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 400MHz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，更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 problem, which is generally NP-complete and hard 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 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
Dept. Knowledge-based Information Engineering, Toyohashi University of Technology
☆ 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, Cooper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]．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，その一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．

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幾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．近年では，充足可能性問題(SAT)にFPGAを応用する研究が多く行われているが，(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．

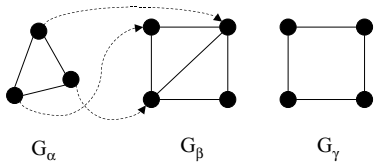

図 1　部分グラフ同型

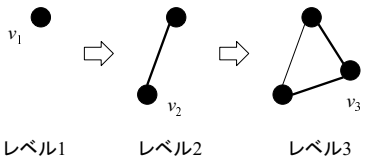

図 2　辺リストの生成

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．例えば**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良い．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の全ての辺に対して 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．${}_{p_\beta}P_{p_\alpha}$個の対応全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．

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 $v_{\alpha_i}(1 \leq i \leq p_\alpha)$ から G_βの頂点 $v_{\beta_j}(1 \leq j \leq p_\beta)$ 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の頂点全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 本の全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．全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，探索木の葉に来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の増大に伴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は省略する方法 (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良く使われる方法の一つである．

Refinement procedure は一種の再帰的な必要条件判定である．探索枝の各節点で一つの写像の結果が部分グラフ同型を導くには色々な必要条件がある．例えば，$deg(v_{\alpha_i}) \leq deg(v_{\beta_j})$ でなければならない ($deg(v)$ は頂点 vの次数)．また，G_αにおいて隣接する頂点同士が，写像された先の 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．必要条件を満たさない写像の可能性は捨てる．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 (例えば隣接の必要条件など) が将棋倒し式に否定される可能性があるので，将棋倒しが終るまで必要条件検査を再帰的に行う．Refinement procedure の結果，可能性のある写像先がなくなった頂点が一つでもあれば，以下の部分探索枝を切り捨てて次の枝へ進む．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，必要なハードウェア資源量は $O({p_\alpha}{p_\beta}^2)$ 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必要条件の検査を行うため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．即ち G_αの部分グラフが G_βの部分グラフになり得るかを検査する．G_α が G_βの部分グラフと同型となるためには，G_α の部分グラフは全て 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)[6)]．

もう少し具体的に説明するため，図 1 の G_αを例にとる．探索木のレベル 1 では，G_αの頂点 v_1が G_βのいずれかの頂点に写像される．以下レベル 2 では v_2，レベル 3 で v_3が写像されて，探索木の葉に達する ($p_\alpha = 3$)．このとき提案アルゴリズムでは，各レベルで写像済み節点からなる G_αの部分グラフを考え

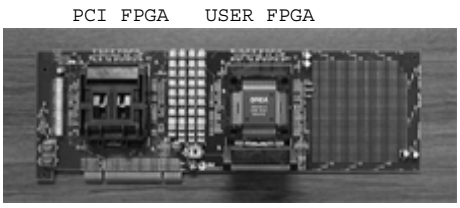

図 3　OPERL ボード

(**図 2**)，その部分グラフが G_β に含まれるかどうか検査する．より具体的には，G_α の部分グラフに含まれる辺の写像が G_β に含まれることを確認する．ただし既に上のレベルで確認済みの辺は，再度確認する必要がないので省く．例えば図 2 のレベル 3 で，細い辺は既にレベル 2 で確認済みであるから再度検査する必要はない．レベル毎の部分グラフ (正確には辺のリスト) 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，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よりは実行時間が大幅に短い．しかし Ullmann のアルゴリズムと比べて枝刈りの効率が悪いため，同じソフトウェアとして実行すると Ullmann のアルゴリズムに劣る．ただし必要なハードウェア資源量が少ないため，後で述べるように現状の FPGA に実装可能となり，専用回路で高速化することが可能になる．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[7] 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) 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 (**表 1**)．FreeBSD や Linux ではユーザプログラムから入出力命令を発行することができるので，データ転送にはプログラムから直接 OPERL のポートを読み書きすることにした．行儀良くデバイスドライバ経由で read/write することも可能だが，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，システムコールのオーバヘッドを避けることを選んだ[7]．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．

表 1　ホスト計算機

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

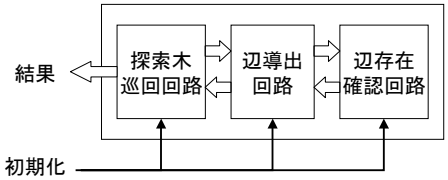

図 4　ブロック図

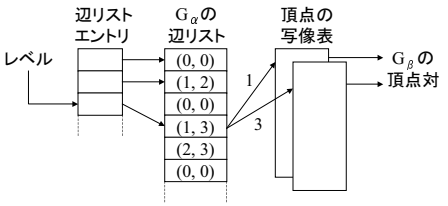

図 5　枝刈りの方法

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 に過ぎない．辺導出部では辺リストを用いて各レベルの検査対象 (辺) 全てについて辺存在確認回路に照会を行い，全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全て終れば上のレベルに上がる．

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．(0, 0) は制御回路への終端指示である．対応する辺リストのエントリからリスト終端までの全ての辺について必要条件を検査する．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 の各頂点が G_β のどの頂点に対

