数理解析研究所講究録 1362 巻 2004 年 38-46

拡張ストラッセン法の連立1次方程式への応用

早稲田大学　古井　充 (Mitsuru Furui) 鈴木　健二 (Kenji Suzuki)
WasedaUniversity
(株) 日立製作所　後　保範 (Yasunori Ushiro)
Hitachi.Ltd

1　はじめに

後が考案した拡張ストラッセン法[1]を、密行列 A を係数とする連立1次方程式

$$Ax = b$$

に適用する方法及び計算解の精度低下を防止する方法について述べる。密行列乗算 $C = A \cdot B$ にストラッセン法[2]を1回適用すると、A, B が実行列の場合、通常の計算方法と比較して演算量が 7/8 まで減少する。ストラッセン法では行列 A, B, C をそれぞれ 2×2 のブロック小行列に分割するのに対して、拡張ストラッセン法では、それぞれ $n \times 2$、$2 \times n$ および $n \times n$ のブロック小行列に分割する。n が大きいとき、拡張ストラッセン法は実行列乗算に1回適用すると、通常の計算方法に比較して演算量が 3/4 まで減少する。連立1次方程式の計算において、行列 A をブロック LU 分解するときに拡張ストラッセン法を適用することができる。ストラッセン法による行列乗算は、分割した小行列単位の要素の値に指数的大きさの差異があるときに通常の方法に比較して桁落ちの影響で精度低下が発生することが言われている。このため、ブロック LU 分解前にスケーリング処理を採用する。通常のブロック LU 分解法に比較して精度低下が防止できるかを評価するため、係数行列はブロック LU 分解のブロック単位に指数的に値の大きさが異なる行列で評価する。また計算速度についても評価を行う。拡張ストラッセン法を適用した場合にはメモリーアクセス回数が増加するため、ブロック LU 分解法に比べ計算時間が増加してしまうと考えられる。そこで計算時間を削減するための方法を考案し、ブロック LU 分解法との計算時間を比較し評価を行なう。

2　拡張ストラッセン法

図1のようにストラッセン法は $C=A \times B$ において、A,B,C をそれぞれ 2×2 のブロック小行列に分割して計算する。拡張ストラッセン法[1]とは、図2のように A,B,C をそれぞれ $n \times 2, 2 \times n, n \times n$ のブロック小行列に分割して、ストラッセン法を複数回利用し、計算する方法である。

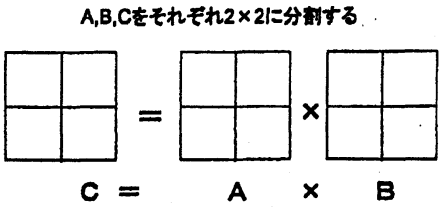

図1 ストラッセン法のイメージ

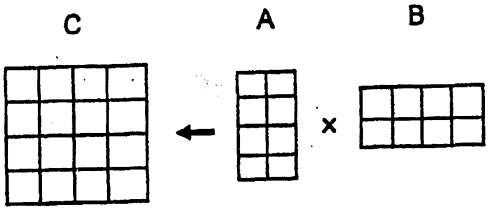

図2 拡張ストラッセン法のイメージ

拡張ストラッセン法を適用した際の例を以下の図3, 図4に示す。

◎	◎	◎	U
◎	L	U	○
◎	U	L	○
L	○	○	○

図3. n=4のときのCの計算式の配置

◎	◎	◎	×	U
◎	◎	◎	U	×
◎	L	U	○	○
◎	U	L	○	○
L	○	○	○	○

図4. n=5のときのCの計算式の配置

図中の $L,U,◎,○$ はそれぞれ C のブロック小行列を計算する際の、計算方法の種類を表してる。演算量は ○ と ◎ の部分を計算する際に小ブロック行列の乗算計算を1回削減することができ、$n=4$ のとき乗算の演算量は27/32に削減することができる。図4のようにまた分割数が奇数のときにはストラッセン法を利用できない箇所が生ずる。図の×の部分が通常計算と同様の計算が必要な箇所を表している。従って、$n=5$ のときは乗算の演算量は21/25に削減することができる。

