2005－HPC－103（6）
2005／8／3

数値計算ポリシー入力型自動チューニング方式

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

従来、行列計算ライブラリは、計算精度、計算時間などの利用属性を入力パラメータで直接制御することはできず、所望の利用属性で計算を行うには多数回試行錯誤が必要であった。そこで、本報告では、計算資源と計算時間のバランスを調整する数値計算ポリシー入力型の行列計算ライブラリ方式を提案する。特に、パラメータの設定によって使用メモリ量と計算時間のトレードオフがあり、試行錯誤を必要としていた疎行列固有値ソルバにおいて本方式を適用した。その結果、MatrixMarket の数万次元の例題に対して、メモリ量と計算時間を理想モデルに対してバランスさせるパラメータを選択することができた。

Automatic Tuning Framework with Numerical Policy Interface

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

So far, the accuracy or the computation time of matrix libraries can not be controlled directly by users in general, thus users have to execute many trials and errors in order to realize expected accuracy or computation time. In this paper, we propose a framework of numerical library to tune the balance between computing resource parameter and computing time. We apply the framework for a sparse eigensolver, which has required many trials and errors to tune between the amount of memory for working area and the computation time. The results show that the framework can select a suitable value for the parameter of amount of memory and computation time for ideal computing models.

1．　はじめに

従来、行列計算ライブラリは、計算精度、計算時間などの利用属性を入力パラメータで直接制御することはできず、所望の利用属性で計算を行うには多数の試行錯誤が必要であった。例えば、1 時間以内で計算を終了させるには、どのような行列の次元数にすればよいか、あるいは、どの程度のワークエリア（作業領域用のメモリ量）を確保すればよいかが計算実行の前には不明である。

しかし、行列計算ライブラリの研究は、より高速なアルゴリズムと、並列計算機などの計算プラットフォームへの実装技術が中心であり、上述の問題は残されたままであった。逆に、CPU 数、ワークエリア、種々の収束判定基準など、取り扱うパラメータの種類が大幅に増えたため、この問題は年々深刻になっている。

そこで、最近になって、パラメータの自動チューニングの技術が研究されるようになってきた[1-10]。これらのうち代表的な研究は、行列の次元数 N などのユーザプログラムで重要なパラメータを与えると、計算時間が最短になるよう、ループアンローリング段数などのユーザプログラムにとっては無関係なパラメータを自動的に算出する技術である。

自動チューニングの研究は、これまで主に、計算時間の短縮化を目標にした研究や、計算時間と計算精度のバランスを検討した研究[11]がなされている。ところが、計算のため確保すべきリソース量と計算時間のトレードオフについては、まだ、十分な検討がなされていない。

そこで本報告では、使用する計算資源の量と計算時間のバランスを調整する数値計算ポリシー入力型の行列計算ライブラリ方式を提案する。特に、パラメータの設定によって使用メモリ量と計算時間のトレードオフがあり、試行錯誤を必要としていた疎行列固有値ソルバにおいて本方式を適用する。

2．　疎行列固有値ソルバのメモリ量と計算時間のトレードオフ

2.1　疎行列固有値ソルバにおける疎行列ベクトル積の実装

本報告では、対称疎行列固有値ソルバのアルゴリズムである Lanczos 法について検討する。Lanczos 法は、標準固有値問題　$Ax=\lambda x$　（λ:固有値、x:固有ベクトル）において、行列 A が疎の対称行列である際に広く利用されているアルゴリズムである。行列 A の次元数を N とする。本報告では、N が数万以上である対称疎行列 A の固有値を計算することを検討する。Lanczos 法の主要な演算は、疎行列 A とベクトルの積と、直交化処理である。直交化処理における数値計算ポリシーは、別途報告[11]によるので、本報告では、疎行列とベクトルの積について検討する。

図1に、疎行列ベクトル積のデータ構造を示す。行列Aの非ゼロ要素数をNZとすると、行列Aは、上半分の要素につき、列方向（横方向）に、非ゼロ要素の部分のみを、圧縮して配列A(NZ)に格納する。補助配列LL(NZ)は、Aの非ゼロ要素が存在する、列方向のインデックスを指す。また、補助配列LC(N+1)は、Aの対角成分の配列A(NZ)におけるインデックスを指す。

