

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

直野 健 山本 有作
(株)日立製作所中央研究所

要旨

MPP 向け的高速行列ライブラリが多数開発されており、I-LIB などは自動チューニング機能までも備えるようになってきている。本報告では、MPP ライブラリを単一メモリ型インターフェイスからコールできるようなライブラリ設計を目的に、性能を左右するパラメータの分類による自動チューニングの定式化を試みた。また、本定式化に基づき、(1)自動チューニング機能付きの MPP ライブラリの構成方式、(2)各 MPP マシンへのインストール方式、(3)MPP ライブラリ実行時の稼働方式について検討した。

A framework for development of the library for massively parallel processors with auto-tuning function and with the single memory interface

Ken Naono and Yusaku Yamamoto
Hitachi, Ltd., Central Research Laboratory

Abstract

A lot of high performance matrix libraries for massively parallel processors have been developed so far. Some of them contain the auto-tuning facility like I-LIB. In this paper, we aim at designing the library for massively parallel processors whose interface can be called from a single memory type. For that, we formulate a framework of the auto-tuning by classifying the parameters sensitive to the performance. Based on the framework, we describe the auto-tuning process that includes (1) the method of developing the library for the massively parallel processors with auto-tuning function, (2) the method of installation of the library, and (3) the method of using the library.

1. はじめに

これまで行列計算ライブラリは、機能の拡充とともに、各世代のスーパーコンピュータやエンジニアリングワークステーションの性能を最大限に引き出す方向で発展してきた。ベクトル計算機構、スーパースカラー命令、キャッシュ等のメモリ階層、共有メモリ並列でのループ分割、分散メモリ並列でのデータ分散、Cluster of SMP での 2 階層の並列など、性能を引き出す上で非常に多くの機構が発明され、ライブラリはそれを取り込む形で発展した。数学アルゴリズムの発展もあり、性能は飛躍的に高まってきている。

また最近、ATLAS[1]、I-LIB[2]など性能に関わるパラメータを自動的に決定する「自動チューニング機能付き」ライブラリが開発

されるようになってきている。上記のような計算機の機構やアルゴリズムの発展に伴い、ライブラリには多くのパラメータ調整が必要になってきた。例えば ScaLAPACK[3]の場合、データの分散におけるブロック幅や、キャッシュマシンの性能を左右するアンロール段数パラメータなど、調整すべきパラメータが非常に多い。また、それらのパラメータは実行させるマシン、CPU 数、行列サイズ等によって最適値が変わるため、性能を出すには相応の知識と経験が要求される。そこでパラメータを定めてくれる自動チューニング機能によって解決することが望まれている。

一方 Ninf[4]、NetSolve[5]などネットワーク型ライブラリも開発されている。これらはネットワークを介して手元のワークステーションなどから遠隔地のスパコンを利用するた

めのエージェントである。ユーザは自分のワークステーションにインストールする必要がなく、容易に遠隔地のハイ・パフォーマンス計算資源でライブラリが利用可能であり、ライブラリの性能チューニングやバージョン管理に悩まされずに済む。

これらの発展を鑑みると、今後のライブラリに求められる特徴は、MPP や Cluster of SMP などの計算機で高速に稼動するアルゴリズムを備えつつ、いろんなマシン上でも性能を十分に引き出して稼動するよう自動チューニング機能を有し、かつ、ネットワークエージェントからコールできるよう、統一されたシンプルなインターフェイスで利用される、という3つであると考えられる。

本研究では、シンプルなインターフェイスであり、かつ自動チューニング機構を持つライブラリを開発するための効率的なフレームワークについて検討する。報告者らは、既に文献[6]において並列ライブラリの構成方法を提案している。本報告では、本構成方法の基本的な考え方を解説した後、それを応用し、単一メモリ型インターフェイスからコールされる自動チューニング機能を有するMPP型並列ライブラリのフレームワークについて検討する。このライブラリフレームワークを簡単のため「SIMPL」(= Single-memory Interface Massively Parallel Library) と呼ぶことにする。

SIMPLにおいて、MPPライブラリを想定するのは、上記3つの特徴のうち、MPPやCluster of SMPなどの計算機で高速に稼動するアルゴリズムを備える要件からである。また、単一メモリ型インターフェイスからコールされる場合を検討するのは、上記3つの特徴のうち、統一されたシンプルなインターフェイスを満たす要件からである。

