VLSI の並列論理シミュレーションに関する考察

9JET1238　高田 賢吾

東海大学工学部通信工学科

指導教員

清水 尚彦 助教授

1　はじめに

VLSI の並列シミュレーションに関する過去 10 年間をまとめた論文の和訳及び、それについての研究を行った。

2 INTRODUCTION

巨大なデジタル･システムの設計、特に VLSI システムの設計は総合的なデザイン過程における、論理シミュレーションの重要性を増強しました。
システムが成長するのに従って、シミュレーションタスクはデザインサイクルで重要なボトルネックになりました。

このボトルネックを扱う試み対し、研究者は平列と分散処理に変わりました。VLSI システムは大きな合意を示します。(それは、彼らの通常操作に固有です)。シミュレーションアルゴリズムの平行なバージョンでシミュレートされたシステムに固有の合意を利用することができるなら、シミュレーションタスクを実行して、ユニプロセッサ構造の上で顕著な性能改良をもたらすのにパラレル･プロセッサを使用することができます。並列処理を使用することによって、2 から 3 桁の性能改良が達成可能であると信じられることは無理ではありません。

5 つの重要な要因が平列論理シミュレーションの性能に影響します。

(1) 同期アルゴリズム
(2) サーキット構造
(3) タイミング粒状
(4) 目標構造
(5) パーティションとマッピング

同期アルゴリズムは複数のプロセッサーを横切ってシミュレーションを調整するために使用されます。
多くの同期アルゴリズムは、同時のアプローチ、保守的なアプローチおよび楽観的なアプローチを含む並列マシン上の個々のイベントにおけるシミュレーションのために提案されました。
さらに、私たちは、イベントに基づかない代替同期アルゴリズム(忘れているアプローチ)について議論していきます。その入力ベクトルと同様に VLSI システムの回路構造も、並列のシミュレーションの実行に劇的な効果がありえます。
問題があると分かっていた一方、いくつかの並列回路シミュレーションはよいパフォーマンスを示します。

同じ回路において、同期アルゴリズムを用いて、異なる入力ベクトル与えることによって劇的に異なる実行を得ることができます。
根本的なロジック･シミュレータのタイミング粒状は、さらにシミュレーション実行に効果があります。きめの細かいこと(例えば 0.1 の ns 時間解決)から粗粒子状の粒状(例えばユニット遅延あるいは 0 遅延)まで及ぶタイミング粒状の広いスペクトルがあります。
それがすべての並列プログラムを行うように、目標アーキテクチャーは並列シミュレーションの実行を密着させます。関連する問題は並列プロセッサー中のシミュレートされた回路の分割です。
並列シミュレーション･アルゴリズムのうちの 1 つを始めるに先立って、回路素子は署名されたか、個々のプロセッサーに写像されたとともに分割されるに違いありません。
並列マシン上のタスク割り当ておよびロード･バランシングの問題。これらの要因の多くは、すべての並列シミュレーションの中にあります。確かに、過去数年にわたって一般的な平列、分配シミュレーションに多くの仕事がありました。不運にも、一部アプリケーションの多くの種類は限られた一般的な結果になりました。論理シミュレーションの大部分はその潜在的な経済的影響のために、重要な注目を集めた 1 つの適用エリアです。スミス[1986]は並列の論理シミュレーションの状態を評価しましたが、多くの研究は 1986 年以来実行されました。
この調査では、私たちが、5 つの要因に注目することにより並列の論理シミュレーションの現在の最先端技術について議論し分析していきます。同期アルゴリズム、回路構造、タイミング粒状、目標アーキテクチャーおよび分割すること。

この調査は論理シミュレーションの簡潔な概観から始まります。次に、私たちは、論理シミュレーション、および並列シミュレーションを同等の連続するものと一致しているようにしておくのに必要な同期アルゴリズムを対応させるための共通のメカニズムを調査します。

3 論理シミュレーション

VLSI回路は、回路レベルから動作レベルまで多数の抽象レベルにおいてシミュレートされます。回路レベルシミュレーションでは、ノード電圧が連続的な値によって表わされます。また、シミュレータは、回路を表わす微分方程式を数的に解決します。論理レベルシミュレーションでは、ノード電圧が個別の量および時間における個別の遷移状態によって表わされます。論理シミュレーションという用語は多くの分野で使用されます。

他方では、回路コンポーネントがトランジスター(理想スイッチとしてモデル化された)と異なるところで、VLSI 回路のどんな個別のシミュレーションも意味するために、ハイ・レベルの動作のモデル(例えばプロセッサー、乗算器)に、基本論理学ゲートを通って、論理シミュレーションを使用して、より広い定義をします。私たちはこの広い定義を無意識に使用します。

最も単純な 2 値論理シミュレーションでは、状態変数が、ブール値(つまり 0 あるいは 1)を表わす2つの量で表されます。最も現代のロジック・シミュレータは、補足情報を表わした多値の変数を使用します。

例えば、多くのスイッチレベルシミュレータが、未知の信号あるいは浮動信号を表わすために X の状態を加えます。また、ゲートレベルシミュレータは、駆動の強度および高いの条件を表わすために状態を加えます。これはタイミング粒状と呼ばれます。また、許しえる時値は時間ポイントあるいは時間ステップと呼ばれます。素晴らしい粒状を使用すると、シミュレータは、実行時間の増加の可能な幅と共に、タイミング依存の振る舞いをより正確にモデル化することができます。

コース粒状を備えたシミュレータは、タイミング振る舞いをほとんど(もしあれば)提供しないが、一般により速い実行が可能です。実行中のタイミング粒状の特定のインパクトはセクション 5 で検証されます。