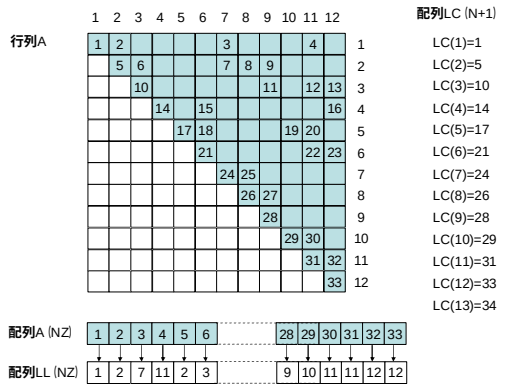


図1 対称疎行列Aのインデックス、補助配列のインデックス

このようなデータ構造および補助配列を与えると、疎行列ベクトル積のプログラム R=A*X (但し R、Xはベクトル) は、＜プログラム1＞のようになる。

＜プログラム1＞

```
REAL*8 A(NZ), LL(NZ), LC(N+1), R(N), X(N)

DO I = 1, N
 S = A(LC(I))*X(I)
 DO JC = LC(I)+1, LC(I+1)-1
   JJ = LL(JC)
   S = S + A(JC) * X(JJ)
   R(JJ) = R(JJ) + A(JC) * X(I)
  ENDDO
  R(I) = R(I) + S
ENDDO
```

2.2 SR11000 向け疎行列ベクトル積の実装におけるメモリ量と計算時間のトレードオフ

上記の疎行列ベクトル積を並列計算機 SR11000 の1ノード (16CPU) に実装することを考える。＜プログラム1＞を 16CPU で並列化しようとすると、R(I)への足し込みの箇所で、インデックス I に関して並列性が確保できない。そこで、これを回避するため、補助配列 RR(N, 16) を追加して、後で R(I)に足し込むことにすると、＜プログラム2＞のようになる。

このプログラム実装により、最外側のループでの16CPU の並列化が可能になる。よって、16CPU での並列化効果による実行時間の短縮が期待できる。しかし、一方で、RR の配列分、即ち 16N の分だけのワークエリアを確保する必要が生じる。

そこで、SR11000 向けの疎行列ベクトル積では、この部分について、確保されるワークエリアの大きさに応じて、配列 RR の列数を変化させるという実装方式が考えられる。

このような実装では、「RR のワークエリア=0」の場合、メインの演算である行列ベクトル積があまり高速ではないものの、使用メモリ量をより少なく出来るという効果が期待できる。一方、「RR のワークエリア=16N」の場合、使用メモリ量を増やしてしまうが、16CPU 実行時には、メインの演算である行列ベクトル積を高速に処理できるという期待がある。しかし、1CPUの場合には、高速化は期待できない。

＜プログラム2＞

```
REAL*8 A(NZ), LL(NZ), LC(N+1), R(N), X(N)
REAL*8 RR(N,16)

DO IP = 1, 16
 DO I = (IP-1)*N/16+1, IP*N/16
   XX = X(I)
   S = A(LC(I)) * XX
   DO JC = LC(I)+1, LC(I+1)-1
    JJ = LL(JC)
    S = S + A(JC) * X(JJ)
    RR(JJ, IP) = RR(JJ, IP) + A(JC) * XX
   ENDDO
   R(I) = R(I) + S
 ENDDO
ENDDO
DO I = 1, N
 S = R(I)
 DO IP = 1, 16
   S = S + RR(I, IP)
 ENDDO
 R(I) = S
ENDDO
```

以上のことから、計算時間は少々長くても、メモリ使用を極力減らして計算したいという場合には、「RR のワークエリア=0」として 1CPU で計算実行するのが適切であり、メモリを多く使用しても、できるだけ計算時間を短くするには「RR のワークエリア=16N」として、16CPU で計算実行するのが良いことが分かる。

しかし、その中間的なケースではどのようにすればよいか明らかではない。例えば、メモリ使用範囲をある一定値に抑えつつも、その範囲ではできるだけ計算時間が短くなるような計算実行を望む場合がある。また、4CPU 使用時にワークエリアを 4N 確保するのと、16N 確保するので、メモリ量と計算時間をバランスはどちらがいいのかが明らかではない。

