解　説

BDD（二分決定グラフ）

6. BDDの並列処理技術†

木　村　晋　二††

1. はじめに

論理関数を効率良く表す方法として，近年BDD（二分決定グラフ）が注目され，広く研究されている[1)~3)]．BDDは真理値表，キューブ表現などのほかの表現法に比べ，多くの実用的な論理関数に関して非常に効率良く関数を表現できるので，BDDの処理に要する時間は比較的少なくて済むことが多い．それでもなお問題によっては，問題のサイズの指数に比例する記憶量と処理時間を必要とする[4)]．また一般に応用によっては多くの処理時間を要することが知られている[5),6)]．その一方で，並列計算機が実用化され，種々の並列計算機が使用できる環境が整いつつある．そこでここでは，BDDの処理の並列化による処理の高速化について述べる．並列計算機の分類や並列アルゴリズムの一般的な解説は文献7)や文献8)にあるので参照されたい．

並列化の結果の一例を**図-1**に示す（共有記憶をもつ並列計算機での並列化[9)]）．図-1(a)は縦軸に所要時間(秒)，横軸にプロセッサ数をとったもの，(b)は，縦軸に1台での実行時間を n 台での実行時間で割った高速化率，横軸にプロセッサ数をとったものである．25台で15倍弱の高速化を達成している．

2. BDD処理の基本操作とその並列化

以下ではBDDの処理のうち，論理式により表された論理関数に対応するBDDを求める問題について考察する．これは，**図-2**に示すような論理回路の出力fのBDDを構成する問題に対応する．通常，以下の手順で行われる．

1. 入力変数 a，b，c に対応するBDDを構成し，
2. 回路中の論理演算結果に対応するBDDを順に構成する．

図-2の回路に対し，実行可能な演算を順に並べると，**図-3**のような演算列が得られる．通常はこれを順次実行してゆく．

BDD処理の並列化の場合には，nd, nc, eなど，複数ある論理演算のうちのいくつかを同時に実行できる（**図-4**）．ただし，このような並列性の抽出だけでは不十分で，論理演算そのものを並列に実行する必要がある．

BDD上での二項論理演算の実行過程は，基本的に以下の二つのフェーズに分かれる．

1. 展開のフェーズ

二つの論理関数を表すBDDの根の節点の対から，下位の節点の対を求めてゆく．

2. 最小化のフェーズ

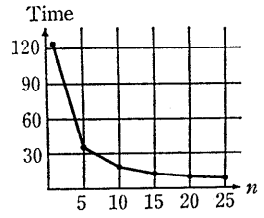
等価な節点，および冗長な節点を削除して，最小のBDDを構成する．

図-2の最後のOR演算の部分の展開および最小化の様子を**図-5**に示す．幅優先で展開すると，演算対象の論理関数を表すBDDの初期節点の対(A, D)から，(A, D)，(0, D)，(B, D)，…，(0, 1)の順で節点対が生成される．(A, D)を変数 a で(Shannon)展開した結果が(0, D)と(B, D)である．節点対(0, D)などはOR演算を行った結果としてDそのものとなるので，それ以上の展開は不要である．展開のフェーズでは，節点の対に対応して演算結果のBDDの節点を生成すればよい．この節点は，対に対応する演算の結果として通常ハッシュベースのキャッシュに記憶され，同じ節点対に対する処理が行われたかどうかの判定に用いられる．

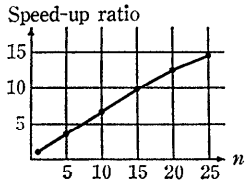
最小化のフェーズでは，展開の逆順に，等価な

† Parallel Binary Decision Diagram Manipulation by Shinji KIMURA (Graduate of Information Science, AIST Nara).

†† 奈良先端科学技術大学院大学情報科学研究科



(a) Execution time.



(b) Speed-up ratio.

図-1 Evaluation on Sequent S-81 for 10-bit multiplier.

