）
- Davidson 法
  - M_k = diag(A) – θ _k I（適用が容易）
- Jacobi-Davidson 法
  - A(u_k + t) = λ (u_k + t) から
    M_k = - (A_P – θ _k I), A_P = (I – u_k u*_k) A (I – u_k u*_k)
    (Ax = λ x を u_k の直交補空間に射影)
    を導出.

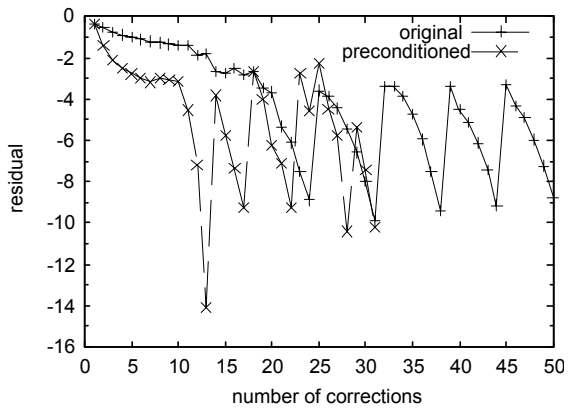
前処理

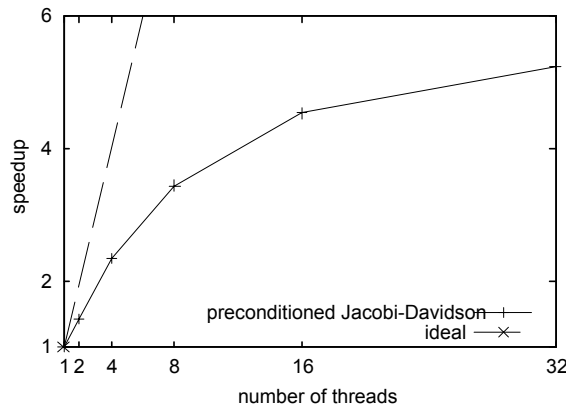
◆修正方程式 M_k t = r は JD 法の計算量の大部分を占めるため, 効率的に解く必要.
M_k は大規模疎行列であることから, 前処理付反復解法が適当
◆Jacobi-Davidson 法の前処理としては,
1. 問題の物理的性質を仮定しないこと
2. 並列性の高い解法であること

を満たす必要

性能評価

◆2 次元 Poisson 差分方程式, 問題サイズ256^2, 最大固有値1個を計算
◆Jacobi 前処理を使用
◆並列化
- zgemv, zdotc, zaxpy, (zxpay) の4関数を並列化
- 最外ループを static に各スレッドに分割
- 使用環境
  - Sun Enterprise 10000, 64-way UltraSPARC II SMP
  - 250MHz, 1MB cache
  - Solaris 7 on Sun Ultra Enterprise 10000
  - コンパイラには Sun WorkShop f90, 及び Omni OpenMP compiler を使用
  - 32スレッドまで評価





担当：西田 晃(東京大学･JST)

SSI
Scalable Software Infrastructure

非線形最適化問題としての固有値解法

非線形最適化問題

◆非線形関数$F(x)$の局所的な最小点を求める.

◆各段階の試行点xからFの値が減少する方向vへ探索を進めると，各段の方向がすべて共役ならば，2次形式

$$F = a + b^T x + 1/2 x^T A x$$

の最小点は，任意の出発点$x^{(1)}$から有限回の降下ステップの計算によって求められる

◆最適化手法を適用する関数が最小点の近傍では2次形式で十分に近似できると仮定

→ 共役方向を用いる傾斜法を一般の関数の最小化に使うためには，関数を2次形式で表したときに共役方向になるような探索の方向を作ればよい.

共役勾配法の導入

◆以上の手法を用いて，実対称行列A, Bに関する一般化固有値問題

$$Ax = \lambda Bx$$

の最小固有値を共役勾配法により求めることを考える

◆これは Rayleigh 商

$$\mu(x) = x^T Bx \ / \ x^T Ax$$

の極値問題に帰着することができ，最急勾配方向が

$$\nabla \mu(x) \equiv g(x) = 2(Bx - \mu Ax) / x^T Ax$$

であることから，Rayleigh 商を局所的に最適化する係数α_iを用いて，共役勾配法

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

$$p_i = -g_i + \beta_{i-1} p_{i-1}, \quad \beta_{i-1} = (g^T_i g_i) \ / (g^T_{i-1} g_{i-1})$$

により固有値を計算する.

