

自動チューニング機能付き並列数値計算ライブラリの開発 — PC クラスタにおける性能の観点から —

片桐 孝洋^{†, ††} 黒田 久泰^{††}
工藤 誠^{†††} 金田 康正^{††}

[†] 日本学術振興会特別研究員

^{††} 東京大学情報基盤センタースーパーコンピューティング研究部門

^{†††} 東京大学大学院情報理工学系研究科コンピュータ科学専攻

E-mail : katagiri@pi.cc.u-tokyo.ac.jp

1 はじめに

我々は科学技術計算用ライブラリ，特に並列数値計算において，以下に示す特徴・機能を有する数値計算ライブラリを開発を目指してきました．

- 利用者が指定するパラメタが少ないこと
- 演算カーネルに関する自動チューニング機能があること
- 通信処理に関する自動チューニング機能があること（並列計算機を用いる場合）
- 利用するアルゴリズムに関する自動選択機能があること

当初これらの特徴・機能は，利用者の観点から使いやすさを考慮して構築されたものでありましたが，性能の観点からも有用であることが判明してきました [1, 2]．これは，

- (1) 利用者が誤って設定した，性能に大きく影響するパラメタを自動修正できること
- (2) パラメタの最適化に関してその組合せ数が非常に多く最適化が困難である場合にも，自動的にほぼ最適化できること
- (3) 実行時にならないと設定が困難なパラメタ，すなわち事前の最適パラメタ設定が困難なパラメタを自動的に設定できること

の理由によるからです．

ところで現在，分散メモリ型並列計算機向きの自由入手可能なライブラリとしては ScaLAPACK[3] などが知られています．これら自由入手可能なライブラリの多くは，利用者が定義しなくてはならないパラメタが多く利用しにくいという欠点があります．こ

本稿は2001年7月25日～27日に沖縄で行われる，情報処理学会並列/分散/強調処理に関する『沖縄』サマー・ワークショップ SWoPP'2001 にて発表する「PC クラスタにおける並列数値計算ライブラリ ILIB の性能評価」(片桐孝洋，黒田久泰，工藤誠，金田康正)をもとに，加筆訂正，削除を行ったものです．

の一方で、数値計算において利用のし易さの向上を目指した研究として自動チューニングを行う研究やソフトウェア開発は米国を中心に行われています。たとえば PHiPAC[4], ATLAS[5], および FFTW[6] などです。残念なことに、これらのソフトウェアは並列処理の観点からはまだ不十分であると言わざるを得ません。なぜなら分散メモリ型並列計算機による並列計算を考慮に入れていなかったり、たとえ並列化がなされていたとしても通信処理の自動最適化機能がなかったりするのが現状だからです。また最適化の対象が行列積など線形代数基本演算 (BLAS) に限定されています。そこで我々は、数値計算ライブラリを自動的に最適化することで使いやすく性能的にも実用となる並列ライブラリ開発の方針をとってきました。

これらを背景として我々は、直接法 (LU 分解, 固有値計算における三重対角化) や反復法 (GMRES 法などの疎行列を係数行列とする連立 1 次方程式の解法) などに自動チューニング機能を付加した並列数値計算副プログラム集 ILIB (Intelligent LIBrary) を現在研究開発中です。これまでは多くの超並列型スーパーコンピュータで性能を発揮するコード—ベクトル計算機を計算要素 (PE) とした分散メモリ型並列計算機において高い性能を発揮できるコード—の開発を主におこなってきました。ところが近年、安価に構成できる PC クラスタを利用して数値計算を行う事例が多くなってきており、PC クラスタにおいても高性能を発揮するライブラリが開発が望まれるようになりました。そこで本稿において、分散メモリ型並列計算機用に開発した ILIB を PC クラスタ上で評価した結果を報告することにします。

2 現在の ILIB が提供する機能

2.1 連立 1 次方程式ライブラリ

現在開発中の並列連立 1 次方程式ライブラリは、式 (1) に示される連立 1 次方程式の解ベクトル x を求めることができます。

$$Ax = b \quad (1)$$

ここで係数行列 $A \in \mathbb{R}^{n \times n}$ は実数の非対称密行列もしくは非対称疎行列です。また右辺ベクトル b は $b \in \mathbb{R}^n$ であり、解ベクトル x は $x \in \mathbb{R}^n$ です。

