マイクロコンピュータ 44－3
（１９８７．３．１３）

ＬＳＩレイアウトデザインルールチェックを行うマルチマイクロプロセッサシステム

溝端 教彦　小野寺 秀俊　田丸 啓吉
京都大学 工学部

ＬＳＩのデザインルールチェック（ＤＲＣ）を高速化するため、階層的なマルチマイクロプロセッサ構成のＤＲＣ専用計算機を考えた。

システムの管理を行う１台のマイクロプロセッサに、ＤＲＣの並列処理を行う複数のマイクロプロセッサが高速な通信路で接続されているという構成を採る。ＤＲＣに用いられる演算の性質を利用し、マスクパターンを小領域に分割し、この各領域の処理を同時に行うという方法で並列処理を行う。領域間に重なりをもたせて分割することで、処理中の計算機間の通信を不必要にし、効率のよい並列化を実現した。以上の方法を用いたシステムを試作し、専用計算機の評価を行った。この結果、数十台以上の並列化が可能であることが分かった。

THE MULTI-MICROPROCESSOR SYSTEM FOR LSI LAYOUT DESIGN RULE CHECK

Norihiko MIZOBATA , Hidetoshi ONODERA and Keikichi TAMARU
Faculty of Engineering, Kyoto University
Yoshida-honmachi, Sakyou-ku, Kyoto-shi, 606 Japan

We designed a special-puopose computer which is composed of hierarchical multi-microprocessor. For high-speed DRC(Design Rule Check).

Multiple microprocessors which perform DRC are connected to a host microprocessor through a high-speed channel. This computer partitions a mask-pattern into small sections and checks design rule in parallel by processing each section at the same time. The partitioning with some amount of overlapping between neighboring section eliminates communication between microprocessors, which leads toefficient parallel processing. We made the experimental system adopting the above method, an devaluate our system. The result shows the possibility of parallel processing by more than several tens of microprocessors.

1．　はじめに

最近のＬＳＩ製造技術の進歩は目ざましく、百万にも及ぶ素子を集積したものが作られている。このためその設計は複雑さを増し、計算機を利用した設計や計算機による自動設計が行われている。しかし完全な自動化は現在ではまだ不可能であり、設計の各段階で人手が介入し、それが設計誤りの発生原因になっている。そこで、設計の過程において計算機による様々な設計検証が行われ、誤りの発生を防いでいる。

ＬＳＩ設計の最終的な検証工程であるレイアウト検証のひとつとしてデザインルールチェック（ＤＲＣ：Design Rule Check）がある。これはＬＳＩ上の素子を多角形の集まりとして表現したマスクパターンが、ＬＳＩ製造上の制約からくる規則を満足しているか調べるものである。この規則は、マスクパターン上の特定の図形の最小幅や、図形間の最小間隔という形で表されている。ＤＲＣは、マスクパターンから最小幅、最小間隔の検査対象になる図形を抽出し、これらの図形が必要な幅や間隔を満足しているか距離を測り調べる、という手順で行われる。図形の重なりや外形などを取り出す論理演算と、実際に図形間の距離、図形の幅を測り違反図形を取り出す距離計算が行える必要がある。従来、このようなＤＲＣは主に大型汎用計算機上のソフトウェアシステムとして実現されてきた[1][2]。しかし、処理対象のマスクパターンはデータ量が非常に多いため処理時間が長くかかり、その結果設計の効率が低下してしまう。このため、ＤＲＣの格段の高速化が望まれている。しかし、速度と精度を両立させたＤＲＣアルゴリズムを開発することは難しく、まだ十分なものができていないのが現状である。

そこで、大型計算機に代表される逐次処理方式を離れ、ＤＲＣを並列処理化する方法を考え、この方法に基づいたＤＲＣ専用計算機を用いることにより処理の高速化を達成することを考えた[3]。ＤＲＣの特徴として、マスクパターン上における処理の局所性と、処理対象データの多さがある。この特徴に注目し、マスクパターンを小領域に分割し、この分割されたデータ量の抑えられた小領域に対して並列にＤＲＣを行う。更に、分割された各領域が互いに隣の領域と境界部分で重なりを持つ分割方法を採用することにより、各領域のＤＲＣを全く独立に行うことができる[4]。このように独立性の高い並列処理が行えるため、効率の良い並列化が実現できる。

本論文ではこの方法に基づくＤＲＣ専用計算機として、入出力や処理全体の管理を行う１台のマイクロプロセッサに、分割されたマスクパターンの各領域のＤＲＣを並列処理するマイクロプロセッサが複数台接続されているという、階層構成を採ったマルチマイクロプロセッサシステムの設計と実験について述べる。並列処理を行う計算機間が独立であるため、計算機間の結合を簡単にでき、このため容易に並列数を増加させ速度の向上が図れるシステムとなった。

