部分グラフ同型判定アルゴリズムの FPGA による実装と評価

市川周一[†] ラターナセンタン ウドーン[†,☆] 小西幸治[†,☆☆]

多くの応用は部分グラフ同型判定問題としてモデル化できるが，部分グラフ同型判定は一般に NP 完全で実用時間内に解くことが難しい．この問題を専用回路で高速に解くため，ハードウェア化に適したアルゴリズムを提案し，FPGA に実装して評価した．試作回路は PCI バス経由でホスト PC に接続され，ホスト上のソフトウェアから利用される．1 チップの Lucent ORCA 2C15A FPGA 上に 2 つの同型判定ユニットが実装され，それぞれ 16.5 MHz で並列に動作する．Pentium-II 400 MHz の PC 上でソフトウェアを用いて実行した場合と比べ，最大 20 倍程度の性能を発揮する．より大規模な FPGA を用い，複数のチップやボードを同時利用することで，さらなる性能向上も容易である．

An FPGA-based Implementation of Subgraph Isomorphism Algorithm

Shuichi Ichikawa,[†] Lerdtanaseangtham Udorn[†,☆] and Kouji Konishi[†,☆☆]

Many applications can be modeled as subgraph isomorphism problems, which is generally NP-complete and difficult to compute practically. To accelerate the solver, an algorithm that is suited for hardware implementation is proposed and evaluated on FPGA. The prototype accelerator operates at 16.5 MHz on Lucent ORCA 2C15A, which outperforms the software implementation of Ullmann's algorithm on a 400 MHz Pentium-II by 20 times in the best case. The performance can be boosted easily by using multiple FPGA chips or multiple boards.

1. はじめに

画像認識分野におけるシーン解析や，化学情報分野における構造活性の推測など，多くの応用が部分グラフ同型判定問題（Subgraph Isomorphism）に帰着できる[1]．しかし一般に部分グラフ同型判定は NP 完全である[2]ため大きな実行時間を要し，実用的に用いることは困難である．

ソフトウェアで多大な実行時間を要する応用に関して専用計算回路で実行を高速化する研究は古くから多く行われてきたが，ハードウェアの開発コスト等の問題から広く使われるには至らなかった．ところが近年，LSI 技術の進歩により FPGA 等の再構成可能ハードウェアが大規模化・低コスト化し，専用計算回路の実用性が急速に高まってきている[3]．

本研究では，部分グラフ同型判定問題の実行を専用回路で高速化する手法を示す．実際に PCI バス用再構成可能論理回路を用いて実装した結果，小規模で動作速度の遅いプロトタイプでも，汎用マイクロプロセッサ上のソフトウェアに比べ最大 20 倍程度の性能を発揮することが実証された．

† 豊橋技術科学大学・知識情報工学系
Department of Knowledge-based Information Engineering, Toyohashi University of Technology
☆ 現在，株式会社トヨタケーラム
Presently with Toyota Caelum Incorporated
☆☆ 現在，NTT ソフトウェア株式会社
Presently with NTT Software Corporation

2. 関連研究

（部分）グラフ同型判定問題を専用回路によって高速に解こうとする研究には，Ullmann による提案[1]と Swain らによる提案[4]があるが，いずれも提案だけで実装や性能評価が行われていない．Ullmann の refinement procedure [1]は問題の性質を利用した巧妙な方法であるが，回路規模が大きく実用性に難がある[5]．それに対して本研究ではハードウェア化に向いた回路を提案し，実際に回路を構成して，汎用マイクロプロセッサ上のソフトウェアを性能面で凌駕することを実証している．

グラフの同型判定は，広い意味で制約充足問題（CSP）の一種と考えることができる．制約充足問題を高速に解決するための専用回路はいくつか提案されてきたが，FPGA が普及するまで実装されたものは少なかったようである[4),6)~8)]．Swain らの提案[4]は，

アークコンシステンシを用いた制約充足問題をハードウェアで高速に解決しようというもので，その1つの応用としてグラフのマッチングについても言及している．このハードウェアは適用範囲の広いものであるが，グラフ同型判定に特化していないため効率の点では必ずしも最適でない．また実装も行われていない．Guらによる DRA チップ[8]は Discrete Relaxation Algorithm をシストリック計算するもので，3.0μ NMOS での実装があるが，グラフ理論への応用は検討されていない．

近年では，応用に特化した専用計算回路を FPGA で実装する研究が多く行われている．本研究と類似の計算困難問題への適用研究としては，充足可能性問題（SAT）への応用研究が多く存在する[9)~20)]．Boolean SAT の問題設定自体が論理式の形をしているため，FPGA の SAT への応用はきわめて素直な発想であり，多くの研究がなされたと考えられる．一方（部分）グラフ同型判定問題に関しては FPGA の適用例は見当たらず，我々が初めてと思われる．

3. アルゴリズム

3.1 部分グラフ同型判定問題

2つのグラフ G_α，G_β を考える．部分グラフ同型判定問題とは，G_α が G_β のいずれかの部分グラフ（subgraph）と同型（isomorphic) であるか否か判定するという問題である．たとえば**図1**において，G_β は G_α と同型な部分グラフを持つ．一方 G_γ は持たない．この判定問題は一般に NP 完全であることが証明されている[2)]．

3.2 列挙アルゴリズム

部分グラフ同型判定問題を素朴な方法で解くには，列挙型のアルゴリズムを用いればよい．G_α の頂点数を p_α，辺数を q_α，G_β の頂点数を p_β，辺数を q_β としよう．$p_\alpha > p_\beta$ であれば部分グラフ同型でないことは自明なので，以下では $p_\alpha \leq p_\beta$ であることを前提とする．G_β から p_α 個の頂点を取り出して G_α の頂点に対応させる方法は ${}_{p_\beta}P_{p_\alpha}$ 通り存在する．こうして取り出した G_β の部分グラフを G'_β とする．G_α のすべての辺に対して G'_β に対応する辺が存在していれば，G_α は G_β の部分グラフと同型（部分グラフ同型判定の結果は成功）である．${}_{p_\beta}P_{p_\alpha}$ 個の対応すべてについて調べても同型な部分グラフがなければ，部分グラフ同型判定の結果は失敗である．

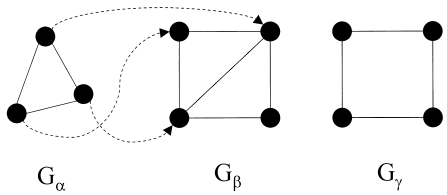

図1 部分グラフ同型
Fig. 1 Subgraph isomorphism.