式 (1) の解法として、直接解法と反復解法の 2 種が存在します。ILIB においては

- 直接解法：密行列の LU 分解に基づくルーチン (ILIB_LU[7, 8])
- 直接解法：各行で可変の帯幅をもつ疎行列用の LU 分解に基づくルーチン (ILIB_RLU[7, 8])
- 反復解法：疎行列反復法的一种である GMRES(m) 法に基づくルーチン (ILIB_GMRES[9, 10])
- 反復解法：疎行列反復法的一种である GCR(m) 法に基づくルーチン [11] (ILIB_GCR)

において既に自動チューニング機能を実装しています。なお本稿では紙面の都合から、ILIB_GMRES の性能についてのみ報告することにします。

2.2 固有値問題ライブラリ

現在開発中の並列固有値ライブラリは、式 (2) に示す標準固有値問題の対角化を行うことができます。

$$X^{-1}AX = \Lambda \quad (2)$$

ここで、係数行列 $A \in \mathbb{R}^{n \times n}$ は実数の対称密行列であり、固有値 λ_i ($i = 1, 2, \dots, n$) $\in \mathbb{R}$ から構成される行列を $\Lambda = \text{diag}(\lambda_i)$ 、固有ベクトル x_i ($i = 1, 2, \dots, n$) $\in \mathbb{R}^n$ から構成される行列を $X = (x_1, x_2, \dots, x_n)$ とします。式 (2) の対角化を行う解法として Householder-二分-逆反復法を用いています [12]。

一般化固有値問題 $Ax = \lambda Bx$ において行列 B が対称実数行列ならば、 B のコレスキー分解をするルーチンを実装することにより、式 (2) の対角化ルーチンを使用すれば、一般化固有値問題も求解可能となります。したがって本ルーチンはライブラリ機能の拡張に役立つ基本的なルーチンです。

さて今回自動チューニング機能を実装した部分は、

- 対称密行列の相似変換に基づいた三重対角化ルーチン
(ILIB_TriRed[13, 1])
- 逆反復法で利用可能な修正 Gram-Schmidt 直交化 (QR 分解) ルーチン
(ILIB_MGSAO[14])
- 三重対角行列の固有ベクトルを元の行列の固有ベクトルに変換する
Householder 逆変換ルーチン
(ILIB_HouseInv[14])

で、本稿ではこの 3 つのルーチンの性能について報告します。

3 自動チューニング機能の説明

3.1 連立 1 次方程式ライブラリ

疎行列反復解法ルーチンの場合には係数行列 A が疎であり、チューニングの要因は係数行列 A の非零要素の位置に依存します。このことは、ライブラリ実行時にしないとチューニングを行えないことを意味しています。結果として自動チューニングを動的 (ライブラリ実行時) に行うことになり、ILIB ではそのように自動チューニング機能を実現しています。

3.1.1 ILIB_GMRES の自動チューニング項目

自動チューニング項目は、以下に示す通りです。

- (i) 疎行列-ベクトル積におけるアンローリング段数
- (ii) 疎行列-ベクトル積における通信方式
- (iii) 前処理方式
- (iv) GMRES(m) 法における特有の処理

(i) では、行方向圧縮形式用の演算カーネルにおけるループ展開数を決めます。具体的には、反復中は非零要素の位置は変化しないので、ループアンローリングされたカーネルを複数用意しておき、反復処理に入る前に1度だけ全てのカーネルの速さを検査することで最適なカーネルを決定することで自動チューニング機能を実現しています。

(ii) では、1対1通信を用いて通信回数を最少にする実装方式か、同期的に全体全通信をする実装方式かを自動選択することで自動チューニング機能を実現しています。この自動チューニングも、反復処理に入る前に1度だけ行えばよいチューニングです。

(iii) では、収束の加速技法である前処理方式を自動的に決定します。最適な前処理方式は行列の性質や通信処理の性能で変わるので、一般的に自動決定が困難です。今回実装したチューニング手法では、複数用意した前処理方式をそれぞれ1回ずつ適用して、最も残差ベクトルを減少させた方式を自動選択するという方針を実装しています。

(iv) では、(i)-(iii) の他に反復解法に特化したパラメタを自動調整します。具体的に GMRES(m) 法では、

- (1) リスタート周期
- (2) 直交化の方式

などが全体の実行時間に大きく影響します。我々が実装した自動チューニング手法では、(1) では、黒田らが提案した 2,4,6,8 と m になるまで各反復で増加して m になると 2 に戻す方法を用いています [15]。この周期 m はメモリ量を考慮して決定される値です。(2) では、並列効率は高いが精度は悪い方法 (Classical Gram-Schmidt 法, 以降 CGS と表記します)、CGS よりも数倍の実行時間は増加するが精度が良い方法 (改良された Classical Gram-Schmidt 法, 以降 IRCGS (Iterative Refinement CGS) と表記します) [16]、および並列効率は低いが精度は高い方法 (Modified Gram-Schmidt 法, 以降 MGS と表記します) の 3 種を実装しています。我々の適用方針は、まず反復処理に入る前に CGS と MGS の実行時間を調べ速い方を選択します。反復を開始した後、一回反復時の誤差の減少がある値 (ここでは 0.5%) 以下になった場合、IRCGS を強制的に選択します。なお残念なことに、これらの自動チューニング手法が最適であり、必ず収束するという理論的な保証は無いことに注意してください。

