2004－HPC－99　(17)
2004／7／31

徒競走型の性能保証レベル向上方法の検討

直野健[1)]，　猪貝光祥[2)]，木立啓之[2)]
1)　（株）日立製作所中央研究所　　2)　（株）日立超LSIシステムズ

行列計算処理に関して、効果的なプログラム性能チューニング方法が定まらない場合に、複数のチューニングパタンでプログラムを同時に処理し、最速の結果を採用する徒競走型の性能保証レベル向上方法を提案した。GRID コンピューティング技術の進展を背景に、複数プラットフォームでの高性能処理の実現を目指す自動性能チューニング型行列ライブラリ技術が注目されているが、提案方法は、同技術によっても解決が困難な性能不安定性の問題を解決できる。5 種類の基本行列演算を対象に、PC 上で提案方法を試行したところ、所定の行列サイズの区間におけるプログラムの最低性能値が最大で約 8.6%向上し、提案方法の有効性を確認できた。

A Race-based method for improving performance assurance level for matrix computation

Ken Naono[1)], Mitsuyoshi Igai[2)] and Hiroyuki Kidachi[2)]
1)　Central Research Laboratory, Hitachi, Ltd.　2) Hitachi ULSI Systems Corporation

We propose a race-based method for improving performance assurance level for matrix computation. The method executes multiple sets of matrix computation with the different tuning parameters to alleviate the performance degradation of the computation with a single tuning parameter. Along with the development of Grid computing technologies, automatically tuned matrix libraries, which enable high performance tuning on several computing platforms, are recently attracting much attention. However, the performance with the existing automatic tuning sometimes degrades dramatically in some cases to cause performance instability problems. Our race-based method is the solutions to such performance instability problems. The experiments of the method for 5 BLAS (Basic Linear Algebra Subprograms) codes on PC (Pentium4, 3.2GHz) show that, in the best case, a 4 member race achieves about 8.6% better in the sense of the lowest-performance in a certain interval.

1.　はじめに

近年、スーパーコンピューティングの利用環境に関するパラダイム・シフトが起こりつつある。従来は、特定の計算センタにあるスーパーコン単体を利用する形態が主流であった。これに対して、近年では、複数のスーパーコンあるいは PC クラスタを連携させて利用する傾向になりつつある。従来、行列ライブラリの開発においては、特定スーパーコン向けの性能チューニングに多くの工数を費やしてきた。これに加えて、種々のPCを含めた他の計算プラットフォーム向けに同様の作業を実施すると、工数は莫大なものとなる。このため、チューニングを自動的に実現する機能、すなわち自動チューニング機能が必要になってくる。

報告者らは自動チューニングの形式的フレームワークを既に提案[1]しており、その行列ライブラリ製品への適用を検討中である。また、同様の研究開発が国内外で活発化しており、国内では、並列行列ライブラリを自動チューニングする東大の I-LIB[2][3]、自動チューニングを支援するスクリプト言語である電通大の ABCLibscript[4]とそのフレームワーク FIBER[5]などの研究プロジェクトが存在する。海外では行列ライブラリ LAPACK を自動チューニングするテネシー大の ATLAS[6]が提案され、PC 上でも高速に稼動する行列ライブラリとして有名になっている。

一方、研究[7][8]において、プログラム実行条件によっては、自動チューニングがかえって性能劣化を招く場合があることが分かってきた。この研究では、行列計算処理の高速化手法として一般的であるループアンローリングが、行列サイズ毎の性能ばらつきを増大させる傾向を明らかにした。その結果、性能保証の必要性を認識するに至り、性能保証レベルを計算するアルゴリズムを提案してきた[9]。しかし、この提案方式では、性能保証レベルを明確に出来るものの、その保証値は比較的低い値に留まるという課題が残った。

そこで本研究では、この課題を解決し、GRID コンピューティング環境でのアプリケーション処理に対して、より高い性能を保証する機能を提供することを目的とする。

2.　行列ライブラリにおける性能上の課題

一般に、行列ライブラリの性能向上方法には、ループアンローリング、整合寸法調整、ブロック化などがあり、それぞれ効果が認められている。しかし、以下に述べるような、部分的な性能劣化、および性能変動という問題があることが分かってきた。

まず、ループアンローリングによる高速化は、行列のサイズ毎に適用効果が大きく異なり、部分的な性能劣化や性能変動を起こすことが分かってきた[7]。そのような例として、図1にスーパーコン SR8000 での、図2に Pentium4 での部分的な性能劣化および性能変動を示す。図1、図2ともに横軸が行列サイズ、縦軸が性能値（Mflop/s）である。図中、(X-Y-Z)の表記は、評価対象プログラムにおけるアンローリング段数パタンを示している。例えば(4-2-1)は、最外ループを 4 段、次のループを 2 段、最内ループを 1 段という形でアンロールしていることを表す。