このような中間的なケースに対して、ユーザの数値計算ポリシーを形式的に扱えるようにするため、次章で、新しいライブラリの入力方式を提案する。

3. 数値計算ポリシー入力型の行列計算ライブラリ方式の提案

3.1 数値計算ポリシーの定義

まず、数値計算ポリシーを定義するための以下、4種類のパラメータを定義する。

第1に、数値計算ライブラリは、一般に、行列サイズ、ベクトル長などの入力パラメータを有する。この入力パラメータは、ユーザが直接的に制御するパラメータであ

る。ここでは、ユーザ制御パラメータ（User Control Parameter，以下 UCP）と呼ぶことにする。

第 2 に、更に、数値計算ライブラリは、並列化数、最大反復回数、アンローリング数など、ユーザにとっては直接的に関係ないが、属性関数 f を左右するパラメータを内部制御パラメータ（ICP）と呼ぶことにする。

第 3 に、CPU 数やワークエリアメモリ量などの計算機資源のパラメータをリソース制御パラメータ（Resource Control Parameter，以下 RCP）として定義する。

第 4 に、数値計算ライブラリは、「計算誤差」や「計算時間」という、計算の結果得られる属性を有する。これを利用属性関数（User Attribute Function，以下 UAF）とする。UAF は、上記 3 種類のパラメータの結果定まるので、UAF = UAF（UCP，ICP，RCP）と書くことにする。

上記の 4 種類のパラメータを用いて、数値計算ポリシーを以下のように定義する。

自動チューニングとは、UCP を与えて、UAF が最適化するよう、ユーザの許す RCS の範囲の元、ICP を自動的に決定することである。

数値計算ポリシーは、「計算誤差」は大きくても、「計算時間」は短く計算したい、というユーザの数値計算属性に対するポリシーと解釈される。更に、どの程度の計算リソースを利用するか、という要素もユーザのポリシーである。

従って、数値計算ポリシーとは、UAF および RCP の組み合わせから構成されると言える。これをポリシーセットと呼ぶことにする。数値計算ポリシーによる数値計算とは、ポリシーセットに対して、ICP を自動的に決定することと言える。

例えば、CPU 数という RCP と計算時間を調整するポリシーにおける定式化は、以下のようになる。まず、

CPU 数(RCP)； Num_CPU = Num_CPU(P)

計算時間(UAF)； Time = Time(P)

（但し P は並列化パラメータ）となる。

これらを用いて、

「並列化パラメータ P を調整して、CPU 数が 100 以下で、計算時間を短く計算したい」

という数値計算のポリシーを実現する問題は、以下の最適化問題と定式化される。

Find P such that Time(P) attains minimum with the condition of Num_CPU(P)≦100

3.2 数値計算ポリシー入力型の行列計算ライブラリ方式

ポリシーセット（RCP，UAF）に対する ICP の定め方として、本報告では、必須ラインと理想ラインという 2 つのインターフェイスを提案する。

まず、RCP と UAF ごとに、それぞれ必須ラインと理想ラインを定義する。

必須ラインは、少なくとも満足すべき限界のラインであり、理想ラインは、必ずしも満足しなくても良いが、できればそれに近くしたい、というラインである。

例えば、RCP として CPU 数を、UAF として計算時間を考える。まず、ある数値計算ライブラリのソルバが、SOLVER(UCP，ICP)というインターフェイスになっており、ICP によって RCP（CPU 数）および UAF（計算時間）が変わるとする。また、簡単のため、UCP は固定とする。すると、上記の理想ラインと必須ラインは、図 2 のようになる。ここでは、ICP が 1 から 7 まで変化しており、いずれかの必須ラインを超えてしまう ICP=1, 2, 7 は除外され、残りの値から選ばれることになる。

次に、必須ライン内に入ったパラメータの選択方法を定義する。必須ライン内のパラメータ選択基準は、計算時間を優先するか、省 CPU 数を優先するか、あるいは計算時間と CPU 数の調整に理想ラインを利用するモデルを与える方法とする。例えば、