3.2 固有値問題ライブラリ

本稿で示す固有値問題ライブラリの各ルーチンは、全て反復解法ではありません。したがって、これらの自動チューニングは静的 (ライブラリインストール時、もしくは問題サ

イズを固定した計算の直前)に1度行えばよいこととなります*。現在のILIBの固有値計算ライブラリでは、そのように自動チューニング機能を実現しています。

3.2.1 ILIB_TriRedの自動チューニング項目

このルーチンはキャッシュ上のデータを再利用するブロック化技法を実装していません。この演算カーネルはBLAS2演算という種類の演算—すなわち行列-ベクトル積システムの演算—になります。このルーチンの k 反復時における演算処理として、

- (i) 行列-ベクトル積： $A^{(k)}u_k$ におけるループアンローリング段数
- (ii) 行列更新： $A^{(k+1)} = A^{(k)} - u_k(x_k^T - \mu u_k^T) - x_k u_k^T$, $k = 1, 2, \dots, n-2$
(ここで $u_k, x_k \in \mathbb{R}^{n-k+1}$, $\mu \in \mathbb{R}$)におけるループアンローリング段数
- (iii) (i)の行列-ベクトル積を実行するために必要なベクトルのリダクション演算の実装方式

を自動選択することで、自動チューニング機能を実現しています。

3.2.2 ILIB_MGSAOの自動チューニング項目

このルーチンはブロック化をしています。演算カーネルはBLAS3演算という種類の演算—すなわち行列-行列積のシステムの演算—となります。以下の各項目、

- (i) 枢軸PE(計算を進めるために必要な情報を所有したPE)用の演算カーネル：

$$u_1^{(j)} = u_1^{(j)} - (u_1^{(i)T} \cdot u_1^{(j)})u_1^{(i)}$$

$$j = k, k+1, \dots, n, i = k, k+1, \dots, j+1, k = 1, 2, \dots, n/\text{PE数}$$
 のループアンローリング段数
- (ii) それ以外のPE用の演算カーネル(枢軸PE用と類似)におけるループアンローリング段数
- (iii) ブロック化したルーチンにおける参照するデータ領域を調整するためのパラメタであるブロック幅 BL の値

を自動決定することで自動チューニング機能を実現しています。なお(iii)は、 BL ごとに演算して放送する実装になっているので、通信処理に影響を及ぼすパラメタとなっています。

3.2.3 ILIB_HouseInvの自動チューニング項目

このルーチンはブロック化していません。演算カーネルはBLAS1演算という種類の演算—すなわちベクトル演算のシステムの演算—となっています。自動チューニング機能は、

* ただし逆反復法ルーチン(固有ベクトル計算時)などを最適化する場合は、動的チューニングが不可欠です。

- (i) 演算カーネル: $u_k^{(i)} = u_k^{(i)} - \mu u_k^{(j)}$, $k = 1, 2, \dots, n-2$, $i = 1, 2, \dots, n/\text{PE数}$, $j = 1, 2, \dots, n$
のループアンローリング段数

を自動調整することで実現します。

4 性能評価

この章では、本稿で示した ILIB ルーチンを PC クラスタで評価します。PC クラスタのノードとして、1 ノード Intel PentiumIII 800MHz Dual を 4 ノード用いました。1 ノード当たりの搭載メモリは 512MB (SDRAM, ECC PC100 CL=2)、ネットワークカードは PCI 10/100Mbps Intel EtherExpressPro100+、マザーボードは Intel L440GX+ Server Board (FSB 100MHz) です。使用 OS は Linux 2.2.13-33smp、コンパイラとして PGI 社の Fortran90 コンパイラ 3.2-3、オプションとしては `-O0†` および `-fast‡` を指定しました。また通信ライブラリとして、双方とも MPI (Message Passing Interface) を利用しました。

