組込みシステム設計の生産性を向上させる超高位設計言語

Algorithm-Level Description Language for Very High-Level Design Improving Embedded System Design Productivity

白井　智
■ SHIRAI Satoshi

東芝は，一つのアルゴリズム記述から，各種のマルチコアプロセッサ向けプログラムやハードウェアエンジンの動作記述など，様々なプラットフォーム向けに最適化したプログラムを自動合成することで，組込みシステムのソフトウェア・ハードウェア開発における設計の生産性を向上させることを目指した，超高位設計手法の開発を進めている。

超高位設計の入力となるアルゴリズムを記述するために新しく設計した言語には，ソフトウェアとハードウェア両方の開発を効率化するための工夫，特にマルチメディア処理に頻出するストリーム処理アルゴリズムの開発を効率化するための工夫が施されている。この言語は，特定のプラットフォームだけに効果があるメモリの使用方法や計算の並列化方法に依存しないように簡潔にアルゴリズムを記述できる性質と，最適化に必要な情報を簡単な解析で抽出できる特長を持つ。

With the aim of accelerating software and hardware design processes, Toshiba has been promoting the development of a very high-level design approach by optimally synthesizing various software and hardware codes automatically from a single algorithm description.

An algorithm description language has been newly designed for the very high-level design approach, featuring not only concise algorithm descriptions independent of the memory usage and parallelization method of the specific platform but also easy algorithm analysis. This allows the generation of C-level codes optimized for each platform.

1　まえがき

ハードウェアの設計では，対象が複雑になってコストが増大するのを抑えるために，過去に何度もパラダイムシフトが起こってきた。すべてに共通するのは，技術変革によって，抽象度の高い記述から低い記述への自動合成を可能にし，設計の複雑さを減らして設計の生産性を向上したことである。

最近では，動作記述からRTL（レジスタ転送レベル）記述の自動生成を行う高位設計技術が開発された。動作記述には多くの場合にC言語が利用されているが，C言語は抽象度が低い記述を完全には排除できていない。ここで抽象度の高い記述とは，プログラムの実行結果を求めるために必要な計算の集合のことを指し，アルゴリズムにおける本質的な部分である。また，抽象度の低い記述とは，プラットフォームに適した計算の実行順序と実行結果の記憶方法のことを指し，アルゴリズムの実装に当たる部分である。

そこで東芝は，動作記述よりも更に抽象度の高いアルゴリズムレベル記述から，ハードウェアに合成可能な動作記述や組込みソフトウェアを自動生成する設計手法である超高位設計のコンセプト（**図1**）を提唱している。また，アルゴリズムの本質を記述するための言語として，組込みシステムで頻出するストリーム処理の記述に特に適している“超高位設計言語”を設計した。超高位設計言語は記述が容易で，この言語で記述されたプログラムは最適化に必要な情報を簡単な解析で抽出することができる。

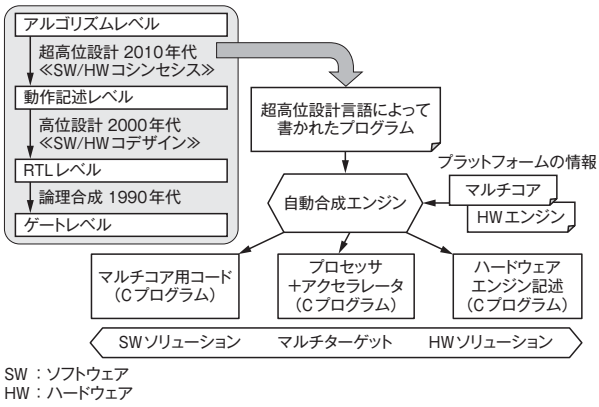

図1．超高位設計のコンセプト ― 超高位設計では，動作記述より抽象度が高いアルゴリズムレベル記述を行うことで設計生産性を高め，高位設計の次のパラダイムシフトを狙っている。

Concept of very high-level design

超高位設計における自動合成エンジンは，超高位設計言語によって書かれたプログラムと，プラットフォームの情報の二つを入力とする。これらを受け取った自動合成エンジンは，計算の並列化を含めた実行順序と，実行結果の記憶方法をプラットフォームに合わせて自動的に最適化し，対象プラットフォームで効率よく動作するCプログラムを生成する。