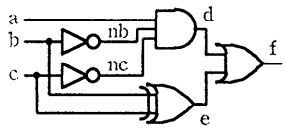


図-2 Example circuit.

```
nb=NOT(b)
nc=NOT(c)
e =EXOR(b, c)
d =AND(a, nb, nc)
f =OR(d, e)
```

図-3 An operation sequence.

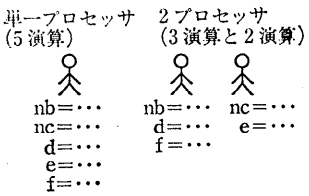
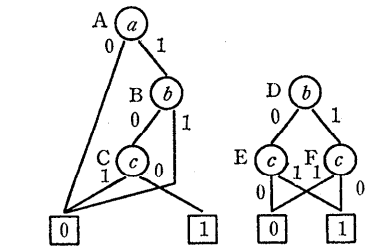


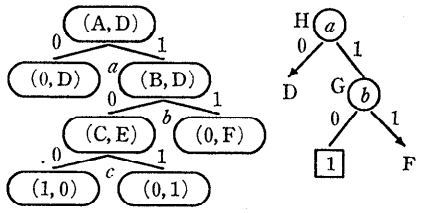
図-4 Parallel execution of operations.

節点および冗長な節点があるかどうかを検査していく．新しく生成した節点と等価な既存の節点を検索するために，通常節テーブルという（チェイン付き）ハッシュテーブルを用いる．展開のフェーズで用いるのはキャッシュであるので，同じ節点対に対応して異なる節点を生成することもあるが，最小化のフェーズでそのような節点も完全に取り除かれる．

単一のプロセッサでは，深さ優先で，展開と最小化を同時に行うことが多い．一方，並列実行の場合には，これら節点対の処理において，幅優先で並列実行できる対を生成することが多い(**図-6**)．



(a) BDD's for $a\bar{b}\bar{c}$ and $b\bar{c}+\bar{b}c$.



(b) Expansion. (c) Minimization.

図-5 Expansion and minimization.

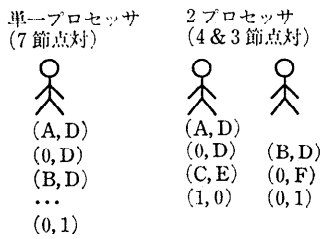


図-6 Parallel execution of an operation.

3. MIMD 型計算機を用いた並列処理方式

ここではまず複数のプロセッサを結合したMIMD 型の並列計算機での並列処理方式について述べる．現在 MIMD 型の並列計算機は商用のワークステーションの世界でも一般的になりつつあり，並列化アルゴリズムの実用性は非常に高い．

3.1 共有記憶型並列計算機での処理方式

共有記憶型の並列計算機で BDD の処理の並列化を行う手法は，Kimura らにより提案されている[9),13)]．文献 13) では幅優先の手法を Encore Multimax で実現した結果が，また文献 9)では深さ優先の手法を Sequent S-81 で実現した結果が示されている．いずれも 15 プロセッサで 10 倍程度の高速化を達成している．

共有記憶をもつ並列計算機は，**図-7** に示すように複数のプロセッサが高速のバスを介して共有

主記憶へ結合されている．主記憶への書き込みに関しては，同時書き込みによるデータの損傷を防ぐために，通常ロックという操作をほかい，ほかのプロセッサから書き込めない状態にしてから行う．なお，読み出しは複数のプロセッサで同時に(ロックなしで) 行えることが多い．

並列化では，先に示したように，複数ある演算を並列実行すると同時に，Shannon 展開を用いて，一つの演算を複数のプロセッサで並列実行する．複数演算の並列実行では，図-3 の OR(d, e) などの演算で，演算子 d や e の演算がほかのプロセッサで実行される可能性があるので，これらの計算の終了を待たねばならない．