性能評価に用いる PC クラスタは、東京大学情報基盤センターが提供している HITACHI SR8000 と同様に、共有メモリ型の並列計算機をノードにもつ分散メモリ型並列計算機であるとみなせます。ところが各ノードには SR8000 のような疑似ベクトル化機能がないので、キャッシュサイズを越えるデータのアクセス時間は、キャッシュ上にあるデータに対するアクセス時間に比べ非常に遅くなります。

4.1 連立 1 次方程式

4.1.1 疎行列反復解法ルーチン ILIB_GMRES

自動チューニングのパラメタは、

- 行列-ベクトル積の種類 = { プリフェッチなし, プリフェッチあり }
- 行列-ベクトル積のアンローリング (行方向) 段数 = { なし, 2 段, 3 段, \dots , 9 段 }
- 行列-ベクトル積のアンローリング (列方向) 段数 = { なし, 2 段, 3 段, 4 段, 8 段 }
- 行列-ベクトル積の通信方式 = { MPI_ALLGATHER, 1PE 集約 \rightarrow MPI_BCAST, MPI_ISEND \rightarrow MPI_IRECV, MPI_IRECV \rightarrow MPI_ISEND, MPI_SEND \rightarrow MPI_RECV }
- 直交化方式 = { CGS, IRCGS, MGS }
- 前処理方式 = { なし, 行列多項式前処理 [15], ブロック不完全 LU 分解前処理 [15] }

の 6 種類です。ここで“プリフェッチあり”とは、行列-ベクトル積のコードにおいてループ中の依存関係を、フロー依存からループ伝搬フロー依存に変更しパイプライン処理向きにしたコード [9] のことです。また MPI_ALLGATHER とは、MPI における全対全の同期ベ

[†] このコンパイラオプションでは各 statement の基本ブロックを生成しますが、各 statement 間のスケジューリングや大域最適化を行いません。

[‡] このコンパイラオプションでは対象となるプラットフォームにおいて、一般的に最適なオプションを選びます。IA-32 のプラットフォームでは“-O2 -Munroll -Mnoframe”と等価となります。

クトル収集関数，MPI_BCAST は 1 対全同期放送関数，MPI_SEND は 1 対 1 同期送信関数，MPI_RECV は 1 対 1 同期受信関数，MPI_ISEND は 1 対 1 非同期送信関数，そして MPI_IRECV は 1 対 1 非同期受信関数を意味しています。

今回の数値実験では，1 行当たりの非零要素個数の最大値が 7 となる，以下に示す問題で性能を測定しました。

- 領域 $\Omega = [0, 1] \times [0, 1] \times [0, 1]$ における楕円型偏微分方程式の境界値問題

$$\begin{aligned} -u_{xx} - u_{yy} - u_{zz} + Ru_x &= g(x, y, z) \\ u(x, y)|_{\partial\Omega} &= 0.0 \end{aligned}$$

右辺は厳密解が $u = e^{xyz} \sin(\pi x) \sin(\pi y) \sin(\pi z)$ となるように決めました。領域を $80 \times 80 \times 80$ のメッシュで 7 点中心差分によって離散化しました。得られた連立 1 次方程式の次元は 512,000 です。R は， $R = 1.0$ で固定しました。

表 1: ILIB_GMRES における PC クラスタ上での自動チューニングの効果

(a-1) GMRES(m) 法 ILIB_GMRES [秒] (コンパイラオプション -fast)

実行時間には自動チューニングの時間も含まれている。

パラメタ 1 はアンローリングを行わないようにしたもの。

パラメタ 2 は通信方式に MPI_Allgather を利用したもの。

パラメタ 3 は直交化で MGS を利用したもの。

PE	1 (1 node)	2 (1 node)	2 (2 node)	4 (2 node)	4 (4 node)	8 (4 node)
実行時間 (1)	664.2	645.1	363.6	332.3	195.6	197.2
反復回数	1049	1024	1024	1004	1004	1004
アンローリング	N(1,7)	P(3,7)	N(1,7)	P(3,7)	N(1,7)	P(4,7)
通信方式	—	Isend	Irecv	Send	Irecv	Send
直交化	CGS	CGS	CGS	CGS	CGS	CGS
前処理	None	None	None	None	None	None
パラメタ 1 の実行時間 (2)	757.0	640.8	377.5	343.5	201.8	205.6
速度向上比 (2)/(1)	1.14	0.99	1.04	1.03	1.03	1.04
パラメタ 2 の実行時間 (3)	—	759.9	744.2	806.8	908.8	1452.7
速度向上比 (3)/(1)	—	1.18	2.05	2.43	4.65	7.37
パラメタ 3 の実行時間 (4)	686.8	647.7	357.7	351.1	210.8	219.7
速度向上比 (4)/(1)	1.03	1.00	0.95	1.06	1.08	1.11

