AutoTuned-RB:再帰 BLAS ライブラリの自動チューニング方法

電気通信大学　大学院情報システム学研究科

情報ネットワーク学専攻

並列処理学講座

0351015　木下　靖夫

指導教員

弓場 敏嗣　　本多 弘樹　　伊藤 秀一

2005 年 1 月 31 日　提出

目次

第 II 部　AutoTuned-RB

図目次

表目次

1 序論

1.1 背景

大規模行列演算を必要とする科学技術計算においては, 高性能な数値計算ライブラリの使用は効果的な手段である. もしユーザがライブラリを計算カーネルとして使用しない場合, 高度なプログラミング技術と実装知識が必要とされる. 現在, 数値計算ライブラリではこれらの計算カーネルが標準化され, 基本線型代数計算副プログラム BLAS (Basic Linear Algebra Subprograms) として知られている. BLAS をライブラリとして用いれば, 簡易にプログラミングが可能になり, 高性能計算がもたらされる. ところが, 提供される BLAS などの数値計算ライブラリのなかには, 性能に関するパラメタの設定を必要とするものがある. このパラメタの設定を誤ると計算性能が劣化する. これを回避するため, 適切なパラメタを設定する必要がある. この性能に関するパラメタ設定を自動的に行うのが, 数値計算ライブラリにおける自動チューニングである.

数値計算ライブラリにおける自動チューニングは, 以下の 2 つに分類できる. まず, ライブラリを新しいマシン環境にインストールする時に行うインストール時型である. もう一方は, ライブラリを実行する時に行う実行時型である. インストール型のライブラリは, PHiPAC[7], ATLAS[4], FFTW[5] が知られている. インストール時と実行時型を融合したライブラリとしては, I-Lib[6]が知られている. また片桐らの FIBER[8]は, 上記 2 種のインストール型, 実行時型に加え, 新たにユーザによるパラメタ設定後に自動チューニングを行う実行起動前型を導入している.

これら自動チューニングソフトウェアにおけるパラメタチューニングは, インストールされる条件（指定される問題サイズ）上では高速化が達

成されるが，それ以外の問題サイズでは性能が劣化する問題がある．それに加えて，数値計算ライブラリには，最適化された性能に再現性があり，かつ維持されるという，性能安定性が重要となる．

1.2 研究の目的

本研究は，前述の性能安定性を達成するために，まずチューニングに必要な問題サイズ（サンプリング点と呼ぶ）を従来のように固定せずに複数とる方式を提案する．次に，対象の BLAS を再帰化して実装する方式[1]を採用する．この BLAS の再帰実装により，階層メモリ型を考慮した再帰段数指定ができるようになり，さらに採用した BLAS が性能不安定でも，それを安定化できる．すなわち，性能の安定性を保証できるようになる．さらに，多種多様な並列計算機上で高性能を実現するため，現在多くの並列計算機で利用できる Posix thread を用いた並列化を行う．この方式を，AutoTuned-RB (Auto-Tuned Recursive BLAS) と呼ぶ．我々は，AutoTuned-RB の自動チューニングによる性能向上および性能安定性の効果検証を行うことを本論文の目的とした．

1.3 本論文の構成

本論文の構成を以下に示す．まず，本論文は 2 部構成となっている．第Ⅰ部では再帰 BLAS における自動チューニングに関して記述する．第Ⅱ部では AutoTuned-RB における自動チューニング関して記述する．

第 I 部は 2 章から 4 章で構成されている．2 章では，行列-行列積演算に

おける BLAS の再帰化アルゴリズムについて述べる. 3 章では, 再帰 BLAS の実装方式の説明を行う．4 章では，再帰 BLAS の性能評価を行う.

第Ⅱ部は 5 章から 7 章で構成されている．5 章では，ATLAS で採用されている自動チューニング機構の説明を行う．6 章では，AutoTuned-RB の自動チューニング手法を提案し，そのインストールの流れを説明する．7 章では，提案する AutoTuned-RB の自動チューニング効果の評価として，ATLAS BLAS との性能比較を行う.

最後に 8 章で、この論文のまとめを行う.

第Ⅰ部
再帰 BLAS

2 BLAS(Basic Linear Algebra Subprograms)

2.1 概要

BLAS には，計算する線形代数演算に応じて大きく以下の 3 つのライブラリがある．

(1)　レベル 1 BLAS
　　ベクトル－ベクトル演算

(2)　レベル 2　BLAS
　　行列－ベクトル演算

(3)　レベル 3BLAS
　　行列－行列演算

これらの 3 つのライブラリを組み合わせて利用することによって，連立 1 次多項式、線形最小二乗問題、固有値問題、特異値分解などの様々な問題を解くことが出来る．実際に，ライブラリは C や Fortran などのプログラムで記述されており，行列およびベクトルに関するデータを BLAS の関数の引数とすることで，簡単に計算を実行することができる．

2.2 GEMM

GEMM は，行列‐行列演算（レベル 3 BLAS）を行うライブラリ･インタフェースの 1 つである．具体的には入力行列 A，B，C に関して，行列積は以下の式(１)の計算を実行する．A×B の行列積を行い，行列 C を足したものを行列 C に入力するものである．

$$C \leftarrow \alpha op(A) op(B) + \beta C \qquad (1)$$

ここで op(A)において，行列 A，転置 A^T，複素共役 A^H の選択が可能である．また α，β は倍精度の浮動小数点型であり，行列積演算の係数を表している．図 1 に関数の引数を示す．頭文字の D は，行列データが double 型であることを意味している．

DGEMM (TransA,TransB,M,N,K,a,A,lda,B,ldb,b,C,ldc)

TransA:行列 A が，ニュートラルなら“n”，転置なら“t”(char 型)
式(１)における op(A)

TransB:行列 B が，ニュートラルなら“n”，転置なら“t”(char 型)
式(１)における op(B)

M:行列 A，C の行の次元数(int 型)

N:行列 B，C の列の次元数(int 型)

K:行列 A の列の次元数，行列 B の行の次元数(int 型)

a:上の式(１)における定数 a(double 型)

A:行列 A の要素の配列ポインタ(double 型)

lda:行列 A の第一次元要素数(int 型)

B:B 行列の要素の配列ポインタ(double 型)

ldb:行列 B の第一次元要素数(int 型)

b:上の式(１)における定数 b(double 型)

C:行列 C の要素の配列ポインタ(double 型)

ldc:行列 C の第一次元要素数(int 型)

図 1　DGEMM の引数

2.3 再帰 BLAS

Gustavson らが提案した再帰 BLAS[1]は，BLAS の演算を再帰により部分小行列に分割し，その分割された部分小行列単位で演算をする．このことにより，データ参照の局所化（キャッシュブロッキング）を促進する手法である．すなわち，階層キャッシュメモリを有する計算機上で高速計算を行うことを可能とする．またコレスキ分解等の線形代数演算も再帰実装可能である．

2.4 再帰 GEMM

DGEMM の頭文字 D は，行列のデータ型が double 型を使用していることを表している．本論文で示す再帰 BLAS の処理は，2 分木構造で表現できる（図 2)．図 2 では，各節はタスクを表している．木の根においては，1 つのタスクから 2 つのタスクに分かれる. 木の葉に向かうにつれて, 各々のタスクが 2 つのタスクに分割されていくので，末端節の数は式(２)となる．

$$(\text{末端節の数}) \quad = \quad 2^{L}\text{（L：再帰段数）} \qquad (2)$$

末端葉部(2 分木の最先端)で BLAS をコールする．たとえば，図 2 中の処理①の BLAS コールあと，②が実行され，その処理のあと，再び末端葉の BLAS コール③がなされる．図 3 に，再帰 DGEMM の Gustavson アルゴリズムの概略を示す．図 3 より，C：M×N，A：M×K，B：K×N の各行列について，再帰分割を制御するパラメタは m，n および k である．

このパラメタは，以下の 3 つの工程で分割される．

（1） m は行列 A，B の行で 2 分割

（2） n は行列 B，C の列で 2 分割

（3） k は行列 A の列，行列 B の行を 2 分割

ここで，末端葉の総数は式(３)で与えられる．分割された行列 A，B，C は小行列 Ab，Bb，Cb と呼ぶ．

$$2^L = n \cdot m \cdot k \qquad (3)$$

具体的には，各パラメタ m，n，k を用いて，以下のように再帰呼び出しされる．

（1） m=max(m,n,k)のとき：
m1 = n / 2 とし，この分割行列 2 つを再帰呼び出しする．

（2） n=max(m,n,k)のとき：
n1 = m / 2 とし，この分割行列 2 つを再帰呼び出しする．

（3） (１),（２）以外のとき：
k1 = k / 2 とし，分割行列 2 つを再帰呼び出しする．

ここで，max(a,b,c)は，a，b，c の中で最大値を返す関数である．また，図 3 において，mb=M/m，nb=N/n，kb= K/k で，末端節，末端葉における行列

サイズに対応している．

再帰の終了条件は，上記の分割処理により n=m=k=1 になったときであり，この DGEMM がコールされる．この再帰葉部の計算が終了すると親節に戻るが，この親節では，再び再帰呼び出しがなされる．ここで注意すべきは，mb，nb，kb で指定される行列のデータ量が，キャッシュサイズを超えないように再帰段数が取られることである．再帰 BLAS は，末端葉を並列化することによって，キャッシュサイズを考慮した並列化を実現できる．

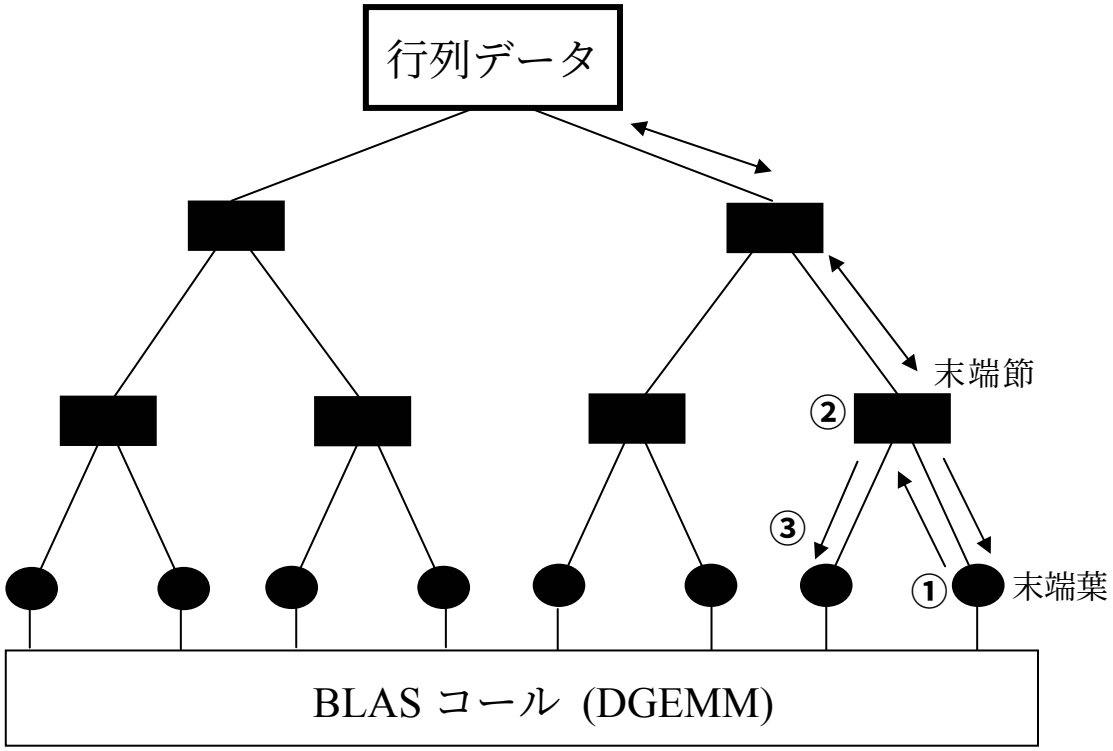


図 2　再帰 DGEMM の処理

```
function [C(1:M,1:N)] = RB-GEMM(C(1:M,1:N),
                        A(1:M,1:K),B(1:K,1:N),m,n,k)
if m = 1 and n = 1 and k = 1 then
    DGEMM(C(1:M,1:N),A(1:M,1:K),B(1:K,1:N));
else if m = max (m,n,k) then
    m1 = m/2;       m2 = m - m1;
    RB-GEMM(C(1:m1・mb,1:N),
            A(1:m1・mb,1:K),B(1:K,1:N),m1,n,k);
    RB-GEMM(C(m1・mb+1:M,1:N),
          A(m1・mb+1:M,1:K),B(1:K,1:N),m2,n,k);
else if n = max (m,n,k) then
    n1 = n/2;       n2 = n - n1;
    RB-GEMM(C(1:M,1:n1・nb),A(1:M,1:K),
                B(1:K,1: n1・nb),m,n1,k);
    RB-GEMM(C(1:M, n1・nb+1:N),A(1:M,1:K),
                B(1:K, n1・nb+1:N),m,n2,k);
else
    k1 = k/2;       k2 = k - k1;
    RB-GEMM(C(1:M,1:N),A(1:M,1:k1・kb),
                B(1: k1・kb,1:N),m,n,k1);
    RB-GEMM(C(1:M,1:N),A(1:M, k1・kb+1:K),
                B(k1・kb+1:K,1:N),m,n,k2);
end
```

図 3　再帰 BLAS の Gustavson アルゴリズム

3 再帰 BLAS の設計と実装

3.1 設計方針

前節で述べた再帰 BLAS を，1CPU マシン，SMP マシンに C 言語を用いて実装をした．図 3 のアルゴリズムでは，各々の行列 A，B，C および小行列 Ab，Bb，Cb を引数としている．図 4 に，実装した再帰 BLAS の擬似コードを示す．Gustavson アルゴリズムからの変更点を以下に示す．

(1) DGEMM のスレッド内処理
DGEMM は，サブルーチン Routine 内でコールしている．Routine はスレッドコールによる処理を行っている．スレッドを管理するための変数 *threadnum を追加した．

(2) グローバル構造体による共有データ
スレッドコールでは引数を一つしか与えられないので，グローバル構造体 Cbuf を定義し，引数を渡した．

(3) 行列データの再帰処理
再帰処理における小行列 Ab，Bb，Cb の引渡しを，分割した小行列の先頭位置を示す Astate，Bstate，Cstate に変更した．

これらの変更をもとに，1CPU マシン，SMP マシンの両方に再帰 BLAS の実装を行った．

```
Recursive(Cstate, Astate, Bstate, m, n, k, M, N, K, *threadnum){
   if(m==1 && n==1 && k==1){
      if(pthread_create(&tid[*threadnum], NULL,
                  (void*(*)(void*))Routine,&cbuf)
}else if( m == max(m,n,k) ){
      m1=m>>1; m2=m-m1; mb=M/m; mm=mb*m1;
      Recursive(Cstate, Astate, Bstate, m1, n, k, mm, N, K, threadnum);
      Recursive(Cstate+(mb*m1)*Nmax, Astate+(mb*m1)*Kmax,
                  Bstate, m2, n, k, (M-mm), N, K, threadnum);
else if( n == max(1,n,k) ){
      n1=n>>1;n2=n-n1;nb=N/n;nn=nb*n1;
      Recursive(Cstate, Astate, Bstate, m, n1, k, M, nn, K, threadnum);
      Recursive(Cstate+(nb*n1), Astate, Bstate+(nb*n1),
                  m, n2, k, M, (N-nn), K, threadnum);
}else{
      k1=k>>1; k2=k-k1; kb=K/k; kk=kb*k1;
      Recursive(Cstate, Astate, Bstate, m, n, k1, M, N, kk, threadnum);
      Recursive(Cstate, Astate+(kb*k1), Bstate+(kb*k1)*Nmax,
                  m, n, k2, M, N, (K-kk), threadnum);
}
      return 0;
}
```

図 4　スレッドコールを用いた再帰 BLAS プログラム

3.2 1CPU マシンにおける実装手法

ここでは，1CPU マシンにおいての処理の流れを述べる．図 5 では処理をプロセス，スレッドの 2 つに分けて表記してある．プロセスとスレッドの処理の内容は以下の通りである．

- プロセス
  行列 A，B，C の入力などの逐次処理

- スレッド
  小行列 Ab，Bb，Cb のコピー，行列計算などの並列化可能な処理

プロセス処理の流れは以下の 4 段階で行われる．

(1) 行列 A，B，C の初期化
計算行列 ABC をグローバル変数で定義し，malloc と初期値の入力

(2) 再帰分割処理
行列 A，B，C の再帰分割処理，スレッドコール

(3) 計算終了通知
スレッド内での計算終了をプロセスへ通知

(4) 計算結果の出力
行列 C の出力

再帰分割数に応じて(２), (３)を繰り返している. またスレッド処理では, 以下の 3 つの処理を行っている.