シミュレーション目的のために、私たちは、抽象(例えばトランジスター、NAND ゲート、フリップ・フロップ、乗数など)およびそれらの相互に連結するワイヤーのレベル論理素子の集合として VLSI システムをモデル化します。

その後、入力ベクトルは回路を検証するために提供されます。3つの論理素子を使用するゲートレベル例は、図1の中で示されます。この例は2つのインバーター(ラベルが付けられたαおよびβ)および単一の AND ゲート(ラベルが付けられたγ)を持っています。相互に連結するワイヤーは直線 e と呼ばれます。

4 並列論理シミュレーション

並列のシミュレーションを実行するに先立って、論理素子は決まって個々のプロセッサーに割り当てられます。その後、各論理素子の機能的な評価はその割り当てられたプロセッサーによって実行されます。正確なシミュレーション時間を維持するために、プロセッサー間の調整をすることは大事です。シミュレーションクロックはこの調整のための通常のメカニズムです。
連続するイベントドリブンシミュレーションでは、イベントが決まった時系列のキュー上で維持されます。イベントがキューから取り除かれるとともに、シミュレートされた時間は更新され、イベント評価されます。それは他のイベントをキューに置かせてもよい。並列シミュレーションでは、多くの場合 1 台のプロセッサー当たりに多数のキューがあります。

イベント評価の調整およびこれらのキューの管理は正確なシミュレーションを保証するのに必要です。正確さを保証するためのいくつかのメカニズムがあります。私たちはこれらを時間同期法と呼びます。私たちは、このセクションで最も一般的な同期法を要約します。一般並列個別のイベントシミュレーションにおける研究の現状についてのより多くの完全な記述については、Fujimoto[1990]を参照してください。

4-1 Oblivious Simulation（気付かないシミュレーション）

忘れている方法はイベントドリブンではありません。代わりに、それらの入力が変わっても変わっていなくても、すべての回路素子はすべての時間ステップの間に評価されます。仕事量は、ここに各時間ステップで固定します、したがって、スケジューリングで静的に実行することができます、時間をコンパイルする、また、オーバーヘッドを予定しないことはランタイムで招かれます。
ランク要求は、要素評価を予定する手段としてこれらのシミュレータの中でしばしば使用されます。すべての要素、一般にゲートの入力の有効性によって命じられます。さらに、その入力がシミュレーションへの入力であるゲートはランク 0 にあります。その入力がすべて、ランク i でゲートによって生産される場合、ゲートはランク i にあります。

4-2 同期アルゴリズム

多くの明白な同期アルゴリズムは同時の密集行進法方法の中で同時ステップにプロセッサーをすべて作用させることです。
シミュレートにより生じる時間がすべてのプロセッサーを横切って共通であるので、これもグローバルタイムアルゴリズムと呼ばれます。図 1 のシミュレーション例を考えてください。各ゲートが別個のプロセッサーに割り当てられると仮定します。同時アルゴリズム中のオペレーションのシーケンスはテーブル 3 の中で例証されます。最初に(t=0 ns)、メッセージは、入力ベクトルについて記述するゲートαおよびβへの主要な入力から述べられます。
グローバルタイム t=1 ns では、αが、αから t=6 ns のタイムスタンプを備えたγまでメッセージに送らせて評価されます。時間 t=2ns では、βが、βから t=4 ns のタイムスタンプを備えたγまでメッセージに送らせて評価されます。時間 t=4 ns では、γが、β(より小さなタイムスタンプを備えたメッセージ)からのメッセージを評価します;また、時間 t=6 ns では、γが、t=9 ns のために出力にメッセージを送って、αからのメッセージを評価します。
2 つの評価がシミュレートされた時間で同じポイントが生じないので、並列はこの例において開発されません。現実的な回路において利用可能な並列の量はセクション 5 に検証されます。
このアルゴリズム中の障害はプロセッサーがすべて時間ステップ、および次の時間ステップがであるべきものをいつ終えたか断定することを含んでいます。完成の決定は、並列のアーキテクチャーかソフトウェアに支援されるかもしれない、単純な障壁で遂行されるかもしれません。
イベントがどのように管理されるかによって次のタイムステップが決定する。中央のイベントキューがある場合、中央のイベントキューは単に最低の時間を見つけます。しかしながら、キューからの挿入および削除は連続させることができます。各プロセッサーがローカルのキューを持っている場

合、グローバルな最小のオペレーションが行なわれるに違いありません。さらに、別の問題としてロード･バランシングのそれへ発展します。プロセッサーが恐らく与えられた時間ステップの間に活発なイベントの異なる数を持つので、いくつかのプロセッサーは潜在的に重要なロード･インバランスに帰着して、他のものより早く終了するかもしれません。

4-3 保守的な非同期アルゴリズム

ロード･インバランスおよび中央のキューの主張の問題を縮小するために、独立したキューおよび時計を備えた独立した割合でプロセッサーが進むことを可能にするアルゴリズムは魅力的です。各プロセッサーあるいは論理素子がそれ自身のローカルのシミュレーション時間を維持する場合、アルゴリズムはローカルクロックあるいは非同期のアルゴリズムとして知られています。ローカルクロックのアルゴリズムには 2 つのクラスがあります。保守的と楽観的です。
図 3 は、個々のプロセッサーに割り当てられた各ゲートを備えた回路を示します。各プロセッサーに関連したクロックシンボルは、プロセッサー内に、シミュレーション時間がローカルに維持されるという事実を示します。ローカルのシミュレーション時間の価値は、したがって 1 台のプロセッサーから次のものとは異なるかもしれません。
保守的な非同期アルゴリズムは、Chandy と Misra[1981]のそれらの起源およびブライアント[1977]を持っています。それらは、論理素子に関連したローカルのシミュレートされた時間が程度まで単に進められることを必要とします、それ、その、進歩がモデル化されている(つまり、論理素子が t にそのローカルのシミュレートされた時間を進?る前にそれは t 未満のタイムスタンプを備えた追加のメッセージを受け取らないだろうことを知るに違いありません)システムでの因果関係を破ることができないという程度まで進められました。メッセージのタイムスタンプに関する結論を引き出すことができるために、それは今後受け取るかもしれません。保守的なアルゴリズムは、ある論理素子から別の論理素子までのメッセージが減少しないタイムスタンプ順に送られることを必要とします。