超高位設計では，アルゴリズムの本質だけを記述すればよ

いのでプラットフォームに精通していないプログラマーでも設計でき，精通しているプログラマーであればC言語よりも短時間で設計できる。更に，超高位設計言語で記述されたプログラムは，プラットフォームの情報を差し替えるだけで，異なるプラットフォーム向けのCプログラムを自動合成できる。

2　ストリーム処理におけるC言語設計の問題

組込みシステムでは，膨大なデータ系列の要素を順次読み出し，その要素を使って定型の計算を行い，計算結果を別のデータ系列の要素として書き込むストリーム処理が頻出する。動画や音声などのマルチメディア処理をはじめとして，通信処理や暗号処理はいくつものストリーム処理で構成される。ここでは，1章で概要を述べたC言語設計における問題点を，ストリーム処理の観点からより詳しく述べる。

2.1　記述方法

C言語にはストリームという概念が存在しないので，ストリーム手順を直接記述するための専用の構文がない。そのため，ストリーム処理をC言語で記述する場合には，通常は入力データ系列と出力データ系列の記憶方法を決め，入力データ系列に対する定型の計算を，汎用的な繰返し構文を用いて記述する必要がある。このため，表現したい内容を文法に合わせる手間が発生し，また本来最適化で決めるべき記憶方法を人間が決めなければならない。更に，複数のストリーム処理を多段に組み合わせるような場合には，中間データ系列の記憶方法やそれぞれの処理の同期も考慮しなければならない。また，ストリーム処理の流れの情報が繰返し構文の中に分散するので，最適化が難しく，可読性も落ちる。

2.2　順序の変更

C言語で記述したストリーム処理プログラムを自動的に最適化するには，実行順序の変更が必要になる。ところが，C言語で記述されたストリーム処理は実行順序が既に決まっているので，最適化によって実行順序を変更したプログラムの結果が，元のプログラムの結果と同じになることを保証する必要がある。

しかし，そのためには異なる実行順序を持つ二つのプログラムがあらゆる入力データに対して同じ結果を出すことを検証しなければならない。これには強力な解析能力と多くの解析時間が必要なので現実には難しく，最適化の障害となる。

2.3　処理間の依存関係の解析

依存関係とは，ある変数の値がどの計算式で計算され，どの計算式で利用されるか，という関係である。依存関係の解析によってストリーム処理間の関係も把握できる。どのストリーム処理の出力がどのストリーム処理の入力になるのか，どのストリーム処理とどのストリーム処理が独立しているのかといった関係を把握できると，複数のストリーム処理を含むプログラムを最適化できる。

ところが，C言語では，同じ変数名や同じ関数を同じ引数で呼び出しても値が常に等しくなるとは限らないので，ある変数の依存関係を確認するにはプログラムの広い範囲を解析する必要がある。このためC言語では，ストリーム処理間の関係を完全に解析するのは実質的に不可能になる。

3　超高位設計言語の設計方針

当社は，2章で述べたC言語設計の問題点を解決するため，超高位設計言語に，ストリーム処理に適したループ構文と，関数型言語の性質である遅延評価，参照透明性，副作用のない関数呼出しなどを導入した。これらは，アルゴリズムを簡潔に記述でき最適化するための情報を取得しやすくする効果を持つ。ここではそれらの特長を述べる。

3.1　ストリーム処理に適したループ構文

2.1節で述べたストリーム処理記述方法の問題を解決するために，簡潔で各繰返しサイクルの依存関係を取得しやすい，ストリーム処理に適したループ構文を導入した。ループ構文では，繰返し処理の対象となるデータを“loop”という予約語で示された構文への引数として明示し，その後に個々の要素に行う計算を記述し，出力を保持する変数に代入する。また，決まった回数の繰返し処理を行うには，繰返し回数を“times”という予約語で示された構文への引数として明示し，その後に，各サイクルで行う計算を“loop”の場合と同様に記述する。