これは一種の探索問題なので，探索木を深さ優先探索する形で実装できる．探索木の第 i レベルでは，G_α の頂点 v_{α_i} $(1 \leq i \leq p_\alpha)$ から G_β の頂点 v_{β_j} $(1 \leq j \leq p_\beta)$ への写像 $p_\beta - i + 1$ 個を列挙する．探索木の葉（レベル p_α）では G_α の頂点すべての写像が定まるので，選ばれた G'_β に対応する辺が存在するかどうか確認する．q_α 本のすべてについて対応する辺が存在すれば同型な部分グラフを発見したことになる．すべての葉を調べても同型な部分グラフがなければ判定は失敗となる．このアルゴリズムは，上位に探索木巡回部を置き，探索木の葉にきたとき辺存在確認部を呼び出すという形で実装できる．

3.3 Ullmann のアルゴリズム

素朴な列挙アルゴリズムは，p_α，p_β の増大にともなって実行時間が長大となり実用性が低い．そこで，望みのない探索枝の探索を省略する方法（枝刈り；pruning）が重要になる．Ullmann は探索枝の節点のそれぞれで refinement procedure という同型可能性判定を行い，同型にならない部分探索木を刈り込むことを提案した[1)]．この方法は，現在でも部分グラフ同型判定に最もよく使われる方法の1つである．以下にその概略だけを示す．

Refinement procedure は一種の再帰的な必要条件判定である．探索枝の各節点で1つの写像の結果が部分グラフ同型を導くには，G_α において隣接する頂点どうしが，写像先（G_β）においても隣接するような写像でなければならない．Ullmann は，この条件を以下のように定式化した．

G_α，G_β の隣接行列を，それぞれ $A = [a_{ij}]$ $(1 \leq i, j \leq p_\alpha)$，$B = [b_{ij}]$ $(1 \leq i, j \leq p_\beta)$ としよう．Ullmann のアルゴリズムでは行列 M を計算しながら探索を行う．行列 $M = [m_{ij}]$ $(1 \leq i \leq p_\alpha, 1 \leq j \leq p_\beta)$ は，$v_{\alpha_i} \in V_\alpha$ から $v_{\beta_j} \in V_\beta$ への写像で部分グラフ同型になる可能性がある場合に $m_{ij} = 1$，なければ $m_{ij} = 0$ という意味を持つ．たとえば m_{ij} の初期値を，$deg(v_{\alpha_i}) \leq deg(v_{\beta_j})$ のとき $m_{ij} = 1$，それ以外は $m_{ij} = 0$ と定める（$deg(v)$ は頂点 v の次数である）．

Refinement procedure では，以下の式を再帰的に適用して M が変化しなくなるまで m_{ij} を計算する．r_{xj} $(1 \leq x \leq p_\alpha, 1 \leq j \leq p_\beta)$ は中間変数である．

$$r_{xj} = (\exists y)(m_{xy} \cdot b_{yj}) \quad (1)$$
$$m_{ij} = m_{ij} \cdot (\forall x)(\bar{a}_{ix} \vee r_{xj}) \quad (2)$$

Refinement procedure を言葉で説明すれば以下のようになる．必要条件を満たさない写像は部分グラフ同型を導かないので，写像の可能性を捨てる．しかしある写像の可能性を捨てることにより，今まで満たされていた必要条件が将棋倒し式に否定される可能性がある．したがって将棋倒しが終わるまで必要条件検査を再帰的に行う．

Refinement procedure の結果 M のいずれかの行がすべて 0 になったとすれば，それはある頂点の写像可能性がすべて否定されたことを意味し，同時にこれ以下の部分探索枝に部分グラフ同型が発見される可能性がないことを意味するので，以下の探索枝を切り捨てて次の枝へ進む．一方まだ部分グラフ同型を導く可能性が残されている場合は，次の頂点の写像を行う（下のレベルに進む）．末端の節点（葉）で refinement procedure の条件を満たせば，その写像は部分グラフ同型になっている．Ullmann のアルゴリズムは，上位の探索木巡回アルゴリズムから探索木の各節点で refinement procedure を呼び出すという形で実装できる．Ullmann の方法では内部節点のオーバヘッドが増加するが，枝刈りによる探索空間削減の効果により全体の探索時間は劇的に減少する．

Ullmann は，refinement procedure が組合せ回路で実装できることを示した[1]．しかし必要なゲート数が $O(p_\alpha {p_\beta}^2)$ と大きく☆，大きなグラフを扱うハードウェアを実装するには無理がある[5]．

3.4 提案アルゴリズム

Refinement procedure は，まだ割り当てていない頂点も含めて再帰的に必要条件の検査を行う．そのため探索木の浅いレベルから枝刈りを行うことができて枝刈りの性能は良いが，ハードウェアによる実装で多量の資源を要求する．そこで本論文では簡略化した同型可能性判定を採用することにより，必要な資源量を削減してハードウェアに実装することにした．

本研究では，必要条件判定で写像済みの頂点だけを扱うことによって必要条件の判定を緩める．すなわち refinement procedure と同様に必要条件の検査を行うが，レベル i で隣接条件を検査する際に写像済みの頂点 $(v_{\alpha_1}, \ldots, v_{\alpha_i})$ の間の辺に関してだけ写像先の接続関係を検査する．これは G_α の部分グラフが G_β の部分グラフになりうるか検査していることに相当する．

☆ 必要なメモリは隣接行列 A，B と作業記憶（M など）だけで，メモリ量は $O(p_\beta{}^2)$ にとどまる．

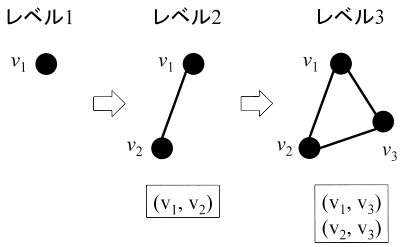

図 2 辺リストの生成
Fig. 2 Edge list generation.