図1、図2いずれにおいても、アンローリング段数を増やしていくと、部分的な性能劣化や性能の変動幅が大きくなる現象が観測できる。例えば、図1では、(4-4-1)という比較的少ないアンロール段数では性能が安定しているのに対し、(5-5-2)では3060-3080次元という部分的な領域において性能劣化を起こしている。また、図2では、(2-1-1)などが性能は低いものの比較的安定した性能を示しているのに対し、(4-2-1)では1020-1030次元という部分的な領域で性能劣化を起こしており、全般に高い性能ながらも交互に性能が大きく変わるという性能変動が見られる。

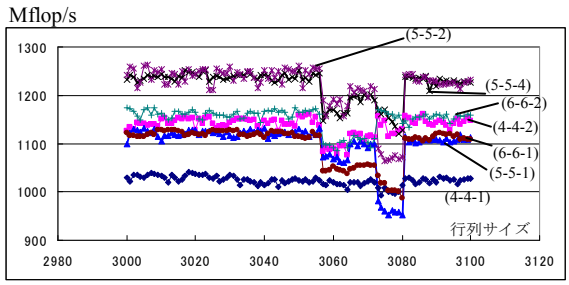

図1　固有値ループでの各アンロール段数パタンでの性能劣化の例
（日立SR8000/F1, 1CPU）

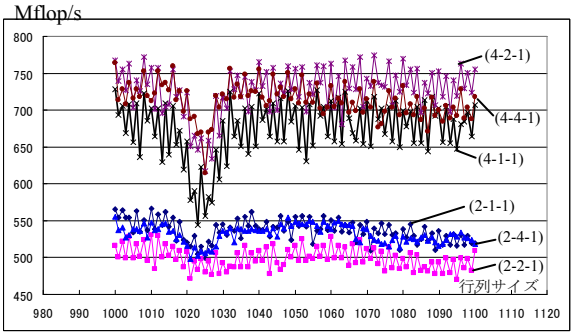

図2　ランク2 更新ループでの各アンロール段数パタンでの性能劣化の例(Pentium4(HT)3.2GHz)

このように、アンローリングには、部分的な性能劣化と性能変動の問題がある。性能劣化の理由としては、キャッシュカラム競合が指摘されており、その競合を回避するためにデータ配列にスペーサを挿入するアルゴリズムが提案されている[8]。しかし、性能変動については、以下のような要因も知られており、上記のアルゴリズムのみでは対処が困難である。

- プログラム実行時のデータ領域割当アドレス：インテル Pentium4 プロセッサでは、下位ビットを無視したエイリアシング判定処理を行うため、異なる番地へのメモリアクセスが番地競合とみなされ、待ち時間が発生する場合がある[10]。このため、プログラム実行時にデータ領域が割り当てられるアドレスによって、性能劣化が起きたり、起きなかったりする。
- プログラムのオブジェクトコードサイズ：RS/6000 サーバでは、プログラム処理内容が全く同じであっても、そのオブジェクトコードサイズが異なれば性能変動を起こす可能性がある[11]。例えば、行列計算の部分は全く同じであったとしても、その前の処理内容が異なり、実行オブジェクトが違う場合、行列計算で処理されるデータ領域の割当アドレスが異なる。このため、行列計算の処理条件が全く同じであっても、メモリアクセスにおける番地競合の状況が異なり、性能変動が発生する。研究[11]では、オブジェクトコードに適切なスペーサを挿入することで、この性能変動を回避しているが、行列計算プログラムのソースコード上でこの変動を回避することは困難である。

以上の状況を纏めると、行列ライブラリの性能上の課題は、アンローリングによる高速化を実施する一方で、上記のような性能劣化の度合いを緩和しつつ、不確定に発生する性能変動をできるだけ回避し、性能保証レベルを向上することである。

3.　徒競走型の性能保証レベル向上方法

3.1　基本的な考え方

前章の性能保証レベルを向上させる課題に対し、以下の方法を検討する。

図1、図2によれば、多くのケースで、隣り合う行列サイズで性能が変動している。これは、2つの連続する行列サイズで計算を実行すれば、どちらか一方は比較的良い性能になることを示している。この特徴を利用すれば、ユーザの与える行列サイズに対し、{+0, +1}という複数の整合寸法を加え、それぞれのケースについて同時に計算を実行し、その中で早く計算が終了した方から計算結果を得る、という徒競走型の処理方式が考えられる。これによって、上記の不確定に発生する性能変動を緩和するという課題の解決が図られる。