本報告ではSIMPLにおける開発方式、インストール方式、利用方式の一連のフレームワークを提示することを目的とする。

2. 自動チューニングの定式化

本節と次節では文献[6]に基づき、インターフェイス設計を鑑みた自動チューニング機構を有するライブラリを開発するための効率的なフレームワークを説明する。

2.1 自動チューニングの例

詳しい定義は後述するが、およそ自動チューニングとは、ライブラリ自身が計算機の性能を最大限に引き出して、ライブラリを実

行するようにパラメータを調整することである。例えば、内積演算でのアンロール段数 M を上手く定めることで性能は大幅に変わることが知られている。

(例1) 内積ループのアンロール段数 $M=2$ の場合

```
DO I=1, N, 2
  A1 = A1 + B(I)      * C(I)
  A2 = A2 + B(I+1)* C(I+1)
ENDDO
A = A1 + A2
```

この M は、マシンのレジスタ数やコンパイラに依存する。また、一般には同じマシン、コンパイラ上でもループ長 N に依存する。以下、マシン、OS、コンパイラなど、ライブラリの実行オブジェクトを準備するセットを単にマシン環境と呼ぶことにする。

上記の内積ループ計算をライブラリ化するため、そのインターフェイスについて検討する。まず、性能に関わるパラメータを全てインターフェイスに記述すると、

Inner_Product(A, B, C, N, M)
.....インターフェイス (D)

となる。このライブラリの性能を最大限に引き出す場合、マシン環境と N を固定するとある固定値 M を定めることになる。内積計算を行いたいユーザにとって、 M の決定は、内積演算という機能とは無関係な操作である。そこで、 M を取り除き、

Inner_Product_User(A, B, C, N)
.....インターフェイス (U)

というインターフェイスにすると、ユーザにとってシンプルな形となる。但し、 M はマシン環境のみからは決まらず、 N の関数となっている； $M = F(N)$ 。

また、この関数 F はマシン環境ごとに定められる。即ち、異なるマシン環境では、 $M = G(N)$ などと違う関数 G で表現することにする。

もし、マシン環境ごとに M を決定する関数 (F や G など) がそれぞれ準備されていれば、ユーザはアンロールという性能チューニングを気にすることなく上記インターフェイス (U) によって内積演算をコールし、どのマシンにおいても十分に高性能な内積演算機能を利用することが可能になる。

M を定める関数 F や G は、ライブラリをマ

シン環境にインストール際に決定できる。もちろん、ハードウェア資源の追加時、OSやコンパイラのバージョンアップの際などマシン環境が変化する場合には構築しなおす必要があるが、ユーザがライブラリを使用するごとに変更する必要はない。

2.2 種々のチューニングパラメータ

次に、固有値計算における実対称密行列の三重対角化におけるチューニングパラメータに関して説明する。三重対角化は、主にリダクションと行列ベクトル積から構成され、それぞれが $(2/3)N^3$ の演算量である。

リダクション部分は、高効率な計算アルゴリズムとしてブロック化ハウスホルダ法が知られている。これは、MB 回の部分ハウスホルダ変換と行列ベクトル積を行った後に MB ブロック分のハウスホルダ変換をいっしょに行う手法である。

チューニングパラメータは第一に、ブロック化ハウスホルダ法でのブロック幅 MB である。

次に、MB ブロック分のハウスホルダ変換における 3 重ループ (例 2-1) の各アンロール段数もチューニングパラメータとなる。

(例 2-1)

```
do M = 1, MB
  do J = 1, N
    do I = 1, J
      A(I, J) = A(I, J) -
        ( V(I, M)*W(J, M) + W(I, M)*V(J, M) )
    enddo
  enddo
enddo
```

これらを M1、M2、M3 とする。但し M1 は MB を超えないという制約となる。

そして、行列ベクトル積 (例 2-2) の各アンロール段数もチューニングパラメータとなる。

(例 2-2)

```
do J = 1, N
  do I = 1, J
    W(I, M) = A(I, J)*Q(J)
  enddo
enddo
```

それぞれアンロール段数を M4、M5 とする。他にも内積演算などがあるが演算量は無視で

きるのでここでは考えないことにする。最後のチューニングパラメータとして、行列 A の整合寸法 NA が挙げられる ; A(NA, N)。

このように実対称密行列の対角化におけるチューニングパラメータは多く、これらをマシン環境毎にユーザが調整するのは非常に手間がかかる。性能チューニングパラメータを全てインターフェイスに書くと、

Tridiagonal(A, N, NA, D, D1, MB, M1, M2, M3, M4, M5)

.....インターフェイス (D)

となる。ここで N は行列 A の次元数、D(N) は対角化後の対角成分。D1(N-1)は副対角成分とする。

このうち、マシン環境が定まると MB、M1、M2、M3、M4、M5、NA が N に応じて定められる。よって、{MB, M1, M2, M3, M4, M5, NA} = F(N) となる。ここで { } で書いたのはセットという意味で用いた。これらが F によって自動的に定められると、ユーザ向けのインターフェイスは、