G_α が G_β の部分グラフと同型となるためには，G_α の部分グラフはすべて G_β のいずれかの部分グラフと同型でなければならないからである（必要条件）[21]．

もう少し具体的に説明するため，図 1 の G_α を例として考える．以下，$v_i \in V_\alpha$ の写像を $f(v_i) \in V_\beta$ と表すことにする．**図 2** は，探索木の各レベルで写像済みの節点からなる G_α の部分グラフを示したものである．探索木のレベル 1 では，頂点 v_1 が $f(v_1)$ に写像される．この時点では $f(v_1)$ が G_β のどの頂点であっても矛盾はない．次にレベル 2 で v_2 を $f(v_2)$ に写像するとする．G_α において v_1 と v_2 は隣接しているので，$f(v_1)$ と $f(v_2)$ は G_β において隣接していなければならない．$f(v_1)$ と $f(v_2)$ が隣接していなければ，これ以下の探索木に部分グラフ同型な写像はない（必要条件を満たさない）ので，以下の探索木を切り捨ててよい．

各レベルの部分グラフが含む辺のリストを，図 2 の枠線内に示した．レベル k の辺リストを L_k と表すなら，本節で述べた必要条件は「$\forall (v_i, v_j) \in L_k$ に対して，$(f(v_i), f(v_j)) \in E_\beta$ であること」と表現することができる．実際には，すでに上のレベルで確認済みの辺を再度確認する必要はないので，辺リストから省いて実行時間を節約することができる．たとえば図 2 のレベル 3 では，辺リストに (v_1, v_2) が含まれていない．(v_1, v_2) はレベル 2 の辺リストに含まれており，すでに確認済みだからである．レベルごとの辺リストは探索木の全域で不変なので，探索前に生成して用意しておくことができる．

提案アルゴリズムは，Ullmann のアルゴリズムと同様に探索木の内部節点で条件を判定して枝刈りを行うため，列挙アルゴリズムと比べて実行時間が大幅に改善される．しかし Ullmann のアルゴリズムよりは簡略な枝刈りであるため，データによって Ullmann のアルゴリズムより長い実行時間を要する可能性がある．その一方，必要なハードウェア資源量が Ullmann のアルゴリズムより少なく，（後で述べるように）現状の

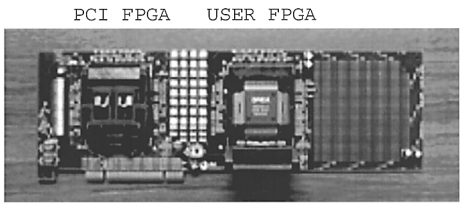

図 3 OPERL ボード
Fig. 3 OPERL board.

FPGA でも実装可能であるため，専用回路による高速化のメリットを享受することができる．ソフトウェアによる Ullmann のアルゴリズムと，ハードウェアで高速化した提案アルゴリズムの性能比較については 6 章で述べる．

4. 設　　計

4.1 システム構成

実装には OPERL ボード[22]を利用する．OPERL は Lucent 社の OR2C シリーズ FPGA を 2 個搭載した，PCI バス用再構成可能論理ボードである（**図 3**）．USER FPGA にはホストからソフトウェアで任意の bit stream をダウンロードし，ユーザの設計した回路を動的に再構成することができる．本論文では USER FPGA として OR2C15A（19200〜44200 ゲート相当[23]）を使用する．PCI FPGA（OR2C15A）には，PCI インタフェース回路と USER FPGA の再構成制御回路が搭載されている．

ホスト計算機には一般的な仕様のパーソナルコンピュータを用いた．ホスト計算機の仕様を**表 1** に示す．ホスト側から OPERL ボードを利用する方法としては，(1) システムコール (read/write) でデバイスドライバを呼び出す方法と，(2) 入出力命令で直接入出力ポートをアクセスする方法が可能であるが，本研究ではユーザプログラムから入出力命令を発行する方法を採用した☆．今回の応用ではボードとホスト間のデータ転送量が非常に少ないため，システムコールを利用するよりも直接入出力命令を発行する方がオーバヘッドが少ないからである[22]．ポートアドレスは ioctl で OS から取得するようにしているため，PCI の Plug&Play にも対応しており，ソフトウェアの移植性も損なっていない．

4.2 回 路 構 成

図 4 にブロック図を示す．探索木巡回回路では，G_α から G_β への可能な組合せを順次生成する．探索木の各節点で 3.4 節で述べた必要条件判定を行うため，辺導出回路は G_α の（検査対象となる）辺の両端点を G_β に写像する．辺存在確認回路では，辺導出部から送られた G_β の 2 頂点間に辺があるか否かを返す．現実には辺存在確認回路は G_β の隣接行列を格納する RAM にすぎない．辺導出部では辺リストを用いて各レベルの検査対象（辺）すべてについて辺存在確認回路に照会を行い，すべての辺が必要条件を満たすか否かを探索木巡回回路に返答する．条件を満たせば探索木巡回回路は下のレベルの写像に進む．条件を満たさなければ，このレベルで可能な次の写像を試みる．このレベルの写像がすべて終われば上のレベルに上がる．

表 1 ホスト計算機
Table 1 Host computer.

CPU	AMD K6-III 400 MHz
Memory	64 MB
Chip Set	Intel 430HX
OS	FreeBSD 2.2.1R
Compiler	gcc-2.7.2

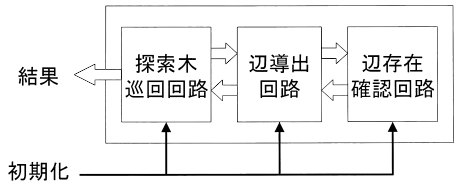

図 4 ブロック図
Fig. 4 Block diagram.

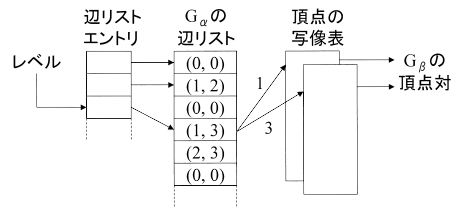

図 5 枝刈りの方法
Fig. 5 Pruning procedure.

図 5 は，図 2 に示した例に関して辺導出回路の概念を示したものである．探索木巡回回路から現在の探索レベルを受け取り，レベルに対応する辺リストを参照する．辺は頂点番号の組でできており，(1, 3) は (v_1, v_3) の辺を意味する．制御回路は，対応する辺リストのエントリからリスト終端までのすべての辺について順番に必要条件を検査する．(0, 0) は制御回路への終端指示である．頂点の写像表は探索木巡回回路が管理しており，探索木の現節点において G_α の各頂点

☆ OS には手を加えていない．FreeBSD や Linux では元々このような利用法が提供されている．

が G_β のどの頂点に対応するか，番号順に格納している．辺の両端点の写像を同時に参照するため，この写像表は 2 ポート RAM になっている．読み出された G_β の頂点対を用いて辺存在確認回路（G_β の隣接行列）を参照すると，G_β における対応辺の有無が判明する．