また、複数のアンローリング段数パタンについて徒競走型の処理方式を適用すれば、アンローリング段数を引き上げる際に、性能変動が激しくなった結果、大きく性能劣化を起こしても、他のアンローリング段数パタンがある程度の性能を維持していれば、そちらで大きな性能劣化を回避できる。これによって、アンローリング段数を引き上げる際の性能劣化を緩和するという課題の解決が図られる。

図3に行列サイズの整合寸法に着眼した徒競走の例を示す。整合寸法{+0}, {+1}の各ケースの実行結果のうち、性能値が高いほうが徒競走方式での性能になる(図3の太線)。そのため、整合寸法{+0}と整合寸法{+1}でそれぞれ単独に実行した場合の最低性能値が、徒競走型の最低性能値まで向上する。このような形で、徒競走型の処理方式は性能保証レベルを向上させることが期待できる。

徒競走型の処理方式は、最も早く計算を終了させるプロセッサ以外は、計算資源を無駄に利用することになる。しかし、以下の2つの理由から、情報ユーティリティ時代のサービス提供手段として有効であると考える。

- 高性能汎用プロセッサが普及し、多くの計算資源を使用するコストが低下し、余剰計算資源も増えている。
- ユーザにとっては、計算機システムが十分にその性能を出し切るというスループット性能よりも、処理すべきアプリケーションが定められた時間で確実に終了するという性能保証のほうが、より価値が高い。

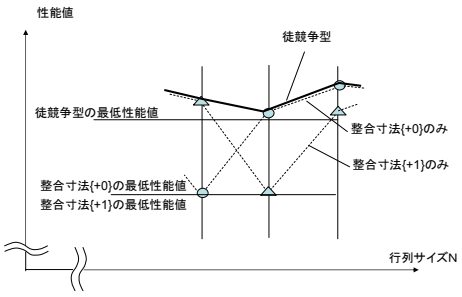

図3　徒競走型の性能保証レベル向上の例

3.2　実装方式

行列計算におけるチューニングパラメータとしては、アンローリング段数パタンと行列サイズの整合寸法の他、ブロックアルゴリズムでのブロックサイズ、反復回数など種々考えられる。ここでは説明の簡易化のため、アンローリング段数パタンおよび行列サイズの整合寸法のみに着眼し、また、そのうち行列サイズの整合寸法のみを徒競走の対象とする。図4に、この場合の徒競走型性能保証レベル向上方法の実装例を示す。

(1) 対象とする行列サイズの区間、およびアンローリング段数パタンの範囲を定める。例えば、行列サイズ N=1000-1100、アンローリング段数パタン unroll＝{(1-1-1) (1-2-1) (1-4-1)}などである。
(2) 各アンローリング段数パタンについて、対象とする行列サイズ区間での性能の平均値を計算する。その平均値が最大であるアンローリング段数パタンを unroll-max とする。
(3) 整合寸法 LA を{+0, +1, +2, +3}の4通りとし、アンローリング段数パタンを unroll-max として、行列計算を実行する(すなわち、行列サイズは N+0, N+1, N+2, N+3 の4通り)。
(4) 4通りの整合寸法 LA の候補の中で最も早く計算が終了したものから計算結果を得て、他の候補の計算をキャンセルする。

図4 徒競走型の性能保証レベル向上方法実装例

図4の(1)では、対象とする行列サイズの区間およびアンローリング段数パタンの範囲を定める。行列サイズをある特定の値ではなく区間で考えるのは、ピンポイント的な性能情報のもつ不確定さを排除し、区間において平均的によい性能としてのアンローリング段数パタンを選ぶためである。アンローリング段数パタンの範囲は、ループの多重度と対象とする計算機のレジスタ数から経験的に定める。

図4の(2)では、対象とする行列サイズの区間において、平均的な性能が最も良いアンローリング段数パタンを決定する。これは、アンローリング段数パタンごとに、当該区間の行列サイズごとにすべて性能を計測するか、あるいは、いつくかの行列サイズをランダムサンプリングして推定する方法がある。

図4の(3)(4)では、(2)で定まったアンローリング段数パタンに対し、整合寸法 LA を{+0, +1, +2, +3}の4通りとして行列計算をそれぞれ実行し、最も早く計算が終了した候補から計算結果を得て、他の候補は計算をキャンセルする。これによって、3つ連続して性能が落ちたとしても、残りの候補の性能が高い場合、性能保証値を引き上げることが期待できる。

