計算機アーキテクチャ 123－1
（1997. 3. 7）

VPP300/500 における BLAS のベクトル化

傳田紀代美†, 山口あづさ †, 竹重和明 ‡

† 理化学研究所計算科学研究室, ‡ 富士通 (株)HPC 本部第一開発統括部第二開発部

VPP300/500 において BLAS(Basic Linear Algebra Subprograms) のベクトル化を、富士通 HPC 本部との共同研究 / 開発として行った。今回の仕事は、並列ベクトル化 LAPACK を VPP300 へ実装するための第一歩として位置づけられ、アーキテクチャに最適化したベクトル化 BLAS を作成して、BLAS を参照している数値計算ライブラリ LAPACK(Linear Algebra PACKage) の性能向上を目指すことを目的としている。アーキテクチャの特性を考慮し、ベクトルレジスタを最も有効に利用するようにソースを改良した。また行列サイズに応じて最も速いアルゴリズムを選択させることで、広い範囲で高性能が得られるようになった。

Vectorization of BLAS for VPP300 and VPP500

Kiyomi Denda†, Azusa Yamaguchi†, Kazuaki Takeshige‡

† Computational Science Laboratory, RIKEN, ‡ FUJITSU LIMITED

We have vectorized BLAS in order to be fit for VPP300 or VPP500. This has been done as a joint research between Computational Science Laboratory of RIKEN and FUJITSU LIMITED. This is the first step to our final goal of this work; a development of a vectorized-parallel LAPACK for VPP300. The optimization of BLAS results in a high-performance LAPACK, because LAPACK refers BLAS as subroutines. We have revised BLAS in order to use the fastest algorism case by case and to make use of the vector registers effectively. We have thus obtained a good performance in a wide range of matrix sizes.

1 はじめに

LAPACK は、線形方程式系や固有値問題などの線形代数問題に対して、数値的な解法を与えるサブルーチン ライブラリであり、行列やベクトルの演算などの基本的な演算は BLAS をコールしている。VPP シリーズには、既に同様なサブルーチンライブラリ SSL II（ベクトル化版及び並列化版）が実装されているが、より多くのユーザーのニーズを満たすために、ベクトル化版及び並列化版 LAPACK も実装が予定されている。本稿では、そのための第一歩として理研と富士通の共同で行った、VPP300 と VPP500 での BLAS のベクトル化について報告する。

富士通 VPP300/VPP500 は分散メモリ型のベクトルパラレル スーパーコンピュータであり、その性能は表 1 のようになる。ベクトルレジスタは、レジスタの個数と各レジスタ上に保持できるベクトル長の積が一定の範囲で、その構成を変えることができる。また、VPP500 が素子として GaAs を使用しているのに対し、後継機である VPP300 は CMOS テクノロジーを採用している。

2 ベクトル化版 BLAS

LAPACK と BLAS のソースは netlib[1] によって WWW 上で公開されているが、そのソースは移植性の高いものであるために、必ずしも個々のマシンのアーキテクチャ特性に適してはいない。そこで VPP300 及び VPP500 の特性に適合したベクトル化を行うのが、今回報告する仕事の目的である。ただし、netlib 版 BLAS をチューニングするのではなく、VP2200 用のベクトル化 BLAS(BLASVP_0) をプロトタイプとし、チューニングを行った。

BLAS は、ベクトル同士の演算を扱うレベル 1、ベクトルと行列の演算を扱うレベル 2、行列同士の演算を扱うレベル 3 と疎行列などを扱うスパースの 4 つに分けられる。それぞれ単精度実数、倍精度実数、単精度複素数及び倍精度複素数を扱うプログラムのグループを持ち、全部で 150 本以上のプログラムセットである。プロトタイプ版 BLASVP_0 ではさらに、それぞれのプログラムがアルゴリズムのタイプや行列、ベクトルの扱い方の種類（転置するとか、しないなど）に依って、さらに複数のサブルーチンに分れている。

最適なアルゴリズム選択を判断させる部分はアーキテクチャ特性に対して依存度が高く、今回作成した改良版ベクトル化 BLAS(BLASVP_1) での代表的な改良点の一つである。また、VPP300 のベクトルレジスタサイズを最も有効に利用したオブジェクトを得るために、ソースの改良を行った。

