研究課題別評価

1 研究課題名:

ハードウェア・プログラミングによる超細粒度並列処理

2 研究者氏名:

井口 寧

3 研究のねらい:

近年, VLSI の集積度は急速に向上しており, 近い将来数十億ゲートのコンピュータチップが出現すると予想される. CPUチップ上の回路に占める演算器の回路量は, 現在では既に全体の数パーセントと非常に小さくなっていて, 今後は CPU チップの回路量が増加しても, それに比例する処理性能の向上は困難だと考えられる. 一方で, 多数のマイクロプロセッサを結合した超並列計算機は, 自然科学分野におけるシミュレーションなどの多くの分野で, 大規模な計算を高速に実行することができる計算機として盛んに研究され, 並列処理に関する技術が蓄積されてきた.

そこで本研究では, これまでの超並列処理技術を VLSI チップの回路設計に適用することによって, VLSI チップ内の大規模な論理回路上に高並列な専用演算回路をプログラムの都度ごとに合成し, 将来大容量の論理回路が与えられた時にも高い処理性能を実現するための手法について研究した.

研究の進め方として, 近年利用可能なCレベル設計ツールに対して, 超並列処理技術の助けを借りながら並列化解析を行い, 専用並列化回路を合成するアプローチをとった. C レベル設計ツールは C 言語に近い記述でハードウェア回路を設計可能なツールであり, 近年盛んに開発・利用されるようになってきている. しかし, これらの処理系はコストを重視しコンパクトな回路を合成するための最適化がなされているため, そのままでは高い並列性を持った回路を合成することは困難である. そこで, 元のプログラムに含まれる並列性を解析・抽出し, C レベル設計ツールが並列演算回路を合成できるように, ディレクティブ(並列化指示)を挿入したりループを展開することによって, ソフトウェア・アルゴリズムを高い並列性を持つ演算回路として展開するアプローチを採り, 研究を行った. プログラムは, ディレクティブやループ展開などによって並列性を陽に含む記述に変換された後, C レベル設計ツールや配置配線ツールによって並列演算器を含むハードウェア回路として展開できるので, VLSI チップ上に並列演算回路を容易に構築できる. 超並列計算機での並列性解析の知見を大規模 VLSI 上での回路設計に導入することによって, 将来の大規模 VLSI チップのための超細粒度並列処理の実現を試みた.

4 研究成果:

4.1 超細粒度並列処理のためのテストベット構築

まず最初に, 本研究で実験に用いるテストベットを構築した. 図1は, 構築したシステムによる超細粒度並列処理の仕組みである. 実験用VLSIとして, FPGA 素子を利用した. FPGA は, ユーザーが使用時に内部回路をプログラムできる素子であり, ユーザーは任意の回路を繰り返し LSI チップ内に構成することができる. 本研究では, C 言語で記述されたソフトウェアを入力とし, そのソフトウェアを並列化演算回路として展開し, FPGA 上に実装する. プログラムごとに要求される内部回路が異なるので,

プログラムの実行の都度回路合成を行い，FPGA の内部回路を指定するコンフィギュレーションファイルを得てFPGAにダウンロードし回路をスタートさせる．本テストベッドを用いて，提案スキームの評価などを行った．