4. 基本行列演算による提案方法の適用効果検証

4.1 評価実験対象および評価実験環境

評価実験対象プログラムは、基本行列演算 BLAS から選択した表 1 の 5 種類である。これらは、BLAS の中でも比較的よく利用されるものである。

表1 実験対象の基本行列演算

レベル	名称	処理内容
BLAS1	DAXPY	ベクトル内積
BLAS2	DGEMV1	行列ベクトル積
BLAS2	DGEMV2	転置行列ベクトル積
BLAS3	DGEMM	行列乗算(但しN×40 の行列による)
BLAS3	DSYR2K	ランク2更新(但しN×40 分の更新による)

評価実験環境は以下のとおりである。

- ハードウェア・プラットフォーム:Hitachi FLORA370DG(CPU:Pentium 4 (HT) 3.2GHz, L1 キャッシュ:8KB, L2 キャッシュ:512KB, 主記憶:512MB DDR-SDRAM)
- コンパイラ:Intel FORTRAN Compiler Version 7.1
- コンパイルオプション:-O3 -prefetch -unroll -align -lowercase -nodps -fpp2 -tpp7 -xW -vec_report3 -opt_report

対象とするチューニングパラメータは、アンローリング段数パタンおよび行列サイズの整合寸法である。表2に各 BLAS 演算の対象アンローリング段数パタンを示す。整合寸法は、全ての BLAS 演算で{+0}, {+1}, {+2}, {+3}の 4 通りとする。また、各 BLAS の展開プログラム例を図5、図6に示す。なお、次節での評価においては、本プログラム例に加えて、文献[10]を参考にしたアーキテクチャ上のチューニングを施した。

表2 実験対象の基本行列演算のアンローリング段数パタン

名称	アンローリング段数パタン
DAXPY	(1), (2), (4), (8), (16)
DGEMV1	(1-1), (2-1), (4-1), (8-1), (16-1)
DGEMV2	(1-1), (2-1), (4-1), (8-1), (16-1)
DGEMM	(1-1-1), (1-2-1), (1-4-1), (2-1-1), (2-2-1), (2-4-1), (4-1-1), (4-2-1), (4-4-1)
DSYR2K	(1-1-1), (1-2-1), (1-4-1), (2-1-1), (2-2-1), (2-4-1), (4-1-1), (4-2-1), (4-4-1)

```
BLAS1 (DAXPY) (4)展開--------------------
      DO 50 I = 1,N,4
        Y(I) = Y(I) + A*X(I)
        Y(I + 1) = Y(I + 1) + A*X(I + 1)
        Y(I + 2) = Y(I + 2) + A*X(I + 2)
        Y(I + 3) = Y(I + 3) + A*X(I + 3)
 50 continue
------------------------------------------------

BLAS2 (DGEMV1) (2-1)展開-----------------------
        DO 240 J=1,N,2
          S0=ALPHA*X(J   )
          S1=ALPHA*X(J+1)
          DO 230 I=1,M
            Y(I)=Y(I)+A(I,J   )*S0
     *              +A(I,J+1)*S1
230       CONTINUE
240     CONTINUE
------------------------------------------------

BLAS2 (DGEMV2) (2-1)展開-----------------------
        DO 340 J=1,N,2
          S=0.0D0
          S1=0.0D0
          DO 330 I=1,M
            S =S +A(I   ,J)*X(I)
            S1=S1+A(I,J+1)*X(I)
330       CONTINUE
          Y(J   )=ALPHA*S
          Y(J+1)=ALPHA*S1
340     CONTINUE
------------------------------------------------

BLAS3 (DGEMM) (4-2-1)展開----------------------
      PARAMETER (LF=2,LG=4)
      LGMOD=MOD(LGX,LG)
      LFMOD=MOD(LFX,LF)
      DO 1040 NG=0,LGX-1-LGMOD,LG
        DO 1010 NF=0,LFX-1-LFMOD,LF
          S11=0.0D0
          S21=0.0D0
          S12=0.0D0
          S22=0.0D0
          S13=0.0D0
          S23=0.0D0
          S14=0.0D0
          S24=0.0D0
          DO 1000 K=0,LHX-1
            S11=S11+A(K,NF   )*B(K,0)
            S12=S12+A(K,NF   )*B(K,1)
            S13=S13+A(K,NF   )*B(K,2)
            S14=S14+A(K,NF   )*B(K,3)
            S21=S21+A(K,NF+1)*B(K,0)
            S22=S22+A(K,NF+1)*B(K,1)
            S23=S23+A(K,NF+1)*B(K,2)
            S24=S24+A(K,NF+1)*B(K,3)
1000      CONTINUE
          C(NF   ,NG   )=C(NF   ,NG   )+S11
          C(NF+1,NG   )=C(NF+1,NG   )+S21
          C(NF   ,NG+1)=C(NF   ,NG+1)+S12
          C(NF+1,NG+1)=C(NF+1,NG+1)+S22
          C(NF   ,NG+2)=C(NF   ,NG+2)+S13
          C(NF+1,NG+2)=C(NF+1,NG+2)+S23
          C(NF   ,NG+3)=C(NF   ,NG+3)+S14
          C(NF+1,NG+3)=C(NF+1,NG+3)+S24
1010    CONTINUE
1040  CONTINUE
```