(*) （実際の計算時間/理想ライン時間）+（実際の CPU 数/理想ライン CPU 数）が最小になる

というモデルである。

図 2 の例では、計算時間を優先する場合は ICP=6 が、CPU 数を少なくすることを優先する場合は ICP=3 が選択され、また、両者の理想ラインのバランスを上記のようにとる場合には ICP=4 が選択される。

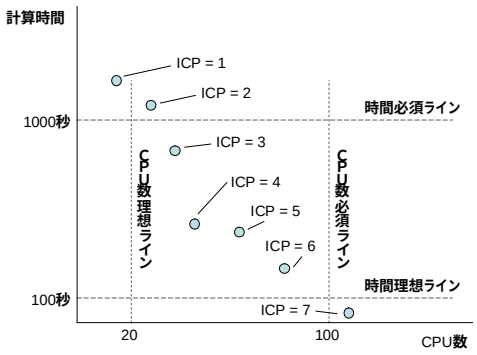


図 2 CPU 数と計算時間の必須ラインと理想ライン例

次に、プログラム 3 に、数値計算ポリシーを実装するスクリプト指示文の例を挙げる。CPU 数の必須ラインを CPU_limit とし、CPU の理想ラインを CPU_best とする。計算時間の必須ラインを time_limit とし、計算時間の理想ラインを time_best とする。

また、必須ライン内のパラメータ選択基準を selection と表し、計算時間を優先する場合は selection = time とし、CPU 数を優先する場合は selection = CPU とし、上記のモデル(*)を与える場合には、selection = min(time/best_time+CPU/CPU_best)と表すこととする。

本プログラムは、元々は、6 行目のライブラリ SOLVER-A を呼び出すのに対し、1 行目から 5 行目まで数値計算ポリシーを記述している。1 行目はポリシーを与えるライブラリ名称、2 行目は RCP として CPU 数を指定し、その必須ラインを 100 個、理想ラインを 20 個としている。3 行目は UAF として実行時間を指定し、その必須ラインを 1000 秒、理想ラインを 100 秒としている。4 行目は選択ポリシーを表し、5 行目は、調整する対象のパラメータを指定している。

＜プログラム3＞

```
1 $ POLICY libname = SOLVER-A
2 $ POLICY RCP = CPU, CPU_limit = 100, CPU_best = 20
3 $ POLICY UAF = time, time_limit = 1000sec, time_best = 100 sec
4 $ POLICY selection = min; time/time_best + CPU/CPU_best
5 $ POLICY tune ICP
6               CALL SOLVER-A (UCP, ICP)
```

4. SR11000向け疎行列ベクトル積に対する適用

本章では、2章で述べた疎行列固有値における疎行列ベクトル積をSR11000向けに実装したプログラムに対し、3章で提案した数値計算ポリシーの定式化を適用した際の効果を予測する。

4.1 数値実験に用いた例題および計算環境

数値実験は、疎行列ソルバのテストとして一般的に使用されるMatrixMarketからダウンロードした表1に示す3種類のデータを利用した。いずれも、structural engineering分野からのデータである。

表1 数値実験に用いた3つの例題

例題	行列名称	次元数(N)	非零要素数(NZ)	疎度(NZ/(N*N))
1	bcsstk39.rsa	46772	1068033	0.0488%
2	ct20stif.rsa	52329	1375396	0.0502%
3	Pwtk.rsa	217918	5926171	0.0125%

また、数値実験に用いた計算機プラットフォーム、コンパイラオプションについて表2にまとめる。

表2 数値実験環境

マシン名称	SR11000(スーパーコンピュータ)
ハードウェアプラットフォーム	Hitachi SR11000 model J1, 1node (16CPU), CPU: Power5 1.9GHz, L1キャッシュ:32KB, L2キャッシュ:1.875MB, L3キャッシュ:288MB, 主記憶:128GB
コンパイラ	Hitachi FORTRAN for SR11000, 01-01
コンパイルオプション	f90 -64 -Os -noscope −parallel (1CPU, 16CPU 共通。実行時、1CPU の場合は %> a.out -F'PRUNST(THREADNUM(1))' 16CPU の場合は %> a.out -F'PRUNST(THREADNUM(16))' となる。)

