修士学位論文

シストリックアレーの効率的構成法に関する研究

東北大学大学院工学研究科
電気・通信工学専攻

佐藤　英

目 次

図目次

表目次

第 1 章

序論

1.1 本研究の背景

1946 年に Von Neumann によって計算機が産声を上げて以来，約半世紀になる．そしてその発展は実に急速なものであった．半導体などの内部素子の高速化，高精度化が，まさに日進月歩の勢いで進み，計算機の計算速度はおよそ 10 年毎に一桁速くなっている．その一方では，集積回路技術の発達によって，かつては部屋いっぱいを占領していた回路が，現在では手のひらに乗るほどの大きさで実現されている．

そして，この，計算機の高速化，高精度化は，人間社会に多大なる寄与をもたらしたと言えよう．認識科学への応用やネットワーク通信，計算機シミュレーションや，その他たくさんの分野に計算機は利用され，もはや計算機は学問研究や，その他たくさんの分野において必要不可欠の存在になったとさえ言える．

しかしながら，その反面，更なる計算機の高速化が必要な分野も存在する．信号処理や記号処理，コンピュータグラフィックスなどはその一例である．また，気象データの計算などの大規模科学技術計算もあるだろう．

これらの計算に対し，過去半世紀の研究者たちは，そのほとんどが，シングルプロセッサによる，ノイマン式計算機の高速化に精力を注いできた．具体的には，

- CPU の性能向上
- メモリ容量の増大
- CPU-メモリ間のバス幅を拡大，データ/命令転送速度の上昇
- メモリ・インタリーブ

- パイプライン
- RISC (Reduced Instruction Set Computer)
- スーパースカラ
- VLIW (Very Long Instruction Word)

などである [1].

しかし，これらの方法による高速化にも段々と限界が見え始めている．本来，プロセッサの処理速度をいくら上げようとも，結局，物理的な限界が存在するのがその要因である．そこで，ノイマン計算機のように計算を最初から順番に解いていく**逐次処理**ではなく，個々の小さな計算を複数個のプロセッサに割り当てて，同時刻に複数の計算を行うことによって計算時間を小さく抑える**並列処理**の有効性が認識され，その方法がさかんに研究されるようになった．

これまで研究されている，並列処理を実行するアーキテクチャは，大別すると以下の2つに分類することができる [2].

- ベクトル計算機
- アレープロセッサ

ベクトル計算機は，ソフトウェア的な並列性を有したもので，演算をいくつかの基本ステップに分解して，各ステップ毎に，基本演算装置を割り当て，時分割で多数のデータを処理するようにした方式である．一方，アレープロセッサは，ハードウェア的な並列性を有したもので，計算素子 (セル) を並列的に配置し，それぞれのセルに計算を割り当てることによって計算の高速化を計っている．

アレープロセッサは多数のプロセッサのごく一部しか使用されないことが多いが，ベクトル計算機は，部分的活動のほとんどを一定に保ちながら，ハードウェア量を減らすことができる．また，ベクトル計算機は，任意のサイズのベクトルやアレーも取り扱うことができ，スーパーコンピュータなどに応用されている．一方，アレイプロセッサは，ベクトル計算機が苦手とする分岐命令や割り込みなどの処理にも対応でき，ハードウェア自体の製作コストも年々低くなっていることから，並列計算アーキテクチャとしてはより多くの研究がなされている．シストリックアレーの接続形態やマルチプロセッサなどがアレープロセッサに含まれる．

1978 年に Kung および Leiserson によって提案されたシストリックアレーは，アレープロセッサの一種で，並列処理アーキテクチャとして非常に有効であり [3]，記号処理，信号処理など，一部の分野では実用化もされている．シストリックアレーは，単純計算を行

う素子 (セル) を規則正しく配列し，計算に要するデータをパイプライン的に流し込むことによって並列計算をおこなう．構造が一様であるため拡張性に優れ，また通信が局所的であるため故障などにも強いといった利点がある．

しかし，シストリックアレーを構成するに当たっては，ある問題が存在する．それは，与えるプログラムに対して，ほとんどの場合，複数種の構成が考えられ，その結果，どの構成が最良であるかという問題が発生することである．このことを具体例を挙げて説明する．

以下のアルゴリズムは，緩和法によるラプラス方程式の求解アルゴリズムである．

for $i = 1$ **to** L
for $j = 1$ **to** M
for $k = 1$ **to** N

$$u_1 = u(i_1 + 1, i_2)$$
$$u_2 = u(i_1, i_2 + 1)$$
$$u_3 = u(i_1 - 1, i_2)$$
$$u_4 = u(i_1, i_2 - 1)$$
$$u(i_1, i_2) = \frac{1}{4}(u_1 + u_2 + u_3 + u_4)$$

上の処理をシストリックアレー上で実現した場合，$L = M = N = 3$ とすると，可能な構成が複数個存在している．そのうちの二つの例を図 1.1，図 1.2に示す．図 1.1，図 1.2は，どちらも最終的には同じ処理を完了し，計算結果も同一となる．しかし，処理を行うシストリックアレーは，使用セル数が異なっている．この場合，ハードウェア的な資源を小さく抑えたいと考えるのであれば，図 1.1のシストリックアレーを選択する方がよい．

このように，最終的な計算結果は同じでも，計算に要するセル数や計算時間などが異なる場合が多い．そのため，いかにして構成の結果が効率的になるように設計するかが重要であると言える．

1.2 本研究の目的

本研究では，シストリックアレーの構成に関し,「その効率性はどのように表わされるか」，また,「いかに効率的に構成できるか」を追求することが目的である．

現在，シストリックアレーを評価する際に使用されうる関数，変数は実に多数のものが定義されている [4,5]．しかし，これらの値を融合した評価関数についてはほとんど研

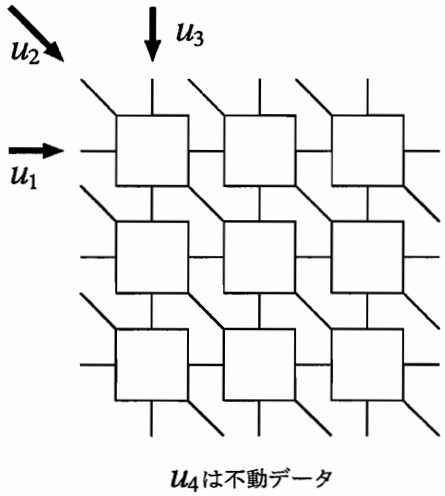


図 1.1: 緩和法の処理を行うシストリックアレー (1)

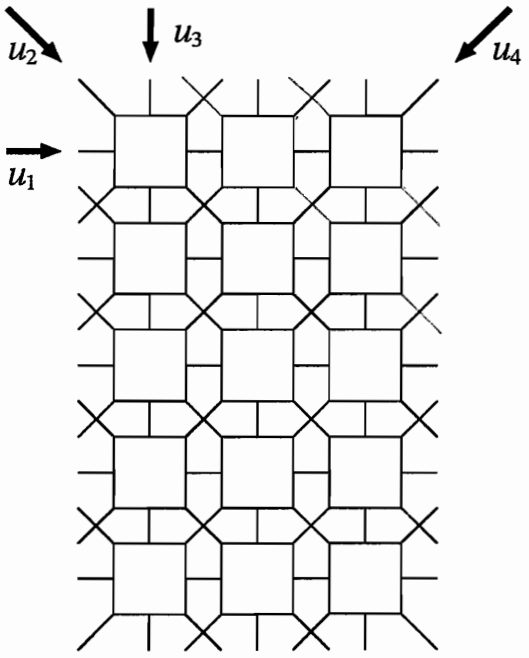


図 1.2: 緩和法の処理を行うシストリックアレー (2)

究がなされていない．この理由としては，実際にシストリックアレーを製作する段階になれば，そのときの製作事情によって，ハードウェア量の削減を優先させたい場合もあれば，逆に，計算時間の短縮を優先させたい場合も存在するため，一概に関数を定義してしまうのは危険であるとの考えがあるからである．しかし，ばらばらに存在する評価値の，それぞれの意味を少しずつ含ませた融合的関数が存在すれば，設計，評価に関して1つの指針ができる．

また，効率性の更なる上昇を実現する設計法を検討することも大変重要である．コスト面での効率化はシストリックアレーを構成するに当たっては頻繁に出て来る要求であり，体系的に，効率的な設計をする必要がある．

そこで本研究においては，まず主な評価値に関する考察，再検討を行ったのち，製作事情に柔軟に応じられるように重み変数を用いた関数を提案し，その関数の特徴を考察する．

さらに，効率性を上昇させる設計法の1つとして BR 法 (Blocking and Retiming method) を提案する．BR 法は，時間的資源の消費を小さく抑える方法である．BR 法は，シストリックアレーの計算時間を単にステップ数のみから考えるのではなく，セル内での計算時間も考慮し，細粒度のアルゴリズムを意図的に粗粒度に変換して，レジスタグラフを用いて計算時間を小さくする方法である．

1.3　本論文の構成

本論文の構成は以下のとおりである．

第1章では，本研究の背景，目的及び本論文の構成について述べる．

第2章では，シストリックアレーの概要及び一般的な構成法について述べる．

第3章では，シストリックアレーに関する個々の評価値についての考察，及びシストリックアレーを空間的資源，時間的資源の両面から融合的に評価する関数について述べる．

第4章では，シストリックアレーの効率的設計を実現する1つの方法として，BR 法を提案し，具体例を用いてその有効性を述べる．

第5章では，本論文の結論及び今後の問題点について述べる．

第 2 章

シストリックアレーの構成の原理

2.1 はじめに

シストリックアレーは,「蜂の巣アレー」などとも呼ばれている並列計算アーキテクチャであり，世界中に数多くの研究者を持っている．特定のアルゴリズムに対して，その問題を解くシストリックアレーは現在まで，実に様々なものが提案されている [6–10]．

シストリック (systolic) とは，心臓が収縮してどきどきしている様を意味する形容詞である．単純な処理を行うセルを心臓に見立てており，個々のセルが周囲のセルへデータを送り出し，また，自分自身も周囲のセルからデータを受け取っている様子を心臓収縮にたとえている．結合形態もさまざまなものが提案されており，図 2.1の線型，図 2.2の六角形型，図 2.3の木結合型などはその一例である． シストリックアレーでは，隣接するセル同士が通信線で結ばれ，その線に沿ってデータがアレー中をパイプライン的に流れる．そして，複数のデータが同じ時刻に同じセルで出会うと，そのセルにおいて処理が行われる．この処理は，普通，四則演算や論理演算など，ごく単純な処理である．時間がたつごとに同時刻に複数のセルで処理が並列的に行われる．そして，データがアレーから出るとき，計算結果となるデータ流を出力して計算を終了する．各ステップは，全部のセルに対して制御装置が同期信号を送ることによって同期的に処理を実行しているが，セル数が増加すると，同期信号のずれなどのために，全部のセルに正しい信号を行き渡らせることが難しくなるため，各計算ステップを非同期的に処理するように改良されたウェーブフロントアレー [11] も研究されている．

2.2 シストリックアレーの構成法

2.2.1 数学的準備

本論文では，依存グラフや，レジスタグラフといった，グラフを用いてアルゴリズム設計法を記述している．本節では，説明の際に用いる用語についてまとめる．

グラフ用語

グラフとは，図2.4に示すようなものである．グラフでは，**ノード**を丸で，**エッジ**を矢印で記述する．図2.4のグラフでは，$a_i(i=1,2,\cdots,6)$ がノードであり，$w_{ij}(i,j=1,2,\cdots,6)$ がエッジである．w_{ij}の値は，ノード a_iとノード a_jの間の重みを表わす．この値は，ノード a_iからノード a_jに行くのに要するコストと考えることもできる．

zero-edge，non-zero-edge は，それぞれ重みが 0 のエッジ，重みが非 0 のエッジのことである．

zero-path とは，zero-edge のみを通過するパスのことである．zero-edge，zero-path の概念はレジスタグラフ[1]を考えるときに重要となる．

依存ベクトル・依存グラフ

アルゴリズムを以下のようなループプログラムで記述する．

for $i_1 = i_{1b}$ **to** i_{1e}
for $i_2 = i_{2b}$ **to** i_{2e}
　　$\cdots$
for $i_n = i_{nb}$ **to** i_{ne}
　　代入文

ここで，代入文中で用いられているある変数 a を考える，なお，転置行列を表わす記号を Tとする．変数 a がある時刻 t において，ループプログラムのインデックス

$$I^t = {}^T\begin{pmatrix} i_1 & i_2 & \cdots & i_n \end{pmatrix} \tag{2.1}$$

で定義され，その次に，時刻 $t+\delta t$ にインデックス

[1]第4章で説明する．

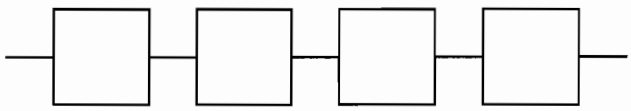


図 2.1: 線型シストリックアレー

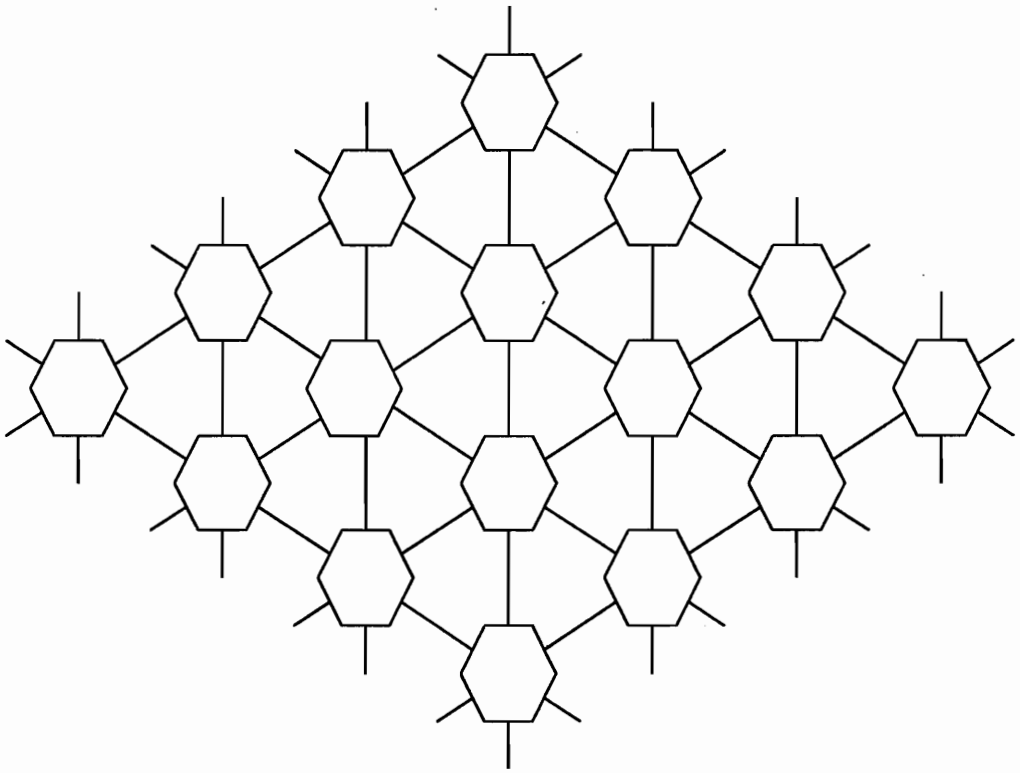


図 2.2: 六角形型シストリックアレー

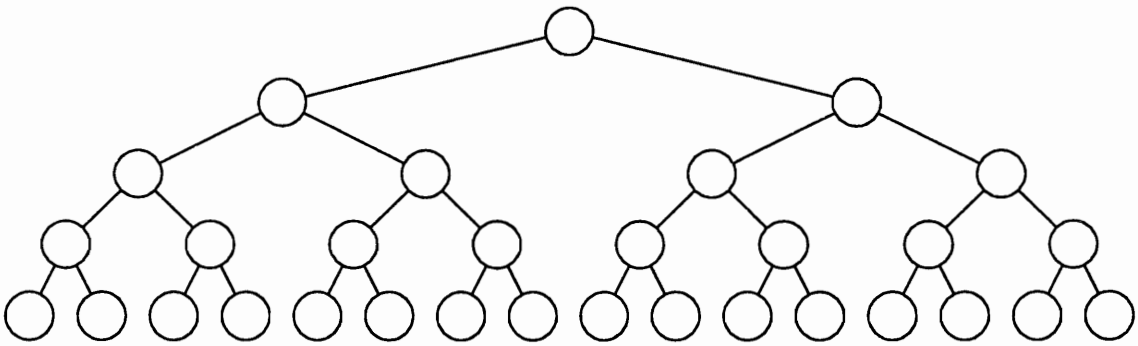


図 2.3: 木結合型シストリックアレー

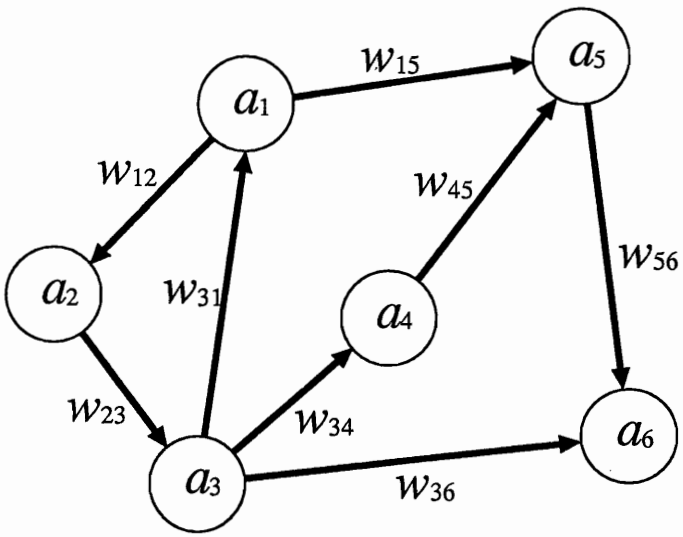


図 2.4: グラフ

$$I^{t+\delta t} = {}^T\begin{pmatrix} i'_1 & i'_2 & \cdots & i'_n \end{pmatrix} \tag{2.2}$$

で，変数 b を定義するために参照されたとする．このとき，δt の時間で，

$$\begin{aligned} I^{t+\delta t} - I^t &= {}^T\begin{pmatrix} i'_1 - i_1 & i'_2 - i_2 & \cdots & i'_n - i_n \end{pmatrix} \\ &= {}^T\begin{pmatrix} d_{ba1} & d_{ba2} & \cdots & d_{ban} \end{pmatrix} \\ &= \boldsymbol{d}_{ba} \end{aligned} \tag{2.3}$$

だけ，インデックスが変化したことになる．このときのインデックスの変化を表わすベクトル$\boldsymbol{d}_{ba}$を，変数 a から変数 b への**依存ベクトル**と呼ぶ．依存ベクトルは，並列処理アルゴリズムを扱うに当たって，データの動きを解析するのに重要な要素である．また，このようにして得られた，プログラム中のすべての依存ベクトルを 1 つにまとめて行列にしたものを**依存行列**と呼ぶ．

依存グラフ (Dependence Graph : DG) は，依存ベクトルの関係をグラフで示したものである．図 2.5は，依存グラフの例である．図 2.5では，変数 a から変数 b への依存が$\boldsymbol{d}_{ba} = {}^T\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$，変数 b から変数 e への依存が$\boldsymbol{d}_{eb} = {}^T\begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$，変数 d から変数 fへの依存が$\boldsymbol{d}_{fd} = {}^T\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$，その他の依存が${}^T\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$ であることを示している．なお，本論文の依存グラフの図では，表記を簡単にするため，依存ベクトルを転置行列で表わし，転置の記号を省略している．

2.2.2 Moldovan 法

Moldovan 法は，1983 年に Moldovan によって提案された，シストリックアレー構成法である [12]．この方法は，これまで多くの研究者たちによって提案されているほとんどの構成法の基礎となっており，この方法を改良した種々の方法が後に数多く提案されている [13–23]．

Moldovan 法は，インデックスのセットに線形変換を施し，時間座標と空間座標に変換することによって，イタレーションの計算が行われるセルの座標及び時刻を導出する．原理的には，n 次元ループを，$n-1$ 次元空間に広がるアレーに変換することができるが，実際に設計するときには，現実に実装可能な 3 次元以下に変換することがほとんどである．そのため，Moldovan 法で 5 次元以上のループプログラムを変換すると，シストリックアレーは 4 次元以上の空間に広がるものとなり，現実性に欠ける．5 次元以上の多次元ループに Moldovan 法を施して，かつ，変換アレーを 3 次元以下にする研究もされているが，多数の条件がつき，変換自体も難しいといった欠点がある [19]．

以下に，Moldovan 法の手順を示す．対象とするループプログラムを，下の n 次ループとする．

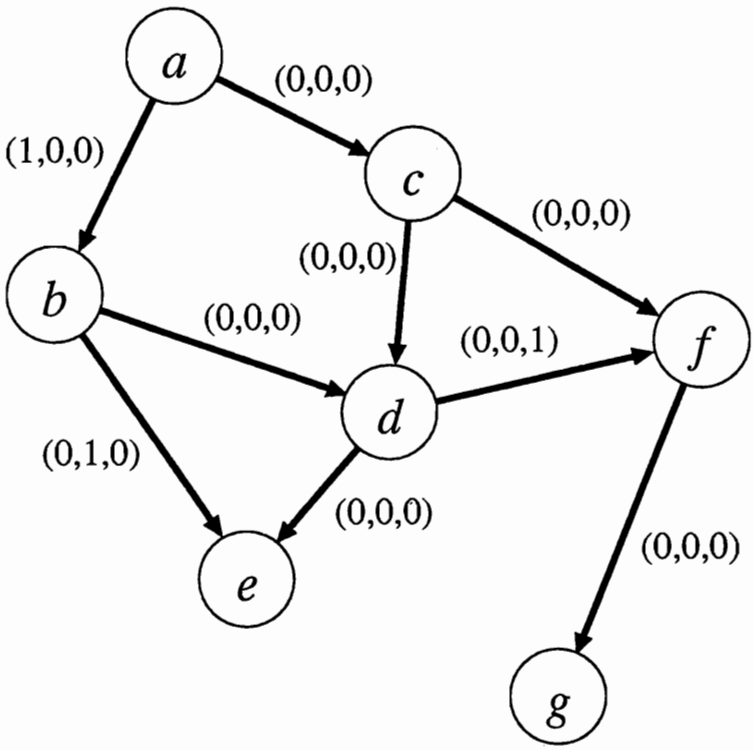


図 2.5: 依存グラフ

for $i_1 = i_{1b}$ **to** i_{1e}
for $i_2 = i_{2b}$ **to** i_{2e}
……
for $i_n = i_{nb}$ **to** i_{ne}
　代入文

このループプログラムに関して，依存ベクトル

$$\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_m \in \boldsymbol{N}^{n\times 1} \tag{2.4}$$

を導出する．なお，これらの依存ベクトルによって構成される依存行列 D を

$$D = \left[\begin{array}{cccc} \boldsymbol{d}_1 & \boldsymbol{d}_2 & \cdots & \boldsymbol{d}_m \end{array} \right] \in \boldsymbol{N}^{n\times m} \tag{2.5}$$

とおく．

このとき変換行列

$$T = \left[\begin{array}{c} \Pi \\ S \end{array} \right] = \left[\begin{array}{c} T_1 \\ T_2 \\ \vdots \\ T_n \end{array} \right] \tag{2.6}$$

に対して，

$$J = {}^T\left(\begin{array}{cccc} j_1 & j_2 & \cdots & j_n \end{array} \right) = TI \tag{2.7}$$

を定義する．ただし，Π は n 次の行ベクトル，S は $n-1$ 行 n 列の行列であり，

$$\Pi = T_1 \tag{2.8}$$

$$S = \left[\begin{array}{c} T_2 \\ T_3 \\ \vdots \\ T_n \end{array} \right] \tag{2.9}$$

を満たす．$T_1, T_2, \cdots, T_n$ は n 次の行ベクトルである．

インデックスのセットを

$$I = {}^T\left(\begin{array}{cccc} i_1 & i_2 & \cdots & i_n \end{array} \right) \tag{2.10}$$

とするとき，

$$j_1 = \Pi I \tag{2.11}$$

が計算の依存関係の時間成分を示し，

$$^T\begin{pmatrix} j_2 & j_3 & \cdots & j_n \end{pmatrix} = SI \tag{2.12}$$

が空間成分を示す．式(2.11)と式(2.12)とによって，インデックスのセットがIで表わされるイタレーションの計算が，どの座標のセルでどの時刻に行われるかが決定される．

ただし，変換行列Tは，全く任意に選択してよいわけではない．Tは線形なので，依存ベクトル$\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_m$に対して以下の式が成立する．

$$\Pi(I + \boldsymbol{d}_i) - \Pi I = \Pi \boldsymbol{d}_i = T_1 \boldsymbol{d}_i \qquad (i = 1, \cdots, m) \tag{2.13}$$

一方，この値はデータが隣接セルへ移動するのに要するステップ数を表わすので，正の整数である必要がある．ゆえに，

$$T_1 \boldsymbol{d}_i > 0 \quad \text{かつ} \quad T_1 \boldsymbol{d}_i \text{は整数} \qquad (i = 1, \cdots, m) \tag{2.14}$$

が成立するようにTを選択しなければならない．

2.2.3 一般化 Moldovan 法

Moldovan 法を一般化し，プログラムの形式的な変換によって，流れるデータ間の関係を方程式の形式で導出し，そのような値を適宜決定することによってシストリックアレーを構成する方法が提案されている [17,24]．前述の手法では，n 次元ループは$n-1$ 次元ループまでにしか変換できず，変換するとしても問題点が多く存在していた．しかし，この手法では，n 次元ループを，現実に実装可能な 3 次元以下に変換できるところが利点である．この処理の流れに沿ってプログラムを半自動的に標準形に変換し，設計者がパラメータを選定することによって，要求を満たすシストリックアレーを X-Window 上に表示するシステム SDS (Support system for Design and development of Systolic algorithms) も開発されている [24]．

この手法では，ループプログラムに以下の変換を施して標準形とする．

1. 処理内部化
2. if 文除去
3. 添え字統一

4. 流れ化変換

5. 分流・記憶処理

その後，アレー中に流れるデータ間の関係を方程式の形態で導出する．この，データ間の関係式は，空間成分，時間成分の両者に対して成立する．そこで，空間成分 (n 次元)，時間成分 (1 次元) のそれぞれにおいて，方程式を満たすようにパラメータを選定し，データの進行方向及びデータが進むのに要する時間を決定できる．たとえば，プログラム中には a，b，c の 3 つの変数があり，これらのデータ間には，

$$v_a + v_b = v_c \tag{2.15}$$

という関係が成立しているとする．ただし，v_a, v_b, v_cは，空間成分もしくは時間成分のパラメータを表わす．n 次元空間成分の各軸に関して，式 (2.15) を満たすようなパラメータ値の組合せを n 通り設定し，同じように時間成分に関しても 1 通りのパラメータ値の組合せを設定する．このパラメータによって，アレーにおける処理の進行方向と，ステップ数とが得られ，設計を行うことができる．たとえ，プログラムが n 次元ループであっても，パラメータ方程式は導出され，パラメータ値の組合せを空間成分に関して 3 通り設定すれば，3 次元のアレーに変換できることになる．