一方，一演算の並列実行では，幅優先の展開を適当な段数だけ行い，そこから先を異なるプロセッサで行う．展開を2段行えば，(理想的には) 4つの節点対が得られ，それらから下の展開と最小化を並列に実行すればよい．ただし，並列実行の後，4つの部分問題の結果から，最小化の部分を行う必要がある．図-5 の演算で，(A, D) を4つに分ける（すなわち変数 a, b で展開する）と，(0, D)，(0, D)，(C, E)，(0, F) が得られる（一部無駄な展開を含む）．(A, D) から展開後の各対を求める部分も各々のプロセッサで行う．論理演算をいくつに分割するかについて，文献9）では，外部入力からの最大の段数で各論理演算をレベル化し，レベル毎の演算の数に応じて展開する段数を決定する手法を提案している．

Sequent S-81 (80386 (16 MHz)×28, 86 MB 主記憶) を用いた評価を示す．先に示した図-1 は，10 ビットの乗算器 (20 入力，20 出力，680 演算，生成節点数 538, 753) に対するものである．また，ISCAS ベンチマーク[14]に対する結果を**表-1** に示す．節点数の多い例題に対しては良い結果が得られている．なお，1台での実行時間は単一プロセッサ用のプログラムで計測した．並列プログラムを1台で実行すると，共有記憶へのロックのために 1.3 倍強の時間を必要とする．

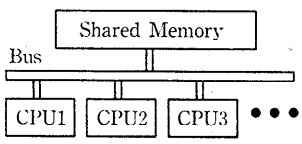


図-7 A shared-memory multi-processor system.

表-1 Evaluation on S-81 for ISCAS benchmarks.

Data	# nodes	台数-所要時間（秒）			高速率 (1：25)
		1台	10台	25台	
c 432	27, 531	5.76	2.00	1.20	4.80
c 499	65, 561	7.85	1.72	1.06	7.40
c 880	527, 689	48.2	19.50	16.0	3.01
c 1355	189, 835	18.1	3.35	2.41	7.50
c 1908	264, 912	51.5	8.72	4.42	11.60
c 3540	962, 465	159	34.20	22.60	7.04
c 5315	155, 094	14.2	4.18	2.67	5.33

使用している並列計算機の CPU は最新の RISC プロセッサに比較すると低速であるが，最新のプロセッサを用いた並列計算機においても同じだけの高速化率を得られることが期待できる．

3.2 非共有記憶型並列計算機での処理方式

非共有記憶型計算機では，BDD の (共有) 節点をどのようにして記憶するかが問題となる．木村ら[15]は，SBDD の節点を各プロセッサで独立に管理し，最終的な結果の BDD の節点をどこかのプロセッサに構築する，という方法を提案している．負荷の分散方式については Shannon の展開を直接用い，展開された演算列を各プロセッサで受け持つ方式を用いる．図-2 に対応する演算列を4つに分割する場合には，入力変数 a, b について $a=b=0$; $a=0, b=1$; $a=1, b=0$; $a=b=1$ とした4つの演算列を構成する．

展開された結果の各演算列は，各プロセッサで独立に処理され，演算の処理にともなう通信は存在しない．ただし，最終的な結果の BDD を構成するためには，各プロセッサで作成した BDD の節点をある一つのプロセッサへ送らねばならず，通信のオーバヘッドを生じる．この方式では，各プロセッサで論理的に等価な節点を重複してもつ可能性があるものの，各プロセッサで扱う節点数は少なくなり，かつ通信のオーバヘッドが少ないという利点がある．なお，論理照合（二つの論理関数の等価性の判定）などは，分割したままで判定ができるので，通信のオーバヘッドはほぼ0に等しい．

AP 1000 を用いた評価を以下に示す．AP 1000 はセルと呼ばれるプロセッサ (15 MIPS SPARC, 16 MB 主記憶) が2種類の通信ネットワークでホストのプロセッサと接続されている．ホストとすべてのセルは，コマンドバスと呼ばれる共通バス

