計算機アーキテクチャ　127－2
設　計　自　動　化　86－2
（1997．12．11）

非一様構造を考慮した シストリックアルゴリズム記述言語と処理系

○ 菅谷 至寛，阿曽 弘具

東北大学大学院工学研究科

あらまし　シストリックアルゴリズムをハードウェアとして実現するために，その実現可能性を検討するための情報が必要である．しかし，シストリックアルゴリズムレベルで得られる情報は，そのためには不十分であり，設計等に対する論理合成を行ってそのゲート数などを検討する必要がある．シストリックアレーの論理設計から実際のハードウェア記述言語による記述に変換することには定型的作業が多い．しかし，論理設計におけるバグやハードウェア記述後にわかる不具合の解消のため論理設計を見直すこともある．そのようなシストリックアレーの開発過程の高効率化のため，非一様構造を含むシストリックアレーの記述言語を考案し，ハードウェア記述言語による記述を生成する処理系を開発した．

和文キーワード　シストリックアルゴリズム，記述言語，並列処理

Systolic algorithm description language including un-uniform structure and Translation system

Yoshihiro SUGAYA, Hirotomo ASO

Graduate School of Engineering, Tohoku University, Sendai-shi, 980-77 Japan

Abstract　When systolic array is designed for hardware, an evaluation of the capability of realization is needed. Because the information known in systolic algorithm level is not enough for the realization, logic synthesis is necessary to know the number of gates and delay time, etc. Translation from systolic algorithm into hardware description language is usually fixed-form work. But some time the design is changed because of some bugs of the design or problems that are known after logic synthesis. To raise the efficiency of the design process, systolic algorithm description language is proposed, and the system that translate from the language into hardware description language is produced.

英文 keywords　systolic algorithm, description language, parallel processing

1. はじめに

データの依存性が比較的規則的な問題を，効率良く並列に計算するアルゴリズムとして，シストリックアレーが知られている．

シストリックアレーは問題の規則性やデータ依存の局所性などの特徴を利用した問題指向のアレープロセッサの一種であるので，解く問題によってシストリックアレーを構成するPE(Processing Element) や PE 間の結合が大きく異なる．したがって，各応用毎に異なるシストリックアレーをその都度設計する必要がある．また，一つの問題に対しても，PE 数や実行ステップ数の異なるシストリックアルゴリズムが多数存在する．

そこで，与えられた問題をシストリックアルゴリズムに変換する一般的な手法が必要とされ，これまで多くの研究が行なわれてきている[1][2][3][4][5]．これらの研究によって，半自動的にシストリックアルゴリズムの設計を行なうことが可能であり，シストリックアルゴリズムを半自動的に設計するシステムも開発されている．

しかし，シストリックアルゴリズムレベルで得られる情報は，実際にハードウェア化を考慮して実現可能性を検討する上で不十分であり，ゲート数などの，より物理的条件に即した情報を得るためには，少なくとも論理設計を行なうことが必要である．

論理設計には人手によって行なう方法と，ハードウェア記述言語によって回路の機能／動作を表現し，処理系によって論理回路を得る方法がある．後者の方が設計者にとって負担が少ないが，それでも，シストリックアルゴリズムそのものや，その他の仕様を変更する場合には，その都度ハードウェア記述言語による回路記述の書き直しが必要である．前述したように，シストリックアルゴリズムの設計は自動設計法が研究されているが，この部分は人手で行なわなければならなかった．

そこで本研究では，シストリックアルゴリズム記述言語を提案し，その記述言語による記述からハードウェア記述言語による記述へ変換する処理系を作成した．

2. シストリックアルゴリズム記述言語

計算機でシストリックアルゴリズムを正確かつ効率的に扱うためには，仕様記述言語などによる形式的な表現が不可欠である．そこで本研究では，ハードウェア記述言語への変換を考慮した同期式シストリックアルゴリズムの仕様記述言語 (SAL: Systolic Algorithm description Language) を提案する．

2.1 基本構文

SAL の基本記述単位はモジュール (module) であり，C++ におけるクラス (class) の定義に相当する．モジュールとは入力ポートと出力ポートを持つ処理単位であり，PE 一つ一つやシストリックアレー全体などに対応する．モジュールは階層的に記述可能で，モジュールには他のモジュールのオブジェクトを含む事ができる[1]．