Tridiagonal_User(A, N, D, D1)

.....インターフェイス (U)

となり、非常にシンプルな形となる。

通常のマシン環境では、M1、M2、M3、M4、M5 などのアンロール段数は、非常に小さい場合を除いて、次元数 N にはよらず、マシン環境のみに依存すると考えられる。しかし N に対する定数項で表現されたと考え、{MB, M1, M2, M3, M4, M5, NA} = F(N) と書くことにする。

マシン環境毎に F のような関数が定まり、N から最適な {MB, M1, M2, M3, M4, M5, NA} が与えられインターフェイス (D) のライブラリが稼動すると、ユーザは上記 (U) のようなインターフェイスでライブラリコールを行えばパラメータ調整を気にすることなく、任意のマシン環境で高性能な対角化機能を利用することができるようになる。

なお F のような関数は、本ライブラリをマシン環境にインストールする際に定められるとする。

2.3 自動チューニングの定式化

上述のように、ライブラリソフトウェアには多くのパラメータがある。そこで、上記の例の N、M、MB、M1、NA など、性能を大きく左右するスカラー値を、インターフェイス設計の観点から分類し、自動チューニング

の定式化を行う。

まず、性能を大きく左右するパラメータ CP、UCP、ICP を定義する。

Definition 1 (CP : Critical Parameter)

スカラー値 M が CP である、とは、 M がライブラリの性能に影響を与える場合とする。

Definition 2 (UCP : Users' Critical Parameter)

スカラー値 M が UCP である、とは、 M が CP であり、かつ、ライブラリのインターフェイスに現れる場合とする。

Definition 3 (ICP : Internal Critical Parameter)

スカラー値 M が ICP である、とは、 M が CP であり、かつ、ライブラリのインターフェイスに現れない場合とする。

CP の定義から明らかなように、実行時間 Exec_Time は、CP の関数、つまり UCP と ICP の関数である。

$$\begin{aligned}\text{Exec_Time} &= \text{Exec_Time}(\text{CPs}) \\ &= \text{Exec_Time}(\text{UCPs}, \text{ICPs}).\end{aligned}$$

自動チューニングとは実行時間を最小にするよう上手く CP を定めることである。ところが、CP のうち UCP は性能と無関係にユーザの意思で定められる。一方、ICP は UCP によって影響を受ける場合がある。例 1 では最適な M は N によって変わるし、例 2 では MB は N によって変わる。従って、自動チューニングとは、UCP の制約条件の上で ICP を定めることと言える。

Proposition 1 (A framework of auto-tuning)

自動チューニングとは、以下の最適化問題 (OPT) であると定式化できる。

(OPT)

Decide ICPs such that

$\text{Exec_Time} = \text{Exec_Time}(\text{UCPs}, \text{ICPs})$
attains minimum for given UCPs.

言い換えると、自動チューニングとは、ユーザにとって本来は意味がないものの、性能上クリティカルな影響を及ぼすパラメータを適切な値に定める行為を指す。例 1 や例 2 の F などは、 Exec_Time で定まる上記の最適化問題を解くことで得られる UCP から ICP を決定する関数であると言える。

一般に、個別ライブラリ毎に実行時間を予

測するモデル $\text{Exec_Time}(\text{UCPs}, \text{ICPs})$ を構成すると UCP から ICP を定める関数が決定できる。 Exec_Time を構成するには、アルゴリズムからモデルを導出するか、あるいは、実際の実行でのデータから補間して計算するモデルを構築する。いずれも、 $\{\text{UCPs}, \text{ICPs}\}$ とそれに対応する実行時間の測定結果からモデルを構築する必要がある。本報告では、その具体的な実装と実証実験は今後の課題として、 Exec_Time が何らかの手段で構築されるという前提で以下の議論を行う。

3. 自動チューニングライブラリの構成・インストール・利用の手順

前節の自動チューニングの定式化に基づき自動チューニングライブラリの構成方法、インストール・利用の手順について説明する。

3.1 自動チューニングライブラリの構成

ライブラリ構成は以下の手順で行う。

【手順 1】

1. CP を抽出する。全ての CP を記述するインターフェイス (D) を作成する。
2. CP を UCP と ICP に分別する。
3. CP による実行時間予測モデルを作成する。

1. の CP はマシン依存のもの、コンパイラ依存のもの、アルゴリズム依存のもの、あるいはそれらの組み合わせのもの等がある。よって、漏れのないよう、多くの種類のマシン環境での経験から集積する必要がある。

