基盤研究(B)(1)「科学技術計算に現れる超大規模線形方程式の数理的諸問題と高速解法の総合的開発」経過報告

長谷川 秀彦 (筑波大学 図書館情報学系)

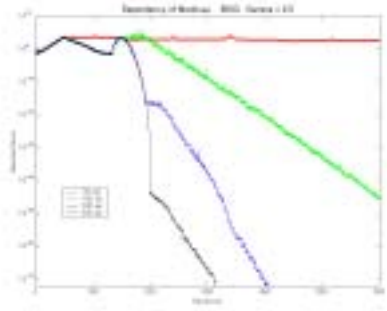
どのように評価するか？

1. 浮動小数演算の効果を調べる
2. 異なる精度で同じ問題を解く
   Omni OpenMP Compiler を使用
   http://phase.hpcc.jp/Omni/
3. まずは、収束の履歴をみよう！
4. それから、コストを考える

2003 年度に何をしたか

- 積型反復法の計算精度依存性の分析
- 帯行列に対する直接解法の並列化
- 非対称行列に CG 法を適用する

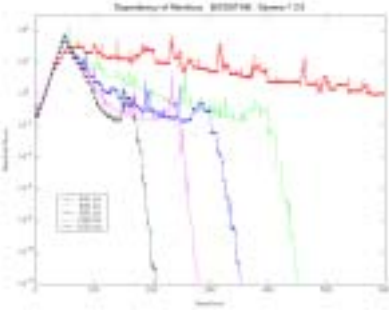
BiCG　Gamma = 2.5



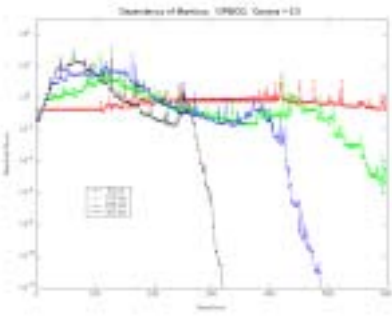
クリロフ部分空間法の計算精度依存性

筑波大学･図書館情報学系
長谷川 秀彦

BiCGSTAB　Gamma = 2.5



GPBiCG　Gamma = 2.5



Double with ILU vs Quadruple

	Double with ILU	Quad.	ratio
Sun (WS)	$0.268*10^{-1}$	0.602	22.4
VPP500 (Vector)	—	—	—
VPP300* (Vector)	$0.341*10^{-1}$	6.63	194
VPP800 (Vector)	$0.367*10^{-2}$	$0.378*10^{-1}$	10.2
SR8000 (SMP)	$0.287*10^{-2}$	$0.540*10^{-2}$	1.8
MP5800 (Mainframe)	$0.796*10^{-2}$	$0.171*10^{-1}$	2.1

わかったこと

- 計算が正確なら、素直に早く収束する
- 必要な仮数ビット数は問題依存：
  BiCG　53 bit for 1.3
  100 bit for 1.7
  200 bit for 2.1
  200 bit for 2.5
- 必要な仮数ビットは算法依存：
  Gamma = 2.5　BiCG 200 bit 190 itr.
  CGS 300 bit 160 itr.
  BiCGSTAB 1500 bit 210 itr.
  GPBiCG 300 bit 310 itr.

まとめ（とりあえず）

1. 収束を安定させるにはもっと仮数を
2. Quadruple はおすすめ：
   簡単・単純・高速
3. 最適な算法は計算環境によって違う
4. 正確な演算環境ではBi-CG がベスト

正確な計算が可能なソフトウェア

効果・コスト

◎○Multiple Precision Package (GNU MP)
Omni OpenMP Compiler
BNCpack with MPI

◎△Symbolic Computing(Computer Algebra)

○△Interval Arithmetic

○◎Quadruple Floating-Point Operations

ー☆Double Floating-Point Operations

これから

1. A、β、ベクトル量に対する解析
2. 実問題：大規模＆悪条件？
3. 改善と高速化（並列化？）
4. 簡単で効果的なツールを探す
   評価基準はいろいろ、ラクをするだけではない

OpenMP を用いた

帯行列に対する直接解法の並列化

筑波大学　長谷川 秀彦

帯行列に対する直接解法

- 帯行列を対象
  (対角の両側 i-m1 ≦ j ≦ i+m1 以外はゼロ)
- 基本的にはガウスの消去法 (直接解法)
- 非ゼロ要素＋Fill-in部のみをメモリに格納
  (ゼロでなくなる部分)
- 消去領域が局所的
  ー＞キャッシュにはよいが、並列性希薄

