Vol.2012-HPC-133 No.25
2012/3/27

多倍長疎行列積を用いたKrylov部分空間法の性能評価

幸谷智紀[†1]

本稿では多倍長疎行列・ベクトル積を用いたKrylov部分空間法の性能評価を行う．我々は任意精度浮動小数点演算ライブラリMPFR/GMPをベースにした多倍長数値計算ライブラリBNCpackを構築してきた．これには疎行列を扱う機能がなかったため，今回それを実装し，行列・ベクトル積の高速化が達成できることをベンチマークテストによって確認する．また，Krylov部分空間法にこれを適用し，UF疎行列コレクションで提供されているような大規模問題がメモリ制約のある環境でも高速に解けることを示す．

Performance Evaluation of Krylov Subspace Methods using Multiple Precision Sparse Matrix-Vector Multiplication

Tomonori Kouya[†1]

We evaluate the performance of Krylov subspace method by using multiple precision sparse matrix-vector multiplication(SpMV). BNCpack is our own multiple precision numerical computation library based on MPFR/GMP, which is one of the most efficient arbitrary precision floating-point arithmetic libraries. This does not have functions which manipulate multiple precision sparse matrices, so we show that the SpMV in these functions we have implemented can be more efficient by using benchmark tests. Finally, we also show that Krylov subspace methods such as BiCG and GPBiCG in which we have embedded the SpMV can efficiently solve large-scale linear systems of equations provided by the UF sparse matrix collections in memory-restricted computing environment.

†1 静岡理工科大学
Shizuoka Institute of Science and Technology

1. 初 め に

現在，大規模な科学技術計算のニーズは増え続け，標準的な倍精度浮動小数点演算ではフォローしきれない悪条件問題が顕在化している．浮動小数点演算精度の不足を補うための多倍長計算のニーズも増えており，実用的も四倍，八倍精度が必要なケースを見聞きすることが増えてきている．そのようなニーズに対応するため，様々な多倍長数値計算ライブラリの開発が行われている．これらのライブラリは，高速な基本多倍長浮動小数点演算の機能の上に，数値計算アルゴリズムを積み上げる形で実装されている．CPUやGPUの，特に並列計算性能の向上に伴い，ユーザの要求する精度で，かなり大規模な数値計算が実行できる環境が個人のPC上で可能な時代になっている．

しかし，精度の向上に伴って要求される記憶領域も莫大なものとなりつつある．メインメモリとプロセッサ内部のキャッシュメモリとのアクセス速度が格段に開きつつある昨今では，計算には必要最小限のメモリ使用量で済ませることが求められる．ことに，基本線型計算では疎行列を非零要素のみ扱えるようにすることが重要である．倍精度計算用としては既にSuiteSparse[7]やIntel Math Kernel Library(IMKL)[4]に，基本線型計算から直接法，反復法のルーチンまで取り揃えられている．LAPACK[3]のソースを土台にして多倍長化したMPACK[8]のように，これらをベースにして多倍長化することも可能と思われるが，今回我々は多倍長数値計算ライブラリBNCpack[2]の線型計算機能をベースに，行列・ベクトル積のみサポートした多倍長疎行列機能を追加することにした．

BNCpackが土台にしているMPFR[6]/GMP[1]は零演算を高速化する機能が備わっており，疎行列計算を実装してもメモリの節約には繋がりこそすれ，どの程度の計算時間の短縮に繋がるか，皆目見当が付かなかった．今回，多倍長疎行列・ベクトル積を実装して性能評価を行ったところ，思いのほか効用があることが分かった．その結果，Krylov部分空間法の計算時間短縮も可能となり，メモリの少ない貧弱なPC環境でもある程度大規模な計算が実現できた．

本論文ではまず実装した疎行列計算の機能を，土台となったMPFR/GMPとBNCpackの機能の説明から行う．次にPC環境で疎行列・ベクトル積のベンチマークテストを行い，性能がどの程度上がっているかを検証する．最期にこれを積型Krylov部分空間法に適用し，その有効性を確認する．

2. BNCpack の疎行列形式

現在の BNCpack がサポートしている多倍長数値計算機能は，多倍長浮動小数点演算ライブラリ MPFR/GMP(MPFR over MPN kernel of GMP) の上に構築されている．特長として

