JIS Full BASIC 実時間機能単位の実装に向けての課題

白石　和夫*

Issues with Implementation of the Real-time Module of JIS Full BASIC

Kazuo SHIRAISHI

要旨　LazarusプロジェクトのObject Pascal言語に翻訳することで，日本工業規格Full BASICの実時間機能単位の実装を図った．マルチコアCPUの下での計算処理の高速化の観点で必要とされる機能はすべて実装した．しかし，プロセス制御を目的とする命令は未実装であり，特に，この手法ではプロセス制御で必要とされる機能の一部は実装の困難が予想される．

キーワード：プログラミング言語　Full BASIC　実時間機能単位　マルチスレッド

1．はじめに（研究のねらいなど）

Full BASIC[1]は，初学者向きのプログラミング言語であるが，数値計算目的であれば十分な実用性がある．整数型を持たないのは高速実行に不向きであるが，最近のPCは浮動小数点演算が高速化しており，それは大きな弱点ではない．

Full BASIC 図形機能単位＋モジュール機能単位で書かれたプログラムは，そのほとんどがLazarusプロジェクト[2]によるObject Pascalプログラムに翻訳可能であり，機械語レベルへの翻訳による高速実行が実現している[3]．

しかし，近年，CPUの動作クロックの上昇が頭打ちし，かわりにマルチコアCPUが普遍化して，計算の高速化のためには，CPU全コアに負荷を分散する並行プログラムを書くことが必要な時代となってきた．

本研究では，前著[3]に従ってLazarusプロジェクトにおけるObject Pascalに書き換えて実行する手法を採用する．以降，「Object Pascal」は，Lazarusプロジェクトにおけるそれ（FPCと，Lazarusにおける追加）を意味する．

2．Full BASIC 実時間機能単位について

2.1．並行単位

Full BASICの実時間プログラムでは，主プログラムの代わりに複数の並行単位（PARACT ～ END PARACT）を書く．並行単位を並べた後に，外部手続きが書かれる．複数の並行単位から同じ外部手続きを呼び出すプログラムを書くことは禁止されていない．だから，外部手続きは，スレッドセーフである必要がある．

プログラムが実行を始めると，一番上に書かれた並行単位が実行を始める．他の並行単位は並行単位名を指定するSTART文で実行を開始する．

各並行単位は，実行を終えると停止状態になる．停止状態にある並行単位は，START文で再度，実行を開始させることができる．これは，並行単位の終了時に終了を知らせるためのシグナルを発することにすれば，並行単位をサブルーチンのように使えることを意味する．

* しらいし　かずお　文教大学教育学部学校教育課程数学専修

PARSTOP文を実行すると，当該並行単位が実行を終える．PARSTOP文を外部手続きに書くことは禁止されていない．

2.2. 変数の共有と同期

並行単位はプログラム単位の一つであるから，変数を共有しない．Full BASICの外部手続き単位は静的変数を持たないから，実時間プログラムには複数の並行単位からアクセス可能な変数が存在しないことになる．この事実は，マルチスレッドプログラミングで問題となる同一変数への同時書き込みが起こらないことを意味する．

代わりに，スレッド間でデータをやり取りするための手段が，複数，用意されている．

一つは共用域を用いるもので，共用域へは，書き込みと読み出しの命令が用意されている（共用域を直接にアクセスすることはできない）．共用域入出力は排他制御され，同時に複数の並行単位で実行されることはない．

共用域を介してデータをやり取りするとき，SIGNAL文とWAIT EVENT文を用いて並行単位間の同期を取ることができる．

2.3. 通報域

通報域自体は共用域とほぼ同じものであるが，データが空であるという状態を持つ．空な通報域からデータを読み取ろうとすると，データが書き込まれるまで待たされる．また，通報域が空でないとき，データ書き込みは通報域が空になるまで待たされ，データを書き込むとデータが読み取られるのを待つ．そして，データ読み取りが終わると，通報域は空にリセットされる．この機構を用いて並行単位間の同期を取ることができる．

2.4. デッドロックの可能性

Full BASIC規格では変数の同時アクセスの可能性を文法的に回避しているけれども，マルチスレッドプログラミングで問題になるもう一つの重大事であるデッドロックについては考慮されていない．SIGNAL文－WAIT EVENT文を用いる場合と通報域を用いる場合のいずれでもデッドロックを起こす可能性がある．

たとえば，通報域を用いる次のコードは，互いに相手方のRECEIVE文の実行終了を待ってSEND文の実行が完了しない（200行と210行を入れ替えれば正しく動作する）．

```
100 DECLARE STRUCTURE struct1:3 OF NUMERIC
110 DECLARE MESSAGE mes1 OF struct1
120 DECLARE MESSAGE mes2 OF struct1
130 PARACT p1
140 START p2
150 SEND TO mes1 FROM 1,2,3
160 RECEIVE FROM mes2 TO a,b,c
170 PRINT a,b,c
180 END PARACT
190 PARACT p2
200 SEND TO mes2 FROM 4,5,6
210 RECEIVE FROM mes1 TO a,b,c
220 PRINT a,b,c
230 END PARACT
```