3 最適なアルゴリズム選択

BLASVP_0 及び BLASVP_1 では、次に述べる方法で最適なアルゴリズムを評価している。

3.1 一般的な場合

一般にベクトルや行列の演算では、データのアクセスの仕方から二つのアルゴリズムが考えられる。すなわち、データ アクセスが列優先のアルゴリズムと行優先のアルゴリズムである。Fortran では、データの連続

表 1: PE 主要緒元[2]

	VPP300	VPP500
マシンサイクル時間	7nsec	10nsec
ピーク性能 /PE	2.2GFLOPS	1.6GFLOPS
ベクトルレジスタ	128KB	128KB
主記憶メモリ容量	2GB/PE (SDRAM)	256MB/PE* (SRAM)

*; 最大値

アクセス方向は列方向となっているが、行列の形によっては必ずしも列優先アクセスが得になるとは言えない（行数に比べて列の数が非常に大きい場合など）。そこで次のような評価式を用いて、最適アルゴリズムの選択を行う。

それぞれのアルゴリズムの命令実行のタイムチャートを作成し、それを基に実行時間の評価式を用意しておく。この評価式は、行列（ベクトル）のサイズと、DO ループの立ち上がり時間などの基本パラメタやバンク コンフリクトによるペナルティの関数として表せる。この評価式に行列のサイズや各パラメタを代入し、実行時間が短いと評価されるアルゴリズムを選ぶ。ここで、DO ループの立ち上がり時間などの基本パラメタは given とする。データ アクセスに関するペナルティは、ロードのみの場合とロードとストアの両方を行う場合の二種類についてバンク コンフリクトの影響を実測したデータを利用する。

1) 列方向優先アクセス

```
do j=1,n
 do i=1,m
   a(i,j)=a(i,j)+alpha*x(i)*y(j)
 end do
end do
```

実行時間の評価式

```
T1=n*(T0+2/LSPIPE*1.15*m/RVP)
```

2) 行方向優先アクセス

```
do i=1,m
  do j=1,n
    a(i,j)=a(i,j)+alpha*x(i)*y(j)
  end do
end do
```

実行時間の評価式

```
T2=m*(T0+2/LSPIPE*FACT*n/RVP)
```

LSPIPE; ロード / ストアパイプライン数、RVP; 出力要素数 /1 サイクル
FACT; バンクコンフリクトによるペナルティ

3.2 GEMM 型

行列同士の掛け算では、大別して内積型、中間積型や外積型の三通りのアルゴリズムが考えられるが、どのアルゴリズムが個々の問題において最適であるかは扱う行列のサイズなどの条件によって異なる。一般的な行列同士の掛け算を扱う GEMM 系では、行列 A、B、C について次のような演算を行う。

$$C = \alpha * \mathrm{op}(A) * \mathrm{op}(B) + \beta * C \tag{1}$$

α, β はスカラー定数、行列の次元は op(A)(M,K), op(B)(K,N), C(M,N) である。op は実数の場合、転置なし (N), 転置 (T) の二種類がある。複素数ならばエルミート共役を取る (C) 場合も加わって三種類となる。

BLASVP_0 と BLASVP_1 では、行列サイズの組み合わせ (M,N,K) と行列 op(A) の整合寸法の組み合わせの代表的な場合に、各アルゴリズムに対応するサブルーチンの実行時間を実測し、最適なアルゴリズムを判断させる関数表をあらかじめ用意している。実際の計算を行う時は、その関数表を参照して最適なアルゴリズムを選択する。

3.3 三角行列型

三角行列と行列の掛け算を扱う TRMM 系や線形方程式を解く TRSM 系では、もう少し複雑なアルゴリズム選択を行っている。列優先アクセス型と行優先型アルゴリズムの実行時間の評価式の大小関係を比較し、列方向に優先アクセスした方が得な部分の大きさを求め、行列を分割して演算を行う。

4 ソースのチューニング

BLASVP_0 はベクトルレジスタサイズやメモリの小さいマシンを対象にチューニングされていたために、想定されている問題のサイズは比較的小さい。すなわち、行列やベクトルのサイズが大きい問題を扱う場合に、