2.3 シストリックアレー設計の具体例

上述の 2 種類の方法を用いて，具体的なループプログラムをシストリックアレーに変換する．

2.3.1 Moldovan 法

以下のループプログラムは，N次正方行列$\boldsymbol{A}(= a_{ij})$ と N次正方行列$\boldsymbol{B}(= b_{ij})$ の積を求め，N次正方行列$\boldsymbol{C}(= c_{ij})$ に代入するアルゴリズムである．実際には，それぞれの行列の要素 a_{ik}，b_{kj}，c_{ij}に対して

$$c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj} \tag{2.16}$$

を計算する．式 (2.16) をループプログラムで書き表わすと，

for $i = 1$ **to** N
for $j = 1$ **to** N
for $k = 1$ **to** N
$c[i,j] = c[i,j] + a[i,k] * b[k,j]$

となり，このループプログラムの依存ベクトルを求めると，

$$\boldsymbol{d}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{d}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \boldsymbol{d}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.17}$$

が得られる．これらの依存ベクトルに対して，式 (2.14) の条件が成立するように変換行列Tを選択する．ステップ数を小さくすることを考慮して，

$$\Pi = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{2.18}$$

とし，また，空間要素行列として，

$$S = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

を選択した場合，$\boldsymbol{d}_1$によるデータ流は $S\boldsymbol{d}_1 = {}^T\begin{pmatrix} 1 & 0 \end{pmatrix}$ 方向に進み，セルからセルへの1つの区画を移動するのに必要なステップ数は$\Pi\boldsymbol{d}_1 = 1$ となる．$\boldsymbol{d}_2$，$\boldsymbol{d}_3$についても同様に考えることができるので，

- $\boldsymbol{d}_2$によるデータ流は $S\boldsymbol{d}_2 = {}^T\begin{pmatrix} -1 & 0 \end{pmatrix}$ 方向に，ステップ数$\Pi\boldsymbol{d}_2 = 1$ で移動
- $\boldsymbol{d}_3$によるデータ流は $S\boldsymbol{d}_3 = {}^T\begin{pmatrix} 0 & 1 \end{pmatrix}$ 方向に，ステップ数$\Pi\boldsymbol{d}_3 = 1$ で移動

といった結果が得られる．また，各イタレーションの計算位置セル座標及び計算時刻は，TIを計算することにより求められる．${}^T\begin{pmatrix} i & j & k \end{pmatrix}$ のイタレーションの計算時刻は，

$$\Pi \begin{pmatrix} i \\ j \\ k \end{pmatrix} = i + j + k \tag{2.20}$$

であり，計算位置セルの座標は，

$$S \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} i - j \\ k \end{pmatrix} \tag{2.21}$$

である．このときのハードウェア構成を図2.6に示す．ただし，N=4 としている．図2.6において，変数 a は$-x$ 方向に，変数 b は x 方向に，変数 c は y方向にそれぞれ進み，データ同士が衝突したセルで，各イタレーションの計算が実行される．

2.3.2 一般化 Moldovan 法

行列積計算ループに，第2.2.3節の手法を適用し，ループプログラムに変換を施すと，以下の標準形が得られる．

for $i = 1$ **to** N
for $j = 1$ **to** N
for $k = 1$ **to** N {
$a[i,k] = a[i,k]$
$b[k,j] = b[k,j]$
$c[i,j] = c[i,j] + a[i,k] * b[k,j]$
}

この標準形から，リンク候補集合を導出し，パラメータ方程式を求める．この例の場合，

$$v_a = l \quad v_b = m \quad v_c = n \quad (l, m, n \text{ は任意}) \tag{2.22}$$

が得られる．式(2.22)より，v_a，v_b，v_cは，空間成分，時間成分ともに，互いに独立に値をとることができる．そこで，以下のように，それぞれのパラメータを設定したとする．

- 空間成分の x 軸方向に対しては，$v_a = 1$，$v_b = -1$，$v_a = 0$
- 空間成分の y軸方向に対しては，$v_a = 0$，$v_b = 0$，$v_c = 1$
- 時間成分に対しては，v_a=1，$v_b = 1$，$v_c = 1$

この場合，データ流の進行方向及びセルからセルへの1つの区画を移動するのに必要なステップ数は，Moldovan 法を用いたときの例と同じである．ゆえに，構成されるシストリックアレーのハードウェア構成は図2.6と同じものになる．

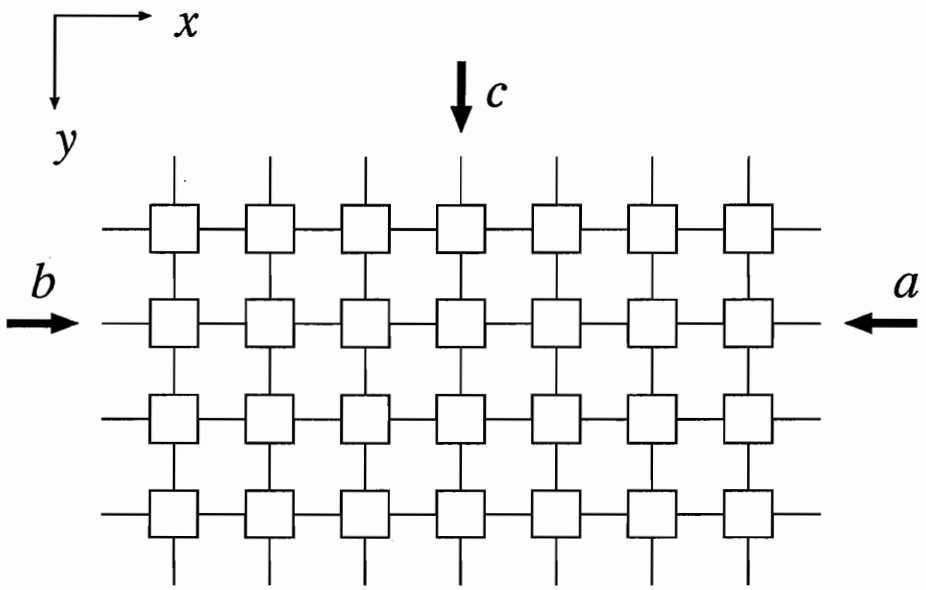


図 2.6: 行列積計算のハードウェア構成

第 3 章

シストリックアレーの効率性の評価法

3.1 はじめに

第 1 章で述べたように，シストリックアレーは 1 つのプログラムに対して複数の構成が考えられる．このことは，第 2 章で述べた 2 つの構成法に対しても言えることである．Moldovan 法での変換行列 T のとり方は一通りではないし，一般化 Moldovan 法でのパラメータのとり方も様々である．そのため，設計者は，どのシストシックアレーを採用するかを，様々な観点から評価する必要がある．

「効率的なシストリックアレーを求める」ことは，言い換えれば，「構成するシストリックアレーの消費資源をいかに小さく抑えることができるか」を考えることと等価である．本論文では，この「消費資源」を，

- 空間的資源
- 時間的資源
- その他の資源

の 3 種類に分類し，それぞれの資源に属する評価値の性質やその定義を考える．

空間的資源としては，セル数，シリコン面積，I/O ピン数などが挙げられ，言わば，「ハードウェア的なコストを反映する」資源である．また，時間的資源としては，計算時間，データ転送時間などがあり，こちらは，同様に，「ソフトウェア的なコストを反映する」資源である．

また，空間的，時間的のどちらにも属さない資源も考えられる．その例として，フォールトトレランス性が挙げられる [5]．しかし，ほとんどの評価値，資源は，空間的，ある

いは時間的な資源に属すると思われるので，本章では，この2つの場合を中心に考えることにする．

さらに本章では，評価値に対する考察により得られた知見を基に，シストリックアレーの評価に有効な融合評価関数を提案する．

3.2 評価値・指標に関する考察

3.2.1 空間的資源

空間的資源には，おもに次の種類の評価値が考えられる．

- セル数 (セル総面積)
- シリコン面積
- I/O ピン数

空間的資源の代表格としてよく用いられる評価値がセル数である．セル数はおおよそ必要なハードウェア量にも比例し，計算も容易であることから，用いられることが多い．また，単にセル数で比較するより，もっと厳密に評価値を比較したい場合には，セル数の他に，遅延素子面積，通信線の総面積なども考慮した，シリコン面積を用いるとよい．また，I/O ピンの数による比較もよく行われている．

セル数

セル数は図3.1のアルゴリズムによって計算される．ただし，セル数をm_cという変数で表わしている．セル数は，各イタレーションについてセル座標を計算し，そのセル座標が，初めて登場するセルならば，m_cの値をインクリメントして求める．

シリコン面積

シリコン面積は，計算機のハードウェアにおいて，セルや通信線など，シリコンを用いて構成する部分の面積を表わすものであり，以下の定義式で与えられる [25]．

$$S_{si} = S_c + S_d + S_n \tag{3.1}$$

式 (3.1) において，S_c, S_d, S_nは，それぞれ，セルの総面積，遅延素子の総面積，通信線の総面積を表わし，

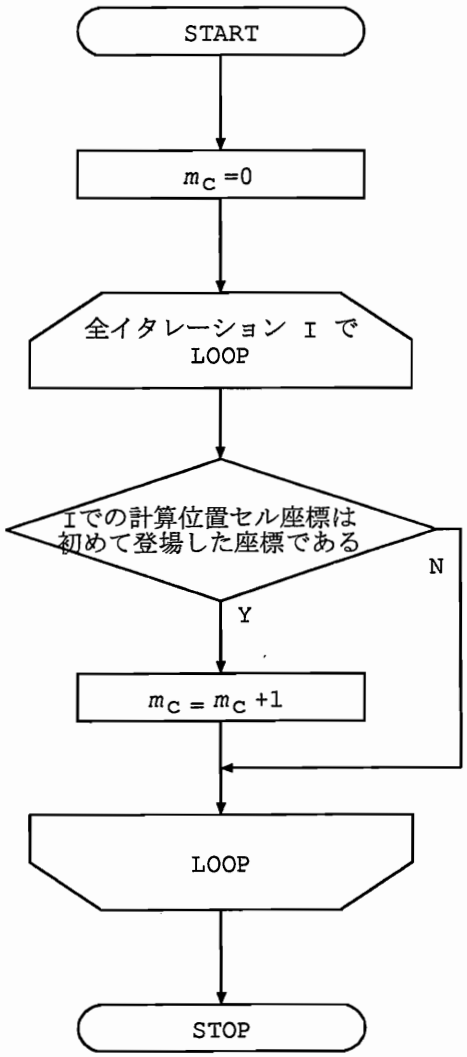


図 3.1: セル数を計算するアルゴリズム

$$S_{\mathrm{c}} = m_{\mathrm{c}} \times A_{\mathrm{c}} \tag{3.2}$$

$$S_{\mathrm{d}} = m_{\mathrm{c}} \times A_{\mathrm{d}} \times \sum_{i=1}^{m} \mid \Pi \boldsymbol{d}_i - 1 \mid \tag{3.3}$$

$$S_{\mathrm{n}} = K \times m_{\mathrm{c}} \times A_{\mathrm{L}} \tag{3.4}$$

である．ただし，m_{c}はセルの総数，A_{c}は各セルの面積，A_{d}は遅延素子の面積，m は依存ベクトルの個数，A_{L}は単位長さ当りの通信線を敷くのに要する面積を表わす．また，K は通信パターンの複雑さを示し，

$$K = \sum_{j=2}^{n} \sum_{i=1}^{m} \mid T_j \boldsymbol{d}_i \mid \tag{3.5}$$

で表わされる．

I/O ピン数

ハードウェア上のアレーには，データを入出力させるためのI/O ピンが使われる．ハードウェアを構成するとき，単一の部品の価格を考えると，プロセッサや通信回線などと比較して，このI/O ピンは特にコストが大きい．そのため，I/O ピン数 n_{io}をなるべく小さく抑えることは特に大事なことであり，n_{io}の値の大小が効率性の問題として論じられることもある．

3.2.2 時間的資源

時間的資源には，主として次の 2 種類が考えられる．

- 計算時間
- データ転送時間

計算時間は，純粋に，セルが全ての計算を処理するのに要する時間であり，データ転送時間は，全セルの現時点のステップでの計算がすべて終了してから，その次のステップに進むまでの間に，データを隣接セルへ転送するのに要する時間である．しかし，本論文では，以下のように計算時間を定義することから，データ転送時間も計算時間の概念のなかに含め，これを改めて「計算時間」と呼ぶことにする．

計算時間を以下の式で定義する．

$$t = n_{\mathrm{sys}} \times t_{\mathrm{sys}} \tag{3.6}$$

ここで，n_{sys}はステップ数である．処理が始まってから終了するまでに何ステップ要したかがこのn_{sys}で示される．また，t_{sys}はセル内 (ブロック内) 処理時間である．

また，n_{sys}は，

$$n_{sys} = n_{sys}^{in} + n_{sys}^{ex} + n_{sys}^{out} \tag{3.7}$$

と書くこともできる [26]．式 (3.7) において，n_{sys}^{in}は**データ入力ステップ数**と呼び，データをアレーに最初に入力してからどこかのセルで計算が開始するまでのステップ数を表わす．n_{sys}^{ex}を，**計算ステップ数**と呼び，アレー中のどこかのセルで計算を開始してから，すべての計算を終了するまでのステップ数を表わす．また，n_{sys}^{out}を，**データ出力ステップ数**と呼び，計算を終了してから，すべてのデータがアレーから出力されるまでのステップ数を表わす．

さらに，n_{sys}^{ex}の値は，以下の式で計算することができる．

$$n_{sys}^{ex} = \max \Pi I_1 - \min \Pi I_2 + 1 \tag{3.8}$$

ただし，I_1，I_2は，プログラム中で実行されるイタレーションのインデックスのセットである．

一方，t_{sys}は，

$$t_{sys} = \tau_c + \tau_L \tag{3.9}$$

で示される．ここで，τ_cはセル内計算時間で，純粋に計算の処理に要する時間を表わし，τ_Lはセル間のデータ転送時間を表わす．

τ_Lは以下の式で定義される．

$$\tau_L = \max_{j \leq m} \left[\sum_{i=2}^{n} \mid T_i \boldsymbol{d}_j \mid \right] \times \tau_{Le} \tag{3.10}$$

ただし，τ_{Le}は，4 近傍方向に隣接しているセル同士をつなぐ通信線を通過するデータの転送時間 (基本データ転送時間) である．

τ_Lの定義により，セル同士が隣接する方向によってτ_Lの値が変化すると言える．このことを図 3.2を用いて説明する．図 3.2において，左部分がセル同士の配線パターン，右部分が実際にハードウェア化する際の配線である．4 近傍隣接，8 近傍隣接，16 近傍隣接のそれぞれを比較すると分かるように，配線を斜め方向に伸ばす場合があるときでも，実際には斜めには敷かず，縦方向もしくは横方向のみに配線を敷くので，斜めに隣接していればそれだけτ_Lが大きくなる．したがって，τ_Lの値は，隣接セルの方向に依存していると言える．

式 (3.6) の，計算時間の定義により，シストリックアレーの処理に要する時間要素を記述できる．今までの研究では，時間に関する評価は，そのほとんどがステップ数のみを考

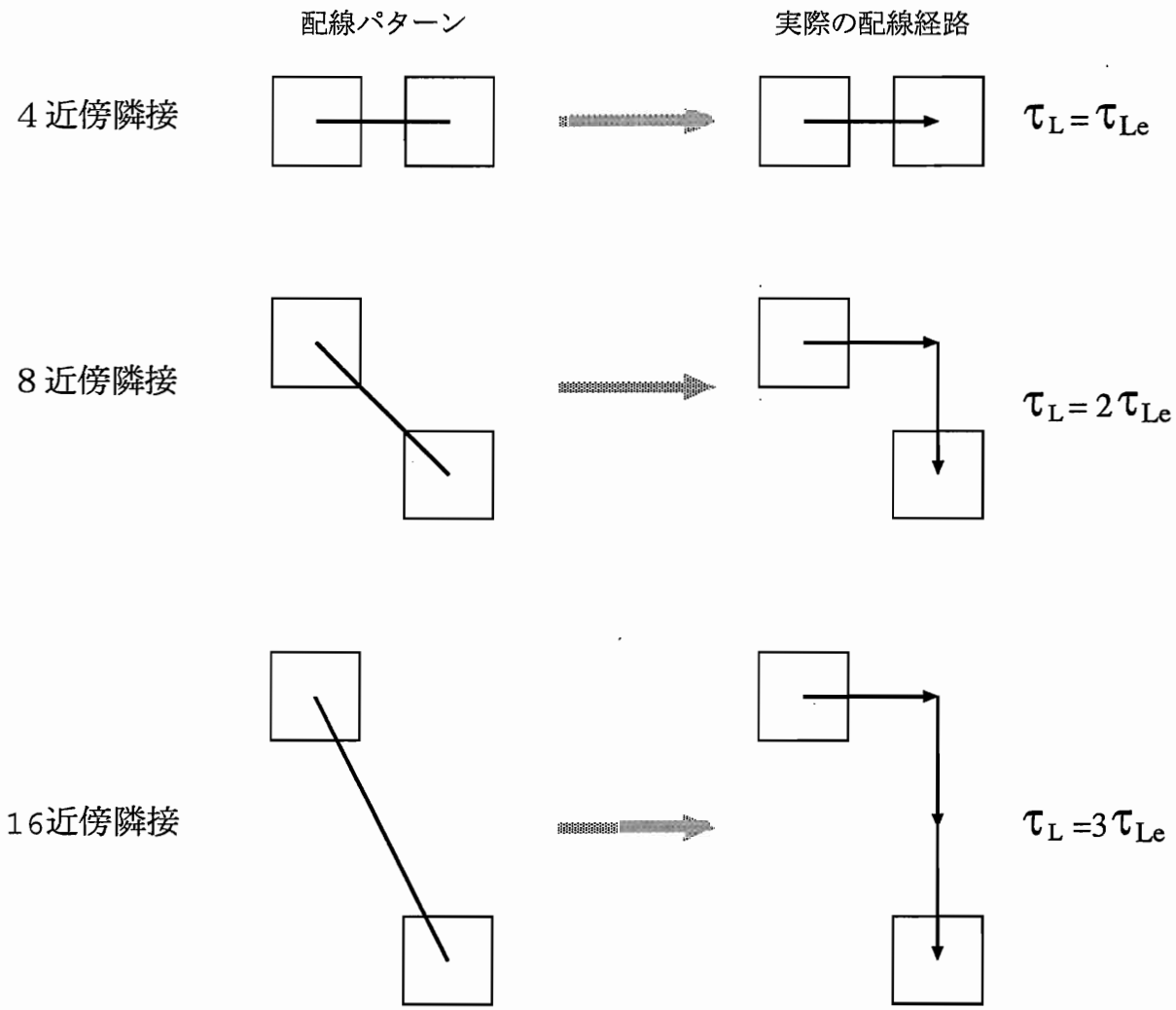


図 3.2: セルの隣接方向によるτ_Lの値の相違

慮したものであった．これは，式(3.6)のn_{sys}に相当する．ステップ数は確かに最も簡単に数えることのできる評価値であり，時間的資源の中では，最も一般的な評価値であるとも言えるだろう．しかしこの値だけでは，厳密な計算時間の大小を表わすことはできないのは明らかである．なぜなら，実際には，ステップ数は小さくても，1つのステップに要する計算処理時間(t_{sys}に関連)は大きいといったことも考えられるからである．例えば，プログラムのブロック化[1]によって複数の処理を1つのブロックでまとめ，そのブロック内の処理をすべてまとめて1ステップと数えれば，ステップ数(n_{sys})はどんどん小さくなりうるが，それに伴ってセル内での処理時間(t_{sys})は増加していくことが考えられる．このようなことから，計算時間の定義にt_{sys}の項を含めることは非常に意義の大きいことであり，また，式(3.6)の定義によって，真に時間的要素の大小を論じることができるようになる．

3.2.3 その他の資源

フォールトトレランス性

これまで述べた評価値は，空間的要素もしくは時間的要素に分類されうるものばかりであった．しかし，この二種類には分類できない評価値も考えられる．それが，フォールトトレランス性に関する評価値である．

シストリックアレーのフォールトトレランス性については，多くの研究がなされているが，評価値の面から考慮すると，**不動データ**の有無が重要となる．対象としているシストリックアレーに不動データが存在する場合は，存在しない場合と比較すると，セル数が小さくなるなど，空間的要素の面からは効率的になることも多い[26]．

しかし，その反面，フォールトトレランス性は悪化する[5]．その理由は，パイプライン的にデータが動く場合，常にそのデータの流れ方を監視することによって，故障，誤動作(フォールト)のチェックを行うことができる．一方，データが，計算処理中もセル内から外に出ずに留まっている場合，流れ方を監視することはできない．このため，別の誤動作チェック方法として，各セルに誤動作チェック機構回路を付加する必要が生じるからである．

3.3 融合評価関数

これまで．空間的資源，時間的資源の両者について，その定義や性質などを考察した．これらの評価値を設計の目的，仕様に合わせて選択して比較することによって，適切なシ

[1]第4章で詳説する．

ストリックアレーを構成することができるが，その一方，空間的資源，時間的資源の両者を融合した関数もいくつか提案されている．

本節では，空間的，時間的資源の両者を融合した総合的な評価関数についての考察を行う．

3.3.1 従来の融合評価関数

以下の式で定義される，セル総面積とステップ数の平方とをかけた値で比較する評価値は，シストリックアレーに限らず，一般的なVLSIの設計などで用いられている．

$$f_1 = S_c \times {n_{sys}}^2 \tag{3.11}$$

式(3.11)は以下のように意味付けられる．式(3.11)の両辺の平方根をとると，

$$\sqrt{f_1} = \sqrt{S_c} \times n_{sys} \tag{3.12}$$

である．ここで，$\sqrt{S_c}$はセルの辺の長さの合計を表わすので，それにステップ数n_{sys}をかけることによって，計算開始から終了までで,「最大，どれだけのデータが流れうるか」を表わせることになる．

もちろん，セル総面積S_cはセル数m_cに比例するので，式(3.11)と同様に，

$$f_1' = m_c \times {n_{sys}}^2 \tag{3.13}$$

などとしても，関数の意味付け自体は変わらない．

一方，式(3.14)で定義される評価値は，式(3.11)と似ているが，その意味付けは少し異なる．

$$f_2 = S_c \times n_{sys} \tag{3.14}$$

式(3.13)のときと同様にして，式(3.14)は，

$$f_2' = m_c \times n_{sys} \tag{3.15}$$

として考えてもよい．このとき，式(3.15)は，**並列度100%のときに実行されうるイタレーションの計算処理の個数**を表わしていると言える．ただし,「並列度100%」とは，計算開始から計算終了までのすべてのステップにおいて，すべての時刻ですべてのセルが計算に使用されている状態である．ゆえに，シストリックアレー中の全セルの使用効率r_{eff}は，

$$r_{eff} = \frac{n_{cal}}{m_c \times n_{sys}} \tag{3.16}$$

と表わせる．ただし，n_{cal}は，プログラム中の全イタレーションの計算処理回数である．以上のことから，式(3.14) の評価値は,「セルの使用効率に反比例」していると言える．

以上，式(3.11) 及び式(3.14) についての意味付けを考えた．その結果，式(3.11) は，流れうるデータの量を表わし，式(3.14) は，セルの使用効率に反比例するということを示した．このような意味付けと，式の形態から，式(3.11) は時間的資源を重視し，式(3.14) は空間的資源を重視した評価値であると言えるだろう．シストリックアレーを評価する段階となれば，そのときの設計状況や目指す仕様によって，時間的資源を重視することになる場合もあれば，空間的資源を重視しなければならないこともある．従って，上記の2つの評価値のうち，どちらの評価値を採用するかは仕様によって決まる．

3.3.2 統計量を用いた重みつき融合評価関数

従来の融合評価関数とは違い，関数に重みの変数を用いることによって，一層の「仕様による評価値の多様化」をおこなうことが可能である．そこで，文献 [26] では，式(3.17) の評価式を提案した．

$$f_3 = g_{\mathrm{s}}\frac{s_{\mathrm{ave}} - s}{\sigma_{\mathrm{s}}} + g_{\mathrm{t}}\frac{t_{\mathrm{ave}} - t}{\sigma_{\mathrm{t}}} \qquad (g_{\mathrm{s}} + g_{\mathrm{t}} = 1) \tag{3.17}$$

ここで，s, t は空間，時間それぞれにおいて考慮する評価値を表わす．文献 [26] では，s にはセル数を，t にはステップ数を用いているが，セル数，ステップ数に限らず，対象としたい評価値を式(3.17) 中に組み込んで計算してよい．また，$g_{\mathrm{s}}, g_{\mathrm{t}}$は，空間成分，または時間成分の重みを示し，$s_{\mathrm{ave}}, t_{\mathrm{ave}}$はそれらの値の平均値，$\sigma_{\mathrm{s}}$,$\sigma_{\mathrm{t}}$はそれらの値の標準偏差を示す．平均値，標準偏差といった統計量は，各々のデータ流が流れる向きのすべてのパターンに対して求める．

この評価関数では，f_3の値が大きいほど，より効率的であると言える．重み変数を評価関数中に導入するに当たっては，空間要素と時間要素とをどのように重み付けして融合させるかが重要である．空間要素と時間要素は本来単位が違うものなので，単純な加算はできない．そのため，式(3.17) では右辺第1項と第2項の単位を揃えるため，統計量を導入している．

3.3.3 新しい融合評価関数

式(3.17) は，重み変数の導入によって，シストリックアレーの設計の多様性に対応できるところが利点であるが，その反面，統計量である分散や標準偏差を用いているため，比較対象となる構成以外の，考えられうるすべてのシストリックアレーの構成に関しても，そのセル数やステップ数を計算しなければならない．データ流の個数や次元数を増や

すと，その計算量は指数関数的に増加する．こういった理由から，この評価値を採用することは，対象がごく単純なアルゴリズムであるとき以外は現実的ではない．

この問題を解決するため，統計量を用いず，かつ，重み変数を導入した新しい評価値が求められる．そこで本論文では，以下の評価値を提案する．

$$f_4 = g_s \cdot w_s \cdot s + g_t \cdot w_t \cdot t \qquad (g_s + g_t = 1) \tag{3.18}$$

ただし，w_sは，ハードウェア量が単位量分増える毎に増加する生産コストを示し，単位は[コスト/ハードウェア量] である．また，w_tは，計算時間が単位時間増える毎に増加する計算コストを示す．単位は[コスト/計算時間] である．式全体の意味は,「どれだけコストがかかるか」を表わすので，f_4の値が小さいほど，より効率的であると言える．式(3.18)は，式(3.17) のときと同様に，重みの値を生かすために加算を用い，かつ，新しくw_s，w_tを導入して，その単位をもって，空間成分，時間成分のそれぞれの単位を相殺して同一にしている．式(3.18) は，式(3.17) で用いていた統計量を用いていないので，単純に比較対象のみの評価値だけを計算すればよく，評価値を計算する際の無駄な計算をせずに済む．

3.4　評価値・評価関数の比較

本節では，2.3節での例題を用いて，これまでの評価値の計算を行う．なお，2.3節と同様に $N = 4$ とする．

イタレーションの計算セル座標や計算時刻が変わるように，変換行列として，

$$T_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad T_4 = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

を選ぶことにする．T_1によって構成されるシストリックアレーは図2.6に示した．T_2，T_3，T_4によって構成されるシストリックアレーのハードウェア構成を図3.3，図3.4，図3.5に示す．なお，T_2のシストリックアレーにおいて，変数cは不動データとして，処理過程の最中でも移動せずにセル内に留まっている． これらの3種類のアレーに対して，まず，空間的資源の評価値を計算すると，表3.1のとおりになる．ただし，セル1個当たりの面積を$A_c = 2.5 \times 10^6[\mu m^2]$, 遅延素子1個の面積を$A_d = 5.0 \times 10^4[\mu m]$，単位長さ当たりの通信線を敷くのに要する面積を$A_L = 4.8 \times 10^3[\mu m^2]$ とする．