(a-2) GMRES(m) 法 ILIB_GMRES [秒] (コンパイラオプション -O0)

PE	1 (1 node)	2 (1 node)	2 (2 node)	4 (2 node)	4 (4 node)	8 (4 node)
実行時間 (1)	981.7	684.7	500.2	379.3	277.3	222.5
反復回数	1051	991	991	1038	1038	1008
アンローリング	N(8,7)	N(8,7)	N(8,7)	N(8,7)	N(8,7)	N(8,7)
通信方式	—	Irecv	Send	Send	Isend	Send
直交化	MGS	CGS	CGS	CGS	CGS	CGS
前処理	None	None	None	None	None	None
パラメタ 1 の実行時間 (2)	1125.5	747.2	576.9	400.0	309.2	232.2
速度向上比 (2)/(1)	1.15	1.09	1.15	1.05	1.12	1.04
パラメタ 2 の実行時間 (3)	—	792.1	869.7	874.2	1049.3	1476.1
速度向上比 (3)/(1)	—	1.16	1.74	2.30	3.78	6.63
パラメタ 3 の実行時間 (4)	—	716.7	515.4	379.6	285.9	248.1
速度向上比 (4)/(1)	—	1.05	1.03	1.00	1.03	1.12

4.1.2 結果

実行時間 (単位は秒) 及び自動選択された方式を表 1 (a-1), (a-2) に示します。用語の説明は次の通りです。

アンローリング：例えば $P(2,3)$ はプリフェッチありで行列の列方向 2 展開，行方向 3 展開するという意味です． $N(2,3)$ はプリフェッチなしで同様の展開をすること意味します．

通信方式：Send は $\text{MPI_Send} \rightarrow \text{MPI_Recv}$ の順に使う，Isend は $\text{MPI_Isend} \rightarrow \text{MPI_Irecv}$ の順に使う[§]，Irecv は $\text{MPI_Irecv} \rightarrow \text{MPI_Isend}$ の順に使うことを意味します．

表 1(a-1)，(a-2) から，実行 PE 数の違いやコンパイラオプションの違いで選択されるパラメタが変わることがわかります．コンパイラオプションを `-fast` にしたとき，1 ノード内に 2 つのプロセスを起動すると，プリフェッチ型のコードが選択されています．これは同一ノード内でのメモリアクセスの衝突を避ける方が有効となるためと思われる．`-O0` の場合には，最もアンローリング段数の多いルーチンが選択されています．パラメタ 1-パラメタ 3 のように本来最適でないパラメタを指定すると，自動チューニングした場合に比べ実行時間に差が出ます．特に実行 PE 数が多くなるとそれが顕著になります．

4.2 密行列固有値問題ライブラリ

4.2.1 三重対角化ルーチン ILIB_TriRed

自動チューニングのパラメタは，

(I) 行列-ベクトル積のアンローリング (TMatVec) = { なし，2 段，3 段， \dots ，6 段，8 段，16 段 }；

(II) 行列更新のアンローリング (TUpdate) = { なし，2 段，3 段， \dots ，6 段，8 段，16 段 }；

(III) 通信方式 (Tcomm) = { Tree，MPI }；

の合計 3 種類です．(I)，(II) の処理は BLAS2 演算です．ここで (III) の “Tree” とは，1 対 1 通信を用いた二分木通信形態でベクトルリダクションを実現する方式であり，“MPI” とは MPI の関数 MPI_ALLREDUCE を利用する方式です．省略時パラメタは (TMatVec ， TUpdate ， Tcomm) = (8, 6, Tree) です．

4.2.2 MGS 直交化ルーチン ILIB_MGSAO

自動チューニングのパラメタは，

(I-1) 枢軸ブロック更新処理の最外ループのアンローリング (MBOuter)
= { なし，2 段，3 段，4 段 }；

(I-2) 枢軸ブロック更新処理の第 2 ループのアンローリング (MBSecond)
= { なし，2 段，3 段， \dots ，6 段，8 段，16 段 }；

[§] 非同期通信時に受信を先行発行すると，対応する送信からの通信要求の受信までの待ち時間が削減され，高速化される場合があります．また実装方式によっては，受信側に問い合わせることなく送信を行える場合があります [17, 18]．したがって，この実装方式により通信時間の高速化が期待できる場合は多いと思われる．

(II-1) それ以外の更新処理の最外ループのアンローリング (MOOuter)
 = { なし , 2 段, 3 段, 4 段 } ;

(II-2) それ以外の更新処理の第 2 ループのアンローリング (MOSecond)
 = { なし , 2 段, 3 段, ..., 6 段, 8 段, 16 段 } ;

(III) ブロック幅 (MBlklen) = { 1 , 2, ..., 6, 8, 16 } ;

の合計 5 種類です . (I-1)–(II-2) の処理は BLAS3 演算です . 省略時パラメタは (MBOuter, MBSecond, MOOuter, MOSecond, MBlklen) = (4, 8, 4, 8, 4) です .