適当でないコーディングとなってしまっている場合があった。また、ハード コーディングによるループ アンローリングに加えてコンパイラによってさらにループ アンローリングが行われてしまい、ベクトルレジスタサイズが不必要に小さいオブジェクトが生成されるという問題点もあった。時間的な制約もあったため今回はソースの大幅な書き直しは行わずに、次の2点について改良を行った。（以下最大ベクトル長をMaxVLと表す。）

4.1 最大ベクトル長によるDOループの分割

扱う問題のサイズが最大ベクトル長を越えた場合、DOループをMaxVLで分割することで、より良い性能が得られることは周知の事実である。このように分割することで、問題のサイズがMaxVLからMaxVL+1に変化したときの性能の低下を最小限に食い止めることができる。

```
* BLASVP_0のソース
DO  J = 1, N-1 , 2
 DO  I = 1, M
  A(I,J)=A(I,J)+X(I)*Y(J)*ALPHA
  A(I,J+1)=A(I,J+1)+X(I)*Y(J+1)*ALPHA
 END DO
END DO
```

```
* BLASVP_1のソース
      do ii=1, m, maxvl
       m1=min(ii+maxvl-1, m)
       DO J = 1, N-1 , 2
*vocl loop, repeat(maxvl)
        DO I = ii, m1
          A(I,J)=A(I,J)+X(I)*Y(J)*ALPHA
          A(I,J+1)=A(I,J+1)+X(I)*Y(J+1)*ALPHA
        END DO
       END DO
      end do
```

*vocl loop,repeat(maxvl) は直後のDOループの最大回転数がMaxVLであることを指示する最適化行である。

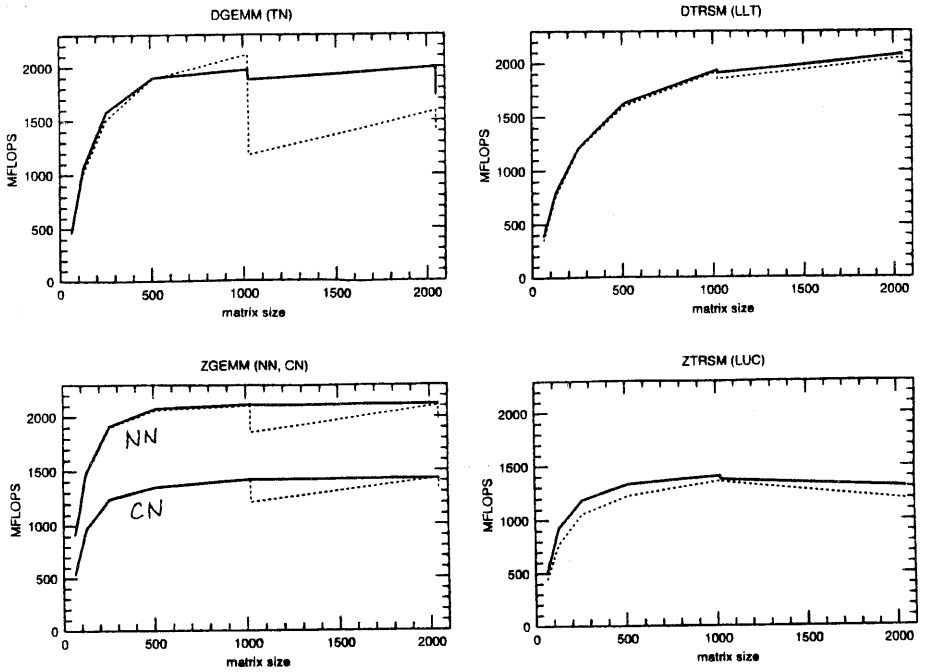
4.2 過剰なループ アンローリングの抑止

何重にもループアンローリングが行われると、ロードすべき変数の数が増加し、ベクトルレジスタの数が足りなくなる。そのためベクトル長の短いレジスタ構成を指定したオブジェクトができるが、行列（ベクトル）のサイズが大きい場合の性能はあまり期待できなくなる。