(a) 小行列 Ab，Bb，Cb のコピー
再帰によって分割された小行列のコピー

(b) DGEMM コール
小行列 Ab，Bb，Cb の行列積

(c) 計算結果 C の足し込み
計算によって得られた小行列 Cb を，行列 C への足し込み

(a)，(b)，(c)では，スレッドとプロセスを同時に処理させないためにスレッド内処理を行っているときはプロセスを停止している．それゆえに，立ち上がっているスレッド数は常に 1 つである.

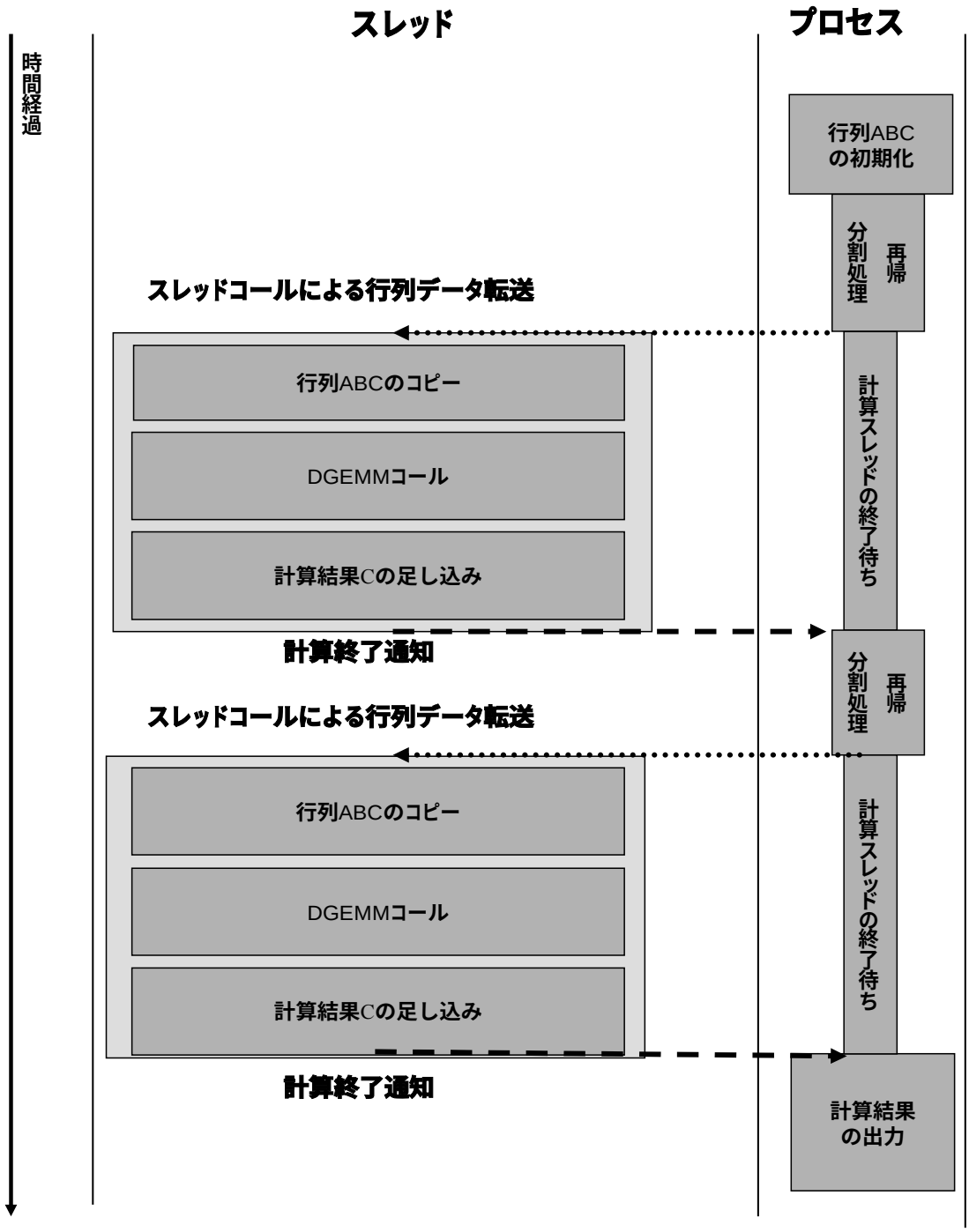


図 5　1CPU マシンにおける処理フロー

3.3 SMP マシンにおける実装手法

ここでは，SMP マシンにおける再帰 GEMM の処理の流れを述べる．図 5 に示すように，4CPU で構成される SMP マシンを例に処理を説明する．再帰段数は 2 で，末端葉数は 4 である．処理の流れは，行列の初期化から始まり，行列の結果を出力する所で終わる．また，末端葉はスレッドによる処理を行う．プロセスでは，以下の 5 つの処理を行う．

(１) 行列 A，B，C の初期化
計算行列 A，B，C をグローバル変数で定義し，malloc と初期値の入力

(２) 同期変数の初期化
スレッド間でバリア同期を取るための変数の初期化

(３) 再帰分割処理
行列 A，B，C の再帰分割処理，スレッドコール

(４) 計算スレッドの終了待ち
計算スレッドが全て終わるまで待機

(５) 計算結果の出力
行列 C の出力

1CPU マシンと同様に，再帰分割数に応じて(３)，(４)を繰り返している．

スレッド処理では，1CPU マシンと同様の処理を行っているが，スレッドースレッド間とスレッドープロセス間では，以下の二つの違いがある．

(a)　スレッドコール後は再帰分割を継続

(b)　小行列 Cb の行列 C への足し込み際にバリア同期

(a)は，複数のスレッドを立ち上げて並列処理している．(b)は，行列 C への足し込みの際に，複数のスレッドからの同アドレスへのアクセスによるデータ入力ミスを防ぐためである．

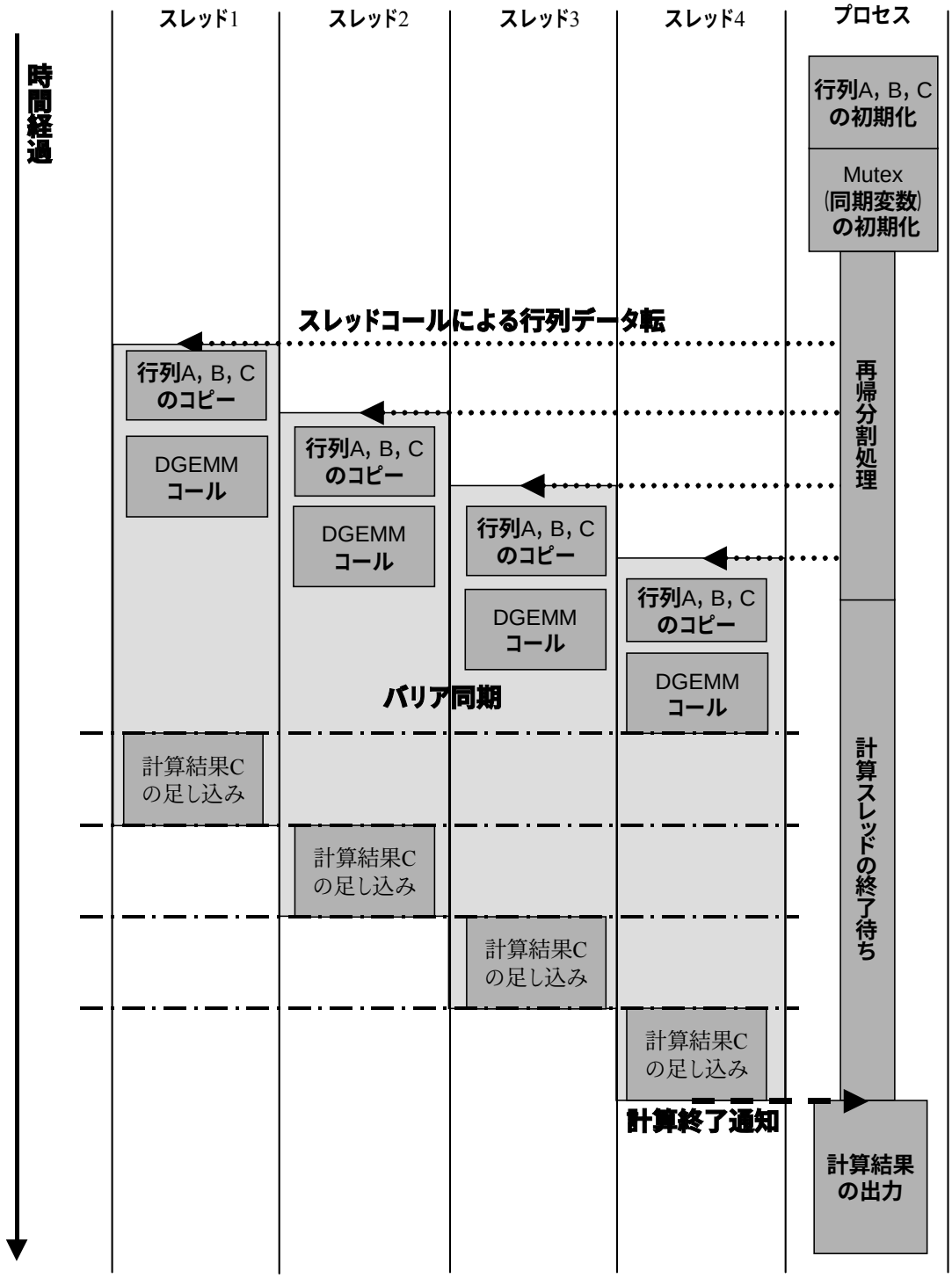


図 6　SMP マシンにおける処理フロー

3.4 再帰 GEMM のオーバヘッド

ここでは，2CPU における再帰 GEMM のオーバヘッドを測定した．(測定環境は表 1 の Opteron を参照のこと．) このオーバヘッドとは，以下の４つの処理時間である．

(1) 再帰分割
行列の再帰分割処理

(2) 計算結果 C の足し込み
再帰 GEMM で計算された小行列 Cb をグローバル型の行列 C への足し込みをスレッド間でバリア同期

(3) malloc
スレッド内で使用する小行列の malloc

(4) DataCopy
グローバル型の行列データを小行列にコピー

図 7 は縦軸が全処理時間をオーバヘッドの割合を示す．図 7 より，DataCopy が全オーバヘッドの約 60%を占め，計算結果 C の足し込みが約 40%を占めていることがわかる．図 7 より，再帰分割，malloc は，見えないくらい小さく，1%未満という結果が得られた．また，行列サイズが小さくなるにつれてオーバヘッドの割合が増え，性能が劣化することがわかる．

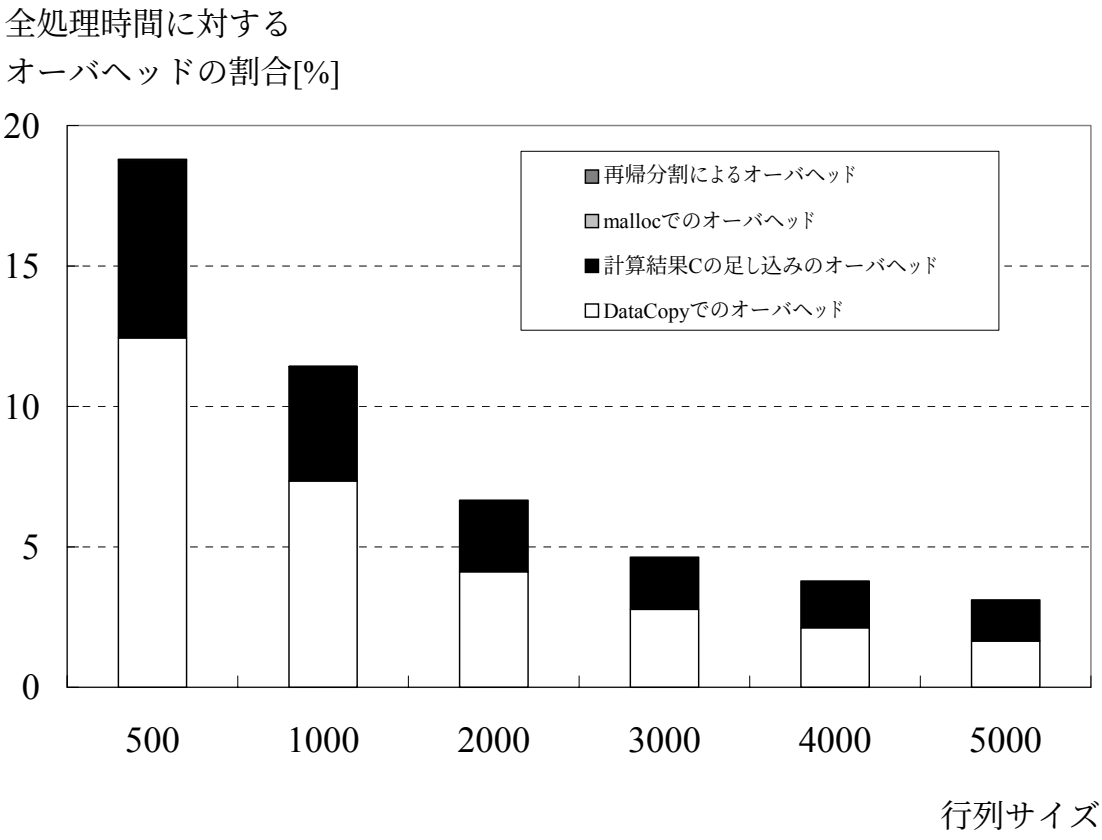


図 7　SMP マシンにおける再帰 BLAS の
オーバヘッドの割合

4 再帰 BLAS の性能評価

4.1 評価環境

本章では，再帰 BLAS の性能を評価する．性能評価では，並列処理学講座の 1CPU の PC-SOFTEK（以下 Pentium4），2CPU の OPT01（以下 Opteron），電気通信大学総合情報処理センターの 16CPU の SGI Origin3400（以下 Orgin3400）による 3 種類のマシンを使用する．プログラミング言語は C 言語を使用している．評価に用いた BLAS ライブラリは，ATLAS(Ver.3.4.2)がチューニングを行った DGEMM(以下 DGEMM_P)，スレッド間通信は Posix Thread(以下 Pthread)である．また，行列は密な正方行列で，各要素野データは乱数で生成した．全ての行列積の試行回数は 10 回である．

表 1　測定環境の仕様

測定マシン	OS	CPU	メインメモリ	L1Cache	L2Cache	CPU 数
Pentium4	Red Hat Linux 7.1	Intel Pentium4 2.0GHz	1GB	8KB	512KB	1
Opteron	Red Hat Linux 7.1	AMD Opteron 1.8GHz	2GB	64KB	1MB	2
Origin3400	IRIX 6.5	MIPS R14000 500MHz	8GB	32KB	8MB	16

表 2　再帰 BLAS の測定条件

測定マシン	行列サイズ N	再帰段数 L	スレッド間の同期
Pentium4	1000, 1500, 2000, 2500, 3000, 3500	0〜5	無し
	4000, 4500, 5000,5500, 6000	1〜8	無し
Opteron	100, 200, 300, 400 (L1 キャッシュ付近)	0〜5	有り
	500, 1000, 2000, 3000, 4000, 5000 (L1 キャッシュ外)	0〜5	有り
Origin3400	100, 200, 300, 400 (L1 キャッシュ付近)	0〜4	有り
	500, 1000, 2000, 3000, 4000, 5000 (L1 キャッシュ外)	5〜9	有り

4.2 再帰 BLAS の性能評価

4.2.1 Pentium4 における性能評価

DGEMM_P が性能劣化するサンプリング点以外の行列サイズの範囲と，劣化しない次元数の範囲で評価を行った. Pentium4 は 1CPU マシンである. コンパイラは，gcc ver.2.96 で，オプションは “-O3” を指定した．図 8，9 は縦軸 Mflops，横軸再帰段数で再起 BLAS の性能を示す．図 8，9 から，行列サイズ N=1000～2000 では，再帰段数 L=0，N=2500～4500 では L=3，N=5000～6000 では L=6 で最高性能を示している．これは，DGEMM_P では行列サイズが大きくなると性能劣化するが，再帰によって行列を分割することにより，高速に計算できる最適な行列サイズを選択できることを意味している.