2. の UCP と ICP の区別はユーザによって異なる。例えば分散メモリを意識したライブラリを記述する人はプロセッサ数 NPU を UCP とするが、単一メモリ型インターフェイスでコールする場合には ICP としたいだろう。従って、場合に応じて UCP の基準を設けたり、あるいはデフォルトの UCP を用意して場合に応じて UCP を ICP 化させたりあるいはその逆にできたりする機能も必要となる。なお、「単一メモリ型インターフェイス」とは、利用するメモリ空間が複数ではなく単一であるようなインターフェイスを指すものとする。

3. は、 $\text{Exec_Time} = \text{Exec_Time}(\text{UCPs}, \text{ICPs})$ のモデルを用意する。具体的には未定係数をもつ形式が一つの方法と思われるが、アルゴリズムやマシン環境によって実行時間の状況は大きく変わるため、今後の検証が必要となる。本報告では、これが可能という前提

で議論を進める。

3.2 インストールの手順

インストールでは、Exec_Time を決定する。モデル化の実現方式の検証は必要であるが、Exec_Time が未定係数付きの関数で表現可能だった場合、手順は以下のようになる。

【手順 2】

4. いくつかの UCP と ICP のセットで実行時間を計測する。
5. 複数の実行時間の計測から、Exec_Time の未定係数を決定する。

4. のために予めマシン環境上のタイマーを定義しておく必要がある。また、どの値の UCP や ICP を用意しておくべきかは、今後検討すべきである。5. では、例えば未定係数の決定方法は最小 2 乗法などによって計算されることになる。

3.3 ライブラリ利用の手順

ライブラリ利用の手順は以下のようになる。

【手順 3】

6. ユーザは UCP を決定する（ソースレベル）。
7. 自動チューニング機能が UCP から Exec_Time が最小になる ICP を計算する（オブジェクトレベル）。
8. 定められた UCP、ICP によってライブラリが実行される（実行命令レベル）。

6. では、インターフェイス（U）にあるパラメータを決定する。7. では、最適化問題（OPT）を解いて UCP から ICP を計算する。8. は決定された ICP と元々与えられた UCP によってインターフェイス（D）の形式でライブラリが実行される。

4. 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機能付き並列ライブラリの構築方法

本節では、前節の結果を応用し、本報告のタイトルにある通りの、単一メモリ型インターフェイスを有しつつ、MPP 環境において自動チューニング機能を有するライブラリ、即ち「SIMPL」の構築方法について説明する。

4.1 チューニング対象パラメータ

単一メモリ型インターフェイスから、MPP 並列ライブラリをコールする場合、特徴的な ICP として行列の分割幅 LB や MPP 環境のプロセッサ数 NPU が挙げられる。両者とも分散メモリ環境で始めて登場するパラメータであり、単一メモリ型インターフェイスには登場しないゆえ ICP である。

例えば正方行列の乗算 $A=B * C$ では、CP として、行列サイズ N、行列の整合寸法 NA、分割ブロック幅 LB、プロセッサ数 NPU 等が挙げられる。単一メモリ型インターフェイスで必要とされるパラメータは N のみであるので、これが UCP であり、残りは ICP である。なお分割幅 LB は行方向と列方向の 2 つあるが、ここでは同じとしておく。

4.2 プロセッサ数 NPU の扱い

NPU は ICP であるが、特別な扱いをする必要がある。第 2 節で定式化した「自動チューニング=最適化問題(OPT)」では、UCP を与えた上で ICP を解く、といったように、ICP は UCP から決定される属性となっている。しかし、プロセッサ数 NPU は、ライブラリの実行時に初めて利用可能な数が特定されるため、行列サイズ N などの UCP のみから決定できず、実行時にユーザの指示が必要となる。従って、最適化問題(OPT)の枠組みのみでは自動チューニングを定式化することができない。

そこで、NPU = 1 の場合の部分最適化問題 (OPT-1)、NPU = 2 の場合の部分最適化問題 (OPT-2) という風に、各 NPU 毎に個別の部分最適化問題を定式化しておき、実行時には利用可能な NPU の範囲で各部分最適化問題の結果の中から最適な ICP (NPU を除く) を決定する。

即ち前節の【手順 1】と【手順 2】は同様であるが、【手順 3】は以下のように変わる。

【手順 3 : SIMPL】

6. ユーザは UCP を決定する。
（ソースレベル）。
7. 自動チューニング機能が UCP から NPU を固定させた部分最適化問題 (OPT-1), (OPT-2), ..., (OPT-n) を解き、ICP-1, ICP-2, ..., ICP-n を求める。
（オブジェクトレベル）
9. 実行直前に使用可能な NPU の範囲が決定され、その中で最適な ICP-o と UCP によってライブラリが実行される。