4-4 楽観的なアルゴリズム

オリジナルの楽観的な非同期なアルゴリズム(Time Warp)はジェファーソン[1985]によって工夫されました。
論理素子でメッセージを受け取るときはいつも、ローカルのシミュレートされたメッセージに関するタイムスタンプの時間や入力メッセージの効果をシミュレートなどの過程を進める。
このシミュレートされた時間の進歩は、将来のメッセージには下側のタイムスタンプがあるかもしれないという事実を前提に実行されます。その結果、潜在的にオリジナルのメッセージが到着したとき行われた仕事を無効にします。
最初の例のサーキット(図 3)では、シミュレーションはゲートの入力からαとβまでのメッセージがある保守的なアルゴリズムのように始めます(Table5 を見てください)。
それぞれメッセージをγに送って、ゲートαとβは(平列で)ともに評価されます。
γはこれらのメッセージの両方をどんな追加メッセージもβから 4ns と 6ns の間来ないと仮定して、そして t=6ns まで楽観的にγの現在の時間をアップデートし評価します。
ゲートγは t=9ns 間の出力メッセージを送ります。
2 番目の例のサーキット(図 4)では、メッセージが全く「過去」のときに到着しないと仮定して、両方のプロセッサ共にメッセージを処理するでしょう。
先頭のゲートに 3ns の遅れがあるなら、この仮定は成立し、シミュレーションは正しく続きます。
しかしながら、先頭のゲートに 1ns の遅れがあると、下側のプロセッサは、メッセージを 4ns 処理して、3ns 遅れてメッセージを受け取るでしょう(Q から)。
それは 4ns の変化を処理することによってもたらされたダメージを元に戻さなければなりません。
したがって、Q からのイベントは、セットイベントを評価する前の下側のプロセッサが持っていた状態に戻す引き金となります。
さらに、セットからのイベントで、誤ったイベントを先頭のゲートに送ったかもしれません。

このイベントを取り除くために、「反-メッセージ」を下側のプロセッサからトッププロセッサに送るでしょう。
対応する本当のメッセージが処理される前に「反-メッセージ」が届くなら、イベントを取り除きます。さもなければ、「反-メッセージ」は受信プロセッサの上で復帰の引き金となります。
復帰を実行するために、プロセッサはサーキットの状態を節約したに違いありません。
復帰の間、ローカルのシミュレートされた時間は入力メッセージに関連している値により保持されます。論理的な過程に関連しているシステム状態は以前のコピーから回復します。そして、タイムスタンプが新しいローカルのシミュレートされた時間より長い状態でどんな以前に伝えられたメッセージも無効にするために出力チャネルに沿って「反-メッセージ」を出します。
彼らの目的地の論理的な過程でこれらの「反-メッセージ」を受け取るとき、それらはまた、それらの過程で復帰の引き金となるかもしれません。
明確に、Time Warp アルゴリズムは復帰を実行して、状態を節約するためにオーバーヘッドを必要とし、救われた状態はメモリを取り上げます。
メモリ要件を減らすために、もう必要でないと、古い状態を取り除くことができます。
通過におけるプロセッサのローカルとメッセージの最小限はグローバルな仮想の時間(GVT)として知られています。
メッセージが全く GVT より早くプロセッサに到着することができないので、化石と呼ばれる GVT を捨てることができる前に、状態とメッセージは保存されました。
1988 年の Gafni の「怠惰なキャンセル戦略」はシミュレーション性能上の復帰に関する影響を減少させます。
復帰が起こるときはいつも、積極的に以前に送られたメッセージを取り消すことの代わりに、怠惰なキャンセルアルゴリズムは、間違ったメッセージを送ってあったのが知られるまでメッセージを取り消すのを待っています。
したがって、間違った理由で正しいイベントを渡したなら、過度の因果関係規制のために受信プロセッサを禁止しません。
保守的なアルゴリズムのように、異形が楽観的な戦略にあります。
そのような変化の 1 つはソコル他[1998]によって提案されたムービング･タイム･ウィンドウ(MTW)アルゴリズムです。
　それは、復帰するのに最も好ましいイベントが、シミュレートされた時間において最も遠いものであるという観測を利用し試みます。
MTW アルゴリズムは、GVT のすぐ前でウィンドウを確立して、そのタイムウィンドウの中で論理的な過程で達せられるポイントにおいてのみ、ローカルのシミュレートされた時間を許容します。
もし入力メッセージがウィンドウの前にタイムスタンプを持つなら、それはローカルのイベントの待ち行列に置かれて、いったん GVT が高度になると、タイムスタンプがウィンドウの中に下がり処理されます。