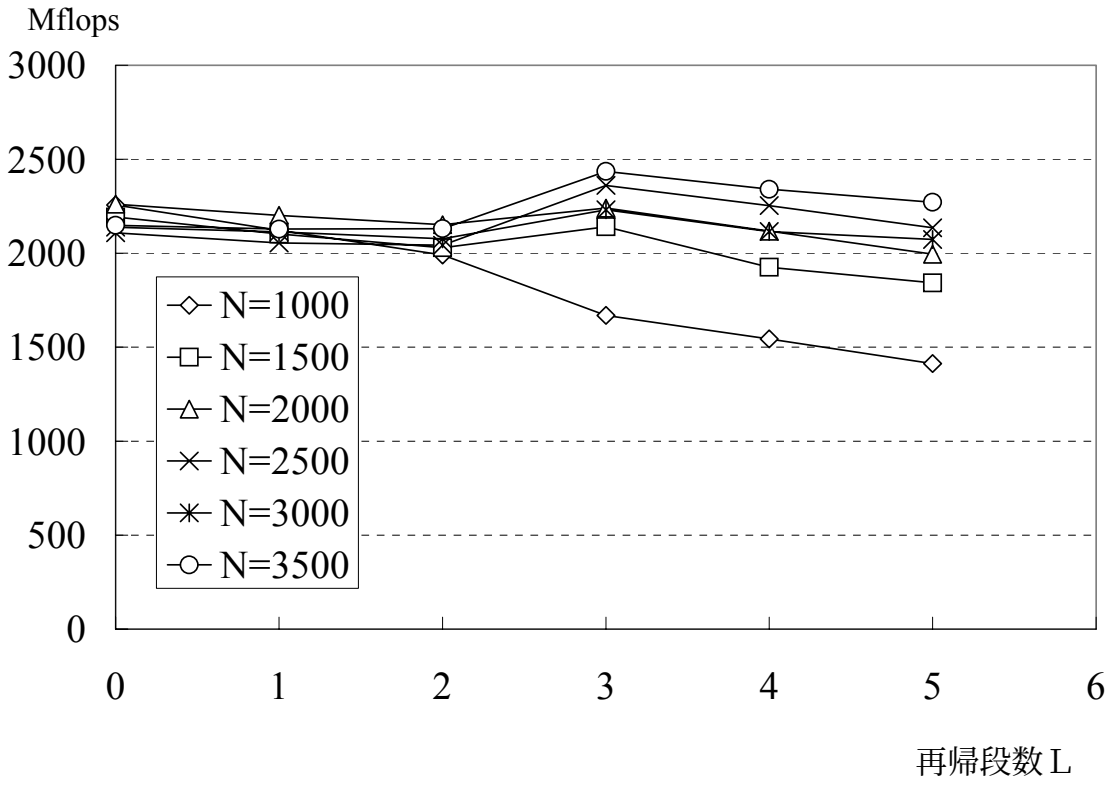


図 8　Pentium4 における再帰 BLAS の

Mflops 値(L=1000～3500)

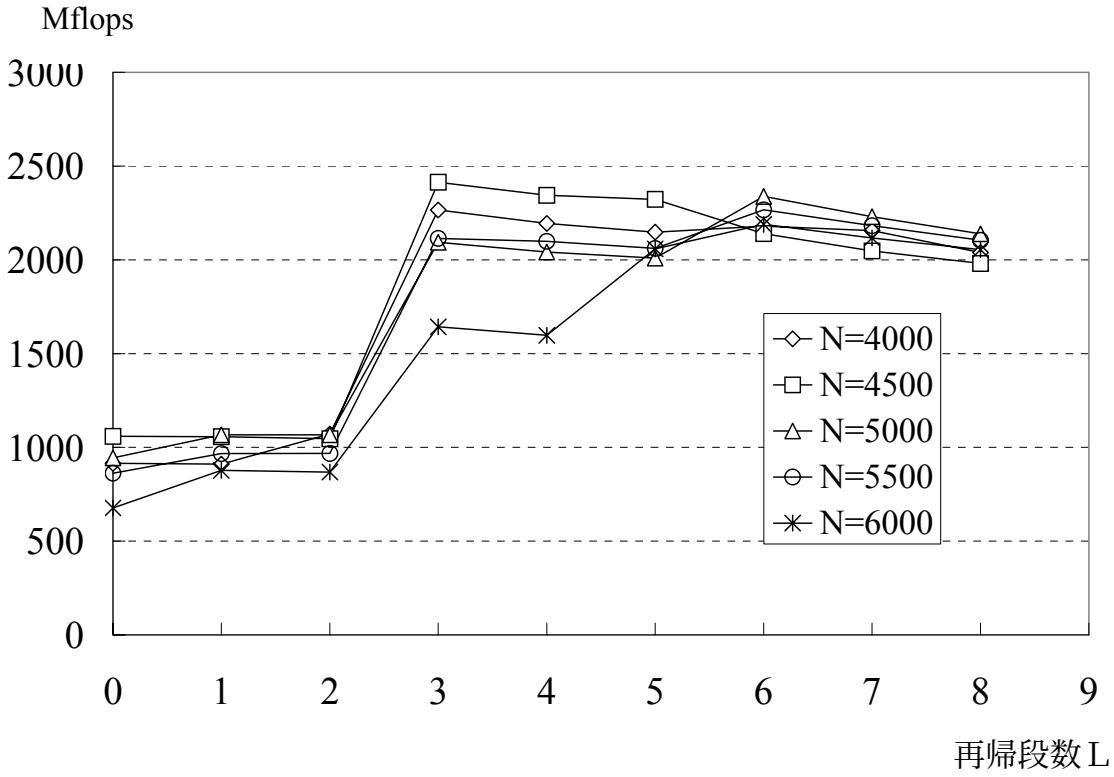


図 9　Pentium における再帰 BLAS の

Mflops 値(L=4000～6000)

4.2.2 Opteron における性能評価

ここでは，2CPU で OS として Red hat Linux を用いた SMP マシンを利用した．Opteron のコンパイラは，gcc ver.2.96 で，オプションは “-O3” を指定した．図 10 から，次元数 N=100 では再帰段数 L=0，N=200～400 では L=1 で最高性能を示す．これは，行列のデータ量が L1 キャッシュ以下になり，再帰分割処理時間がオーバヘッドとなったためと推定される．そのため L > 1 では遅くなっている．図 11 では，全ての次元数で L=1 で最高性能を示す．理由として，DGEMM_P が L1，L2 キャッシュの最適化を行っているため(5 章 ATLAS 参照)，再帰分割した行列の大きさが L1 を超えた時点で CPU 数に適した L が BLAS 内で選択されているからと考えられる．図 10，11 から，L1 キャッシュ外においては末端葉数が CPU 数に等しいことがわかった．また，次元数が大きくなるに従って，Mflops 値が上昇しているにもかかわらず, 図 10 の N=300 と N=400 の Mflops 値が逆転している．これは, ATLAS BLAS の L1 キャッシュ最適化による影響だと推定される．さらに，図 11 の次元数 N=5000 における Mflops 値が再帰段数 L=4，5 で急激に低下している．これは複数のスレッド内での行っている小行列の malloc が主記憶サイズを超えたために起きたスワップによると推定される．

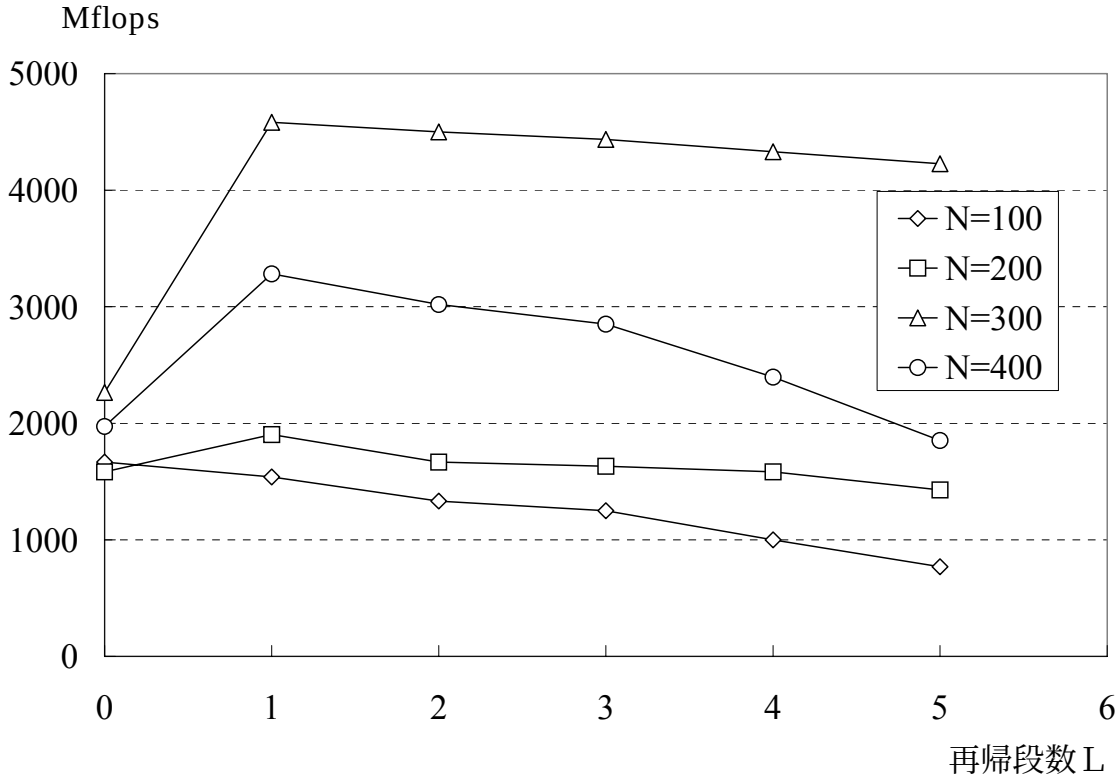


図 10　Opteron における再帰 BLAS の

Mflops 値(L=100～400)

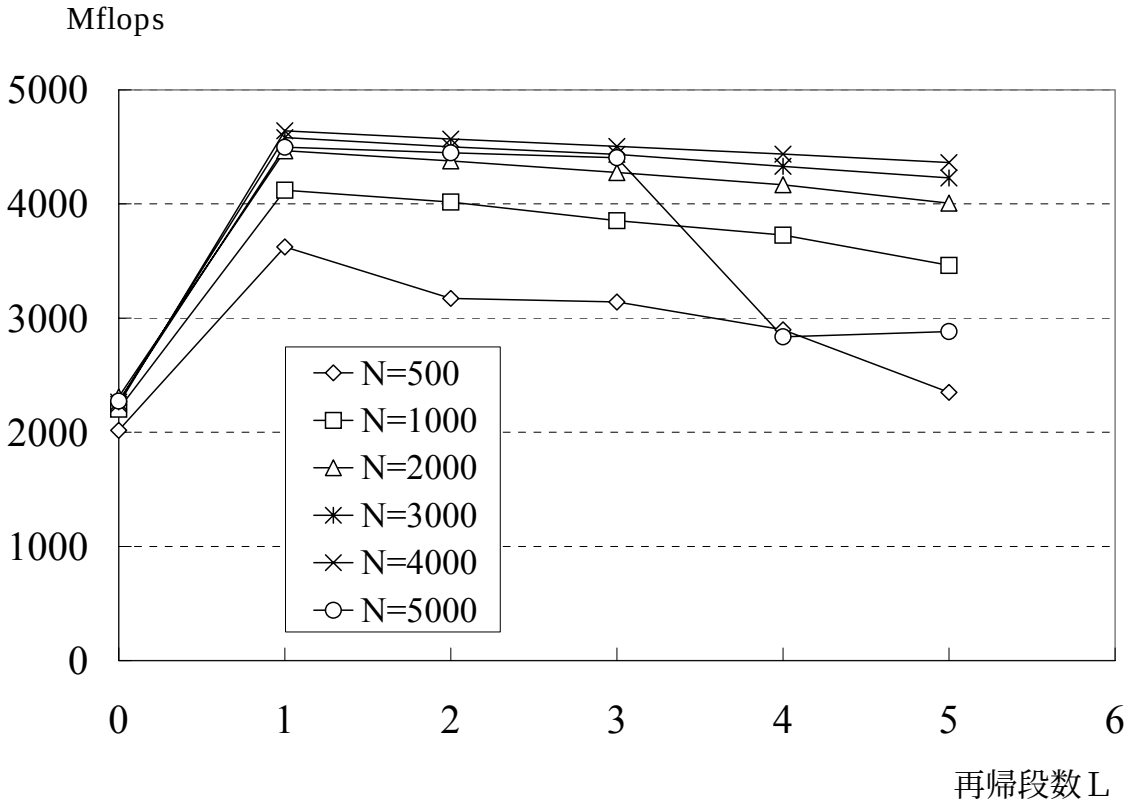


図 11　Opteron における再帰 BLAS の

Mflops 値(L=500～5000)

4.2.3 Origin3400 における性能評価

ここでは, 16CPU で OS として IRIX6.5 を用いた SMP マシンを利用した. Orgin3400 の評価では，IRIX C コンパイラ，オプションは ”-64 -O3” を用いた.

図 12, 13 から, 次元数 N=100 では再帰段数 L=1, N＝200～400, 1000～5000 では L=5 で最高性能を示している. N=100 で再帰段数 L=1 が最高性能を出した理由は Opteron と同様に, 行列のデータ量が L1 キャッシュ以下になり, 再帰分割処理時間がオーバヘッドとなったためと推定される．N=100 以外では，最高性能を出す再帰段数 L=5 から，末端葉数が 32 個である．このことより，末端葉数と CPU 数が一致しないことがわかる．これは Opteron の結果と異なっている. このことから CPU 数から最も速くなる再帰段数を指定することが出来ないことがわかる.