変換行列	m_c	$S_c[\times 10^6 \mu m^2]$	$S_d[\times 10^4 \mu m]$	$S_n[\times 10^3 \mu m^2]$	$S_{si}[\times 10^6 \mu m^2]$	n_{io}
T_1	28	70	0	403.2	70.40	22
T_2	16	40	0	153.6	40.15	16
T_3	37	92.5	0	710.4	93.21	42
T_4	16	40	80	153.6	40.95	16

表 3.1: 空間的資源の評価値の計算結果

また，時間的資源の評価値及び f_1，f_2の計算結果は，表3.2のようになる．ただし，τ_c を $10[ns]$，基本データ転送時間を$\tau_{Le} = 1.7[ns]$ とする．

変換行列	n_{sys}	$\tau_L[ns]$	$t[ns]$
T_1	17	1.7	1728.9
T_2	11	1.7	1118.7
T_3	11	3.4	1137.4
T_4	20	1.7	2034.0

表 3.2: 時間的資源の計算結果

これらの計算結果から明らかなように，変換行列が T_2，T_4のときのシストリックアレーは，空間的資源の消費を小さく抑えていると言える．しかしT_4の方は，時間的資源の消費は他の場合と比較して大きい．また，T_2によるシストリックアレーと T_3によるシストリックアレーとを比較してみると，時間的資源の面では，ステップ数は同じであるが，T_2の方はデータが 4 近傍方向のみに流れているのに対し，T_3の方はデータが 8 近傍方向にも流れているため，データ転送時間τ_Lが大きくなっている．しかし，実際には，今回用いた数値データのように，データの転送時間は，セル内での処理時間と比較してかなり小さいので，τ_Lによる計算時間の差は無視できるほど小さいことが多い．

次に，同一の例を用いて，融合評価関数の評価値の計算を行う．重み変数を導入していない関数である，f_1, f_2の値はそれぞれ表3.3のとおりになる．この 2 つの関数を用いて，空間的資源，時間的資源両者を融合した評価値の計算が行える．

変換行列	$f_1[\times 10^{12}\mu m^4]$	$f_2[\times 10^6\mu m^2]$
T_1	20230	1190
T_2	4840	440
T_3	11192.5	1017.5
T_4	16000	800

表 3.3: f_1, f_2の計算結果

次に，重み付けをした評価値の計算を行う．まず，f_3の値を計算する．f_3では，空間的資源の評価値としてセル数を，時間的資源の評価値としてステップ数を式 (3.17) に適用して計算を行う．その結果を表 3.4に示す．

変換行列	$g_s = 0$	$g_s = 0.25$	$g_s = 0.5$	$g_s = 0.75$	$g_s = 1$
T_1	−0.403	−0.375	−0.348	−0.320	−0.293
T_2	1.151	1.166	1.182	1.197	1.213
T_3	1.151	0.508	−0.135	−0.779	−1.422
T_4	−1.180	−0.960	−0.736	−0.514	−0.293

表 3.4: f_3の計算結果

また，f_4の計算結果は表 3.5のとおりである．ただし，空間，時間それぞれの資源の評価値としてセル数とステップ数を考慮することとし，空間，時間要素の単位数当たりのコスト w_s，w_tは，$w_t = 1$，$w_s = 3$ であると仮定する．

変換行列	$g_s = 0$	$g_s = 0.25$	$g_s = 0.5$	$g_s = 0.75$	$g_s = 1$
T_1	51	45.25	39.5	33.75	28
T_2	33	28.75	24.5	20.25	16
T_3	33	34	35	36	37
T_4	60	49	38	27	16

表 3.5: f_4の計算結果

f_3, f_4の計算結果をグラフ化したものを図 3.6，図 3.7に示す．どちらのグラフでも，変

換行列がT_2やT_3のときのように，ステップ数の小さい構成は$g_s = 0$ではf_3の値が大きく，f_4の値は小さいので，どちらの関数も，これらの構成がより効率的であることを示している．また，T_2での構成のようにセル数が小さいものは，$g_s = 1$でのf_3の値が大きく，f_4の値が小さいので，より効率的であると言える．そして，重み変数を変化させることによって，評価値の大小の順番が入れ替わるといった現象も見られる．例えば，$g_s = 0.6$付近ではT_3はT_1よりも効率的であるが，$g_s = 0.8$付近ではT_1の方がT_3よりも効率的であるといった具合である．このことより，f_3, f_4のどちらの関数でも重み変数の効果が表われていると言える．しかし，f_3の計算に関しては，T_1からT_4の行列によるシストリックアレー以外にも構成を考えてセル数，ステップ数を計算している．今回の例では，全部で27通りの構成が考えられた．この全ての計算の終了後に統計量を計算してf_3を計算している．しかしf_4の計算においては純粋に4通りのシストリックアレーのセル数，ステップ数から評価値を求められる．このことから，f_4がf_3よりも簡単に評価値を計算でき，しかも重み変数の効果を期待できる関数であることがわかる．

3.5 むすび

シストリックアレーの評価値は，主に，空間的資源と時間的資源に分類されるが，一般的に，空間的資源と時間的資源の両者は互いにトレードオフの関係にあると言える．空間的資源を省力化して，少ないハードウェアで全計算を行おうとすると，ある時刻において，実行されるイタレーションの個数が少なくなり，ステップ数が増加して，時間的資源の消費が大きくなってしまう．また逆に，時間的資源を省力化するとなると，どうしても，ある時刻での計算イタレーションの個数を増やさなければならないので，それを賄うだけのハードウェアが必要になり，空間的資源の消費が増える．このような，お互いのトレードオフの関係が，シストリックアレー構成の効率化の障害となっていえると言えよう．

そこで本章では，まずシストリックアレーを評価する評価値や評価関数についての考察を行った．空間的資源や時間的資源に属するそれぞれの評価値について，その定義をまとめるとともに，意味付けを行った．さらに，評価関数は，シストリックアレーの設計事情の多様性に対して，重み変数を導入した関数を利用することが効果的であることを述べるとともに，従来提案されていた，重み付きの評価関数を改良し，統計量を用いない，計算の簡単な関数を提案した．また，以上の評価値・評価関数について，具体例を適用して比較を行った．

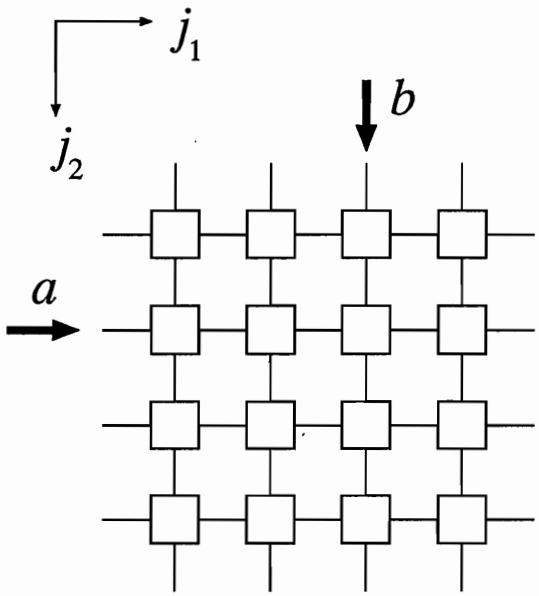


図 3.3: T_2によって構成されるシストリックアレー

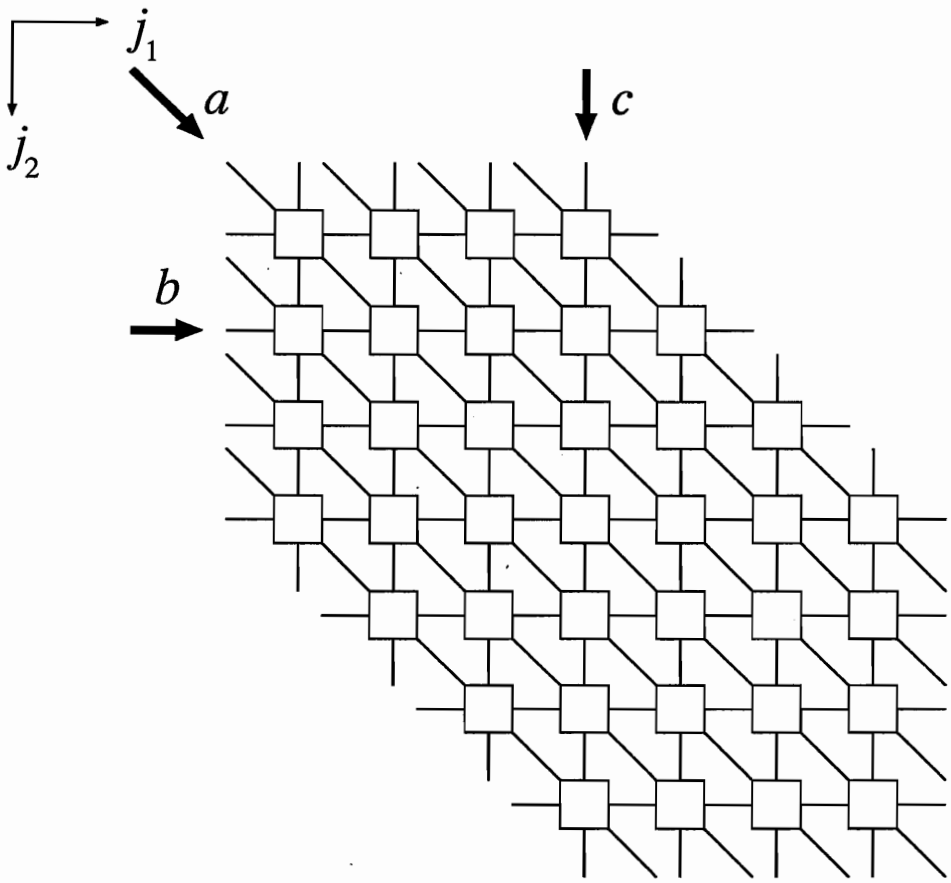


図 3.4: T_3によって構成されるシストリックアレー

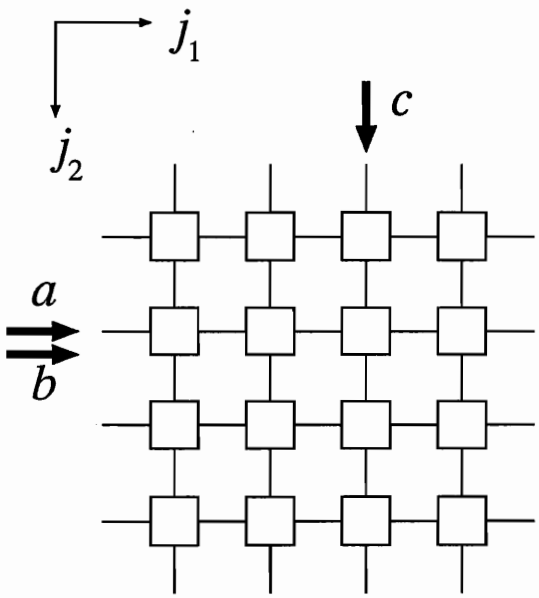


図 3.5: T_4によって構成されるシストリックアレー

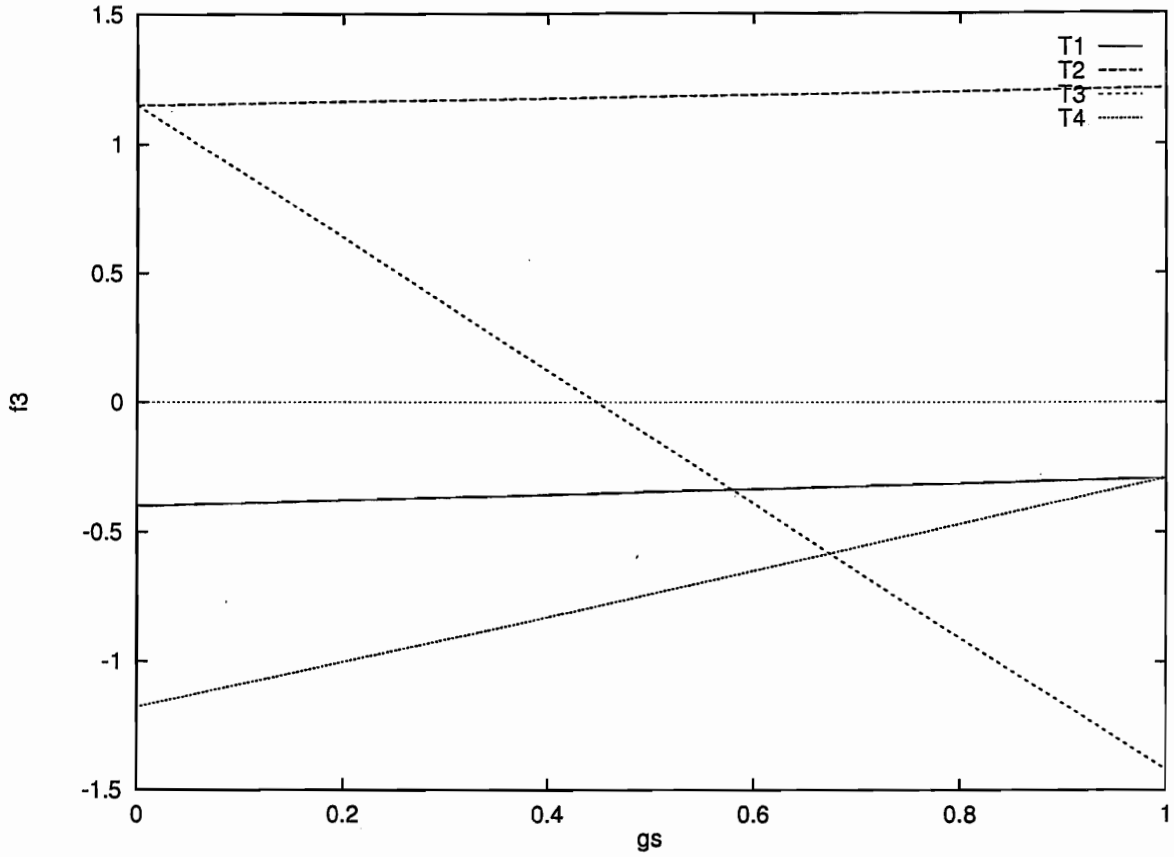


図 3.6: 重み変数を変化させたときの f_3の値の変化

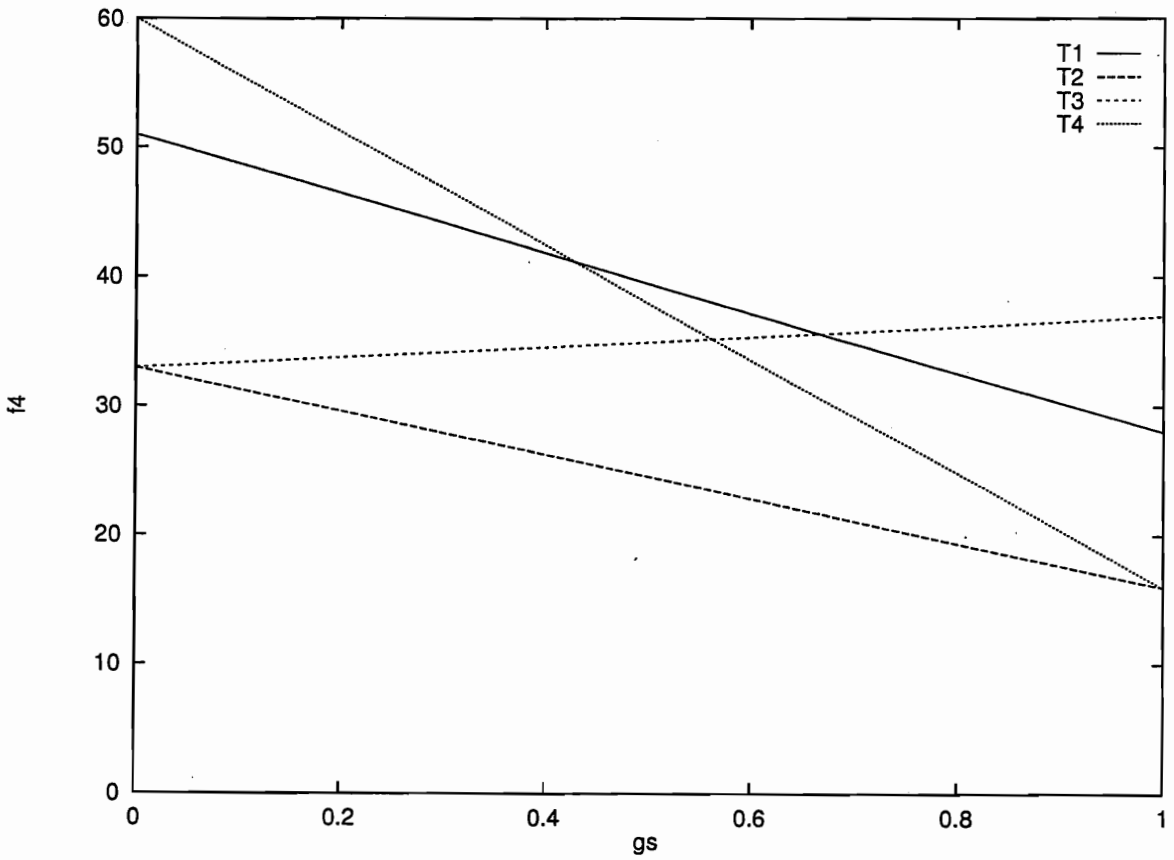


図 3.7: 重み変数を変化させたときの f_4の値の変化

第 4 章

BR 法による時間的資源の効率化

4.1 BR 法の導入

第 3 章で述べたように，消費資源は大別すると，空間的資源，時間的資源，その他の資源に分けられる．これらのそれぞれの資源は互いに密接に関連しており，3 種類の資源の消費すべてを最小に抑えられる構成が存在すれば，それは最も有効な構成と言えるであろう．

しかし，一方で，評価値同士が互いにトレードオフになることが少なくないことも第 3 章に示した．そのため，3 種類の資源の消費をすべて小さく抑えることは実際には非常に困難である．そのため，各設計段階において最も優先すべき資源を対象にして，効率化を考えることが現実的には有効であると言える．

その中でも，並列処理の場合は，まず時間的資源を第一に考慮することが必要と考えられる．ここでは，時間的資源を小さくする方法として BR 法 (Blocking and Retiming method) を提案し，その有効性を具体的な適用例を用いて示す．

4.2 retiming

retiming は文献 [25] によって提案された手法で，時間的資源を小さくする手法である．retiming は，ある程度粗粒度のアルゴリズムに対して有効であり，セル内の演算装置の配置や演算手順を再構成することによって，セル内計算時間 t_{sys} を小さくし，全体の計算時間 t も小さくするといった方法である．その方法を以下に示す．

4.2.1 レジスタグラフ

依存グラフのエッジは，データ間の依存関係 (依存ベクトル) を表わすが，このベクトルにに時間要素行列をかけたものが，データ間の遅延時間になる．例えば，依存ベクトル$\boldsymbol{d}_{ba}$に時間要素行列をかけると，データ間の時間遅延Δt_{ba}になる．依存グラフのノードに，ノード中の変数を計算するのに要する時間に関する記述をしたものは**レジスタグラフ** (Register Graph : RG) と呼ばれる．

図 4.1は，図 2.5の依存グラフに時間要素行列$T = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$をかけてレジスタグラフに変換したものである．$\Pi_a$,$\Pi_b$,$\cdots$,$\Pi_g$は，そのノード中の変数を定義するのに要する時間を表わす．また，エッジの重みは，変数間の参照を行うのに必要な遅延時間を表わす．v_1, v_2を各々の変数とするとき，δ_{v_1,v_2},Π_{v_1} は，時間を表わす変数なので，すべて整数値であり，

$$\Delta t_{v1,v2} \geq 0 \quad (v1, v2 \text{ は各々の変数}) \tag{4.1}$$

$$\Pi_{v_1} \geq 0 \quad (v_1 \text{は各々の変数}) \tag{4.2}$$

の条件を満たす．依存グラフからレジスタグラフに変換する際には，式 (4.1)，式 (4.2) の条件を満たす時間要素行列をかける必要がある．

ところで，レジスタグラフで表わされている一連の処理が，1 つのセル中で行われているとすると，t_{sys}の値をこのレジスタグラフから求めることができる．このことを図 4.1を用いて示す．今，シストリックアレーが計算を開始してから t ステップ目であると仮定する．変数 b は変数 a を参照することによって定義されるが，このとき遅延時間は 1 である．つまり，t ステップ目での変数 a は，$t+1$ ステップ目で変数 b を定義するのに用いられるので，t ステップ目のセル内の計算においてこの処理は考慮する必要がない．また，変数 c は，変数 a を参照することによって定義されるが，その処理を行なう際に必要な遅延時間は 0 である．そのため，この処理は t ステップ目で行なわなければならない．以上のことをまとめると，t ステップ目においては，zero-edge で結ばれている変数間の処理を行えばよい．

一方，c の変数を定義するには，Π_cだけの時間が必要である．同様のことをグラフ全体に対して考えると，1 つのセル内の処理を全部行うのに必要な時間は，すべての zero-path においてΠの値の和を計算したときの最大値であると言える．

なお，実際にはレジスタグラフによって値を求められるのはτ_{c} なのであるが，第 3 章で示した例のように，実際の設計においてはτ_{L}の値がτ_{c}と比較して無視できるほど小さいので，$t_{\mathrm{sys}} \simeq \tau_{\mathrm{c}}$ としてよい．そのため，本章ではレジスタグラフによってt_{sys}の値を知ることができると記述している．

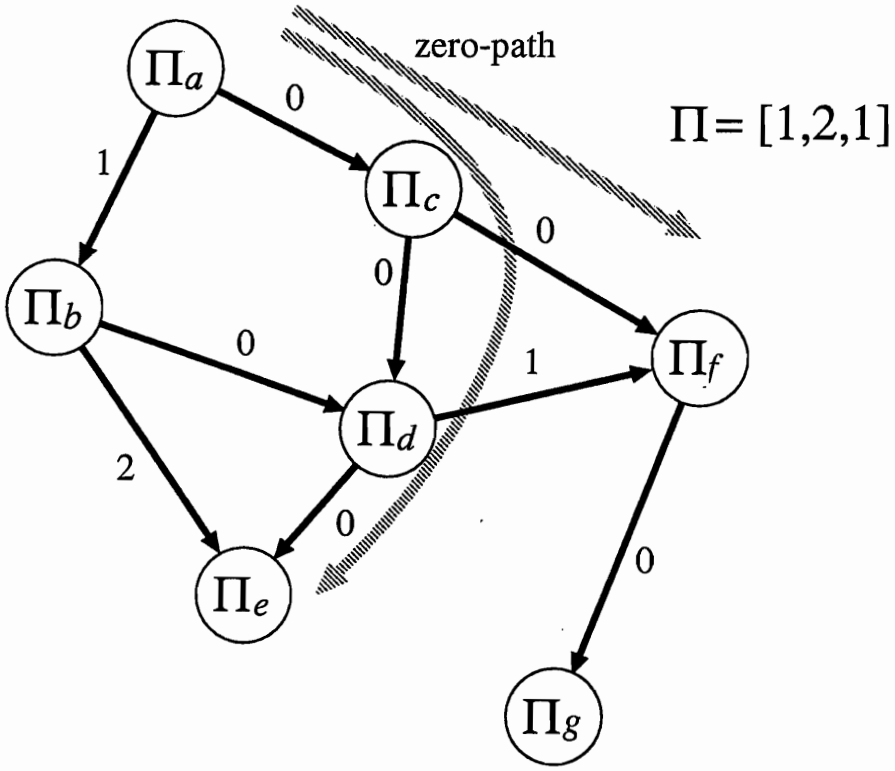


図 4.1: レジスタグラフ

4.2.2 reindexing

依存グラフ中の依存ベクトル(エッジの重み)は，プログラムの意味を変更することなしに，ベクトル要素の値を変更することができる．

ノード vからノード u への依存ベクトルを$\boldsymbol{d}_{uv}$とするとき，

$$\overline{\boldsymbol{d}_{uv}} = \boldsymbol{d}_{uv} + \delta_v - \delta_u \tag{4.3}$$

で得られる$\overline{\boldsymbol{d}_{uv}}$を，$v$から u への新たな依存ベクトルとすることができる．この変換手法を reindexing とよぶ．δ_u,δ_vは，ノード u, vに関する重みであり，**offset vector** と呼ぶ．offset vector は，

$$\delta \in \boldsymbol{N}^n \tag{4.4}$$

である．図 4.2は，図 2.5の依存グラフを，ノード bにおける offset vector を$\delta_b = (1, -1, 0)$として reindexing したあとの依存グラフである．この手法によって一部の依存ベクトルの値を変更することは，実際には，該当する変数の添え字を変更することと等価である．

4.2.3 retiming

reindexing した依存グラフに，時間要素行列をかけて，レジスタグラフに変換する操作を retiming とよぶ．レジスタグラフによって，1 つのセルまたはブロック内でのt_{sys}の値を得ることができるが，retiming はt_{sys}の値を変換前と比較して小さくなるようにすることがその目的である．図 4.3は，図 4.2の依存グラフに時間要素行列$T = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$をかけたものである．図 4.1と比較すると，ノード b の周辺のエッジの重みが変化していることが分かる．t_{sys}の値が，レジスタグラフの zero-path の，ノードの重みの和で示されることは前述した．このことから，重みが 0 のエッジの周辺を retiming して，エッジの重みを非 0 にしていけば，全体としてt_{sys}の値が小さくなると言える．

retiming は，t_{sys}の値を小さくするということから，時間要素を小さくする手法として非常に有効である．しかしその反面，この方法は，プログラムが粗粒度の場合にしか適用できないという欠点が存在する．何故なら，retiming は，1 つのセル内で実行される依存に対する変換であり，そのような変換が不可能であるほどセル内の構造が単純である場合には，もはや retiming を行うだけの余地はなく，t_{sys}を小さくできないからである．