モジュールは，入力ポート・出力ポート宣言部と，動作・構造定義部からなる．

module *module-name* {
interface:
　入力ポート・出力ポート宣言部
structure:
　動作・構造定義部
}

入力ポート・出力ポート宣言部では，そのモジュールに存在する入力ポートや出力ポートの宣言を行なう．**inport** 識別子で入力ポートを，**outport** という識別子で出力ポートをそれぞれ宣言する．出力ポートを宣言する際，隣の PE にデータを送る際に挿入する遅延ステップ数についても同時に指示する．遅延数が 0 の場合，省略できる．

動作・構造定義部では，PE 間の結線の構造や，PE 内での実際の動作について記述する．モジュール内では他のモジュールを部品として利用し，部品である子モジュールのオブジェクト間の関係や，そのモジュールで定義した入出

[1] なお，一つのファイルで定義できるモジュールは一つのみであり，もし複数のモジュールが一つのファイル内に存在する場合には，最後に現れる一つのモジュールのみ有効とする．

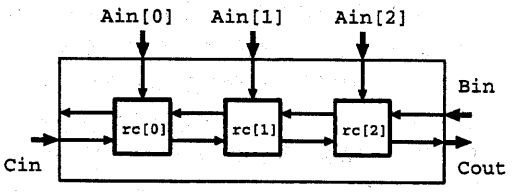

図 1. 行列・ベクトル乗算を行なうシストリックアルゴリズム.

力ポートと子モジュールのオブジェクトとの関係を記述できる．また，ポート間での演算を記述する事もできる．

以下に行列・ベクトル乗算 $c = Ab$ を行なうシストリックアルゴリズムを挙げ，記述例を示す．ここで，A は式 1 で表される行列，b, c はそれぞれ縦ベクトルである．

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1N} & & 0 \\ & a_{22} & \cdots & a_{2,N+1} & \\ & & \ddots & & \\ 0 & & & & \end{pmatrix} \quad (1)$$

この演算を行なうシストリックアルゴリズムの代表的なものとして，図 1($N = 3$ のとき) がある．図 1で正方形で表した rc[0], rc[1], rc[2] がセル (PE) であり，

$$c = c + a * b$$

の演算を行う．

図 2 に本記述言語による構造定義部の記述例を示す．ここでは次のような事を記述している．

(1) **interface:** 識別子で指定される入力・出力ポート宣言部において，アレー全体の入力・出力ポートの宣言を行なう．

(2) **structure:** 識別子で指定される動作構造定義部の先頭部分において，部品として使用するモジュール（オブジェクト）の宣言をする．

(3) 動作·構造定義部において，各オブジェクト間またはアレーの入出力ポートの接続の定義を行なう．例えば “Cout = rc[i].Cout;” はアレー出力ポート Cout にオブジェクト rc[i] の 出力ポート Cout を接続することを定義している．

```
#include "cell.scds"

module band_mat {
interface:        // 入出力ポート宣言部
  inport Ain[N], Bin, Cin;
  outport Cout;
structure:              // 構造定義部
  module cell rc[N];// セルの宣言
  int i;          // 構造記述のための
                  // 仮想的な変数
  for (i = 0; i < N; i++) {
      if (i == 0) {
        Cout = rc[i].Cout;
        rc[i].Bin = Bin;
      } else if (i == N-1) {
        rc[i].Cin = Cin;
      }
      if (i != 0) {
        rc[i].Bin = rc[i - 1].Bout;
        rc[i - 1].Cin = rc[i].Cout;
      }
      rc[i].Ain = Ain[i];
  }
};
```

図 2. 行列・ベクトル乗算を行なうシストリックアルゴリズムのアレー構造の記述.

int 型の変数 i は for 文，if 文や配列を用いて効率的に構造を記述するための記述用変数であり，コンパイル時に評価され，ハードウェア記述には使われない．N はセル数を表すパラメータであるが，これは define マクロで定義するもので，前処理 (preprocessing) のマクロ展開によって，指定された定数に置換される．記述用変数を用いた式には，inport, outport 変数が現われてはならない．

次に，セル機能記述部の記述例を図 3 に示す．文法的にはアレー構造の記述とセル機能の記述は区別しておらず，同一である．

ここでは，次のような事を記述している．

- 一つのセルの入力・出力ポートの宣言．
- セル内での演算．
- 各出力ポートに出力するデータの指定．
- データを出力する際の遅延ステップ数．**outport** における括弧内の数値が，データを隣の PE に送る際の遅延数を示す[2]．

[2] アレー構造の記述においても同様に遅延数は記述できる．

```
module cell {
interface:
  inport Ain, Bin, Cin;
  outport Aout(1);
  outport Bout(1);
  outport Cout(1);// 括弧内は遅延数
structure:
  Aout = Ain;
  Bout = Bin;
  Cout = Cin + Ain * Bin;
};
```

図 3. 行列・ベクトル乗算を行なうシストリックアルゴリズムのセル機能の記述.