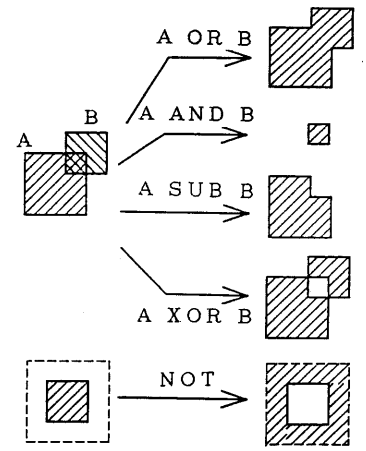
まず採用したＤＲＣ手法の説明を行う。次にＤＲＣに必要な機能をまとめ、そして今回検討を行ったＤＲＣの並列化の方法を説明し、それに基づくＤＲＣ専用計算機の構成と製作した実験システムについて述べる。次に実験システムによる処理実験結果について述べ、それにより本ＤＲＣ専用計算機の評価を行う。

2．　ＤＲＣの方法

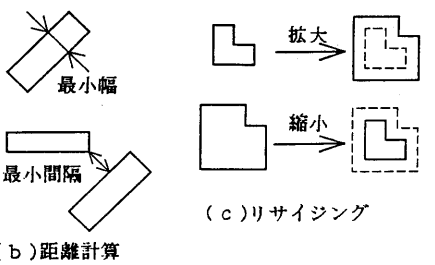
2．1　ＤＲＣに用いられる図形演算

・　論理演算（図２－１（ａ））

マスクパターンやこれと同様な図形の集合として表現されているデータ（例えば、マスクパターン



（ａ）論理演算



（ｂ）距離計算　　（ｃ）リサイジング

図２－１　ＤＲＣに用いられる図形演算

に図形演算を行った結果。以降このようなデータを図形集合データと呼ぶ。）に対して、これらの２層間の図形の重なりや図形の外形を取り出す演算である。この演算は１層または２層の図形集合データを入力とし、演算結果の図形集合データを出力とする。

・　距離計算（図２－１（ｂ））

図形集合データ内の各図形が必要な幅や間隔を満たしているかを調べる。この演算は図形演算データを入力とし、結果として最小幅や最小間隔の規則を違反している図形の辺を取り出し出力する。

・　リサイジング（図２－１（ｃ））

図形集合データ内の各図形を拡大、縮小する演算である。ここでいう拡大とは図形の回りに拡大幅だけ幅づけを行うものである。この演算は入出力とも図形集合データである。

２．２　採用するＤＲＣ手法

ＤＲＣ専用計算機において使用する図形演算手法として、演算対象図形の限定方法にワークリスト法、論理演算の方法にベクトル法、距離計算の方法にエッジ法を用いる[5]。ベクトル法は任意の角度の線分を含む図形を扱える演算方法である。エッジ法は図形の辺と辺の距離をユークリッド距離で測れる精度の高い方法である。これらの方法を採用し、大型計算機におけるＤＲＣシステムと同等の質のＤＲＣを実現する。

ベクトル法は図形がベクトルにより表現されていることを前提にした演算方法である。図２－２に図形のベクトル表現の例を示す。図形の各辺を図形内部を左にみるベクトルで置き換え、余分な情報である垂直ベクトルを取り去ったものである。これ以降では、水平ベクトルと平行で右向きをｘ方向、ｘ方向と直交し上向きをｙ方向と呼ぶことにする。ベクトルを構成するデータは、ベクトルの、左端点のｘ，ｙ座標、右端点のｘ，ｙ座標、向き、属する層の名前である。

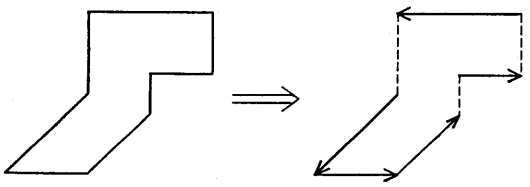


図２－２　図形のベクトル表現

次に採用するＤＲＣ手法について説明する。

・　論理演算

まず、図形上をｙ方向に延びるスイープラインで走査していき、ベクトルの端点か交点に出会うと停止させる。このスイープラインの位置と、一つ前のスイープラインの停止位置との間をスリットと呼び、このスリット内にかかるベクトルについて演算を行う。スリット内のベクトルを下から順にみていくと、図形の重なりを認識でき、演算結果として必要なベクトルが決まる。このようなベクトルだけを残していく。図形の重なりの認識と同時にベクトルの交点を求めておく。この交点とベクトル端点の座標から次のスイープラインの位置を決め、全図形上を走査していくことにより、図形全体の論理演算が行える。図２－３は、別の層に存在する２図形Ａ，ＢのＡＮＤ演算の例である。現在のスイープラインがＳｒとすると、Ｓｌとの間のスリットについて演算を行う。スリット内を下からみていけば、図に示してある重なり数が数えられ、重なり数が１１の部分を構成するベクトルを残せばＡＮＤ演算に必要なベクトルが残ることになる。論理演算は上記のように行われるため、演算を行うために注目しなければならない図形は１本のベクトルだけである。

・　距離計算

距離を正確に速く測定するためには、対象図形を効率よく限定しなければならない。距離計算は図形の全ての辺が対象となるため省略されている垂直ベクトルを復元する。対象の限定は次のようにＯＳＬ法と呼ぶ２次元限定を行う[5]。検出しなければならない最小距離をＤとする。まずスイープラインでマスクパターンをｘ方向に走査する。このスイープラインから左側へ幅Ｄの帯状領域を考えここをスリットと呼ぶ。このスリットから外れた直後のベクトル（測定対象ベクトル）とスリットにかかっているベクトルはｘ方向の距離がＤ以下である可能性がある。このスリット内