4.3 ブロック化

前節で述べた retiming の欠点を解決する方法として，本節では**ブロック化**を提案する．

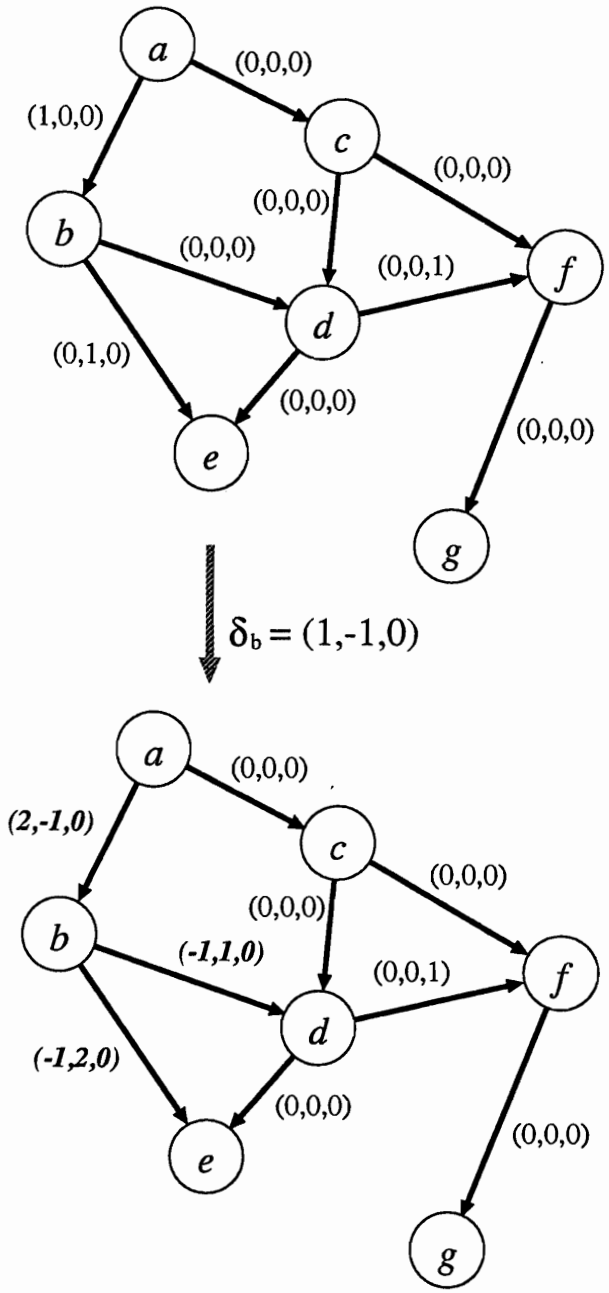


図 4.2: reindexing

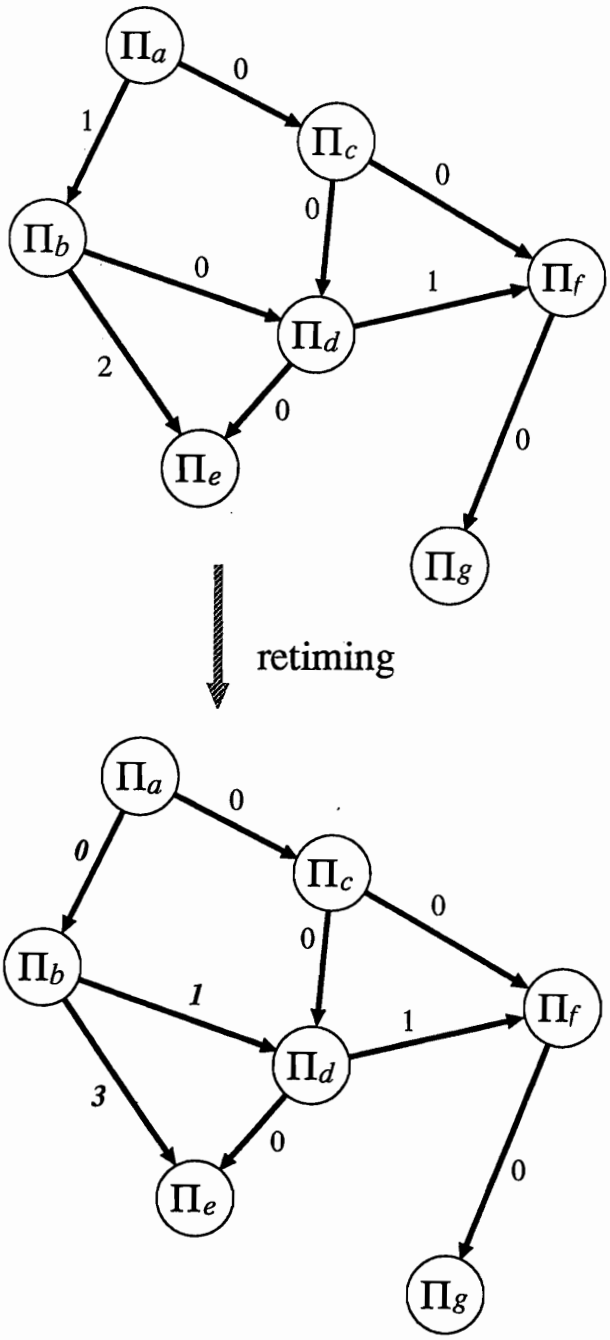


図 4.3: retiming

ブロック化は，プログラムが細粒度である場合に，retiming が適用できるように，意図的に粗粒度に変換する方法で，具体的には，いくつかのイタレーションを 1 つにまとめる変換である．細粒度のプログラム自身には retiming を行うだけの余地がないが，いくつかのイタレーションを 1 つのブロックに集めることによって，そのブロックに対して retiming を行う分の余地を与えることができる．

また，ブロック化を適用する利点がもう 1 つ存在する．それは，n_{sys}の値が小さくなることである．ブロック化により，1 つのセル内での単一の計算が，1 つのブロック内での複数のイタレーションに変換されるが，同一ブロック内で同時刻に複数の計算が行われるので，計算時間の高速化が図れる．このことは，換言すると，セル外部の並列化のみでなく，セル内部も並列化を行うために計算が高速になるということである．計算時間は n_{sys} と t_{sys}の積で表わされるので，計算時間の値が小さくなる．

ブロック化の様子を図 4.4に示す．ここでは，簡単のため，2 次元のイタレーションに関しての図を示している．図 4.4において，各黒丸は，それぞれのイタレーションを示し，網掛の部分はそれぞれのブロックを示す．ブロック化前は，各黒丸に 1 つのイタレーションが割当てられ，実行されるが，ブロック後は，各ブロックに複数のイタレーションが割り当てられ，ブロック内で同時に実行される．

ループプログラムをブロック化する様子を下に示す．

```
for i_1 = i_1b to i_1e
    ...
for i_n = i_nb to i_ne
    代入文

          ⇓

for j_1 = i_1b to i_1e by j_1s
    ...
for j_n = i_nb to i_ne by j_ns
    for i_1 = j_1 to j_1+j_1s − 1
        ...
    for i_n = j_n to j_n+j_ns − 1
        代入文
```

即ち，ブロック化前は 1 つの代入文を 1 つのセルで実行し，一方ブロック化後は，最内 n 重ループを 1 つのブロックで実行する．なお，この場合，1 つのブロック内では最大

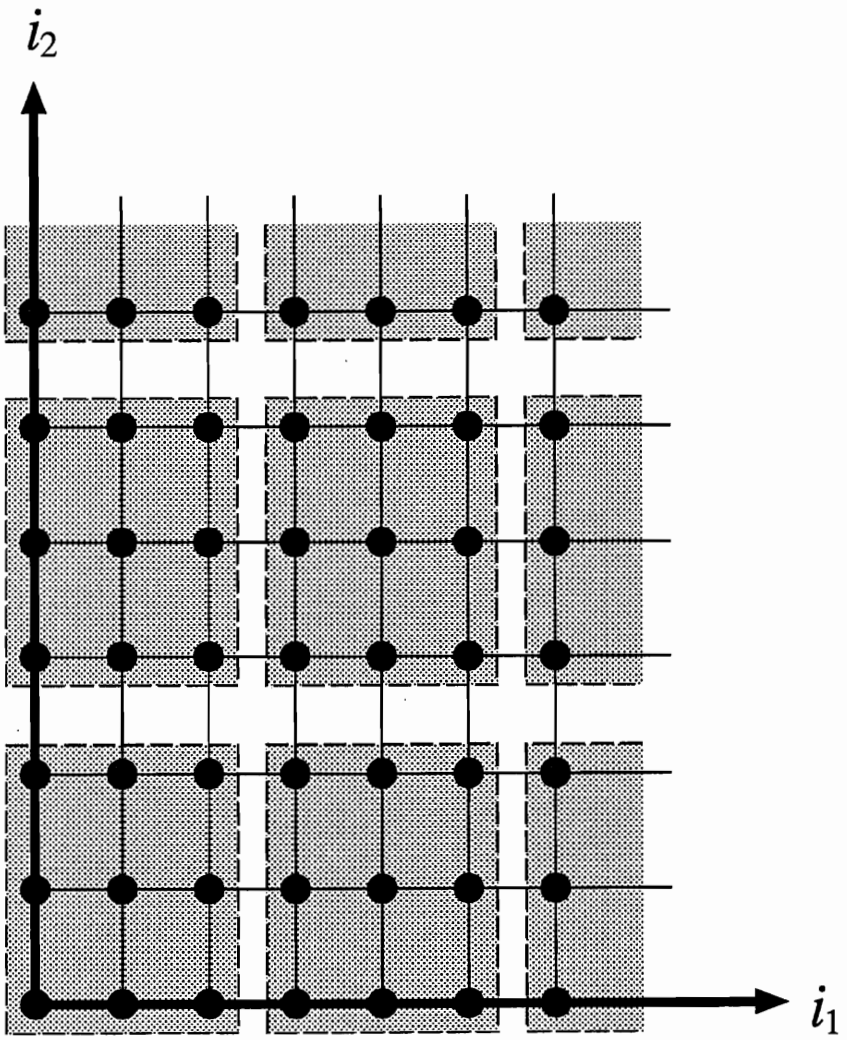


図 4.4: インデックス空間から見たブロック化

$$n_{\mathrm{i}} = j_{1s} \times j_{2s} \times \cdots \times j_{ns} \tag{4.5}$$

個のイタレーションの計算が実行されることになるが，すべてのブロックでこれだけのイタレーションが実行されるわけではない．$i_{1e} - i_{1b} + 1$ が j_{1s} で割り切れないとき，次々に並んだブロックのうち，最後の 1 個のブロックは，j_1 の値として j_{1s} 個未満の種類しか取ることができない．そのため，実際にハードウェア化する際には，余った部分には**ダミーデータ**として 0 値を流すことにする．また，このダミーデータが流れるイタレーションを**ダミーイタレーション**と呼ぶ．この考察から，この変換において生成されるブロックの個数は

$$n_{\mathrm{b}} = \left\lceil \frac{i_{1e} - i_{1b} + 1}{j_{1s}} \right\rceil \times \left\lceil \frac{i_{2e} - i_{2b} + 1}{j_{2s}} \right\rceil \times \cdots \times \left\lceil \frac{i_{ne} - i_{nb} + 1}{j_{ns}} \right\rceil \tag{4.6}$$

である．式(4.5) 及び式(4.6) から，各ブロック中のイタレーションがダミーイタレーションでない割合 r_{ud} は，

$$r_{\mathrm{ud}} = \frac{n_{\mathrm{cal}}}{n_{\mathrm{i}} \times n_{\mathrm{b}}} \tag{4.7}$$

と求められる．この r_{ud} の値が大きくなるようにブロック化を行うことは，ブロック化後のハードウェア構成の無駄を小さく抑えることと等価であり，このようなブロック化により構成されたシストリックアレーは，より効率的な構成であると言える．

ところで，ブロック化を行った後でも，シストリックアレーのイタレーションの状況や，各データの計算位置，計算時刻を求めることができるように，以下のブロック化インデックス (blocked index) を導入する．

$$\begin{aligned} I^B &= {}^T\left(i_1^B \quad i_2^B \quad \cdots \quad i_n^B \right) \\ &= {}^T\left(\left\lfloor \frac{i_1}{r_1} \right\rfloor \quad \left\lfloor \frac{i_2}{r_2} \right\rfloor \quad \cdots \quad \left\lfloor \frac{i_n}{r_n} \right\rfloor \right) \end{aligned} \tag{4.8}$$

ここで，$r_1, r_2, \cdots, r_n$ は，ブロック化によって，最外 n 重ループのインデックスが，いくつおきに値をとるようになったかを示す数字である．この数を**ブロック化係数**とよぶ．上の変換例では，ブロック化係数は，それぞれ，j_{1s}，j_{2s}，$\cdots$，j_{ns} である．

ブロック化後のシストリックアレーでの，各イタレーションの計算位置ブロック及び計算時刻は，

$$j_1^B = \Pi I^B \tag{4.9}$$

$${}^T\left(j_2^B \quad j_3^B \quad \cdots \quad j_n^B \right) = S I^B \tag{4.10}$$

で得ることができる．また，ブロック化の前後では，空間要素行列Sは変わっていないので，全体のシストリックアレーの，セル(ブロック)の配列形態は，ブロック化前と比較して，ブロック化係数の分だけ縮小したものになる．

ブロック化係数は，この値が大きいほど，ブロックの粒度も粗くなる．1つのブロック内で実行される処理の個数も増加し，粒度が大きくなるので，全体の計算が少ないステップ数で済み，n_{sys}の値も小さくなる．しかしながら，ここで注意が必要である．このようにしてどんどんブロック化係数を上げていけば，ステップ数も小さくなり，遂には，全部の処理を1つのブロックのみで行うようなシストリックアレーができるとも思われるが，実際には，粗粒度にしたことによって新たに生じる問題を考えなければならない．それは，オーバーヘッドの問題である．

大きなブロック化係数でブロック化すると，そのブロック内では，多数の計算を順番に規則正しく行わなければならないため，ブロック内に，各計算素子を統轄・管理し，制御する機構が必要となる．これは，1つのセル内で，単純でかつ少量の加減乗除演算のみを行うときにはほとんど必要とされないものである．ブロック化係数の増大に伴い，この統轄・管理機能は，巨大なものとなり，もはやそれによるオーバーヘッドが無視できない状況になるのである．

式(4.7)でのr_{ud}の値を考えて，無駄のないブロック化の条件を考えたが，これと同時に，このようなオーバーヘッドによるコストも考慮して，ブロック化係数を適切な値に設定する必要がある．

ところで，処理の粒度の問題とは別に，ブロック化の際に注意しなければならないことは，「計算の実行順序は変わっても，プログラムの意味(計算結果)は変わってはならない」ということである．たとえば，二重ループを以下のように変換したとする．

for $i_1 = i_{1b}$ **to** i_{1e}
for $i_2 = i_{2b}$ **to** i_{2e}
　　代入文

⇓

for $j_1 = i_{1b}$ **to** i_{1e} **by** j_{1s}
for $j_2 = i_{2b}$ **to** i_{2e} **by** j_{2s}
　　for $i_1 = j_1$ **to** $j_1 + j_{1s} - 1$
　　for $i_2 = j_2$ **to** $j_2 + j_{2s} - 1$
　　代入文

このとき，ブロック化の前とブロック化の後では，イタレーションの実行順序は，そ

れぞれ図 4.5，図 4.6のようになる．ただし，図 4.6は，図を簡単化するため，$i_{1e}-i_{1b}+1$，$i_{2e}-i_{2b}+1$ がそれぞれ i_{1s}，i_{2s} で割り切れるものとして，ダミーイタレーションはないものと仮定している．

図 4.5，図 4.6とを比較すると，ブロック化の前後で計算の実行順序が変わっている．そのため，ブロック化後の実行順序で計算を行っても，プログラムの意味は変換前から保存されなければならない．つまり，データ間に，順序による依存がないことが要求され，これは BR 法を適用する上で大きな障害であるかのように見える．しかし，科学技術計算において，解くべき問題は比較的単純なアルゴリズムの組み合わせである場合が多く，このような計算には順序による依存がないものが多い．例えば，行列積計算や LU 分解，コンボリューションなどの典型的な問題では，ブロック化しても意味は保存される[1]．このため，特に科学技術計算の分野においてはブロック化できる問題が多く，この BR 法は有効な手段だと言える．

4.4 BR 法の適用例

ここでは，大規模科学技術計算に用いられる，いくつかの基本的なアルゴリズムに BR 法を適用し，その有効性を示す．

科学技術計算では，より高精度な計算を行おうとすると，その問題は大規模なものになる．この傾向は今後一層顕著になるものと思われる．大規模科学技術計算において，計算時間の多くは，行列演算やベクトル演算に費やされ，この部分の計算時間をどのようにして短縮するかが高速化の鍵である．以下では，行列積計算，コンボリューションを例に，BR 法により行列，ベクトル演算の計算時間を短縮できることを示す．

4.4.1 行列積計算

まず，2.3.1節で取り上げた行列積計算を例に挙げる．

for $i = 1$ **to** N
for $j = 1$ **to** N
for $k = 1$ **to** N
$\quad c[i,j] = c[i,j] + a[i,k] * b[k,j]$

ループの依存ベクトルは，

[1]適用例は 4.4 節で説明する．

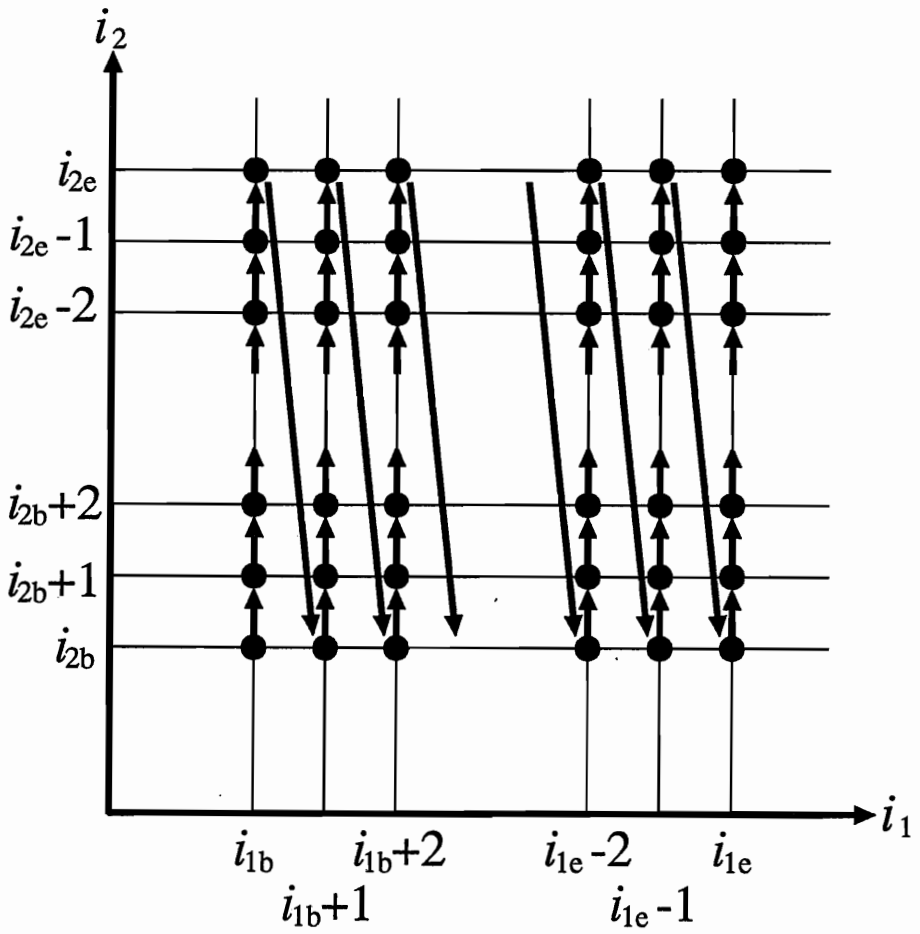


図 4.5: ブロック化前のイタレーションの実行順序

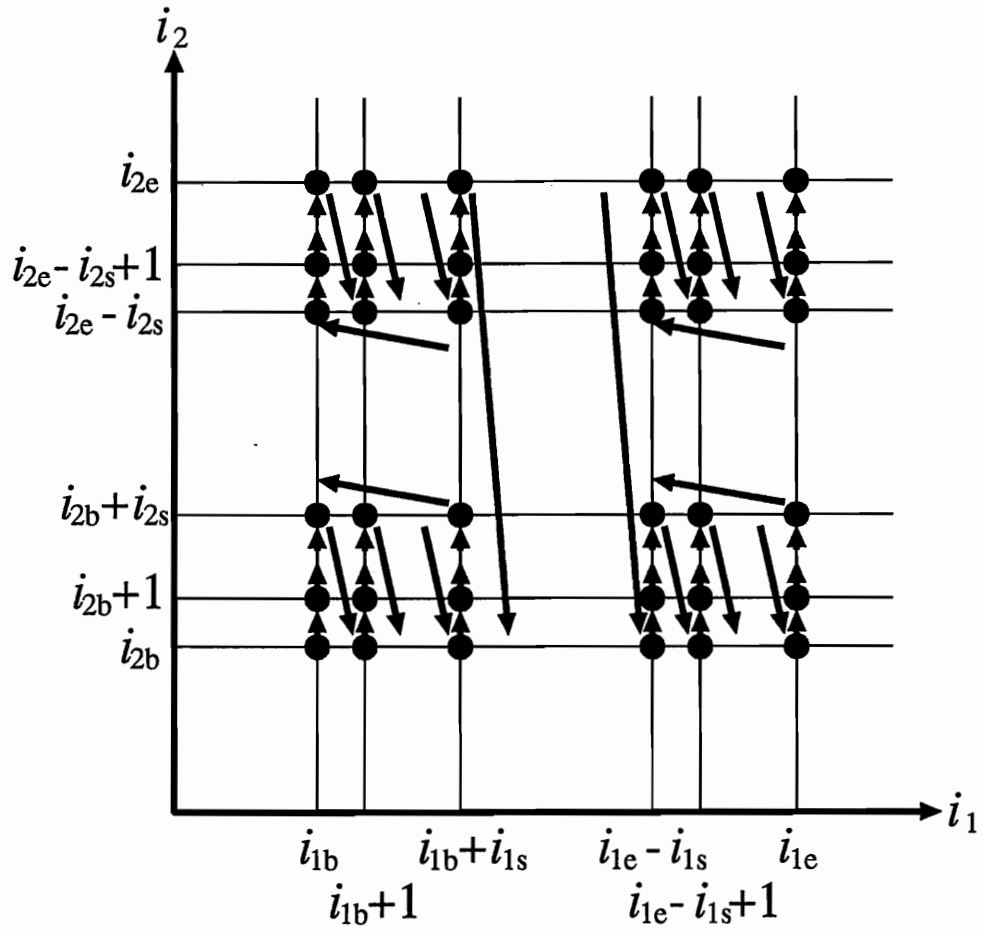


図 4.6: ブロック化後のイタレーションの実行順序

すべての記号において

$i_{1s} \leftarrow j_{1s}$, $i_{2s} \leftarrow j_{2s}$ と置換

$$\boldsymbol{d}_{ac} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \boldsymbol{d}_{bc} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{d}_{cc} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{4.11}$$

と求められる．このときの，1 つのセル内，もしくは1 つのブロック内で実行される処理の依存グラフ及びレジスタグラフを図 4.7，図 4.8に示す．ただし，時間要素行列として，

$$\Pi = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{4.12}$$

を選んでいる．

次にブロック化を施す．このループは，各インデックスのブロック化係数を $i_s = j_s = k_s = 2$ として，次のようにブロック化しても意味は等価である．

```
for i' = 1 to N by 2
for j' = 1 to N by 2
for k' = 1 to N by 2
    for i = i' to i' + 1
    for j = j' to j' + 1
    for k = k' to k' + 1
        c[i,j] = c[i,j] + a[i,k] * b[k,j]
```

このようにブロック化したときのブロック内の依存グラフ及びレジスタグラフを図 4.9，図 4.10に示す．更に retiming をおこなう．retiming は，図 4.10 の依存グラフの，zero-edge を，nonzero-edge に変換する方針で行う．図 4.9の依存グラフで，retiming をおこなった結果を図 4.11，図 4.12に示す．ただし，ここでは，セル内 (ブロック内) において，加減算に 1，乗除算に 5 の時間を要すると仮定している．また，図 4.8，図 4.10，図 4.12のレジスタグラフでは，細線のエッジ重みが 0，太線のエッジ重みが 0 以外であることを示す．

以上の処理から，n_{sys}及びt_{sys}の値を求めると，それぞれ表 4.1のようになる．ただし，空間要素行列 Sを

$$S = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.13}$$

と置いた．

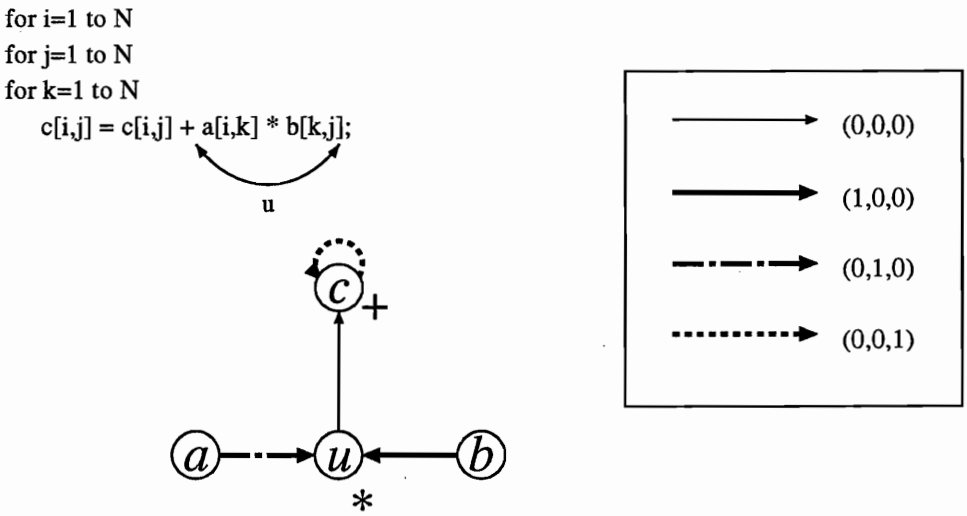


図 4.7: ブロック化前のセル内依存グラフ (行列積)

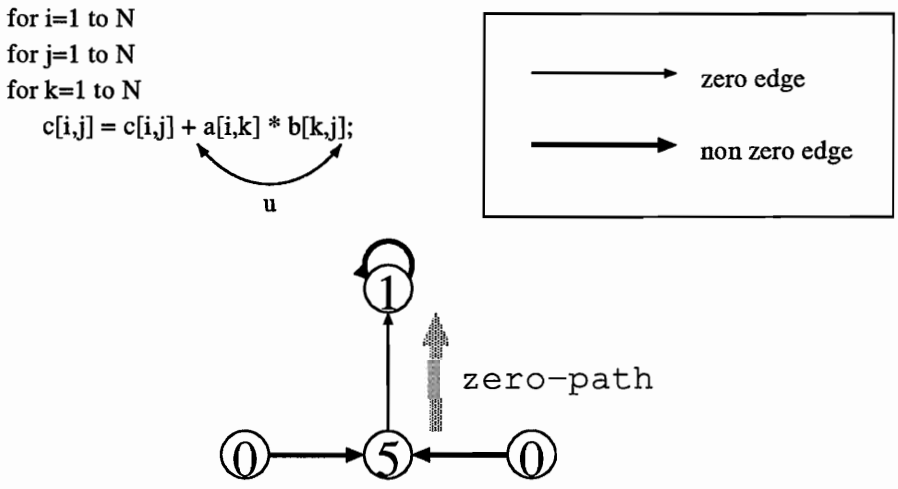
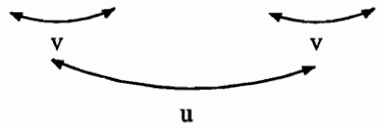
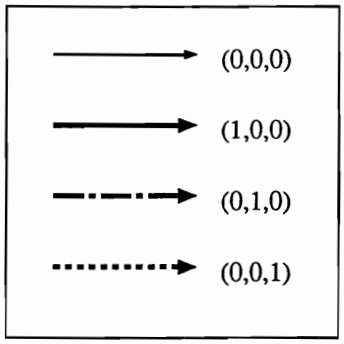


図 4.8: ブロック化前のセル内レジスタグラフ (行列積)

```
for i=1 to N by 2
for j=1 to N by 2
for k=1 to N by 2
  c[i,j] = c[i,j] + a[i,k] * b[k,j] +a[i,k+1] * b[k+1,j];
  c[i+1,j] = c[i+1,j] + a[i+1,k] * b[k,j] + a[i+1,k+1] * b[k+1,j];
  c[i,j+1] = c[i,j+1] + a[i,k] * b[k,j+1] + a[i,k+1] * b[k+1,j+1];
  c[i+1,j+1] = c[i+1,j+1] + a[i+1,k] * b[k,j+1] + a[i+1,k+1] * b[k+1,j+1];
```