◆<u>適切な前処理と組み合わせることによって高速に固有値を計算可能！</u>

前処理付固有値解法のアルゴリズムは，前処理行列$T \approx A^{-1}$, TA, TB に関するm_k次多項式$P_{m_k}(TA, TB)$を用いて以下のように行う

（１）初期ベクトル$x^{(0)}$を選択する

（２）m_k回の反復により$x^{(k)} = P_{m_k}(TA, TB)x^{(0)}$を計算する

（３）$\mu^{(k)} = (x^{(k)}, Bx^{(k)}) \ / \ (x^{(k)}, Ax^{(k)})$を計算する

◆前処理付共役勾配法の反復は，適当な初期ベクトル$x^{(0)}$と対応する修正ベクトル$p^{(0)} = 0$を用いて，

$$\mu^{(i)} = (x^{(i)}, Bx^{(i)}) \ / \ (x^{(i)}, Ax^{(i)})$$

$$r = Bx^{(i)} - \mu^{(i)} Ax^{(i)}$$

$$w^{(i)} = Tr$$

$$x^{(i+1)} = w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} p^{(i)}$$

$$p^{(i+1)} = w^{(i)} + \gamma^{(i)} p^{(i)}$$

と書ける

◆行列束$Bx^{(i)} - \mu^{(i)} Ax^{(i)}$に関する span$\{w, x^{(i)}, p^{(i)}\}$ 上のRitz 値，Ritz ベクトルはRayleigh–Ritz 法により計算可能 （3x3の固有値問題）

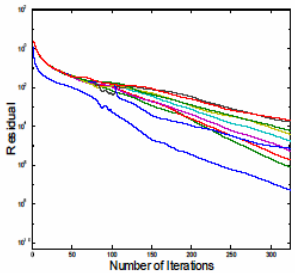
◆最大Ritz 値に対応するRitz ベクトルを$x^{(i+1)}$とする

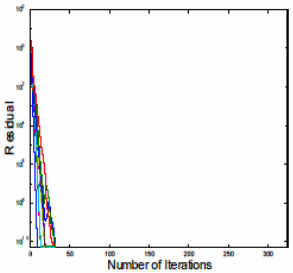
◆係数 $\tau^{(i)}$, $\gamma^{(i)}$ の値は， span$\{w, x^{(i)}, p^{(i)}\}$上での局所的な最適解をもとに計算

性能評価

◆代数的マルチグリッド前処理（BoomerAMG，縮約は Cleary–Luby–Jones–Plassman）

- 7点3次元 Laplace 問題の係数行列 (1003元，非零要素数6,940,000)について，共役勾配法を用いて10 個の固有値を計算した場合の各前処理手法の収束特性を反復回数により評価
- 相対残差ノルムの閾値は10^{-6}. 最も収束の遅い固有対が求まるまで反復





利点

◆ブロック化が容易 → Jacobi–Davidson 等 （減次を使用） と比較して有利

◆必要な記憶容量が少ない → Lanczos 法等 （反復ベクトル） と比較して有利

→ 従来のKrylov部分空間法と比較して効率的な計算が可能

◆適切な前処理により収束特性の改善が可能

- AMG 前処理等との組み合わせ → 有効

◆スケーラビリティ → 良好 （CG+並列前処理）

課題

◆非対称問題への拡張

- 残差 $r = \lambda Bx - Ax$, $\lambda = (Ax, Bx) / (Bx, Bx)$ を用いて汎関数 $F(r) = (r, r)$ の極値問題に帰着可能

◆Smoothed Aggregation MG 前処理との組み合わせ

◆さまざまな非対称問題を用いた評価

◆大規模アプリケーションへの適用

担当：西田 晃（東京大学・JST）

高速フーリエ変換

SSI
Scalable Software Infrastructure

高速フーリエ変換（FFT）とは

FFTは離散フーリエ変換を高速に計算するアルゴリズムであり、現在科学技術計算、大規模シミュレーション、マルチメディア等非常に幅広い分野で用いられている。

代表的なライブラリソフトウェア

◆最適化されたベンダー提供のライブラリ

- Intel MKL(Math Kernel Library)
  - IA-32,IA-32e,IA-64向けに高度にチューニング
  - 逐次、SMP並列をサポート。Ver.8でMPI対応予定
- IBM ESSL/PESSL Library
  - IBMのPowerシリーズアーキテクチャにチューニング
  - 逐次，SMP並列，MPI並列をサポート
  - サブルーチンに与えるパラメータが豊富