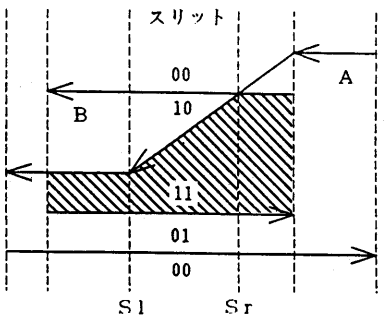


図２－３　論理演算（ＡＮＤ）

のベクトルについて、y方向にも同様のスイープラインとその下側に幅Dの領域（y方向スリット）を考え、このスリットでy方向に走査する。両方のスリットの交差内にかかるベクトルが、測定対象ベクトルからの距離がD以下の可能性のあるものとして限定される。この限定されたベクトルと、測定対象ベクトルの距離を正確に計算する。図２－４では、測定対象ベクトルとベクトルAが最小間隔Dを違反している可能性があるため距離を正確に計算する。距離計算は測定対象ベクトルとその周囲Dの範囲に注目すれば処理が行える。

・　リサイジング

論理演算と同様にスイープラインを走査していき、スイープライン上のベクトル端点を拡大後または縮小後の位置へ移動する。リサイジングを行うとベクトル端点はそのベクトルの周囲、拡大（縮小）幅だけの範囲に移動する。そのためこの範囲の図形に注目すればリサイジングは行える。

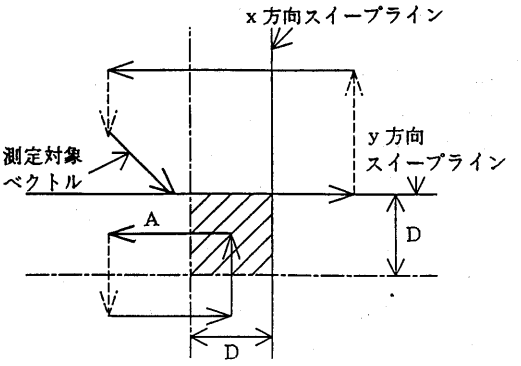


図２－４　距離計算（最小間隔D）

３．　マイクロプロセッサによるＤＲＣ専用計算機

３．１　ＤＲＣに必要な機能

まずマスクパターンの図形情報だけを対象にした基本的なＤＲＣを行うために必要な機能をあげる。

ＤＲＣへの入力は、ＤＲＣ対象のマスクパターンデータである。この数百Mbyte にもなる可能性のある大規模なデータを扱えなければならない。ＤＲＣの出力は、入力マスクパターン上のデザインルール違反の図形である。このデータもかなり大きなものである。このため大容量の外部記憶装置を持ち、高速なデータ転送が行える必要がある。

またＤＲＣを実現するために図形演算としてＡＮＤ、ＯＲ、ＳＵＢ、ＸＯＲ、ＮＯＴ、最小幅、最小間隔、拡大、縮小などが必要である[6]。マスクパターンは任意の角度をもつ線分で構成されており、この様なデータに対して上記の図形演算が行えなければならない。さらにＤＲＣを行うためには上の演算を順に組み合わせて用いる必要があるため、図形演算結果に対してさらに図形演算が行える必要がある。

距離測定の精度を上げ疑似エラーを防ぐため、図形間の距離をユークリッド距離で測る必要がある。このため、浮動小数点演算が必要になり、これを高速に行うために専用のプロセッサを持つことが望ましい。

３．２　並列処理方法

ＤＲＣの処理速度を向上するために並列計算機で処理することにする。扱うデータ量が非常に多いため、計算機間のデータ移動に多くの時間を必要とし、データ移動時間が処理速度を制限することになる。そこで効率の良い並列処理の実現のためデータ転送をできるだけ抑えた並列化の方法を採ることにする。図形演算は比較的簡単な処理で実現できるため、並列計算機はマイクロプロセッサを用いて構成することにする。

ＤＲＣを並列処理化する方法として、マスクパターンを小領域に分割し、この各領域の処理を並列に行う方法が適していると考えられる。これは、２．２で述べたように、ベクトル法で図形演算を行うためには現在処理対象になっているベクトルと、そのごく近傍にあるベクトルに注目すればよいという点に基づいている。このような方法を採ると、各領域に対する処理がほぼ独立に行え、効率の良い並列処理化が実現できる。分割することによりデータ量が小さく制限されデータの複雑さも減少するため、領域毎の処理をマイクロプロセッサで構成した計算機に割り当て並列処理を行うことが考えられる。分割の一般的な方法としては、マスクパターンを垂直、水平、小矩形などの領域に分割する方法が考えられている[7]。