5 SYNCHRONIZATION ALGORITHMS 同期アルゴリズム

一般的な並列シミュレーションのための同期アルゴリズムを比較するのは難しいです。
しかしながら、論理シミュレーションドメインには、これらのアルゴリズムを比較するのに形式モデルを使用することで、何らかの成功がありました。
私たちはこれらの結果をここへまとめます。
サーキット平列関係へのタイミング粒状の効果を考えている間、ベイリー[1992b]は 2 つの同期戦略を考えました。同期戦略と理想主義的な保守的戦略。
すべての保守的なアルゴリズムのための実行時間に理想主義的な保守的戦略は低境界(同時であって、保守的な非同期な状態両方を含んでいて)です。
彼女は、オーバーヘッドが無視されて、すべてのイベントが同評価時間を持っていると、理想主義的で保守的なアルゴリズムが、少なくとも同期アルゴリズムと同様に働くのを示します。
2 つのアルゴリズムがユニットディレイ・タイミングのために同様に働くでしょう。
同期アルゴリズムが保守的な非同期なアルゴリズムより一般に簡単であるので、そして、妥当な負荷バランスをとることで、ユニットディレイ・タイミングが使用されているなら、同期シミュレーションが保守的な非同期シミュレーションより優れていると予想するでしょう。
ベイリーと Lin[1993]は 4 つの異なった同期戦略を含むようにこの仕事を広げています。同期戦略、保守的で非同期な戦略、楽観的で非同期な戦略、および保守的な最適戦略。
保守的な最適戦略はイベントの最適なスケジューリングを構成するのに、与えられたプロセッサに関するメッセージがタイムスタンプオーダーで評価されるという規制でシミュレーションし、すべてのイベントに関する知識を使用する人工の戦略です。
2 つの仮定により、すべての同期戦略が分析、制御しやすくなりました。
まず最初に、各論理素子に関連した不変の、積極的な時間の遅れがあると思われます。
これは、いくつかのシミュレータに起こることができるゼロ遅延を持っているので、素子を排除します。
また、これは、RNL ターマン[1983]で見つけられるような同じ要素のために異なった時間の遅れを持っているのを排除します。
2 番目に、それ自身のプロセッサ上にあらゆる評価素子があると思われます。
ベイリーと Lin の最初の結果は、同期戦略が保守的な最適戦略より遅いことを示しました
コミュニケーションコスト、グローバルイベント待ち行列を維持するための排除コスト、そしてそれぞれの時間ステップの終わりの時の同期は、同期シミュレーションにおいて取るにたらないと思われます。
次に、保守的な最適戦略は、null メッセージがある保守的な非同期戦略より速くなるように示されます。
保守的な非同期戦略のためのコミュニケーションコストはゼロであると思われません、ヌルメッセージの存在がコミュニケーション構造の増加するリソース主張か評価時間 ont を取るのによる目標プロセッサでシステムの性能を下げないと思われますが。
同時の、そして、保守的な非同期戦略を比較するのにおいて複雑な結果があります。
サーキットが強くつなげられるなら、論理シミュレーションのための思いも寄らない事態であり、保守的な非同期戦略は優れているかもしれません。
残っている結果は Time Warp か楽観的な戦略に関係します。
状態を節約するコストは復帰の間、状態を回復するコストと同様に無視されます。
送付「反-メッセージ」のコストなどの他の復帰コストは含まれています。
これらの仮定で、攻撃的であるか怠惰なキャンセルがある Time Warp は保守的な最適戦略より優れています。
また、限られたプロセッサの問題を記述します。
与えられたプロセッサの上のすべての論理素子が、ただ一つの過程であるとみなされるなら、上の分析は成立します。
しかしながら、これはそれぞれの個々の論理素子と対照的に、論理ブロックへのすべての入力が知られるまで進行が遅れるのを意味します。

これは性能を下げることができます。
同様の問題は復帰のときに Time Warp に起こります。
この仮定で、まさしく「反-メッセージ」を受ける素子をロール・バックする代わりに全体の論理ブロックはロール・バックされます。
これらの制限がなければ、上の結果を立証することができません。より多くの研究が、これらの問題を記述するのに必要です。

6 CIRCUIT STRUCTURE AND TIMING

シミュレートされるサーキットとサーキットを運動させるのに使用される入力ベクトルの固有情報はシミュレーションアルゴリズムの性能に大きい影響力を持つことができます。
サーキット構造はサーキットトポロジー、サーキットサイズ、抽象化レベル、fanout、フィードバック、サーキットタイプ、およびサーキット活動のような局面を含んでいます。
サーキットトポロジーは回路素子の間のインタコネクトパターンを参照します。
抽象化レベルは個々の要素(例えば、スイッチレベル、ゲートレベルなど)のために考えられた基本的なモデルです。
サーキットタイプはそのデザインスタイルと目標に関してサーキットを分類します、組み合わせ回路と順序回路(時間を計られて自己によって調節されたサーキット)を見分けて
頻繁に、信号が値、同時の値の変化の数などを変えるのを値がどのようにを変えるかという信号のダイナミックな本質にサーキット活動を心配させます。
サーキット構造と他の素子の間の相互関係はサーキット構造だけが平列シミュレーションの性能のときに持っている影響力を隔離するのが難しく、そして(不可能)であるほど重要です。
この理由で、サーキット構造の影響は主として個別でというよりむしろ他の素子に関連して記述されるでしょう。
これへの例外はサーキット活動です。(その活動は広範囲な研究を受け取りました)。
性能に影響する素子の中の特に理解されている関係の 1 つはサーキット活動とタイミング粒状の間の関係です。
この領域での研究は、単にサーキット活動を測定することによって始まりました。
より最近、サーキット活動とタイミング粒状を関係づける形式モデルが開発されました。