必要な計算資源

	直接解法			反復解法
行列形式	密行列	帯行列	対称正定値帯行列	非対称 7点差分 BiCG
メモリ容量	n^2	$3nm$	nm	$7n+10n$
	80GB	720MB	240MB	13.6MB
積和演算	$n^3/3$	$2m^2n$	$m^2n/2$	$30np$
	600Tflops	36Gflops	9Gflops	1.8Gflops

帯半幅 m=300，元数 n=10^5, 反復回数 r=300

どんなマシンで？

- DELL PowerEdge 6350@**筑波大**:
  4 * Pentium III (550MHz; 550MFLOPS)
- HITACHI SR8000 1 node@**原研**:
  8 * PowerPC? (375MHz, 1.5GFLOPS)
- SGI Altix@**東京大**:
  32 * Itanium II (1.3GHz, 5.2GFLOPS)
- NEC SX-6/4B @**原研**: 4 * ?? (8GFLOPS)

現在ではどれくらい？

M1=500, N=250,000, Itanium2(1.3GHz) 8CPU, 3GB

	プログラム	左辺の分解	右辺の計算 10 回
一般帯	BGLU4 & BGSLV4	41 sec (5.9G)	40 sec (190M)
右辺重視	BHLU4 & BHSLV4	43 sec (5.8G)	16 sec (461M)
対称	BSLU4 & BSSLV4	10 sec (5.8G)	10 sec (500M)

帯ガウスのアルゴリズム：原型

```
DO k = 1, N
   部分軸選択など
   DO i = k+1, min(k+m1,N)
      DO j = k+1, min(k+2*m1,N)
         a(j-i,i) = a(j-i,i)-a(k-i,i)*a(j-k,k)/a(0,k)
      END DO
   END DO
END DO
```

コード様々-1　m1=100/m1=200

原型	PE6350 (DELL/PGI)	SR8K (HITACHI/局左)	Altix (SGI/Intel)	RS6K (IBM/IBM)
配列のまま	83M 36M	252M 295M	957M 811M	284M -
スカラー T 使用	84M(15%) 35M	254M(16%) 294M	1810M(34%) 811M	348M -
1次元配列 W 使用	82M 35M	251M 288M	316M 1130M	280M -
T & W 使用	82M 35M	256M 290M	1330M 1131M	354M(22%) -

逐次コードの高速化

- 並列化の前に高速化を！
- スカラー・作業配列を使うと性能アップ
  （まだまだコンパイラは頼りない！？）
- 原型のままだと　15-34% of Peak
- アンローリング：2段同時 1.1-1.9 倍
  2段2行同時 1.5-2.1 倍
- 最終的には　26-52% of Peak
- ATLAS/BLAS の LAPACK はもっとよい！

Unrolling Summary　* $m1^2$ N

Unrolling	Computation	LOAD	STORE	BRANCH
No Unrolling	1 / loop	4	2	2
2段: k	2 / loop	3	1	1
2行: i	2 / loop	3	2	1
2列: j	2 / loop	3	2	1
2段2行: k & i	4 / loop	2	1	0.5

OpenMP を用いた並列化

- コンパイラによる（半）自動並列化
- ループなどに並列化指示を与える
- 並列化指示は逐次コードではコメント
- 並列度は実行時の環境変数で決定
- アルゴリズムの正しさはプログラマ次第
- 安易な並列化：
  （いちばん重要な部分だけにディレクティブを）

アンローリング　m1=100/m1=200

	PE6350 (DELL/PGI)	SR8K (HITACHI)	Altix (SGI/Intel)	SX-6 (NEC)
原型	82M(1.0) 35M	256M(1.0) 290M	1.33G(1.0) 1.13G	1.68G(1.0) -
2段同時 k で展開	152M(1.8) 65M	492M(1.9) 593M	1.57G(1.1) 1.45G	2.60G(1.5) -
2段2行同時 k と i に展開	145M(1.7) 69M	504M(1.9) 617M	2.70G(2.0) 2.49G	2.99G(1.7) -

OpenMP の実例

```
!$OMP Parallel Do Private(T)
      DO i = k+1, min(k+m1,N)
         T= -a(k-i,i)/a(0,k)
          DO j = k+1, min(k+2*m1,N)
            a(j-i,i) = a(j-i,i)+T*W(j)
         END DO
      END DO
```

Performance(Speedup) 100/200

Threads	1	4	8	16
PE6350	82M 145M	290M(3.5) 423M(2.9)	-	-
SR8K	185M 504M	-	1.3G(7.0)/1.5G(8.1) 2.4G(3.9)/3.5G(7.1)	-
Altix	1.1G 2.4G	1.6G(1.4) 4.2G(1.7)	1.5G(1.3) 2.9G(1.2)	1.0G(0.9) 1.5G(0.6)

OpenMP's Parallelization のまとめ

- OpenMP は中間のループに使おう.
- 最適な directive マシンに依存する.
- なんてってたって簡単. cost-effective.
- Tools は完全ではないが、使える.
- 性能はどう評価するべきか?
- Not Scalable – Shared machine.
- Unrolling と Tuning は依然として必要.

