Xevolver を用いた GMP コードへの自動変換機能の実装

丸地賢† 佐々木信一† 菱沼利彰† 藤井昭宏† 田中輝雄† 平澤将一‡
工学院大学† 東北大学‡

1 はじめに

任意多倍長ライブラリの 1 つに，GMP (GNU Multi-Precision)ライブラリがある[1]．GMP ライブラリを用いた任意多倍長精度のコード（GMP コード）は変数の宣言，初期化，演算ルーチンを GMP 特有の書き方で表す必要があるため，実装コストが高いという問題点がある．

一方，東北大学の滝沢らが開発した Xevolver[2]は，コードを編集する代わりにディレクティブを追記して宣言することで変換されたコードが出力されるフレームワークである．本研究では，Xevolver 上で，倍精度のコードを GMP コードに自動変換する機能を実装した．

2 Xevolver

Xevolver はコードを編集する代わりにディレクティブを追記して宣言することで，変換されたコードを出力する．ディレクティブとはコンパイラへの指示である．図 1 に Xevolver の処理フロー(1)~(3)を示す．

(1) もとの C コードを ROSE コンパイラ[3]で構文解析し構文木(XML)に変換する．
(2) ディレクティブに合う変換ルーチン(XSLT)を使い，構文木を書き換える．
(3) 書き換えた構文木を C コードに変換する．

3 Xevolver を用いた GMP コードへの変換

3.1 ディレクティブ

本研究では Xevolver に対し，ユーザがデフォルトの精度を任意に指定するディレクティブと，指定した変数に任意の精度を反映させるディレクティブを用意することで，倍精度コードを GMP コードに自動変換できる機能を実装した．

3.2 GMP コードへの変換手法

本研究で作成した変換ルーチンで構文木が変換される具体例を図 2 に示す．まず，ROSE コンパイラを用いて倍精度コード(sum=a+b;)を構文解析し，和算の構文木に変換する．次に，和算における変換ルーチンを使い，和算の構文木を GMP 和算関数の構文木に書き換える．最後に，ROSE コンパイラを用いて構文木を GMP コード(mpf_add(sum,a,b);)に変換する．

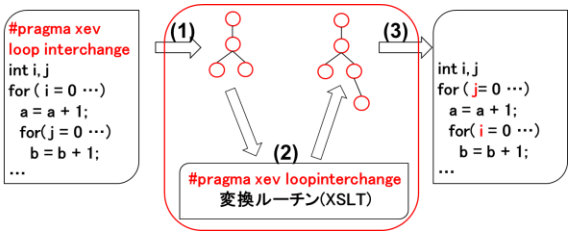

図 1　Xevolver の処理フロー

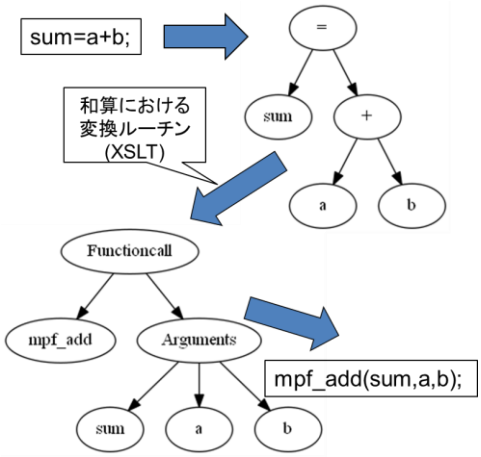

図 2　和算における GMP コードへの変換手法

3.3 実験

ここでは，ノルム演算の倍精度コード(図 3)を GMP コード(図 4)に自動変換する実験結果を示す．

このとき，ユーザが C コードに追加する必要があるのは，以下の 2 点のみである．

(1) #pragma xev gmp default(128)（図 3，#1）
関数外に追記することで，倍精度変数を 128bitGMP 型に変更する．
(2) #pragma xev gmp set(512)（図 3，#6，#8）
#pragma xev gmp set(256)（図 3，#18，#20）
倍精度変数の宣言を囲むことで，指定した精度の GMP 型変数に変換する．