4.2.3 Householder 逆変換ルーチン ILIB_HouseInv

自動チューニングパラメタは ,

(I) 更新処理のアンローリング (HITkernel) = { なし , 2 段, 3 段, ..., 6 段, 8 段, 16 段 } ;

の 1 種類です . この更新処理は BLAS1 ルーチンですが , 変換するベクトルの個数分 n だけ演算されるので二重ループとなります . ここでは , 外側のループのアンローリングをします . 省略時パラメタは (HITkernel) = (なし) です .

表 2: ILIB_TriRed における PC クラスタ上での自動チューニングの効果

(b-1) 三重対角化 ILIB_TriRed [秒] (コンパイラオプション -fast)

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.284	0.593	0.951	1.38
あり	0.282	0.592	0.950	1.38
(パラメタ)	(なし,16,Tree)	(8,5,Tree)	(5,16,Tree)	(16,8,Tree)
速度向上	1.007 倍	1.001 倍	1.001 倍	1.000 倍

チューニング	1,000 次元	2,000 次元	3,000 次元	4,000 次元
なし	8.14	44.8	136	304
あり	8.07	44.4	134	298
(パラメタ)	(8,8,Tree)	(8,16,Tree)	(8,16,Tree)	(6,16,Tree)
速度向上	1.008 倍	1.009 倍	1.014 倍	1.020 倍

(b-2) 三重対角化 ILIB_TriRed [秒] (コンパイラオプション -O0)

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.286	0.619	0.997	1.49
あり	0.285	0.606	0.999	1.48
(パラメタ)	(6,16,Tree)	(5,4,Tree)	(3,4,Tree)	(6,5,Tree)
速度向上	1.003 倍	1.021 倍	0.997 倍	1.006 倍

チューニング	1,000 次元	2,000 次元	3,000 次元	4,000 次元
なし	9.26	52.4	159	363
あり	9.26	51.7	157	357
(パラメタ)	(8,16,Tree)	(16,4,Tree)	(16,4,MPI)	(16,4,Tree)
速度向上	1.000 倍	1.013 倍	1.012 倍	1.016 倍

表 3: ILIB_MGSAO における PC クラスタ上での自動チューニングの効果

(c-1) MGS 直交化 ILIB_MGSAO [秒] (コンパイラオプション -fast)
 パラメタ中の“N”の表記は，“なし”を意味している．

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.084	0.592	1.97	4.67
あり	0.081	0.200	0.253	0.463
(パラメタ)	(3,4,3,N,4)	(8,2,3,N,4)	(8,N,16,2,2)	(8,3,8,2,2)
速度向上	1.0 倍	2.9 倍	7.7 倍	10.1 倍
チューニング	500 次元	600 次元	1,000 次元	2,000 次元
なし	9.20	15.9	73.8	591
あり	6.46	8.12	5.50	24.8
(パラメタ)	(8,3,8,N,2)	(8,3,N,N,2)	(8,2,N,N,4)	(8,4,3,2,8)
速度向上	1.4 倍	1.9 倍	13.4 倍	23.8 倍

(c-2) MGS 直交化 ILIB_MGSAO [秒] (コンパイラオプション -O0)

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.086	0.613	2.03	4.84
あり	0.036	0.135	0.274	0.544
(パラメタ)	(3,3,2,N,3)	(8,2,3,N,4)	(8,2,5,2,2)	(8,2,4,2,2)
速度向上	2.6 倍	4.5 倍	7.4 倍	8.8 倍
チューニング	500 次元	600 次元	1,000 次元	2,000 次元
なし	9.57	16.5	76.7	616
あり	4.99	6.67	6.55	37.27
(パラメタ)	(8,3,3,N,2)	(8,N,4,N,2)	(8,2,2,N,4)	(8,N,2,2,8)
速度向上	1.9 倍	2.4 倍	11.7 倍	16.5 倍