本計算機では１台の計算機に対してマスクパターンを次のように２段階に分割する方法を採る[8]。

i)　y方向に横切る線で分割（垂直分割）

採用したＤＲＣ手法より分割線も１本のスイープラインと見なせるため、分割したことによる処理内容の変化はない。このような分割線で分割した各領域は別々の計算機に割り付け並列に処理を行う。分割線にかかるベクトルは、分割線で切断され２つに分かれてしまうためベクトルデータが増加してしまう。

ii)　x方向に横切る線で分割（水平分割）

採用した手法ではスリット内を下から順に処理を行うため、水平分割を行った場合、上側の領域の処理を行うためには分割線上の図形の情報（重なり数など）が必要であり、これは下側の領域の処理が終らないと得られない。このため、水平分割された各領域は1台の計算機が逐次処理を行うことにする。このようにすれば、分割線上の図形の情報は計算機内に残しておけば良いためデータ転送を行わずに処理ができる。分割線で切断されデータの増加原因になる可能性のあるベクトルは斜めベクトルだけであるため、分割によるデータの増加は垂直分割の場合より少ない。さらに、スリットの高さが低くなり、スリット数が減少するため、処理の高速化が期待できる。

分割線で単純に分割した場合、分割線付近の図形同士の距離が正確に測れず、そのため疑似エラーや、距離違反の検出の失敗（図３－１）が生じる。また分割線付近の図形のリサイジングが正確に行われない。このように処理が不正確になるのは、距離計算の場合は分割線から検出すべき最小距離Ｄだけの範囲であり、リサイジングの場合はリサイジング幅Ｗの範囲である。そこで各領域を（Ｄ＋Ｗ）だけオーバサイズし、隣の領域と重なりを持つような分割を行う。（以降、この重なり部分をオーバラップと呼ぶ）オーバサイズした範囲に含まれる図形に対して処理しておけば、不正確な処理結果が含まれているのはオーバラップ部分だけとなり、処理終了後にオーバラップ部分を除去すればマスクパターン全体が正確に処理されることになる。さらにこのような分割方法を採ることにより、分割された各領域の処理は全く独立に行うことができる。

分割方法をまとめる。まず並列処理のために並列度だけに垂直分割を行う。次に各領域のデータ量をマイクロプロセッサによる計算機で扱える程度に制限するために水平分割を行う。さらに分割線付近の処理を正確に行い、かつ各領域を独立に処理するために、オーバラップさせて分割を行う。分割方法の概念図を図３－２に示す。

以上のような分割方法により並列処理を行うためには、次のように処理を進めればよい。まず、マスクパターンを小領域に分割する。各領域別々にＤＲＣに必要な図形演算を並列に行う。演算結果のオーバラップ部分を除去し、正確なＤＲＣ結果だけを得る。

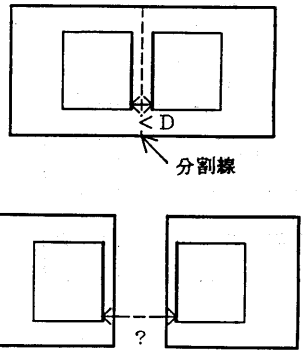


図３－１　検出できない距離違反

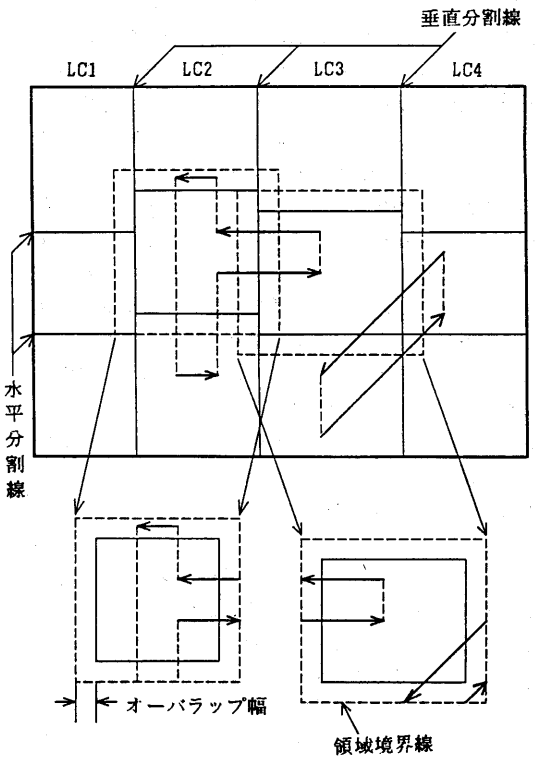


図３－２　分割方法の概念図

３．３　ＤＲＣ専用計算機のシステム設計

３．２で述べたＤＲＣの並列処理の流れから考え、ＤＲＣ専用計算機の基本的な構成は、図３－３に示すような２層の階層構成を採ることにする。処理

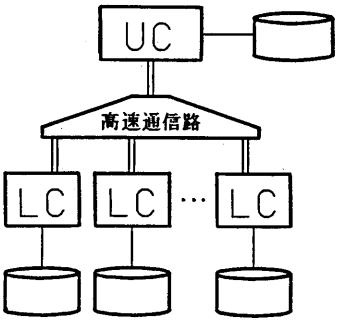


図３－３　計算機の構成

全体の制御や、システム全体の入出力の管理やデータの管理を行う１台の計算機（以降、ＵＣ：Upper level Computerと書く）と、それに接続された、並列的に処理の行える仕事を実行する複数台の計算機（以降、ＬＣ：Lower level Computerと書く）からなるシステムとする。３．２で述べた処理の流れに従い各部の基本的な動作を説明する。まずＵＣは、外部記憶装置からマスクパターンデータを読みだしＬＣに渡す。ＬＣは、ＵＣから送られてきたデータの中から担当領域のデータだけを取り出す。このデータに対して図形演算を行い、オーバラップ部分を除去し、結果をＵＣに返す。このようにして、ＤＲＣの並列処理が行える。