4.3 回路規模

本節では提案アルゴリズムの実装に必要な回路規模について簡単に述べる．最初に明記しておくべきことは，アルゴリズムを実現する回路にはいろいろな選択肢があるということである．今回の設計では，部分グラフ同型判定専用回路の適用可能性を調べるため，FPGA に実装して評価することを第 1 の目的とした．したがって高性能を求めるより回路資源の節約を最優先して設計を行った．同じアルゴリズムでも，速度を優先しようとすれば組合せ回路や並列化を積極的に利用することにより回路資源のオーダは大きくなる．性能と実装コストの最適なトレードオフに関しては，今後の課題と考えている．

今回の設計では同型可能性判定を順序回路化する（辺リストを順序回路でアクセスする）などしてゲート数を減らしたため，必要なゲート数は $O(p_\alpha \log p_\beta)$ にすぎない．Ullmann の提案が $O(p_\alpha {p_\beta}^2)$ であるのに比べ，著しくゲート数が少ないことが分かる．

メモリに関しては，問題定義自体が G_α，G_β の隣接行列を要求することから，最低限 $O({p_\beta}^2)$ は必要となる．今回の設計では G_α を辺リストで表現したため，辺リストに $O({p_\alpha}^2 \log p_\alpha)$ の資源量が必要になるが，これは本質的ではない．FPGA を用いた実装では組合せ論理より SRAM の集積度が高いことを考慮して，実装密度の高くなる実現法を選んだ結果である．その意味でメモリの本質的な必要量は $O({p_\beta}^2)$ といってよい．

5. 実　　装

5.1 実装方法

USER FPGA の論理はすべて VHDL で記述し，Synopsys 社の Design Compiler で論理合成を行った．ただし実装を考慮した VHDL 記述を行うため，Lucent 社の OR2CxxA 用ライブラリ（たとえば LUT を使う RAM マクロ）を直接指定している箇所が多い．

OR2C は SRAM ベースの FPGA なので，マッピング RAM を多用する設計がアーキテクチャによく適合する．提案アルゴリズムを含む部分グラフ同型判定問題は非常に表引きの多い実装になるので，SRAM ベース FPGA に適した応用であるといえる．

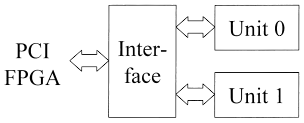

図 6 USER FPGA の構成
Fig. 6 Configuration of USER FPGA.

論理合成には Sparc Ultra1/170 を用いて 10 分程度を要する．論理合成の出力（EDIF ネットリスト）は，Lucent 社 ORCA Foundry 9.35 でマッピングし，配置配線する．マッピングと配置配線には，Pentium-II 450 MHz の PC で 2 時間ほどを要する．

5.2 実装結果

今回 USER FPGA として利用した OR2C15A は，論理の構成単位として PFU（Programmable Function Unit）を 400 個内蔵している．1 PFU は，5 入力 1 出力の組合せ回路 2 組，または 16×2 bit の SRAM2 組に相当する[23]．OR2C FPGA で $(p_\alpha, p_\beta) = (15, 15)$ まで扱える回路を実装すると 160 PFU 程度で実現できるので，1 つの OR2C15A に図 4 の回路を 2 ユニット搭載できることになる．そこで USER FPGA を**図 6** のような構成にすることとした．

2 ユニット搭載するかわりに，より大きな (p_α, p_β) を扱う回路を実装することも考えられる．今回 2 ユニット搭載を選択したのは，部分グラフ同型判定の応用では一般に多くのパターンに対して判定を行うスループット指向であることを考慮したためである．また，実装に用いた OR2C FPGA のマッピング RAM が 16×4 という構成であるため，p_α，p_β が 16 を超えると実装効率の低下が見込まれることも考慮した．しかし，たとえば OR3C シリーズ FPGA を使えばマッピング RAM も 32×4 であるし，カスタム LSI などで実装すれば自由な RAM 構成がとれるので，応用からの要求に応じて，より大きな (p_α, p_β) を扱う回路を実装することも容易であると思われる．

図 6 のインタフェース回路は，PCI FPGA から送られてくるアドレスをデコードし，read 要求に対してはユニットの状態を送出し，write 要求に対しては該当ユニットの制御や初期化を行う．各ユニットはまったく独立しており，ホスト側のソフトウェアからは 2 つの同型判定を並列に実行できる．インタフェース部は PCI バスと同じクロック（33 MHz）で動作する．各ユニットは単純なマルチサイクル実装になっており，バスクロックの倍周期（16.5 MHz）で動作する．技術的にはパイプライン化して 33 MHz で動作させることも可能と思われるが，今回は人月の不足により断念した．

表 2 論理規模
Table 2 Logic scale.

回路	PFU 数
インタフェース	23
Unit 0	160
Unit 1	160
合計	343

表 3 ユニットの内訳
Table 3 Sub-unit.

回路	PFU 数	(RAM)
探索木巡回	79	(8)
辺導出＋辺存在確認	56	(24)
その他	24	
合計	159	

表 2 に実装回路の PFU 数を示す．合計 PFU 数は 343 で，OR2C15A の 85%に相当する．OPERL に搭載可能な最大の FPGA は OR2C40A（900 PFU, 43200〜99400 ゲート相当[23]）であり，OR2C40A を用いると提案アルゴリズムは 4 ユニット実装可能である．参考のため，Ullmann の提案どおりに refinement procedure を組合せ回路で実装してみた結果，同じ $(15, 15)$ の回路で 2754 PFU となり 2C40A でも到底実装不能であることが分かった．

表 3 には，各ユニット内の論理規模を PFU 数で示した．ユニットの合計 PFU 数（159）は，表 2 の値（160）とわずかに異なっているが，これはテクノロジマッピングによる揺れである．マッピングではまったく無関係の回路でも PFU 内に空き資源があれば相乗りさせて詰め込もうとする．また手頃な回路がなければ PFU の一部が空いていても放置する．このため回路を切り分けてマッピングを行うと詰め込みの都合で多少 PFU 数が変わることがある．表 3 の（RAM）の項は，サブユニットの PFU 数のうち RAM マクロとして利用されている PFU 数である．マッピング RAM を積極的に利用して集積度を上げている様子が見てとれる．

専用回路の処理結果が正しいことを確認するために，“G_α と同型な部分グラフが G_β の中に何個含まれているか” を計数する回路が各ユニットに含まれている．これは動作には関係ない検証用回路であるが，表 3 の “その他” の PFU 数に含まれている．

6. 性能評価

6.1 測定方法

本節では部分グラフ同型判定の性能を測定する．ただし部分グラフ同型判定の実行時間は入力データ（G_α と G_β）に依存して大きく変動するため，ランダムに生成した 100 パターンのグラフに関して実行時間を測定し，平均値で優劣を評価することにする．