一般に分割数を n とすると、拡張ストラッセン法と通常計算との演算量の比較は以下のようになっている。

表1. 拡張ストラッセン法と通常計算の計算回数比較

(単位は回)

乗算方式	計算種類	実数行列の計算回数	
		nが偶数	nが奇数
定義	乗算	$2n^2$	$2n^2$
方式	加減算	n^2	$2n^2$
拡張ストラッセン法	乗算	$3n^2+n-1$	$(3n^2-1)/2+n$
	加減算	$6(n-1)+2$	$6n(n-1)-1$

従って、拡張ストラッセン法による乗算の演算量削減の割合を λ とすると、$n \to \infty$ において、

$$\lambda = \frac{\frac{3n^2-1}{2}+n}{2n^2} \longmapsto \frac{3}{4}$$

となる。

3　拡張ストラッセン法の連立１次方程式への応用

3.1　ブロック *LU* 分解法

今回は連立１次方程式 $Ax = b$ をブロック *LU* 分解法で解く。ブロック *LU* 分解法とは、連立１次方程式 $Ax = b$ をＬＵ分解をして解く際に、１回のステップごとに、*LU* 分解法と同様のの計算をブロック幅 MB ごとにまとめて計算を行う計算方法である。ブロック *LU* 分解法では各ステップごとにおける *LU* 分解の消去計算[4]は、図５のような縦長なブロック行列と横長なブロック行列との行列乗算によって行なわれる。ブロック *LU* 分解法は、分解における軸交換は部分軸交換を採用し、消去計算前のブロック処理において行う。

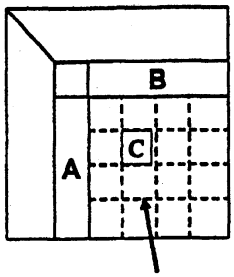

図5. ブロックLU分解法の適用イメージ

ブロック行列乗算において、キャッシュメモリに入るようにブロック行列を分割して計算を行うキャッシュブロック対策を施すことによって、ブロック *LU* 分解法は計算時間を短縮させることができる。このブロック *LU* 分解法にキャッシュブロック対策を施した方法をキャッシュブロック付きブロック *LU* 分解法と呼ぶことにする。

3.2　拡張ストラッセン法の連立１次方程式への応用

ブロック *LU* 分解法では、各ステップごとにおける *LU* 分解の消去計算の際に、縦長なブロック行列と横長なブロック行列との行列乗算が生じる。今回は図６のように、そのブロック行列に対して、それぞれ $n \times 2$,$2 \times n$ に分割して、拡張ストラッセン法を適用し乗算計算を行いブロック *LU* 分解を行なう。

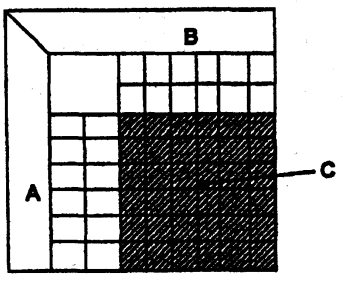

図6. 拡張ストラッセン法の適用イメージ

この方法を拡張ストラッセン法適用法と呼ぶことにする。

3.3 2段階拡張ストラッセン法

拡張ストラッセン法適用法では、拡張ストラッセン法を適用するために、小ブロック行列の乗算の演算量は削減されるが、メモリーアクセス回数が増えるため、ブロック LU 分解法比べ、計算時間が増加すると考えられる。そこで計算時間を短縮するために次のような方式を提案する。提案方式のイメージとして図7を示す。