3. 非一様シストリックアルゴリズムのための SAL の拡張

シストリックアルゴリズムは一様構造のものが多いが，実用的なシストリックアルゴリズムの中にも非一様構造のもの，すなわち，異なる種類の PE を含むものが存在する．このようなシストリックアルゴリズムをも効率的に扱えるようにするために，SAL を拡張する．

3.1 オブジェクト置換

そもそも，シストリックアルゴリズムは問題の規則性などの特徴を利用したアレープロセッサの一種であるため，非一様構造とは言ってもシストリックアレーを構成する全ての PE が異なるわけではなく，ごく一部（境界部分の PE であることが多い）の PE だけが他と異なることが多い．したがって，一部の他と異なる PE を疑似的に同じものであるとし，非一様であるものでも疑似的に一様であるようにみなす仕組みを言語仕様に採り入れることによって，非一様シストリックアルゴリズムを効率的に扱うことができる．

そこで，オブジェクト置換という概念を導入する．オブジェクト A をオブジェクト B で置き換えるという事を，オブジェクト置換演算子 "@ =" を用いて次のように表現する．

$$A \ @= B;$$

両辺のオペランドは，部品として使用するモジュールの型を持つオブジェクトでなければならない．

このオブジェクト置換文によって，非一様シストリックアルゴリズムをあたかも一様であるかのように見なし，結線構造を記述することができる．一様でない部分の実際のオブジェクト割り当ては，オブジェクト置換文で行なう．また，アレー構造の記述において，結線構造の記述と実際に用いられるオブジェクトのを割り当てを分離できることになる．

3.2 継承

オブジェクト置換を無制限に許せば，セル間の結線が保証されないという問題が起こる．置換したオブジェクトが置換されたオブジェクトが持っているはずのポートと同じポートを持つとは限らないからである．

オブジェクト指向言語の重要な概念の一つに継承がある．public 継承の場合，派生クラスは基底クラスの性質を持つ（is-a 関係）ため，派生クラスは基底クラス（の一種）であると言える．この性質を利用することによって，不適切な置換を回避できる．

SAL によるシストリックアルゴリズムの記述では，アレー構造の記述とセル機能の記述は独立している．したがって，アレー構造の記述においては，セル内でどんな処理を行なうのかという事には関知せず，ブラックボックスとみなせる．必要な情報は入出力ポートの名前のみである．よって，各セルの入出力端子が同一ならば，アレー構造記述においては同種のセルとみなせる．

以上をふまえ，SAL における継承を次のように定義する．

- 派生クラスでは，基底クラスのポートを継承する．派生クラスでは基底クラスには存在しない入出力ポートを追加する事ができる．
- 基底クラスのオブジェクトから，その派生クラスのオブジェクトへの置換だけを許す．
  これは，C++ において派生クラスのポインタが基底クラスのポインタに変換可能である事に対応する．これが許されるのは is-a 関係が成立するからである．

これによって，オブジェクト置換に伴う結線の矛盾を言語仕様の枠内で防止でき，セル間の結線の記述と実際のセルの割当の分離が可能になる．

本記述言語において，継承の指示はモジュールの定義の際にコロンの後に基底クラスモジュールを指定する事で記述する．*base-class-name* というモジュールを継承して，*module-name* という名前のモジュールを新たに定義する場合，次のように記述する．

module *module-name* : *base-class-name* {
interface:
　　入力ポート・出力ポート宣言部
structure:
　　動作・構造定義部
}

最初の 1 行以外は，基本的に通常のモジュールの定義と同じである．ただし，新たに定義するモジュールで基底クラス（基底モジュール）で定義されているポートのみを持つ場合，**interface**: で始まる入力ポート・出力ポート宣言部は省略できる．