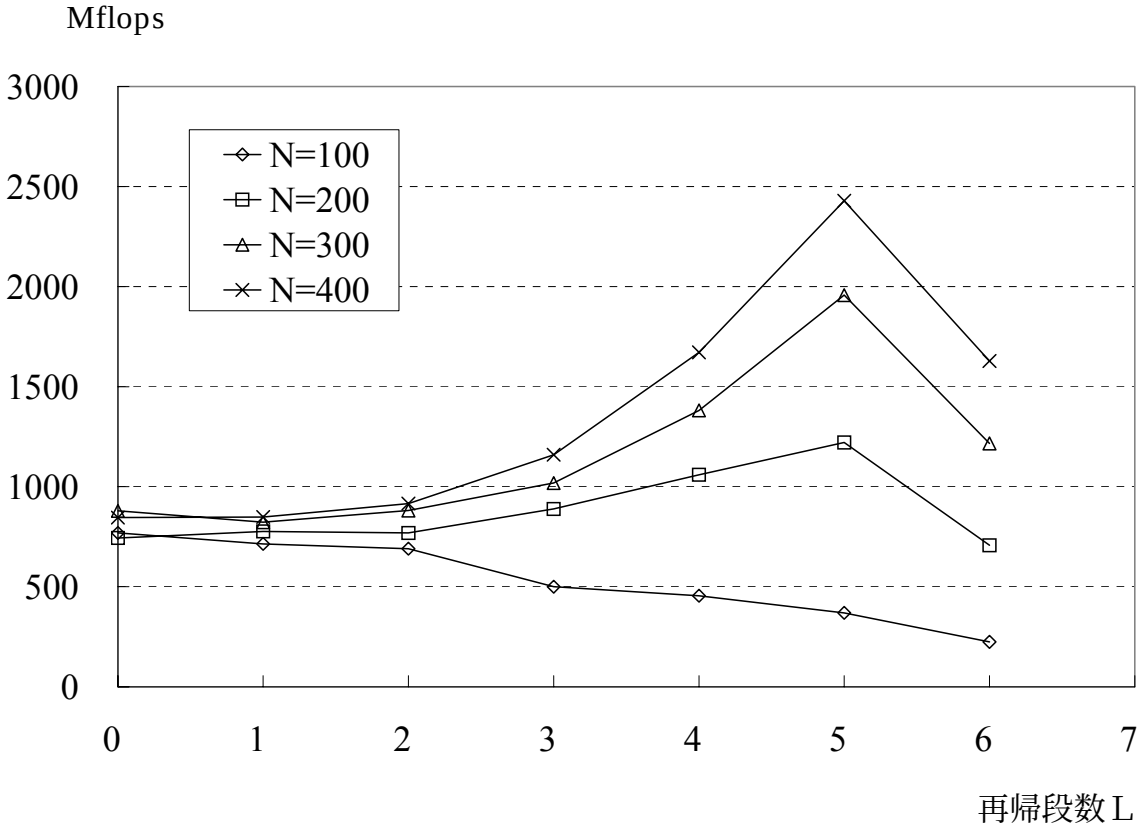


図 12　Origin3400 における再帰 BLAS の

Mflops 値(L=100～400)

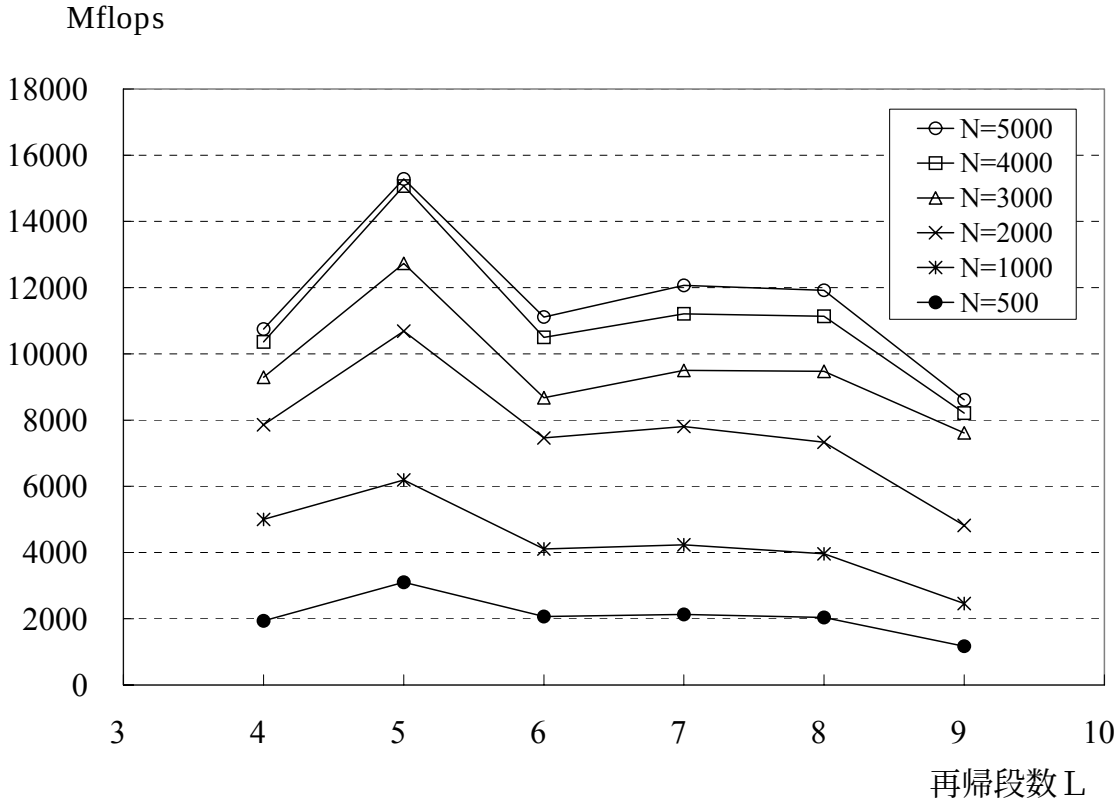


図 13　Origin3400 における再帰 BLAS の

Mflops 値(L=500～5000)

4.3 再帰 BLAS の有効利用

第 I 部では，BLAS と Gustavson らが提案した再帰 BLAS アルゴリズムを 1CPU マシン，SMP マシンにおける実装方式を提案し，実装を行った．再帰 BLAS の性能評価から 1CPU マシン(Pentium4)において，以下の 2 つが分かった．

- 1CPU マシン
  - (a) 行列サイズから性能劣化する再帰段数を特定することができない
  - (b) 最高性能を出す再帰段数は固定値ではない

また，SMP マシン(Opteron，Origin3400)において以下の 3 つが分かった．

- SMP マシン
  - (a) L1 キャッシュサイズ未満の行列のデータサイズでは，再帰によるオーバヘッドにより性能劣化する
  - (b) 再帰によって分割された末端葉数が CPU 数と一致しない
  - (c) 小行列の malloc による主記憶のオーバーフロー

SMP マシンでは，再帰分割によって測定できる最大の行列サイズが小さくなる特徴がある．また，1CPU マシン，SMP マシンともに，最高性能が出せる再帰段数は一定値で求めることができない．このことより，BLAS ライブラリを用いた行列積で最高性能を出すためには，マシン環境にあわせた再帰段数の最適化を行なう事が重要である．

インストール型の自動チューニング機構に再帰 BLAS を組み込み，再帰段数を自動チューニングすることで，行列積における最高性能を出すことができる．第Ⅱ部に，再帰 BLAS の自動チューニングを有する AutoTuned-RB の提案，実装，性能評価を行っている．

第 II 部
AutoTuned-RB

5 自動チューニングソフトウェア

5.1 自動チューニング方式

1 章で述べたように，自動チューニング方式は大きく分けて次の 2 つが存在する．

(１)　インストール時自動チューニング
(２)　実行時自動チューニング

(１)は，ライブラリをマシンにインストールする際に，対象マシン最適なパラメタをチューニングしてインストールする方式である．(２)は，ライブラリを実行するたびに最適なパラメタをチューニングする方式である．(１)の利点は，ライブラリインストール後にマシン環境が変化 (例:OS の変更，メインメモリの増設，CPU 数の増減，等) しない限り，サンプリング点の範囲内で最適な処理が可能であることである．欠点としては，サンプリング点の範囲外の実行では性能劣化する可能性があり，マシン環境を変更する場合にはライブラリを再インストール (再チューニング) しなければならない点である．計算負荷が動的に変動しないスタンドアロンで利用される SMP マシンや 1CPU マシンには，(１)の方式が適している．

次に(２)の利点は，ライブラリを実行する際にサンプリング点を決定でき，最適な値を選択できる点である．環境に応じてパラメタを選択しているので，環境が変化しても性能劣化しない．欠点としては，ライブラリ実行毎にパラメタを動的に選択しなくてはならず，その選択処理がオーバヘッドになってしまうことである．したがって，パラメタ選択処理時間の短縮が必要となる．数値計算ライブラリにおいては，チューニング対象のパ

ラメタとして, CPU のキャッシュサイズにあわせたブロック化, キャッシュラインがコンフリクトしないようなループアンローリング等があげられる[9][10].

5.2 ATLAS

ATLAS(Automatically Tuned Linear Algebra Software)[4]は，テネシー大学の Clint Whaley らが提案した, 線形代数計算用自動チューニングソフトウェアである．ATLAS の自動チューニング方式（パラメタ決定方式）はインストール時チューニング型で，物理的な L1，L2 キャッシュサイズを参照することによって, 推定される最適なブロックサイズを決定する．この推定には，AEOS[4]という経験的手法(ヒューリスティック)の枠組みが利用される．具体的には, 決められたブロックサイズを増減させ実行時間を測定し，最も速くなるパラメタ値を推定する．そのパラメタ値に基づき, ATLAS BLAS を生成する．ATLAS のインストールパッケージには，主に以下の 5 つが含まれている．

(１) BLAS ライブラリ(対応言語：Fortran，C)
(２) L1, L2 キャッシュブロックの最適化を行う自動チューニング機構
(３) 様々な CPU アーキテクチャ，OS に応じたインストーラ
(４) Multi Thread 実行(スレッドセーフな実装)
(５) 速度測定用テストプログラム

スレッドセーフな実装とは，BLAS 関数(DGEMM 等)が複数のスレッ

ド内でコールされてもデータ競合を起こさない実装方式である．ATLAS BLAS は BLAS と同様なインターフェースを持っており, レベル 1 BLAS, レベル 2 BLAS，レベル 3 BLAS 全てに対応している．

5.3 ATLAS の問題点

ATLAS では，サンプリング点以外の行列サイズによる実行では，性能が劣化することがある．すなわち，性能安定性の問題が生じる．さらにライブラリは MultiThread 実行可能ではあるが，完全に並列化された GEMM 関数はサポートされていない．

6 AutoTuned-RB による自動チューニング

6.1 提案する自動チューニング方式

提案する AutoTuned-RB における自動チューニングは，ATLAS と同様にインストール時に行う．再帰 BLAS のチューニング対象のパラメタは，再帰の段数である．一般にパラメタ最適化手法では，以下の 2 つの方式が知られている．

- 全探索方式
  確実に最高性能が出るパラメタを求めることができる．しかし，探索する幅が大きくなると計算量が増えるので，インストール時の自動チューニング時間が増加する．

- ヒューリスティック方式
  経験的に最高性能に近い値を達成できるサンプリング点を選ぶ．それによって，チューニング時間を短縮する．

本論文では，1CPU マシンには全探索法，SMP マシンにはヒューリスティック方式により再帰段数を決定する．いま，BLAS として ATLAS が自動チューニングを行ったものを採用することを考える．SMP マシンと 1CPU マシンでの再帰段数決定方式を以下に述べる．

- 1CPU マシン
  チューニング対象パラメタの再帰段数 L を L=1，2，3，…と順に変化させ，最も性能が出るものを最適パラメタとして採

用する

- SMP マシン
  ATLAS では，L1，L2 キャッシュに対してチューニングを行っている．そこで，その L1 キャッシュサイズ付近と，L3 キャッシュサイズ付近での最適化に焦点を当てる．最適な再帰段数 L は，CPU 数に依存すると考えられる．再帰段数 L を CPU 数から仮採用する．しかし，キャッシュサイズに依存する可能性もあるので，L-2，L-1，L，L+1，L+2 段で速度測定を行い，最も速くなる再帰段数を採用する．

行列サイズに関するサンプリング点を固定し，上記の再帰段数 L において実行時間を測定し，それをもとに最適化をすると，行列サイズが大きい（または小さい）場合に，本来最高性能を与える再帰段数が推定できない可能性がある．そのため本提案方式では，測定する行列サイズを複数用意する．各々の測定結果から，行列サイズに関するサンプリング点における最高性能を与え再帰段数を 1 次多項式関数化する．このことで，行列サイズに依存する再帰段数の切り分けという特徴をもつ．

6.2 1CPU マシンにおける行列サイズによる再帰段数決定モデル

1CPU マシンにおける主な自動チューニングの目的は，性能の安定化である．

BLAS が性能劣化する行列サイズにおいて，高速に実行できる行列サイズ以下まで行列サイズを分割する．最適な再帰段数の決定は全探索法を用いる．また，どの行列サイズで性能が劣化するかが不明なため，サンプリング点は均等な行列サイズの間隔で測定を行う．図 14 は，横軸が行列サイズ，縦軸がピーク性能達成時の再帰段数である．サンプリング点 s_1，s_2，s_3，s_4，…を等間隔にとる．サンプリング点から得られる行列データの合計サイズが，物理的な主記憶サイズを超えると、ページのスワップが発生する．したがって，性能を安定化するために，インストール環境の主記憶サイズを上限とする．いま，サンプリング点間のパラメタは，1 次多項式である式(４)で推定する．

$$f(N)=a+bN \qquad (4)$$

0～サンプリング点 s_1 までの推定関数を f_1(N)，サンプリング点 s_1 より大きな値～サンプリング点 s_2 までの推定関数を f_2(N)，サンプリング点 s_2 より大きな値～サンプリング点 s_3 までの推定関数を f_3(N)のように，計算する行列サイズの上限まで 1 次多項式を定義する(図 14 参照)．

1CPU における再帰段数決定手法は，複数のサンプリング点について，ライブラリが最も速く計算することができる再帰段数を選択する．決定した再帰段数を 1 次多項式でモデル化し，行列サイズによって切り分ける．

いま D_1，D_2 は，自動チューニングした再帰 BLAS を使用するユーザ

が指定した行列サイズとする．点 D_1，D_2 で測定する場合，以下のように最適段数を決定する．

点 D_1 では，f_1(D_1)からこの値に最も近い整数値を推定段数とする．D_2 では，s_1 の段数を推定段数とする．AutoTuned-RB では，このような 1 次多項式を用いてモデル化し，行列サイズに応じて適切な再帰段数を切り分ける．

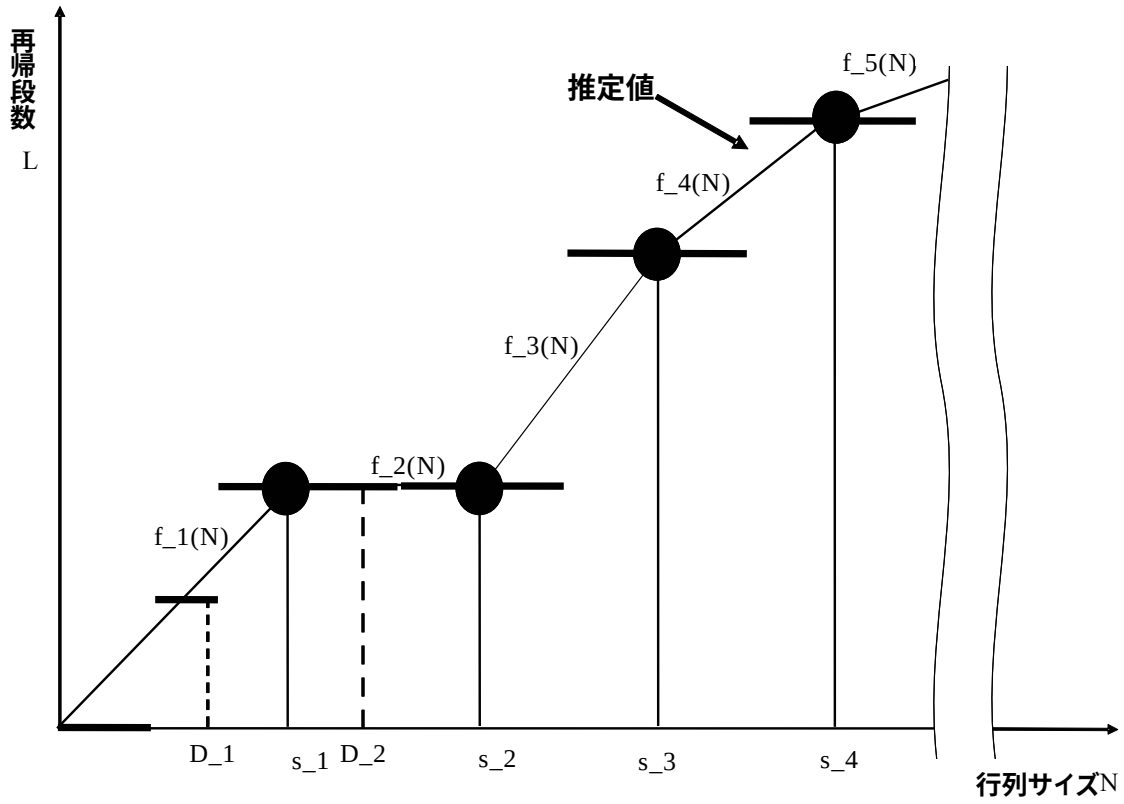


図 14　1CPU マシンにおける再帰段数決定モデル

6.3 SMP マシンにおける行列サイズによる再帰段数決定モデル

SMP マシンにおける自動チューニングの対象は並列化である．図 15 は，本論文で用いる行列サイズ依存の SMP マシン用の再帰段数決定モデルである．図 14 と同様に，横軸が行列サイズ，縦軸がピーク性能達成時の再帰段数である．