で接続されており，セル間はセルネットワークと呼ぶ2次元トーラスのネットワークで接続されている（図-8）．

10 ビットの乗算器に対する評価結果を示す．図-9 は出力をまとめあげず，単に計算のみを行った結果である．二つの論理関数の等価性の判定などがこれに対応する．一方，表-2 は 20 個すべての出力の BDD をホストに構成した場合の結果である．通信の部分がオーバヘッドになり，8倍程度で高速化が飽和している．

3.3 データ駆動計算機での処理方式

甲村ら[16]は，電総研のデータ駆動計算機 EM-4 を用いた BDD の並列処理手法を提案している．EM-4 は，12.5 MHz のクロックで動作する RISC プロセッサを，プロセッサ数 n に対して $\log n$ に比例した段数で通信ができるサーキュラオメガネットワークで結合した．並列計算機である．プロセッサ間通信には，パケット通信方式を用いている．各プロセッサは，1クロックで1命令を実行でき，2クロックで1パケットの出力ができる．EM-4 では，パケットによりほかのプロセッサに関数を起動でき，ほかのプロセッサで関数を起動するためのオーバヘッドは非常に小さい．

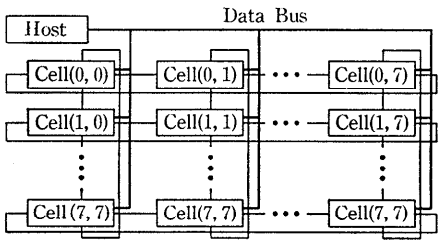
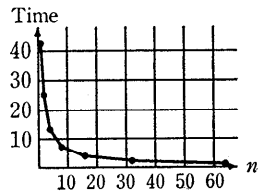
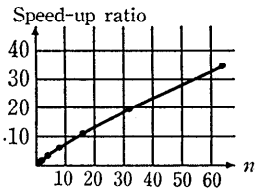


図-8 AP 1000 architecture.



(a) Execution time.



(b) Speed-up ratio.

図-9 Evaluation on AP 1000 for 10-bit multiplier.

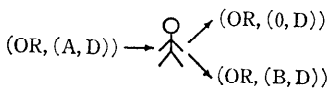
表-2 Execution on AP 1000 with output merge.

台 数	1	8	32	64
秒	48.9	9.96	5.53	5.89
率	1.00	4.91	8.85	8.31

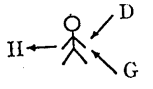
そこで，まず節点の集合を各プロセッサに分けてもたせ，ある節点に対する処理はその節点をもつプロセッサで行う方式をとっている．具体的には節点の情報をキーにしてプロセッサの番号を決めて情報を格納し，パケットの転送により処理を行う．

二項論理演算の節点対の処理においては，対の大きいほうの節点をもつプロセッサで処理を行うこととしている．並列化では幅優先の展開を用いて，並列実行できる節点対を求めている．たとえば，図-5 の例では，図-10 に示すように (OR, (A, D)) を受けとったプロセッサは，0 枝の (0, D) と1枝の (B, D) を求め，それらから計算したプロセッサに対して，(OR, (0, D)) と (OR, (B, D)) を送る．これを繰り返すことにより，展開ができる．また，最小化では，展開の逆にパケットを返してゆく．この方式では，同一の節点対は同じプロセッサに割り当てられるので，展開時の同一節点対の検査は，同一プロセッサ内での検査のみでよい．

文献 16) によると，80 プロセッサの EM-4 のプロトタイプを用いた評価では，直列版と並列版の比較で 20 倍強の高速化を達成している．また，ワークステーション SUN 4/330 (25 MHz SPARC) との比較でも，10 倍強の高速化になっている．評価で用いられた論理関数は，50-bit の加算器で，節点数が多くなるような入力順序を用いたものである．



(a) Expansion.



(b) Reduction.

図-10 Data driven method.