- IEEE754-1985 が規定する非数・無限大・丸めモードをサポートした任意精度浮動小数点演算が可能
- 浮動小数点数型 (`mpfr_t` 型, BNCpack では `mpf_t` として GMP の浮動小数点数型と共通化している) は C の構造体として定義されており (**図 1**)，仮数部はポインタとして分離して定義できる
- 混合精度演算が自在に可能．実行途中での精度変更も可能
- 独自の高速化機能があり，例えば固定精度型 (整数型，float, double 等) やゼロとの演算は極めて高速に実行できる

ということが挙げられる．

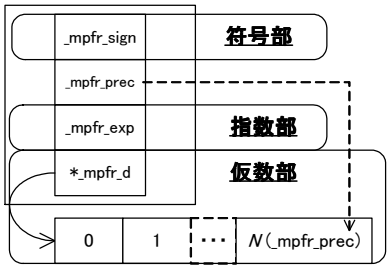

図 1　MPFR/GMP の構造体
Fig. 1　Structure of MPFR/GMP variable.

多倍長数値計算の機能はこの MPFR/GMP をベースに実装してある．今回実装した Compressed Sparse Row (CSR) 形式の多倍長疎行列の要素 (`element`) もこれを用いている．疎行列を定義した構造体は以下のようになる．

```
typedef struct {
    unsigned long prec;          // 精度 (bit)
    mpf_t *element;              // 行列成分
    long int row_dim, col_dim;   // 次元数
    long int **nzero_index;      // 非零要素の添字
    long int *nzero_col_dim;     // 非零数 (行)
    long int *nzero_row_dim;     // 非零数 (列)
    long int nzero_total_num;    // 非零要素数
} mpfrsmatrix;
```

これを使うと，例えば 5 × 5 のランダム疎行列

$$\begin{bmatrix} 0 & 1 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

は，**図 2** のような形式で格納できる．

行列の次元数が十分大きければ，記憶領域は零要素の分だけ確実に減らせることから，メモリ制限のために読み込めなかったサイズの疎行列でも扱えるようになる．しかし，前述したように MPFR/GMP の演算省力化機構の働きにより，計算時間がどの程度削減できるかはベンチマークテストを行ってみなければ分からない．

今回は Krylov 部分空間法に適用するため，多倍長実疎行列のデータ型の定義と，実疎行列・実ベクトル積のみ実装した．疎行列以外のデータ型の定義は既存の BNCpack のものをそのまま利用している．現状ではまだデバッグやテストが十分とは言えないが，ソースコードは全て公開してあるので，詳細は BNCpack 最新版[2] を参照されたい．

3. 疎行列・ベクトル積 (SpMV) の性能評価

性能評価は次の環境で行った．並列計算は行っておらず，全てシングルスレッドでの逐次計算である．

H/W　Intel Core i7 920, 8GB RAM
S/W　CentOS 5.6 x86_64, gcc 4.1.2
MPFR/GMP　MPFR 3.0.1, GMP 5.0.1
BNCpack　BNCpack 0.7

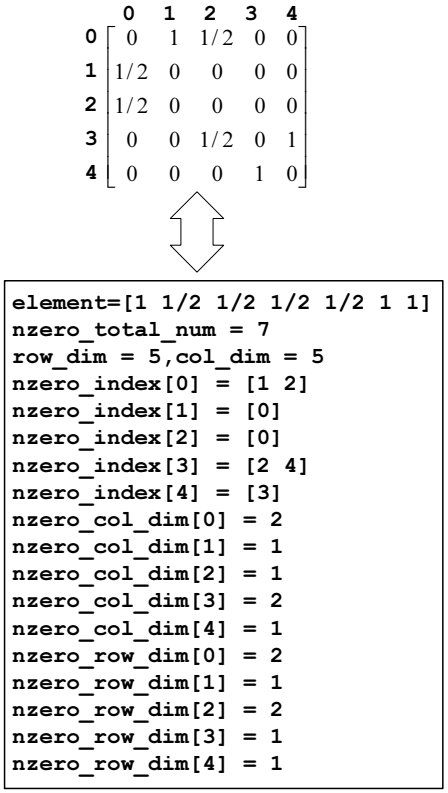

図 2　BNCpack の疎行列形式
Fig. 2　Structure of Sparse Matrix implemented in BNCpack.