6-1 Circuit Activity サーキット活動

VLSI デザイナーは長い間、彼らのサーキットで活動を測定したがっていました。
サーキット活動には、例えば、それは直接 CMOS デザインにおける所要電力に影響するなどの論理シミュレーションに沿うより広い関心があります。
ラトナーは、1970 年代前半にシミュレーション走行(個人的なコミュニケーション)の間アクティブであったゲートの平均した数を測定するために論理シミュレータに器具を取り付けました。
彼は、その時々でシミュレーション走行の間、平均しておよそ 2.5 パーセントのゲートがイベント待ち行列上にあったのがわかりました。
数年後に、サーキット活動における研究はイベント駆動の平列シミュレーションにおける重要性のため増加しました。
焦点は、イベント待ち行列のシミュレーション素子の測定から同じ時間ステップで評価されたシミュレーション素子の平均した数の割合を測定することへ変化しました。
フランク[1985;1986]は平行なデータ駆動型論理シミュレーションエンジン(Fast-1)への彼の作業の一部として、サーキット活動について多くの研究を発表しました。
このシミュレーションエンジンがイベント駆動のアルゴリズムを使用したので、サーキット活動へ潜在的スピードアップとして影響を及ぼされました。
フランクは直接サーキット活動を測定しないで、多くの命令が連続し、並列なバージョンであるという考えによって、uniprocessor バージョンを超える平列 Fast-1 の潜在的スピードアップが必要であるとむしろ見積もりました。
連続した命令の数と平列な命令の数の比率はスピードアップとサーキット活動の概算の上限を規定しました。
78 から 2 万 300 トランジスタのサイズの幅のある 13 のサーキットを使用したところ、彼は、潜在的スピードアップが 4.1 から 192.1 (平均 49.5) まで及んでいるのがわかりました。
低値はフランクを驚かせました、そして、彼は平行な Fast-1 エンジンの可能性に関して楽観的ではありませんでした。

フランクの研究のすぐ後に、他の実験が、平行なイベント駆動のシミュレーションの可能性を考えるのに既存の連続したシミュレータを使用することで行われました。
通常、これらの実験に使用されるメートル法はサーキット平列関係と呼ばれます。(それは、活発なシミュレーション時間のステップ単位で実行された出来事の平均した数になるように定義されます)。
イベント駆動のシミュレータでそれらをスキップするためのオーバーヘッドが全くないので、イベントが全く実行されない時間ステップは無視されます。
サーキット並列関係は、1 つが得ることができるスピードアップのときに並列であると同時に、イベント駆動のシミュレータを使用することで上限を提供しました。
ウォン他[1986]は 1 番目に実際のサーキット平列関係測定値を報告します。
彼らは、ゲートとスイッチレベルシミュレータを使用して、固定ディレイ・タイミングを使用する 650－8000 個のトランジスタまで変化する 5 個のサーキットの平列関係を測定しました。
平列関係値は平均 18.6 で 2.1－55 まで及びました。
彼らは、10 万個のコンポーネントサーキットのサーキット並列性を見積もるためにこれらの平列関係値をスケーリングしました。
スケーリングされた値は平均 1,279 で 80－3,294 まで及びました。
フランクと対照して、ウォン他はスケーリングされた平行関係値に基づいて、平列シミュレーションの可能性に関して楽観的です。
スーレ、Blank[1987]、およびスーレ[1992]はサーキット平列関係で異なった抽象化レベルの影響を考える最初の研究者でした。
4 つの異なった抽象化レベルが提示されました。命令、行動の RTL、およびゲートトール、多レベル(イベント駆動のシミュレータ)は、4 つの抽象化レベルを使用することで 3 個のサーキットの理想化されたスピードアップを測定するのに使用されました。
2 個のサーキット(3400 と 5000 素子)がゲートレベルでシミュレートされました。
3 番目のサーキットは、異なった機能的なシミュレータを使用することでシミュレートされました。
イベント評価のためにコストを全く持っていない平列で「理想的な」シミュレータ（スケジューリング、メモリ競合がなく、および等しいコスト）でイベント跡をシミュレートすることによって、スピードアップ測定値を得ました。
1000 台のプロセッサに関しては、スピードアップは 1 個のサーキットと抽象化レベルでは 10 未満でした。(そこに、ほぼ 100 にはスピードアップがありました)。
さらに、彼らは、スピードアップが任意な特定の時間の点 0.1%から 0.5%の間で、4 つの抽象化レベル、および素子活動の上で比較的一定であることがわかりました。
次の 2 年間、ベイリー[1992a]、ベイリー、およびスナイダー[1988]は、スイッチレベルシミュレータ RNL を使用することで追加サーキット平列関係測定値を提示しました。
RNL は電圧で制御されたスイッチとの抵抗シリーズとしてトランジスタをモデル化して、0.1 ナノセカンド間の解決 Ternab[1983]を時間見積りに提供します。
これらの測定値で使用される 9 個のサーキットが 200 から 61600 トランジスタに変化しました。
結果として起こるサーキット平列関係値は 2.8 と 23 の間に及びました。
異なった活動メートル法であることで含まれていたベイリー[1992a]、待ち行列におけるメートル法の測定値。(その測定値は、より密接にラトナーの早めの測定値に対応します)。
シミュレーションの平均した長さのメートル法が列に並ばせる待ち行列。
イベント待ち行列の電流で実行されない通常より高い追加素子が時間であったなら待ち行列のメートル法の意志を使用することで測定された値は踏まれます。
同じ 9 個のサーキットを使用して、それらは、0.22%から 8.9%の間の待ち行列にノードのがそれぞれの時間のステップにあったのがわかりました。
これらの値で、彼の測定値でラトナーよりはるかに多くの変化を見つけます。
さらに、ベイリー[1992a]は、一般に、サーキット平列関係がウォンの楽観的な平列関係測定値で想定されたサーキットサイズで直線的に比例しないのを示すために実証的証拠を提示します。
1 サーキット族、シフトレジスタでは、平列関係はほとんど直線的に比例しました。
他のサーキット族にとって、これはそうではありませんでした。