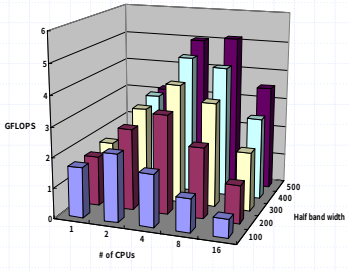
Performance(speedup) on Altix

Threads	Serial	1	2	4	8	16
m1=400	3.06G (1.15)	2.60G (1.00)	4.80G (1.84)	6.04G (2.40)	4.87G (1.84)	2.78G (1.03)
m1=500 (3.0GB)	No Memory	0.73G (1.00)	4.87G (6.57)	6.77G (9.17)	5.97G (8.08)	3.58G (4.79)

Information to use

Machine	Methods	m1	Decomp.	Solve #10
DELL(2.2G)	OpenMP,4	200	15s(.47G)	11s(41M)
SR8000(12G)	OpenMP,8	400	38s(2.6G)	9.7s(.39G)
SR8000	Auto, 8	400	26s(3.8G)	8.3s(.46G)
SR8000	Auto, 8	500	64s(3.8G)	13s(.55G)
SX-6(8G)	Serial, 1	300	10s(3.1G)	22s(70M)
Altix(83.2G)	OpenMP,8	500	41s(5.9G)	40s(.19G)

Size and Number of CPUs BGLU2



非対称行列を対称化して得られる行列に対する CG 法

筑波大学　長谷川秀彦
東京大学　曽我部知広
早稲田大学　荻田武史

非対称 $\begin{pmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ r & 0 & 2 & & \\ & r & 0 & \ddots & 1 \\ & & r & 0 & 2 \end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} b \end{pmatrix}$ (A) Toeplitz

⇩

対称 $\begin{pmatrix} & & r & 0 & 2 \\ & & 0 & 2 & 1 \\ & & 2 & 1 & \\ r & & & & \\ 0 & & & & \\ 2 & 1 & & & \end{pmatrix} \begin{pmatrix} x \end{pmatrix} = \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix} \begin{pmatrix} b \end{pmatrix}$ (PA, P)

CG法にて良好な収束

条件数不変

曽我部 NAS2003

Ayachour

$$\begin{pmatrix} \lambda I & A \\ A^T & O \end{pmatrix}$$

Saunders

$$\begin{pmatrix} \delta I & A \\ A^T & -\delta I \end{pmatrix}$$

前処理として使用

反復法として使用

λ, δ をコントロール

$\lambda x = A^T x' + \ldots$

$\lambda \to 0$

対称な拡大行列

$$\begin{pmatrix} & A^T \\ A & \end{pmatrix} \begin{pmatrix} x \\ a \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

$$\frac{1}{2} \begin{pmatrix} A+A^T & (A-A^T) \\ A^T-A & -(A+A^T) \end{pmatrix} \begin{pmatrix} x \\ x \end{pmatrix} = \begin{pmatrix} b \\ -b \end{pmatrix}$$

条件数不変

MINRES, SYMMLQ

荻田 NAS2003

JSIAM2003

A 移流拡散方程式の離散化

$Ax = b \longrightarrow$ BiCG

A′ 単純な対称化 & 右辺

例: $\begin{pmatrix} & A^T \\ A & \end{pmatrix} \begin{pmatrix} x \\ a \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \longrightarrow$ CG

---

persymmetric matrices

A 非対称 Toeplitz 行列

PA 対称化

PA BiCG

＋ BiCGStab

A QMR

CGS etc.