図5 BLAS の展開プログラム例
(DAXPY, DGEMV1, DGEMV2, DGEMM)

```
BLAS3 (DSYR2K) (4-2-1)展開-------------------------
      NMOD=MOD(N,4)
      KMOD=MOD(K,2)
      DO 50 J=1,N-NMOD,4
         DO 20 L=1,K-KMOD,2
            Q1=ALPHA*Q(J  ,L  )
            Q2=ALPHA*Q(J  ,L+1)
            Q3=ALPHA*Q(J+1,L  )
            Q4=ALPHA*Q(J+1,L+1)
            Q5=ALPHA*Q(J+2,L  )
            Q6=ALPHA*Q(J+2,L+1)
            Q7=ALPHA*Q(J+3,L  )
            Q8=ALPHA*Q(J+3,L+1)
            U1=ALPHA*U(J  ,L  )
            U2=ALPHA*U(J  ,L+1)
            U3=ALPHA*U(J+1,L  )
            U4=ALPHA*U(J+1,L+1)
            U5=ALPHA*U(J+2,L  )
            U6=ALPHA*U(J+2,L+1)
            U7=ALPHA*U(J+3,L  )
            U8=ALPHA*U(J+3,L+1)
            DO 10 I=1,J
         A(I,J  ) = A(I,J  ) + U(I,L  )*Q1 + Q(I,L  )*U1
     *                       + U(I,L+1)*Q2 + Q(I,L+1)*U2
         A(I,J+1) = A(I,J+1) + U(I,L  )*Q3 + Q(I,L  )*U3
     *                       + U(I,L+1)*Q4 + Q(I,L+1)*U4
         A(I,J+2) = A(I,J+2) + U(I,L  )*Q5 + Q(I,L  )*U5
     *                       + U(I,L+1)*Q6 + Q(I,L+1)*U6
         A(I,J+3) = A(I,J+3) + U(I,L  )*Q7 + Q(I,L  )*U7
     *                       + U(I,L+1)*Q8 + Q(I,L+1)*U8
   10       CONTINUE
   20    CONTINUE
   50 CONTINUE
-----------------------------------------------------
```

図6 BLAS の展開プログラム例(DSYR2K)

4.2 評価結果

所定の行列サイズ N の区間(例えば、N=1000-1100)において、区間内最低性能値がどの程度向上したかで性能保証レベル向上度を評価した。また、区間内での最低性能値と最高性能値の差を小さくした率を性能変動幅削減率(徒競走後の差／徒競走前の差)として評価した。ここで、整合寸法の組(すなわち、徒競走の組)については、{+0}, {+0, +1}, {+0, +1, +2}, {+0, +1, +2, +3}の 4 通りとした。以後、これらの徒競走の組をそれぞれ、{+0}, Max{+0, +1}, Max{+0, +1, +2}, Max{+0, +1, +2, +3}と表記する。

表3に各 BLAS の区間最良アンローリング段数パタンと各徒競走での区間内最低性能値、および性能変動幅削減率の結果を示す。

Max{+0,+1,+2,+3}の徒競走により、DSYR2K では区間最低性能が約 8.6%も向上し、徒競走の効果が比較的大きい結果となった。他の BLAS では、区間最低性能の向上率が DAXPY において約 0.77%および 0.72%、DGEMM において約 1.02%および 2.83%などそれほど大きくないが、性能変動幅については約 23%～60%削減できており、徒競走の効果が確認できた。