サーキットサイズに従って、平列関係は一般に増加しますが、それは簡単な一次関数ではありません。
活動、Briner[1988]、および Briner 他を定義するメカニズムとしてサーキット平列関係を使用するよりむしろ 1988 は新しくメートル法で a について工夫して、見積りへのメートル法のわたるのはモデル評価の相互依存を使用する平行なシミュレーションの可能性です。
平列関係の 2 つの追加源がこのテクニックで測定されます。
信号変化が fanout による 1 つ以上のモデル評価を引き起こすかもしれないので、まず最初に、追加平列関係は測定されます。
評価をモデル化するために比べて、イベント取り扱いが安価であるなら、これはシミュレーションに利用可能な平列関係のより正確な測定です。
2 番目に、いろいろな時間にただイベントが起こるので、それらの間に原因がある。
したがって、以前のイベントによって引き起こされたモデル評価が後のイベントに影響を与えないなら、異なった時間ステップのイベントは平列に処理されるかもしれません。
連続したシミュレーションから原因のデータを抜粋することによって、Briner 他によって、700 から 1 万 5000 のトランジスタサイズにねらいを定めて、3 個のサーキットで並列活動を見積もったところ、値が 4.7 から 19.5 の間に及んでいるのがわかりました。
彼らが同じサーキットのためにこれをサーキット平列関係と比較して、それを見つけた、わたる、メートル法の、しかし、サーキット平行関係を使用しているのが見つけられたより 4 から 10 倍潜在的の並列活動。
したがって、サーキット活動のいくつかの研究が過去数年間にわたって闘争結論と共にあります。
これらの違いのいくつかの理由があります。
まず最初に、異なった研究者は、サーキット活動を評価するのに異なった測定基準を使用しました。
異なったモデル抽象化と異なったタイミング解決を持っていて、2 番目に、異なった連続したシミュレータが使用されました。
最終的に、異なったサーキットは直接個々のベンチマークサーキットの構造に当然の違いをもたらして、様々な測定値に使用されました。

6-2 Timing Granularity タイミング粒状

論理シミュレーションは、それぞれ異なったタイミング粒状が非常に細かいタイミング(0.1 ナノセカンドなどの)から大きななタイミング(そのような広告ゼロ遅延)まで及んでいて、モデル表現の広いスペクトルをカバーしています。
タイミング粒状はシミュレータ性能にかなり影響を与えることができます。
シミュレータは細かいタイミングを使用し、シミュレータにおける正確な時間を構成しようとします。。
しばしば、これらのシミュレータは 0.1ns 以下の範囲で時間の解決を使用します。
極粒状のシミュレータの追加設定は与えられた素子には同じ遅延がいつもあるかどうかということです。
例えば、ゲートの出力値の如何にかかわらず、いくつかのゲートレベルシミュレータが与えられたゲートタイプにただ一つの遅延を使用するかもしれません。
他のものには、信号が出力キャパシタンスによって、上昇するか、または下降しているかにかかわらず異なった遅延があります。
極粒状のタイミングがあるトランジスタレベルシミュレータでは、ただ一つの信号のための多くの異なった見込み遅延をもたらして遅延計算はさらに複雑である場合があります。
私たちは、極粒状のタイミングがサーキットの小さい時間の解決と多くの見込み遅延の両方を含んでいると考えています。
見込み遅延値の数が減少するか、または時間の解決が増加するのことは、私たちはタイミング解決が粗雑であると言います。
粗雑なタイミング粒状の極端にはユニット遅延とゼロ遅延･タイミングがあります。
両方が論理シミュレーションで全く一般的です。

ユニット遅延･タイミングは、あらゆる素子には 1 ユニットの伝播遅延があると仮定します。
順序回路に使用されるゼロ遅延はさらに粗雑です。
ここに、機能性だけがタイミングを測定する試みなしで保存されます。

6-3 Relating Circuit Activity and Timing Granularity

実証的研究と形式モデルの両方を通してサーキット活動へのタイミング粒状の効果を調査してあります。
最初の実証的研究はベイリーの[1992a]のサーキット平列関係結果の拡大でした。
ユニット遅延シミュレータ(SwitchSim)は、以前に RNL を使用することで測定されたサーキットのサーキット平列関係を測定するのに使用されました。
フランクによって書かれた SwitchSim は Fast-1 シミュレーションエンジンのために開発されたアルゴリズムに基づいています。 ユニット遅延シミュレータによって測定されたサーキット平列関係は RNL によって測定されたそれよりいつも大きかったです。
平列関係値は、200 から 61600 トランジスタでサイズを定めながら、サーキット上に 35 から 593 まで及びました。
これらの値は RNL 測定値より 3.6 から 25.8 の間の素子に及んで大きかったです。
この研究は 2 つの異なったタイミング粒状のサーキット平列関係への効果を比較しましたが、それはタイミングと平列関係との間に良い特性を提供しませんでした。
しかしながら、ベイリー[1992b;1993]は、サーキット平列関係における時間の解決の効果を比較する二つの形式的モデルを展開しています。
両方のモデルは同じ初期の抽象化、与えられたサーキットの実行を表すグラフで始まります。
グラフでは、ノードはシミュレーションでイベントに対応し、縁は因果関係を表します。
1 回未満の変化が時間内にどんな瞬間にも起こると思われ、無限の解決時計はタイミングに使用されます。
また、ちょうど 1 回のイベントがその後のイベントを引き起こすと思われます。
これらの 2 つの仮定が、グラフがツリー状であることを確実にします。
ツリーの縁は出来事とその親の間の遅延でラベルされます。
無限の解決時計のために、あらゆるイベントがそれ自身の時間ステップにあります、そして、サーキット平列関係は全くありません。
平列関係と時間の解決との関係を調査するために、各モデルにはシミュレーションのタイミング粒状を増加させるためのメカニズムがあります。
ベイリー[1992b]によると、時間のベースモデルでは、イベントはそれらのシミュレーション時間により、因果関係規制を保存して大きい解決の時間ステップに置かれます。
これはシミュレータがイベントを時間ステップに置く方法と異なっていますが、結果として起こる分析はより単純です。
例えば、第一に時間 0、二番目に時間 1 で起こる最初のものと、三番目に時間 4 で発生する 3 つの関連のあるイベントを考えてください。
シミュレーション時計に 2 の時間ベースがあるなら、これらのイベントは、それぞれ時間ベースのモデルを使用して、時間ステップ 0、2、および 4 に置かれます。
このモデルを使用して、サーキット平列関係が時間の解決の非減少機能であることを示すことができます。
ユニットディレイ･タイミングを使用することで見つけられた平列関係はサーキット平行関係で上限をすべての時間の解決に提供します。
ベイリー[1993]第 2 代モデル、遅延ベースのモデルでは、イベントの間の遅延に応じて、イベントは時間ステップに置かれます。
これが、より密接に実際のイベント駆動のシミュレータのイベントのプレースメントに対応しているので、それは時間のベースのモデルより現実的です。
上の例では、最後のイベントは、それとその親イベントの間の遅延が 3 であるので、時間ステップ 4 よりむしろ時間ステップ 6 に置かれます。