上でＤＲＣ専用計算機の基本的な構成について述べたが、ここではさらに具体的に各部の機能などについて説明する。システム全体は、１台のＵＣに、複数台のＬＣが結合されている。ＵＣとＬＣの間は多量のマスクパターンデータが転送されるため、高速通信路で接続されている。以下では、ＵＣ、ＬＣ、高速通信路に分けて説明する。

・　ＵＣ

マスクパターンデータを記憶しておくためのハードディスクや磁気テープなど外部記憶装置と入出力装置をもち、システム全体の制御や、ＤＲＣの処理やマスクパターンデータの管理、システム外部とのインターフェースをとるなどの仕事を行う。ハードディスクの容量はマスクパターンデータが格納できる程度（数百Mbyte）必要である。ＤＲＣの処理では、外部記憶からマスクパターンデータを読みだしＬＣに転送する。そして、ＬＣから送られてくるＤＲＣ結果をディスクに記憶する。マスクパターンデータは非常に多量であるため、高速にディスクからデータを読みだし、ＬＣに転送できなければならない。

・　ＬＣ

担当領域のマスクパターンデータやＤＲＣの途中で生成された図形データを記憶するために外部記憶としてハードディスクを持つ。これらのデータが記憶できる程度の容量が必要である。この量は並列度により異なり一概にはいえない。この計算機が実際にＤＲＣを行う。ＵＣからマスクパターンを受け担当領域のデータだけ取り出し、これに対して処理を行い、結果をＵＣに送り返す。ＬＣの数は１６台から１２８台程度を考えている。ＬＣに用いるマイクロプロセッサは次のようなものでなければならない。マスクパターンデータ上のベクトルの座標を表現するためには３２bitの数値が必要であるため３２bit のデータを効率よく扱えるものでなければならない。また多量のデータに対して効率よくアクセスするためには広い線形のアドレス空間を持つものであることが必要である。

・　高速通信路

ＵＣとＬＣ間の通信はマスクパターンデータの転送が大部分であり、この多量のデータを高速に転送できなければならない。このため多量のデータを連続に高速に転送できる通信路である必要がある。また、ＵＣからは全ＬＣへ同じデータを同時に転送を行う。そこで、通信路はＦＩＦＯによるバッファを２本持ちこれを切り換えて用いることにより通信の高速性を確保している。さらに、このバッファをＬＣ１台に付き１つ設けることにより、同時に同じデータを全ＬＣへ転送することを実現している。

ここでは、上で述べた計算機でＤＲＣを行う場合の処理の流れを説明する。ある１つの規則を検証する場合について説明を行う。

①　ＵＣはハードディスクからマスクパターンデータを読みだし、それを全ＬＣに転送する。

②　各ＬＣは転送されてきたデータの中から担当領域の図形データを選び出し、ディスクに記憶する。

③　規則を検証するために必要な全ての層のマスクパターンデータがＬＣに送られるまで①、②を繰り返す。

④　各ＬＣは、担当領域の図形データのうち今から行う図形演算に必要なものをディスクから読みだし、そのデータに対して図形演算を行い、演算結果をディスクに記憶する。

⑤　④を検証に必要な演算が全て行われるまで繰り返す。

⑥　各ＬＣは、検証結果のデータから不正確な結果が含まれているオーバラップ部分のデータを除去する。

⑦　各ＬＣは、ＤＲＣ結果をＵＣに転送する。

⑧　ＵＣはＬＣから送られてきたＤＲＣ結果をディスクに記憶する。

以上の処理を行うことで、上記の計算機でＤＲＣの並列処理が行える。

３．４　実験システムの製作

３．３で述べたＤＲＣ専用計算機の評価を行うために、実験システムを製作した。ここではこの製作したシステムについて説明する。

・　ＵＣ

ＣＰＵにＭＣ６８０００を用い、ＯＳとしてＣＰ／Ｍ－６８Ｋがのっているマイクロコンピュータシステムを利用した。外部記憶装置として４０Mbyte のハードディスクを持つ。ＬＣのプログラムをＵＣから転送することによりＬＣの動作の制御を行っている。

・　ＬＣ

ＣＰＵとしてはＭＣ６８０００（クロックは８MHz）を用いた計算機を製作した。このプロセッサは、内部構成が３２bit データを扱えるようになっており、１６Mbyte の線形なアドレス空間を持っている。メモリは１Mbyte を実装した。ＬＣの動作はＵＣから送られてくるプログラムによって決定される。この実験システムでは構成を簡単にするため、ＬＣは外部記憶装置を持っていない。ＬＣが外部記憶を使う必要のある場合は、そのデータをＵＣに送り、代わりにＵＣのディスクに記憶させることにした。そしてＬＣがディスクからデータを読みだしたい場合は代わりにＵＣが読みだしＬＣに転送する。製作したＬＣは２台である。