3.4 PRAM 上での理論的な並列アルゴリズム

二つの論理関数を表す BDD の間の論理演算は，PRAM (Parallel Random Access Machine) 上で，BDD の節点数の多項式に比例した個数のプロセッサを用いることにより，節点数の対数の多項式に比例した計算時間で計算できることが分かっている[10)~12)]．PRAM は共有記憶にランダムアクセスすることが可能なプロセッサが複数あるという並列計算のモデルである．実際の共有記憶をもつ並列計算機との違いは，共有記憶へのアクセスに通信時間を考えない点と同時アクセスの（ためのロックの）手間を考えない点である．

論理演算に対する並列アルゴリズムの考え方を示す[10),11)]．論理演算は，先に述べたように二つのフェーズで行う．まず展開のフェーズでは，演算を行う対象の二つの関数を表す BDD の節点のすべての対に対し，0 で展開した先の対，および 1 で展開した先の対に対応する節点を求める．対の情報からそれに対応する節点を求めるときに，対全体の中の探索が必要となるが，対の数 s に対し $\log s$ に比例した時間でできる．これをすべての対の数だけのプロセッサを用いて行えばよい．

最小化のフェーズでは，まず演算結果の BDD（節点の数を s とする）のすべての節点の対 (v, v') を考え，v と v' が表す論理関数の EXOR を表す BDD を構成する（先ほどの展開と同じ）．この EXOR を表す BDD の節点の数は s^2 であり，s^2 個のプロセッサを用いて $\log s^2 = 2\log s$ に比例した時間でできる．EXOR を表す BDD で，(v, v') から 1 へ到達できないとき（すなわち恒等的に 0 のとき）は，v と v' とは論理的に等価である．

到達可能性は，節点間の隣接行列 A を節点の個数回かけた結果の行列を用いて判定できる．なお，行列の n 乗を求めるときには，A, A^2, $A^4 = A^2 \times A^2, \cdots$ のようにして $O(\log n)$ 時間で求める手法を用いる．行列の乗算は $A[i, j]$ を保持するプロセッサを行数だけ用意して，$A[i, 1] \times A[1, k]$, $A[i, 2] \times A[2, k], \cdots$ を同時に同じ番地へ（同時に）書き込みにいかせれば，定数時間で実行できる．A のサイズは，EXOR の BDD の節点数 s^2 に対して $s^2 \times s^2$ で，A の各要素を s^2 個用意する必要があるので，プロセッサの数は $(s^2)^3$ となる．

つぎに，等価な節点の中から節点番号が最大のものを代表元として選び，最小の BDD を構成する．ある節点と等価な節点番号が最大の節点を探すには，節点数を s^2 として s^2 個のプロセッサで $O(\log s^2) = O(\log s)$ 時間で実行できる．各節点についてこれを行うので，全体で $(s^2)^2$ 個のプロセッサを用いる．あとは，枝のつなぎ換えを行えばよい．最悪のところでも $O(\log s)$ 時間，$O(s^6)$ プロセッサで実行できる．ここで，$O()$ は括弧内の関数に比例した量で上から押さえられることを表す．

なお，BDD の処理を行うアルゴリズムを，究極の並列計算機構である組合せ論理回路で実現する場合には，回路の段数が $O(\log^2 s)$ で実現できるという結果も得られている[12)]．組合せ論理回路は，PRAM とは異なる並列計算のモデルで，並列計算量の正確な議論をするときに用いられる．

4. ベクトル計算機（SIMD 型並列計算機）を用いた処理方式

越智ら[17)]は，ベクトル計算機を用いた BDD の処理方式を提案している．ベクトル計算機は，演算パイプライン方式の並列計算機（ベクトルユニット）を備えた計算機であり，ベクトルユニットはパイプライン演算装置およびこれに高速にデータを供給するベクトルレジスタからなる．パイプライン演算装置は，演算器の内部の処理が多段化され，あるデータの演算の 1 段分の処理が終了した時点で次のデータの演算を開始することができる（**図-11**）．図-11 では，データは左から右に流れ，前処理を左で行い，後の処理を右側で行う．ベクトル計算機は特に配列に格納されたデータに対して同じ演算を繰り返して行うような場合には，短時間に多量のデータを処理することができる．

ベクトル計算機上の処理では，幅優先の手法を用いている．最初は，根の節点対一つであるが，幅優先で生成していくに従い，同時に（ベクトル

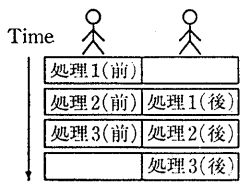


図-11 Pipeline method.