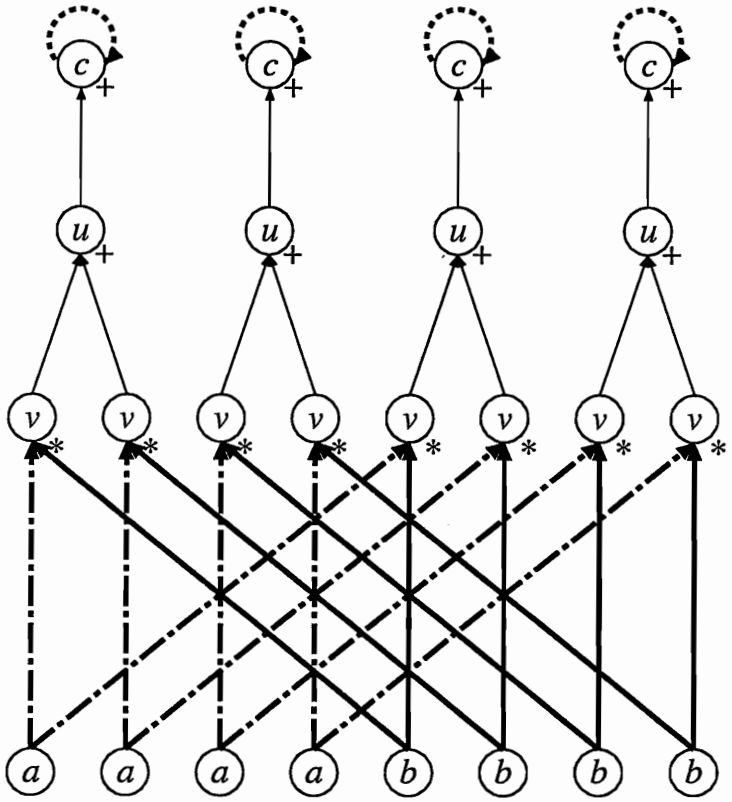
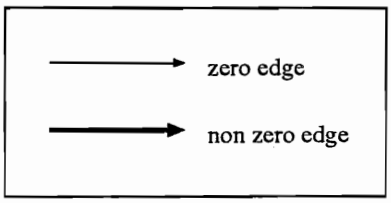
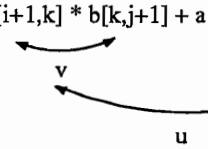


図 4.9: ブロック化後のブロック内依存グラフ (行列積)

```
for i=1 to N by 2
for j=1 to N by 2
for k=1 to N by 2
  c[i,j] = c[i,j] + a[i,k] * b[k,j] +a[i,k+1] * b[k+1,j];
  c[i+1,j] = c[i+1,j] + a[i+1,k] * b[k,j] + a[i+1,k+1] * b[k+1,j];
  c[i,j+1] = c[i,j+1] + a[i,k] * b[k,j+1] + a[i,k+1] * b[k+1,j+1];
  c[i+1,j+1] = c[i+1,j+1] + a[i+1,k] * b[k,j+1] + a[i+1,k+1] * b[k+1,j+1];
```





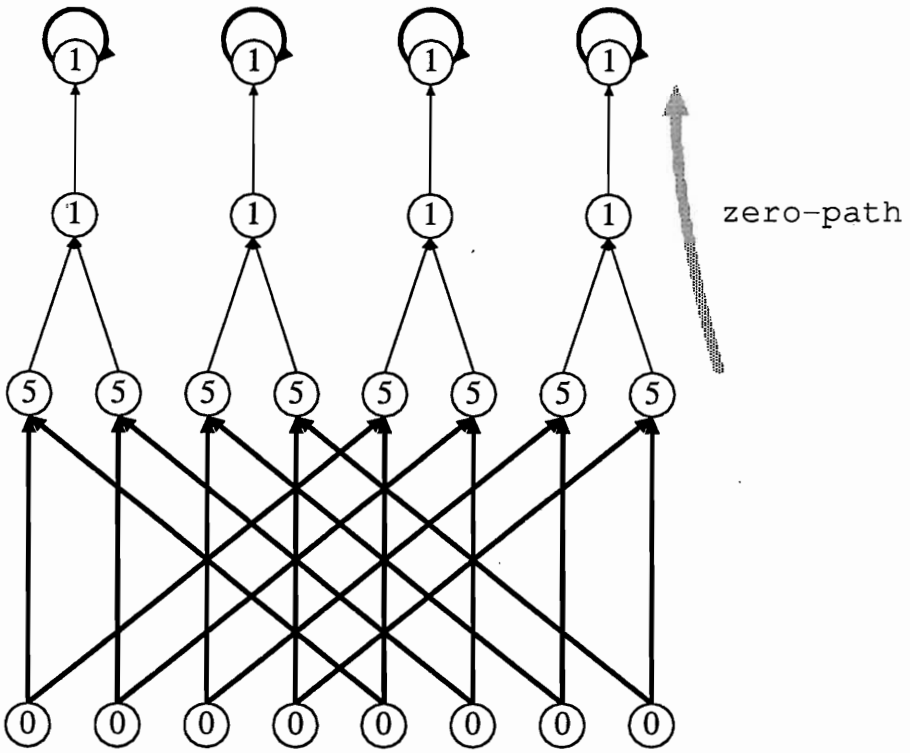
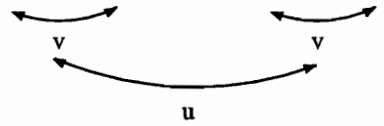
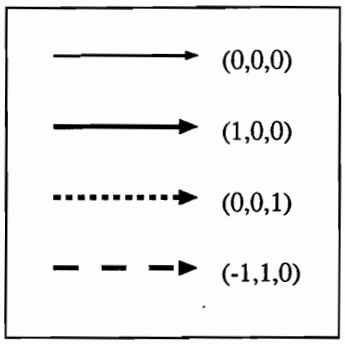


図 4.10: ブロック化後のブロック内レジスタグラフ (行列積)

```
for i=1 to N by 2
for j=1 to N by 2
for k=1 to N by 2
  c[i,j] = c[i,j] + a[i,k] * b[k,j] +a[i,k+1] * b[k+1,j];
  c[i+1,j] = c[i+1,j] + a[i+1,k] * b[k,j] + a[i+1,k+1] * b[k+1,j];
  c[i,j+1] = c[i,j+1] + a[i,k] * b[k,j+1] + a[i,k+1] * b[k+1,j+1];
  c[i+1,j+1] = c[i+1,j+1] + a[i+1,k] * b[k,j+1] + a[i+1,k+1] * b[k+1,j+1];
```





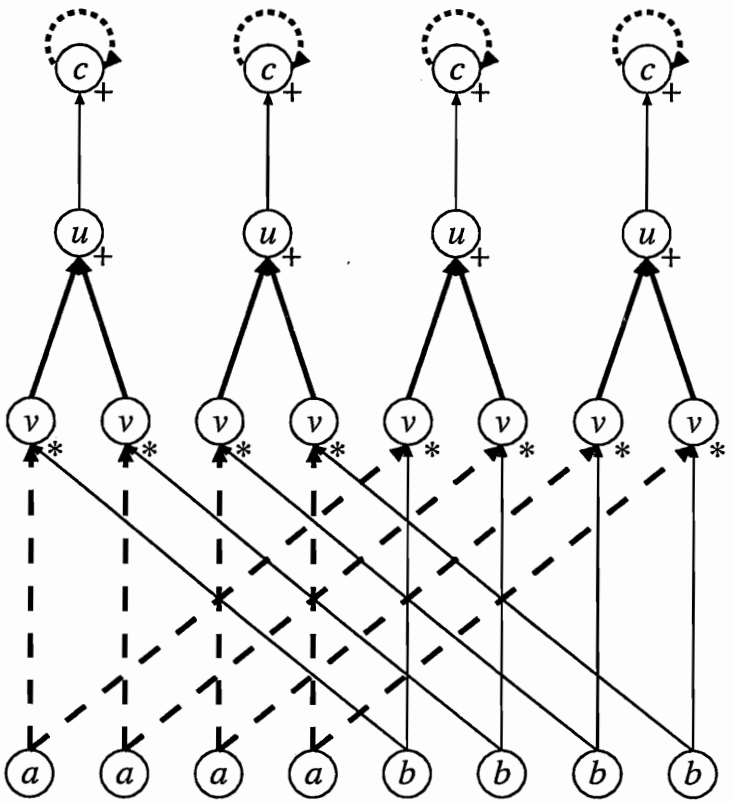
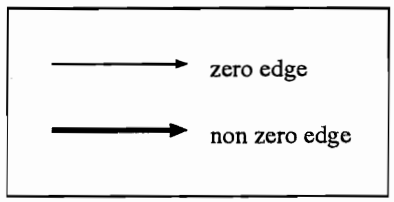
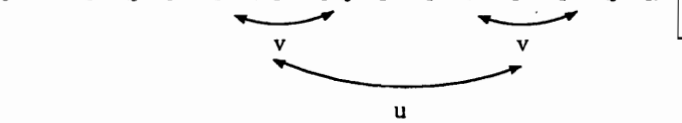


図 4.11: retiming 後のブロック内依存グラフ (行列積)

```
for i=1 to N by 2
for j=1 to N by 2
for k=1 to N by 2
  c[i,j] = c[i,j] + a[i,k] * b[k,j] +a[i,k+1] * b[k+1,j];
  c[i+1,j] = c[i+1,j] + a[i+1,k] * b[k,j] + a[i+1,k+1] * b[k+1,j];
  c[i,j+1] = c[i,j+1] + a[i,k] * b[k,j+1] + a[i,k+1] * b[k+1,j+1];
  c[i+1,j+1] = c[i+1,j+1] + a[i+1,k] * b[k,j+1] + a[i+1,k+1] * b[k+1,j+1];
```





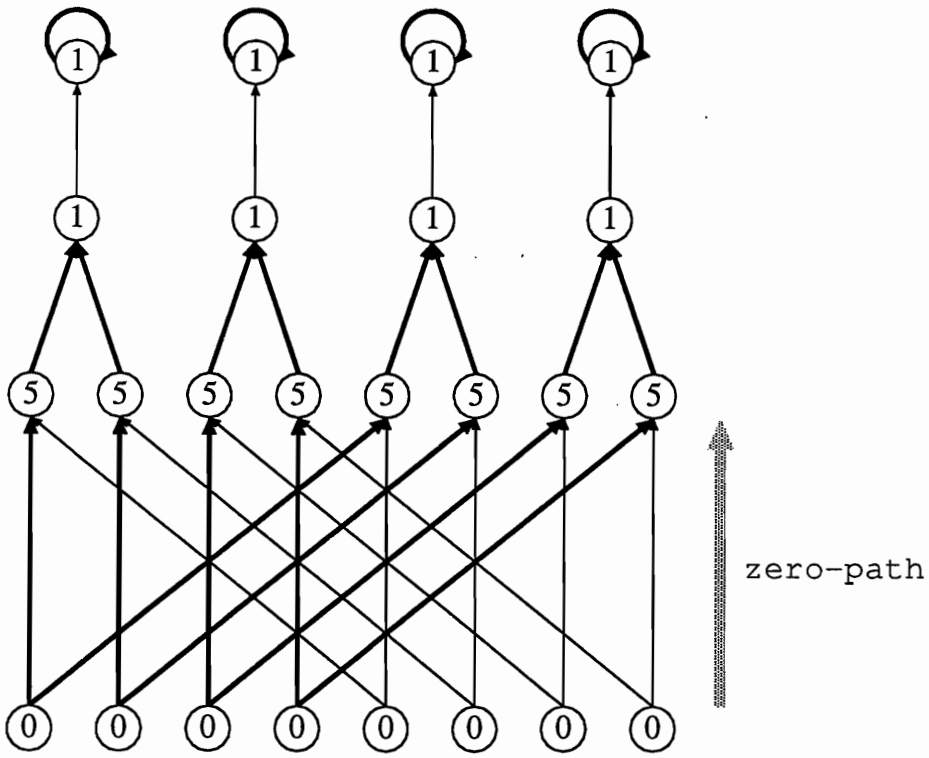


図 4.12: retiming 後のブロック内レジスタグラフ (行列積)

	original	blocked	blocked&retimed
n_{sys}	20	8	8
t_{sys}	6	7	5
$n_{\mathrm{sys}} \times t_{\mathrm{sys}}$	120	56	40

表 4.1: 行列積計算アルゴリズムの各計算時間

BR 法を適用しないで構成したシストリックアレー (original) と比較すると，ブロック化のみを行って構成したシストリックアレー (blocked) は計算時間が半分以下に減少し，また，ブロック化と retiming の両者を行って構成したシストリックアレー (blocked& retimed) は，3 分の 1 の計算時間で済むことがわかる．

ブロック化前後のハードウェア構成を図 4.13，図 4.14に示す．ただし，$N = 4$ とした．基本的にブロック化前と比較して，ブロック化後はブロック化係数の分だけ縮小したものとなる．ブロック内部では，複数のイタレーションの処理を行い，かつ，それらのイタレーションを管理するため，普通のセルよりも構成は複雑である．

次に，この結果に，第 3 章で定義した融合評価関数を適用して評価をする．空間的資源の評価値としては，ブロック化を行ったために単純にセル数を比較することはできないので，セル (ブロック) 面積で評価する．また，時間的資源の評価値には計算時間を当てはめる．ブロック化により，1 つのブロック内では最大 4 個のイタレーションの計算が行われるが，計算処理を制御するユニットがブロック内に存在するので，シリコン面積は，セルを 1 とし，ブロックを 4.5 と仮定する．$w_{\mathrm{t}} = 1, w_{\mathrm{s}} = 3$ として f_4を計算すると，図 4.15を得る．このグラフより，g_{s}の値が小さいほど，つまり g_{t}の値が大きいときほど，BR 法を施さない構成と，ブロック化を施した構成，あるいはブロック化と retiming を施した構成との評価値の差が大きくなり，BR 法がより有効であることがわかる．

次に，Nの値及び，ブロック化係数を変化させたときの計算時間の動きを考える．簡単のため，各インデックスでのブロック化係数は $i_s = j_s = k_s = L$ と置く．L は正の整数である．

各イタレーションの反復回数は$\left\lceil \frac{N}{L} \right\rceil$ で表わされる．n_{sys}の値は，この値に比例し，

$$n_{\mathrm{sys}} = 6 \left\lceil \frac{N}{L} \right\rceil - 4 \tag{4.14}$$

と表わされる．一方，t_{sys}の値は，1 つの c の変数に足されるべき a と b との積の結果は，ブロック化係数の個数分だけ存在するので，セル内計算時間は乗算 1 回分と加算 L 回分だけ必要になる．このことから，ブロック化係数の大小が t_{sys}に与える影響は，加算の回数だけであり，

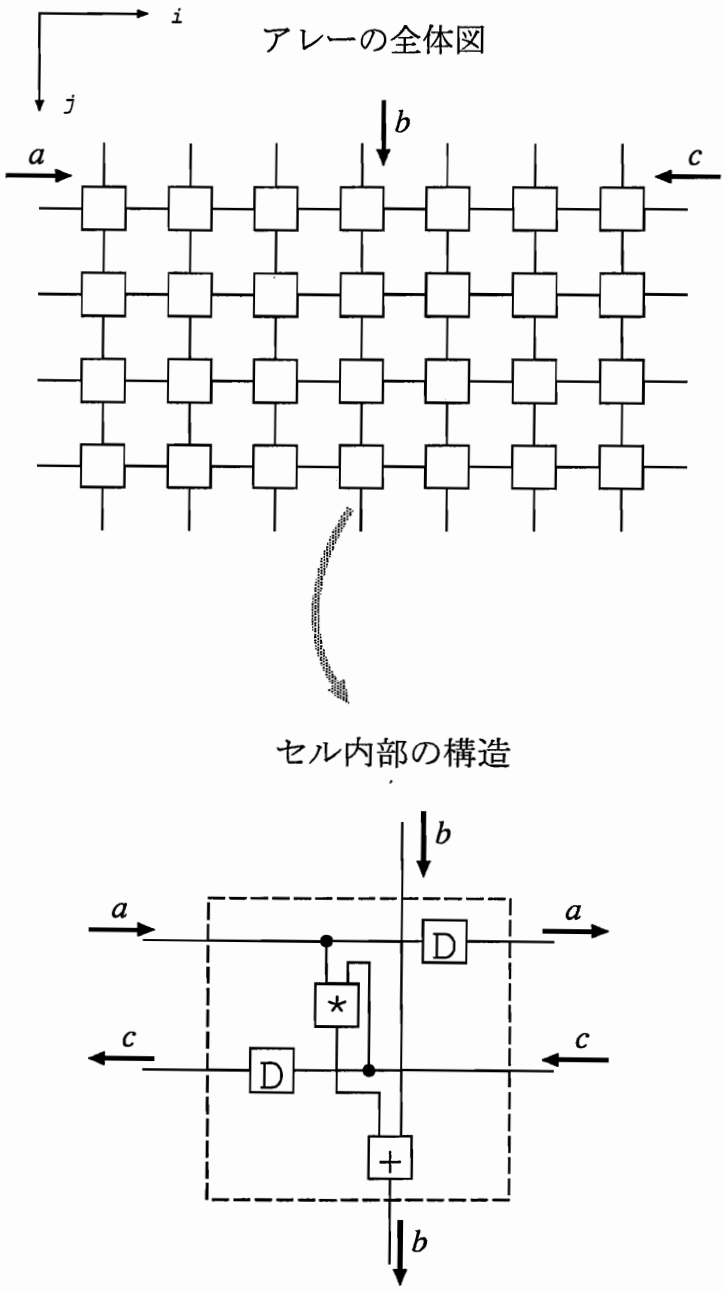


図 4.13: ブロック化前のハードウェア構成 (行列積)

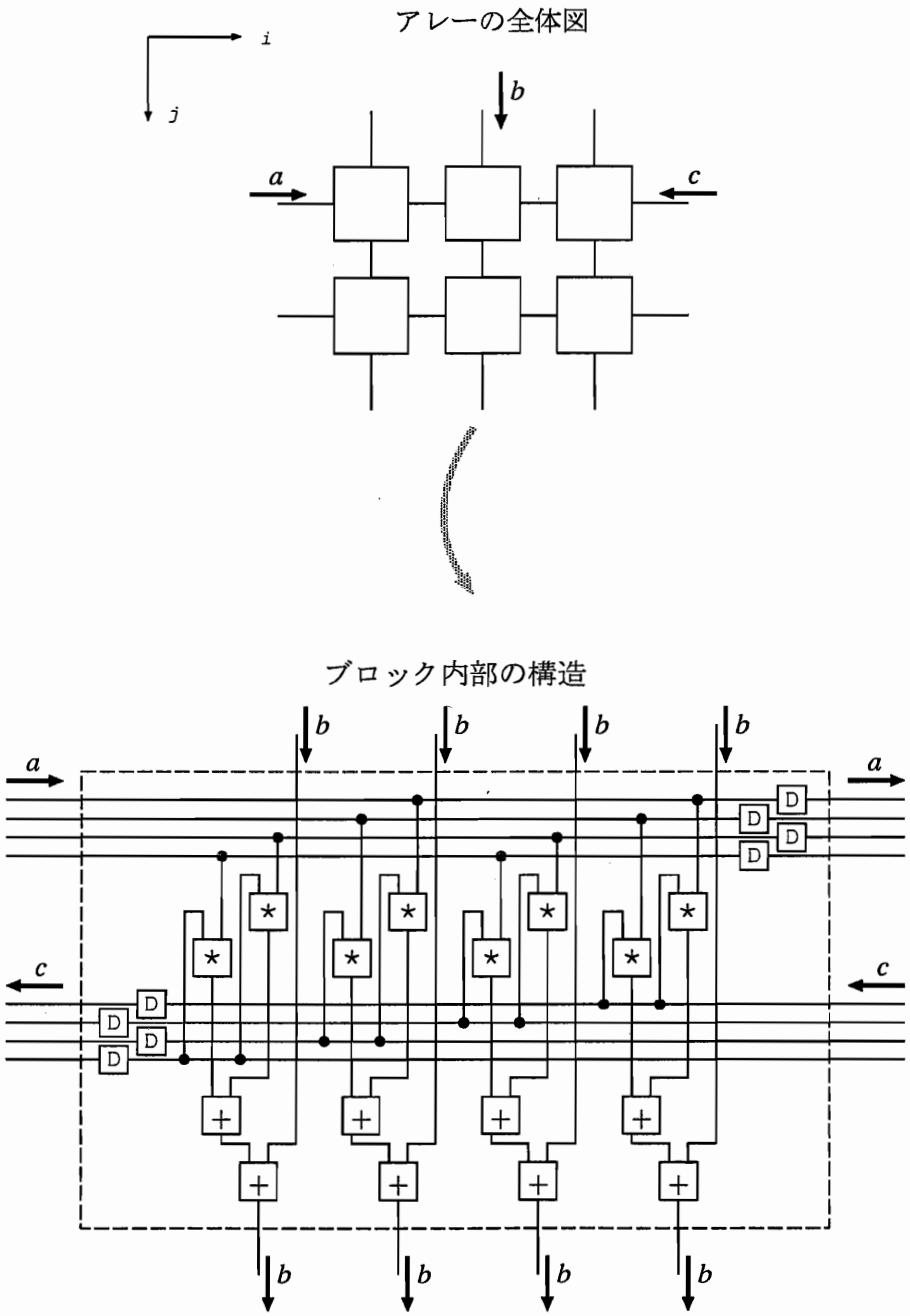


図 4.14: ブロック化後のハードウェア構成 (行列積)

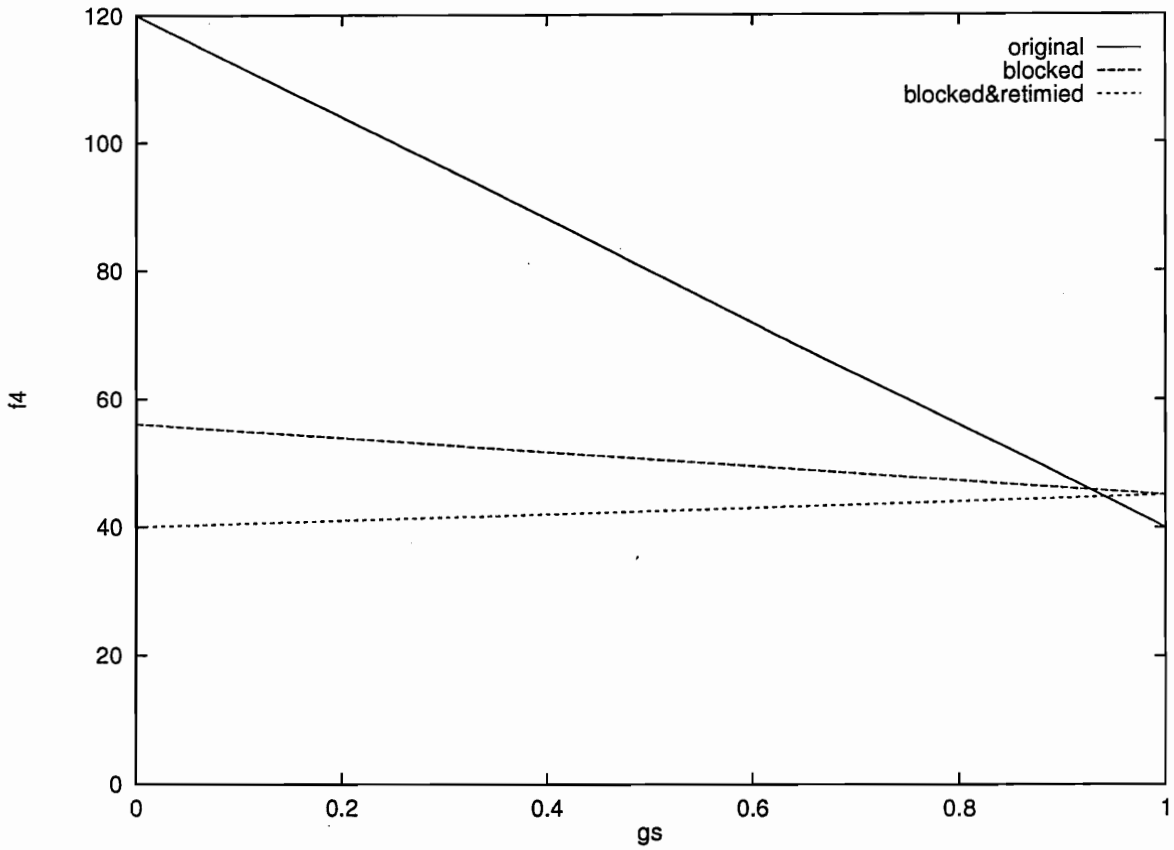


図 4.15: 行列積計算アルゴリズムの f_4の値

$$t_{\mathrm{sys}} = L + 5 \tag{4.15}$$

となる．式(4.14) と式(4.15) より，L と Nを変化させたときの計算時間 t の変化は図 4.16，図 4.17のようになる．ブロック化の効果はグラフよりわかるが，図 4.16より，Nの値が小さいときは，ブロック化係数を上げても一概に計算時間が小さくなるとは限らない．図 4.17より Nの値が大きくなると，次第にブロック化係数の影響が大きくなる．

4.4.2 コンボリューション

次にコンボリューションを例として取り上げる．

$$c_i = \sum_{j=0}^{i} a_{i-j} b_j \tag{4.16}$$

```
for i = 0 to n
for j = 0 to i
    c[i] = c[i] + a[i − j] * b[j]
```

ループの依存ベクトルは

$$\boldsymbol{d}_{ac} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \quad \boldsymbol{d}_{bc} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \boldsymbol{d}_{cc} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{4.17}$$

となる．このときの，1 つのセル内，またはブロック内で実行される処理の依存グラフ及びレジスタグラフをぞれぞれ図 4.18，図 4.19に示す．ただし，時間要素行列として，

$$\Pi = \begin{bmatrix} 2 & 1 \end{bmatrix} \tag{4.18}$$

を選んでいる．

次にブロック化を施す．このループは，下のようにブロック化しても，意味は等価である．

```
for i' = 0 to n by 2
for j' = 0 to i' by 2
    for i = i' to i' + 1
    for j = j' to j' + 1
        c[i] = c[i] + a[i − j] * b[j]
```