・　高速通信路

１６Kbyte のＦＩＦＯバッファを２本持った通信路を、ＵＣ・ＬＣ間に設けた。通信を行う際、この２本のバッファをそれぞれ送信側からの書き込み用と受信側からの読みだし用に使う。このようにすることで、送信側、受信側どちらの計算機も、他方が通信路にアクセスしているかどうかに関係なくアクセス可能となり、高速なデータ転送が行える。この通信路へのアクセスは、６８０００からメモリアクセスと同じ速度で行える。この通信路を使いプログラムによりデータ転送を行う場合約７００Kbyte/sec で転送できる。

・　ＤＲＣプログラム

処理の流れは、３．３で述べたものとＬＣがディスクを持たないこと以外は同じである。実現した図形演算は、ＡＮＤ、ＯＲ、ＳＵＢ、ＸＯＲ、最小間隔、最小幅の検査であり、図形演算の手法は２．２で述べたものを用いている。ベクトルの座標は３２bitの数値で表現している。

４．　実験と検討

製作した実験システムを使い、実際のマスクパターンを領域分割し図形演算を行ってみた。

４．１　分割数とデータの増加

並列化のためにマスクパターンを分割するとデータ量は増加してしまう。データ量が増加するということは処理時間が増えることを表しており、並列度を増すことによる処理時間の短縮の効果が小さくなることを意味している。そこで分割によりどの程度データが増加するか調べてみた。分割数とデータの増加の関係を見積り、実際のマスクデータを分割した場合と比較してみる。ここで簡単化のために斜めベクトルは含まないとする。また、マスクパターンデータは全体に均一に分布しているとする。

・水平分割

斜めベクトルがないと仮定したので、（水平分割によるデータの増加）＝（オーバラップ内に含まれるベクトル数）といえる。ここで、Ｄ：オーバラップ幅、Ｈ：マスクパターン存在領域のｙ方向幅、ｎｖ：分割前のベクトル数、ｎｃ：水平分割線本数、とおくと、分割により増加するベクトル数は、

$$2 \cdot D \cdot nc \cdot nv / H \qquad (4.1)^*$$

となると考えられる。図４－１が（４．１）式で計算した結果（実線）と実際にマスクパターンを分割した結果である。（４．１）式は実際のマスクパターンを分割した場合のよい近似を与えるといえる。

・垂直分割

垂直分割によるデータの増加は、オーバラップ部分に少しでもかかるベクトルが２本に切断されたり２つの領域に属したりするためである。このことより、

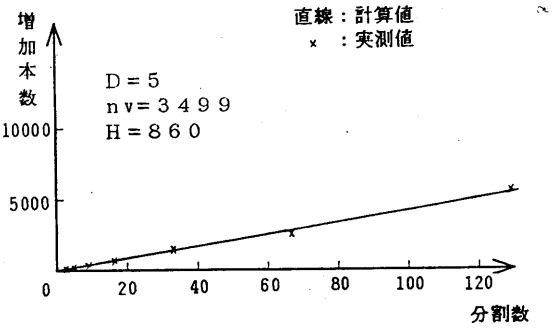


図４－１　水平分割によるデータの増加

（垂直分割によるデータの増加）＝（オーバラップ部分にかかるベクトル数）といえる。ここで、Ｄ：オーバラップ幅、Ｌ：マスクパターン存在領域のｘ方向幅、ｎｖ：分割前のベクトル数、ｎｃ：垂直分割線本数、ｌ：ベクトルの平均長、とすると、分割により増加するベクトル本数は、

$$(2\cdot D+1)\cdot nv\cdot nc/(L-l) \qquad (4.2)^{**}$$

となると考えられる。（４．２）式による計算値と、実際のマスクパターンを分割した結果をグラフに表したものが、図４－２である。このグラフから分かるように（４．２）式は垂直分割によるデータの増加をよく近似している。

図４－１と図４－２を比較すると垂直分割によるデータの増加の方が多いことが分かる。

---

*)･･･分割線１本につき両側にオーバラップがＤだけ取られるため分割線１本当りのオーバラップ部分の幅は２・Ｄである。分割線数がｎｃであればオーバラップ部分の幅の合計は２・Ｄ・ｎｃとなる。ここでベクトルが均一に分布しているとの仮定から単位ｙ方向幅に含まれるベクトルの数はｎｖ／Ｈとなり、オーバラップ部分に含まれるベクトルの数は（４．１）式のようになる。

**)･･･オーバラップ部分は分割線の両側に幅Ｄずつある。ベクトルが均一に分布しているとの仮定から、幅２・Ｄのオーバラップ部分に少しでもかかるベクトルは、ベクトルの中点がオーバラップ部分とその外側１／２以内に存在するものである。すなわち幅（２・Ｄ＋１）内に中点を持つベクトルである。ベクトルの中点の存在範囲は（Ｌ－１）であるため、１本の分割線のオーバラップ部分にかかるベクトル数は（２・Ｄ＋１）・ｎｖ／（Ｌ－１）である。故に分割線がｎｃ本あれば、ベクトルの増加は（４．２）式で表される。