繰返しの各サイクルの計算が直前のサイクルの結果に影響されない場合はこれを依存関係のないループと呼ぶ。各サイクルの計算が直前のサイクルの結果を用いる場合は，これを依存関係のあるループと呼び，変数名が“@”で始まる状態変数を使って直前のサイクルの結果を受け取る。また，最初のサイクルでの状態変数の値は，ループ構文の2引数目以降に与えられた値である。

依存関係のないループの例を**図2**に示す。元画像から切り出された画像ブロック集合（blockset）の個々の要素（block）に対してブラー（blur）処理をかけノイズ除去し，その結果である画素（pixel）を画像のフレーム（frame）として出力する例である。ブラー処理は切り出された画像ブロックだけを用いて計算できる。それに対応して，この例では状態変数が現れておらず，各サイクルをどの順序で計算を行ってもよいことが明示されている。

サイクル間に依存関係があるループ構文の例を**図3**に示す。これはビット列の入力に対して誤り訂正符号であるビタビ（Viterbi）符号化を施すプログラムである。ビタビ符号化では，入力ビットと現在の状態に依存して，次の出力符号と次の状態が変化する。

ループ構文を用いることで，最適化するときにはストリーム

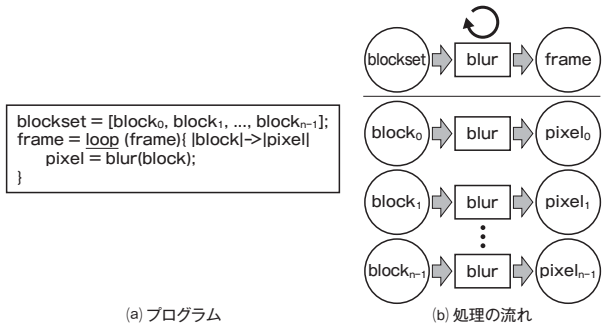

図2. 依存関係のないループ — ストリーム処理の記述に最適に設計されたループ構文を導入したことで，ストリーム処理の書きやすさと自動最適化のしやすさを同時に実現する。

Stateless loop

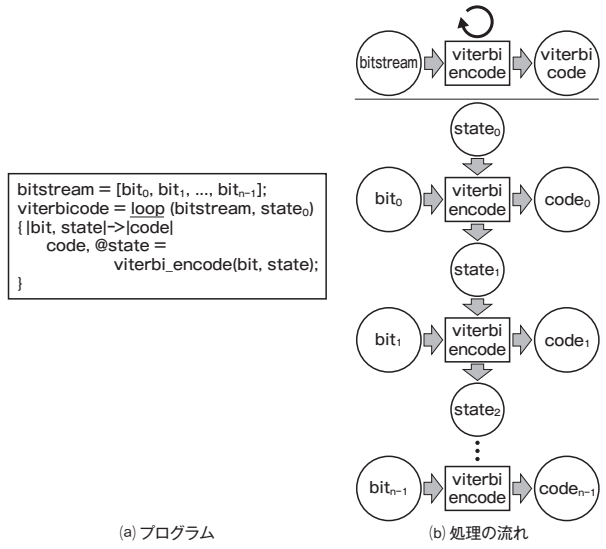

図3. 依存関係のあるループ — あるサイクルと次サイクル間の依存関係の有無を@付き変数で明示するループ構文を導入したことで，ストリーム処理の書きやすさと自動最適化のしやすさを同時に実現する。

Stateful loop

処理を容易に認識でき，各サイクル間の依存関係も状態変数の有無で簡単に判断できる。したがって，並列化などの最適化を行うことが容易である。

3.2 実行順序の自由度を上げる遅延評価

2.2節で述べた順序の変更の問題を解決するために，依存関係にさえ従えば，記述順序にかかわらず実行順序を自由に決められる，遅延評価を導入した。

動画プレーヤのプログラムのイメージを**図4**に示す。遅延評価では，出力として要求された動画のフレームの復号（decode）に必要な計算だけを行えばよい。最初の動画のフレーム復号に必要なだけのデータがロード（load）され，そのデータを使って復号処理が行われる（図4(a)）。更に次のフレームが要求されると，先ほどロードしたデータに加えて必要なデータがロードされ，それらを使って次のフレームの復号が行われる（図4(b)）。