このようにブロック化したときのブロック内の依存グラフ，レジスタグラフは図 4.20，図 4.21のようになる．

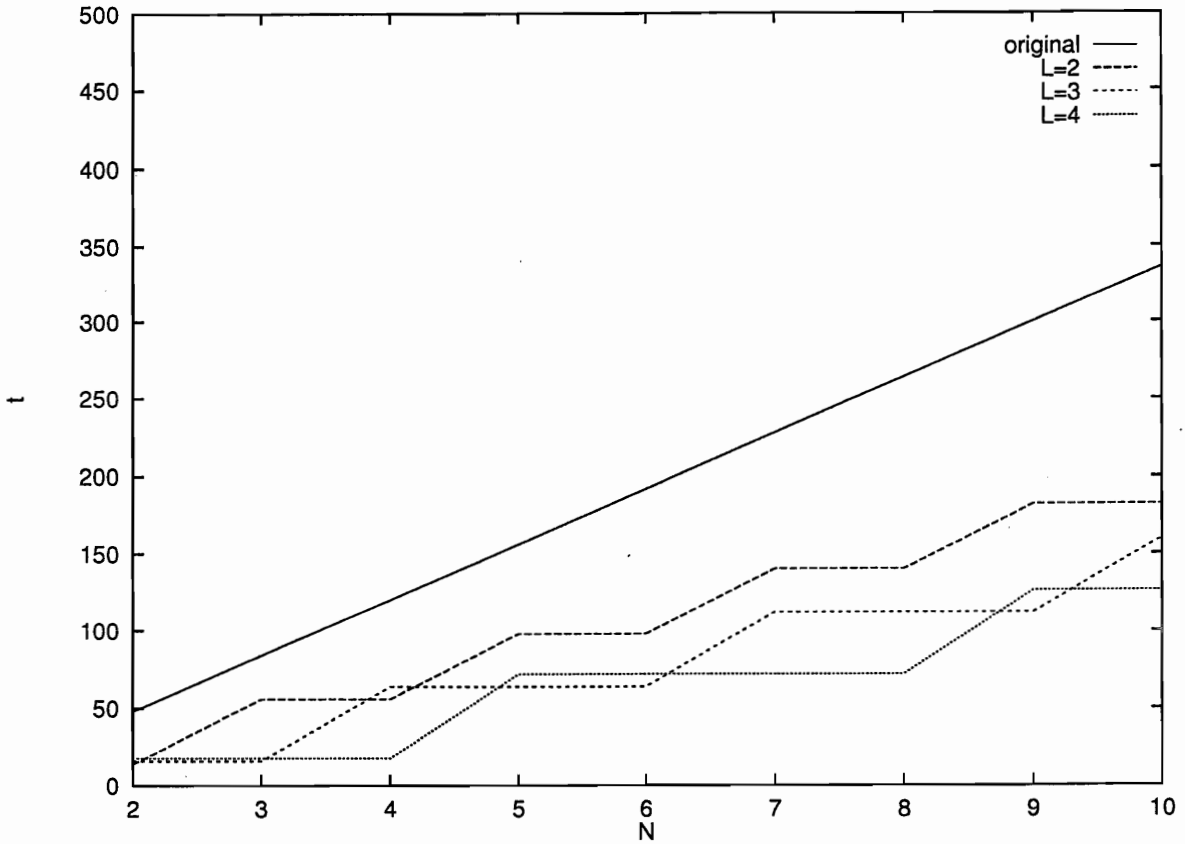


図 4.16: L と Nを変化させたときの t の動き (1) (行列積)

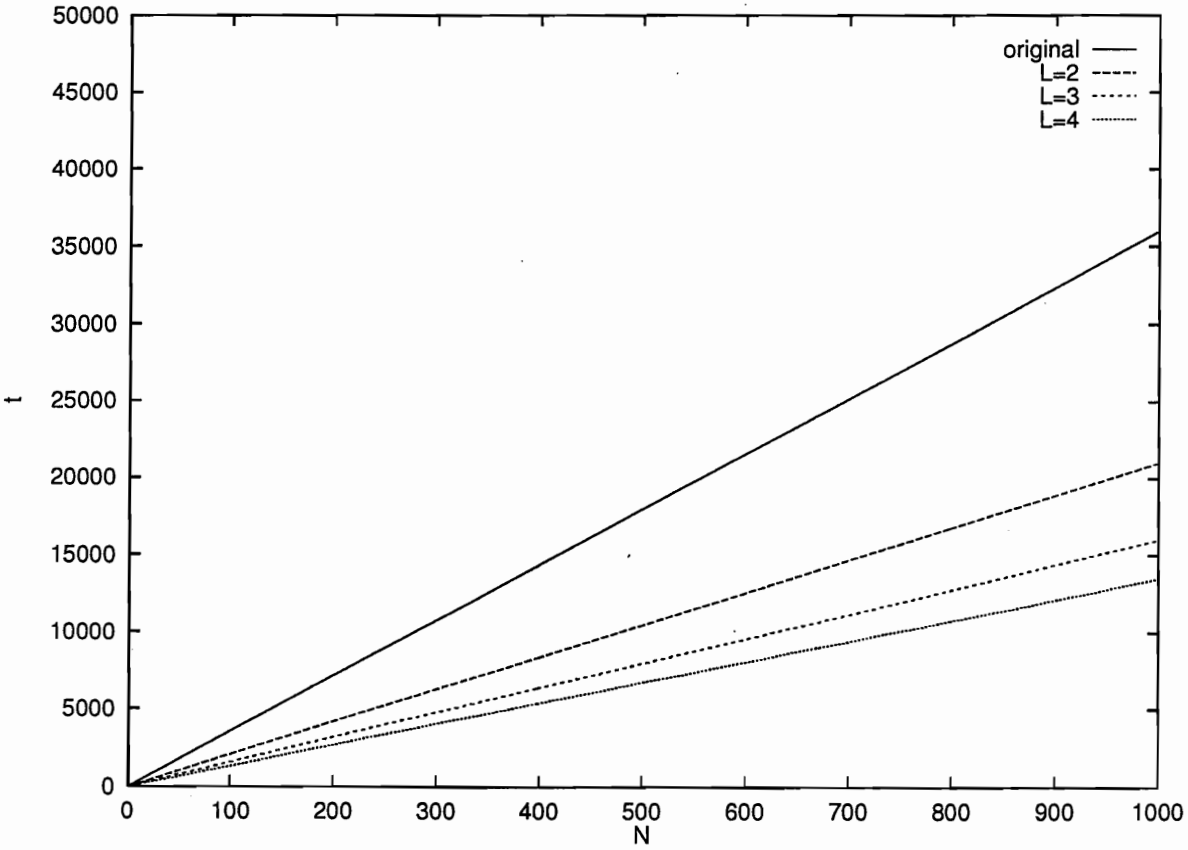


図 4.17: L と Nを変化させたときの t の動き (2) (行列積)

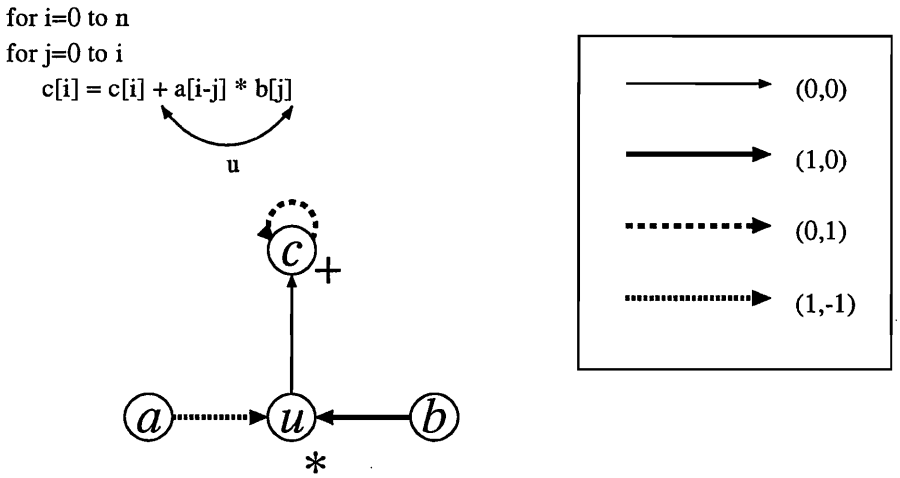


図 4.18: ブロック化前のセル内依存グラフ (コンボリューション)

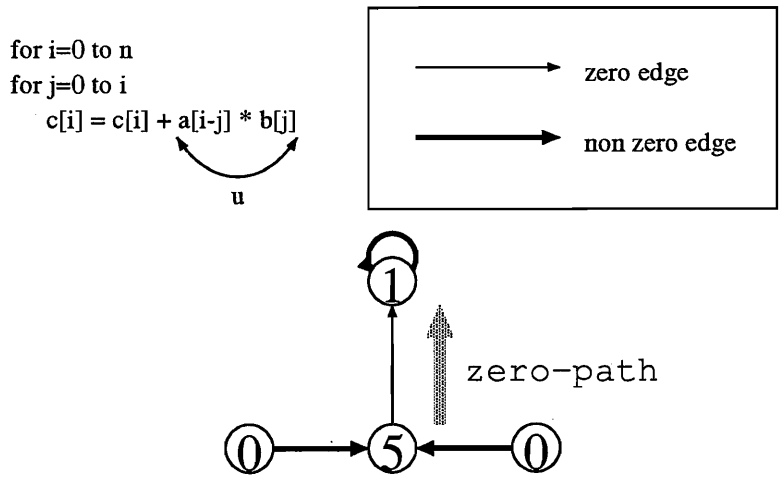
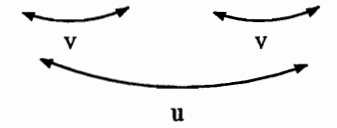
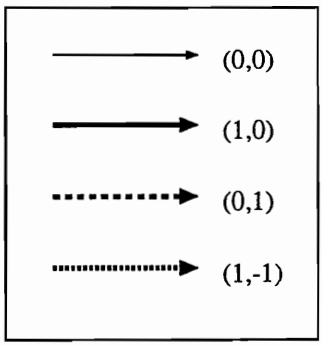


図 4.19: ブロック化前のセル内レジスタグラフ (コンボリューション)

```
for i=0 to n by 2
for j=0 to i by 2
  c[i] = c[i] + a[i-j] * b[j] +a[i-j-1] * b[j+1]
  c[i+1] = c[i+1] + a[i-j+1] * b[j] + a[i-j] * b[j+1]
```





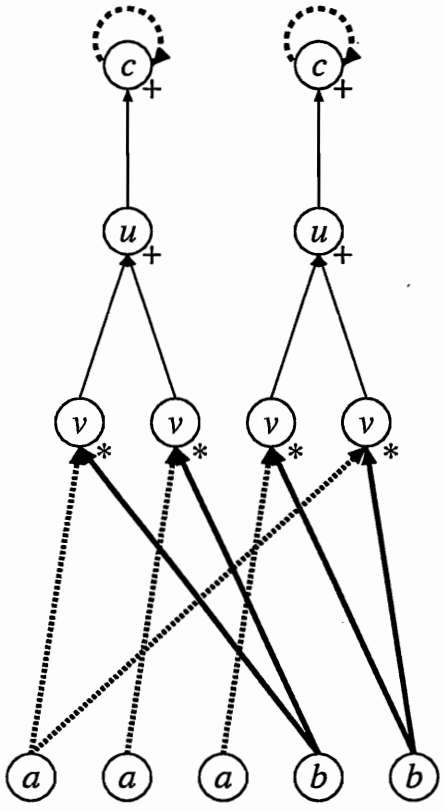


図 4.20: ブロック化後のブロック内依存グラフ (コンボリューション)

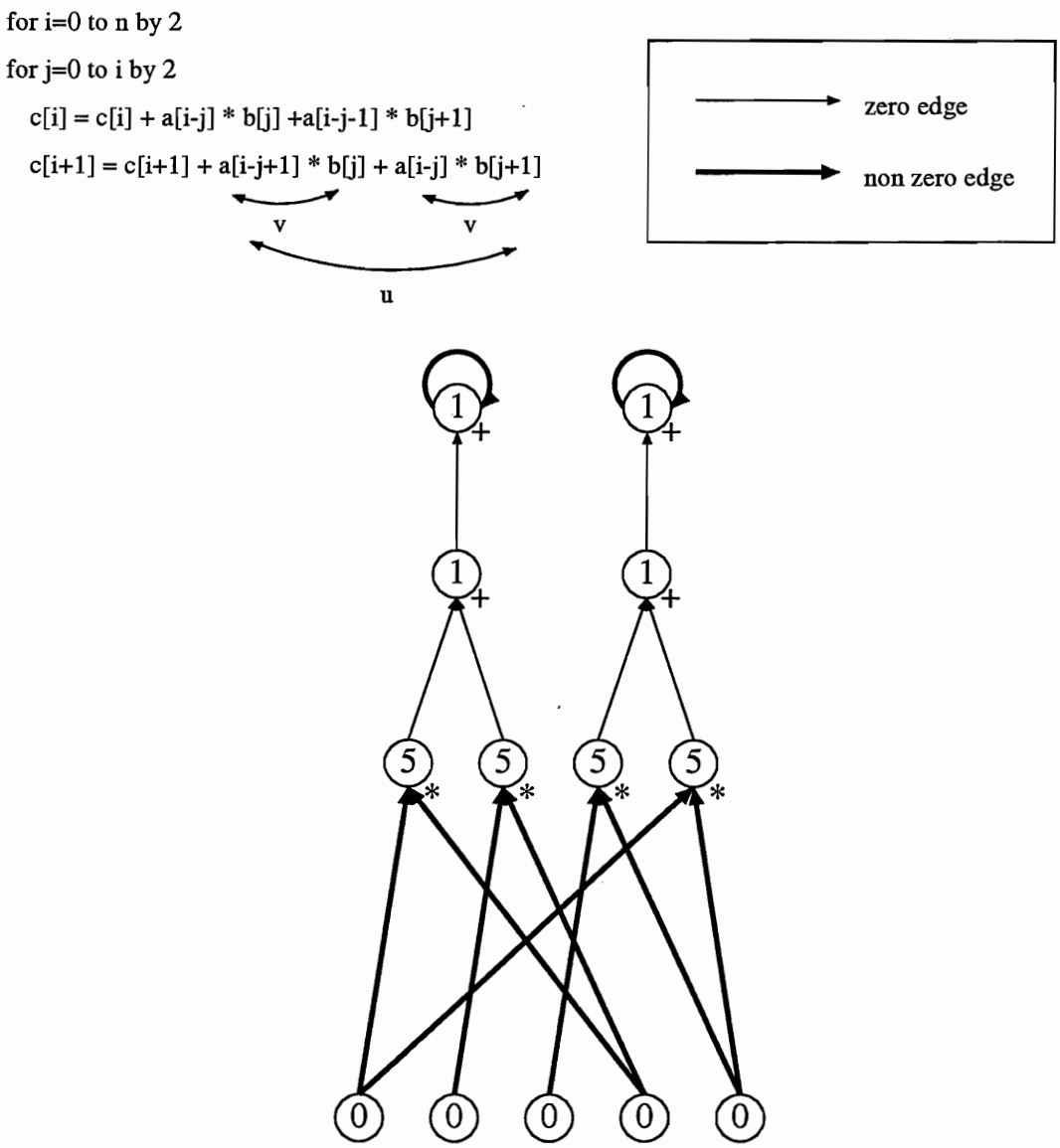


図 4.21: ブロック化後のブロック内レジスタグラフ (コンボリューション)

この結果に対して，更に retiming をおこなう．retiming を行う際の方針は行列積計算のときと同じである．図 4.20の依存グラフに対し，retiming を行った結果を図 4.22，図 4.21のレジスタグラフに対し，retiming を行った結果を図 4.23に示す．ただし，行列積計算の例と同様に，セル内 (ブロック内) において，加減算に 1，乗除算に 5 の時間を要すると仮定している．また，図 4.19，図 4.21，図 4.23のレジスタグラフでは，細線のエッジ重みが 0，太線のエッジ重みが 0 以外であることを示す．

コンボリューションに対し，n_{sys}及びt_{sys}の値を求めると，それぞれ表 4.2のようになる．ただし，空間要素行列 Sを

$$S = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{4.19}$$

と置いた．

	original	blocked	blocked&retimed
n_{sys}	30	16	16
t_{sys}	6	7	5
$n_{\mathrm{sys}} \times t_{\mathrm{sys}}$	180	112	80

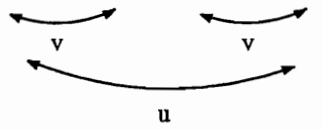
表 4.2: コンボリューションの各計算時間

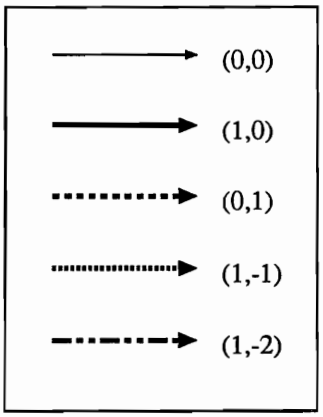
この結果から，BR 法を適用した場合，計算時間は半分またはそれ以下に減らすことができることがわかる．

なお，ブロック化前後のハードウェア構成を図 4.24，図 4.25に示す．ただし，$n = 4$ とした．行列積の場合と同様に，ブロック化前と比較して，ブロック化後の構成はブロック化係数の分だけ縮小したものとなる．ただし，この例では，ブロック化係数が i，j ともに 2 であるが，インデックス jは，i の値によって取りうる値の個数が異なる．図 4.26では，ブロック化前に 1 つのセルで実行される処理を黒丸で示し，1 つのブロックで実行される処理を網掛で表わしている．この図からわかるように，1 つのブロック内では最大で 4 個のイタレーションが実行されることになるが，処理されるイタレーションの個数が 4 個未満のブロックも存在する．そのため，このような空きの部分のイタレーションのデータを参照しなければならないときは，ダミーデータとして 0 値を用いることにする．

ここで，行列積の場合と同じく，評価関数を当てはめてその変化の様子を調べる．w_{s}，w_{t}の値，シリコン面積の比を，行列積のときと同じにしたとき，図 4.27のグラフのとおりになる．行列積の場合と同様，この例でも，g_{s}の値が小さいほど，すなわち g_{t}の値が大きいほど，BR 法の効果が大きくなることがわかる．

```
for i=0 to n by 2
for j=0 to i by 2
  c[i] = c[i] + a[i-j] * b[j] +a[i-j-1] * b[j+1]
  c[i+1] = c[i+1] + a[i-j+1] * b[j] + a[i-j] * b[j+1]
```





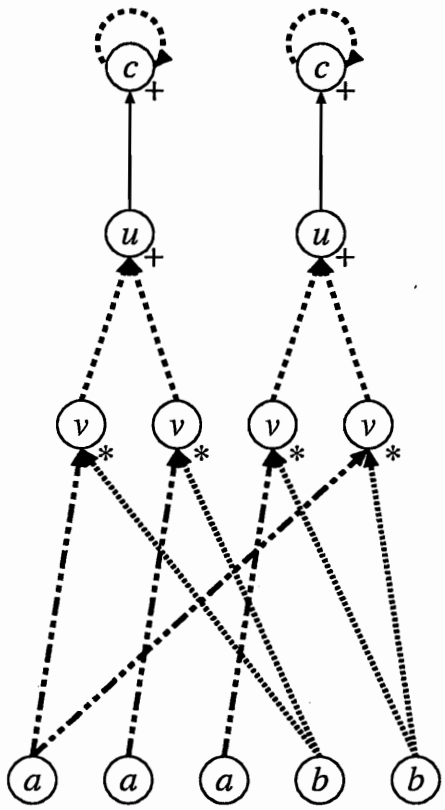


図 4.22: retiming 後のブロック内依存グラフ (コンボリューション)

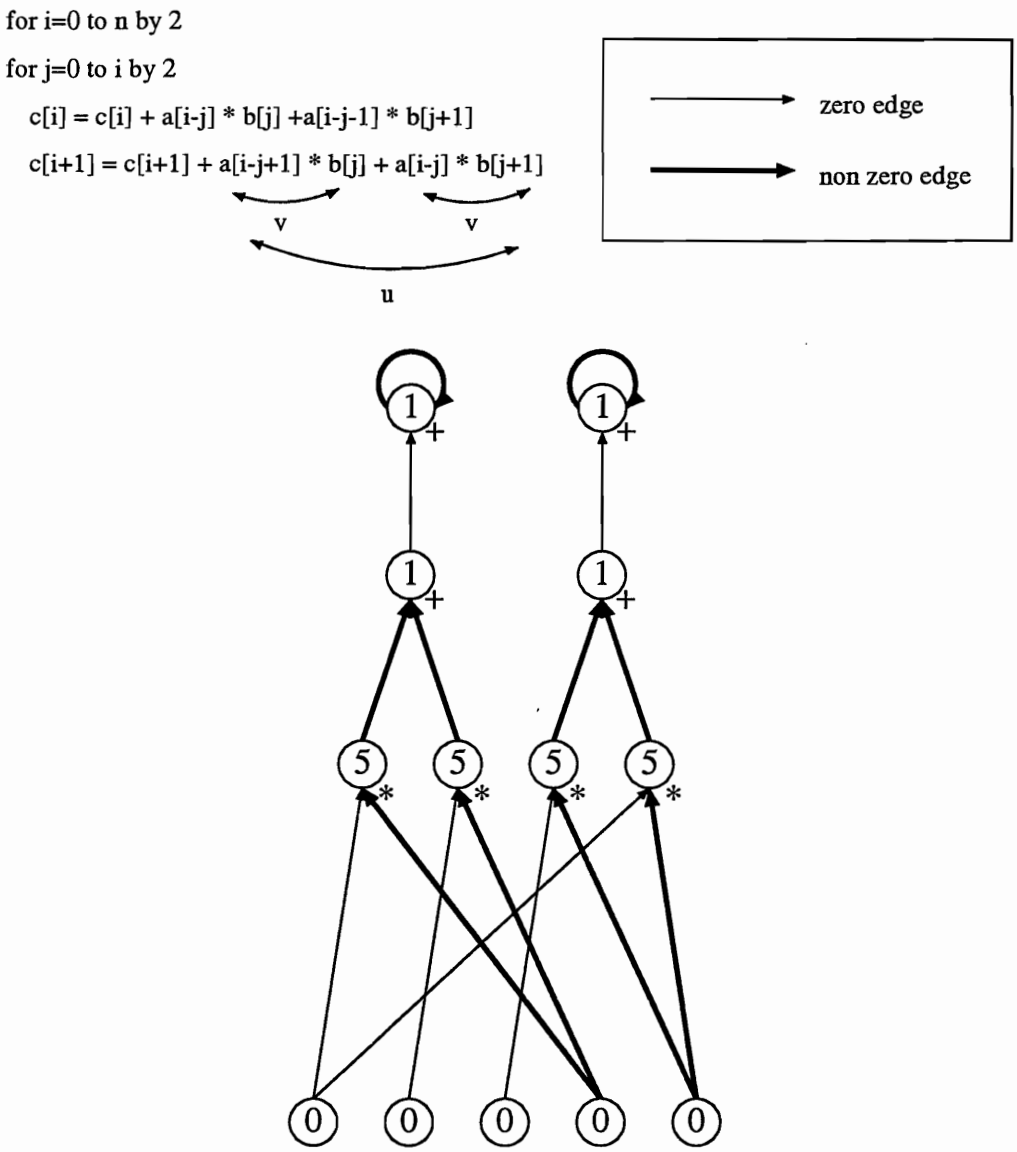


図 4.23: retiming 後のブロック内レジスタグラフ (コンボリューション)

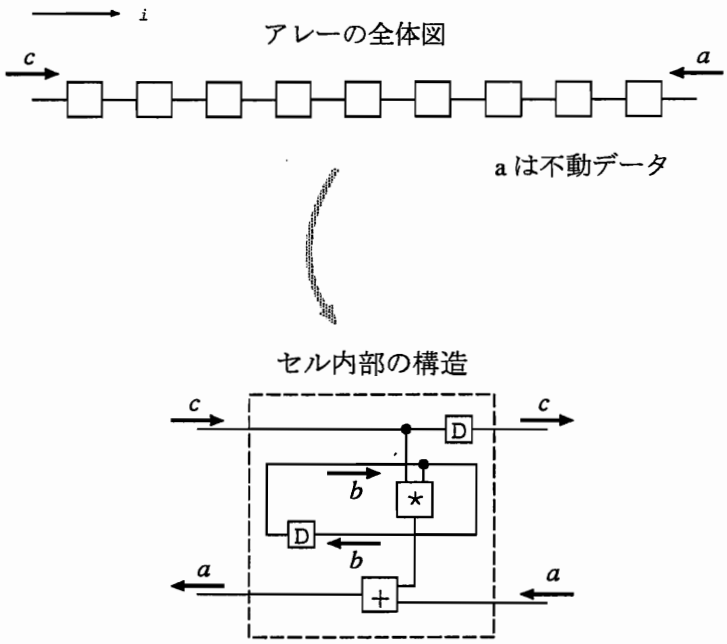
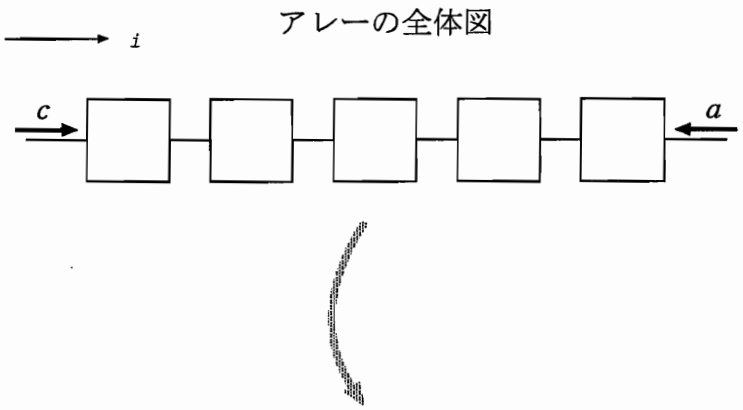


図 4.24: ブロック化前のハードウェア構成 (コンボリューション)



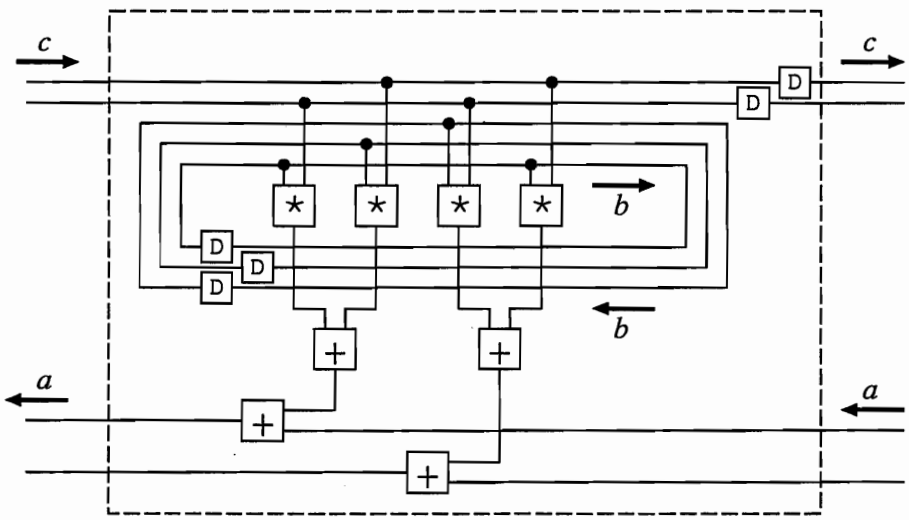


図 4.25: ブロック化後のハードウェア構成 (コンボリューション)

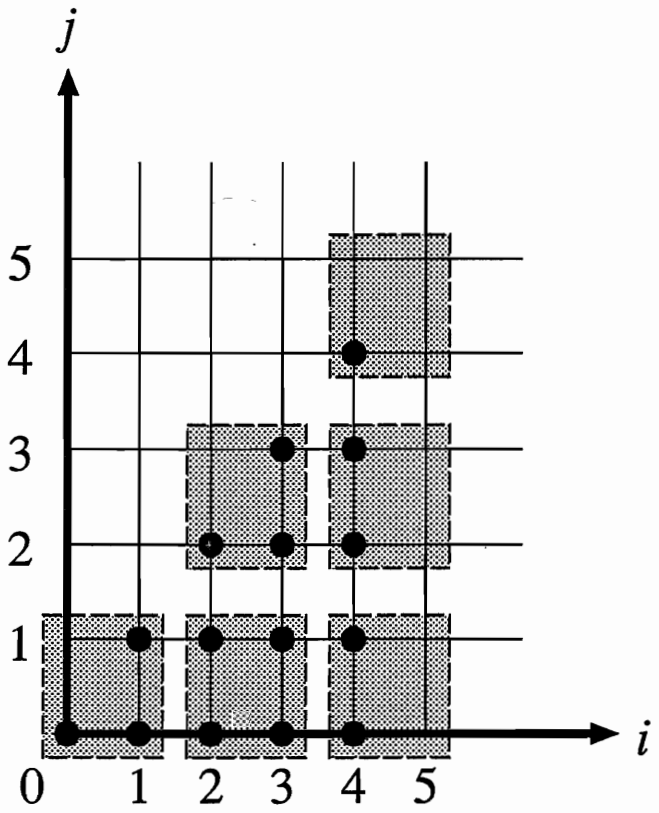


図 4.26: 処理されるイタレーション (コンボリューション)