疎行列は，仮数部の桁が全て埋まるよう，次の Lotkin 行列

$$A = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1/2 & 1/3 & \cdots & 1/n \\ \vdots & \vdots & & \vdots \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix}$$

の各要素を精度 p(bit) で生成し，ランダムに非対角要素を 0 に置き換えて疎性 s %(100%で零行列，0%で密行列を意味する) の疎行列に変換した $A^{(s)}$ を使用している．実ベクトルは

$$\mathbf{b} = A^{(s,p)}[1\ 2\ \cdots n]^T$$

とし，$A^{(s,p)}\mathbf{b}$ の計算時間 (秒) を計測した．比較のため，疎行列 $A^{(s)}$ を密行列形式で格納したものを $A^{(s)}_{Dense}$ とし，その計算時間 (sec) との比 (密行列/疎行列) を速度向上率 (Speedup Ratio) とする．この結果を**図 3** に示す．比較のため，倍精度の結果も載せてある．

倍精度疎行列の実装の効果が現れるのは 2000 次元の時で，ほぼ零要素の比率だけ計算時間は削減されている．それに対して，多倍長疎行列の計算時間削減効果は低く，約 1.3 倍〜3.2 倍程度に留まっている．MPFR/GMP の演算省力化機構が働いているため，零要素との積は非常に短い時間で計算できることがこの低性能向上比の原因であろう．実際，精度が大きくなると性能向上比は低くなることから，これが裏付けられる (**図 4**)．

以上の結果から，s と p が大きい時程計算時間は短縮され，大規模行列で比較的精度が低い場合は特に効果が大きいことが判明した．

4. Krylov 部分空間法への適用

Krylov 部分空間法が演算精度に敏感なことは古くから知られており，多倍長化してその影響を減らすことができることも報告されている．ここでは多倍長疎行列・ベクトル積を用いた Krylov 部分空間法の性能評価と多倍長化によって得られる効果について述べる．

今回は BiCG 法と GPBiCG 法 (**図 5**) に適用した．それぞれ下線部の行列/ベクトル積の部分を疎行列対応のルーチンに書き換えたことになる．

行列は全て UF Sparse Matrix Collection[5] の MTX ファイルをそのまま使用した．従って，倍精度の行列データを読み込んで多倍長化し，真の解を $\mathbf{x} = [1\ 2\ \cdots\ n]^T$ とし，定数ベクトルを $\mathbf{b} = A\mathbf{x}$ として多倍長計算して問題を作成した．収束判定条件 (図 5 における (*)) は

$$||\mathbf{r}_k||_2 \leq 10^{-20}||\mathbf{r}_0||_2 + 10^{-50}$$

とした．

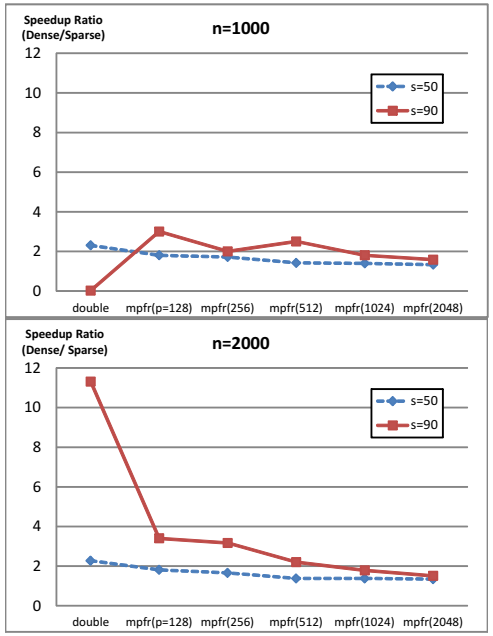

図 3　疎行列・ベクトル積の速度向上比 (vs. 密行列・ベクトル積 (MV))
Fig. 3　Speedup Ratio of SpMV (vs. Dense MV).

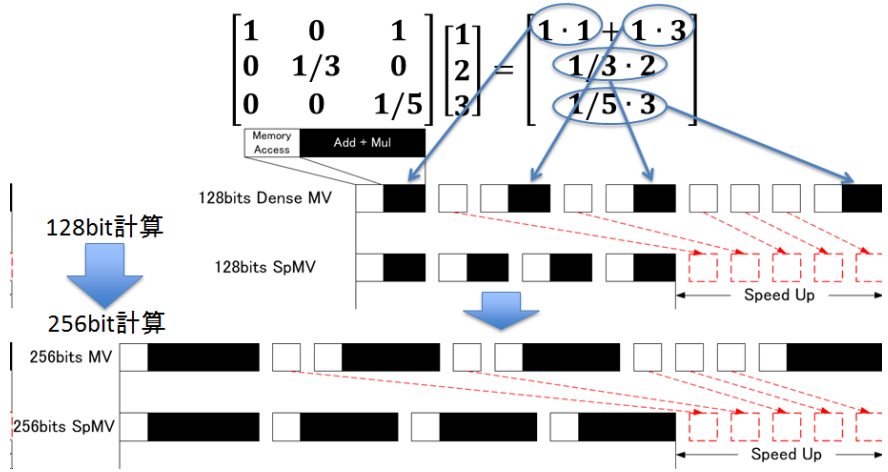

図 4　SpMV 使用による速度向上
Fig. 4　Explanation of Speedup by using SpMV.