行列サイズのサンプリング点 s_S，s_M，s_L は，AutoTuned-RB で自動チューニングする際に用いた行列サイズである．このサイズは概述の通り複数用いるが，以下の基準で設定する．

- サンプリング点 s_S
  行列 A，B，C の合計サイズが，L1 キャッシュサイズより大きくなる行列サイズのうち最も小さいもの

- サンプリング点 s_M
  行列 A，B，C の合計サイズが，L3 キャッシュサイズより小さくなる行列サイズのうち最も大きいもの

- サンプリング点 s_L
  s_M サイズより大きい行列サイズのうち適当な値

これらのサンプリング点では，実測を基に最も高速となる再帰段数を決定する．L3 キャッシュが搭載されていない場合，s_M の指定は行わずに s_M サイズが L2 キャッシュより大きい適当な行列サイズを指定する．ここで，推定するサンプリング点の 1 次多項式は式(４)と同様のものを用い

ている．

ここで，0～サンプリング点 s_S までの推定関数を f_S(N)，サンプリング点 s_S より大きな値～サンプリング点 s_M までの推定関数を f_M(N)，サンプリング点 s_M より大きな値～サンプリング点 s_L までの推定関数を f_L(N)とする．

1CPU マシン同様に，自動チューニングした再帰 BLAS を使用するユーザが指定する行列サイズ D_1 では，f_M(D_1)からこの値に最も近い整数値を推定段数とする．D_2 では，s_L の段数を推定段数とする．

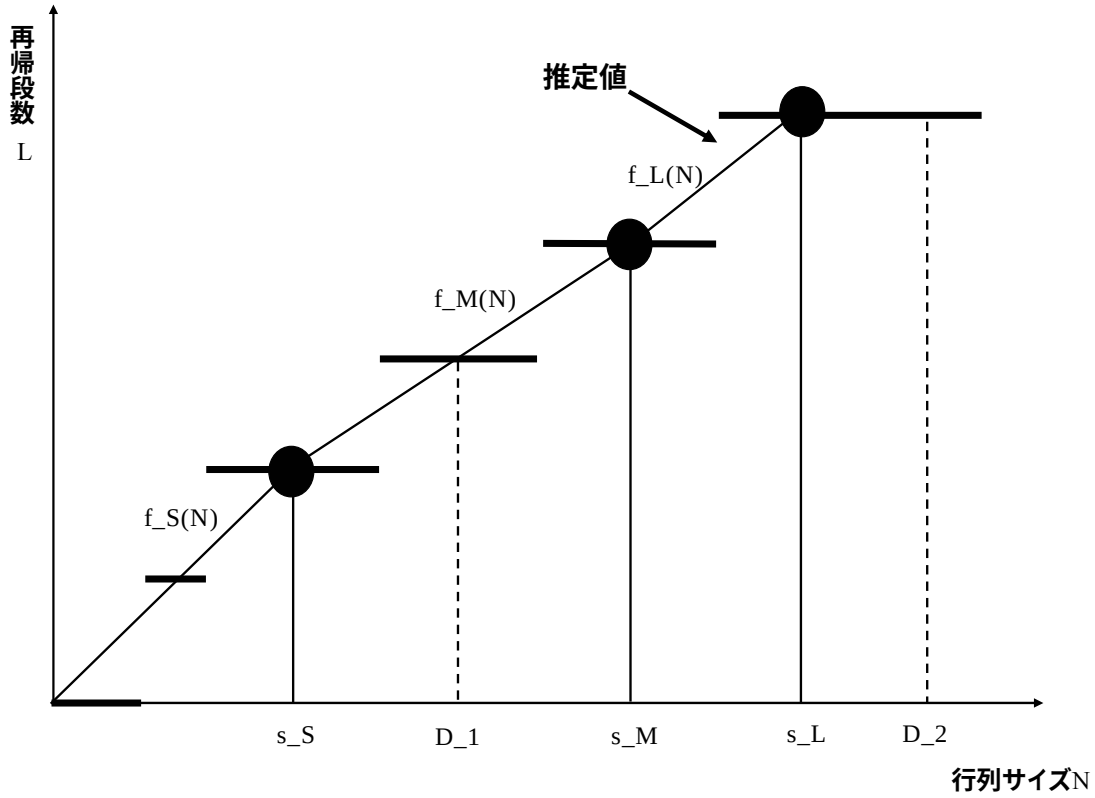


図 15　SMP マシンにおける再帰段数決定モデル

6.4 AutoTuned-RB のインストールの流れ

AutoTuned-RB の計算機へのインストールの流れを図 16 に示す．BLAS ライブラリとして ATLAS BLAS を用いる．しかし，どのような BLAS でもインストール可能である．図 16 中の①，③は，ATLAS で行っている処理部分である．AutoTuned-RB で新たに実現する機能は②，④の部分である．①～④の説明を以下に示す．

① ATLAS をインストールする過程で, 最初に行う処理部分である．ここでは，チューニングに必要な L1，L2 キャッシュサイズ, CPU の種類, インストールで用いるコンパイラの種類, およびコンパイラオプションをユーザが指定する．

② ATLAS の BLAS において, ①で作成した環境ファイルを元にキャッシュブロックサイズを決定する．

③ AutoTuned-RB でインストールの際に必要な Configure file を生成する部分である．Pthread を使用しない場合は, CPU 数の入力を行わずに Configure file を生成する．Pthread 使用する場合は，計算機環境での CPU 数をユーザが指定した後，Configure file を生成する．

④ Configure file を基に，②でチューニングした BLAS 用いて，再帰 BLAS をチューニングする．**(1)**では，ヒューリスティック法(SMP マシン), 全探索法(1CPU マシン)で定められた再帰段数と計測用の行列サイズを決定する．その後，**(2)**ではサンプルプログラムを用いて再帰 BLAS のチューニングを行う．

つぎに，求めた測定データから再帰段数パラメタ L に関する実行時間を 1 次多項式化する．

以上の①～④から、AutoTuned-RB ライブラリを生成する．
インストール中に作成する Configure file には，以下の 6 つの情報が含まれている．

(１) Pthread サポートの指定
(２) CPU 台数
(３) L1 キャッシュサイズ
(４) L2 キャッシュサイズ
(５) L3 キャッシュサイズ
(６) 主記憶サイズ

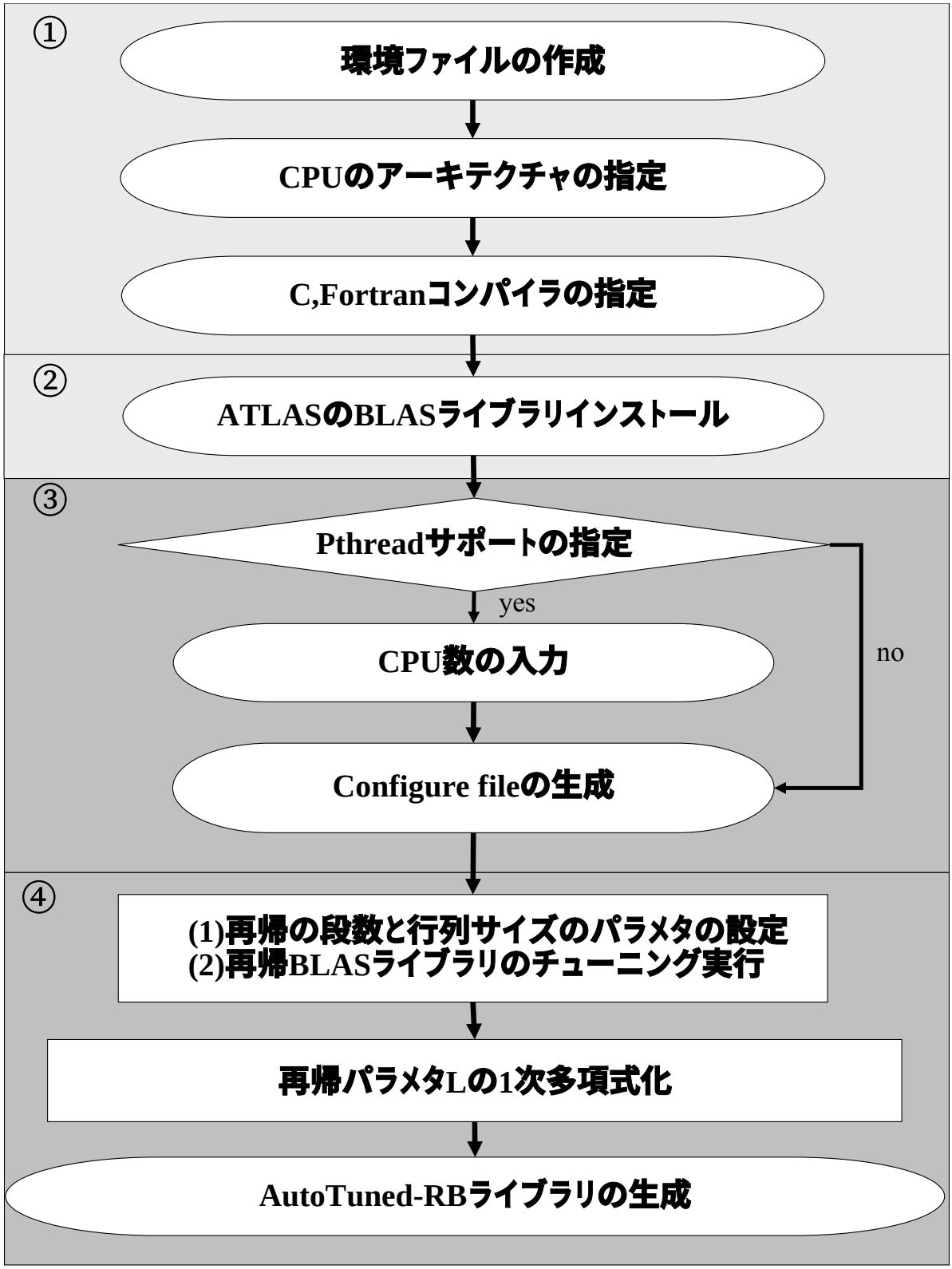


図 16　AutoTuned-RB インストール

7 AutoTuned-RB の性能評価

7.1 測定環境

本章では，ATLAS と提案する AutoTuned-RB との性能比較，および AutoTuned-RB のチューニング効果の評価を行う．性能評価環境は 4 章の再帰 BLAS の測定と同様に，並列処理学講座の 1CPU の Pentium4，2CPU の Opteron，電気通信大学総合情報処理センターの 16CPU の Origin3400 である．マシン仕様を表 2 に示す．測定プログラムは，ATLAS がチューニングを行った DGEMM_P を利用して，それを再帰化したもの（以下 RDGEMM）を使用した．なお ATLAS のバージョンは 3.4.2 である．表 2 における条件のもとに，各行列サイズ N において，再帰段数 L を変化させたときの Flops 値変化を評価した．また AutoTuned-RB で使用したサンプリング点を表 3 に示す.計算カーネルは DGEMM_P をコールしている．行列は密の正方行列で，行列の各要素は乱数値で生成した．

表 3　AutoTuned-RB のサンプリング点

測定マシン	サンプリング点 s_S	サンプリング点 s_M	サンプリング点 s_L
Opteron	120	設定なし	500
Origin3400	150	設定なし	2500

測定マシン	サンプリング点(行列サイズの間隔 500)
Pentium4	500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000

7.2 ATLAS と AutoTuned-RB の性能比較

ここでは, ATLAS の DGEMM_P と AutoTuned-RB との性能比較を行う. AutoTuned-RB は前節 4.1 の再帰 BLAS の性能評価で, 各々の行列サイズにおいて最高性能を示す再帰段数を用いている.

7.2.1 Pentium4 における性能比較

図 17 は，Pentium4 における DGEMM_P と AutoTuned-RB との比較を示す．DGEMM_P は，行列サイズ N=4000～6000 で激しく性能劣化していることがわかる．これは，ATLAS による L1，L2 キャッシュのチューニングが，行列サイズ N=4000 以上において不具合があったのではないかと推測される．AutoTuned-RB は計測を行った全ての次元で安定した性能を出せることが確認できた.

AutoTuned-RB は，次元 N=4000～6000 においては， DGEMM_P に比べ約 2 倍の性能向上を達成した．この理由は，DGEMM_P が ATLAS の局所チューニングによって, 次元 N=4000 以降で激しく性能劣化する. しかし，AutoTuned-RB では，行列を再帰分割することによって，キャッシュサイズに合うように DGEMM_P が高速に計算できる行列サイズに分割して計算するからだと考えられる.

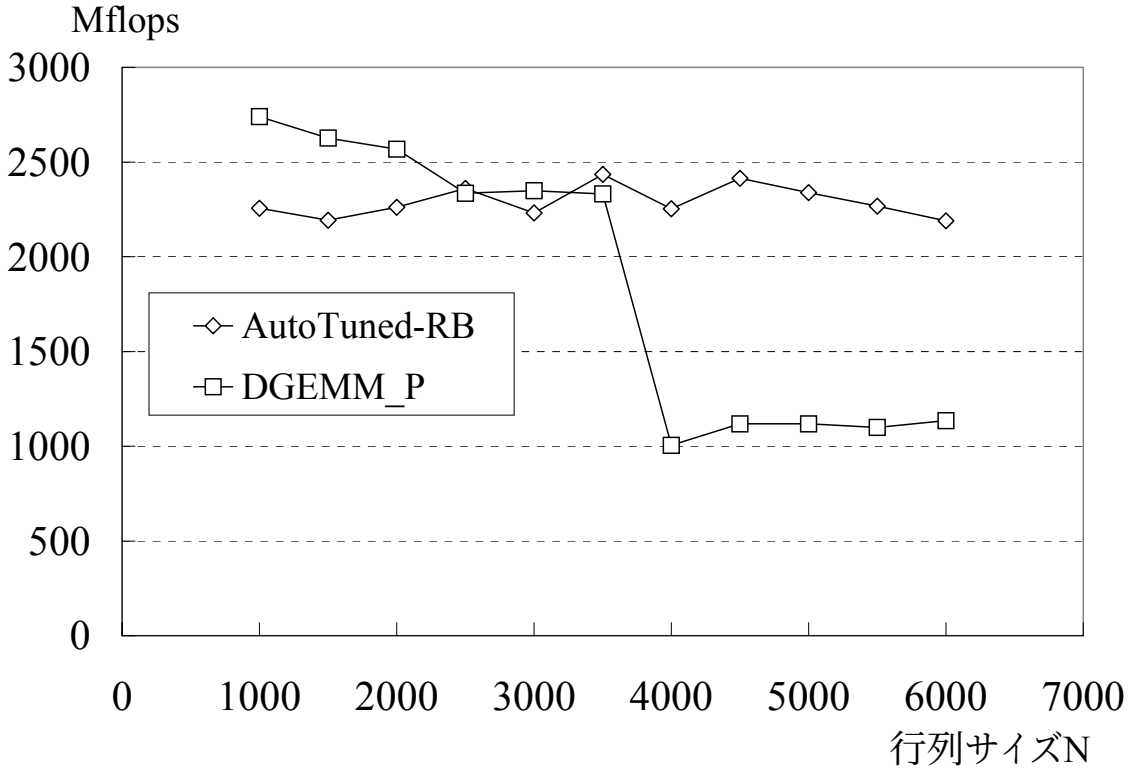


図 17　Pentium4 における AutoTuned-RB と

DGEMM_P の Mflops 値

7.2.2 Opteron における性能比較

図 18 は，Opteron における ATLAS の Multi-thread 実行可能なライブラリ DGEMM_P（以下，並列版 DGEMM_P）と AutoTuned-RB との比較を示す．測定を行った全ての行列サイズにおいて，ほぼ同等の安定した性能が出ることを確認した．