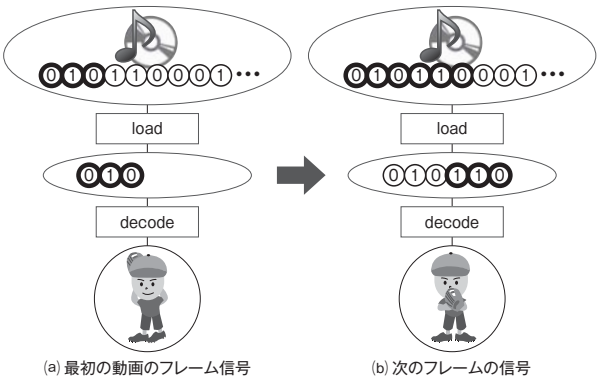

図4. 遅延評価での実行 — 遅延評価を導入したことで，巨大なデータを扱うストリーム処理の書きやすさと自動最適化のしやすさを同時に実現する。

Lazy evaluation for program of video player software

遅延評価を導入すると，実行順序を自由に決められるので，最適化時の自由度も高まる。

3.3 依存関係の解析のための参照透明性と副作用の排除

2.3節で述べた依存関係の解析の問題を解決するために，同じ名前の変数は常に同じ値である参照透明性という性質と，同じ引数で呼び出した関数は必ず同じ結果を返す副作用のない関数呼出しという性質を導入した。

参照透明性を導入すると，同一名の変数への代入は1か所に限られ，副作用のない関数呼出しを導入すると，関数の返り値も引数だけで決定される。この結果，解析すべき範囲が大幅に狭まり，変数と関数の依存関係の情報を得やすくなる。

4 超高位設計言語の適用例

4.1 超高位設計言語のプログラム例と合成例

画像のフィルタ処理を行うfilter関数を超高位設計言語プログラムで記述した例を，**図5**に示す。filter関数は，幅Mピクセル，高さNピクセルである静止画像iを入力として，関数f，gで表わされるノイズ除去処理，エッジ検出処理などの画像フィルタ処理を施してM×Nピクセルの出力静止画像oを計算する。

この例のストリーム処理は，二重のtimesループ構文で書かれており，3.1節で述べたとおり，状態変数が現れないことから，fとgのM×N回の計算は互いに独立に行えることがわかる。

3.2節で述べたとおり，遅延評価が導入されているので，自動合成エンジンは，fとgの依存関係を守る範囲であれば計算順序を自由に決めることができる。また，3.3節で述べたとおり，計算間の依存関係は簡単に抽出できる仕様になっている。抽出した依存関係をたどると，中間静止画像bがM×Nピクセルであること，bの1ピクセルの値はfの1回の計算によって