4.1　DRICAV/cavity04

まず比較的小さい問題例として DRIVCAV/cavity04($n = 317$) を取り上げる．非零成分数は 7327(全要素数の約 7.3%) である．この程度の規模の問題であれば，密行列として扱ってもメモリ内に収まるため，性能向上比の確認に使用できる．要素の精度を 512bit(10 進約 154 桁) から 8192bit(同 2466 桁) まで増やしながら，既存の MV 計算を用いた計算時間と比較し，その性能向上比を調べた．

この行列の構造と，BiCG, GPBiCG 法の性能向上比を**図 6** に示す．

この問題の場合，2048bit 以上の精度にしないと次元数以内の反復回数で収束することはなかった．精度を増やすと収束までの反復回数が減ることから，多倍長化のメリットは見られる．

4.2　Averous/epb3

より大規模な例として，Averous/epb3($n = 84617$) を係数行列とした問題を使って数値実験を行った (**図 7**)．真の解 $\mathbf{x}$ と定数ベクトル $\mathbf{b}$ は同じ方法で作成した．

非零要素数は 463625 (0.0065%) となる．もしこれを密行列として読み込むとメモリ容量は，要素の精度が 128bits の場合は約 320GB, 8192bits にすると約 70TB になる．今回実

$\mathbf{x}_0$: Initial guess
$\mathbf{r}_0$: Initial residual ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}_0}$: $= \mathbf{r}_0$.
K: $= I$
for $i = 1, 2, ...$
  Solve $K\mathbf{w}_{i-1} = \mathbf{r}_{i-1}$ on $\mathbf{w}_{i-1}$.
  Solve $K^T\ \widetilde{\mathbf{w}_{i-1}} = \widetilde{\mathbf{r}_{i-1}}$ on $\widetilde{\mathbf{w}_{i-1}}$.
  $\rho_{i-1} = (\widetilde{\mathbf{w}_{i-1}}, \mathbf{w}_{i-1})$
  if $\rho_{i-1} = 0$ **then** exits.
  if $i = 1$ **then**
    $\mathbf{p}_1 = \mathbf{w}_0$
    $\widetilde{\mathbf{p}_1} = \widetilde{\mathbf{w}_0}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $\mathbf{p}_i = \mathbf{w}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1}$
    $\widetilde{\mathbf{p}_i} = \widetilde{\mathbf{w}_i} + \beta_{i-1}\widetilde{\mathbf{p}_{i-1}}$
  end if
  $\underline{\mathbf{z}_i = A\mathbf{p}_i}$
  $\underline{\widetilde{\mathbf{z}_i} = A\widetilde{\mathbf{p}_i}}$
  $\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{p}_i}, \mathbf{z}_i)$
  $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i$
  $\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i\mathbf{z}_i$
  $\widetilde{\mathbf{r}_i} = \widetilde{\mathbf{r}_{i-1}} - \alpha_i\widetilde{\mathbf{z}_i}$
  Convergence chech(*).
end for