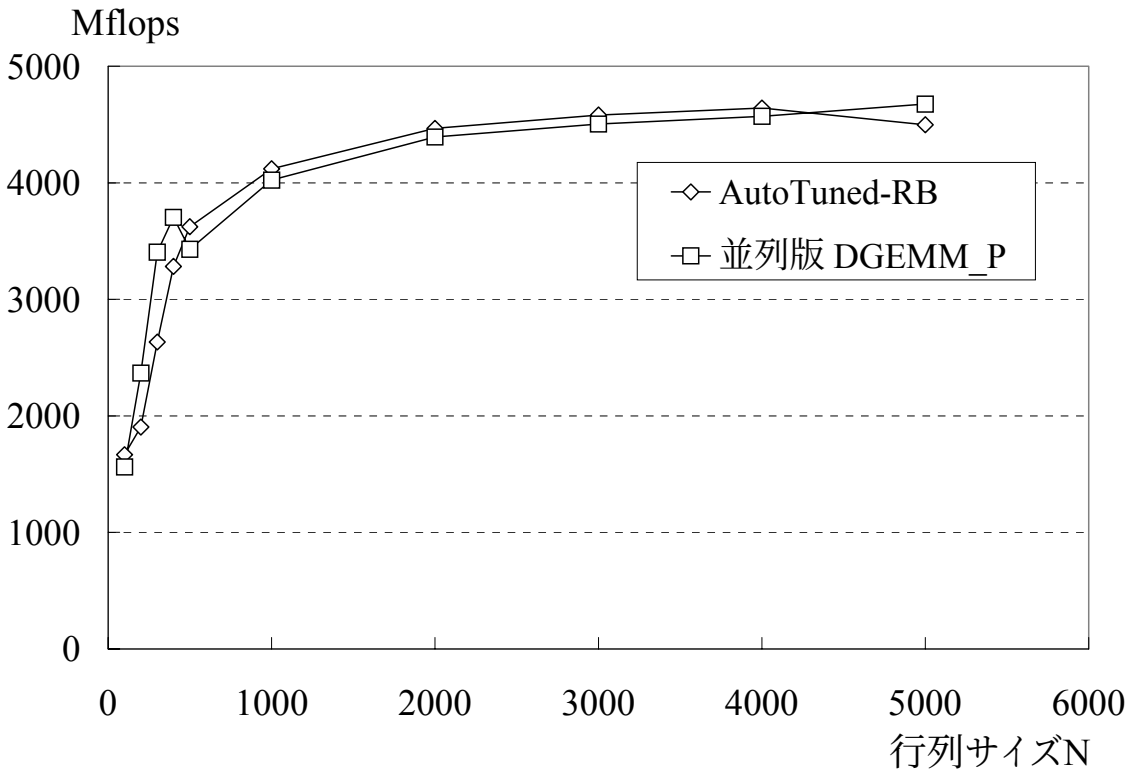


図 18　Opteron における AutoTuned-RB と
並列版 DGEMM_P の Mflops 値

7.2.3 Origin3400 における性能比較

図 19 は，Origin3400 における並列版 DGEMM_P と AutoTuned-RB との比較を示す．行列サイズ N=100～1000 において，並列版 DGEMM_P の性能が良い．これは，AutoTuned-RB の再帰分割のオーバヘッドによる性能劣化だと考えられる．

行列サイズ N=2000～6000 において AutoTuned-RB は, 並列版 DGEMM_P に比べて大幅に性能が向上している．

特に次元 N=5000～6000 では, 並列版 DGEMM_P の性能が劣化している．この結果，DGEMM_P の性能は実行時間に比べ，AutoTuned-RB が約 3.3 倍の性能向上を達成した．これは，DGEMM_P の Multi-thread 実行では，性能が激しく低下するためである考えられる．AutoTuned-RB はライブラリレベルで並列化を重視した自動チューニングを行っており，ATLAS では完全に並列化されたライブラリはサポートされていない．一方，N=100～1000 では, DGEMM_P のほうが性能がよい. これは AutoTuned-RB では，L1 キャッシュ以下のとき，再帰分割のオーバヘッドが行列の分割による速度向上に対して顕著になるためと考えられる．1CPU での DGEMM_P では，理論ピーク性能に対する効率が約 90%であるが，16CPU で N=4000 のときは，約 62%になる．それに対して AutoTuned-RB では，16CPU で N=5000 のとき，ピーク性能に対する効率は約 90%となる．したがって，並列処理効果は十分高い．

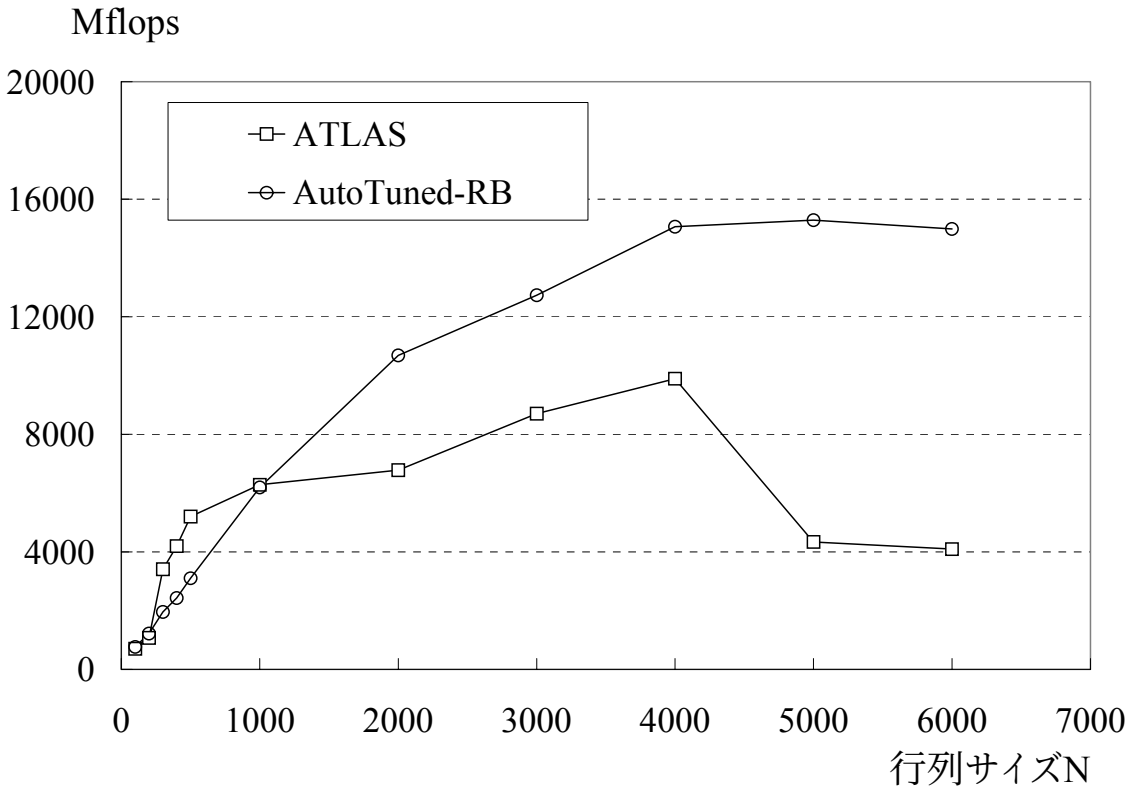


図 19　Origin3400 における AutoTuned-RB と

並列版 DGEMM_P の Mflops 値

7.3　AutoTuned-RB のチューニング効果

7.3.1　1 次多項式化についての評価

図 20 は, AutoTuned-RB でチューニングした最適な再帰段数を示す. 各々の直線は AutoTuned-RB でチューニングして求めた 1 次多項式グラフである．また，各□，◇，△は，節 4.2 で求めた最高性能である．

Pentium4 は行列サイズ 500 毎に測定を行い, 再帰段数は L=0~8 とした．Opteron では s_S=120, L3 キャッシュは備えていないので s_M は設定なし，s_L は（L2 キャッシュサイズにおける行列サイズ）+200 より s_L=500 とした．Origin3400 におけるサンプリング点は s_S=150，s_M 設定なし，s_L は（L2 キャッシュサイズ+200）より s_L=2500 である．ここでは L2 を超える適当な行列サイズは 200 である．

これより，前節 7.2 で求めた最高性能は AutoTuned-RB でチューニングした値に等しい結果が得られた．このことから AutoTuned-RB による自動チューニングの精度はきわめてよいことがいえる．Opteron に関しては，L1 キャッシュ以上では最適な段数が固定値になり, 変化がない. これは，L3 キャッシュがないので，最適化の余地がなかったためである．Origin3400 での最適な段数についても，Opteron と同等の理由であると考えられる．Pentium4 には L3 キャッシュはないが，N=1000～2000 での 1 次多項式は f(N)=0，N=2000～2500 では f(N)= (3/500)N-12，N=2500～4500 では f(N)=3，N=4500～5000 では f(N)= (3/ 500)N-24，N=5000～6000 では f(N)=6 となった．この場合 N=2000～6000 では，AutoTuned-RB の最適な段数選択により，ATLAS の DGEMM_P による性能の不安定性を改善することができた．

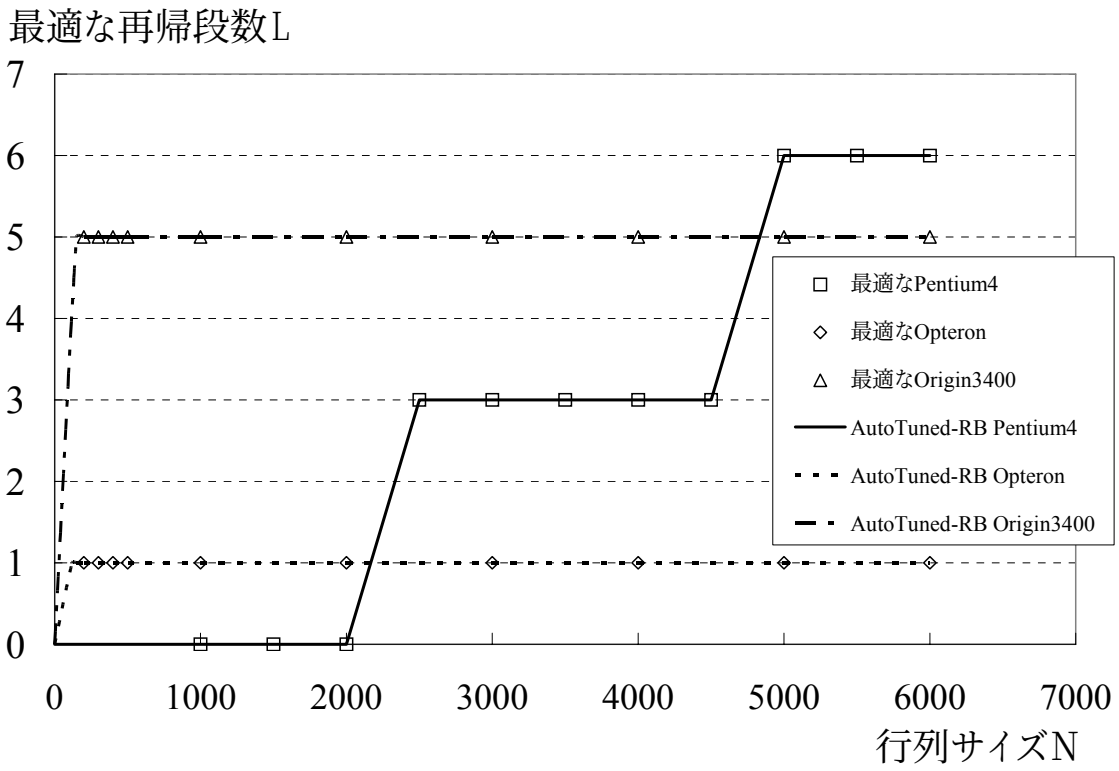


図 20　AutoTuned-RB の最適な再帰段数

7.3.2 1 次多項式上の最適解についての評価

ここでは，1CPU の PC-SOFTEK における行列サイズ N=2000～2500 での 1 次多項式が最適であるかどうかを評価する．図 21 は，縦軸が測定で求めた最適な再帰段数，横軸が行列サイズである．図 21 が示すように，最適な再帰段数 L が 0 と 3 で激しく振動している．2000 から 2500 に向かうにつれて振動の幅が減少し 3 へ収束していると推測できる．この結果から，再帰段数が 0 から 3 に変化するような行列サイズは，激しく性能が不安定であることがわかった．このような条件では，1 次多項式で正確に最適値を定めることができない．しかし行列サイズ N=2000～2500 においての再帰段数 L=0, 1, 2, 3 での Mflops 差は最大でも 10%，平均 5%程度である．したがって，再帰段数が大幅に変化する行列サイズにおいては，擬似的に 1 次多項式で適切な再帰段数を求めても問題ないといえる．

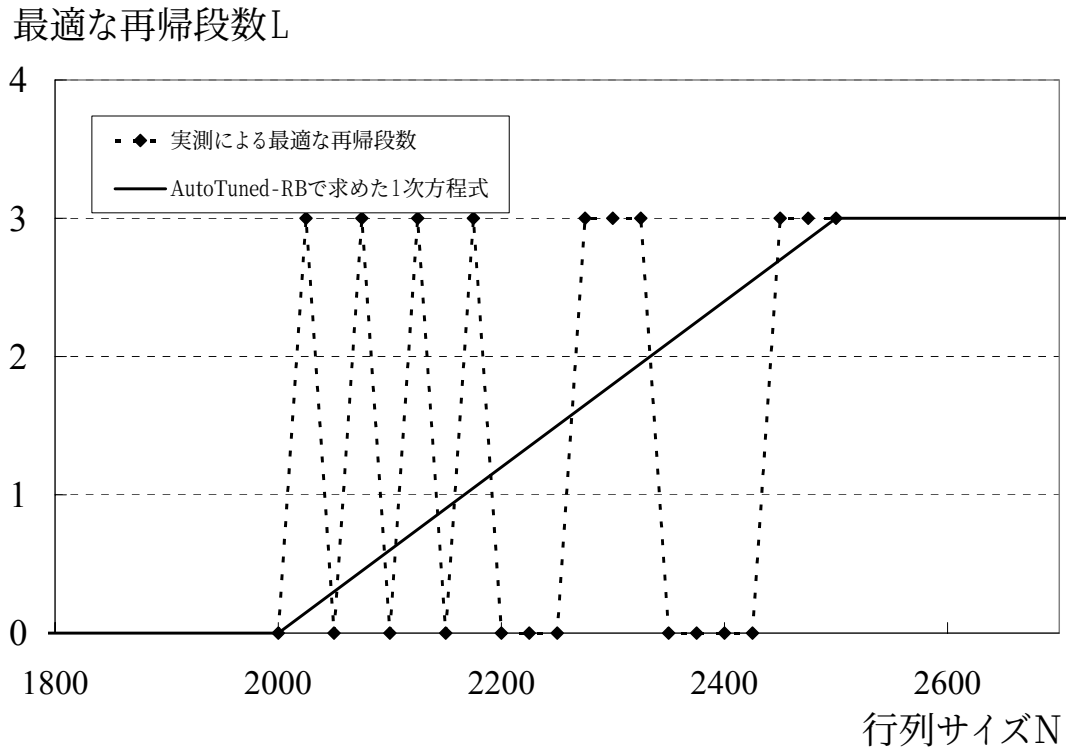


図 21　PC-SOFTEK における AutoTuned-RB と実測値の比較

8 結論

8.1 まとめ

本論文では，AutoTuned-RB と呼ぶ，BLAS ライブラリ新しい自動チューニング方式を提案することで，以下の 2 点を解決した．

まず，従来ソフトウエアで問題となっていた，チューニング処理に必要とされるサンプリング点以外の行列サイズにおける性能不安定性の問題を改善した．

本論文で提案したパラメタ決定手法は，1CPU マシン(Pentium4)には，以下の 3 点の特徴をもつ．

(1) メインメモリサイズから行列サイズの上限を決め，全探索法を用いて再帰段数を決定する

(2) 決定した再帰段数の値を振らして，サンプルプログラム上での行列サイズを変化させ，各次元での最高性能となる段数を決める

(3) 性能劣化する可能性がある行列サイズを意識し，行列次元に対する再帰段数の挙動を，複数のサンプリング点で 1 次多項式化

また，SMP マシンにおいて CPU 数と L3 キャッシュサイズに適応するように，自動的に行列サイズに依存する再帰段数を決定し，並列化する手法を提案した．このことで，SMP マシン上で従来ソフトウエアが十分に