4.2.4 結果

各ルーチンにおける処理の違い，およびコンパイラオプションにより最適化されたパラメタの値や，省略時パラメタ時の実行時間に対する速度向上の効果を表 2-表 4 に示します．表 3(c-1)，(c-2) から，MGS 直交化ルーチンにおいては 1.4-23.8 倍と高速化され自動チューニングの効果があることがわかります．ところが，表 2(b-1)，(b-2) から，三重対角化ルーチンでは高々 1.02 倍しか高速化されません．これらの結果からブロック化してキャッシュ上のデータを最利用する実装方式が，PC クラスタでは特に重要であるといえるでしょう[¶]．

表 4(d-1) の Householder 逆変換では，コンパイラオプションが -fast のとき，自動チューニングの効果がほとんどありません．ところが表 4(d-2) のようにコンパイラの最適化能力を下げると，自動チューニングの効果がでてきます．すなわちループアンローリングしたコードの方が高速になるわけです．また表 3(c-1)，(c-2) の 100，200，500 および 600 次元における場合のように，コンパイラの最適化能力を下げた自動チューニングの方が

[¶] 三重対角化ルーチン中の BLAS2 演算でも，ブロック化した実装は可能です．しかし現在の ILIB では，ベクトル化を考慮する理由 — すなわち最内のループ長をできるだけ長くする理由 — から，最内ループのループアンローリングは行っていません．

表 4: ILIB_HouseInv における PC クラスタ上での自動チューニングの効果

(d-1) Householder 逆変換 ILIB_HouseInv [秒] (コンパイラオプション -fast)

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.019	0.006	0.020	0.045
あり	0.001	0.006	0.020	0.045
(パラメタ)	(16)	(なし)	(なし)	(なし)
速度向上	19.0 倍	1.00 倍	1.00 倍	1.00 倍

チューニング	1,000 次元	2,000 次元	3,000 次元	4,000 次元
なし	3.89	35.7	125	305
あり	3.89	35.7	117	305
(パラメタ)	(なし)	(なし)	(2)	(なし)
速度向上	1.00 倍	1.00 倍	1.06 倍	1.00 倍

(d-2) Householder 逆変換 ILIB_HouseInv [秒] (コンパイラオプション -O0)

チューニング	100 次元	200 次元	300 次元	400 次元
なし	0.020	0.021	0.070	0.203
あり	0.003	0.017	0.058	0.151
(パラメタ)	(6)	(8)	(16)	(16)
速度向上	6.66 倍	1.23 倍	1.20 倍	1.34 倍

チューニング	1,000 次元	2,000 次元	3,000 次元	4,000 次元
なし	5.91	48.7	165	392
あり	4.89	43.7	140	349
(パラメタ)	(4)	(2)	(2)	(2)
速度向上	1.20 倍	1.11 倍	1.17 倍	1.12 倍

高速であるという結果は衝撃的です。

表 4(d-1), (d-2) のように, 多くの場合はコンパイラの最適化能力が自動チューニングの効果に大きく影響するはずですが, 一般的にプログラムの最適化をコンパイラ任せにする書き方では, 計算自体が本来持っている性能を引き出すことは容易ではありません。したがってライブラリ開発者側がコンパイラの最適化を補助する書き方をしたプログラムを多数用意するという ILIB のアプローチは, 計算機ごとにコンパイラの最適化能力が異なる環境下でいかにより高速なライブラリを構築するのかという問題を解決する一つの解になると言えます。

5 おわりに

本稿ではスーパーコンピュータ用に開発された自動チューニング機能付き並列数値計算ライブラリ ILIB における, PC クラスタ上での自動チューニング機能の効果について述べました。実行前チューニングが適用できる固有値問題における各ルーチンにおいて, ブロック化による実装を施したルーチンの速度向上が 23 倍にもおよぶ例があり, パラメタの自動調整機能が PC クラスタでは特に重要であることが示されました。今後 ILIB に

おける各演算（疎行列および密行列に対する BLAS2 演算など）をブロック化して実装し、そのパラメタも自動的に調整する機能を付加する予定です。

今後ブロック化に関するパラメタも性能パラメタとして付加すると、最適パラメタに関する探索時間の削減手法が実行時自動チューニングでは特に重要となります。ループアンローリングなどを手で実装するのは手間がかかるばかりか、バグの原因となります。したがって自動でループアンローリングを行い自動チューニング機能を有する言語、もしくは自動チューニングに関するコードを自動生成できるトランスレータなどの開発をする必要があるでしょう。