例えば、DGEMMのTN（行列Aを転置して行列Bに掛ける場合）では最大ベクトル長のレジスタ構成を想定したソースになっているが、実際のベクトルレジスタのサイズはMaxVLの半分になっていた。コンパイルリストを見ると、過剰なループアンローリングをしていることが分る。そこで121番と122番のDOループの開始行の直前にループアンローリングを抑制する最適化行を付け加えることで、各ベクトルレジスタのベクトル長が最大となり、行列サイズが大きい場合の性能も向上した。

```
00002203 s              DO 122, L = 1, K-3, 4
00002204 s2               DO 121, J = 1, N
00002205 v2                 B0=ALPHA*B(L,J)
00002206 v2                 B1=ALPHA*B(L+1,J)
00002207 v2                 B2=ALPHA*B(L+2,J)
00002208 v2                 B3=ALPHA*B(L+3,J)
00002209      *VOCL LOOP,REPEAT(MAXVL)
00002210 v2                 DO 120, I = 1, MMOD
00002211 v2                   C( I, J ) = C( I, J ) + B0 * A(L,I)
00002212           &                                  + B1 * A(L+1,I)
00002213           &                                  + B2 * A(L+2,I)
00002214           &                                  + B3 * A(L+3,I)
00002215 v2     120         CONTINUE
00002216 v2     121       CONTINUE
00002217 v      122     CONTINUE
```

図 1: BLASVP_0(点線) と BLASVP_1(実線) の性能値の比較例。いずれも VPP300 での結果。



また、複素数の場合はデータのロード / ストア量が多くなるので、実数の場合と異なり最大可能なベクトル長は MaxVL の半分となってしまう。そこで複素数に関しては、レジスタサイズが MaxVL の半分になるよう最適化を行った。

5 結果

これまで述べた方法で、なるべく良い性能値をなるべく広い行列のサイズで達成できるように改良を行った。その際、ある特定のサイズで性能値が急激に落ち込むなどのがたつきがでないような配慮も行った。チューニング結果の具体例として、一般的な行列の掛け算を扱う GEMM 系と線形方程式の解を求める TRSM 系について、倍精度実数 (D____) と倍精度複素数 (Z____) の場合の結果を図 1 に示す。図から分るように、程度の差はあるが BLASVP_1 のほうが性能のがたつきが小さくなっている。

複素数の場合は、実数の場合と異なり、データのアクセスをたとえ列優先としても変数を複素数として扱う限りストライド アクセスとなるために、良い性能が出にくい。そこで、BLASVP では複素数の行列 A(M,N) を実数の行列 A(M*2,N) として扱うことで、ストライド アクセスに因る性能の低下を防いでいる。ただし、この方法はどの場合にも適用できて、良い性能値が得られる訳ではない。ZGEMM 系で行列 A、B を転置せずに掛ける場合 (NN) と A のエルミート共役を取ったものを B に掛ける場合 (CN) で最大の性能値に差が出ているが、それはこのためである。ZGEMM(NN) の場合は、上記の方法が適用できるので、ピーク性能を達成している。

6 結び

前節で示した結果は数例ではあるが、他のケースでも良い性能値が行列サイズの広い範囲で得られるようになった。最適なアルゴリズムの選択則をどう評価するかが、一番苦労したところである。タイムチャートに基づく評価式は、ある理想的な条件の下で列優先アクセスのアルゴリズムと行優先のアルゴリズムの実行時間の大小を判断する。ただし、実際に演算が行われる時にはchainingのため、先のステップのストアと次のステップのロードがコンフリクトして余分な時間がかかる場合があり、評価式を経験的に変更する必要がある。今後BLASVP_1の並列化を進めるにあたって、最適なアルゴリズムをどうやって選択させるかは重要な検討課題となると予測している。

また本稿では触れなかったが、動作性に関しては、代表的な行列サイズと係数（DGEMMで言うならばαやβ）の組合わせ（100通り以上）で確認している。ごく単純なアルゴリズムで計算して「正解」を求めておき、BLASVP_1で計算した値との誤差が計算機イプシロン（動作性テストをする毎に計算して求める）に対して何倍であるかで、計算結果の正当性を確認している。

今回の仕事を進めるにあたって、富士通(株)HPC本部の伊奈 博氏と理研計算科学研究室の戎崎俊一氏から多数の有益なコメントを頂いたことに、感謝している。

1) http://phase.etl.go.jp/ からnetlibのホームページへたどることができる。LAPACK,BLASに関するドキュメント及びソースの入手可能。
2) http://www.fujitsu.co.jp/hypertext/Develop/magazine/vol47-6/index.html にVPP300の情報が掲載されている。