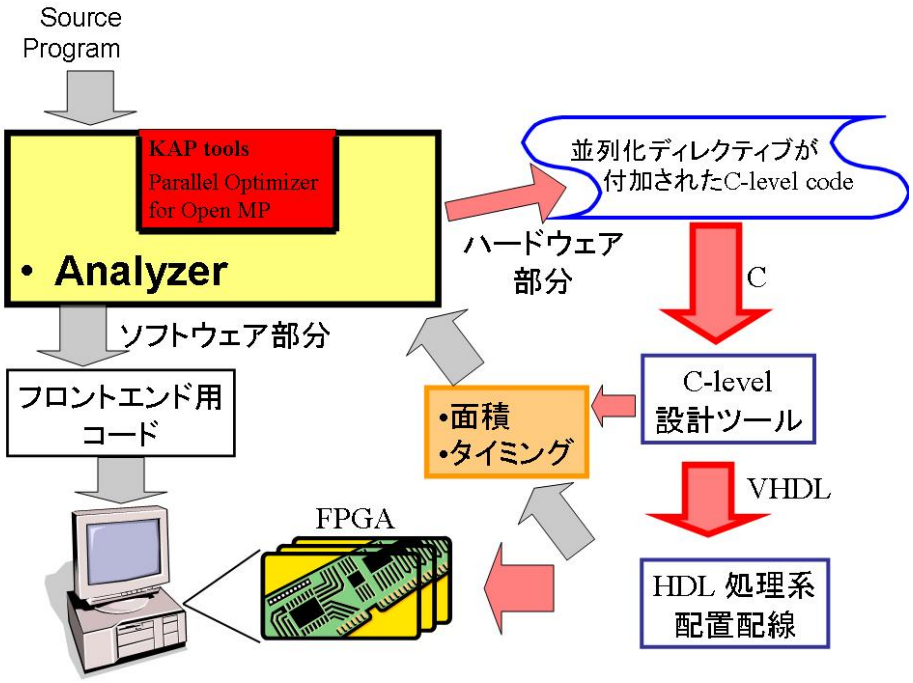

図 1　超細粒度並列処理のためのテストベット

4.2 並列化回路合成スキーム

次に C 言語のソースコードから並列演算器を合成するための手順を確立した．従来の C レベル設計ツールでは，C レベルのソースコードを与えると逐次演算回路を合成するが，逐次演算回路では利用可能なゲート数が増加しても処理速度の向上に結びつけることができない．一方，ハードウェア記述用の C レベル設計ツールは，並列な演算器を合成するための手段が用意されている．例えば，”par”ディレクティブを指定すると，par 文に続くブロック内のステートメントが並列化回路として合成される．そこで，ループを展開後，平行して実行可能な演算子群に par ディレクティブを挿入することによって，並列化演算回路を合成する手法を開発した．ディレクティブ挿入の際，従来のソフトウェア超並列処理で用いられてきた並列性解析ツールの出力を参照しながら，回路合成に必要なディレクティブを挿入する．

4.3 チップ面積を最大効率で使うための並列化最適化手法

プログラム内の各行が並列化可能だとしても，論理回路量には制限があるので，与えられた回路量の範囲内で最も効果的に並列化する必要がある．つまり，並列化しても回路量の増加が少なく処理速度向上の効果が大きい部分の並列度を高くする一方，並列化すると回路量が大きく増加する割には処理速度向上の効果が少ない部分は並列度を低くする．プログラムの各部分の並列度を決定するため

の手法を明らかにした. 並列度決定は以下の手順で行なう.

Step1: ハード化部分の選択

最初にソフトウェアとしてプログラムを実行し, 関数ごとに実行時間の割合を求める. この分析で処理時間を多く消費すると判定された関数が, ハードウェア並列化の対象となる.

Step2: 逐次版での実行速度/面積評価

行ごとに適切な並列度を決定する. コードプロファイラを用いて行ごとの実行時間を求める. 行 *i* の実行時間を *T(i,1)*と置く. *T*の最初の引数は行番号, 2つ目の引数は並列度で, 逐次版なので並列度は *1* となる. また, 対象となっている関数全体の実行時間を *Tseq* と置く. C レベル設計ツールを用いて, 逐次版の関数を論理回路に展開した時の使用論理回路量を求め, *Aseq* と置く.

Step3: 試行的並列化によるデータ収集

行 *i* をある並列度 Pt で並列化し, 使用回路量を評価する. この時の論理回路量を *A(i,P_t)* と置くと, 行 *i* の 1 並列当りに要する論理回路量は次式で記述できる.

$$\Delta A(i) = (A(i,Pt) - Aseq)/(Pt - 1)$$

Step4: 並列化後の実行時間/面積の推定

これらの情報を元に, 各行の並列化後の実行時間と面積を推定する. 行 *i* を *Pi* 並列で実装した時の推定実行時間 *T(i,Pi)*および面積 *A(i,Pi)*は次式で推定する.

$$T(i,Pi) = Tseq + T(i,1)(1-Pi)/Pi$$

$$A(i,Pi) = Aseq + \Delta A(i) \cdot Pi$$

Step5: 他の行の評価

Step3, Step4 の手順を他の行についても適用し, 各行ごとの並列度の効果を求める.

Step6: 行ごとの最適並列度の決定

計算時間を多く消費する各行に対する並列化後の実行時間と面積の推定結果より, どの行を何並列するのが最適なのか求める. これは, $\Sigma \Delta A(i) \cdot Pi + Aseq \leqq Achip$ の条件下で $\Sigma T(i,Pi)$ を最小にする *Pi* を求めることによって, 各行ごとの最適な並列度を算出できる.

4.4 応用として電子透かし検出アルゴリズムへの適用

本並列化実装手法の実用的な応用として音楽向け電子透かし検出プログラムに適用し, 性能向上を確かめた. 電子透かしは, デジタル化著作物の著作権保護や流通の制御を可能とする技術として, 近年注目されている技術であるが, CPU を用いた処理では十分な処理速度が得られない問題がある. そこで, 本研究の並列化手法を適用して, 電子透かし検出を並列化されたハードウェア回路として実装し, 電子透かし検出速度の大幅な向上を試みた.

電子透かし技術は, 著作者の情報などのビット列を人間が知覚できないようにデジタル化著作物に埋め込む方法である. 電子透かしの検出には様々なアルゴリズムが提案されているが, 本実験で用いた P. Bassia (2001)らによるアルゴリズムは, プログラム内の並列性が非常に高く, 本並列化手法

による大幅な高速化が期待できる.