表3 各 BLAS における区間内最低性能値の向上率および性能変動幅削減率

BLAS/ unroll	区間	整合寸法	区間内最低性能値 (向上率)	性能変動幅削減率
DAXPY/ (4)	9000-9100	{+0}	1724.14 Mflop/s (-----)	---
		Max{+0,+1}	1735.06 Mflop/s (1.0063)	49.6%
		Max{+0,+1,+2}	1737.36 Mflop/s (1.0077)	60.0%
		Max{+0,+1,+2,+3}	1737.36 Mflop/s (1.0077)	60.0%
DAXPY/ (2)	30000-30100	{+0}	535.73 Mflop/s (-----)	---
		Max{+0,+1}	536.42 Mflop/s (1.0013)	8.9%
		Max{+0,+1,+2}	538.77 Mflop/s (1.0057)	39.3%
		Max{+0,+1,+2,+3}	539.58 Mflop/s (1.0072)	49.8%
DGEMV1/ (2-1)	600-700	{+0}	729.43 Mflop/s (-----)	---
		Max{+0,+1}	761.20 Mflop/s (1.0436)	36.4%
		Max{+0,+1,+2}	762.66 Mflop/s (1.0456)	38.2%
		Max{+0,+1,+2,+3}	762.66 Mflop/s (1.0456)	38.2%
DGEMV1/ (2-1)	1300-1400	{+0}	774.81 Mflop/s (-----)	---
		Max{+0,+1}	778.07 Mflop/s (1.0043)	7.1%
		Max{+0,+1,+2}	782.99 Mflop/s (1.0106)	17.7%
		Max{+0,+1,+2,+3}	793.67 Mflop/s (1.0243)	40.8%
DGEMV2/ (2-1)	800-900	{+0}	761.06 Mflop/s (-----)	---
		Max{+0,+1}	767.03 Mflop/s (1.0078)	9.1%
		Max{+0,+1,+2}	781.58 Mflop/s (1.0270)	31.4%
		Max{+0,+1,+2,+3}	786.22 Mflop/s (1.0331)	38.5%
DGEMV2/ (2-1)	1700-1800	{+0}	688.49 Mflop/s (-----)	---
		Max{+0,+1}	703.74 Mflop/s (1.0222)	16.6%
		Max{+0,+1,+2}	712.95 Mflop/s (1.0355)	26.6%
		Max{+0,+1,+2,+3}	713.11 Mflop/s (1.0358)	26.8%
DGEMM/ (1-4-1)	1000-1100	{+0}	3071.78 Mflop/s (-----)	---
		Max{+0,+1}	3101.94 Mflop/s (1.0098)	22.0%
		Max{+0,+1,+2}	3102.48 Mflop/s (1.0100)	22.3%
		Max{+0,+1,+2,+3}	3103.25 Mflop/s (1.0102)	22.9%
DGEMM/ (1-4-1)	2500-2600	{+0}	3056.02 Mflop/s (-----)	---
		Max{+0,+1}	3119.58 Mflop/s (1.0208)	41.6%
		Max{+0,+1,+2}	3132.03 Mflop/s (1.0249)	49.7%
		Max{+0,+1,+2,+3}	3142.42 Mflop/s (1.0283)	56.5%
DSYR2K/ (4-4-1)	500-600	{+0}	740.15 Mflop/s (-----)	---
		Max{+0,+1}	792.86 Mflop/s (1.071)	35.4%
		Max{+0,+1,+2}	792.86 Mflop/s (1.071)	35.4%
		Max{+0,+1,+2,+3}	803.94 Mflop/s (1.086)	42.9%
DSYR2K/ (4-2-1)	1800-1900	{+0}	662.12 Mflop/s (-----)	---
		Max{+0,+1}	681.05 Mflop/s (1.029)	29.9%
		Max{+0,+1,+2}	687.90 Mflop/s (1.039)	40.7%
		Max{+0,+1,+2,+3}	697.02 Mflop/s (1.053)	55.2%

また、ATLAS(バージョン 3.6)と、Max{+0, +1, +2, +3}、および{+0}単独実行とを比較した結果を図7に示す。DAXPY、DGEMM、DSYR2K では、ATLAS よりも Max{+0, +1, +2, +3}および{+0}単独実行いずれもが高い性能を示した。

一方、DGEMV1 では、区間内の最高性能では ATLAS が優位にあるものの、最低性能値では ATLAS が最も低く、Max{+0, +1, +2, +3}が最も高い値になっている。特に、区間 N=600-700 においては、{+0}単独実行と ATLAS の最低性能値がほぼ同様であるのに対し、Max{+0, +1, +2, +3}では約 40Mflop/s 高い値になっている。このことから、徒競走型の性能保証レベル向上が有効であることが分かる。DGEMV2 では、ATLAS が非常に安定かつ、高い性能となっており、より多くのアンローリング段数パタン、および何らかの性能安定化方法を採用しているものと予想される。

5. 関連研究

行列ライブラリの高性能化という課題に対して、国内外で自動チューニング技術に関する研究が盛んに行われている。東大の黒田ら[2]、工藤ら[3]は反復法の動的最適化方法を提案し、電通大の片桐らは可搬性のある自動チューニング・フレームワークを提唱しており[4][5]、また、電通大の今村らが性能の安定化技術を提案している[8]。海外では、米テネシー大の Dongarraらが自己適合型の行列ライブラリチューニング技術 SANS(Self-Adapting Numerical Software)[12]を発表している。