◆オープンソースライブラリ

- FFTWライブラリ
  - MITで開発
  - 多数のプラットフォームをサポート
  - 自動チューニング機能を搭載
  - 各プロセッサの拡張命令に対応
  - 逐次、SMP並列、MPI並列をサポート

インタフェース

ユーザがあるFFTのライブラリ関数を使う場合

◆従来のライブラリ

①ライブラリのマニュアルを読む
②必要な作業領域を確保
③要求される格納方式に変換／再配置
④必要な引数を渡す
という一連の作業が必要。

PESSLの3次元FFT

```
PDCFT3(X,Y,N1,N2,N3,ISIGN,SCALE,ICONTXT,IP)
```

◆**本ライブラリ**

①どのプラットホームでも関数のインタフェースは共通
②作業領域は自動的に確保
③多数の格納方式を利用可能
④引数の数も少ない

SILCの場合

```
SILC("Y=FFT(X)")
```

変数Xが3次元配列であれば3次元FFTと解釈される。
また各アーキテクチャに最適化されたライブラリも同じ形式で呼び出すことができる。

データの分散方式

従来のライブラリでは入力データの分散方式は決められており、また通常同じ分散方式で出力データを返す。3次元データをZ軸方向に1次元分割した場合、3次元FFTの計算は**図1**のような通信パターンになる。

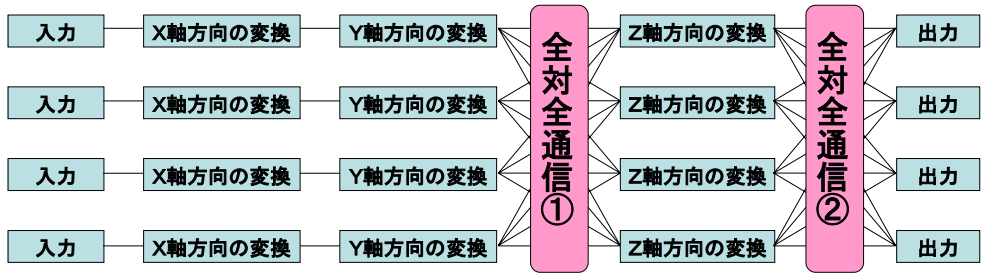


図1　並列3次元FFT計算の通信パターン

全対全通信は**①Z軸方向のデータ交換、②出力データの再分散**の二つが必要となる。

入力データと出力データの分散方式を異なるものにすることで**②出力データの再分散**が不要になり、実行時間を大幅に短縮できる。

既存のライブラリ関数の置き換えが容易

再分散の回数を減らして実行時間を短縮

FFTカーネルの開発

FFTの並列計算においてノード間の通信時間が占める割合が大きいものの、ノード内の計算の中心部分であるFFTカーネルの高速化も重要である。我々は既存のライブラリが採用するものより高速なカーネルの研究を進めている。一例として積和演算命令に適した8基底FFTカーネルを提案した。これはLinzerらの積和演算化手法を図3のカーネルに適用することで得られる。Linzerの8基底カーネルと比べてメモリアクセス命令数が少ないという利点がある。

$wr * x \pm wi * y \Rightarrow wr * (x \pm (wi/wr) * y)$

積和演算

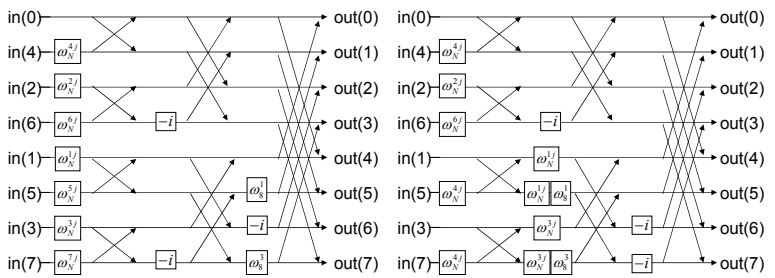


図2　従来の8基底FFTカーネル　　図3　提案する8基底FFTカーネル

表1　カーネルの命令数の比較

	Linzer	proposed
Load データ	16	16
Load ひねり係数	15	**14**
Store	16	16
積和演算	66	66

担当：額田彰（JST・東京大学）

高速球面調和関数変換ライブラリの構築

SSI
Scalable Software Infrastructure

球面調和関数変換とは

- ◆球面調和関数は2次元球面上の直交関数系で、1次元円周上の三角関数に相当
- ◆緯度・経度格子上の関数値と、球面調和関数展開の係数との間の変換が「球面調和関数変換」
- ◆球面上の（球座標による）偏微分方程式の標準的な解法を与える
- ◆気象予報・気候予測・地磁気分析などに日常的に用いられるほか、画像処理などにも応用