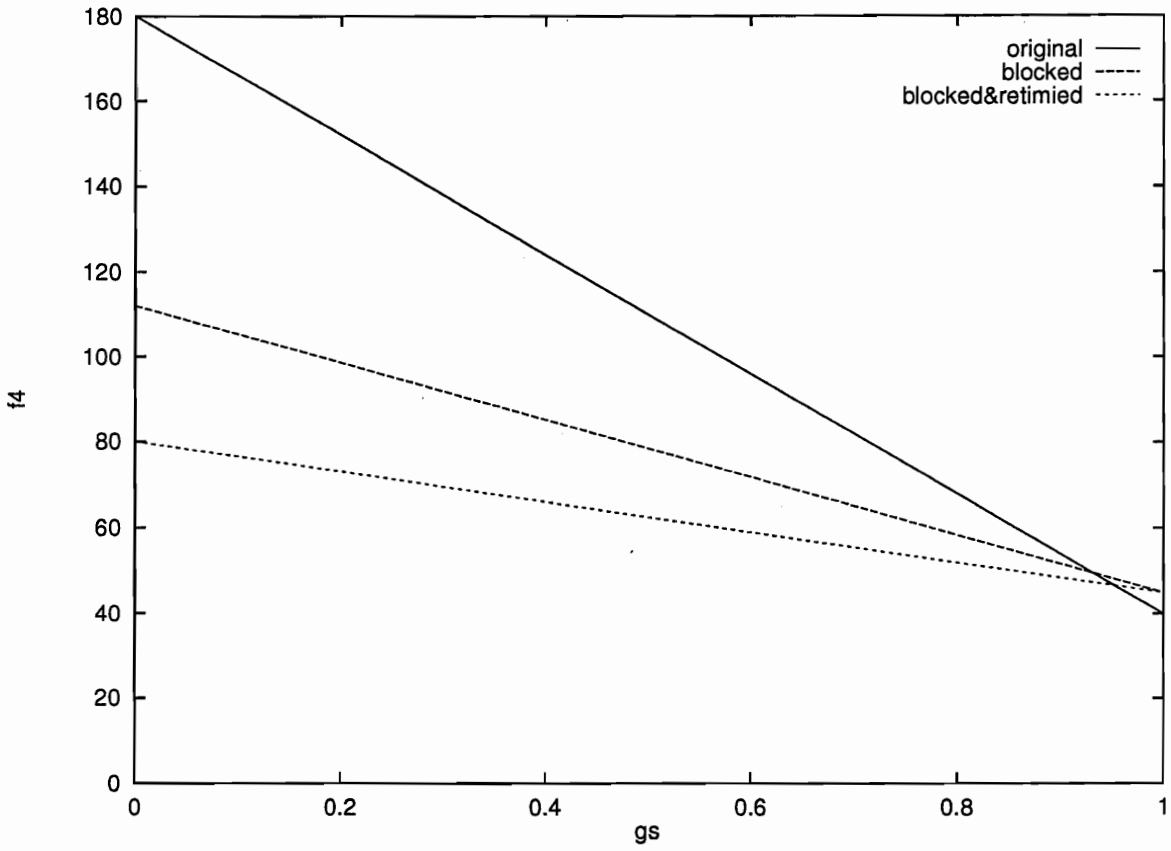


図 4.27: コンボリューションの f_4の値

次に，Nの値及び，ブロック化係数を変化させたときの計算時間の動きを考える．行列積のときと同様に，各インデックスのブロック化係数はすべてLであるとする．

行列積の場合と同様に，コンボリューションの場合も，n_{sys}の値は$\left\lceil \frac{N}{L} \right\rceil$に比例し，

$$n_{sys} = 7\left\lceil \frac{N}{L} \right\rceil + 2 \tag{4.20}$$

と表わされる．一方，t_{sys}の値は，行列積の場合と全く同一で

$$t_{sys} = L + 5 \tag{4.21}$$

となる．式(4.20)と式(4.21)より，LとNを変化させたときのtの動きは図4.28，図4.29のようになる．ブロック化を行うことで，計算時間を小さくすることができるが，Nが小さいときは，ブロック化係数を大きくすれば計算時間もそれだけ小さくなるとは限らない．Nが大きいと，ブロック化係数の効果が表われるが，反面，ブロックが大きくなることに対してのオーバーヘッドを考慮しなければならない．

4.5 むすび

本章では，時間的資源を小さくすることに有効な手段である，BR 法を提案し，具体例に適用してその有効性を確かめた．

今まで，時間的資源を小さくすることへの努力は，すなわち，ステップ数を小さくすることであった．しかし，ステップ数n_{sys}とセル内計算時間t_{sys}は本来トレードオフの関係にあり，ステップ数を減らすことによって外見上は計算時間が小さくなったように見えても，実際には，その分セル内計算時間が増加することもある．したがって，n_{sys}の値のみに注目するのではなく，t_{sys}の値にも注目するべきである．

t_{sys}を小さくする手法として，かつてretimingという手法が提案された．しかしretimingだけでは，プログラムの粒度が細かいときにはt_{sys}の値を小さくすることはできない．そこで，ブロック化という手順を付加することを提案した．ブロック化を行うことによって，細粒度の問題を意図的に粗粒度に変換し，その結果，細粒度のプログラムに対してもretimingを行うことが可能になる．また，ブロック化により，n_{sys}の値が小さくなり，n_{sys}とt_{sys}との積で表わされる計算時間も小さくなる．最後に，大規模科学技術計算における典型的なアルゴリズムである行列積計算やコンボリューションに対し，ブロック化の効果，及びBR 法の有効性を確認した．

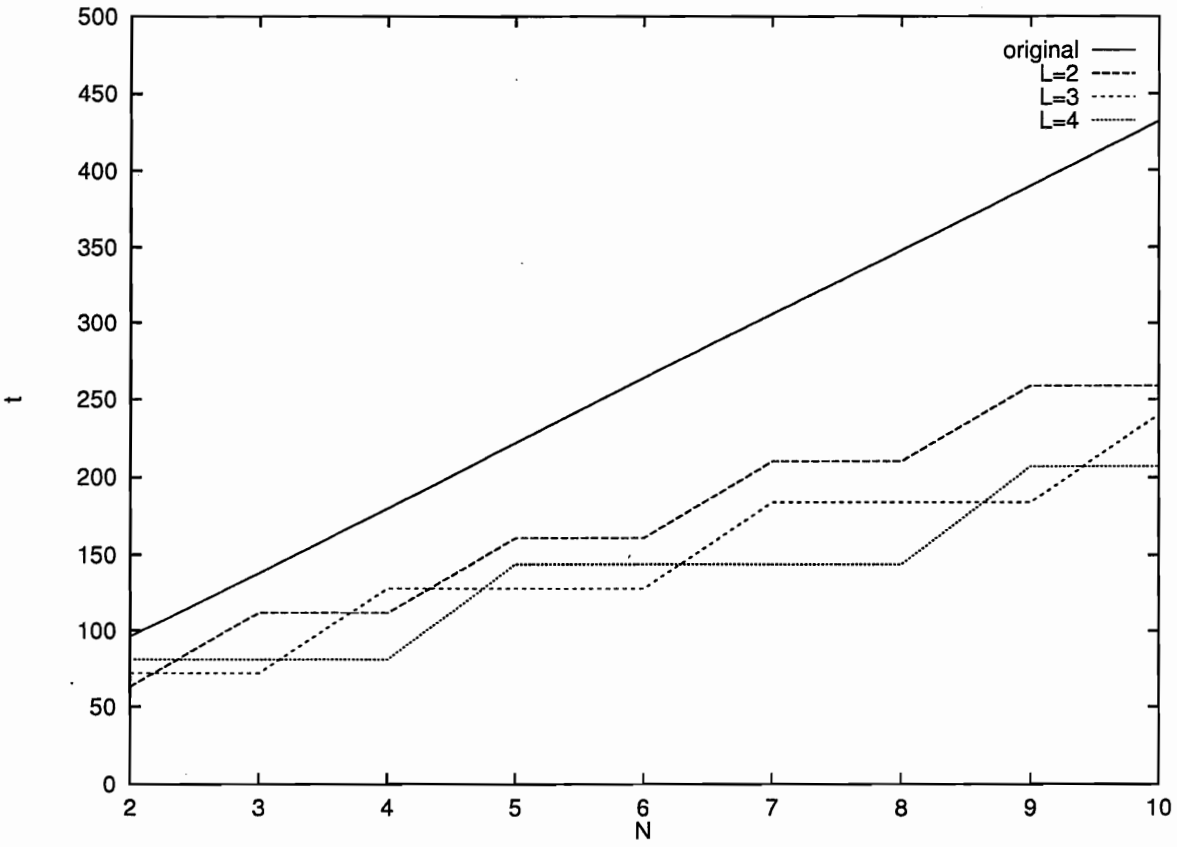


図 4.28: L と Nを変化させたときの t の動き (1) (コンボリューション)

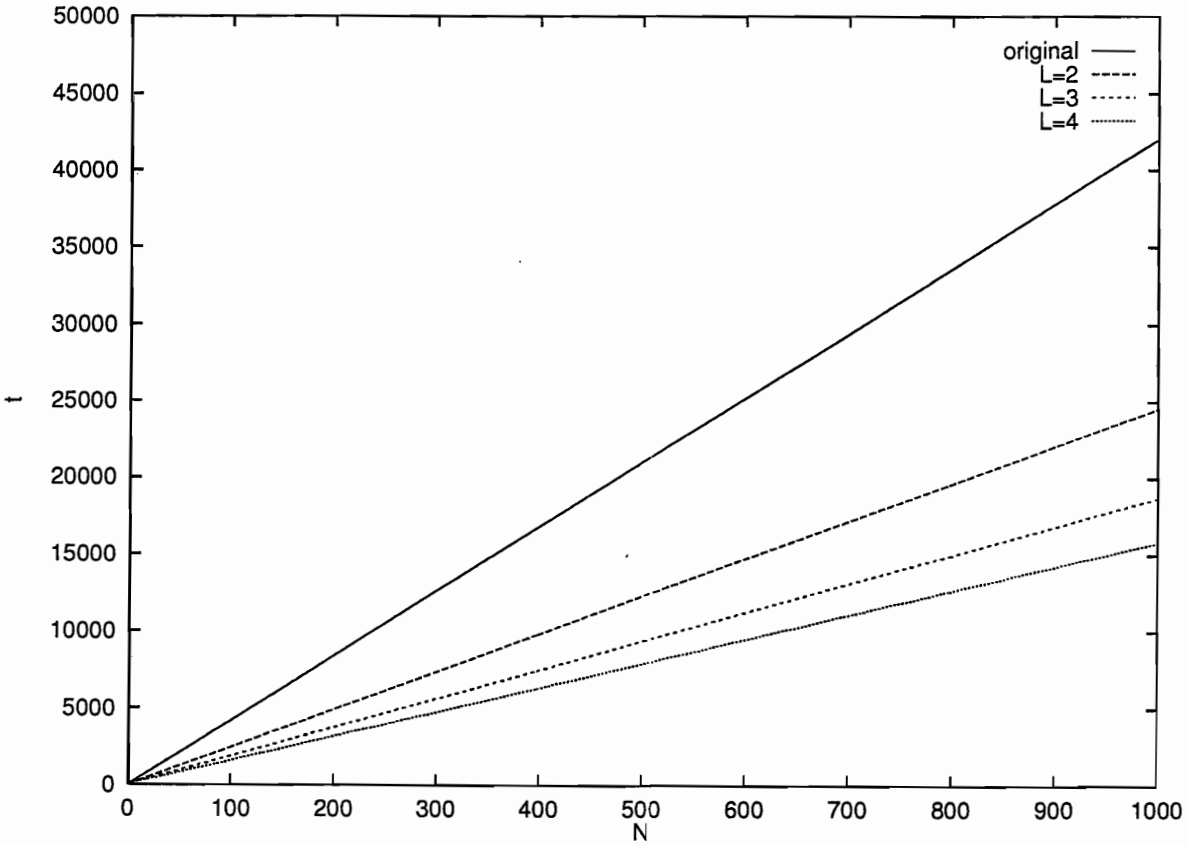


図 4.29: L と Nを変化させたときの t の変化 (2) (コンボリューション)

第 5 章

結論・検討

5.1 結論

本研究では，まず並列処理アーキテクチャのシストリックアレーに関して，その効率性を評価する際に不可欠な，種々の評価値についての考察を行った．

第 3 章においては，

- シストリックアレーの評価値の検討
- 空間，時間の両者の資源を融合した関数の検討

を行った．評価値の検討については，種々の評価値を，空間的資源と時間的資源に大別し，そのそれぞれに属する評価値の再検討を行った．また，融合関数については，シストリックアレーの設計時の仕様の多様性にも対応する関数となるように，重み変数を導入した関数についての考察を行った．従来提案されていた，重みつきの関数は，式の中に統計量を用いていたため，比較対象としたい構成のシストリックアレー以外の組合せについても考慮し，その評価値を計算してから関数を計算する必要があった．そこで本論文では，統計量を用いない関数を提案し，計算の無駄を省いた．また，具体例を用いて，その関数における重み変数の効果，有効性を確認した．

また，並列処理では，時間的資源を小さく抑えることが必要である．そのため，第 4 章においては，

- BR 法の提案
- 適用例による BR 法の有効性の確認

を行った．BR 法は，時間的資源を小さくすることができる有効な手法である．BR 法は，ブロック化と retiming の 2 つの方法により成り立っている．retiming によって，セル内のレジスタの配置を再構成し，セル内計算時間 t_{sys} を小さくする方法がかつて提案された．しかし，この方法は，粗粒度の問題にしか適用できないといった問題点が存在したため，解決策として，retiming の前にブロック化を行うことを提案した．ブロック化を行なってから retiming を行うことは，細粒度の問題を意図的に粗粒度にすることになり，ブロック化の適用可能範囲を広げることができる．また，ブロック化は，問題の全ステップ数 n_{sys} を小さく抑えることもできるので，n_{sys} と t_{sys} の積で表わされる，アレーの計算時間も小さく抑えることができる．そして，科学技術計算の中でも典型的なアルゴリズムである，行列積やコンボリューションのアルゴリズムを例にとって，この BR 法を適用し，実際に計算時間を小さくする効果があることを確認した．

5.2 検討

重み付き融合評価関数においては，w_s, w_t の数値を現実に即して与えることは難しい．特に w_t の値は,「時間が余計にかかったときのコスト」を意味するが,「時間」から「値段」への変換は設計環境に依存するからである．また，本論文では，単純に，空間成分の評価値 1 種類と，時間成分の評価値 1 種類のみを用いて，関数を計算しているが，実際には，それ以外の要素を考慮しなければならないこともある．例えば，アレー中の最大消費電力を考慮しなければならない場合などである．たくさんの評価値を考慮して関数を決定することは，繁雑性も増すことになるので，本論文ではこのような場合は考えなかったが,「シストリックアレー設計の多様性により対応」させるためには考慮しなければならない課題であると言える．

BR 法においては，ブロック化を行う際の条件が厳しいことが問題である．科学技術計算においては，比較的単純なアルゴリズムが繰り返し使われることが多いのでブロック化が行えることが多いが，データ依存が複雑に絡み合ったアルゴリズムに対しては意味を変えないブロック化は容易ではない．また，ブロック化係数を上げるに従って増加する，ブロック内の統轄・管理装置によるオーバーヘッドは，本論文ではふれなかった．しかし，ブロック化によって，どの程度粒度を粗くすれば，総合的にコストが小さく抑えられるかを考えることは重要である．また，retiming での offset vector の選び方は，現在は，zero-path に着目し，エッジの重みが非 0 となるように方針を立てて offset vector を選んでいるが，体系的にまとめられた方法はない．そのため，t_{sys} の値を最小にするような，体系的な reindexing，retiming の方法を求めることも 1 つの課題である．

謝辞

本研究を進めるに当って，全般的に御指導いただいた，東北大学工学部阿曽弘具教授に心より感謝いたします．

本研究をまとめるに当って，適切な御助言をいただいた，東北大学工学部西関隆夫教授，亀山充隆教授に大変感謝いたします．

いつも熱心に御討論していただいた，東北大学工学部助手成富敬氏，東北大学大学院工学研究科博士課程藤岡豊太氏をはじめ，東北大学工学部通信工学科阿曽研究室並列グループの皆様に深く感謝いたします．

また，日頃より研究活動を共にし，計算機環境を常に快適な状態に整備していただいた，東北大学工学部通信工学科阿曽研究室の皆様，ならびに同研究室の同級生諸氏に深く感謝いたします．

参考文献

[1] 梅尾：“超並列計算機アーキテクチャとそのアルゴリズム”, 共立出版 (1991).

[2] 奥川：“並列計算機アーキテクチャ”, コロナ社 (1991).

[3] M. S. Lam: “A Systolic Array Optimizing Compiler”, Kluwer Academic Publishiers (1989).

[4] C.N.Zhang, J.H.Weston and Y.-F.Yan: “Determining Objective Functions in Systolic Arrays”, IEEE TRANS. ON VLSI SYSTEMS, **2**, 3, pp. 357–360 (1994).

[5] M.O.Esonu and A.J.Al-Khalili: “Systolic arrays: how to choose them”, IEE PROCEEDINGS-E, **139**, 3, pp. 179–188 (1992).

[6] D.Ait-Boudaoud, M.K.Ibrahim and B.R.Hayes-Gill: “Novel cell architecture for bit level systolic array multiplication”, IEE PROCEEDINGS-E, **138**, 1, pp. 21–26 (1991).

[7] K.-L. Wong and W.-C. Siu: “Data Routing Networks for Systolic/Pipeline Realization of Prime Factor Mapping”, IEEE TRANS. ON COMP., **40**, pp. 1072–1074 (1991).

[8] S.Sarker and A.K.Majumdar: “Tagged systolic arrays”, IEE PROCEEDINGS-E, **138**, 5, pp. 289–294 (1991).

[9] P.Murtagh, A.C.Tsoi and N.Bergmann: “Bit-serial systolic array implementation of a multilayer perception”, IEE PROCEEDINGS-E, **140**, 5, pp. 277–288 (1993).

[10] C.-L. Wang: “Bit-Level Systolic Array for Fast Exponentiation in GF(2^m)”, IEEE TRANS. ON COMP., **43**, 7, pp. 838–841 (1994).

[11] S.-Y. Kung, K.S.Arun, R. J.Gal-Exer and D. rao: "Wavefront Array Processor: Language Architecture, and Applications", IEEE TRANS. ON COMP., **c-31**, 11, pp. 1054–1066 (1982).

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

[13] D. I. Moldovan and Jose.A.B.Fortes: "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays", IEEE TRANS. ON COMP., **c-35**, 1, pp. 1–12 (1986).

[14] 三浦, 阿曽, 稲垣 : "多重ループプログラムを処理するシストリックアルゴリズムの構成法", 信学論 (D), **J70-D**, 8, pp. 515–524 (1987).

[15] 飯国, 酒井, 得丸 : "データ依存グラフを用いたシストリックアレーの自動設計", 信学論 (A), **71-A**, 6, pp. 1282–1290 (1988).

[16] 飯国, 酒井, 得丸 : "1 次元シストリックアレーの設計法", 信学論 (D), **J71-D**, 8, pp. 1480–1486 (1988).

[17] 阿曽 : "シストリックアレーの自動設計法", 信学論 (D), **J71-D**, 8, pp. 1487–1495 (1988).

[18] 前場, 辰巳 : "正射影に基づくシストリックアルゴリズム設計手法", 信学論 (A), **J71-A**, 10, pp. 1878–1887 (1988).

[19] P.-Z. Lee and Z. M. Kedem: "Mapping Nested Loop Algorithms into Multidimentional Systolic Arrays", IEEE TRANS. ON P.D.S., **1**, 1, pp. 64–76 (1990).

[20] J.-P. Sheu and C.-Y. Chang: "Synthesizing Nested Loop Algorithms Using Nonlinear Transformation Method", IEEE TRANS. ON P.D.S., **2**, 3, pp. 304–317 (1991).

[21] 小川, 前場, 阿部 : "多次元シストリックアレーの系統的設計手法", 信学論 (D-I), **J75-D-I**, 9, pp. 879–881 (1992).

[22] M. E.Wolf and M. S.Lam: "A Loop Transformation Theory and an Algorithm to Maximize Parallelism", IEEE TRANS. ON P.D.S., **2**, 4, pp. 452–471 (1991).

[23] 阿曽 : "シストリックアレー設計理論の最近の動向", 電気情報通信学会秋季大会 (1993).

[24] 白石 : "シストリックアルゴリズム開発支援システムに関する研究", 東北大学大学院工学研究科修士学位論文 (1992).

[25] Y. Wong and J.-M. Delosme: "Optimization of Computation Time for Systolic Arrays ", IEEE TRANS. ON COMP., **41**, 2, pp. 159–177 (1992).

[26] 佐藤："最適並列処理方式設計システムの構築", 東北大学工学部情報工学科学士学位論文 (1994).

研究業績

- “シストリックアレーの最適化法”
  佐藤　英，阿曽弘具
  1994 年度 電気関係学会東北支部連合大会，2C-5 (1994-8)

- “シストリックアレーの高速化手法”
  佐藤　英，阿曽弘具
  1995 年 電気情報通信学会ソサイエティ大会，D-37 (1995-9)

修士学位論文本審査用ＯＨＰ資料

シストリックアレーの効率的構成法に関する研究

東北大学大学院工学研究科
電気・通信工学専攻

佐藤　英

1　序論

本研究の背景

処理速度が速い計算機の追求

→プロセッサの処理能力：限界あり

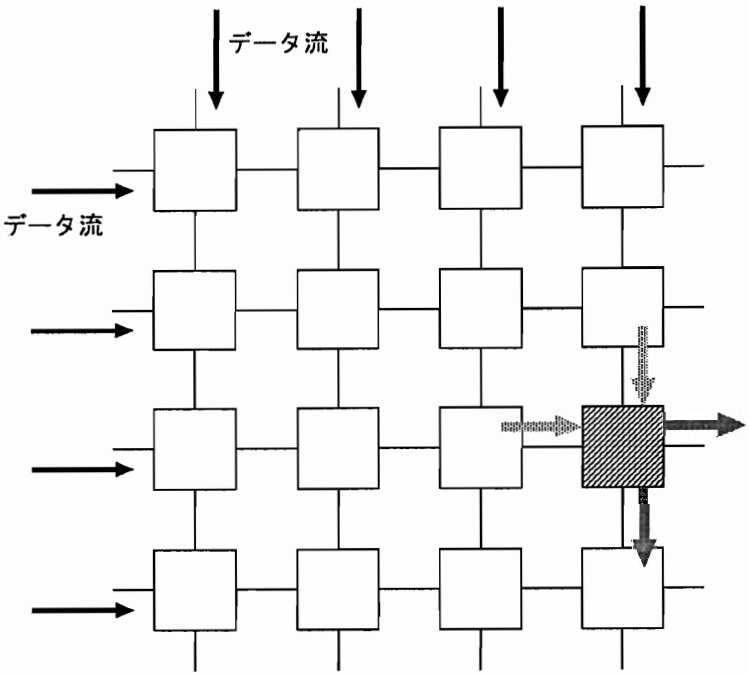
→並列処理の必要性(同時に複数個の計算)

シストリックアレー (1978:Kung,Leiserson)

- 拡張性 (構造が一様)
- フォールトトレランス性 (通信が局所的)

→信号処理，記号処理

シストリックアレー



- パイプライン的なデータ流
- データ同士が衝突したセルで計算
- セル内でごく単純な処理

シストリックアレー設計時の問題点

同一の問題に対する構造が複数個存在

$\longrightarrow$ どの構成のものを採用するか

for $i = 1$ **to** 3

for $j = 1$ **to** 3

for $k = 1$ **to** 3

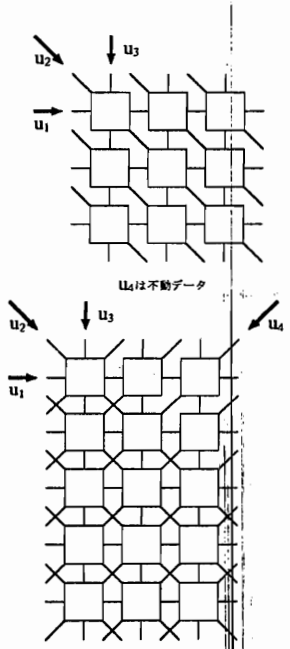
$$u_1 = u(i_1 + 1, i_2)$$

$$u_2 = u(i_1, i_2 + 1)$$

$$u_3 = u(i_1 - 1, i_2)$$

$$u_4 = u(i_1, i_2 - 1)$$

$$u(i_1, i_2) = \frac{1}{4}(u_1 + u_2 + u_3 + u_4)$$



本研究の目的

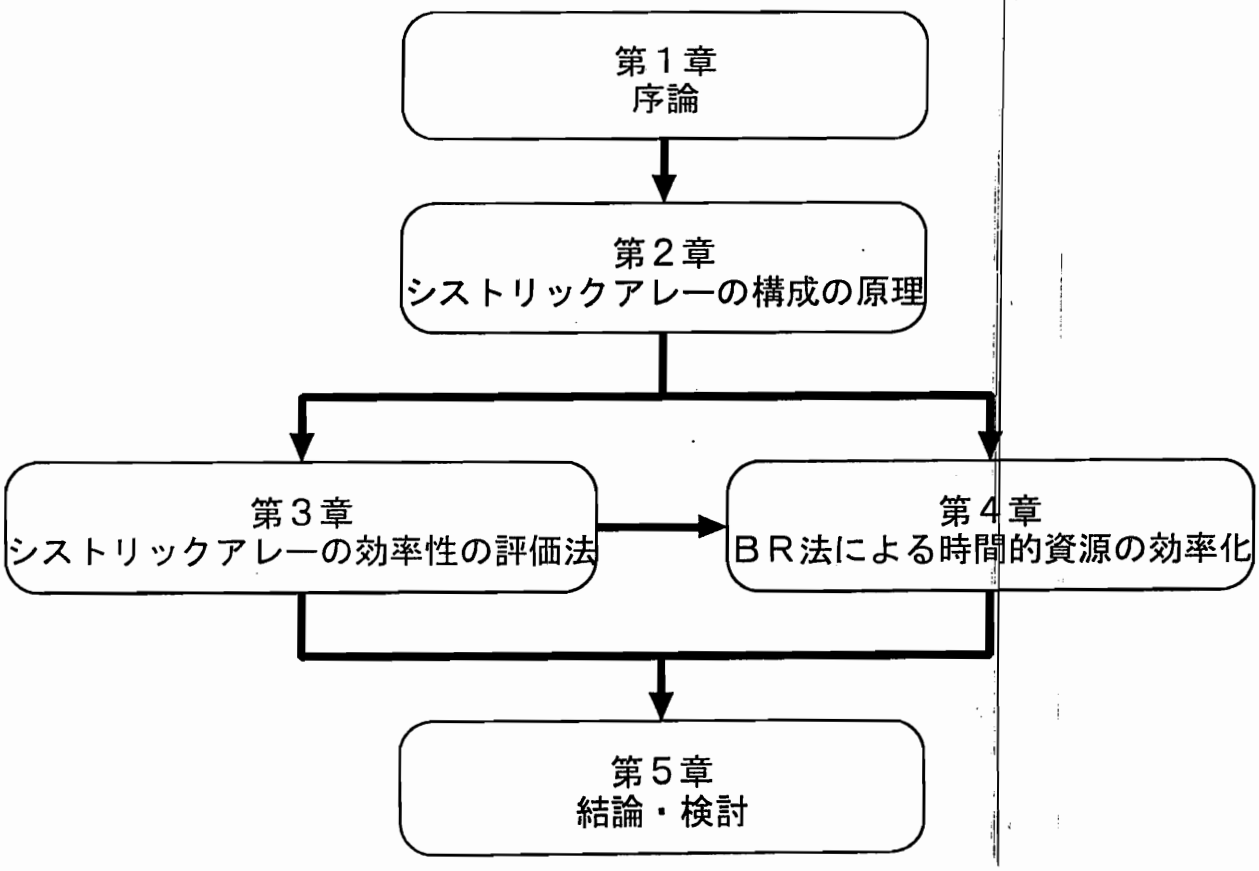
異なる構造のシストリックアレー

⇓

シストリックアレーの効率性に関する考察

- 効率性を表わす評価値
- 効率化を実現する手法の提案

本論文の構成



2　シストリックアレーの構成の原理

Moldovan 法

$\mathbf{I} = {}^t(i_1\ i_2\ \cdots\ i_n)$　インデックスベクトル

$\mathbf{T} = \begin{bmatrix} \Pi \\ S \end{bmatrix}$　変換行列 (Π:時間, S:空間)