グラフの頂点数を p，辺数を q とおいたとき，辺密度 ed は以下の式で定義される．

$$ed = 2\,q\,/\,p\,(p-1) \tag{3}$$

ed は，グラフの辺数 q と，同じ頂点数の完全グラフ K_p の辺数の比であるから，グラフの辺の多さを表すパラメータと解釈してよい．その値は $0 \le ed \le 1$ となる．ただし，グラフが連結であるためには $q \ge p-1$ でなければならないので，連結グラフに関しては $2/p \le ed \le 1$ となる．一般に部分グラフ同型判定問題で扱うグラフが連結である必要はないが，実用的には非連結なグラフの同型判定にはあまり意味がないので，本研究では G_α，G_β ともに連結なグラフに限定して性能測定を行った．

本論文では，入力グラフの属性として頂点数と辺密度を取り上げ，G_α と G_β について p_α，p_β と ed_α，ed_β をそれぞれ変えて傾向を観察する．

表 4 参照システム
Table 4 Reference system.

CPU	Intel Pentium-II 400 MHz
Memory	256 MB
Chip Set	Intel 440BX
OS	FreeBSD 3.1R
Compiler	gcc-2.8.1

6.2 ソフトウェアの性能

性能の評価基準としては，汎用マイクロプロセッサ上で Ullmann のアルゴリズムをソフトウェアで処理した場合の実行時間を用いる．以下，これを参照システムと呼ぶ．参照システムの構成は**表 4** のとおりである．この参照システム上で Ullmann のアルゴリズム（ソフトウェア）を実行し，平均処理時間を測定した．**図 7** に $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，**図 8** に $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．以下の性能測定では，この参照システムでの実行時間を基準とした性能比だけを示すこととする．

次に提案アルゴリズムと Ullmann のアルゴリズムの効率を比較するため，提案アルゴリズムをソフトウェアで実装して参照システム上で実行し，Ullmann のアルゴリズムとの性能比を調べた．**図 9** は $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，**図 10** は $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の性能比である．

図 9 では，多くの点で提案アルゴリズムが Ullmann のアルゴリズムを上回る性能を示している．これは以下のような理由によると考えられる．$ed_\alpha < ed_\beta$ と

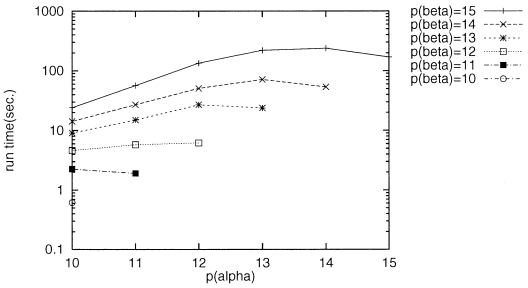

図 7 ソフトウェアの実行時間．$(ed_\alpha, ed_\beta) = (0.2, 0.4)$
Fig. 7 Execution time of software. $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

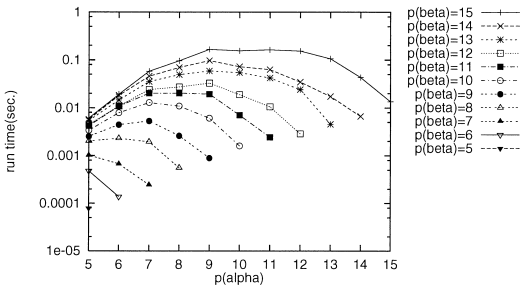

図 8 ソフトウェアの実行時間．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 8 Execution time of software. $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

いう条件では疎なグラフを密なグラフの中で探すため，部分グラフ同型になる写像が多く発見される可能性が高い．つまり本質的に枝刈りが有効に機能しないので，時間のかかる refinement procedure を実行しても得るところは少ない．それに対して提案アルゴリズムの枝刈り手続きは短時間で済むので，結果として Ullmann のアルゴリズムより高い性能を発揮する．

図 10 では多くの点で Ullmann のアルゴリズムが勝っている．$ed_\alpha \geq ed_\beta$ という条件下では，部分グラフ同型になる写像の数が少ないため refinement procedure による枝刈りが有効に働くためである．

このようにアルゴリズムの性能は入力データ（グラフ）の性質によって大きな影響を受ける．入力の性質はユーザの意図や応用のような外部的要因によって決まるので，そのような前提を抜きにして一概に性能の優劣を語ることはできない．目的とデータの性質に合ったアルゴリズムを選んで用いる必要がある．

6.3 専用回路の性能

参照システムと同じグラフ（100 セット）を専用回路で処理し，実行時間を測定した．今回設計した回路には，独立した要素回路が 2 ユニット搭載されているので，まず要素回路（単一ユニット）の性能から示す．**図 11** は，$(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合の測定結果を参照システムとの性能比（実行時間の逆数の比）で

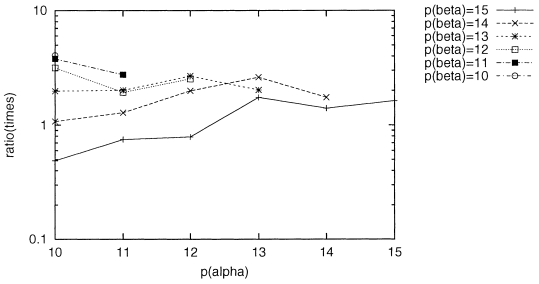

図 9 提案アルゴリズムの性能．$(ed_\alpha, ed_\beta) = (0.2, 0.4)$
Fig. 9 Performance of proposed algorithm. $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

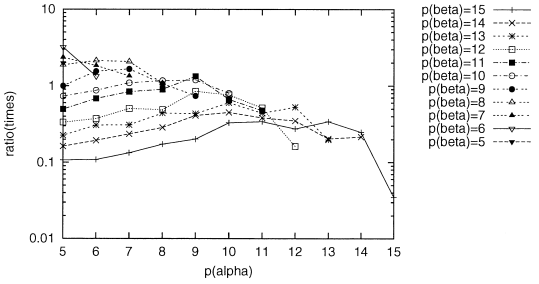

図 10 提案アルゴリズムの性能．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 10 Performance of proposed algorithm. $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