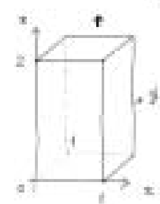
CG

対称化 PA は問題依存

渡部 JSIAM2003

テスト問題

$$-\mathrm{div}(k \nabla u) + v \frac{\partial u}{\partial x} = f$$



$f = 100$

$k = 1$

$v = \ldots$

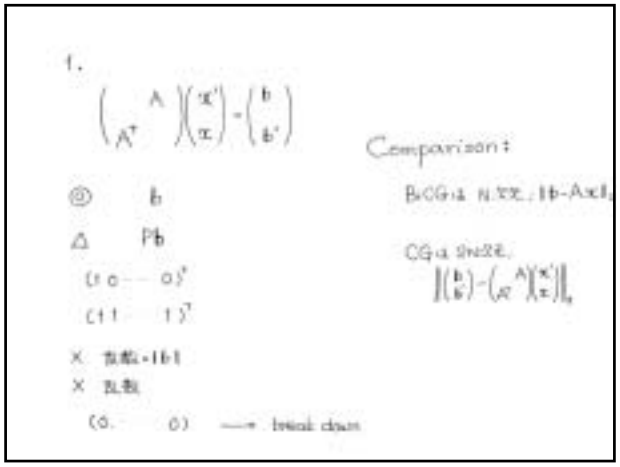
$x = 0$, $y = 0$, $z = 0$ } ディリクレ条件 $u = 0$

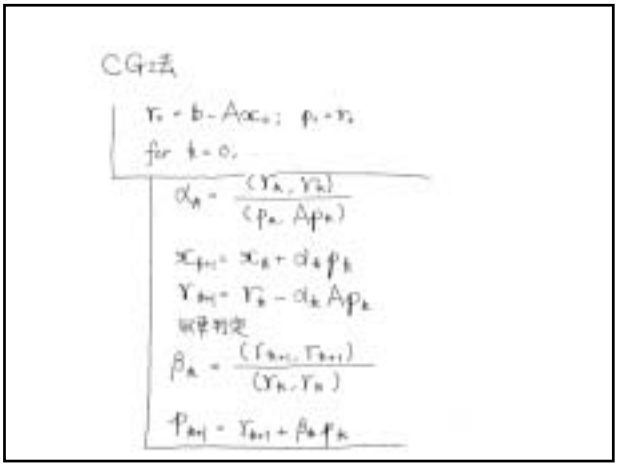
$x = 1$, $y = 1$, $z = 2$ } ノイマン条件 $\nabla u \cdot n = 0$

$5 \times 5 \times 10$ セルペクレ数 $\frac{v}{5}$

$\to N = 2000$

反復法 Toeplitz 1996





右辺とセルペクレ数への依存性

	0	1	2	4	8	16	32	64
BiCG	46	52	47	54	71	89	116	140
b	47	278	250	212	182	169	188	210
Pb	358	329	290	246	210	198	216	250
(1,0 …)	238	332	296	246	208	200	218	258
(1,1 …)	230	330	294	242	208	194	216	250
||b|| 乱数	398	382	332	276	236	216	238	262
乱数	500	496	410	324	252	242	268	318

非対称行列に対する CG 法アルゴリズム

初期ベクトル $\boldsymbol{x}_0$, $\boldsymbol{y}_0$ を用意

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$; $\boldsymbol{r'}_0 = \boldsymbol{b'} - A^T\boldsymbol{y}_0$

$\boldsymbol{p}_0 = \boldsymbol{r}_0$, $\boldsymbol{p'}_0 = \boldsymbol{r'}_0$

for $k = 0, 1, \cdots$

$$\alpha_k = \frac{[\,(\boldsymbol{r}_k,\boldsymbol{r}_k)+(\boldsymbol{r'}_k,\boldsymbol{r'}_k)\,]}{[\,(\boldsymbol{p}_k,A\boldsymbol{p'}_k)+(\boldsymbol{p'}_k,A^T\boldsymbol{p}_k)\,]} \quad \left\{ = \frac{\|\boldsymbol{r}_k\|^2+\|\boldsymbol{r'}_k\|^2}{2\,(\boldsymbol{p}_k,A\boldsymbol{p'}_k)} \right\}$$

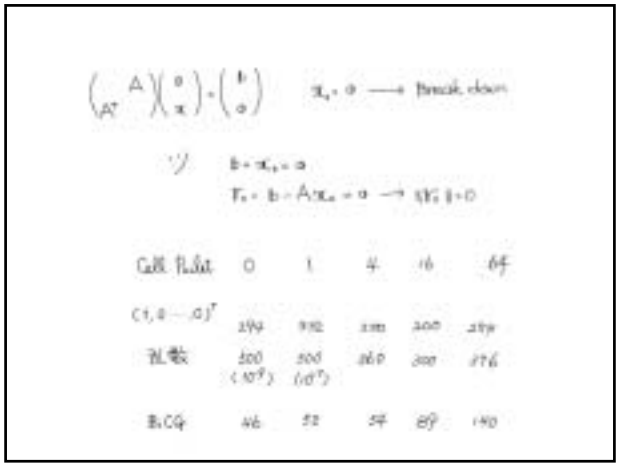
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p'}_k$ {; $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \alpha_k\boldsymbol{p}_k$}