高速球面調和関数変換アルゴリズム

- ◆球面調和関数は、東西方向の三角関数と南北方向のルジャンドル陪関数との積に分解できる
- ◆南北 N 点、東西 $2N$ 点の緯度・経度格子では、計算量が以下のようになる

$$O(N^2 \log N) + O(N^3) = O(N^3)$$

FFT で高速計算　　ルジャンドル陪関数変換を直接計算：律速になる

⇒ ルジャンドル陪関数変換の高速アルゴリズムが必要！

球面調和関数/ルジャンドル多項式の高速変換アルゴリズムの比較

手法	計算量	アイディア	安定性
我々	$N^2 \log N$	FMM	**安定**
Mohlenkamp	$N^2 \log^2 N$	FWT	**安定**
Driscoll-Healy	$N^2 \log^2 N$	FFT/FCT	不安定
Orszag-Mori	$N \log N$	FFT	$m = 0$ のみ
Alpert-Rokhlin	$N \log N$	FFT/FMM	$m = 0$ のみ

ライブラリへの進展1：非分離内挿アルゴリズム

- ◆従来、我々のアルゴリズムでは分離ルジャンドル関数を用いていた
- ◆今回、我々が開発した一般化高速多重極子展開法により、分離ルジャンドル関数を用いないアルゴリズムを構築した
- ◆利点1：コード量が激減
- ◆利点2：計算量も若干減少
- ◆課題：前処理時間が増加した

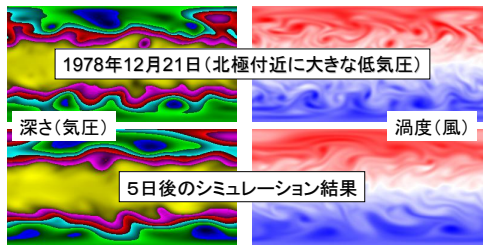
直接計算との計算量比

N	精度 10^{-6}		精度 10^{-12}	
	分離	**非分離**	分離	**非分離**
127	1.37	**1.42**	1.21	**1.24**
170	1.47	**1.53**	1.28	**1.31**
255	1.64	**1.72**	1.40	**1.44**

ライブラリへの進展2：応用問題での評価

- ◆STSWM のルジャンドル陪関数変換のみを我々のプログラムに入れ替え、Williamson の 7 つのテストケースすべてを評価
  - □ Williamson の 7 つのテストケース：移流のみ、定常、非定常、実データを含む球面上の偏微分方程式のベンチマーク問題
  - □ STSWM (NCAR Spectral Transform Shallow Water Model) Williamson の 7 つのテストケースを球面調和関数変換法で解く fortran77 プログラム
- ◆すべての問題が安定に解けた（1年までの長期シミュレーションも実証）
- ◆設定精度と計算誤差は十分に一致

Williamson のテストケース 7 の結果



その他：http://www.na.cse.nagoya-u.ac.jp/~reiji/fltss/

設定精度と計算誤差

精度	L1	L2	Linf
1e-6	1.3e-5	1.5e-5	2.1e-5
1e-9	2.8e-9	3.0e-9	4.3e-9
1e-12	3.7e-12	4.0e-12	5.7e-12
直接計算	1.1e-14	1.3e-14	1.9e-14

Case 2, Height error, $\alpha = 0$, N = 127, dt = 900s

今後の研究

- ◆さらなる高性能化
  - □ 計算量の削減（チューニング）
  - □ キャッシュ/ベクトル向けチューニング
- ◆他の計算への展開
  - □ 他の重要な直交関数展開への適用
  - □ 一般化 FMM のさらなる応用
- ◆SILC による性能・利便性の向上
  - □ スペクトル空間での演算を SILC 上で
    ← 特殊な配列形式（i.e. 三角切断）
  - □ 並列時のデータ分散の自動的最適化
    ← 三角切断のため特殊な分散が必要
  - □ 高性能 FFT との連結

担当：須田礼仁（東京大学・JST）