$\mathbf{u} = \mathbf{z} = 0$
for $i = 1, 2, ...$
  $\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$
  if $\rho_{i-1} = 0$ **then** exits.
  if $i = 1$ **then**
    $\mathbf{p} = \mathbf{r}_0$
    $\underline{\mathbf{q} = A\mathbf{p}}$
    $\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
    $\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$
    $\underline{\mathbf{v} = A\mathbf{t}}$
    $\mathbf{y} = \alpha_i\mathbf{q} - \mathbf{r}_{i-1}$
    $\mu_2 = (\mathbf{v}, \mathbf{t})$, $\mu_5 = (\mathbf{v}, \mathbf{v})$
    $\zeta = \mu_2/\mu_5$
    $\eta = 0$
  else
    $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\zeta)$
    $\mathbf{w} = \mathbf{v} + \beta_{i-1}\mathbf{q}$
    $\mathbf{p} = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p} - \mathbf{u})$
    $\underline{\mathbf{q} = A\mathbf{p}}$
    $\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
    $\mathbf{s} = \mathbf{t} - \mathbf{r}_{i-1}$
    $\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$
    $\underline{\mathbf{v} = A\mathbf{t}}$
    $\mathbf{y} = \mathbf{s} - \alpha_i(\mathbf{w} - \mathbf{q})$
    $\mu_1 = (\mathbf{y}, \mathbf{y})$
    $\mu_2 = (\mathbf{v}, \mathbf{t})$
    $\mu_3 = (\mathbf{y}, \mathbf{t})$
    $\mu_4 = (\mathbf{v}, \mathbf{y})$
    $\mu_5 = (\mathbf{v}, \mathbf{v})$
    $\tau = \mu_5\mu_1 - \overline{\mu_4}\mu_4$
    $\zeta = (\mu_1\mu_2 - \mu_3\mu_4)/\tau$
    $\eta = (\mu_5\mu_3 - \overline{\mu_4}\mu_2)/\tau$
  end if
  $\mathbf{u} = \zeta\mathbf{q} + \eta(\mathbf{s} + \beta_{i-1}\mathbf{u})$
  $\mathbf{z} = \zeta\mathbf{r}_{i-1} + \zeta\mathbf{z} - \alpha_i\mathbf{u}$
  $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p} + \mathbf{z}$
  Convergence check(*)
  $\mathbf{r}_i = \mathbf{t} - \eta\mathbf{y} - \zeta\mathbf{u}$
  exits if $\zeta = 0$.
end for

図 5　BiCG 法 (左) と GPBiCG 法 (右) のアルゴリズムと SpMV 適用箇所 (下線部)
Fig. 5　Algorithms of BiCG(left) and GPBiCG(right). Underlined parts mean the usage of SpMV.

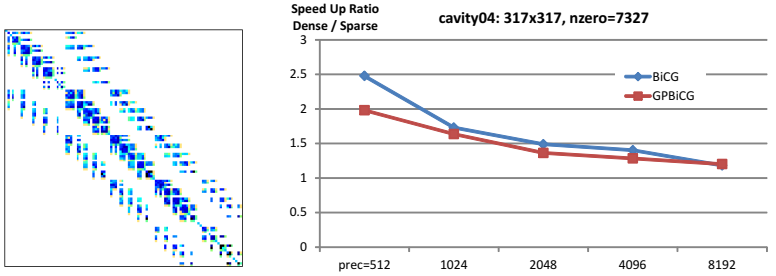

図 6　cavity04 の行列形式 (左) と速度向上比 (右)
Fig. 6　Matrix Structure of cavity04(left) and Speedup Ratio(right).

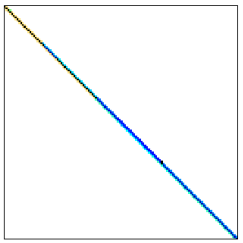

図 7　epb3 の行列形式
Fig. 7　Matrix Structure of epb3.

装した疎行列形式にすると 128bits で約 3.5MB, 8192bits にしても約 456MB で済むため，普通の PC 単体でも十分実行可能なメモリ量となる．

この問題における反復回数と実行時間の表を**表 1** に示す．×印は収束しなかったことを示している．

精度が増すにつれて反復回数が減っていることから，多倍長計算の効果が得られていることは確認できる．

Vol.2012-HPC-133 No.25
2012/3/27

表 1　epb3 問題の反復回数 (計算時間)
Table 1　Iterative Times and Computational Time (in parenthesis) of epb3 problem.

	512bits	1024	2048	4096	8192
BiCG	×	×	8351(4335 sec)	5251(7824 sec)	3567(15551 sec)
GPBiCG	×	×	×	7039(12473 sec)	4449(24179 sec)

5. 結論と今後の課題

事前にあまり期待していなかった多倍長疎行列の実装であるが，記憶領域の削減効果に加えて，ある程度の計算時間削減効果があることも判明した．特に比較的精度が低く，大規模な場合は効果が大きくなることも分かった．

今後の課題としては，より大規模な科学技術シミュレーションにこの疎行列の機能を適用することが挙げられる．

参 考 文 献

1) Swox AB. The GNU Multiple Precision arithmetic library. http://gmplib.org/.
2) Tomonori Kouya. BNCpack. http://na-inet.jp/na/bnc/.
3) LAPACK. http://www.netlib.org/lapack/.
4) Intel MathKernel Library. http://www.intel.com/software/products/mkl/.
5) The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/.
6) MPFR Project. The MPFR library. http://www.mpfr.org/.
7) SuiteSparse. http://www.cise.ufl.edu/research/sparse/SuiteSparse/.
8) 中田真秀. Multiple precision arithmetic blas and lapack. http://mplapack.sourceforge.net/.