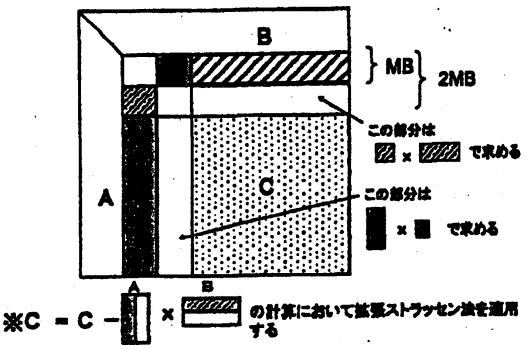

図7. 二段階提案方式のイメージ

拡張ストラッセン法では消去計算の前処理として縦長な行列 A, 横長な行列 B に対して1列または1行ずつ LU 分解を行なっている。それに対して、図7のように提案方式では、拡張ストラッセン法適用法の2倍のブロック幅 $2MB$ でブロック LU 分解を行う。前処理のブロック行列作成計算において、MB までは拡張ストラッセン法適用法と同様に行い、残りの MB をブロック行列乗算によって行う。これにより計算時間を短縮することができる。さらに消去計算においても計算をキャッシュメモリに入るように計算に使う行列サイズを分割して計算を行うようにする。この方法を2段階拡張ストラッセン法適用法と呼ぶことにする。

3.4 分割数、分割サイズの決定

ブロック幅 MB でブロック LU 分解を行なうとき、あるステップにおける消去計算に使用する縦長なブロック行列, 横長なブロック行列, 消去部分の正方行列を改めてそれぞれ、A,B,C とする。拡張ストラッセン法を適用する場合、各ステップごとの A の行の長さを nz とすると、A,B の行列サイズは $nz \times MB$,$MB \times nz$ である。このとき nz は各ステップごとに変化するため、A,B を分割する数 (分割数) と分割したときのブロック小行列のサイズ (分割サイズ) はともに一定にすることはできない。よって、各ステップごとに分割数、分割サイズを決定することにする。
分割の方法は、ストラッセン法の計算方法から考えて、A,B,C をそれぞれ $MB/2 \times MB/2$ の正方小行列に分割するのが望ましい。正方行列に分割できない場合には、それぞれを $MB/2 \times MB/2$ の正方小行列に最も近い大きさの長方形小行列に分割する。また各ステップごとの分割数は拡張ス

トラッセン法の性質から乗算計算をより多く削減するため、偶数になるように分割する。従って分割数は、偶数かつ分割された A,B,C が $MB/2 \times MB/2$ の正方小行列あるいは $MB/2 \times MB/2$ の正方小行列に最も近い大きさの長方形小行列となるように定める。分割サイズを ns とすると、ns は分割数によって決定することができ、A,B,C はそれぞれ、$ns \times MB/2$,$MB/2 \times ns$,$ns \times ns$ のブロック小行列に分割される。
分割サイズ ns で A,B,C を分割したときに、割り切れない端数部分が存在する場合がある。これに対して図 8 のように、乗算計算において行列サイズの調整を行なう。

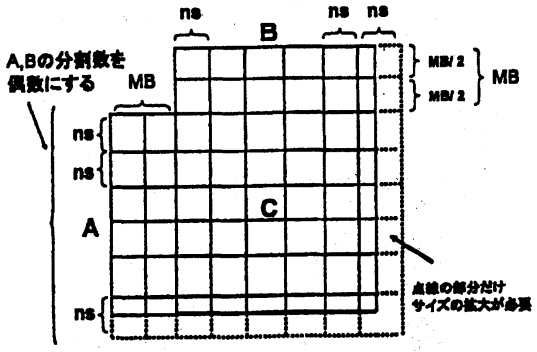

図8. A,Bの行列サイズの調整

端数部分が存在する場合には、A,B に対して、端数部分を含む A ならば $ns \times MB/2$,B ならば $MB/2 \times ns$ のブロック小行列ができるように、実際より大きな（図 8 の点線部分まで）行または列サイズを取って計算を行う。このとき分割数は端数部分を含むブロック小行列も入れて A,B が偶数個に分割するように調整を行う。この方法によって、実際に計算する行列のサイズは大きくなるので、計算に必要な乗算回数は増える。しかし、拡張ストラッセン法を適用することにより乗算回数を削減することができるので、全体として演算量を削減することができる。