2.5. 排他制御

あるピクセルに色を塗りたいとき，Full BASICでは，たとえば，

```
SET POINT COLOR n
PLOT POINTS: x,y
```

のように，色を指定する命令と座標を指定して実際に色を塗る命令とを続けて実行する．複数の並行単位でこれらの命令を実行するとき，PLOT POINTSの実行の前に別の並行単位で別の色を指定するSET POINT COLOR文を実行している可能性があるから，排他制御が必要になる．

Full BASICには，排他制御を行うための機構としてSEIZE区（SEIZE ～ END SEIZE）が用意されている．SEIZE文は指定された資源を確保するまで待ち，END SEIZE文は資源を解放する．

2.6. 実時間機能単位と図形機能単位

前項でグラフィックスの例を挙げたけれども，Full BASICには図形機能単位（または簡易図形出力機能単位）と実時間機能単位の双方を包含する機能単位の組が定義されていない．だから，厳

密には，図形描画命令を含む実時間プログラムはどちらの機能単位に対しても規格合致ではない．

2.7. モジュール

モジュール機能は，日本工業規格の付属書に参考として収録されている．モジュール機能単位では，主プログラムに外部手続きまたはモジュールを続けて書いたものがプログラムの定義に追加される．並行単位を並べたものは主プログラムではないから，規格上，モジュールと実時間機能は両立しない．これは，この章のはじめに述べた共有変数を持たないことによる安全性を担保することにつながる．

3．実時間機能単位の実装

3.1. 入出力の別スレッド化

3.1.1. 画面入出力

Lazarusのプログラムでは，画面表示などGUIを扱うスレッドはメインスレッドでなければならない．そのため，Full BASICの各並行単位をメインスレッドと異なるスレッドに割り当てる．だから，翻訳されたプログラムは，並行単位数＋１個のスレッドを持つ．

PRINT文やPLOT文を実行するときにはスレッドからの同期実行に頼るのが通常の手法と思われる．しかし，この方法は画面表示が終わるまで計算の進行を待たせることを意味するから，実行速度の向上につながらない．そこで，キューを用いてデータを送信する手法を採用した．この手法は，現時点で同期実行が正常に機能しないMAC版Lazarusでも有効という利点も持つ．

3.1.2. テキスト入出力

PRINT文は，出力すべきテキストをキューに登録し，画面表示を担うスレッドがキューから取り出してそれを行う．

INPUT文による入力をメインスレッドに行わせるために，INPUT文を実行すると入力要求のフラッグを立て，INPUT文はメインスレッドが入力処理を終えてフラッグを降ろすのを待つ．

3.1.3. 描画出力

テキスト出力の場合と異なり，描画内容は多岐にわたるから，座標データなどとともに描画コマンドを渡さなくてはいけない．Object Pascalのクラスを一つ基底クラスとして宣言し，コマンドの種類ごとに継承クラスを定義し，そのオブジェクトをキューに登録する．

描画命令には，SET POINT COLORなどのSET文がある．他の描画命令と実行順が入れ替わると困るから，これらの命令も上述の描画コマンドに準じてキューに渡す．さらに，SET文には，対応するASK文がある．ASK文とSET文も順序を変えることができないから，それらもまた，同一のキューに登録する．ASK文を実行したスレッドは，メインスレッドが処理を終えるまで待つ．

Full BASICの図形機能には行列を用いた図形変形の機能がある．変形行列はプログラム全体で一つである．変形行列に対する操作も描画コマンドとして実行順を保存しなければならないから，上述のキューに併合される．これは，変形行列の計算のみで済む処理にとっては過度なオーバーヘッドであり，計算速度が低下する．

3.2. 並行単位

Full BASICの並行単位は，Object Pascalのスレッドに割り当てられる．これは，OSレベルのスレッドでもある．Object PascalのTThread型を継承するクラスTMyThreadを定義し，TMyThreadにFull BASICの並行単位をprocedureに翻訳したものを登録し，スレッドが実行を始めたときに実行されるようにする．

Object PascalのTThreadとFull BASICの並行単位には厄介な相違がある．Full BASICの並行単位は一度終了した後，再度実行することができる．一方で，明示的な記述はないものの，Object PascalのTThreadは一度きりの実行しか想定されていない．そのため，START文でTMyThreadインスタンスの生成から実行まで行う．スレッドの実行が終わるとTMyThreadのインスタンスを消去する．これによって並行単位は何度でも実行