４．２　並列度と処理時間

並列度を増加させた場合の処理時間の短縮の効果を調べてみる。まず実験システムについて動作をシミュレートし処理時間を調べ、それと実験結果と比較してみる。次に３．３で設計したシステムについて動作のシミュレートを行い調べてみる。図４－３に実験システムの場合を示す。マスクパターンの存在範囲（以降、チップ幅と呼ぶ）とベクトルの平均長の比が、実験をしたマスクパターンと同じ場合と、チップ幅が１０倍になった場合（マスクパターンの規模が大きくなった場合を想定）のそれぞれについて、負荷が全て

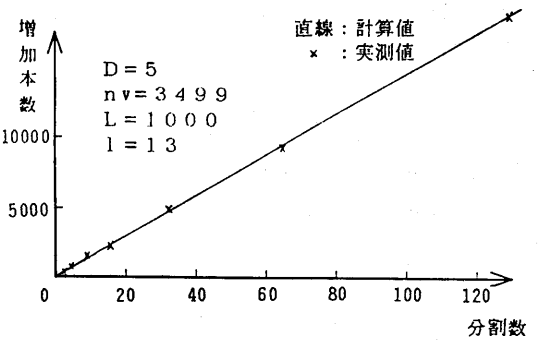
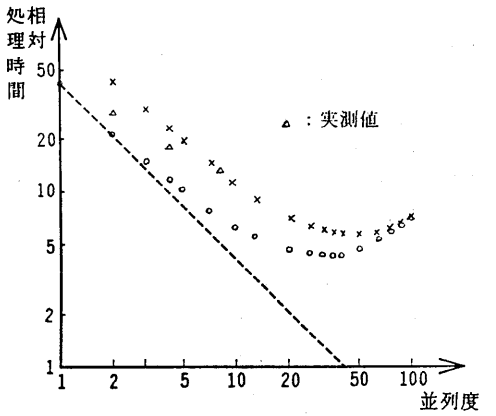
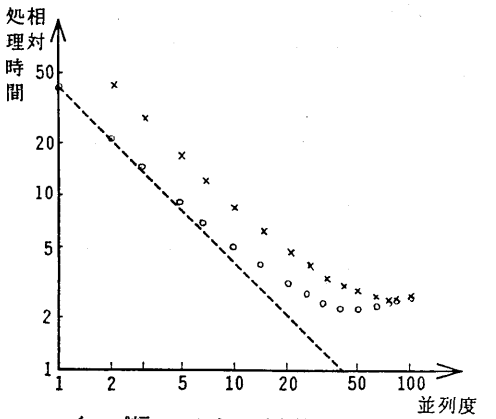


図４－２　垂直分割によるデータの増加



チップ幅：ベクトル平均長＝1000：13



チップ幅：ベクトル平均長＝10000：13

○：負荷均一
×：負荷偏り
オーバラップ幅＝5

ベクトル１本当りの演算時間に対するディスクアクセス時間の比　＝４１

図４－３　並列度と図形演算時間（実験システム）

のＬＣに均一にかかる場合と偏りのある場合について調べた。図に実測値も示す。実測結果が妥当な位置にきておりシミュレーションの結果は適当なものであるといえる。図をみると並列度が約５０以上になると並列度を増しても処理時間は増加してしまう。これはＵＣのディスクアクセスがボトルネックになっているためである。実験システムの場合の並列度の限界はディスクアクセス速度と演算速度のかねあいで決まる。図４－４は３．３で設計したシステムの場合である。図４－３と同じ条件で調べた。ディスクが各ＬＣにあるため並列度を増加していくと処理時間は短くなっていくが、次第に飽和する。実験システムの場合と比較すると並列化の効率はよいといえる。また図の（ｂ）のグラフより、マスクパターンの規模が大きい方が並列化の効果が大きく並列度を大きくできることが分かる。

４．３　実験システムの処理速度

実験システムではＬＣのプロセッサとしてＭＣ６８０００を用いた。このプロセッサがどの程度の処理能力を持っているか、マスクパターンの論理演算を行いその演算時間を測定してみた。表４－１に結果を示す。１秒間に処理できるベクトル数は約７～３０本である。ＬＣの図形演算プログラムとほぼ同じプログラムが大型計算機上で実現されておりその処理速度は３００本／秒～９００本／秒である。この処理速度は同じマスクパターンを処理したわけではないため一概には比べられないが、マイクロプロセッサでもかなりの処理能力が得られることが分かった。ここで、図形演算速度と分割処理速度を比較してみる。分割処理速度は実際のマスクパターンを用いて調べてみると約７５０ベクトル／秒であった。図形演算速度と比べるとかなり速く、ＤＲＣを行うためには図形演算は何度か繰り返されることを考えると、ＤＲＣ全体にかかる時間に対して分割処理時間は無視できるといえる。

表４－１　実験システムの論理演算時間（１層のＯＲ）

	ベクトル数（本）	演算時間（秒）
レイヤ１	３４９９	１６０
レイヤ４	２２９３	７１
レイヤ５	１１６８	３５
レイヤ６	２８４３	４１４

４．４　システムの評価

３．２に述べた並列処理の方法を採ることにより、各ＬＣはＵＣからマスクパターンを受け取ると、検証結果が出るまで全く独立に動作することができる。このため並列度を増した場合も計算機間の通信による処理速度の低下はない。並列度を増加させた場合に処理速度の向上の効率が悪くなる原因は分割によるデータの増加によるものである。図４－４から分かるようにマスクデータの規模により効率よく並列化できる並列度の限界が決まる。処理速度の向上がＬＣ１台の場合に比べて（並列度／２）倍になるような並列度を図４－４から読み取ると、（ａ）のような小規模なマスクパターンの場合で約５０、（ｂ）のようにベクトルの平均長がチップ幅に比べて非常に小さい場合は１００以上にすることができると考えられる。