残念ながら、このモデルから得られた結果は時間のベースのモデルのそれらより複雑で、サーキット平列関係はもう時間のベースの非減少機能ではありません。
時間ベースを増加させると平列関係が実際に減少する例があります。
しかしながら、解決が増加するのに従って、サーキット平列関係は、増加するか、または一定のままで残っている傾向にあります。
より正確に、ベイリーは同じユニット遅延平列関係があるすべてのサーキット族を考えます。
ユニット遅延平列関係はすべての時間の解決の上でサーキット平列関係の上限をこれらのサーキットに供給します。
これらのサーキットへのサーキット平行関係における下限は時間解決の非減少機能です。(時間の解決が十分大きいときに、それはユニット遅延平行関係と等しいです)。
ベイリーは事実上、RNL の時間の解決を変えて、サーキット平列関係への結果として起こる効果を測定することによって、遅延ベースモデル予測を確認します。
図5は、ベイリー[1992a]スーレと Blank Briner[1990;1987]ウォン他[1986]の多くの平列関係結果の合成グラフを示します。
含まれているのは、さまざまなタイミング粒状(極粒状、固定遅延、ユニット遅延している)と、スイッチ的から機能的に及ぶさまざまな抽象化レベルからの測定値です。
最も大きいサーキットから最も粗いタイミング粒状(ユニット遅延)を使用することで最も大きい平列関係値を得ます。
より上手により小さいサーキットとして結果を見るために、図の左下の四分円の拡大は図 6 で示されます。
全体的に見て、より高い平列関係値はベイリーのモデルによって予測されるように、より粗いタイミング粒状で取られた測定値から生じます。
抽象化レベルが測定値の重要な違いを作るかどうかは明確ではありませんが、サーキット(入力ベクトルに伴う)のタイプは、重要な違いを作るように見えます。
したがって、私たちにはより粗い粒状タイミングが、より高いレベルのサーキット平列関係を劇的にもたらすことができるという証拠を持ち、タイミング解決とサーキット活動との明確な関係があります。
サーキット活動の、より高いレベルが平列シミュレータでより良い性能を含意するなら、より粗い粒状のタイミングにおける平列シミュレータは更に有望に見えます。
結果が、すべてのタイミング粒状、例えばゼロディレイ・タイミングをカバーしないのに注意してください。
さらに、、同じシミュレータを用いて並列測定をし、サーキット活動がサーキット構造の他の見地に依存するのか示してください。
残念ながら、これらの研究はこれらの関係の正確な本質を見抜くことはできませんでした。