3.5 スケーリング処理

ストラッセン法による行列乗算では、計算の対象が、要素の値の大きさが部分的に大きく異なっているような行列の場合には、桁落ちが発生し、精度低下が起こる。提案方式でも同様の問題が起こると考えられる。そこでこれを防ぐために、あらかじめ行列 A の各行に適当な値を乗じてデータの大きさを揃えるスケーリング処理を行う
$Ax = b$ に対して分解処理前の行列 A の各行 $i(1 \leq i \leq n)$ ごとに、A の各要素を a_{ij} とすると、

$$S_i(a_{i1} + a_{i2} + \ldots + a_{in}) = 1.0$$

となるような S_i を i 行の各要素 $a_{ij}(1 \leq j \leq n)$ および b_i に掛け合わせる。

4 数値実験の方法

今回の数値実験は、A,x,b の次元数を 2000 として、あらかじめ解を作成した連立 1 次方程式 $Ax = b$ に対して、以下の 4 種類の解法についてそれぞれ、スケーリング処理有り無しの 2 つの場合で解きその精度, 計算時間を比較する。ブロック LU 分解におけるブロック幅は 100 とする。

(1) ブロック LU 分解法 (BLU と呼ぶ)

(2) キャッシュブロック付きブロック LU 分解法 (CBLU と呼ぶ)

(3) 拡張ストラッセン法適用法 (ESTLU と呼ぶ)

(4) 2 段階拡張ストラッセン法適用法 (WESTLU と呼ぶ)

解 x は一様乱数によって作成する。次にテスト行列 A に対して、$Ax = b$ を計算して b を作成して、連立 1 次方程式を作成する。密行列 A は 5 種類用意してそれぞれに対して全ての解法で実験を行う。プログラムの実装は $FORTRAN$ 言語で行った。プログラムは可能な限り連続アドレスとして、ソースプログラム上でループアンローリングは行なわないものとした。
測定マシンは以下を使用し実験を行なった。

(1) Pentium3(866Mhz), 使用コンパイラ (Digital FORTRAN Ver5.OA)

(2) Pentium4(3.06Ghz), 使用コンパイラ (GNU Ver3.3.1)

(3) AMD64 Opteron (1.5Ghz), 使用コンパイラ (GNU Ver 3.2.2)

(4) Itanium2(1.5Ghz), 使用コンパイラ (Intel FORTRAN Compiler Ver8.0)

精度の算出方法は、あらかじめ一様乱数により発生させた誤差のない解を x_i とし、連立 1 次方程式を解き求めた解を x_i^* とすると、解の精度 ω は、

$$\omega = \frac{\sqrt{\sum_{i=1}^{n}(x_i^* - x_i)^2}}{\sqrt{\sum_{i=1}^{n} x_i^2}}$$

によって求めることにする。この ω を、$\log_{10}\frac{1}{\omega}$ とした値を有効桁数と呼ぶことにする.

4.1 テスト連立 1 次方程式の作成方法

テスト行列 A は以下の方法で作成することにする。

(1) 一様乱数により作成する。

(2) ブロック単位に要素の値が指数的に異なる行列
行列 A の要素を 0.0〜1.0 の乱数で作成する。次に A をブロック小行列に分割して、各ブロックごとに、$R = 0.0 \sim 1.0$ の乱数、P は定数として、
$\alpha = 10^{R \cdot P}$
を用意して、ブロック内の各要素に掛け合わせる。