$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p'}_k$; $\boldsymbol{r'}_{k+1} = \boldsymbol{r'}_k - \alpha_k A^T\boldsymbol{p}_k$

収束判定

$$\beta_k = \frac{[\,(\boldsymbol{r}_{k+1},\boldsymbol{r}_{k+1})+(\boldsymbol{r'}_{k+1},\boldsymbol{r'}_{k+1})\,]}{[\,(\boldsymbol{r}_k,\boldsymbol{r}_k)+(\boldsymbol{r'}_k,\boldsymbol{r'}_k)\,]}$$

$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$; $\boldsymbol{p'}_{k+1} = \boldsymbol{r'}_{k+1} + \beta_k\boldsymbol{p'}_k$



一般の BiCG 法アルゴリズム

初期ベクトル $\boldsymbol{x}_0$, $\tilde{\boldsymbol{x}}_0$ を用意

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$; $\tilde{\boldsymbol{r}}_0 = \tilde{\boldsymbol{b}} - A^T\tilde{\boldsymbol{x}}_0$

$\boldsymbol{p}_0 = \boldsymbol{r}_0$; $\tilde{\boldsymbol{p}}_0 = \tilde{\boldsymbol{r}}_0$

for $k = 0, 1, \cdots$

$$\alpha_k = \frac{(\boldsymbol{r}_k,\tilde{\boldsymbol{r}}_k)}{(A\boldsymbol{p}_k,\tilde{\boldsymbol{p}}_k)}$$

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$

$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$; $\tilde{\boldsymbol{r}}_{k+1} = \tilde{\boldsymbol{r}}_k - \alpha_k A^T\tilde{\boldsymbol{p}}_k$

収束判定

$$\beta_k = \frac{(\boldsymbol{r}_{k+1},\tilde{\boldsymbol{r}}_{k+1})}{(\boldsymbol{r}_k,\tilde{\boldsymbol{r}}_k)}$$

$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$; $\tilde{\boldsymbol{p}}_{k+1} = \tilde{\boldsymbol{r}}_{k+1} + \beta_k\tilde{\boldsymbol{p}}_k$

2. $\begin{pmatrix} & A^T \\ A & \end{pmatrix}\begin{pmatrix} x \\ x' \end{pmatrix} = \begin{pmatrix} b' \\ b \end{pmatrix}$

$\begin{pmatrix} & A \\ A^T & \end{pmatrix}$ のケースに方程式し

評価の難しさ

- 収束判定をどうするか？
- 問題 A への依存性
  (どんな場合に対称化すべきか)
- 初期値 x_0 への依存性
- どの方程式系が有効か

異なる対称化法

荻田の提案する方程式

$$\begin{pmatrix} A+A^T & A-A^T \\ -(A-A^T) & -(A+A^T) \end{pmatrix}\begin{pmatrix} x \\ x \end{pmatrix} = 2\begin{pmatrix} b \\ -b \end{pmatrix}$$

荻田の方法に対する初期値 x_0 依存性（反復回数）

	0(対称)	1	2	4	8	16	32	64
BiCG	46	52	47	54	71	89	116	140
$(1,0,\ldots,0)^T$	259	331	295	246	204	202	242	260
乱数	500	500	448	358	319	300	352	374

A に対する前処理は有効か

$$\begin{pmatrix} & A^T(LDU)^{-T} \\ (LDU)^{-1}A & \end{pmatrix}\begin{pmatrix} x \\ y' \end{pmatrix} = \begin{pmatrix} b' \\ (LDU)^{-1}b \end{pmatrix}$$

$$\begin{pmatrix} & (LDU)^{-1}A \\ A^T(LDU)^{-T} & \end{pmatrix}\begin{pmatrix} y' \\ x \end{pmatrix} = \begin{pmatrix} (LDU)^{-1}b \\ b' \end{pmatrix}$$

あるいは
全体系に対する前処理を考えるべきか？

現時点で言えること

- BiCG 法の限界では (M行列でなくなると)
  対称化して CG 法を使うとよさそう
- ちょっとした工夫で収束が大きく変わる
  (b, x_0 に対する解析が必要)
- 対称なら、条件数の概算が容易
- 桁数を増やすと収束が改善できそう
  (反復過程で大きな値がでてくる)

試してみる価値はある

- CG 法向きの問題は？
- より大規模な問題
- 前処理の可能性
- 正規方程式との比較
- もっと桁数を
- 特化したアルゴリズムの試作