例として下三角線形問題を解くシストリックアルゴリズムを挙げる．下三角線形問題とは，次のような問題である．A, b をそれぞれ

$$A = (a_{ij}): \text{ 正則 } n \times n \text{ 下三角行列}$$
$$b = (b_1, b_2, ..., b_n)^t: \ n \text{ 次元ベクトル}$$

とする．A, b が与えられているとき，

$$Ax = b \tag{2}$$

となるような $x = (x_1, x_2, ..., x_n)^t$ を求める．ベクトル x は，次の漸化式で計算できる．

$$y_i^{(0)} = 0 \tag{3}$$
$$y_i^{(k)} = y_i^{(k-1)} + a_{ik}x_k \tag{4}$$
$$x_i = \left(b_i - y_i^{(i-1)}\right) / a_{ii} \tag{5}$$

下三角線形問題を解くシストリックアルゴリズムの一つに，図 4 で示すシストリックアルゴリズムがある．図 4 において，一番左側の円形で示したセルが式 5 の計算を行なうためのもので，他のセルとは異なる構造を持つ．正方形で記述した他のセルはすべて同一であり，式 4 の計算を行なう．

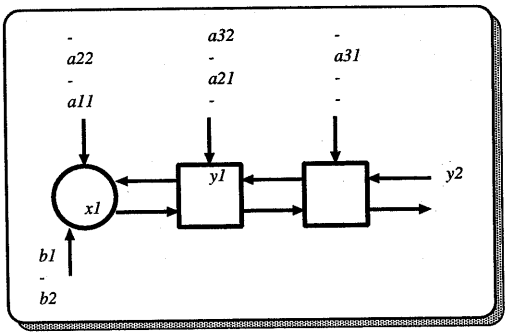

図 4. 下三角線形問題を解くシストリックアルゴリズム．

```
module inn_prod {
interface:
  inport Ain, Xin, Yin;
  outport Aout(1), Xout(1), Yout(1);
structure:
  Aout = Ain;
  Xout = Xin;
  Yout = Yin + Ain * Xin;
};
```

図 5. 基本セルの記述（1）

まず，一般的なセル（図 4 において正方形で描かれたセル）を記述する（図 5）．このセルでは次の処理を行なう．

- $y = y + a * x$ の演算を行う．
- Ain から入力されたデータを Aout に，Xin から入力されたデータを Xout に，上記の演算の結果を Yout にそれぞれ出力する．その際，1 step の遅延を入れる．

次に，基本セルを継承して特殊セル（図 4 において円形で描かれたセル）を図 6の様に記述する．このセルでは，基本セルに存在する入出力ポートに加えて Bin, Bout を定義している．このセルでは，次の処理を行なう．

- $x = (b - y)/a$ の演算を行う．

```
#include "inn_prod.scds"

module lin_cell : inn_prod {
interface:
  inport Bin;
  outport Bout(1);
structure:
  Aout = Ain;
  Bout = Bin;
  Yout = Yin;
  Xout = (Bin - Yin) / Ain;
};
```

図 6. 特殊セルの記述（1）

```
#include "inn_prod.scds"
#include "lin_cell.scds"

module lin {
interface:
  inport Ain[N], Bin, Yin;
  outport Xout;
structure:
  module inn_prod inn[N];
  module lin_inn lc;
  int i;

  inn[0] @= lc; /* モジュール置換 */
  for (i = 0; i < N; i++) {
    if (i == N-1) {
        inn[i].Yin = Yin;
    }
    if (i != 0) {
      inn[i].Xin = inn[i - 1].Xout;
      inn[i - 1].Yin = inn[i].Yout;
    }
    inn[i].Ain = Ain[i];
  }
  inn[0].Bin = Bin;
};
```

図 7. アレー構造記述（1）

- 隣のセルに出力を流す．その際，それぞれ 1 step の遅延を入れる．

アレー構造記述は図 7 のようになる．片端の基本セルタイプ (inn_prod) のオブジェクト (inn[0]) を特殊セルタイプ (lin_inn) のオブジェクト (lc) で置き換え，基本セルタイプのオブジェクト名 (inn[i]) によって全ての構造記述を行なっている．この記述の概念を図で表すと，図 8 のようになる．

