並列論理型言語GHCによる論理回路設計ルール検証プログラムの記述

5Q-7

佐藤令子' 太組 孝' 森 和男"
'三菱電機(株)情報電子研究所 "(財)新世代コンピュータ技術開発機構

1. はじめに

並列論理型言語と呼ばれる言語が設計され、それらの処理系が開発されるにしたがって、「どのような問題を取り上げて並列にプログラミングするのか」と言う問題が出てきた。本稿では、この問題に答えるための試みの一つとして、CAD分野の論理回路設計ルール検証プログラムを取り上げ、並列論理型言語GHC[Ueda](Flat GHC)を用いてその記述を行った。

CAD分野のプログラムを取り上げた理由としては、

① 高速かつ大容量処理を求められる分野であるため、並列処理のニーズが多い。

② 論理回路という対象の構造が並列プログラム向きである。

等があげられる。

またこのプログラムの記述を行うことにより、並列プログラム向きの問題の種類に関する考察を行った。

2. 論理回路とアルゴリズム

論理回路を見るときに、各素子が永続的なプロセスであり、その間をデータの流れるストリームが繋いでいると考えることは、ごく自然(図1)である。逐次型の言語では意識して記述しなければならないデータ(素子に対するピンからの入力)の待ち合わせなども、並列型言語を用いれば、言語の同期機構をそのまま用いることによって可能とすることができる。

現在用いられている検証プログラム[Muroi]は、従来の手続き型言語で記述され、マシンも言語も逐次型であって、入力信号を仮定し、その信号に様々な情報(どのような素子を通ったか、信号はどのように変換されたか、など)を付加しながら回路上をトレースする、というアルゴリズムを採用している。

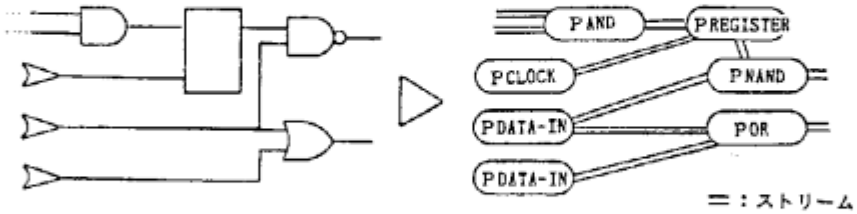

図1. 論理回路とプロセス・ネットワーク

これに対し、前述の、素子=永続的プロセス、結線=ストリームの考え方を用いれば、該当する全ての永続的プロセスを起動し、データ・ドリブンに動作させるだけで良い。

このように、並列であることを前提にプログラムの設計を行なうと、論理回路の場合非常にアルゴリズムの設計が容易であるように思われる。これは、対象自身の並列性によるところもあると思うが、大部分は、対象を見てプログラミングする人間の意図によっているということではないだろうか。つまり、並列アルゴリズムを考えるためには、対象を並列にみる努力が必要かつ重要だ、ということだと思われる。

3. 論理回路設計ルール検証プログラム

本稿で示すプログラムは、論理回路を設計する際に設計者が守らなければならない多数のルールに対して、設計済みの回路が適合しているかどうかを検証するためのものである。(設計ルールを厳守することにより、設計後の様々な試験が容易になる、という効果がある。)設計ルールとしては、以下の2つを取り上げた。

①ゲート・ループ禁止規則

「基本素子から成るループがあってはならない。」

※基本素子:AND、OR、EXOR、NAND、NOR、EXNOR などのゲート

②同相転送禁止規則

「2つのレジスタ間でデータを転送する場合、両レジスタのクロックが同相であってはならない。」

これらの設計ルールを検証するための方法として、2種類のアプローチが考えられる。

1)トポロジカルな方法

2)セマンティックな方法

前者は純粋に回路構造に対して行われる探索であり、後者は、論理機能に対して行われる探索である。上に述べた2つの設計ルールは非常に単純なものであるため、セマンティックな探索を行なう必要はなく、トポロジカルな探索で済む。

この2つの設計ルールを検証

The verification system for logic circuits written in GHC
Reiko SATOH', Takashi DASAI', Kazuo TAKI"
'MITSUBISHI ELECTRIC Corp., "ICOT