6. 纏めと今後の課題

本報告では、複数の性能チューニングパタンで行列計算プログラムを同時実行させて最速の結果を採用する徒競走型の性能保証レベル向上方法を提案した。本方法では、行列計算プログラムを処理するにあたって、例えば、所定の行列サイズの区間内における平均性能が最も高いアンローリング段数パタンを定めた後、行列サイズの整合寸法について{+0}, {+1}, {+2}, {+3}といった複数の候補を設定した上で、それぞれの条件で計算処理を同時実行させ、最も早く計算が終了した結果のみを採用する。

PC(Pentium4、3.2GHz)上で5種類の基本行列演算を対象に、提案方式の適用効果を検証したところ、所定の行列サイズの区間における最低性能値が最大約8.6%向上するという結果を得た。また、その性能向上率が1%に満たないケースであっても、性能変動の幅を最大約60%削減するという結果を得た。

さらに、自動チューニングライブラリATLASとの比較実験を行った結果、5種類中4種類の基本行列演算に対して本報告での提案法が高い保証性能を示し、本提案法での性能保証が効果的であることが明らかになった。ATLASは、PC上で自動チューニングを行う行列ライブラリとして有名であるが、単なる自動チューニングのみでは性能変動を回避できず、本報告で提案した方法が重要であることが確認された。

以下に今後の課題を列挙する。

(1) 今回の評価実験ではPCのみを対象とした。今後、種々のライブラリ対応機種において数値実験を行い、次期行列ライブラリの基盤機能として活用していく。

(2) 今回の実験では基本行列演算機能のうち、DAXPY, DGEMV1, DGEMM, DSYR2KではATLASよりも高い最低性能値を達成できたが、反復法で多用されるDGEMV2ではATLASがより高い最低性能値を実現した。今回計測していない他のアンローリング段数パタンや、他のチューニングパラメータなどの調整によって、より高性能なDGEMV2の実装方式を検討する。

(3) 今回の評価実験では、基本行列演算を対象とした。今後は、この結果を活かし、一般行列演算の処理過程に提案方法を組み込むことを検討する。具体的には、実行時の自動チューニング過程に徒競走方法を組み込み、実効性能を検知しながら性能劣化を回避し、高い性能を維持するような性能保証機能付きの高性能ライブラリ方式を検討する。

謝辞：

本研究に至る上で共同研究を通じ重要な数多くの寄与を頂いた電気通信大学の今村俊幸講師に謝意を表します。研究の方向性に関し有益な助言を頂いた電気通信大学の弓場敏嗣教授、片桐孝洋助手に謝意を表します。また、平成13年度に実施された共同研究で施設利用させていただいた日本原子力研究所に謝意を表します。

参考文献

[1] 直野健, 山本有作:単一メモリ型インターフェイスを有する自動チューニング並列ライブラリの構成方法, 情報処理学会研究報告 2001-HPC-87(SWoPP2001), pp25-30, 2001.

[2] Hisayasu Kuroda, Takahiro Katagiri, and Yasumasa Kanada: Knowledge Discovery in Auto-tuning Parallel Numerical Library, Progress in Discovery Science, Final Report of the Japanese Discovery Science Project. Lecture Notes in Computer Science 2281 Springer 2002, pp.628-639, 2002.

[3] Makoto Kudoh, Hisayasu Kuroda, and Yasumasa Kanada: Parallel Blocked Sparse Matrix-Vector Multiplication with Dynamic Parameter Selection Method, ICCS2003 (International Conference on Computational Science 2003), International Conference, Melbourne, Australia and St. Petersburg, Russia, June 2-4, 2003. Proceedings, Part III, Lecture Notes in Computer Science 2659 Springer 2003, pp.581-591.

[4] 自動ブロック化・通信最適化ライブラリ ABC-LIB: http://www.abc-lib.org/.

[5] KATAGIRI Takahiro, KISE Kenji, HONDA Hiroki, and YUBA Toshitsugu: FIBER: A General Framework for Auto-Tuning Software, Springer LNCS 2858, pp.146--159, The Fifth International Symposium on High Performance Computing (ISHPC-V), 2003.

[6] ATLAS project : http://www.netlib.org/atlas/index.html.

[7] 直野健, 今村俊幸: 自動チューニング型固有値ソルバについて, SWoPP2002, 情報処理学会研究報告, Vol. 2002, No. 91, pp. 49-54.

[8] 今村俊幸, 直野健: 性能安定化を目指した自動チューニング型固有値ソルバーについて, SACSIS (Symposium on Advanced Computing Systems and Infrastructures) 2003, pp.145-152.

[9] 直野健, 今村俊幸, 恵木正史: GRID コンピューティング環境における行列ライブラリ向け性能保証方式の検討, 情報処理学会論文誌:コンピューティングシステム Vol.45 No.SIG6 (ACS6), May 2004, pp105-112.