的に）処理できる節点対は順次増加してゆく．節点対集合をリストベクトルで表すことにより，ベクトル計算機で効率良く処理できる．また，複数の論理演算を処理する場合には，同時に実行可能な初期節点対が複数個存在することになるので，それらを一つのベクトルにまとめることにより，高速化を行っている．

最小化のフェーズは，ベクトル計算機で効率良く処理するために，2段階に分けて行われる．まず第一段階では，展開のフェーズで生成された節点と論理的に等価な節点がないか，あるいは冗長な節点でないかを検査し，印をつける．第二段階で，印のつけられた節点を削除し，最小の BDD を構成する．第一および第二の段階は，おのおの個別にベクトル処理される．

展開のフェーズの演算キャッシュへの同時アクセスでは，「検査，書き込み，再検査」の手法を用いて高速化を達成している．再検査の結果，自分が生成したものと別の節点が登録されているならば，自分の生成した節点を廃棄し，登録されている節点を用いる．また，最小化のフェーズでの BDD の節点を保持している節テーブル（チェインつきハッシュ表）へのアクセスについては，ポインタで結合された節の情報をあたかも二次元配列であるようにみて，最初の列から順次アクセスすることにより，ベクトル計算機での効率化を行っている．

HITAC S-820/80 を用いた ISCAS ベンチマークデータに対する評価結果を**表-3** に示す[17]．表中のS実行は HITAC S-820/80 のスカラユニットのみを用いた場合の実行時間で，V実行がベクトルユニットを用いた場合の実行時間を表す．表から分かるように 24 倍程度の高速化を達成している．

表-3 Evaluation on HITAC S-820/80.

回 路	節点数	CPU 時間（秒）		率 (S/V)
		S実行	V実行	
c 432	104,066	8.637	0.551	15.7
c 499	65,671	2.473	0.256	9.7
c 880	31,378	1.969	0.349	5.6
c 1355	208,324	5.637	0.626	9.0
c 1908	60,850	2.903	0.528	5.5
c 3540	1,029,210	68.111	2.833	24.0
c 5315	48,353	6.361	1.424	4.5

表-4 Evaluation for ISCAS benchmarks.

Data	SUN ELC		S-81 (25台)		S-820/80 (V)	
	#node	秒	#node	秒	#node	秒
c 432	27,531	1.98	27,531	1.20	104,066	0.551
c 499	65,561	2.51	65,561	1.06	65,671	0.256
c 880	27,919	0.85	527,689	16.0	31,378	0.349
c 1355	189,835	6.61	189,835	2.41	208,324	0.626
c 1908	113,732	4.20	264,912	4.42	60,850	0.528
c 3450	—	—	962,465	22.60	1,029,210	2.833
c 5315	155,094	4.80	155,094	2.67	48,353	1.424

5. おわりに

以上，BDD の並列処理技法について述べた．並列化に適した処理の方式としては，論理演算の処理を幅優先で処理する方式と，深さ優先で処理する方式とが提案されている．実際の並列計算機（ベクトル計算機）を用いた評価では，節点数の大きな問題に対して威力を発揮していることが分かる．今後，並列計算機，あるいは分散計算環境がますます普及すると考えられるので，ここで示したような並列化の手法を用いる必要は高まっていくものと考えられる．

ところで，現在のワークステーションとこれら並列計算機の BDD 処理の性能はどうなっているのであろうか．アルゴリズムおよび入力変数の順序づけが異なるので単純には比較できないが，論文で示されている結果をまとめると**表-4** のようになる．生成された節点数（#node）がほぼ等しい場合には比較しても意味があるのではないだろうか．ワークステーションとしては，SUN SPARCstation ELC（33 MHz SPARC, 16 MB 主記憶）を用いた．