するためのトポロジカルな（並列）アルゴリズムは、非常に単純である。

①ゲート・ループ禁止規則

1)各々のゲートをプロセスと考え、各ゲートの入力端子から回路をトレースして、隣接した素子を発見する（素子：外部入／出力端子、基本素子、レジスタ）。

2)発見した素子が

・レジスタあるいは外部入／出力端子

ループは存在しないので、トレースを中止。

・ゲート

そのゲートがそこまでのトレースの履歴に含まれていたら、ゲート・ループが存在。含まれていないなら、そのゲートをトレースの履歴に加え、新たにそのゲートの隣接する素子を発見しにいく。

②同相転送禁止規則

1)各々のレジスタをプロセスと考え、各ゲートの入力端子から回路をトレースして、隣接した素子を発見する。

2)発見した素子が

・レジスタ

そのレジスタが、プロセスを産んだレジスタと同相のクロックを供給されていたら、同相転送が行われている。クロックが同相でなければ、同相転送は行われていないので、トレースを中止。

・外部入／出力端子

同相転送は行われていないので、トレースを中止。

・ゲート

ゲートをトレースの履歴に加え、新たにそのゲートの隣接する素子を捜しにいく。

4．実行結果と問題点

ここでは、トポロジカルな探索を行なう前述のプログラムを作成し、実行した。実行に用いた環境（処理系）が逐次マシン上の逐次型処理系であるため、単純に実行速度を測定しても意味がない。そこでFORTRAN で記述された実用版のアルゴリズムとＧＨＣ版で用いたアルゴリズムについて計算量の比較を行なってみた。

回路上の全素子数をｎ、ゲート数をｇ、レジスタ数をｒ、Ｃ１、Ｃ２を定数とすると計算量は、

・ゲート・ループ禁止規則

実用版	Ｃ１・ｎ	
ＧＨＣ版	ｇ・n^2	（但しｇ≒ｎなのでn^3に近いと考えられる）

・同相転送禁止規則

実用版	Ｃ１・ｎ＋Ｃ２・ｒ	
ＧＨＣ版	ｒ・n^2	（但しｒ＜＜ｎなのでn^2に近いと考えられる）

となる。対象が論理回路であり、実用的にはｎ＝10^3～10^5であるから、ＧＨＣ版の計算量は実用版に比べてかなり大きくなってしまう。ＧＨＣ版を実用とするためには、もっと計算量の小さいアルゴリズムを考える必要があるようである。計算量が大きくなる原因の一つとして、リストでデータを持っているため、そのサーチがオーダｎになってしまうということがあげられるので、配列、表等を用いてそのサーチをコンスタント・ファクタにするというアプローチも可能だと思われる。

また、現在のプログラムは、全く無作為な回路の探索を行なっているので、探索木の刈り込みを行うことによって効率が上がる可能性がある。反面、無駄な探索を意図的に減らそうとして、データのサスペンションが増えてしまう可能性もある。並列プログラムの記述を行なう時には、このようなトレード・オフが問題になっていることが多く今後の検討が必要であると思われる。

5．考察

今回のプログラミングが非常に容易だったことから考えて、論理回路と並列処理はかなり親和性が高いと思われる。論理回路は、処理のイメージと直結し易く、ほとんどそのままプロセス・ネットワークに置換えることができる。このことから、大雑把な言い方ではあるが、プロセス・ネットワークが問題（処理の対象）からイメージしやすいものは、並列プログラム向きといえるのではないかと思う。

6．おわりに

本研究では、「論理回路設計ルール検証プログラム」をＧＨＣで記述することによりＣＡＤ分野への並列プログラムの適用を試みた。結果についてはあまり有望とはいえなかったが、今後セマンティックな方法を用いたプログラムの設計と作成を通じ、さらに検討を行なっていく予定である。尚、本研究はＩＣＯＴプロジェクトの一環であり、ＧＨＣのプログラミング環境としては、ＰＳＩ上のＦＧＨＣ逐次型処理系と、Ｐｓｅｕｄｏ　ＭｕｌｔｉＰＳＩ処理系を用いた。最後に、本プログラムの作成に当たりご協力いただいた三菱電機カスタムＬＳＩ研究所の小迫靖志氏に深く感謝致します。

＜参考文献＞

[TAKAGI87] edited by S.Takagi ;
'A Collection of KL1 Programs -Part1-'
ICOT TM-311 (to appear).

[Ueda] Ueda, K. ;
'Guarded Horn Clauses'
ICOT TR-103.

[Muroi] Muroi, K. , et al. ;
'A HIERARCHICAL LOGIC DESIGN CHECK PROGRAM FOR SCAN DESIGN CIRCUITS,'
ICCAD'86, 1986.