さらに，構造記述にのみ用いる仮想的なセルを導入する事によって，より抽象的に結線構造を記述する事ができる．この方法による記述例

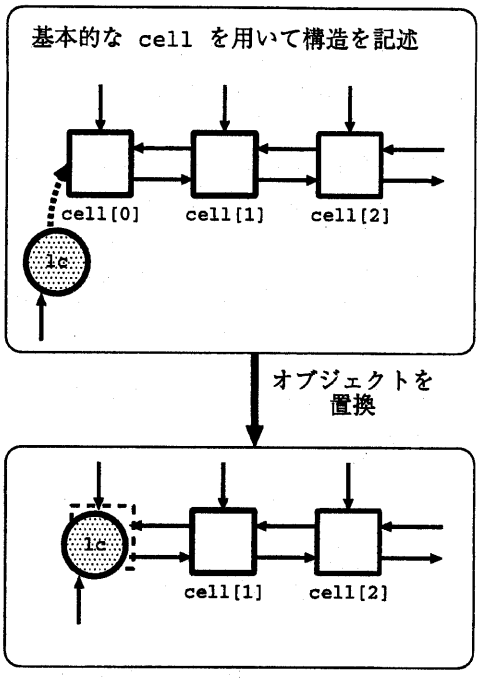

図 8. 記述例（1）の概念

```
module lin_base_cell {
interface:
  inport Ain, Xin, Yin;
  outport Aout(1), Xout(1), Yout(1);
};
```

図 9. 仮想セル記述

を以下に示す．

構造記述に用いる仮想的なセル (lin_base_cell) を定義する（図 9）．このセルはポートの定義のみで，動作・構造定義部は記述しない．インターフェース (interface) のみを記述する．

次に，仮想セル (lin_base_cell) を継承して基本セル (inn_prod_inher) と特殊セル (lin_cell_inher) を定義する．基本セルの記述を図 10 に，特殊セルの記述を図 11 に示す．それぞれのセルでの処理内容は，先の記述例と同様である．基本セルでは，仮想セルで定義した入出力ポート以外にポートは存在しないので，機能・構造記述部 (structure) のみとなる．特殊セルは，モジュール名以外は先の記述例と全く同じとなる．

```
#include "lin_base_cell.scds"

module inn_prod_inher :
         lin_base_cell {
structure:
  Aout = Ain;
  Xout = Xin;
  Yout = Yin + Ain * Xin;
};
```

図 10. 基本セルの記述 (2)

```
#include "lin_base_cell.scds"

module lin_cell_inher :
         lin_base_cell {
interface:
  inport Bin;
  outport Bout(1);
structure:
  Aout = Ain;
  Bout = Bin;
  Yout = Yin;
  Xout = (Bin - Yin) / Ain;
};
```

図 11. 特殊セルの記述 (2)

全ての構造記述は，仮想セル (lin_base_cell) のタイプを持つ cell[i] によって行なう．物理的なセルの割り当ては，モジュール置換によって行なっている．左端のセルにあたる仮想的なセル (cell[0]) を特殊セルのオブジェクト (lc) で，それ以外の仮想セルのオブジェクト (cell[i]) を標準セルのオブジェクト (inn[i-1]) で置換する．したがって，構造記述と物理的なセルの割当がほぼ分離できていると言える．この記述の概念を図で表すと，図 13 のようになる．

4. ハードウェア記述言語への変換

SAL によるシストリックアルゴリズムの記述を，ハードウェア記述言語 (HDL: Hardware Design Language) による記述に自動的に変換する処理系を作成した． 出力 HDL としては，PARTHENON[8] の言語である SFL(Structured Description Language) を採用した．

4.1 変換システム

シストリックアルゴリズム記述言語によって記述されたシストリックアルゴリズムをハード

```
#include "lin_base_cell.scds"
#include "lin_cell_inher.scds"
#include "inn_prod_inher.scds"

module lin {
interface:
  inport Ain[N], Bin, Yin;
  outport Xout;
structure:
  module lin_base_cell cell[N];
  module inn_prod_inher inn[N-1];
  module lin_cell lc;
  int i;

  cell[0] @= lc; /* モジュール置換 */
  for (i = 1; i < N; i++)
    cell[i] @= inn[i-1];

  for (i = 0; i < N; i++) {
    if (i == N-1) {
      cell[i].Yin = Yin;
    }
    if (i != 0) {
      cell[i].Xin = cell[i - 1].Xout;
      cell[i - 1].Yin = cell[i].Yout;
    }
    cell[i].Ain = Ain[i];
  }
  cell[0].Bin = Bin;
};
```