Implementation of automatic conversion system to GMP code on Xevolver
Ken Maruchi†, Shin'ichi Sasaki†, Toshiaki Hishinuma†, Akihiro Fujii†, Teruo Tanaka†, Shoichi Hirasawa‡
†Kogakuin University
‡Tohoku University

```
1.  #pragma xev gmp default(128)
2.  #include<stdio.h>
3.  #include<math.h>
4.  void nrm(int n,double *a,double *nrm){
5.    int i;
6.  #pragma xev gmp set(512)
7.    double temp=0.0;
8.  #pragma xev gmp set(512)
9.    for(i=0;i<n;i++){
10.     temp=a[i]*a[i];
11.     *nrm=*nrm+temp;
12.     temp=0.0;
13.   }*nrm=sqrt(*nrm);
14. }
15. int main(){
16.   int i,N=50;
17.   double *X;
18. #pragma xev gmp set(256)
19.   double X_nrm=0.0;
20. #pragma xev gmp set(256)
21.   X = (double *)malloc(sizeof(double) * N);
22.   for(i=0;i<N;i++)
23.     X[i]=1.0;
24.   nrm(N,X,&X_nrm);
25.   printf("X_nrm=%lf¥n",X_nrm);
26.   free(X);
27.   return 0;
28. }
```

図 3　ノルム演算の倍精度コード（変換前）

2 種類のディレクティブを追記することで，以下のようなコードの自動変換が行われる．

(1) 倍精度四則演算を GMP コードへ変換
　四則演算関数は第 2，3 引数に整数，浮動小数点数を受け取る場合， mpf_scrptr 型（図 4，#11，#12）にキャストする必要がある．

(2) malloc 関数の GMP コードへの変換
　GMP で用いるデータ型に対しては malloc 関数が使用できないため，calloc 関数（図 3, #21，図 4，#26）に変換する．さらに，精度の初期化のために精度初期化用の for 文（図 4，#27，#28）が生成される．

(3) メモリの解放
　double 型変数から mpf_t 型変数に変換されると，自動的に関数の最後または return 文の直前に mpf_clear 文（図 4，#16，#33，#34）が追加される．

3.4 ユーザへの制約

(1) mpt_t 型を戻り値とする自作関数が作れないため，double 型を戻り値とする自作関数は使用できない．

(2) １つの式に演算子は１つしか使用できない．

(3) 倍精度の配列は１次元までとし，常に動的に割り当てられていること．

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <math.h>
4.  #include <gmp.h>
5.  void nrm(int n,mpf_t *a,mpf_t *nrm){
6.    int i;
7.    mpf_t temp;
8.    mpf_init2(temp,512);
9.    mpf_set_d(temp,0.0);
10.   for (i = 0; i < n; i++) {
11.     mpf_mul(temp, (mpf_srcptr )a[i], (mpf_srcptr )a[i]);
12.     mpf_add( *nrm, (mpf_srcptr )*nrm, (mpf_srcptr )temp);
13.     mpf_set_d(temp,0.0);
14.   }
15.   mpf_sqrt( *nrm, *nrm);
16.   mpf_clear(temp);
17. }
18. int main(){
19.   mpf_set_default_prec(((mp_bitcnt_t )128));
20.   int i_gmp
21.   int i,N = 50;
22.   mpf_t *X;
23.   mpf_t X_nrm;
24.   mpf_init2(X_nrm,256);
25.   mpf_set_d(X_nrm,0.0);
26.   X = ((mpf_t *)(calloc(((size_t )(sizeof(mpf_t ))),((size_t )N))));
27.   for (gmp_i = 0; gmp_i < N; gmp_i++)
28.     mpf_init(X[gmp_i]);
29.   for (i = 0; i < N; i++)
30.     mpf_set_d(X[i],1.0);
31.   nrm(N,X,&X_nrm);
32.   gmp_printf("X_nrm=%Fe¥n",X_nrm);
33.   mpf_clear( *X);
34.   mpf_clear(X_nrm);
35.   return 0;
36. }
```

図 4　ノルム演算の GMP コード（変換後）

4 まとめと今後の課題

本研究では，ユーザが倍精度コードに 2 種類のディレクティブを追記することで，GMP コードに自動変換する機能を Xevolver 上に実装した．これにより，GMP コードの実装コストが高いという問題点が解決できた．

今後の課題としては，1 つの式で複数の演算子を扱えるようにすることが挙げられる．

参考文献

[1] The GNU MP Bignum Library，https://gmplib.org/.

[2] Hiroyuki Takizawa，Shoichi Hirasawa，Yasuharu Hayashi，Ryusuke Egawa，Hiroaki Kobayashi，“Xevolver: An XML-based Code Translation Framework for Supporting HPC Application Migration”，IEEE International Conference on High Performance Computing (HiPC)，2014.

[3] ROSE compiler infrastructure，http://rosecompiler.org/.