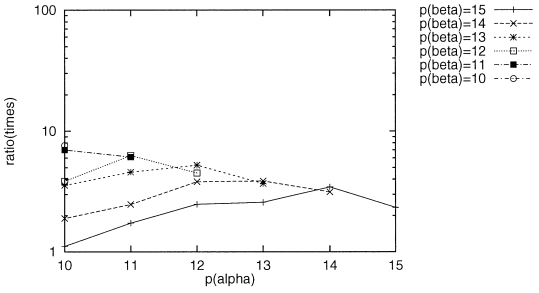

図 11 要素回路の性能．$(ed_\alpha, ed_\beta) = (0.2, 0.4)$
Fig. 11 Performance of unit circuit. $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

示したものである．要素回路は 16.5 MHz と動作周波数が低いにもかかわらず，図上多くの点で Pentium-II 400 MHz の参照システムを凌駕する性能を見せている．ソフトウェアが逐次的な処理を行うのに対して，専用回路では 1 クロック内に複数の表（RAM）を読み出すなどの形で並列性を利用しているためである．

次に 2 ユニット両方を利用した測定結果を**図 12** に示す．ホスト側のソフトウェアでは，OPERL 上の Unit0 と Unit1 にグラフを 1 組ずつ割り当てたあと，IN 命令でユニットの状態を監視しながらループして待ち（ポーリング），いずれかのユニットが処理を終了したら新たなグラフを割り当てるという素朴な処理を行っ

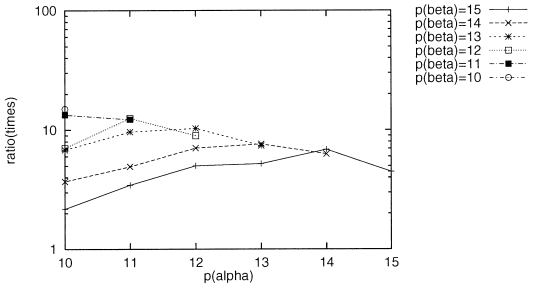

図 12 専用回路の性能．$(ed_\alpha, ed_\beta) = (0.2, 0.4)$
Fig. 12 Performance of accelerator.
$(ed_\alpha, ed_\beta) = (0.2, 0.4)$

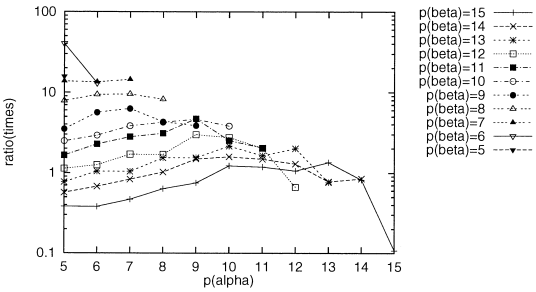

図 13 専用回路の性能．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 13 Performance of accelerator.
$(ed_\alpha, ed_\beta) = (0.4, 0.4)$

ている．図 12 を図 11 と比較すると，性能がほぼ 2 倍になっているのが分かる．ユニット数が多くなるにつれ台数効果は頭打ちになると思われるが，ユニット数が 2 程度では飽和は見られない．そこで以下の評価では 2 ユニット両方を使った結果だけを示す．

図 13 に $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．6.2 節でも述べたとおり，この入力パラメータでは枝刈りの効率が問題となる．そのためハードウェアによる加速を含めても，性能比が 1 を下回る場合がある．

6.4 ハードウェアとソフトウェアの選択的利用

前節で示したように専用回路は多くの点で参照システムを上回る性能を発揮するが，参照システムより性能が低い場合があることも事実である．たとえば図 13 では，$p_\beta = 15$ の場合に広い範囲で 1 倍を下回っている．このような場合は，データの性質に合わない専用回路を用いるかわりに，ホスト上のプロセッサで Ullmann のアルゴリズムを用いて処理すればよい．しかし問題となるのは，実際に部分グラフ同型判定を行う前に専用回路とソフトウェアの実行速度を予測するということである．個別の G_α，G_β に関する実行時間は予測不能（データ依存）だが，p_α，p_β，ed_α，ed_β の組について事前に統計をとることはできるので，専用回路の性能が低下しそうなパラメータ・セットに関してはソフトウェアによる処理に切り替えるという方法を試みた．

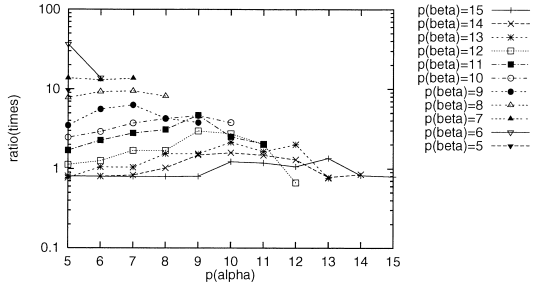

図 14 選択法の性能．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 14 Performance of selection approach.
$(ed_\alpha, ed_\beta) = (0.4, 0.4)$

まず 6.3 節の測定結果を用いて，いろいろなパラメータ対に関して専用回路の平均性能を求めておく．同じ入力データセットに関して，参照システム上での実行時間を基にホスト上での Ullmann のソフトウェアの性能を推測する．実測データによれば，参照システムでの実行時間に 1.31 を乗じるとホスト上での実行時間とほぼ一致するので，これを推測値として用いる．

実際の処理時には，入力されたグラフ G_α，G_β の p_α，p_β，ed_α，ed_β を計算し（これは簡単に求まる），そのパラメータ対で専用回路が勝ると予測されれば専用回路に投入し，ホスト上のソフトウェアが勝ると予測されればホスト上で Ullmann のアルゴリズムを実行することにする．以下，この方法を選択法とよぶ．

事前のデータ収集に用いたグラフ（100 セット）と別のグラフ（100 セット）を用意して，実機上で選択法の性能を測定した．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$ に関する結果を**図 14** に示す．図 13 と比べると，予測がよく的中して性能低下が抑えられていることが分かる．グラフが性能比 1 でなく 0.76 近辺で平らになっているのは，ホストの性能が参照システムの 76%程度しかないためである（グラフは参照システムの性能を 1 として正規化してある）．例外的に $(p_\alpha, p_\beta) = (12, 12)$ の性能が 0.76 より低いのは，予測が不正確だったため遅い方（専用回路）を使ってしまったからである．

6.5 ハードウェアとソフトウェアの協調

さらに性能低下を避け，または性能向上を得る方法を考える．ホスト側は通常，ポーリングをして専用回路の終了を待っている．これは無駄なことなので，専用回路の終了を待つ間に，ホスト側でも Ullmann のアルゴリズムで部分グラフ同型判定を行うことを考え