並列化できていない問題を解決した．SMP マシン(Opteron，Origin3400)において以下の 3 点の特徴をもつ．

(1) キャッシュサイズからヒューリスティックを用いて再帰段数を決定する．

(2) 決定した再帰段数の値を振らして，サンプルプログラム上での行列次元を変化させ，各次元での最高性能となる段数を決める．

(3) 階層メモリを意識し行列次元に対する再帰段数の挙動を，SMP マシンでは 3 点で 1 次多項式化する．

このことで，自動チューニングソフトウェアにおいて問題となっていた，性能の不安定性のおよび並列化できてない問題を解決した．

1CPU の Pentium4 マシン上で，従来ソフトウエアである ATLAS での DGEMM 性能に対し，提案した AutoTuned-RB は，次元 2000～6000 で最大 2.0 倍，16CPU の SGI Orgin3400 上では次元 2000～5000 で最大 3.3 倍の速度向上を達成した．

Pentium4 では，ATLAS の BLAS が性能不安定になる場合においても，AutoTuned-RB を用いることで，性能安定化できることを示した．一方，Origin ではピーク性能に対する効率 90%を達成した．この結果は，ATLAS が考慮していない並列化を，AutoTuned-RB により再帰レベルで並列化したことによる効果と考えられる．

以上により，AutoTuned-RB の有用性が確認できた．本論文では BLAS に ATLAS を用いたが，最適化されていない BLAS を用いた場合でも性能の向上が期待できる．また，十分に高速化された BLAS を利用した場合で

も，AutoTuned-RB により性能が著しく劣化することはない．

8.2 関連研究と今後の課題

本論文で述べた，性能安定化に関する関連研究として，以下のものがある．問題サイズが増加したときに，急激に性能が悪化しないという安定性に関する自動チューニング方式を，今村ら[9]が提案している．一方，直野ら[10]は，性能劣化が起きた場合，性能保証を考慮した品質に関する自動チューニングの概念を提案している．

今後，本研究で取り組むべき課題について述べる．

(１) 本性能評価では，L3 キャッシュを搭載していないマシンを使用したために，提案した再帰段数チューニング手法の効果を十分に示すことができなかった．したがって，評価環境をさらに増やし，提案方式によるチューニング効果を検証する．

(２) 再帰 BLAS に比べ，ATLAS の Multi-thread 実行が高性能である場合があった．この場合では，再帰 BLAS と ATLAS BLAS を切り分ける方式が有効となる．この方式を検討し，ベンダ提供の BLAS との比較評価を行う．

(３) コレスキ分解などの線形代数演算に AutoTuned-RB を適用し，有効性を評価する．

(４) 定式化した 1 次多項式で再帰段数が変化する行列サイズにおいて，より有用性のある定式化手法を考察し，評価する．

謝辞

本研究を遂行するに当たって，ご指導頂いた弓場敏嗣教授，本多弘樹助教授，片桐孝洋助手，吉瀬謙二助手に深く感謝いたします．また，研究で進める上でご支援やご助言いただきました並列処理学講座の皆様に深く感謝いたします．また，本研究の一部は科学技術振興機構さきがけからの補助金(研究領域「情報基盤と利用環境」)の援助による．

参考文献

[1] Gustavson, F., Henriksson, A., Jonsson, I., Kågström, B. and Ling, P.: Recursive Blocked Data Formats and BLAS's for Dense Linear Algebra Algorithms, *PARA98 (Proceedings of the Fourth International Workshop on Applied Parallel Computing)*, Lecture Notes in Computer Science 1541, pp.195-206, Springer (1998).

[2] Gustavson, F., Henriksson, A., Jonsson, I., Kågström, B. and Ling, P.: Superscalar GEMM-based Level 3 BLAS –The On-going Evolution of a Portable and High-Performance Library, PARA98, LNCS, Vol.1541, pp.216-223, Springer(1998).

[3] Gustavson, F., Henriksson, A., Jonsson, I. and Kågström, B.: Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software, SIAM REVIEW, Vol.46, No.1, pp.3-45(2004).

[4] Whaley, R.C., Petitet, A. and Dongarra, J.J.: Automated Empirical Optimizations of Software and the ATLAS Project, *Parallel Computing*, Elsevier, Vol.27, pp.3-35(2001).

[5] 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).

[6] 片桐孝洋，黒田久泰，大澤清，工藤誠，金田康正:自動チューニング機構が並列数値計算ライブラリに及ぼす効果，情報処理学会論文誌:ハイパフォーマンスコンピューティングシステム，Vol.42，No.SIG12(HPS4)，pp. 60-76 (2001).

[7] Bilmes, J., Asanović, K., Chin, C.-W. and Demmel, J.: Optimizing Matrix Multiply Using PHiPAC: A Port-able, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97，pp．340-347 (2001).

[8] 片桐孝洋，吉瀬謙二，本多弘樹，弓場敏嗣: FIBER : 汎用的な自動チューニング機能の付加を支援するソフトウエア構成方式,情報処理学会研究報告，2003-HPC-94，pp.1-6(2003).

[9] 今村俊幸，直野健: キャッシュ競合を制御する性能安定化機構内蔵型数値計算ライブラリについて，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.SIG 6(ACS 6)，pp.113-121(2004).

[19] 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.SIG 6(ACS 6)，pp105-112(2004).

[11] 木下靖夫，片桐孝洋，弓場敏嗣: AutoTuned-RB:再帰 BLAS ライブラリの自動チューニングの方式, 情報処理学会論文集, Vol.2005, No.2, pp.33-40(2005).

[12] 片桐孝洋，金田康正: 並列固有値ソルバーの実現とその並列性の改良，JSPP'98 論文集，pp.223-230(1998).

[13] 館野諭司，重原孝臣，長谷川秀彦，桧山澄子: 共有メモリ型並列計算機上の行列計算に対する並列化手法の性能評価, 情報処理学会論文誌，コンピューティングシステム，vol.44，No.SIG11(ACS3)，pp.286-295(2003).

[14] 建部修見: 分散メモリ型計算機による LU 分解，情報処理学会研究報告，ハイパフォーマンスコンピューティング，No.57，pp.55-60(1995).

[15] Andersen,S.B., Gustavson,F., Karaivanov,A. and Waśniewski, J., Yalamov, Y.P.: LAWRA Linear Algebra With Recursive Algorithms，Proceeding of the Conference on Parallel Processing and Applied Mathmathics，ONIC technical report，No.UNIC-99-01，pp.1-14(1999).

[16] Andersen，S.B.，Gustavson amd G.F.，Waśniewski，J.: A Recursive

Formulation of Cholesky Factorization of a Matrix in Packed Storage, Proceeding of the 9th SIAM Conference on Parallel Processing for Scientific Computing，pp.99-129(1999).

[17] Cuenca，J.，Giménez and D.，González，J.: Architecture of an Automatically Tuned Linear Algebra Library，Parallel Computing 30，pp.187-210(2004).

[18] 片桐孝洋，黒田久泰，大澤清，工藤誠，金田康正: 自動チューニング機構が並列計算ライブラリに及ぼす効果，情報処理学会論文誌，ハイパフォーマンスコンピューティングシステム，Vol.42，No.SIG(HPS4)，pp.60-76(2001).

[19] 直野健，山本有作: 単一メモリ型インタフェー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，No.2001-HPC-87，pp.25-30(2001).

[20] 木下靖夫，片桐孝洋，弓場敏嗣: SMP マシン上における再帰 BLAS ライブラリの自動チューニング方式，情報処理学会研究報告，No.2004-HPC-99，pp.187-192 (2004).

[21] 寒川光,日本アイ・ビー・エム(株): LU 分解のブロック化と計算順序，情報処理学会研究報告，No.1992-数値解析-43，pp.1-8(1992).

[22] 石井良規，片桐孝洋，本多弘樹，弓場敏嗣: Autopilot を用いた疎行列ソルバーにおける実行時自動チューニング機構の設計，電子情報通信学会総合大会論文集，D-3-9，p.28(2004).

[23] 木下靖夫，片桐孝洋，本多弘樹，弓場敏嗣: SMP マシン上での BLAS ライブラリ用自動チューニング機構の設計と実装，電子情報通信学会総合大会論文集，D-3-10，p.29(2004).

[24] チューニング技法入門: 青山幸也，理化学研究所，情報基盤センター, 2004 年 5 月 26 日版，http://accc.riken.jp/HPC/training/text.html.

[25] はじめての並列プログラミング: 湯淺太一，安村通晃，中田登志之

編，1999 年 7 月 10 日版，共立出版株式会社.

[26] P スレッドプログラミング: Lewis，B. and Berg，J.D. 著，岩本信一訳，1999 年 1 月 10 日版，株式会社ピアソン･エデュケーション.

[27] コンピュータによる連立 1 次多項式の解法: Dongarra，J.J.，Duff，I.S. and van der Vorst，H.A 著，小国力訳，平成 5 年 1 月 30 日発行，丸善株式会社.

[28] ニューメリカルレシピ･インシー C 言語による数値計算のレシピ: Press, H.W., Teukolsky, A.S., Vetterling, T.W. and Flannery, P.B. 著，丹慶勝市, 奥村晴彦, 佐藤敏郎, 小林誠 訳, 平成 15 年 2 月 6 日版，技術評論社.

[29] ソフトウエア自動チューニング－数値計算ソフトウエアへの適応とその可能性: 片桐孝洋，2004 年 12 月 3 日，慧文社.

対外発表

査読つき論文

- 2005 年ハイパフォーマンスコンピューティングと計算科学シンポシウム(HPCS2004)
  「AutoTuned-RB:再帰 BLAS の自動チューニングの方式」
  木下靖夫・片桐孝洋・弓場敏嗣(電通大)
  情報処理学会，HPC 研究会主催
  (2005 年 1 月 18 日発表)

査読なし論文

- 2004 年電子情報通信学会総合大会
  「SMP マシン上での BLAS ライブラリ用自動チューニング機構の設計と実装」
  木下靖夫・片桐孝洋・本多弘樹・弓場敏嗣(電通大)
  電子情報通信学会主催
  (2004 年 3 月 22 日発表)

- 2004 年並列/分散/協調処理に関する『青森』サマー・ワークショップ(SWoPP 2004)
  「SMP 上における再帰 BLAS ライブラリの自動チューニング方式」
  木下靖夫・片桐孝洋・弓場敏嗣(電通大)
  情報処理学会，HPC 研究会主催
  (2004 年 8 月 1 日発表)

AutoTuned-RB

Version 1.00

利用マニュアル

２００５年1月１７日

電気通信大学大学院情報システム学研究科

木下　靖夫

AutoTuned-RB のライブラリインタフェース

AutoTuned-RB は、密行列積用のインストール時チューニングソフトウェアです。
使用できる関数は以下の RB_DGEMM です。

RB_DGEMM (TransA, TransB, M, N, K, a, A, lda, B, ldb, b, C, ldc)

各引き数の意味は以下のとおりです。L3BLAS ライブラリの DGEMM を同様なフォーマットです。

(C←a×Trans(A)×Trans(B)＋b×C) (1)

TransA: 行列 A が、ニュートラルなら“ n ”、転置なら“ t ” (char 型)
TransB: 行列 B が、ニュートラルなら“ n ”、転置なら“ t ” (char 型)
M: 行列 A、C の行の次元数(int 型)
N: 行列 B、C の列の次元数(int 型)
K: 行列 A の列の次元数、行列 B の行の次元数(int 型)
a: 上の式(1)における定数 a(double 型)
A: A 行列の要素の配列ポインタ(double 型)
lda: 行列 A の第一次元要素数(int 型)
B: B 行列の要素の配列ポインタ(double 型)
ldb: 行列 B の第一次元要素数(int 型)
b: 上の式(1)における定数 b(double 型)
C: C 行列の要素の配列ポインタ(double 型)
ldc: 行列 C の第一次元要素数(int 型)

RB_DGEMM は void 型です。

実行オプション

実行オプションの指定はありません。

はじめに

本ライブラリは以下のような特徴をもっています。

インストール時最適化の枠組みで、密行列積をおこなう際の性能安定性に関するパラメタを自動設定します。具体的には密行列を再帰分割し、再帰段数の自動設定を行っています。計算コアでは、BLAS ライブラリをコールしています。特徴として、1CPU マシンにおいては複数のサンプリング点、SMP マシンにおいては以下の３つのサンプリング点を使用しています。

(1) L1 キャッシュより大きい最小のサンプリングデータサイズ
(2) L3 キャッシュより小さく最大のサンプリングデータサイズ
(3) L3 キャッシュより大きく最小のサンプリングデータサイズ

この３点から、SMP マシン向けの自動チューニングを実現しています。

SMP マシンは、以下のチューニング効果が期待されます。

(1) 並列化による性能向上
(2) 行列サイズに依存する性能不安定性の改善

1CPU マシンには、(1)のチューニング効果が期待されます。

上記で述べた自動チューニング方式を、**AutoTuned-RB**（**Automatically Tuned Recursive BLAS**）と呼びます。

AutoTuned-RB では、本来全ての BLAS に関してチューニングする事が可

能ですが、トライアル版（試用版）として、ATLAS BLAS のみ対象となっています。

使用環境は以下の通りです。

(1) 1CPU、SMP マシン
(2) BLAS が使用できる OS
(3) C コンパイラ
(4) Posix Thread

L3 キャッシュを搭載しているマシン向けに最適化をかけていますが、L1、L2 キャッシュ搭載マシン向けの最適化も行っています。

なお本試用版は、多くのユーザに利用してもらうことで、ユーザビリティの改善、バグの発見と修復、および新規開発事項への反映、を目的にリリースされるものです。

したがってマニュアルにおいての説明不備、低いコードのリーダビリティ、および突然の仕様変更、などの問題が生じる可能性があることをご理解ください。

また何かお気づきの点がありましたら、お気軽に著作者までご一報くださいますようお願い申し上げます。

自動チューニング項目

ここでは、本ライブラリで実装されているチューニング方式について説明します。

AutoTuned-RB のチューニング方式は以下の方式を取っています。

- インストール時チューニング
  ライブラリをインストールする際にチューニングを実行

SMP マシン、1CPU マシン共にチューニングパラメタは再帰段数です。

インストール手順

手順は、以下の３つです。

(1) Makefile の設定
(2) make config の実行
(3) Config file 作成後 make install

##
#####

Makefile の設定例
##
#

```
#      AutoTuned-RB Install Makefile
#
#
#
#
########################################

SHELL = /bin/sh
ARCH = Linux_P4SSE2
アーキテクチャ名(例:ATLAS にインストールした際に作成されたライブラリが入っているフォルダ名)

########################################

TOPdir = /home/kinoshita
ATLdir =$(TOPdir)/ATLAS
ATLlibdir = $(ATLdir)/lib/$(ARCH)

########################################
(AutoTune-RB をインストール先の指定)
ATRBdir = $(TOPdir)/ATRB
ATRBobjdir = $(ATRBdir)/source/$(ARCH)
ATRBsrcdir = $(ATRBdir)/source
ATRBincdir = $(ATRBdir)/include/$(ARCH)
ATRBlibdir = $(ATRBdir)/lib/$(ARCH)
#ATRbindir = $(TOPdir)/bin/$(ARCH)
```

```
########################################

CC = cc
CCFLAGS = -O3
NM = -o
OJ = -c

########################################

ARCHIVER = ar
ARFLAGS    = r

########################################

LATL = -latlas
LPTH = -lpthread
#Math = -lm

########################################
all:install

config:
        rm -f cofig
        $(CC) -lm    config.c $(NM) config
        ./config
        mkdir -p $(ATRBincdir)
```

```
	mv Confg.h $(ATRBincdir)/
	rm -f config
install:$(ATRBsrcdir)/ABCLib_BLAS_Src.c
$(ATRBsrcdir)/ABCLib_BLAS_Rec
ursive.c
	rm -f $(ATRBlibdir)/*.a
	$(CC) $(ATRBsrcdir)/ABCLib_BLAS_Src.c $(NM) ¥
	ABCLib_BLAS_Src $(CCFLAGS) -I$(ATRBincdir) -L$(ATLlibdir) ¥
	$(LATL) $(LPTH)
	./ABCLib_BLAS_Src
	mv Mtdev.h $(ATRBincdir)/
	rm -f ABCLib_BLAS_Src
	$(CC) $(OJ) -I$(ATRBincdir)
$(ATRBsrcdir)/ABCLib_BLAS_Recursive.c
	mkdir -p $(ATRBobjdir)
	mv $(ATRBdir)/ABCLib_BLAS_Recursive.o $(ATRBobjdir)/
	mkdir -p $(ATRBlibdir)
	$(ARCHIVER) $(ARFLAGS) $(ATRBlibdir)/libatr.a
$(ATRBobjdir)/¥
	ABCLib_BLAS_Recursive.o

.PHONY : cleanall cleanbin cleanobj cleanlib cleanhead

cleanall: cleanbin cleanobj cleanlib cleanhead

cleanbin:
	rm -f $(ATRBdir)/ABCLib_BLAS_Src
```

```
	rm -f $(ATRBdir)/config

cleanobj:
	rm -f $(ATRBobjdir)/*.o
	rmdir $(ATRBobjdir)

cleanlib:
	rm -f $(ATRBlibdir)/*.a
	rmdir $(ATRBlibdir)

cleanhead:
	rm -f $(ATRBincdir)/*.h
	rmdir $(ATRBincdir)

#####################################################################
####
```