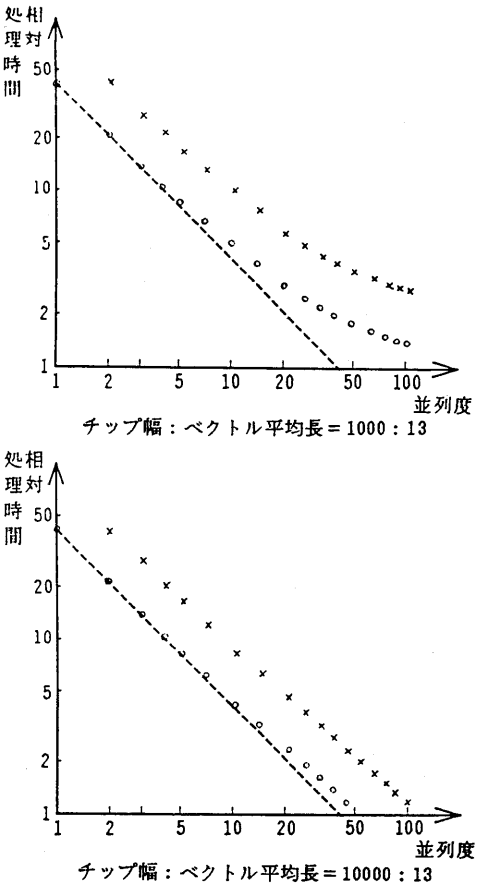


図４－４　並列度と図形演算時間

5. まとめ

ＤＲＣの高速化を図るために階層的なマルチマイクロプロセッサ構成のＤＲＣ専用計算機を考えた。

１台のＵＣに複数のマイクロプロセッサによるＬＣが高速通信路により結合されている。ＵＣはシステム全体の管理を行う。ＬＣはＤＲＣを構成する図形演算を並列に行う。並列処理化は、マスクパターンを分割し分割された各領域のＤＲＣを同時に行うという方法を採った。マスクパターンを垂直方向に分割することにより並列処理を実現し、水平方向に分割することでマイクロプロセッサで扱える程度のデータ量にしている。分割の際領域同士に重なり部分を持たせることにより、分割が原因で不正確な処理が生じることを防ぎ、各ＬＣが全く独立に処理を行えるようにした。計算機間の通信はＤＲＣの開始時と終了時だけである。このように計算機間の通信が少ないため、処理速度が通信により制限されず、効率の良い並列化が可能となる。並列度の限界はマスクパターンの規模によるが、かなり小規模なマスクパターンを処理対象にする場合でさえ数十程度まで効率よく並列化できる。

実験的にマイクロプロセッサにＭＣ６８０００を用いたシステムを作りＤＲＣに使われる図形演算を行った。この程度のマイクロプロセッサでもかなりの処理能力を持っていることが分かった。しかしＤＲＣ専用計算機として大型計算機上のＤＲＣシステムを越える高速性を実現するには、さらに高速なプロセッサを使用する、図形演算処理をハードウェアで行う、ＬＣ内でさらに並列処理化を行う、などの方法を採ることが必要であると考えられる。

謝辞

本研究を行うにあたり、日頃からご指導いただいている田丸啓吉教授、小野寺秀俊助手、安浦寛人助教授に感謝いたします。

参考文献

[1] K.Yoshida,T.Mitsuhashi,Y.Nakada,T.Ogita and S.Nakatsuka: "A Layout Checking System for Large Scale Integrated Circiuts", Proc. of the 14th DA Conf., pp322-330, (1977)

[2] A.Tsukizoe,J.Sakemi,T.Kozawa and H.Fukuda: "MACH: A High-Hitting Pattern Checker for VLSI Mask Data", Proc. of the 20th DA Conf., pp726-731, (1983)

[3] 田丸、小野寺：”ＬＳＩデザインルールチェック専用計算機のシステム設計”、信学論（Ｄ）、Vol.J69-D No.4、pp514-523、(1986)

[4] G.E.Bier,A.R.Pleszkun: "An Algorithm for Design Rule Checking on Multiprocessor", Proc. of the 22nd DA Conf., pp299-304, (1985)

[5] 三木、小野寺、田丸：”レイアウト設計検証アルゴリズムとそれを実現するハードウェアアーキテクチャの考察”、信学技報CAS85-163、pp1-6、(1985)

[6] 築添、小澤：”レイアウト設計検証ＣＡＤと手法”、情報処理、Vol.25 No.10、pp1106-1111、(1984)

[7] 木村、榎原、中野、中西：”設計規則検証の並列化に対する考察：データ構造と論理演算”、信学技報CAS86-206、pp25-30、(1986)

[8] 溝端、小野寺、田丸：”ＬＳＩデザインルールチェックの並列処理化の検討”、電気関係学会関西支部連合大会、G12-1、(1986)