最後になりましたが，本文中で紹介した並列アルゴリズムの解説の執筆においては，引用している論文の著者の京都大学武永康彦助手，京都大学越智裕之さん，三洋電機甲村康人さんから本文にコメントをいただき，また，図表などの使用も許可いただきました．ここに記入して感謝します．

参考文献

1) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Comput.*, Vol. C-35, No. 8, pp. 677-691 (Aug. 1986).
2) Brace, K. S., Rudell, R. L. and Bryant, R. E.: Efficient Implementation of a BDD Package, In *Proc. of 27th DA Conf.*, pp. 40-45 (June 1990).

3) 湊，石浦，矢島：論理関数の共有二分決定グラフによる表現とその効率的処理手法，情報処理学会論文誌，Vol. 32, No. 1, pp. 77-85 (Jan. 1991).
4) Bryant, R.E.: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication, *IEEE Trans. on Comput.*, Vol. C-40, No. 2, pp. 205-213 (Feb. 1991).
5) 崔，小原，石浦，白川：共有二分決定グラフを用いた順序回路のテスト生成法，信学技報，VLD 92-28, pp. 61-66 (May 1992).
6) Hiraishi, H., Hamaguchi, K., Ochi, H. and Yajima, S.: Vectorized Symbolic Model Checking of Computational Tree Logic for Sequential Machine Verification, In *Proc. of 3rd Workshop on Computer Aided Verification*, pp. 1-12 (1991).
7) 安浦：並列計算機と並列計算モデル，情報処理，Vol. 33, No. 9, pp. 1024-1032 (Sep. 1992).
8) 岩間：PRAM 上の並列アルゴリズム，情報処理，Vol. 33, No. 9, pp. 1033-1041 (Sep. 1992).
9) Kimura, S., Igaki, T. and Haneda, H.: Parallel Binary Decision Diagram Manipulation, *IEICE Transactions on Fundamentals*, Vol. E 75-A, No. 10, pp. 1255-1262 (Oct. 1992).
10) 武永，保坂，矢島：二分決定グラフに基づく論理演算の並列アルゴリズム，1992 年電子情報通信学会秋期大会，SD-1-6, pp. 6-369-370 (1992).
11) Sieling, D. and Wegener, I.: *NC*-Algorithms for operations on Binary Decision Diagrams, *to appear in Parallel Processing Letters.*
12) 武永，矢島：二分決定グラフによる論理演算の計算複雑さ，信学技報，COMP 91-86, pp. 21-26 (Jan. 1992).
13) Kimura, S. and Clarke, E.M.: A Parallel Algorithm for Constructing Binary Decision Diagrams, In *Proc. of ICCD '90*, pp. 220-223 (Sep. 1990).
14) Brglez, F. and Fujiwara, H.: A Neutral Netlist of 10 Combinational Circuits, In *Proc. 1985 IEEE International Symposium on Circuit and Systems* (1985).
15) 木村，松本，羽根田：二分決定グラフの並列処理アルゴリズムについて，情報処理学会 DA シンポジウム 92, pp. 105-109 (Aug. 1992).
16) 甲村，児玉，山口：データ駆動計算機 EM-4 における共有二分決定グラフの並列処理について，情報処理学会第 44 回全国大会 3D-5, pp. 6-(43-44) (Mar. 1992).
17) Ochi, H., Ishiura, N. and Yajima, S.: A Vector Algorithm for Manipulating Boolean Functions Based on Shared Binary Decision Diagrams, *SUPERCOMPUTER*, Vol. 8, No. 9, pp. 101-118 (Nov. 1991).

(平成 4 年 12 月 28 日受付)

木村 晋二

1959 年生．1982 年京都大学工学部情報工学科卒業．1984 年同大学院修士課程修了．1985 年より神戸大学工学部助手．1993 年より奈良先端科学技術大学院大学情報科学研究科助教授．形式的論理設計検証，並列 CAD アルゴリズムの研究に従事．IEEE，電子情報通信学会，LA シンポジウム各会員．