できるようになるものの，TThreadインスタンスの生成と消去には時間がかかるから，短時間で済む処理を並行単位に割り当てサブルーチンとして頻繁に呼び出すとかえって遅くなる．

3.3. 共用域と通報域

共用域と通報域を，共通の基底クラスから派生するクラスのインスタンスとしている．STRUCTURE宣言は派生クラスの定義に翻訳され，SHARED宣言（共用域の宣言）とMESSAGE宣言（通報域の宣言）はそのインスタンスに翻訳される．そして，通報域が空であるかどうかの判断をフラグで行っている．

3.4. 実時間＋中核

Full BASICの実時間機能単位は，中核機能単位に実時間機能を追加したものである．PRINT文などのコンソール入出力は中核で定義されている．０番の経路はプログラム全体で広域的とされているから，通常のPRINT文による出力は単一のコンソールに出力される．複数の並行単位からPRINT文を同時に実行したときの挙動について規格は何も規定していないが，たとえば，PRINT文に複数項目を書いたときには，その途中で他の並行単位のPRINT文の出力が割り込むことがないなど，一文の動作の途中で他のスレッドが割り込むことがないようにしている．別のいい方をすると，末尾にコンマまたはセミコロンを書いたPRINT文を実行したときには，それに続けて実行されるPRINT文が他の並行単位のものであるかもしれない．

3.5. STOP文

PARSTOPは，実時間でないプログラムにおけるSTOP文と同様にして実現できる．

しかし，実時間プログラムにおけるSTOPの実装は容易ではない．プログラム（アプリケーション）それ自体を強制終了してしまったのでは，実行結果を見るという計算目的のプログラムとしての使い勝手を損なう．また，PARSTOPも同様であるけれど，STOP文を実行したとき開いている経路がすべて閉じられなければならない．それゆえ，実時間プログラムにおけるSTOP文の実装はむずかしい．

現バージョンでは，およそ，次のようにしてSTOP文を実装している．STOP文を実行するとそれを検知したBASICの並行単位の上位にある隠されたメインスレッドにおいて，まず，全並行単位のsuspendを実行し，次いで，全並行単位の経路を閉じ，最後にKillThreadを実行する．

3.6. SEIZE区

画面へのテキスト出力やグラフィックの実行に排他制御は欠かせない．テキスト出力画面や図形出力画面を資源として管理できるようにしてもよいのだけれども，より柔軟に使えるように，処理系定義の確保項目として利用者定義の任意の名称が書けるようにしている．すなわち，同じ名前を持つSEIZE区はプログラム全体を通して，高々，一つしか実行できないものとしてSEIZE区を実装している．SEIZE区は，クリティカルセクションに翻訳される．

Object Pascalのクリティカルセクションに複数のミューテックスを持たせることはできないから，SEIZE区に複数の確保項目を書いたとき，それらを名前の順に整列して対応するクリティカルセクション開始命令を生成する．これは，Full BASIC規格の「利用可能でない資源が一つでもあると不成功になる」という規定に合わないけれども，確保項目を名前の順に整列し直しているのですくみ（デッドロック）を起こす心配はない．規格の「注意」に「資源全部が確保されるか又はどれも確保されないという意味規定は，すくみを防ぐ」とあり，すくみ回避の視点からすると，実用上の支障はないであろう．

Full BASICの文法は，SEIZE文にTIMEOUT句を書くことを許している．けれど，Object Pascalのクリティカルセクションに当該機能がないので，この機能は実装できていない．

3.7. 図形機能

先述のように，図形機能単位と実時間機能単位は規格上両立を要求されない．しかし，実用上は

欠かせないので，混ぜこぜのプログラムを許すような実装を試みている．

PRINT文の場合と同様に，一文の動作の途中で他の並行単位の出力が割り込むことがないようにしている．逆にいうと，たとえば，末尾にセミコロンを書いたPLOT LINES文を実行するとき，次に実行されるPLOT LINESがどの並行単位のものであるか，不確定である．

3.8. モジュール

モジュールを持つプログラムで主プログラムの代わりに並行単位並びが書けるようにコンパイラを構成することは難しくない．コンパイル時に警告を発したうえで，並行単位並びとモジュールの混在を許している．同一変数に複数の並行単位から同時に書き込むことを文法的に回避するFull BASIC規格の意図に反するけれども，注意深くプログラムを書けば高速なマルチスレッドプログラムが得られる．

4．成果

4.1. 実行速度

4.1.1. 測定例

Windows7 64ビット，Intel Core i5-2500, Intel HD Graphics環境下での実行速度の違いを示す．Lazarusのバージョンは1.6.0である．