(3) 行単位に要素の値が指数的に異なる行列

(2) 同様にして、行列Aを要素を0.0〜1.0の乱数で作成し、各行ごとに、$R = 0.0 \sim 1.0$の乱数、Pは定数として、

$\alpha = 10^{R \cdot P}$

を用意して、行の各要素に掛け合わせる。

作成方法(1) (2)の方法により、行列内の要素の値がブロック単位に異なる行列Aを作成することができ、(3)の方法により、行列内の要素の値が行単位に異なる行列Aを作成することができる。また$R = 0.0 \sim 1.0$の乱数であるから、

$$0.0 \leq \alpha \leq 10^P$$

となるので、行列内の要素の値は最大で10^P差が存在することになる。
今回の数値実験では、以下の5つの行列でテストを行なう。

(1) 一様乱数

(2) ブロック単位,P=5.0

(3) ブロック単位,P=10.0

(4) 行単位,P=5.0

(5) 行単位,P=10.0

5 実験結果と考察

数値実験の結果を以下に示す。それぞれのテスト行列Aについて、連立1次方程式$Ax = b$を4種類の方法で解く操作を*Pentium4*でそれぞれ10回ずつ実験を行ったときの解の精度の有効桁数の最大、最小、平均の桁数を測定した。その結果、どの解法においても平均桁数と最大、最小の桁数の差は1,2桁であった。次の表2, 表3はスケーリング処理を施したときとそうでないときの、それぞれんの解法の解の精度の平均の有効桁数を表している。

表2. 精度の測定結果(スケーリング処理なし)

(単位は平均の有効桁数)

解法 / テスト行列	ブロックLU分解		拡張ストラッセン法	
	BLU	CBLU	ESTLU	WESTLU
一様乱数	11.1	11.1	11.3	11.4
ブロック単位(5.0)	10.6	10.6	10.6	9.7
ブロック単位(10.0)	10.6	10.6	10.2	9.0
行単位(5.0)	10.6	10.6	8.5	9.6
行単位(10.0)	10.6	11.1	5.2	5.2

表3. 精度の測定結果(スケーリング処理あり)

(単位は平均の有効桁数)

解法 / テスト行列	ブロックLU分解		拡張ストラッセン法	
	BLU	CBLU	ESTLU	WESTLU
一様乱数	11.1	11.1	11.0	10.9
ブロック単位(5.0)	10.5	10.4	11.2	9.8
ブロック単位(10.0)	10.5	10.5	10.3	9.4
行単位(5.0)	11.3	11.3	11.1	11.2
行単位(10.0)	11.6	11.6	11.4	11.5

表2をみると、スケーリング処理を施さなくても、拡張ストラッセン法適用法および2段階拡張

ストラッセン法適用法は、一様乱数、ブロック単位に要素の値が指数的に異なる行列に対してはブロック LU 分解法と同様の精度を得ることができた。しかし行単位に要素の値が指数的に異なる行列に対してはブロック LU 分解法に比べ、精度が著しく低下してしまった。特に $P = 10.0$ では行列内の要素の値に最大で 10^{10} の差が生じるため、計算中に桁落ちが最も起こりやすいために、精度が低下したと考えられる。しかし次に表 3 をみると、スケーリング処理を施すことによって、拡張ストラッセン法適用法および 2 段階拡張ストラッセン法適用法はブロック LU 分解法と変わらない精度を得ることができた。また拡張ストラッセン法適用法と 2 段階拡張ストラッセン法適用法ではほぼ同等の精度を得ることができた。

表 4 はそれぞれの測定マシンで実験を行なったときの、計算時間の比較を示している。表の値は計算時間 (s) と性能 ($MFLOPS$) を表し、それぞれ 50 回ずつ実験を行なったときの結果の平均の値である。拡張ストラッセン法を使用した場合には、計算量は削減されているが、性能指標である $MFLOPS$ は次元数を n としたとき、

MFLOPS= $(2n^3 / 3 \times 10^{-6})$/計算時間 ($s$) で計算を行なった。表の解法では、計算時に全てスケーリング処理を施している。

表4. 計算時間の測定結果

(単位は測定時間:s,性能:MFLOPS)