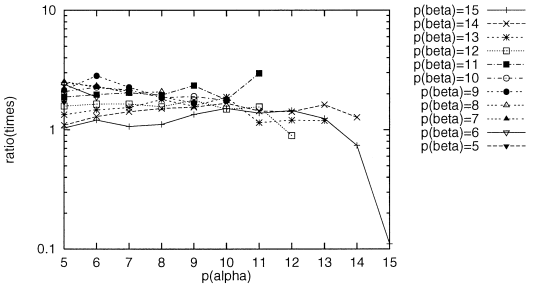

図 15 分担法の性能．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 15 Performance of sharing approach. $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

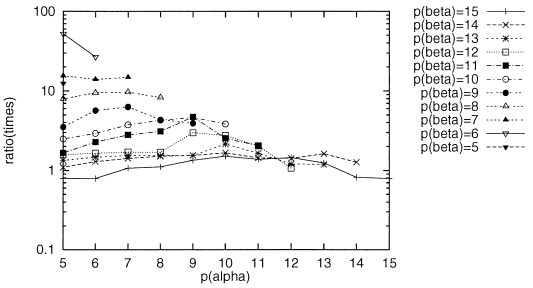

図 16 協調法の性能．$(ed_\alpha, ed_\beta) = (0.4, 0.4)$
Fig. 16 Performance of cooperative approach. $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

る．ホスト側は 1 セットのグラフの判定を終えたときに専用回路（Unit0，Unit1）の状態をチェックし，作業が終了していれば新たなデータを専用回路に割り当てる．割当てが済んだら（または専用回路が作業中であったら），ホストは次のグラフ対について部分グラフ同型判定をはじめる．このような方法（分担法）で性能を測定した結果が**図 15** である．ほとんど自明な結論であるが，極端な性能低下も避けられるかわり，遅い方に足を引っ張られてピーク性能も低下することが分かる．

そこで次に，6.4 節の選択法と分担法を併用することを考える．専用回路がホスト側ソフトウェアより K 倍以上速いと予測される場合は専用回路を使い，ソフトウェアが K 倍以上速いと予測される場合はホスト側で処理する．この中間と予測される場合は分担法を採用する．この方法を協調法と呼ぶことにする．協調法では，K の値を 1 とすれば中間領域がなくなって選択法と同じになる．K の値を大きくしていくと中間領域が広がって，分担法が多く使われるようになる．$K = 2$ の協調法で測定した結果が**図 16** である．性能低下を避けながら，性能比が 1 に近い部分の性能が向上していることが分かる．

7. 考　　察

部分グラフ同型判定問題に対して専用回路を適用することは，どの程度有望であろうか．また，本研究で試作した専用回路はどの程度役に立つだろうか．今回の実装には以下の 2 つの課題が残されているといえる．

- 処理速度が十分でない
- 扱えるグラフが小さい

今回は FPGA を用いて実装評価するため，速度よりも回路規模の削減を優先した．また扱える頂点数も 15 程度と小さい専用回路を作成した．しかし集積度の高い FPGA を利用したり，セミカスタム LSI 化したりするなどの手段により，より大規模なグラフを扱う回路を高い周波数で実現することは可能であると考えられる．特に本研究で提案したアルゴリズムでは，4.3 節でも述べたように，必要なハードウェア資源量のオーダが Ullmann のアルゴリズムよりも小さい．したがって扱うグラフの頂点数を増やすことは容易である．

頂点数を増やした場合に問題になるのは実行時間の指数的増大である．しかし 6.2 節でも見たように，本質的に枝刈りが難しいグラフに関しては提案アルゴリズムは Ullmann のアルゴリズムより高い性能を発揮することが分かっている．このようなグラフを多く扱うならば，提案アルゴリズムを実装することは非常にコスト性能比の良い解を与えると考えられる．

逆に枝刈りが効率的に行えるようなグラフに対しては，提案アルゴリズムは無駄な探索を行って実行時間が大きくなるので適切でない．このようなグラフを多く扱うならば，枝刈りを効率良く行うアルゴリズムを選択し，そのアルゴリズムを少ないゲート数で実装する工夫をすべきである．

入力データ（グラフ）の性質が事前に把握できない場合は，事前にデータの性質に適合した専用回路を用意することができない．しかしその場合でも，本研究の選択法・協調法で示したように，入力データの頂点数や辺密度から実行時に最適な実行法を選ぶことが可能である．

もう 1 つ重要なのは，応用に特化した回路を設計することである．部分グラフ同型判定問題の実応用として化学構造式データベースの検索を考えてみよう．化学構造式は単なるグラフではなく，各頂点は原子やベンゼン環などの区別を持っている．同様に辺も，単結合，二重結合などの区別がある．このようなデータ構造から部分構造の一致を検索する場合，頂点や辺に区別があるため写像可能性が大きく制限され，探索空

間が小さくなる．つまり一般的な部分グラフ同型判定問題よりも解きやすく，同じ時間でも大きな構造を扱うことができる．また，データベースの検索では多くのパターンに対して同型判定を行うので，専用回路の並列化によるスループットの向上がきわめて有望である．入力データの性質も相当部分予測可能であるため，データに合わせた専用回路の最適化も期待できる．

今回試作した回路は「部分グラフ同型判定」という一般的な問題を扱うものであったが，さらに具体的な応用に特化することにより実用規模の計算困難な問題を高速に解く専用回路が実現可能であると考える．このような適用範囲の狭い専用回路は非現実的であると思われてきたが，近年の論理合成と半導体技術（FPGA等）の進歩により，今日では十分に実現可能な解であると思われる．

8. おわりに

本研究では，部分グラフ同型判定問題についてハードウェア化に適したアルゴリズムを提案し，実際にOR2C15A FPGA 上に実装して性能評価を行い，汎用マイクロプロセッサ上で Ullmann のアルゴリズムを実行する場合に比べて最大 20 倍程度の性能を発揮することを実証した．さらに集積度の高い FPGA（OR2C40A）の採用や，複数の FPGA の利用，または複数の OPERL ボードを同時に利用するなどの方法で，より高いピーク性能を得ることは容易である．

提案アルゴリズムでは，ハードウェア資源を削減するため Ullmann のアルゴリズムより枝刈りを簡略化している．このため入力データによっては Ullmann のアルゴリズムをホスト側ソフトウェアで実行する方が実行時間が短くなることがある．本論文では，そのような場合でもホスト上のソフトウェアと専用回路の間で適切な協調戦略をとることにより，専用回路化による欠点を回避していっそうの性能向上を得ることが可能なことを示した．

今回の実装は，小規模な FPGA 1 チップだけを利用した限定的なものである．より大規模な FPGA やセミカスタム LSI 技術を用いることによって，さらに大きな p_α，p_β に関して大きな性能を発揮する専用回路を構成できると考えられる．