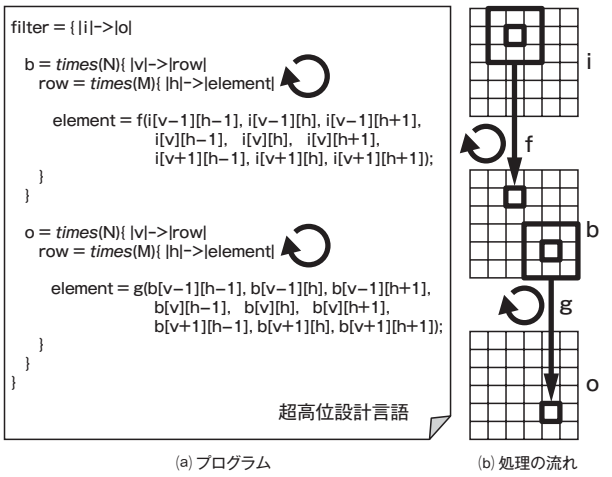

図5. 超高位設計言語プログラムの例 — 超高位設計言語を用いることで，自動最適化を行いやすい画像フィルタ処理プログラムを容易に記述できる。

Example of program written in very high-level design language

求められること，gの1回の計算にはbの9ピクセルの値が必要であることなどがわかる。

更に，bのM×N個分のピクセル値をどのように記憶するか，最適化時に自由に決めることができる。

利用可能なメモリ量が少ないプラットフォーム向けに利用メモリ量を抑えて合成したCプログラムの例を，**図6**に示す。中間画像のM×N個のピクセル値を3×3ピクセル分のサイズを持つメモリbに記憶し，fを3回，gを1回計算する処理をM×N回計算することで出力画像を得る。

```
void filter (unsigned int *i, unsigned int *o){
  unsigned int b[3][3]; // 3 * 3 resisters
  for(int v = 0; v < N; v ++){
    for(int h = 0; h < M; h ++){
      b[0][0] = b[0][1]; b[0][1] = b[0][2];
      b[1][0] = b[1][1]; b[1][1] = b[1][2];
      b[2][0] = b[2][1]; b[2][1] = b[2][2];
      b[0][2] =
        f(i[(v-2)*M+h], i[(v-2)*M+(h+1)], i[(v-2)*M+(h+2)],
          i[(v-1)*M+h], i[(v-1)*M+(h+1)], i[(v-1)*M+(h+2)],
          i[     v*M+h], i[     v*M+(h+1)], i[     v*M+(h+2)]);
      b[1][2] =
        f(i[(v-1)*M+h], i[(v-1)*M+(h+1)], i[(v-1)*M+(h+2)],
          i[     v*M+h], i[     v*M+(h+1)], i[     v*M+(h+2)],
          i[(v+1)*M+h], i[(v+1)*M+(h+1)], i[(v+1)*M+(h+2)]);
      b[2][2] =
        f(i[     v*M+h], i[     v*M+(h+1)], i[     v*M+(h+2)],
          i[(v+1)*M+h], i[(v+1)*M+(h+1)], i[(v+1)*M+(h+2)],
          i[(v+2)*M+h], i[(v+2)*M+(h+1)], i[(v+2)*M+(h+2)]);
      o[v*M+h] = g(b[0][0], b[0][1], b[0][2],
                   b[1][0], b[1][1], b[1][2],
                   b[2][0], b[2][1], b[2][2]);
    }
  }
}
```

図6. 合成したCプログラムの例 — 超高位設計言語で記述された画像フィルタ処理プログラムから，プラットフォームに適したCプログラムを自動合成できる。

Example of automatically generated C program

4.2 効果

これらの例からわかるとおり，3章で説明したループ構文，遅延評価，及び参照透明性と副作用のない関数呼出しを導入したことで最適化の自由度が高まり，与えられたプログラムを様々な観点で最適化しやすくなる効果が得られた。

また，必要以上の実装を記述しないため再利用しやすく，ストリーム処理を簡潔に記述できる，というようにアルゴリズムの本質だけを簡潔に記述できる効果が得られた。

超高位設計言語を用いた超高位設計では，C言語を用いた従来の設計手法と比較して設計コストが減少し，設計資産の再利用性が増すことにより生産性が向上することが期待できる。

5 あとがき

当社が現在開発中である超高位設計の概要と，アルゴリズムを記述する言語の特長について述べた。

当社が設計した超高位設計言語は，ストリーム処理アルゴリズムの記述がしやすく，かつハードウェアと相性がよいループ構文を持つ。更に，最適化のための情報抽出を容易にする参照透明性，遅延評価など関数型言語の特徴も持つ。C言語と比較すると，より簡潔にアルゴリズムを記述でき，また，プラットフォーム向けの最適化プログラムを生成するときに，より高い自由度を提供できる。

これにより，一つの超高位設計言語プログラムから多様なプラットフォームそれぞれに向けて効率のよいプログラムを生成する超高位設計の基盤を確立できた。また，アルゴリズムの再利用性が増し，組込みシステム開発における生産性を向上させる次世代設計環境を実現できると言える。

白井　智　SHIRAI Satoshi

研究開発センター コンピュータアーキテクチャ・セキュリティラボラトリー研究主務。

デザインオートメーション分野の研究・開発に従事。

Computer Architecture & Security Systems Lab.