GMP を用いた混合精度型プログラムの自動生成機構の提案

菱沼 利彰 [1], 藤井 昭宏 [2], 田中 輝雄 [2], 平澤 将一 [3]
[1] 筑波大学, [2] 工学院大学, [3] 東北大学
e-mail : hishinuma@slis.tsukuba.ac.jp

1 はじめに

大規模数値シミュレーションの核である Krylov 部分空間法は，丸め誤差に影響を受ける．高精度演算を用いれば収束を改善できる [1]．しかしながら，高精度演算はコストが高い．

GMP (GNU Multiple Precision Arithmetic Library)[2] は任意の精度で演算ができるライブラリである．GMP を用いたプログラムは，さまざまなコードの変更を必要とする．計算機の発展により，計算コストは問題にならなくなってきた．しかし，実装コストは未だに問題である．

我々はこの問題に対し，コードを構文木として扱い，変換ルーチンを作成できる Xevolver[3] フレームワークを用いて，C 言語の倍精度プログラム (C コード) を，GMP を用いた任意多倍長プログラム (GMP コード) へ自動変換する機構，Xev-GMP を実装している [4]．

一方で，計算時間やメモリデータ量を削減するため，複数の精度を使い分ける研究がある [5]．これに対応するため，複数の精度を使い分けるための機能を実装した．

本研究では，実際に共役勾配法のプログラムを対象に GMP を用いた混合精度コードを自動生成し，これを評価した．

2 GMP コードの自動生成

GMP では，任意多倍長型の浮動小数点変数に “mpf_t” という構造体を定義している．変数の使用前後に独自の構造体を初期化，解放する関数への受け渡しを行ったり，算術演算の式は演算子ではなく手続きの形式で行う必要がある．

Xev-GMP は，C コードにディレクティブとして精度の情報を追記することで，構文木を解析してコード内の倍精度変数をすべて GMP の任意多倍長変数にした GMP コードを生成する．

たとえば，“a = b + c * d” は，図 1 の左のような構文木として扱い，右のような構文木を生成し，以下のようなコードが得られる．

```
mpf_mul(tmp3, c, d);
mpf_add(tmp1, b, tmp3);
mpf_set(a, tmp1);
```

それぞれ順に，GMP の乗算，加算，代入関数である．

手続きの形式への変換において，一時変数の追加が必要になる．算術演算式を構文木として見た際のノード番号から，自動的に一時変数を生成した．

GMP では，混合精度演算を内部的にサポートしているため，変数宣言の際に必要な精度を指定するだけでよい．我々は，Xev-GMP に対して 2 つのディレクティブを実装した．

1) #pragma xev-gmp default(prec)
   コードの先頭に記述する．コード中のすべての倍精度変数やそれに関わる処理を，default() 句で指定した精度にする．
2) #pragma xev-gmp set(prec) target(var1, var2..)
   関数宣言の前に記述する．target() で指定した倍精度変数を，set() 句で指定した精度にする．その関数内では，生成する一時変数は関数内で最も高い精度になる．

これらのディレクティブは一般的なコンパイラには無視されるため，元となるコードもそのまま利用可能である．これにより，C コードの管理のみで，複数の精度のコードを扱うことが可能である．

3 数値実験

我々が作成した共役勾配法の C コードから，以下の 3 つのコードを作成した．

(A) すべての変数を GMP の 64 bit
(B) すべての変数を GMP の 128 bit
(C) 内積に関わる変数のみ GMP の 128bit，その他は GMP の 64 bit

なお，GMP の 64bit は，仮数部が 64bit の “mpf_t” 構造体のため，厳密には倍精度とは異なる．

(A)，(B) は，ディレクティブ 1) をコードの先頭に 1 行追記，(C) は (B) にディレクティブ 2) を main 文と内積関数の宣言前に追記するだけで生成できる．