比較プログラムは充填ジュリア集合を描画するものである．並行処理を用いないプログラム１（次ページ，行番号は省略した）の実行に要する時間は23.98秒（複数回計測した中央値）であった．プログラム２は２個の並行単位に分割して描画するようにしたものである．所要時間は12.20秒（中央値）となり，２倍近くに高速化されている．

4.1.2. 考察

充填ジュリア集合の描画は並行単位間の同期が不要なため，２個の並行単位に分けるとほぼ２倍に高速化する．並行単位間の同期が必要な場合は，上述のように単純には高速化しない．また，計算時間に比して描画量の多いプログラムだと並行単位に分割してもさほど速くならない．目的とするプログラムに対し，どの程度の手間でどの程度の高速化が実現するかは，今後の研究課題である．

4.2. 未実装の機能

先述の他，PROCESS関連の諸命令とSELECT ON PORT区は未実装である．組み込むべき処理対象（process object）が存在しないからである．ただし，処理域およびそれに対する入出力の文は，共用域に対するものとほぼ同じなので，同じ基底クラスからの派生で実現できるであろう．

4.3. まとめ

プログラム実行の高速化を目的としてFull BASIC実時間機能を利用するのであれば今回の実装手法で十分な成果が得られ，Windows, Linux，Macで共通して利用できる．

一方，プロセス制御を目的とする場合など，SEIZE文のTIMEOUT句が必要とされる場合は，異なる手法の開発が必要となるであろう．

References

１）日本工業規格　電子計算機プログラム言語 Full BASIC，日本規格協会，1993

２）https://www.lazarus-ide.org/

３）白石和夫，Full BASICのObject Pascalへの埋め込み，文教大学教育学部紀要第44集，pp.93-100, 2010

プログラム1

```
OPTION ARITHMETIC NATIVE
DECLARE EXTERNAL NUMERIC J.a, J.b
DECLARE EXTERNAL NUMERIC J.dx, J.h, J.dy
DECLARE EXTERNAL SUB J.Julia
LET a=-.748
LET b= .094
LET left = -1.6
LET right = 1.6
LET h = (right - left)
SET WINDOW left,right,-h/2,h/2
ASK PIXEL SIZE(left,-h/2;right,h/2) px,py
LET px=px-1
LET py=py-1
LET dx=(right-left)/px/8
LET dy=h/py/8
SET POINT STYLE 1
CALL Julia(left,right)
END
MODULE J
MODULE OPTION ARITHMETIC NATIVE
PUBLIC NUMERIC a,b,dx,h,dy
PUBLIC SUB Julia
EXTERNAL SUB Julia(x1,x2)
   FOR u= x1 TO x2 STEP dx
      FOR v = -h/2 to h/2 STEP dy
         LET xx = u
         LET yy = v
         FOR n = 1 TO 1000
            LET x = xx
            LET y = yy
            LET xx = x*x - y*y + a
            LET yy = 2 * x * y + b
            IF xx^2+yy^2>4 THEN EXIT FOR
         NEXT n
         IF n>1000 THEN PLOT POINTS: u,v
      NEXT v
   NEXT u
END SUB
END MODULE
```

プログラム2

```
DECLARE STRUCTURE struct5: 5 OF NUMERIC
DECLARE STRUCTURE struct2: 2 OF NUMERIC
DECLARE SHARED buff OF struct5
DECLARE SHARED buff2 OF struct2
PARACT part1
OPTION ARITHMETIC NATIVE
DECLARE EXTERNAL SUB Julia
LET a=-.748
LET b= .094
LET left = -1.6
LET right = 1.6
LET h = (right - left)
SET WINDOW left,right,-h/2,h/2
ASK PIXEL SIZE(left,-h/2;right,h/2) px,py
LET px=px-1
LET py=py-1
LET dx=(right-left)/px/8
LET dy=h/py/8
SET POINT STYLE 1
PUT TO buff2 FROM left,right
PUT TO buff FROM a,b,dx,h,dy
START part2
CALL Julia(left,(left+right)/2)
END PARACT
PARACT part2
OPTION ARITHMETIC NATIVE
DECLARE EXTERNAL SUB Julia
GET FROM buff2 TO left, right
CALL Julia((left+right)/2, right)
END PARACT
EXTERNAL SUB Julia(x1,x2)
OPTION ARITHMETIC NATIVE
GET FROM buff TO a,b,dx,h,dy
FOR u= x1 TO x2 step dx
   FOR v = -h/2 to h/2 step dy
      LET xx = u
      LET yy = v
      FOR n = 1 TO 1000
         LET x = xx
         LET y = yy
         LET xx = x*x - y*y + a
         LET yy = 2 * x * y + b
         IF xx^2+yy^2>4 THEN EXIT FOR
      NEXT n
      IF n>1000 THEN PLOT POINTS: u,v
   NEXT v
NEXT u
END SUB
```