4.2 適用した必須ラインおよび理想ラインの算出モデル

疎行列ベクトル積の実装について、簡単のため、ワークエリアの与え方を以下の3通りとした。

- ICP = 1　疎行列ベクトル積のワークエリアを採らない。
- ICP = 2　疎行列ベクトル積のワークエリアを4本のベクトル分採る。
- ICP = 3　疎行列ベクトル積のワークエリアを16本のベクトル分採る。

という3パタンのみで変化させることとする。その上で、利用するCPU数および理想ライン、必須ラインの設定、および、選択ポリシーからICPを決定するという自動チューニングが適用可能であるかを以下、検討する。

まず、理想ラインと必須ラインは以下のように設定した。

メモリ量における理想ラインおよび必須ラインは、主にユーザが利用するメモリ量、およびその3倍とした。主に利用するメモリ量は、行列Aの非ゼロ要素数分の8*NZバイト、および固有ベクトルNV本分の容量8*N*NVであるので、メモリ量理想ラインは、

memory_best = 8*NZ + 8*NV*N

メモリ量必須ラインは、主に利用するメモリ量の3倍とした。

memory_limit = 3*memory_best

時間理想ラインおよび時間必須ラインは次のように設定した。時間理想ラインは、SR11000 (16CPU) の対ピーク25%で計算処理が行われたとして、更に、Lanczos法の反復が100回行われたとして設定した。Lanczos法の主たる計算は、5回分の疎行列ベクトル積と、3回分のグラムシュミット直交化から構成される。1回分の疎行列ベクトル積の演算量は2*NZ、1回分のグラムシュミット直交化は2*KMAX*Nである。また、SR11000 (1CPU)の対ピーク性能25%時の処理性能は、1900Mflop/sである。従って、時間理想ラインは、

time_best = 100* (5*2*NZ+3*2*KMAX*N) / (1900Mflop/s*16)

となる。一方、時間必須ラインは、対ピーク性能はそのままだが、並列化が有効ではなかったとし、更に、Lanczos法における反復が200回となったとして設定した。

time_limit = 200 * (5*2*NZ+3*2*KMAX*N) / (1900Mflop/s) = time_best*16*2

なお、下記の実験では、NV=10、KMAX=30として行った。

次に選択ポリシーとしては、短時間優先ポリシー、省メモリ優先ポリシー、そしてその中間ポリシーを設定した。中間ポリシーは、以下の関数Fが最小となるICPを選択することとした。

F＝Memory/memory_best + Time/time_bset

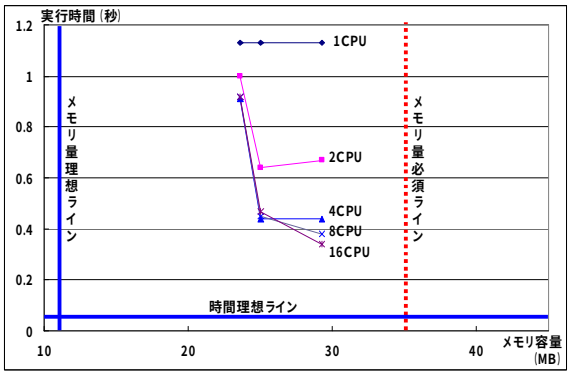
4.3 適用結果

本節では、前節で表現した数値計算ポリシーが選択したパラメータが、他のパラメータに対して有効であるかどうかを評価する。

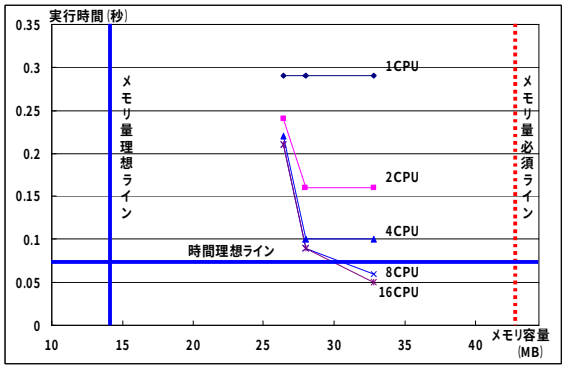
まず、SR11000の1CPU、2CPU、4CPU、8CPU、16CPUにおける実行時間と使用メモリ量を計測し、前節で表したメモリ量の必須ライン、理想ライン、実行時間の必須ライン、理想ラインを計算した。図3の(a), (b), (c)にそれぞれ例題1, 2, 3における結果をまとめる。