表1に本手法によって合成した並列化検出回路の実行速度を示した. この例では, 44.1KHz で PCM 録音された 20.4MByte の wave ファイルの左チャネルに透かしを埋め込み, この音楽ファイルを入力とした時の検出速度を評価した. ソフトウェアによる検出では, 透かしの種類の数だけループを繰り返す必要があるため, 透かしの種類が増加すると検出速度が低下する. これに対してハードウェア並列化された検出回路では, 透かしの種類が増えても, それぞれの透かしのための検出回路は, チップ内に独立した並列化回路として合成され, これらが同時に動作するので, 透かしの種類が増えても検出速度は低下しない. 50 並列の透かし検出回路は, ソフトウェアに比べておよそ 150 倍の検出速度を達成できたことが分かる. 表2には, 合成された回路の使用ゲート量, チップ上の全ゲートに対する使用ゲート量の割合, およびクリティカルパス長を示す. 合成された回路は C 言語で記述されているが, 十分実用的な回路量で実装できていることが分かる. クリティカルパス長は, 回路の動作速度の指標である. 合成された回路の性能は, 従来の回路設計で行なわれている人間が最適化した回路に比べると決して高くはないが, ソフトウェアからの並列化回路合成の容易性という点で十分な成果だと言える.

表1　透かしの検出速度と回路規模

	ハードウェア検出		ソフトウェア検出	
	検出時間	検出速度	検出時間	検出速度
25-並列検出	500ms	326.4Mbps	38,181ms	4.2Mbps
50-並列検出	500ms	326.4Mbps	74,351ms	2.2Mbps

表2　回路量とクリティカルパス

	使用 LUTs 数	回路使用量割合	Critical Path
25-並列検出	29,485	31%	24.727ns
50-並列検出	53,660	57%	24.867ns

5　自己評価:

本研究では, ソフトウェア内の演算並列性を陽に引き出し, 大規模化する VLSI 内に専用並列化回路を合成し, 増大する論理回路を処理速度の向上に結びつけることを目標に研究を行なった. 研究成果として, これまでの超並列処理技術の助けを利用して, ループを独立した演算に並列展開する方法や並列展開されたステートメントを並列化演算回路として合成するための並列化指示行の挿入法, および並列度の決定法などを明らかにした. 更に, 実用的なアプリケーションの並列化合成を行なったところ, C 言語で記述された電子透かし検出アルゴリズムが容易に並列化回路として実装でき, その回路はソフトウェアに比べて十分な速度を得ることができた.

当初はこれらの処理過程を全自動で行なう予定であったが, 処理段階の一部で用いる並列化支援

ツールやFPGA実装ツールの動作環境の都合上，全自動合成が達成できていない．しかし，これらは手段の問題であり，知見としての並列化回路合成手法は当初の目標通り得られたものと考えられる．

今後の予定としては，現在回路の並列度を求めるために非常に長時間を要する問題があるので，予測に基づく並列度決定法の開発を予定している．更に，現在はプログラム単位の専用回路合成となっているが，近年開発されているリアルタイム書き換え可能デバイスを用いれば，ソフトウェアの関数単位ごとの専用回路にすることができるので，これらの新デバイスを用いて処理効率の一層の向上を試みる予定である．

6 研究総括の見解:

近年のハードウェア技術の進歩によりVLSIのゲート数は飛躍的に増大しているが，それらを有効に活用する技術が十分には確立されていない．井口氏の研究は，ソフトウェアの一部を直接ハードウェア化することにより，高速な実行を行わせようとするものである．C言語で記述されたソフトウェアを対象にして，並列化の効果の高いループ構造の展開や，コンパイラへの並列化ディレクティブ挿入による並列化の指示などにより，並列性の高いFPGAを合成する方式の研究を行ったものである．超細粒度並列処理のためのテストベッドの構築，並列化回路合成スキーム，チップ面積を有効活用する並列化手法などの研究を行うと同時に，電子透かし検出アルゴリズムなどの実例に対してその有効性を確認するなどの成果を上げている．各種組み込みシステムなどへの応用も十分に見込めるものであり，今後重要となる技術のさきがけとして大いに評価できる研究である．

7 主な論文等:

論文・口頭発表

1) 榊原 憲宏，井口 寧，“FPGA を用いたオーディオ電子透かしの超高速検出”，電子情報通信学会論文誌 D-II (条件付採録済み)
2) M.M. Hafizur RAHMAN, Yasushi Inoguchi and Susumu Horiguchi, “Modified Hierarchical 3D-Torus Network”, IEICE Transaction on Information System, Vol.E88-D, No.2, pp. 177-186, Feb. 2005
3) Y. Inoguchi, “Outline of the Ultra Fine Grained Parallel Processing by FPGA”, Proc. in IPSJ and IEEE, High Performance Computing in Asia Conference 2004, Workshop on Reconfigurable System for HPC, pp. 434-441, Jul., 2004
4) 井口 寧,”FPGA を用いた超細粒度並列処理の概要”，信学技報，第一回 リコンフィギャラブル研究会 論文集, pp.1-6, Sep.18, 2003

特許

1) 井口 寧，榊原 憲広，”電子透かし検出装置，それを内蔵する中継装置，及び，電子透かし検出方法”，特願 2004-230360, 2004