謝辞 本研究の一部は，(財) 堀情報科学振興財団 研究助成，文部省科学研究費補助金・特定領域研究（B）（2）10205210 および奨励研究（A）11780211 によるものである．

参考文献

1) Ullmann, J.R.: An Algorithm for Subgraph Isomorphism, *J. ACM*, Vol.23, No.1, pp.31–42 (1976).
2) Garey, M.R. and Johnson, D.S.: *Computers and Intractability*, Freeman (1979).
3) Buell, D.A., Arnold, J.M. and Kleinfelder, W.J.: *Splash 2: FPGAs in a Custom Computing Machine*, IEEE Computer Society (1996).
4) Swain, M.J. and Cooper, P.R.: Parallel Hardware for Constraint Satisfaction, *7th National Conference on Artificial Intelligence* (*AAAI '88*), pp.2:682–686, Morgan Kaufmann (1988).
5) 齋藤秀充：Ullmann のアルゴリズムのハードウェアによる実装に関する研究，修士論文，豊橋技術科学大学知識情報工学系 (2000).
6) Cherry, C. and Vaswani, P.K.T.: A New Type of Computer for Problems in Propositional Logic, with Greatly Reduced Scanning Procedures, *Information and Control*, Vol.4, pp.155–168 (1961).
7) Ullmann, J.R., Haralick, R.M. and Shapiro, L.G.: Computer Architecture for Solving Consistent Labelling Problems, *Computer Journal*, Vol.28, No.2, pp.105–111 (1985).
8) Gu, J., Wang, W. and Henderson, T.C.: A Parallel Architecture for Discrete Relaxation Algorithm, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.PAMI-9, No.6, pp.816–831 (1987).
9) Suyama, T., Yokoo, M. and Sawada, H.: Solving Satisfiability Problems on FPGAs, *Proc. 6th Int'l Workshop on Field-Programmable Logic and Applications* (*FPL'96*), pp.136–145, Springer-Verlag (1996).
10) Hamadi, Y. and Merceron, D.: Reconfigurable Architectures: A New Vision for Optimization Problems, *3rd Int'l Conf. Principles and Practice of Constraint Programming* (*CP '97*), pp.209–221, Springer-Verlag (1997).
11) Zhong, P., Martonosi, M., Ashar, P. and Malik, S.: Accelerating Boolean Satisfiability with Configurable Hardware, *Proc. IEEE Symp. FPGAs for Custom Computing Machines* (*FCCM '98*), pp.186–195, IEEE Computer Society (1998).
12) Zhong, P., Martonosi, M., Ashar, P. and Malik, S.: Solving Boolean Satisfiability with Dynamic Hardware Configurations, *Proc. 8th Int'l Workshop on Field-Programmable Logic and Applications* (*FPL'98*), pp.326–335, Springer-Verlag (1998).
13) Rashid, A., Leonard, J. and Mangione-Smith,

W.H.: Dynamic Circuit Generation for Solving Specific Problem Instances of Boolean Satisfiability, *Proc. IEEE Symp. FPGAs for Custom Computing Machines* (*FCCM '98*), pp.196–204, IEEE Computer Society (1998).
14) Suyama, T., Yokoo, M. and Sawada, H.: Solving Satisfiability Problems Using Logic Synthesis and Reconfigurable Hardware, *Proc. 31st Hawaii Int'l Conf. System Sciences* (*HICSS'98*), IEEE Computer Society (1998).
15) Platzner, M. and Micheli, G.D.: Acceleration of Satisfiability Algorithms by Reconfigurable Hardware, *Proc. 8th Int'l Workshop on Field-Programmable Logic and Applications* (*FPL'98*), pp.69–78, Springer-Verlag (1998).
16) Habbas, Z., Herrmann, F. and Singer, D.: A Methodological Approach to Implement CSP on FPGA, *10th IEEE Int'l Workshop on Rapid System Prototyping*, pp.66–71, IEEE Computer Society (1998).
17) Zhong, P., Martonosi, M., Ashar, P. and Malik, S.: Using Configurable Computing to Accelerate Boolean Satisfiability, *IEEE Trans. Computer-aided Design of Integrated Circuits and Systems*, Vol.18, No.6, pp.861–868 (1999).
18) Mencer, O. and Platzner, M.: Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment, *Proc. 32nd Hawaii Int'l Conf. System Sciences* (*HICSS'99*), IEEE Computer Society (1999).
19) Abramovici, M. and de Sousa, J.T.: A Virtual Logic Algorithm for Solving Satifiability Problems Using Reconfigurable Hardware, *Proc. IEEE Symp. FPGAs for Custom Computing Machines* (*FCCM '99*), pp.306–307, IEEE Computer Society (1999).
20) Chan, P.K., et al.: Reducing Compilation Time of Zhong's FPGA-based SAT Solver, *Proc. IEEE Symp. FPGAs for Custom Computing Machines* (*FCCM '99*), pp.308–309, IEEE Computer Society (1999).
21) 小西幸治：部分グラフ同型判定アルゴリズムの FPGA を用いた実装手法，修士論文，豊橋技術科学大学知識情報工学系 (1999).
22) 市川周一，島田俊夫：パーソナル・コンピューティグ指向の動的再構成可能 PCI カード，第 5 回 FPGA/PLD Design Conference & Exhibit, 東京，pp.269–277, 中外 (1997).
23) Lucent Technologies Inc.: *ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series FPGAs Data Sheet* (1996).

(平成 12 年 1 月 28 日受付)
(平成 12 年 6 月 2 日採録)

市川 周一（正会員）

昭和 38 年生．昭和 60 年東京大学理学部情報科学科卒業．昭和 62 年東京大学大学院理学系研究科情報科学専門課程修士課程修了．新技術開発事業団,（株）三菱電機，名古屋大学工学部助手を経て，現在豊橋技術科学大学知識情報工学系講師．理学博士．IEEE, ACM，電子情報通信学会各会員．

ラターナセンタン ウドーン
（学生会員）

昭和 46 年生．平成 10 年豊橋技術科学大学知識情報工学系卒業．平成 12 年同大学院知識情報工学専攻修士課程修了．同年より（株）トヨタケーラムに勤務．

小西 幸治（正会員）

昭和 49 年生．平成 9 年豊橋技術科学大学知識情報工学系卒業．平成 11 年同大学院知識情報工学専攻修士課程修了．現在 NTT ソフトウェア（株）ネットワークプラットフォーム事業部勤務．IEEE 会員．