[10] インテル Pentium4 プロセッサおよびインテル Xeon プロセッサ最適化リファレンスマニュアル: ftp://download.intel.co.jp/jp/developer/jpdoc/24896606_j.pdf.

[11] 高村明裕, 梅原俊治, 三木良雄: オブジェクトコードの配置アドレス改善によるプログラム高速化と評価, 情報処理学会研究報告 HPC, Vol. 2003, No. 93-21,

pp.119–124.
[12] Jack Dongarra, Victor Eijkhout: Self-adapting numerical software for next generation applications, The International Journal of High Performance Computing Applications, Vol. 17, No. 2, Summer 2003, pp. 125–131.

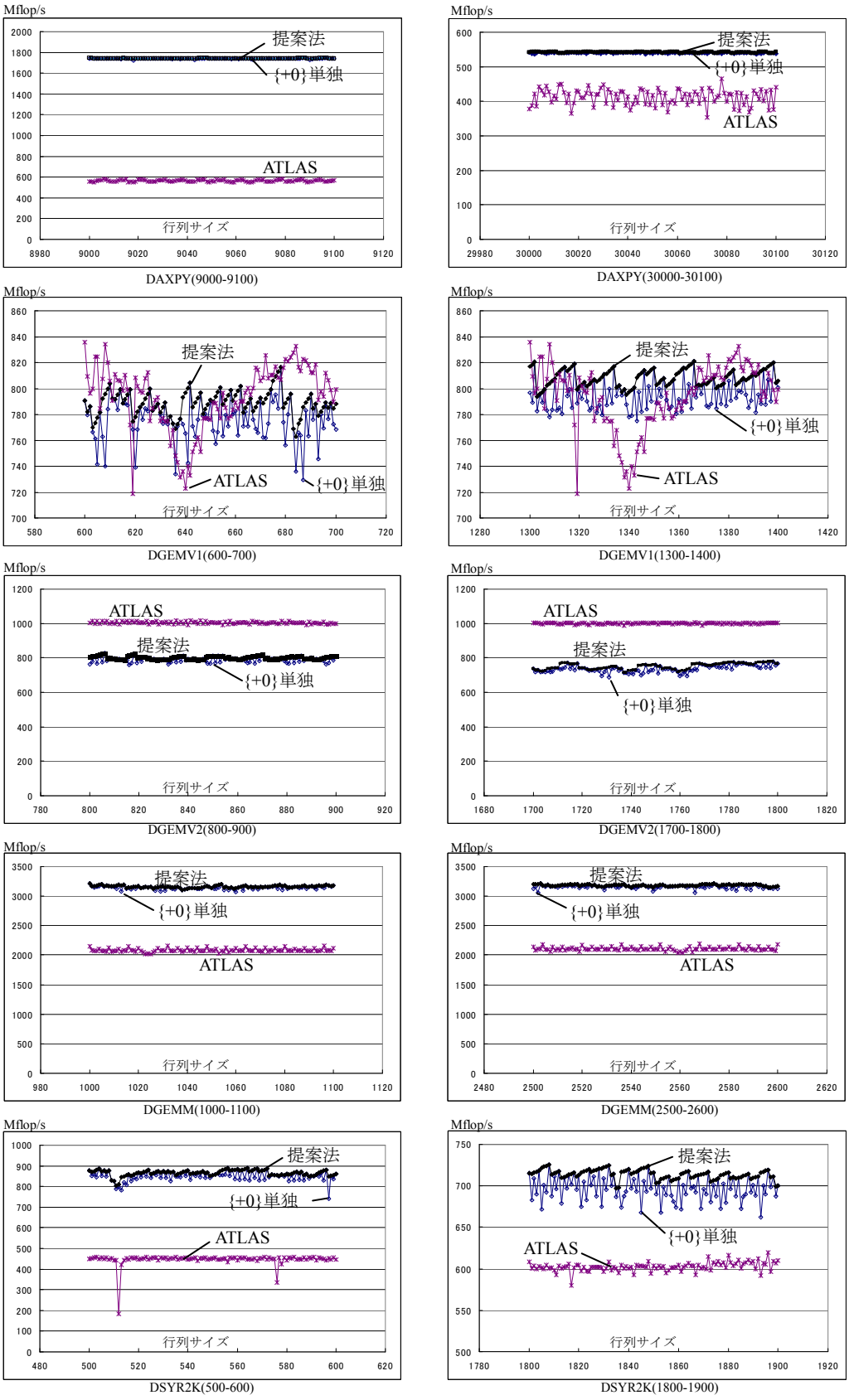

図 7　提案法（Max{+0, +1, +2, +3}）と{+0}単独実行時、および ATLAS の性能比較