Makefile 設定後に、コマンド make config を実行します。ここで入力するデータは、

(1) SMP サポート
(2) CPU 台数
(3) L3 キャッシュサポート
(4) L1 キャッシュサイズ
(5) L3 キャッシュサイズ(L2 キャッシュサイズ)
(6) メインメモリサイズ

の６つです。これらを入力すると Configure file が作成されます。

Configure file 作成後にコマンド make install を実行すると AutoTuned-RB のチューニングが実行されます。インストール時間は計算機環境に依存しますが、経験的には SMP マシンで 10 分程度、1CPU マシンでは 2 時間程度です。

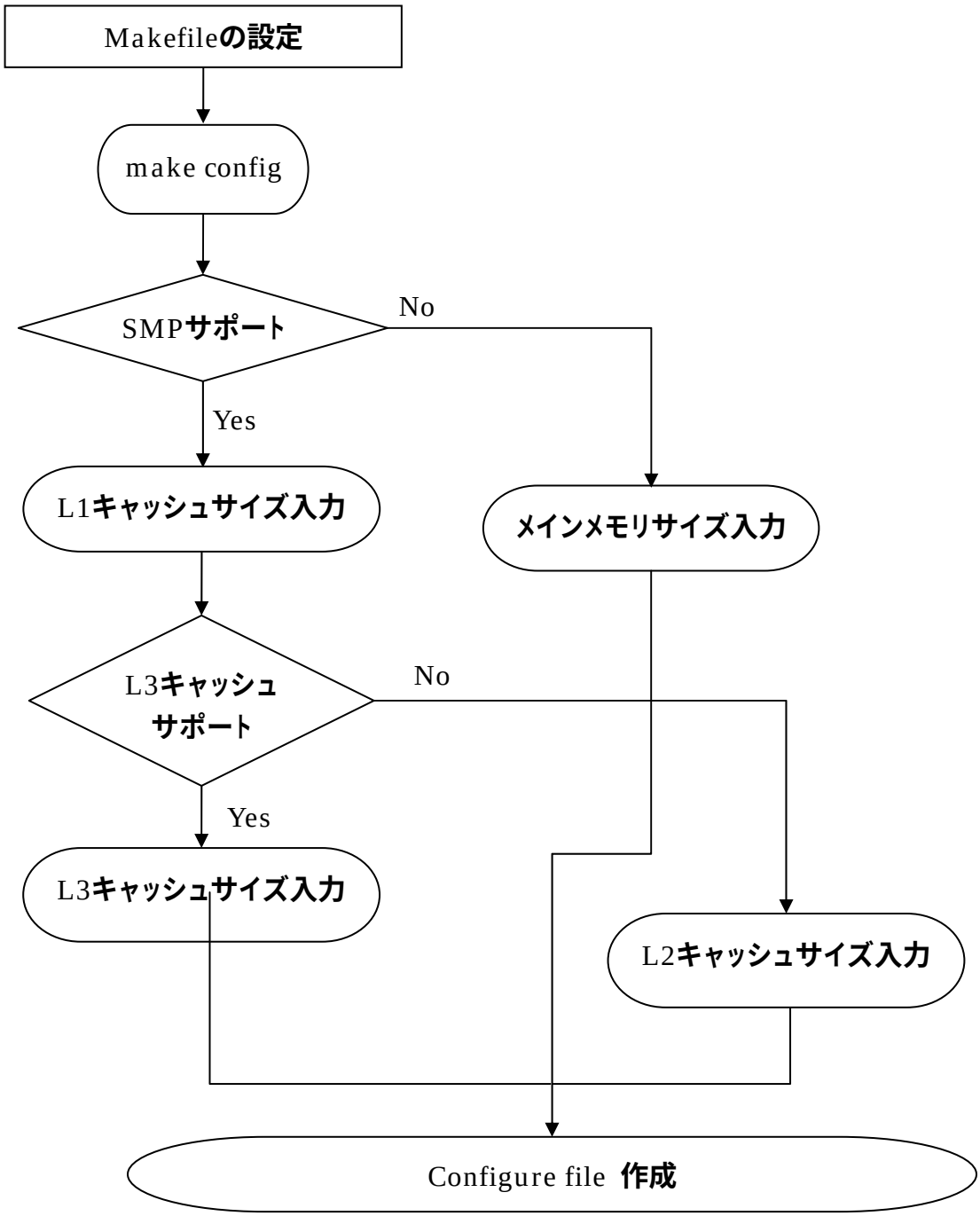


図１ Configure file**作成フロー**

ディレクトリ構造

- ATRB/

  Makefile

  Config.c
- ATRB/source

  ABCLib_BLAS_Recursive.c

  ABCLib_BLAS_Src.c
- ATRB/source/<arch>

  ABCLib_BLAS_Recursive.o

  Recursive_p.o
- ATRB/include/<arch>

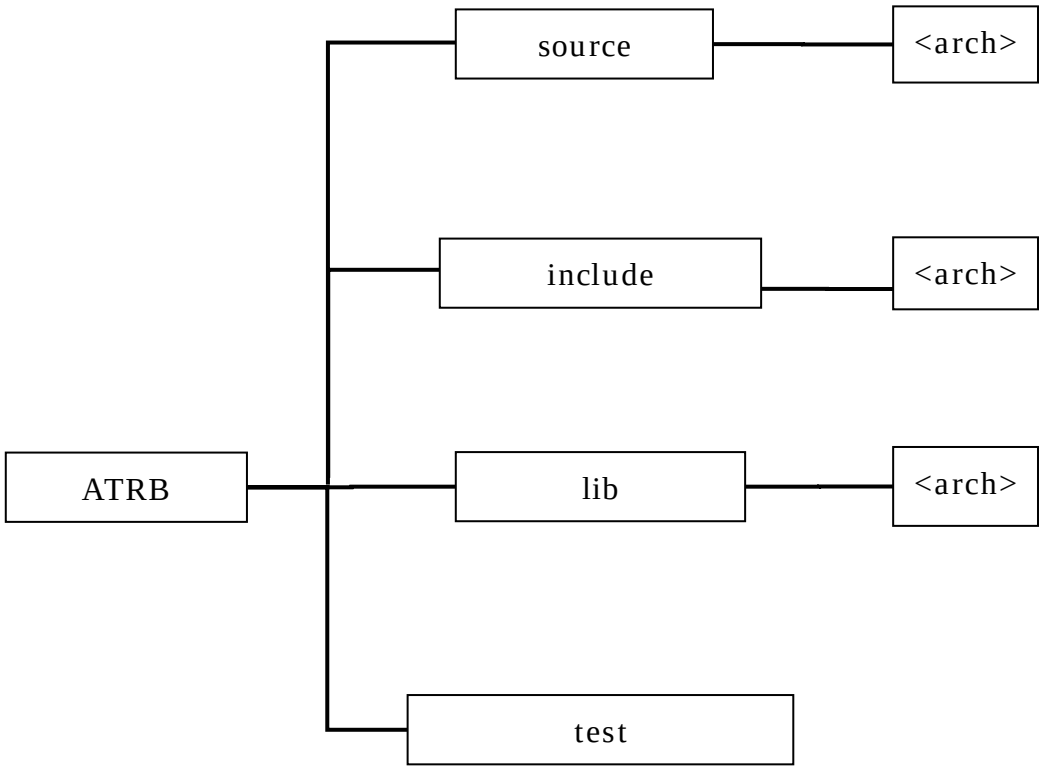
  Confg.h

  Mtdev.h
- ATRB/lib

  Libatr.a
- ATRB/test

  Makefile

  ABCLib_BLAS_Test.c



図Ⅱ　AutoTuned-RB ディレクトリ構造

サンプルプログラム

密行列積を求めるためのサンプルプログラム（ **test/ABCLibTestBLAS.c** ）は、以下のとおりです。MATRIXSIZE を変更することで計算する行列の次元を変更することができます。

```
/*****************************************/
/*                                       */
/*          Posix Pthread version        */
/*      Recursive BLAS Matrix Mutmal     */
/*                                       */
/*              Yasuo Kinoshita          */
/*                                       */
/*****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include <time.h>

#define MATRIXSIZE 4000

int main(int argc、char **argv)
{
   int M、N、K;
   double *C、*A、*B;
```

```
char TransA='n'、TransB='n';
double a=1.0、b=0.0;
int lda、ldb、ldc;
int i、j、x;
struct timeval t1、 t2;
double soltime、sec、usec;
time_t seed;
time(&seed);
srand(seed);
M=MATRIXSIZE;
N=M;
K=M;
lda = K;
ldb = N;
ldc = M;

printf("MatrixC:%d*%d "、M、N);
printf("MatrixA:%d*%d "、M、K);
printf("MatrixB:%d*%d¥n"、K、N);

C = (double *)malloc(sizeof(double)*(M*N));
A = (double *)malloc(sizeof(double)*(M*K));
B = (double *)malloc(sizeof(double)*(K*N));

x=10;

for(i=0 ;i<M ;i++ ){
```

```
    for(j=0 ;j<N ;j++ ){
        *(C+j+i*N) = 0.0;
    }
}

for(i=0 ;i<M ;i++ ){
    for(j=0 ;j<K ;j++ ){
        *(A+j+i*K) =(double)(rand()%x);
    }
}

for(i=0 ;i<K ;i++ ){
    for(j=0 ;j<N ;j++ ){
        *(B+j+i*N) =(double)(rand()%x);
    }
}

gettimeofday(&t1、 NULL);
RB_DGEMM(TransA、TransB、M、N、K、a、A、lda、B、ldb、b、C、
ldc);
gettimeofday(&t2、 NULL);

sec = t2.tv_sec - t1.tv_sec;
usec = t2.tv_usec - t1.tv_usec;
soltime = (sec + usec/1000000.0);
printf("Solve time =  %0.3lf¥n"、soltime);
printf("  flops  =  %0.3lf¥n"、2*M*M*(M/soltime)/1000000);
```

```
    free(C);
    free(A);
    free(B);


    return 0;

}
##########################################################################
####
```

実行例

以下に、AutoTuned-RB のインストール実行例をのせます。
なお以下の出力例は、バージョン変更等により形式が変わることがありますので、ご注意ください。
この例でのインストールは、２CPU の SMP マシンで行いました。

```
##############################################################################


kinoshita@opt01:~/ATRB> make config
rm -f cofig
cc -lm    config.c -o config
./config
###############################################
                        ABCLib-BLAS
                    version      ver.1.0
               composed by Yasuo Kinoshita

   Graduate School of Information Systems、
   The University of Electro-Communications
      /JAPAN SCIENCE AND TECHNOLOGY AGENCY
                        2004/01/16

                  AutoTuned-RB Configure
###############################################
```

========== make Confg.h ==========

SMP SUPPORT[y/n]: y

Input Number of CPU : 2

======== Sampling point Setting ==========

Input L1Cache Size[KByte]: 64

L3Cache Machine?[y/n]: n

Input L2CacheSize[KByte]: 1024
Configration Completed!!
Type "make install" if you continue install

mkdir -p /home/kinoshita/ATRB/include/Linux_UNKNOWNSSE2_2
mv Confg.h /home/kinoshita/ATRB/include/Linux_UNKNOWNSSE2_2/
rm -f config

kinoshita@opt01:~/ATRB> make install
rm -f /home/kinoshita/ATRB/lib/Linux_UNKNOWNSSE2_2/*.a
cc /home/kinoshita/ATRB/source/ABCLib_BLAS_Src.c -o ¥
ABCLib_BLAS_Src -O3
-I/home/kinoshita/ATRB/include/Linux_UNKNOWNSSE2_2

```
-L/home/kinoshita/AutoTuned-RB/ATLAS/lib/Linux_UNKNOWNSSE2_2
-latlas -lpthread
./ABCLib_BLAS_Src
###############################################
                        ABCLib-BLAS
                    version      ver1.0
               composed by Yasuo Kinoshita

   Graduate School of Information Systems、
   The University of Electro-Communications
      /JAPAN SCIENCE AND TECHNOLOGY AGENCY
                        2005/01/16

    AutoTuned-RB Install-time Optimization
###############################################

########## SMP Machine Tuning ##########

########## Near L1 Cache size      #########

   MATRIX SIZE 123 * 123
    RNUM        TIME       MFLOPS
==============================
       0          0.002     1542.6
       1          0.002     1602.1
       2          0.003     1487.1
       3          0.003     1400.2
```

```
##########    In L3 Cache size      #########

   MATRIX SIZE 395 * 395
    RNUM        TIME       MFLOPS
==============================
       0          0.065     1906.9
       1          0.039     3197.5
       2          0.041     3021.7
       3          0.043     2899.4
       4          0.050     2448.9
       5          0.065     1901.1

   Matrixsize         OptiNum
=========================
      123                  1
      395                  1

Tuning Completed !

mv Mtdev.h /home/kinoshita/ATRB/include/Linux_UNKNOWNSSE2_2/
rm -f ABCLib_BLAS_Src
cc          -c          -I/home/kinoshita/ATRB/include/Linux_UNKNOWNSSE2_2
/home/kinoshita/ATRB/source/ABCLib_BLAS_Recursive.c
mkdir -p /home/kinoshita/ATRB/source/Linux_UNKNOWNSSE2_2
mv                          /home/kinoshita/ATRB/ABCLib_BLAS_Recursive.o
/home/kinoshita/ATRB/source/Linux_UNKNOWNSSE2_2/
```

```
mkdir -p /home/kinoshita/ATRB/lib/Linux_UNKNOWNSSE2_2
ar r /home/kinoshita/ATRB/lib/Linux_UNKNOWNSSE2_2/libatr.a
/home/kinoshita/ATRB/source/Linux_UNKNOWNSSE2_2/ABCLib_BLAS_Rec
ursive.o
kinoshita@opt01:~/ATRB>

########################################################################
####
```

サンプルプログラム(**test/ABCLibTestBLAS.c**)

行列 1000×1000 の実行例

```
kinoshita@opt01:~/ATRB/test> make dgemm
cc ABCLib_BLAS_Test.c -o ABCLib_BLAS_Test -O3
-L/home/kinoshita/AutoTuned-RB/ATLAS/lib/Linux_UNKNOWNSSE2_2 ¥
-L/home/kinoshita/ATRB/lib/Linux_UNKNOWNSSE2_2 -latr -latlas -lpthread
-lm
kinoshita@opt01:~/ATRB/test> ./ABCLib_BLAS_Test
MatrixC:1000*1000 MatrixA:1000*1000 MatrixB:1000*1000
s = 2
Solve time =    0.509
    Mflops     =    3926.897
kinoshita@opt01:~/ATRB/test>
```

注) test/makefile の内容を適宜変更して使用してください。

おわりに

本マニュアルでは、BLAS3 用の再帰 BLAS における自動チューニングソフトウエア AutoTuned-RB の利用法について説明しました。

なお、以下のオンラインマニュアルでも、本マニュアルの内容を閲覧できます。

http://www.abc-lib.org/online/abclib.htm