※ Πは n 次行ベクトル，Sは $(n-1)\times n$ 行列

$$\longrightarrow \mathbf{J} = {}^t(\ \overbrace{j_1}^{\text{時刻}}\ \underbrace{j_2\ \cdots\ j_n}_{\text{空間座標}}\ ) = \mathbf{TI}$$

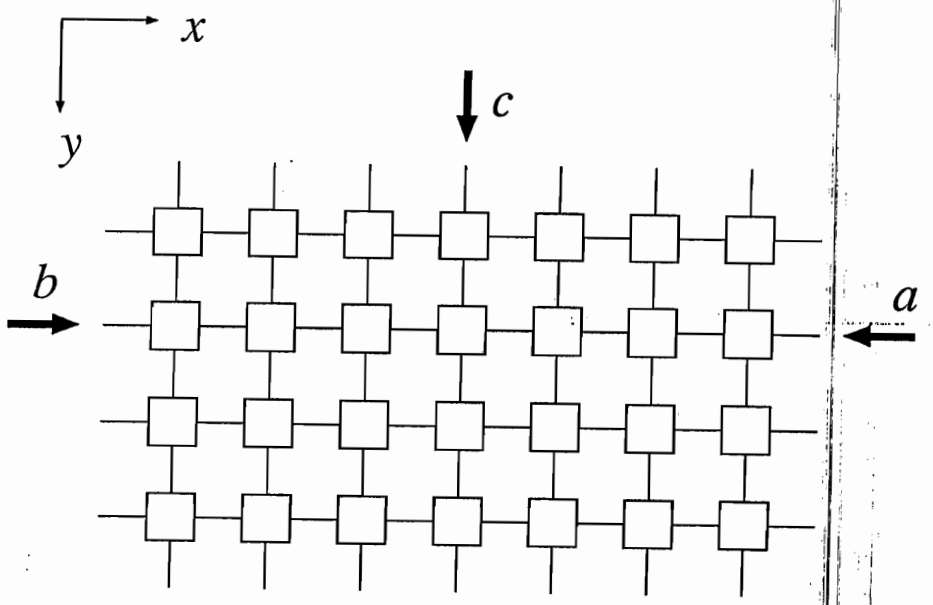
Moldovan 法の制限：$\mathbf{\Pi d}_i > 0 \quad (i = 1, 2, \cdots, m)$

シストリックアレー設計の具体例

for $i = 1$ **to** 4
for $j = 1$ **to** 4
for $k = 1$ **to** 4
　　$c[i,j] = c[i,j] + a[i,k] * b[k,j]$

$$\mathbf{d}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}\ \mathbf{d}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}\ \mathbf{d}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\ T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

行列積計算を行うシストリックアレー



3　シストリックアレーの効率性の評価法

はじめに

複数個の構成の存在

⟶どのシストリックアレーを採用するか

⟶ いかに消費資源を小さく抑えるか

消費資源

- 空間的資源 (セル数，セル面積，I/O ピン数)
- 時間的資源 (計算時間，データ転送時間)
- その他の資源 (フォールトトレランス性)

計算時間の再定義

計算時間 t： $t = n_{\mathrm{sys}} \times t_{\mathrm{sys}}$

n_{sys}：ステップ数　　t_{sys}：セル内計算時間

t_{sys}を考慮$\longrightarrow$ 厳密な計算時間の大小を表記

従来の融合評価関数

$f_1 = S_{\mathrm{c}} \times {n_{\mathrm{sys}}}^2$　　(S_{c}: セル面積)

$\longrightarrow$ 計算ステップ内で流れうるデータ流の量

$f_2 = S_{\mathrm{c}} \times n_{\mathrm{sys}}$　$\longrightarrow$ 全セルの使用効率に反比例

f_1：時間的資源重視 $\Longleftrightarrow$ f_2：空間的資源重視

重み付けをした関数

重み変数の導入：

シストリックアレー設計の多様性に対応

$$f = g_{\mathrm{s}} \cdot w_{\mathrm{s}} \cdot \mathrm{s} + g_{\mathrm{t}} \cdot w_{\mathrm{t}} \cdot \mathrm{t} \quad (g_{\mathrm{s}} + g_{\mathrm{t}} = 1)$$

s, t：考慮する評価値の値

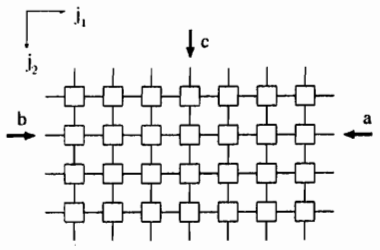
g_{s}, g_{t}：重み変数

w_{s}, w_{t}：評価値の単位量分増加時のコスト

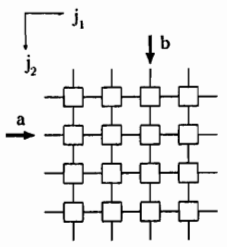
重み変数の導入$\longrightarrow$空間要素＋時間要素

fの具体例へのあてはめ (1)

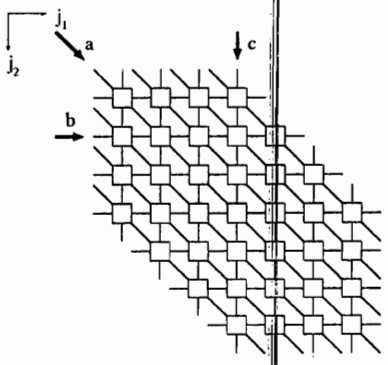
$$T_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad T_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad T_4 = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



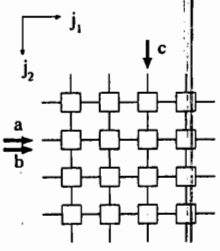
T_1



T_2

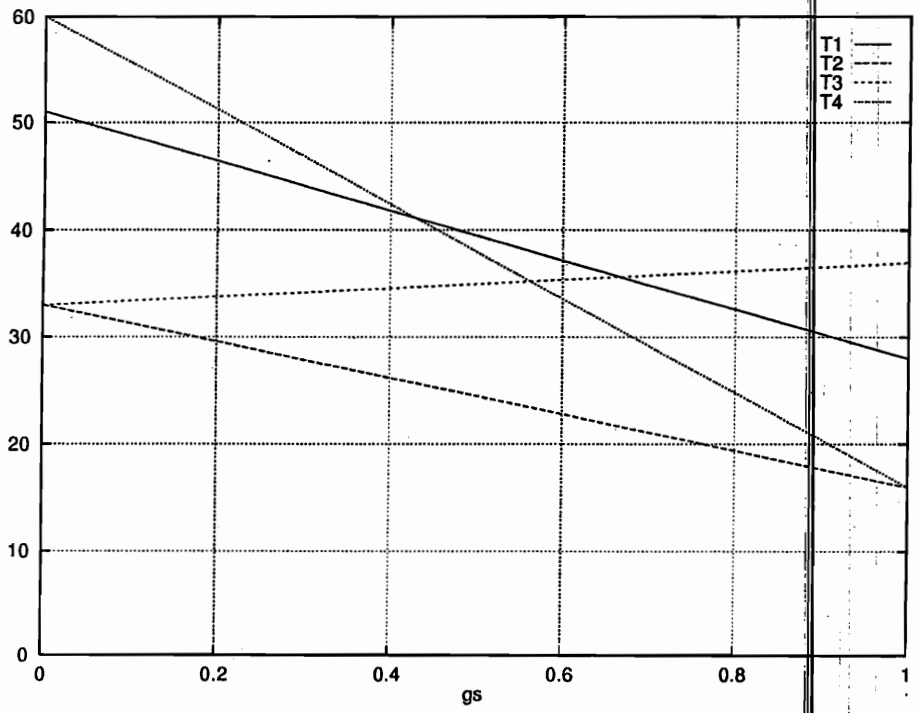


T_3



T_4

fの具体例へのあてはめ (2)



$$w_{\mathrm{t}} = 1, w_{\mathrm{s}} = 3$$

第 3 章のまとめ

- 評価値，融合関数の意味付け，検討
- 重み付き融合関数を定義，検討

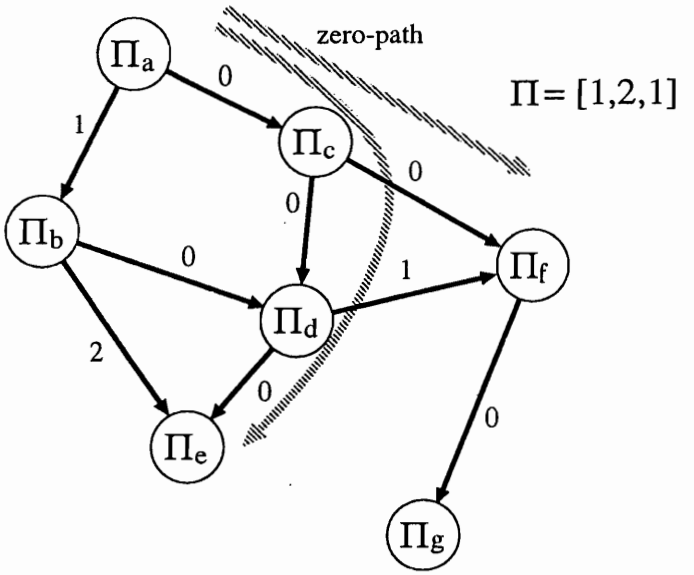
4　BR 法による時間的資源の効率化

BR 法の導入

空間的資源$\longleftrightarrow$時間的資源 (トレードオフ)
　最も優先すべき資源を考慮

時間的資源：最優先で効率化する場合が多い
　$\longrightarrow$**BR 法**
　　(Blocking and Retiming method)
　retiming：Wong,Delosme(1992)

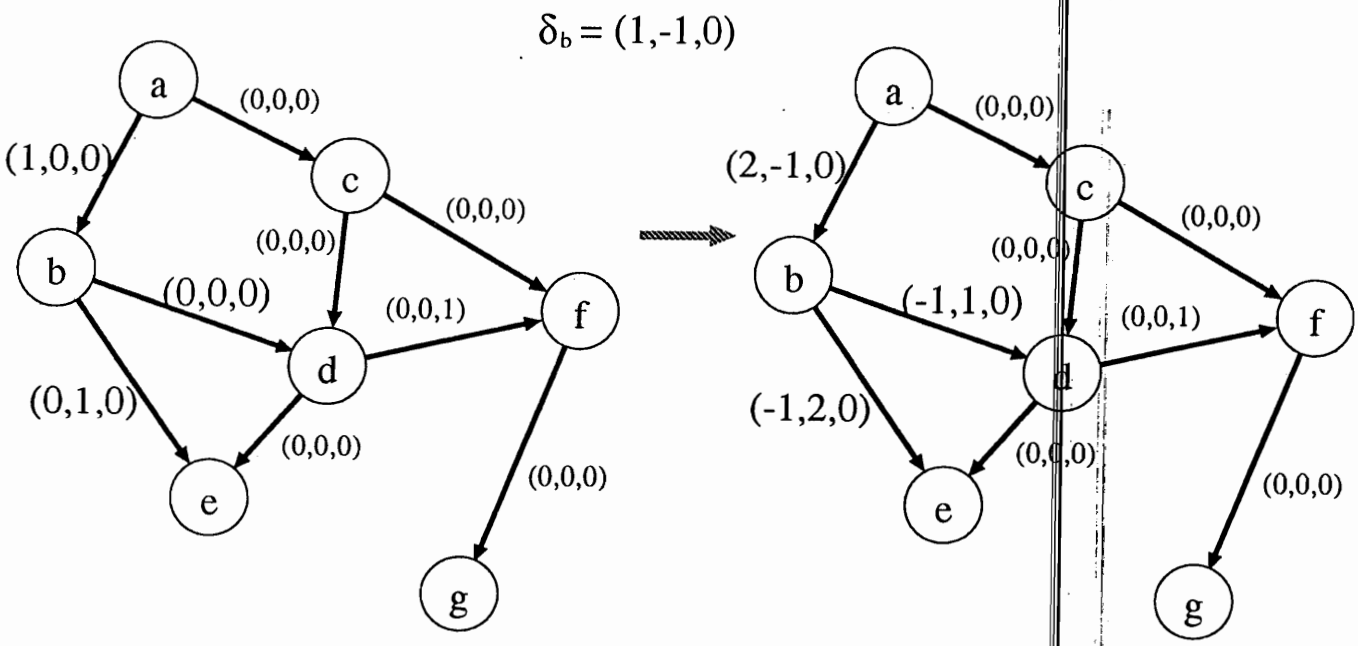
レジスタグラフ



$t_{sys}\cdots$ *zero − path* 中のノード重みの和の最大値

reindexing と retiming

$$\overline{d_{uv}} = d_{uv} + \delta_v - \delta_u$$



retiming の欠点

retiming $\cdots t_{\mathrm{sys}}$の値を小さくする効果

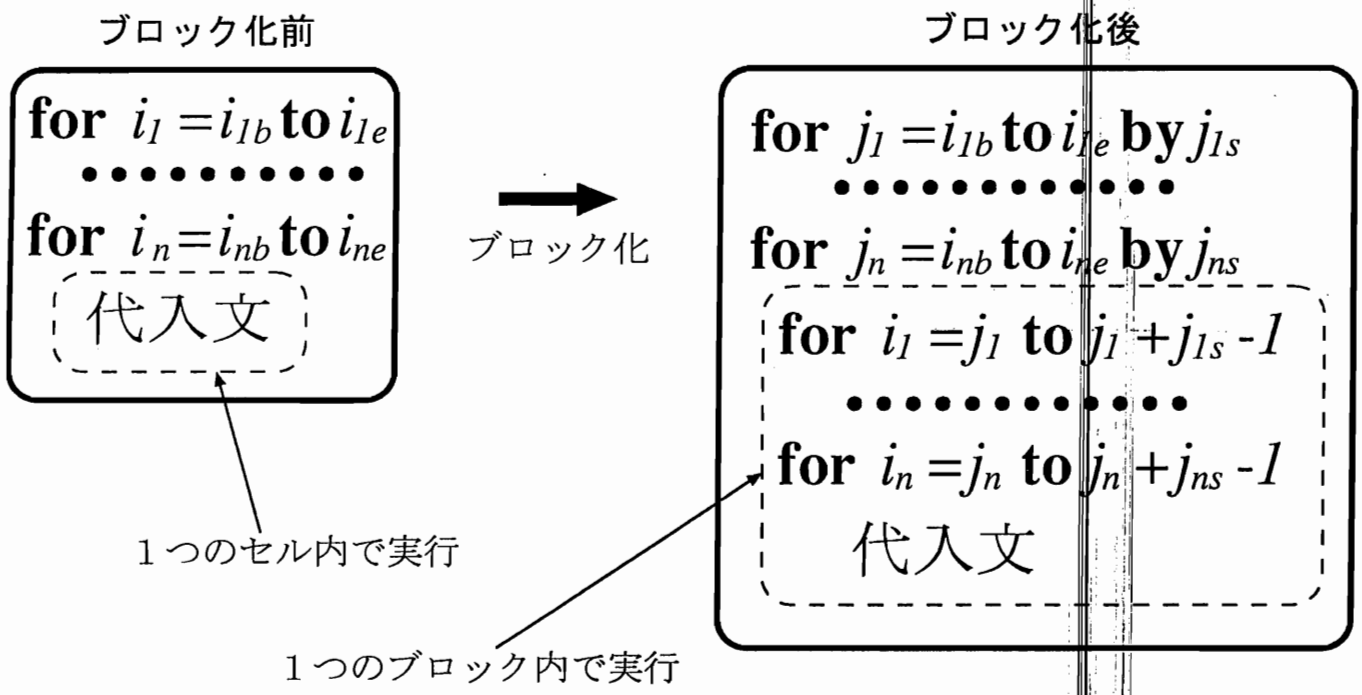
- 欠点：粗粒度の問題にしか適用できない

⇓

いくつかのイタレーションをまとめて
一つのブロックで実行 (ブロック化)

ブロック化 $\cdots n_{\mathrm{sys}}$の値を小さくする効果

ブロック化の方法



効率的なブロック化

- ブロック内での最大イタレーション数：

$$n_i = j_{1s} \times j_{2s} \times \cdots j_{ns}$$

- アレー全体のブロックの個数：

$$n_b = \left\lceil \frac{i_{1e} - i_{1b} + 1}{j_{1s}} \right\rceil \times \left\lceil \frac{i_{2e} - i_{2b} + 1}{j_{2s}} \right\rceil \times \cdots \times \left\lceil \frac{i_{ne} - i_{nb} + 1}{j_{ns}} \right\rceil$$

⇓

◎ イタレーションがダミーでない割合：

$$r_{ud} = \frac{n_{cal}}{n_i \times n_b} \quad \longrightarrow \text{大}$$

BR 法の適用例

ブロック化前

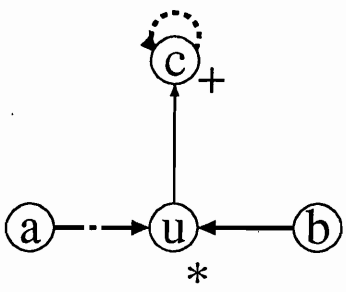
```
for i = 1 to 4
for j = 1 to 4
for k = 1 to 4
  c[i,j]=c[i,j]+
      a[i,k]*b[k,j]
```

ブロック化

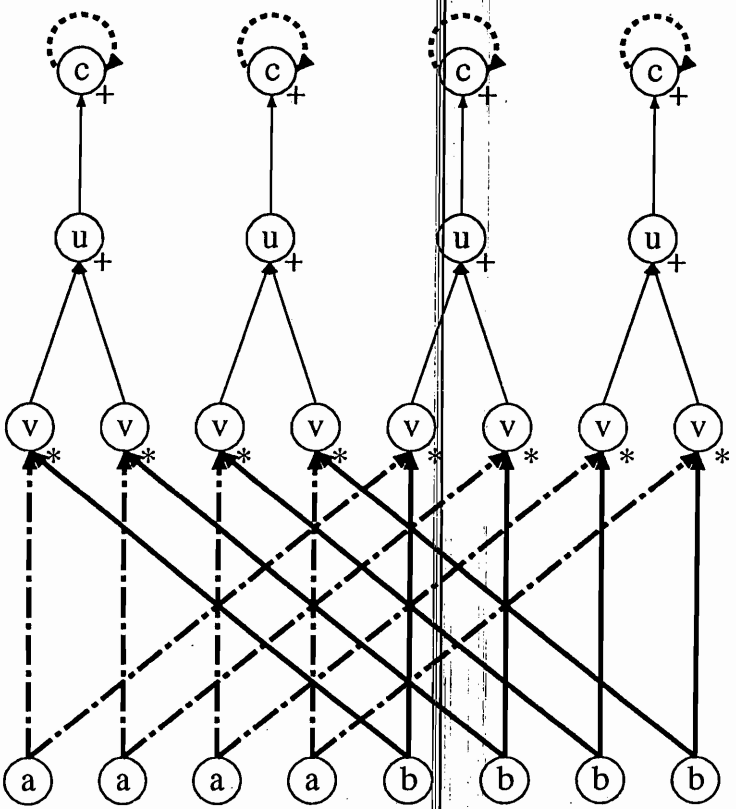
ブロック化後

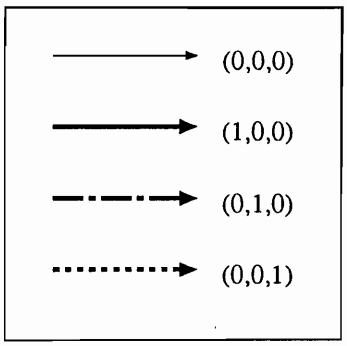
```
for i' = 1 to 4 by 2
for j' = 1 to 4 by 2
for k' = 1 to 4 by 2
 for i = i'  to i' +1
 for j = j'  to j' +1
 for k = k' to k' +1
  c[i,j]=c[i,j]+a[i,k]*b[k,j]
```

ブロック化前後の依存グラフ

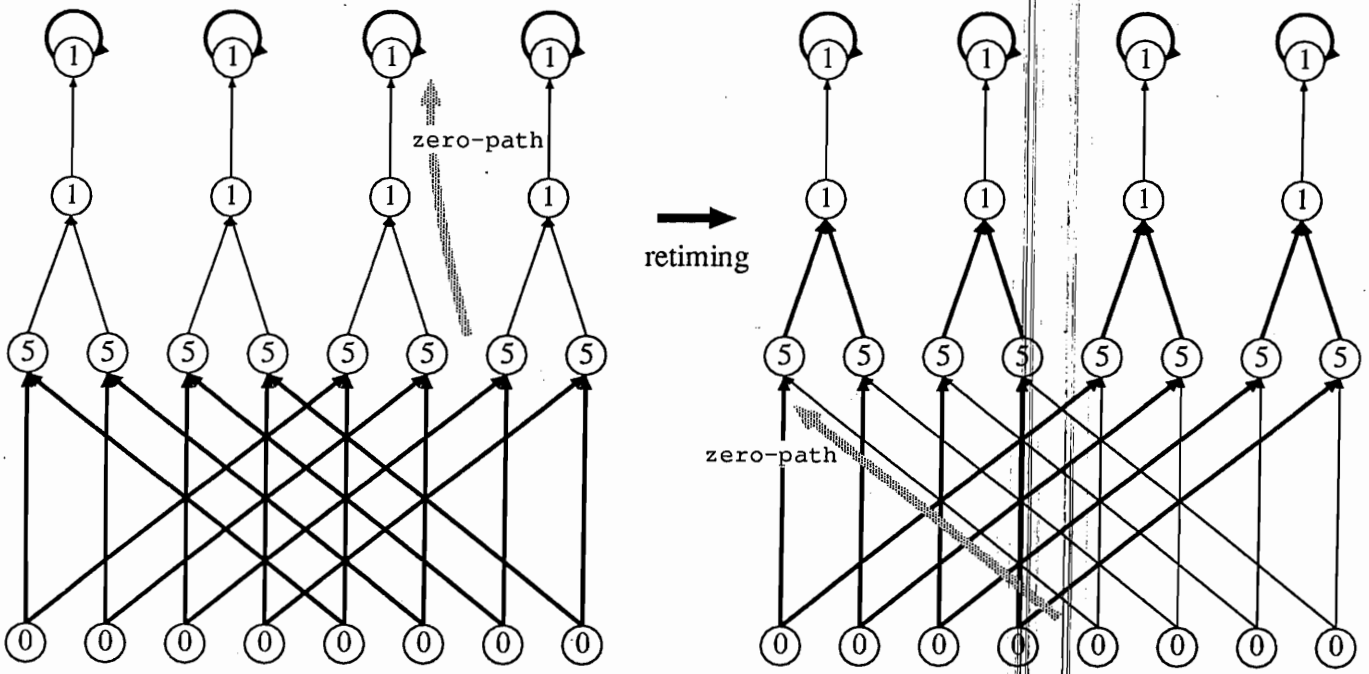


ブロック化





retiming 前後のレジスタグラフ



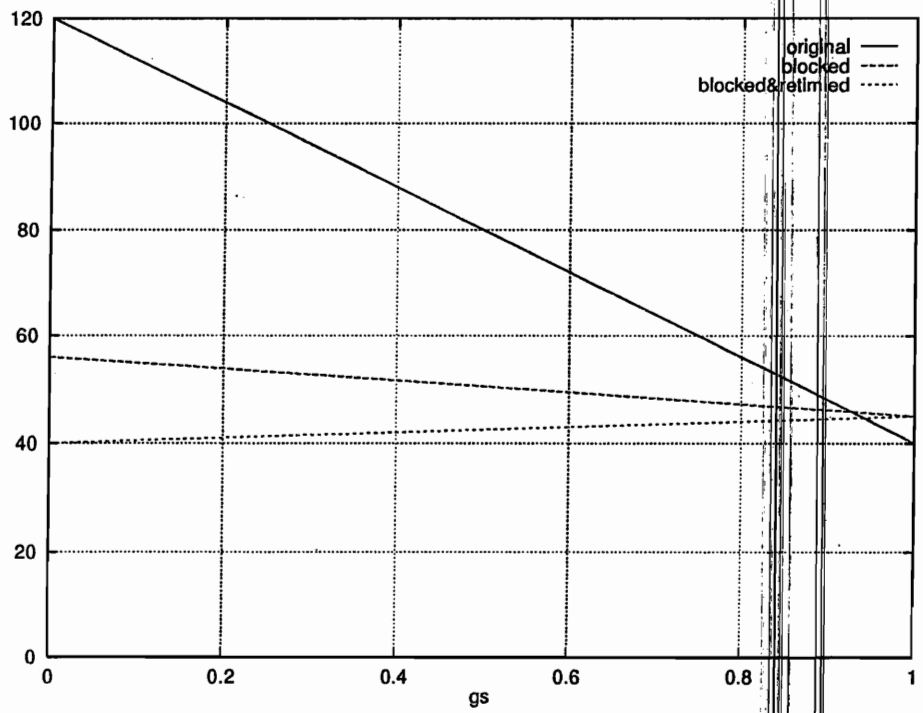
$\delta_v = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$　　$\Pi = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$　　加算：1　　乗算：5　　代入：0

計算時間

	original	blocked	blocked&retimed
n_{sys}	20	8	8
t_{sys}	6	7	5
$n_{\mathrm{sys}} \times t_{\mathrm{sys}}$	120	56	40

- ブロック化＋ retiming を行った場合：
  オリジナルの構成の 3 分の 1 の計算時間

評価関数による評価



空間：シリコン面積，時間：計算時間
$w_S = 10, w_t = 1$，1 ブロックは 1 セルの 4.5 倍のシリコン面積

第 4 章のまとめ

- **BR 法の提案**
  - **retiming** $\longrightarrow t_{sys}$: 小
  - ブロック化 $\longrightarrow n_{sys}$: 小
- 具体例によるシミュレーション，評価

5　結論・検討

● まとめ

- 評価値，評価関数に関する考察
- BR 法の提案，具体例でのシミュレーション

● 課題

- ブロック化での条件
  (管理機能⟶オーバーヘッド)