7 TARGET ARCHITECTURE 目標アーキテクチャ

平列アーキテクチャは、古典的に MIMD に仕切られた、(複数の指示、複数のデータ)各プロセッサが独自にコードを実行するマシンと、SIMD の(ただ一つの指示と、複数のデータ)のマシンで、すべてのプロセッサが独立しているデータを同じ命令で実行する、があります。
一般的なグローバルアドレススペースがデータにプロセッサの間の共有と同期を実装するか、または分配されたメモリ、コミュニケーションであるところに明白なメッセージを実装するのに使用されるところで、共有メモリとしてさらに MIMD マシンを分類することができます。
SIMD アーキテクチャでは、プロセッサは同時に型にはまったやり方で指示を実行します。
プロセッサが、望まれているステップの間、計算するのを避けるようにプログラムされるかもしれません。
すべてのプロセッサが同じ指示を実行しなければならないので、1 つのタイプのゲートだけが一度に、モデル化されます。
テーブルは、この制限を緩和するの助けるとして、しばしば実行されます。
多くのモデルタイプがあると(階級制度記述におけるケースがそうであるように)、シミュレーション性能は大いに減少するでしょう。
プロセッサはしばしばすばやく隣接しているプロセッサと互いにコミュニケートするグリッドによって接続されます。
非隣接しているプロセッサが交信しなければならないなら、他のプロセッサかグローバルなルータを通してメッセージを送信しなければなりません。
これは隣接する最も近いコミュニケーションより高価です。
ほとんどの論理シミュレーションは複雑なパーティションやマッピング、隣接する近いコミュニケーションに制限しません。
共有メモリ MIMD アーキテクチャは、プロセッサの間で交信するのに一般的なグローバルアドレススペースを利用します。
通常、小規模の平列関係はバスアーキテクチャで実装されます。(プロセッサはそれでバスに見つけた、ただ一つの物理メモリへのアクセスを競合します)。
これらのマシンは、プロセッサかメモリアドレスの如何にかかわらず一定のメモリ・アクセス時間を示します。
プロセッサの数が大きいときに、汎用インタコネクトネットワークは、各プロセッサに関連している記憶モジュールの間で交信するのに使用されます。
これらのマシンでは、メモリ・アクセス時間は、不均等であり、参照をつけられたアドレスがローカルかそれともリモートであるかに依存します。
プロセッサのコミュニケーションは、通常マイクロセカンドのオーダー時に比較的速いです。
しかしながら、インタコネクトネットワークに依存することは問題であるかもしれません。
一般的なメモリとローカルキャッシュがあるバスベースのアーキテクチャでは、バス、誤った共有、およびプロトコルオーバーヘッドのための依存は非常に高価である場合があります。
不均等なメモリアクセスを持っているマシンの上では、パーティションの目標の 1 つは、より遅い遠隔記憶として過度のコミュニケーションを避けることです。
分配されたメモリ MIMD アーキテクチャでは、各プロセッサに関連している記憶はそのプロセッサにローカルで、プロセッサのコミュニケーションは明白なメッセージで扱われます。
これらのマシンは、メッシュ、トーラス、または「超-立方体」などのスケーラブルなトポロジーを使用することで通常組み立てられます。
メッセージ潜在は機能評価時間に比例して長い場合があります。
1 台のプロセッサから別のプロセッサまで送信される信号がたまたま回路の同期信号の 1 つであるなら、シミュレーション性能は激しく落ち込むかもしれません。
ポピュラーになった平列な実行プラットホームはワークステーションのネットワークです。
これらのプラットホームには、一般に、スタイルにおいて分配されたメモリ MIMD マシンと同様、いくつかのユニークな特徴があります。
汎用ネットワークを通してメッセージ送信があるという事実によって彼らのコミュニケーション能力は

強く影響を及ぼされます。
これはかなり長いメッセージ潜在を含意します。
また、硬く結びついたコンピュータがリソースとして専念している間、複数のユーザがしばしば、プログラムを実行しています。
[1984 は空白にしてください; Goering1988。]汎用機械に加えて、平行な論理シミュレーションを与えるために建てられて、構造が提案する特別な目的の数がありました。
汎用機械と異なって、これらのエンジンは実行することができるシミュレーションのタイプを通常制限します。
通常、ただ一つの同期メカニズムを採用し、限られたモデルレベルだけが利用可能です。
多くの生産会社が論理シミュレーションエンジンを組立てました。
例えば、ヨークタウン Simulation Engine、Denneau 他 1983; フィスター1986、EVE、Beece 他 1988 は IBM で設計されました; 火星アクセラレータは AT&T で設計されました。Agrawal とダリー1990; NEC は HAL に高崎他 1986 を築き上げました; 富士通は SP 斉藤 1988 を開発しました; そして、Zycad 会社はマシンの全体の線を製造しています。
さらに、数台の論理シミュレーションエンジンが、提案されて、大学において試作されています; ミュンヘン Simulation Engine ハーン 1989 と Mahmood の変更されたデータフロー構造他 1992 はこれらの 2 です。
また、私たちがそうするつもりであるこの調査では、ここで議論した問題の多くがこれらのエンジンの有効性に関係します。

8 まとめ

平列シミュレーションを実装するには、VLSI システムの平列シミュレーションに影響を与える 5 つの要素

(1) 同期アルゴリズム
(2) サーキット構造
(3) タイミング粒状
(4) 目標構造
(5) パーティションとマッピング

を包括することができるのであれば可能である。

サーキット構造が並列シミュレーションの性能に影響を与えることは、サーキット活動に関する研究を通して実際に観測された。
しかしながら、サーキット構造とシミュレーション性能との正確な関係は解明されておらず、どのタイプのサーキットが並列シミュレータとして使用することで、有益に稼動するかを正確に予測することができない。

また、ゼロ遅延のような粗雑なタイミングは並列回路を増加させることから、タイミング粒状と同期アルゴリズムも性能に影響することがわかる。
したがって、同期シミュレーションにおいて、より粗雑なタイミング粒状は、イベント評価回数が比較的一定であると仮定した時、最も適している。
より適したタイミングを使用する平列シミュレーションが高性能をもたらすかは不明瞭である。
非同期にアルゴリズムにおいて、粗雑な粒状を使用することで、これらのシミュレーションが効果をあげる可能性はあるが、これらのアルゴリズムを実行のに必要であるオーバーヘッドは高価である。

アーキテクチャはすべての平列プログラムの性能に影響するが、非常に小さいデータは目標アーキテクチャとシミュレーション性能との関係を理解するのに利用可能である。

パーティションとマッピングは非常に重要である。
これまで、静的なパーティションが最も注目されてきた。
最適なオートパーティションアルゴリズムは、負荷バランス、コミュニケーション、同期コスト、および目標構造との相対的な重要性に依存する。
コミュニケーションにおいて、計算比がわずかであるときに、無作為のパーティションは良い動作をする。計算比が大きい場合、コミュニケーションと同期コストを減らすため、順当なテクニックが必要となる。

論理シミュレーションの領域の進歩は目覚しいが、まだ不明確な部分も多く、改良の余地がある。

9 参考文献

MARY l. BAILEY , JACK V. BRINER. JR. AND ROGER D. CHAMBERLAIN
Parallel Logic Simulation of VLSI systems