例題2では、例題1および3と異なり、8CPUや16CPU実行時に実行時間の理想ラインを下回る実行時間となっている。これは、例題1および例題3では、反復回数が136回、および271回とやや多めであった

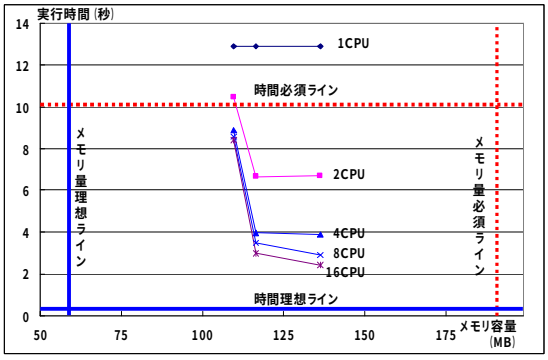
のに対し、例題 2 では、反復回数が 31 回であったためと考えられる。また、例題 3 では、1CPU と 2CPU の一部が必須ラインを超える実行時間となっているが、これは反復回数が 271 回であり、実行時間の必須ライン想定の 200 回の反復回数を超えてしまったためである。メモリ使用量に関しては、いずれの場合も理想ラインと必須ラインの間であり、両者の想定が妥当であることを示している。



(a) 例題1の結果



(b) 例題 2 の結果



(c) 例題 3 の結果

図 3 必要メモリ量－実行時間のトレードオフと理想ラインと必須ライン

表 3 に、前節での 3 つのポリシーを適用した場合に選択されたパラメータ ICP の結果をまとめる。なお、NG は、いずれかも必須ラインを超えた場合である。また、中間ポリシーでは、短時間優先ポリシーの選択パラメータと異なる場合には、何パーセントの計算時間が増加したのか、省メモリ優先ポリシーの選択パラメータと異なる場合には、また何パーセントのメモリ量が増加したのかを示す。

評価の結果、例題 2 の CPU=8 および 16 のケースにおいて、省メモリ優先ポリシーよりも 6.1%のメモリ増で、計算時間が 50%増加するパラメータが選択され、例題 3 の CPU=4, 8, 16 のケースでは、省メモリ優先ポリシーよりも 6.1%のメモリ増で、計算時間を省メモリ優先ポリシーより計算時間が 1.8%, 20.0%, 22.1%増加するパラメータが選択された。

表 3 数値計算ポリシー別のパラメータ ICP の最適値

例題	CPU 数	数値計算ポリシー		
		短時間優先	省メモリ優先	中間ポリシー (対時間/対メモリ)
1	1	ICP=1	ICP=1	ICP=1 (0%/ 0%)
	2	ICP=2	ICP=1	ICP=2 (0%/ 6.1%)
	4	ICP=2	ICP=1	ICP=2 (0%/ 6.1%)
	8	ICP=3	ICP=1	ICP=3 (0%/24.2%)
	16	ICP=3	ICP=1	ICP=3 (0%/24.2%)
2	1	ICP=1	ICP=1	ICP=1 (0%/ 0%)
	2	ICP=2	ICP=1	ICP=1 (0%/24.2%)
	4	ICP=2	ICP=1	ICP=2 (0%/24.2%)
	8	ICP=3	ICP=1	**ICP=2 (50.0%/ 6.1%)**
	16	ICP=3	ICP=1	**ICP=2 (50.0%/ 6.1%)**
3	1	NG	NG	NG
	2	ICP=2	ICP=2	ICP=2 (0%/ 0%)
	4	ICP=3	ICP=1	**ICP=2 (1.8%/ 6.1%)**
	8	ICP=3	ICP=1	**ICP=2 (20.0%/ 6.1%)**
	16	ICP=3	ICP=1	**ICP=2 (22.1%/ 6.1%)**