解法の種類		ブロックLU分解				拡張ストラッセン法			
		BLU		CBLU		EST LU		WES TLU	
		測定時間	性能	測定時間	性能	測定時間	性能	測定時間	性能
測定マシン	Pentium3	66.6	80	22.2	241	21.6	247	22.5	237
	Pentium4	11.4	469	5.7	936	5.7	945	6.1	874
	AMD64	20.7	257	14.8	361	10.5	505	8.9	601
	Itanium2	4.2	1258	13.2	405	4.4	1220	3.0	1798

表 4 の結果を見ると、$Pentium3$,$Pentium4$,$AMD64$ の測定マシンでは、キャッシュブロック付きブロック LU 分解法の方がブロック LU 分解法より計算時間は短く、性能が 2 倍以上であった。反対に $Itanium2$ では、ブロック LU 分解法の方がキャッシュブロック付きブロック LU 分解法よりも計算時間が短く、性能は約 3 倍であった。

$Pentium3$、$Pentium4$ では、拡張ストラッセン法適用法の方が 2 段階拡張ストラッセン法適用法よりも測定時間が短かった。$Pentium3$ の場合には、拡張ストラッセン法適用法はキャッシュブロック付きブロック LU 分解法同等の性能を得ることができたが、$Pentium4$ の場合では、ブロック LU 分解法の方が性能が多少高かった。一方、$AMD64$,$Itanium2$ では、2 段階拡張ストラッセン法適用法の方が、拡張ストラッセン法適用法よりも測定時間が短く、どちらの場合でも 2 段階拡張ストラッセン法適用法はキャッシュブロック付きブロック LU 分解法及びブロック LU 分解法よりも高い性能を得ることができた。

とくに $Itanium2$ では、拡張ストラッセン法適用法と 2 段階拡張ストラッセン法適用法との性能の差が著しく、2 段階拡張ストラッセン法適用法は拡張ストラッセン法適用法の約 1.5 倍もの性能を得ることができた。

6 おわりに

今回の実験では、拡張ストラッセン法適用法および2段階拡張ストラッセン法適用法は、スケーリング処理を施すことによって、ブロック *LU* 分解法やキャッシュブロック付きブロック *LU* 分解法と同等の精度が得られることがわかった。次に性能に関しては、キャッシュブロック付きブロック *LU* 分解法に比べ拡張ストラッセン法適用法および2段階拡張ストラッセン法適用法は、演算量は削減するがメモリーアクセス回数が増加するため、実際の性能は、測定マシンによって優劣が決まると考えられる。そのため、今回の実験では拡張ストラッセン法適用法および2段階拡張ストラッセン法適用法はキャッシュブロック付きブロック *LU* 分解法と比較して、*Pentium*3 ではほぼ同等の性能であり、*Pentium*4 ではブロック *LU* 分解法に多少性能が劣化し、*AMD*64,*Itanium*2 では演算量削減以上の効果が得ることができた。
今回の実験では、測定マシンのキャッシュメモリーの大きさの大小を考慮せずにブロック幅を100に固定し、ループアンローリング等も実施しないプログラムで性能測定を行なった。そのため拡張ストラッセン法の利用の有無による性能の比較評価が不十分であった。今後は各マシンごとに十分にチューニングしたプログラムで性能の評価をする必要がある。

謝辞

本論文の性能及び精度取得のため、*Pentium*4,*AMD*64 及び *Itanium*2 の測定の支援をさせていただいた東京大学基盤情報センターの金田研究室並びに佐藤研究室の皆様、及び *Itanium*2 の測定環境をご提供いただいたインテル株式会社 (intel) 殿のそれぞれに、謹んで感謝の意を表する。

参考文献

1) 後保範:行列乗算におけるストラッセンの方法の拡張、京大数理研考究録、1040(1998)pp.61-69

2) V.Strassen : Gaussian Elimination Is Not Opitimal, NUMER, MATH, 13(1969), pp.354-356.

3) 小国力 著:行列計算ソフトウェア WS、スーパーコン、並列計算機、 丸善 (1991)

4) 洲ノ内 治男: 数値解析 (新訂版)、サイエンス社 (2002)