図 12. 仮想セルを用いたアレー構造の記述

ウェア記述言語によるものに自動変換するシステムを作成した．

本処理系では，大きく分けて次の変換処理を行う．

- 入出力端子・仮想端子・レジスタの管理
  入出力端子は基本的にソースで示された通りに割り当てる．仮想端子・レジスタは処理系側で必要なだけ割り当てる．
- 式の変換
  式を “変数代入式” 又は “2 項演算代入式” の系列に変換し，四則演算をライブラリ呼び出しに適合する形にして出力する．
- ループ・条件文の展開
  for 文，if 文は結線の記述のために用いられるが，これを展開して実際の素子間結線に変換する．
- 配列の展開
  構造記述に用いられる配列を通常の変数に

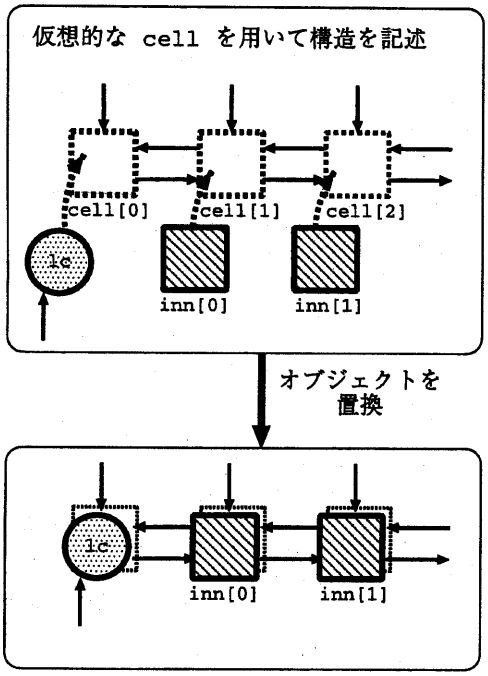

図 13. 記述例（2）の概念

置き換える．その際，添字が計算によって解決できなければエラーにする．

定数式は構文木の組み立ての段階であらかじめ計算しておく．また，演算器は必要なだけ割り当てる．なお，入力記述では C++ と同様なマクロ記述が可能で，その展開のために，これらの処理の前にプリプロセッサ cpp を使用する．

この処理系によって出力されるシストリックアレーの SFL による記述を，論理合成システム PARTHENON の入力とすることによって，対象システムのハードウェア設計結果が得られ，ゲート数，遅延時間などを評価，検討する事ができる．SAL による記述中でマクロを用いてセル数を指定する事によって，記述を全く変更することなく，セル数などをを変えて繰り返し評価する事が可能である．

5. むすび

非一様シストリックアルゴリズムをも効率的に扱う事が可能なシストリックアルゴリズム記述言語を与え，自動的にハードウェア記述言語に変換する為の処理系を開発した．この処理系と既存の PARTHENON を組み合わせて，シストリックアレー開発支援システムを構成した．

参考文献

[1] D. I. Moldovan: "On the Design of Algorithms for VLSI Systolic Arrays", PROCEEDINGS OF THE IEEE, 71, Vol.1, pp.113–120, 1983.

[2] 阿曽弘具: "シストリックアレーの自動設計法", 電子情報通信学会論文誌 (D), Vol.J71-D, No.8, pp.1487–1495, Aug. 1988.

[3] 小川誠治，前場隆史，阿部健一: "多次元シストリックアレーの系統的設計手法", 電子情報通信学会論文誌 (D-I), Vol.J75-D-I, No.9, pp.879–881, Sep. 1992.

[4] 小川誠治，前場隆史，阿部健一: "射影法に基づくシストリックシストリックアレー設計支援システム: ADSAP", 1990 年 電子情報通信学会春季大会, D-108, 1990.

[5] V.K.Prasanna Kumar, Yu-Chen Tsai: "On Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Multiplication", IEEE TRANSACTIONS ON COMPUTERS, VOL.40, NO.6, JUNE 1991.

[6] 菅谷至寛，阿曽弘具: "シストリックアルゴリズムのハードウェア化支援", 1996 年電子情報通信学会ソサイエティ大会, D-68, Sep. 1996

[7] Margaret A.Ellis, Bjarne Stroustrup（足立高徳，小山裕司 訳）: "注解 C++ リファレンス・マニュアル (The Annotated C++ Reference Manual)", トッパン, 1992.

[8] "PARTHENON 講習会資料", NTT データ通信株式会社.