なお本稿で示した ILIB プロジェクトの情報や ILIB の各種ルーチンは、東京大学情報基盤センタースーパーコンピューティング研究部門金田研究室で行われている HINTS プロジェクトのホームページ <http://www.hints.org/> で公開される予定です。

謝辞

日頃討論頂く、東京大学情報基盤センタースーパーコンピューティング研究部門、金田・佐藤研究室の諸氏に感謝致します。なお本研究の一部は平成 13 年度科学研究費補助金（特別研究員奨励費および奨励研究（A））の支援により行いました。

参考文献

- [1] 片桐孝洋, 黒田久泰, 大澤清, 金田康正: ILIB: 自動チューニング機能付き並列数値計算ライブラリとその性能評価, JSPP'2000 論文集, pp. 27-34 (2000).
- [2] 片桐孝洋, 黒田久泰, 工藤誠, 金田康正: スーパーコンピュータおよび PC クラスタにおける自動チューニング機能付き並列固有値ソルバの性能評価, JSPP'2001 論文集, pp. 73-74 (2001).
- [3] Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. and Whaley, R.: *ScaLAPACK Users' Guide*, SIAM (1997).
- [4] Bilmes, J., Asanović, K., Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, *Proceedings of International Conference on Supercomputing 97*, Vienna, Austria, pp. 340-347 (1997).
- [5] Whaley, R. C., Petitet, A. and Dongarra, J. J.: Automated Empirical Optimizations of Software and the ATLAS Project, *Parallel Computing*, Vol. 27, pp. 3-35 (2001).
- [6] Frigo, M.: A Fast Fourier Transform Compiler, *Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, Georgia, pp. 169-180 (1999).

- [7] 大澤清, 片桐孝洋, 黒田久泰, 金田康正: ILIB_RLU : 疎行列を密行列として扱う自動チューニング機能付き LU 分解ルーチンの性能評価, *IPSJ SIG Notes, 00-HPC-82*, pp. 25–30 (2000).
- [8] 大澤清: 自動チューニング機能付きスパースダイレクトソルバ ILIB_RLU の性能評価, *スーパーコンピューティングニュース*, Vol. 2, No. 5, pp. 23–36 (2000). 東京大学情報基盤センター (スーパーコンピューティング部門).
- [9] Kuroda, H., Katagiri, T. and Kanada, Y.: Performance of Automatically Tuned Parallel GMRES(m) Method on Distributed Memory Machines, *Proceedings of Vector and Parallel Processing (VECPAR) 2000*, Porto, Portugal, pp. 251 – 264 (2000).
- [10] 黒田久泰, 片桐孝洋, 金田康正: 異機種並列環境における連立一次方程式ライブラリの性能評価, *IPSJ SIG Notes, 00-HPC-82*, pp. 35–40 (2000).
- [11] 工藤誠, 黒田久泰, 片桐孝洋, 金田康正: メモリ使用量の少ない一般共役残差法の提案, *IPSJ SIG Notes, 2001-HPC-85*, pp. 79–84 (2000).
- [12] 片桐孝洋, 金田康正: 並列固有値ソルバーの実現とその性能, *IPSJ SIG Notes, 97-HPC-69*, pp. 49–54 (1997).
- [13] Katagiri, T., Kuroda, H. and Kanada, Y.: A Methodology for Automatically Tuned Parallel Tri-diagonalization on Distributed Memory Parallel Machines, *Proceedings of Vector and Parallel Processing (VECPAR) 2000*, Porto, Portugal, pp. 265 – 277 (2000).
- [14] Katagiri, T.: *A Study on Large Scale Eigensolvers for Distributed Memory Parallel Machines*, Ph.D Thesis, The University of Tokyo (2001).
- [15] Kuroda, H. and Kanada, Y.: Performance of Automatically Tuned Parallel Sparse Linear Equations Solver, *IPSJ SIG Notes, 99-HPC-76*, pp. 13–18 (1999). in Japanese.
- [16] Balay, S., Gropp, W., McInnes, L. and Smith, B.: *PETSc 2.0 Users Manual*, ANL-95/11 – Revision 2.0.24, Argonne National Laboratory (1999).
- [17] 建部修見, 児玉祐悦, 関口智嗣, 山口喜教: リモートメモリ書き込みを用いた MPI の効率的実装, *情報処理学会論文誌*, Vol. 40, No. 5, pp. 2246–2255 (1999).
- [18] 森本健司, 松本尚, 平木敬: メモリベース通信を用いた高速 MPI の実装と評価, *情報処理学会論文誌*, Vol. 40, No. 5, pp. 2256–2268 (1999).