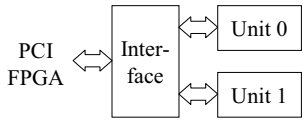

図 6　USER FPGA の構成

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．

5. 実　　装

5.1 実装方法

USER FPGA の論理は全て 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 (例えば 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良く適合する．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．

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 450MHz の PC で 2 時間程を要する．

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 を 400 個内蔵している．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．

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 要求に対してはユニットの制御や初期化を行う．各ユニットは全く独立しており，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 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

表 2　論理規模

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

搭載可能な最大の FPGA は OR2C40A (900 PFU, 43200〜99400 ゲート相当) 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 の提案通りに 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わかった．

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

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．

$$ed = 2\,q \,/\, p\,(p-1) \qquad (1)$$

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あらわ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余り意味がないので，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．

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 3** の通りである．この参照システム上で，Ullmann のアルゴリズムを実装したソフトウェアを実行し，平均処理時間を測定した．例として，**図 7** には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，**図 8** には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ 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．

表 3　参照システム

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

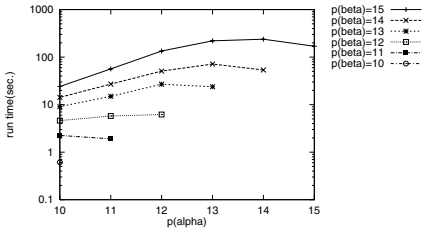

図 7　ソフトウェアの実行時間 $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

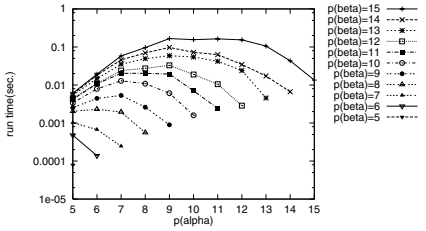

図 8　ソフトウェアの実行時間 $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

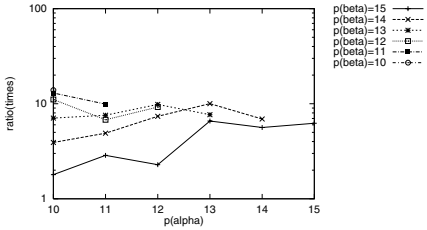

図 9　専用回路の性能 $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

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．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ている．

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) として示したものが**図 9** と**図 10** である．専用回路は 16.5 MHz と動作周波数が低いにも関わらず，図上多くの点で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．

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の性能は多くの点で参照システムを上回るが，一部で参照システムより低い

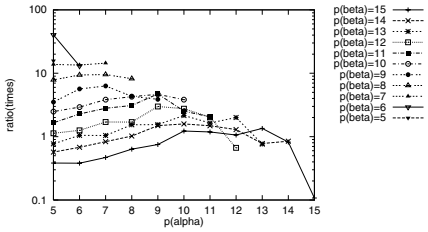

図 10　専用回路の性能 $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

場合もある．例えば図 10 では，$p_\beta = 15$ の場合に広い範囲で 1 倍を下回っている．このような領域では専用回路化による高速化要因よりも，本質的に提案アルゴリズムの枝刈り能力が Ullmann のアルゴリズムに劣るという性能低下要因が支配的になっていることがわかった．

このような場合，アルゴリズムに問題のある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良い．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，$p_\alpha, p_\beta, ed_\alpha, ed_\beta$の組について事前に統計を取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替えるという方法を試みた．

まず 6.3節の測定結果を用いて，色々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元にホスト上での 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 $p_\alpha, p_\beta, ed_\alpha, ed_\beta$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 11** に示す．図 10 と比べると，予測がよく的中して性能低下が押さえられていることがわかる．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) を使ってしまったためである．

6.5　ハードウェアとソフトウェアの協調

ホスト側は通常，ポーリングをして専用回路の終了

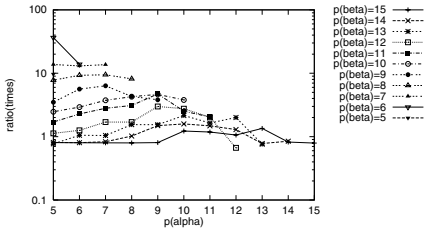

図 11　選択法の性能 $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

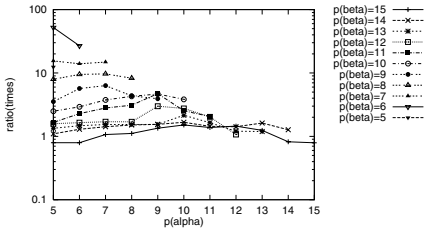

図 12　協調法の性能 $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

を待っている．これは無駄なので，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分担することを考える．ホスト側は 1 セット判定を終える度に専用回路 (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てから，次のグラフ対について部分グラフ同型判定をはじめる．しかしこのような方法 (分担法) では，専用回路とホスト側ソフトウェアの間に大幅な性能差があった場合，遅い方が足を引っ張ってしまう．実際の測定結果でも (紙数の都合で省略)，ピーク性能の低下が観察された．これは望ましくない副作用である．

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．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 の値を大きくしてゆ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 12** 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わかる．

7. 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枝刈りを Ullmann のアルゴリズムより簡略化している．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一層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, *Seventh National Conference on Artificial Intelligence (AAAI '88)*, Morgan Kaufmann, pp. 2:682–686 (1988).
5) 齋藤秀充, ラターナセンタンウドーン, 市川周一: 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, 2000 年電子情報通信学会総合大会 (2000).
6) 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).
7) 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, 中外, pp. 269–277 (1997).