5. 関連研究

本研究で扱った数値計算ポリシーによるパラメータの選択技術は、広くは、数値計算ライブラリの自動チューニング技術の分野に位置付けられる。

自動チューニングの代表的技術として、国内の事例では、東大および電通大のグループがそれぞれ、I-LIB[1, 2]、ABC-LIB[3, 4]という自動チューニング型ライブラリの研究を行っており、様々な計算機向けに多重ループのアンローリングを自動化するツールを開発している。海外の事例では、テネシー大の Dongarra らが SANS(Self-Adapting Numerical Software)[5]という PC クラスタ向け自己適合型の行列ライブラリ構成方式および、ATLAS[6] という PC 上の高品質なライブラリを発表している。また、カリフォルニア大の Demmel のグループによる PHiPAC[7]という自動チューニング型ライブラリが挙げられる。同じくカリフォルニア大の Frigo らが適合型の FFT に関する研究[8]を行い、FFTW[9]というライブラリを発表している。また、応用分野を限定しつつも更に適合度を高める FFT の研究[10]などもあり、適合型ライブラリの研究はプロトタイプソフトウェアが発表される段階にある。

我々は、自動チューニングのインターフェイス形式[12]、自動チューニングモデルと性能安定化[13, 14]、ユーザの計算精度と処理性能に対する数値計算ポリシーを反映させる入力方式[11]を提案した。本研究報告では、計算資源を調整する数値計算ポリシーを入力とする方式が有効であることを示した。

6. 纏めと今後の課題

本報告では、使用メモリ量と計算時間のバランスを調整する数値計算ポリシー入力型の行列計算ライブラリ方式を提案する。本方式は、使用メモリ量および計算時間に対して、それぞれ理想ラインと必須ラインを与え、両必須ライン以内にあるパラメータ内であって、かつ、両理想ラインに対して与えた選択モデルに基づき最適なパラメータを選択する方式である。

パラメータの設定によって使用メモリ量と計算時間のトレードオフがあり、試行錯誤を必要としていた疎行列固有値ソルバにおいて本方式を適用した。その結果、MatrixMarket の数万次元の例題に対して、メモリ量と計算時間を理想モデルに対してバランスさせるパラメータを選択することができた。

今後の課題を以下、列挙する。

(1) 今回の数値計算において、理想ラインおよび必須ラインの算出に、計算予測モデルを用いた。しかし、例題ごとの反復回数の違いによって大きな差があった。今後、研究[15]などで提案されている、反復回数の大小を予測する技術に関して検討していく。

(2) 今回の数値計算ポリシーの検討範囲は、疎行列固有値ソルバにおける疎行列ベクトル積の並列化パラメータに対してのみであった。今後、疎行列固有値ソルバの他のパラメータ、例えば直交化補助ベクトルの本数や、最大反復回数に対しても検討していく。

(3) 今回の数値計算ポリシーの検討は、疎行列計算のうち、固有値計算におけるパラメータ選択のみであった。今後、連立一次方程式の反復解法などに検討範囲を広げていく。

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。

参考文献

[1] 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.

[2] project I-LIB: http://www.super-computing.org/~kuroda/nadia.html.

[3] 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.

[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] 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.

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

[7] Bilmes, J., Asanovic, K., Chin, C.-W., Demmel, J.: Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97, pp. 340-347 (1997).

[8] M. Frigo, S. Johnson: FFTW: An adaptive software architecture for the FFT, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing 3(1998), pp.1381-1384.

[9] FFTW Homepage: http://www.fftw.org/.

[10] E. Sundararajan, M. Premaratne, S. Karunasekera, and A. Harwood: Algorithmic-Parameter Optimization of a Parallelized Split-Step Fourier Transform Using a Modified BSP Cost Model, Proceedings of the Second Symposium on Parallel and Distributed Processing and Applications(ISPA-2004), Springer LNCS 3358, pp.245-256.

[11] 直野健, 猪貝光祥, 木立啓之：数値計算ポリシーを入力とするベクトル群の直交化ライブラリの構成方法, 情報処理学会論文誌：コンピューティングシステム, Vol.46, No.SIG7(ACS10), pp.35-43(2005).

[12] 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), pp.25-30, 2001.

[13] 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,情報処理学会研究報告, 2002-HPC-91(SWoPP2002), pp. 49-54, 2002.

[14] 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.

[15] 岩下武史, 島崎眞昭：非構造型解析中の ILU 分解前処理付き反復法におけるオーダリングの評価法に関する考察, 日本計算工学会計算工学講演会論文集, Vol.10, pp223-226, 2005.