5. 実際のライブラリ構築に向けた考察

ライブラリのソースコードあったとして、おおよそ CP は明らかになるが、どこまでが UCP でどこからが ICP かについては、ユーザによって異なる。前節では SIMPL という利用状況を仮定して LB や NPU を ICP と定めたが、極端には、行列サイズ N ですらどうでもよく、ある程度の適度な精度で計算結果が得られれば、N が 1000 でも 10000 でもどちらでもいいというユーザもいるかもしれない。CP はできるだけ多くを用意しておき、ユーザはその場合、場合に応じてどこまでを DCP でどこからは ICP と決定するようにすべきであろう。

従来のマシンベンダ提供のライブラリでは、ユーザに対する便利さを優先させ、ICP にあたる部分はインターフェイスに出さずにマシンベンダがある固定値を与える場合が多かった。また、任意のマシンでの稼動を目的とするフリーソースのライブラリでは、マシン毎に性能を調整できるように ICP もインターフェイスに登場させる場合が多かったと考えられる。本研究のようなライブラリ開発方式を用いれば、マシン環境毎に異なる ICP の決定の煩わしさを回避しつつ、どのマシン環境でも統一的なインターフェイスで、しかもマシンの性能を十分に引き出した形でライブラリ利用が可能になると思われる。

本報告の方式に従うと、ライブラリ開発者は、CP を全てインターフェイス (D) に含めなくてはならなくなる。するとこれまで定数としていた一部の CP が一般的な値として機能検証する必要があるため、チェックテストを行う回数が増えてしまう。

もちろん、あるマシンでチェックを済ませている場合に別のマシンで結果不正を生じたとすると、おおよそマシン環境自体にバグが含まれていると判断できるので、どこかのマシンで一通りチェックができればよく、その意味では、マシン毎に異なるライブラリを作り続け、それぞれチェックテストを行うよりは効果的かもしれない。ただ、チェックテスト回数はライブラリ開発上、ネックとなるのでこれを少ない回数で行う工夫は今後必要と考える。

また、本報告では、「性能」面と「インターフェイス」面のみを考慮に入れてパラメータの決定を行う方式を検討したが、実際のライブラリ実行には、利用可能メモリ容量、ディスク容量、あるいはマシン環境使用料金などの属性があり、これらも含めて調整する必要がある。例えば、最適化問題(OPT)や(OPT-i)のみでパラメータを決定すると使用料金の意

味では効率が悪いかもしれない。「そこそこの性能」で「そこそこの使用料金」で計算可能なようにユーザが調整できるよう、Exec_Time のみではなく、Exec_Money なる評価関数を用意して最適化問題のセットを解く必要があると思われる。

6. おわりに

本報告では、自動チューニングライブラリを開発する方式の検討を行った。本方式では、性能を大きく左右するパラメータを CP (= Critical Parameter) と定義し、このうち、インターフェイスに現れるものを UCP、現れないものを ICP と定義した。その上で、自動チューニングを、「UCP を制限した上での ICP を最適化する問題」と定式化を行った。

また本方式を SIMPL というケースに応用し、ICP として扱われるパラメータの中でプロセッサ数 NPU は、実行時に数の制限が定められるという性質上、他の ICP と異なる扱いをする必要があることを明らかにした。

今後、本方式に基づき【手順 2】で記述したような実行時間の関数 Exec_Time のモデル化を検証する必要がある。本モデル化は、これまでの並列計算機上の性能予測研究の成果を利用することで十分実現可能と考えられる。もちろん、従来のライブラリ開発と異なり、各ライブラリ機能に対して Exec_Time を準備するという手間が増えてしまう。しかし、ネットワークからライブラリを利用する場合、実行前に実行時間を把握する必要はあると考えられるため、今後は必須の機能になると思われる。

参考文献

- [1] ATLAS project : <http://www.netlib.org/atlas/index.html>
- [2] project I-LIB : <http://pi2.cc.u-tokyo.ac.jp/~katagiri/nadia.html>
- [3] ScaLAPACK project : <http://www.netlib.org/scalapack/index.html>
- [4] Ninf project : <http://ninf.etl.go.jp/>
- [5] NetSolve project : <http://www.cs.utk.edu/netsolve/>
- [6] 直野 健, 山本有作, 伊藤 智 : 特開 2000-276454 ソフトウェアの構成方法, 出願日 1999 年 3 月 26 日。