入力する疎行列データはフロリダ大学の疎行

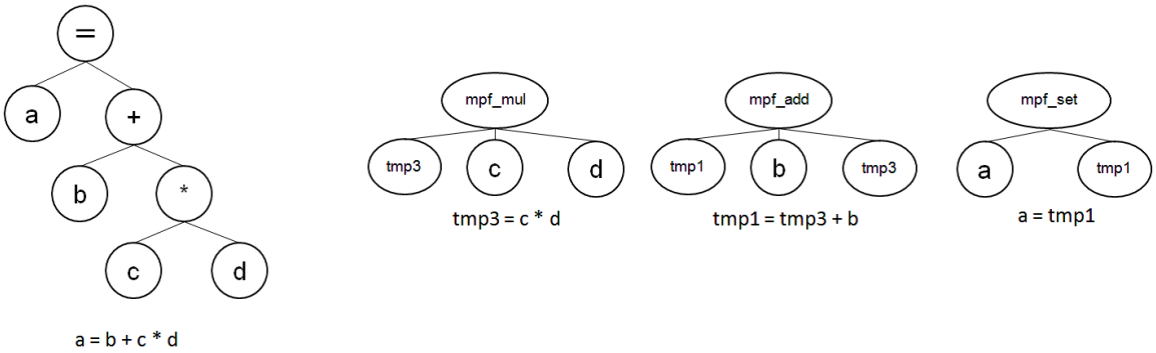

図 1. “a = b + c * d” の構文木 (左: C コード, 右: GMP コード).

列コレクション (`http://www.cise.uhl.edu/research/sparse/matrices/`) から取得した，行数 が 169,422, 非零要素数が 1,279,274 の “c-73b” を用いた，

実験には，CPU は Core i7 4770, OS は CentOS 6.2，コンパイラは gcc 4.6.3，GMP は ver. 6.0.0 を用いた．収束条件は 10^{-12} とし，x_0 と b_0 は倍精度乱数とした．

各コードにおいて以下の結果が得られた．

(A) 反復回数は 21543 回，実行時間は 0.72 [sec]，メモリデータ量は 60.1 [MB].

(B) 反復回数は 9132 回，実行時間は 0.41 [sec]，メモリデータ量は 79.3 [MB].

(C) 反復回数は 12312 回，実行時間は 0.50 [sec]，メモリデータ量は 66.3 [MB].

(B), (C) は (A) の結果と比べて早い．(A) に対し，(B) を用いることで反復回数は 42%に削減，一反復あたりの計算時間は 1.34 倍に増加．(C) を用いることで反復回数は 57%に削減，一反復あたりの計算時間は 1.21 倍に増加した．

(C) は，(B) と比べて一反復あたりの計算時間はあまり変わらないが，必要なメモリデータ量を (B) の 83%に削減できた．

計算時間に変化がないのは，GMP は精度によって内部の乗算アルゴリズムを切り替えており，64 bit と 128 bit はアルゴリズムが同じであるからだと考えられる．

4 結論

C コードから GMP の混合精度コードを自動生成する機構を提案した．

混合精度ディレクティブは実行時間の削減にはあまり効果がないが，必要なメモリデータ量を約 15%以上削減した．

内積を高精度にした混合精度の CG 法のコードの生成には，コード全体で 3 行の記述でよく，変換に必要な実装コストは小さい．

高精度化や混合精度化の効果は問題や解法で大きく異なるが，我々の提案を使えば，アルゴリズムの精度依存性を簡単に確認できる．

今後の課題として，GMP の乗算アルゴリズムを基に，精度の使い分けについて議論する必要がある．また，より高度な解析を行い，一時変数の精度を式ごとに変更する必要がある．

謝辞 本研究の一部は，JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」の支援により行った．また，本研究の一部は JSPS 科学研究費 25330144 の助成を受けた．

参考文献

[1] Tomonori Kouya, A Highly Efficient Implementation of Multiple Precision Sparse Matrix-Vector Multiplication and Its Application to Product-type Krylov Subspace Methods, IJNMA, Vol. 7, Issue 2, pp. 107-119, 2012.

[2] The GNU Multiple Precision Arithmetic Library, `https://gmplib.org/`.

[3] Xevolver, `http://xev.arch.is.tohoku.ac.jp/ja/software/#xevxml`.

[4] 榊原 巧磨, 佐々木 信一, 菱沼 利彰, 藤井 昭宏, 田中 輝雄, 平澤 将一. GMP ライブラリを用いた任意多倍長プログラムへの自動変換機構の提案, 情報処理学会研究報告, HPC-152, No.6, pp.1-8, 2015.

[5] Tsubasa Saito, et. al., Analysis of the GCR method with mixed precision arithmetic using QuPAT, Journal of Computational Science, Volume 3, Issue 3, pp. 87-91, 2012.