計算機アーキテクチャ 137−18
ハイパフォーマンス 80−18
コンピューティング
（2000．3．3）

MPIとマルチスレッドによる静的タイミング解析の並列処理

土屋 竜太　大津 金光
吉永 努　馬場 敬信

宇都宮大学 工学部 情報工学科

近年集積回路の性能向上はめざましく、回路規模もそれにつれて増大している。また、回路規模の増大にともなってCAD(Computer Aided Design)ツールの処理時間も増大し、高速化が望まれている。本研究では、特に処理時間がかかる静的タイミング解析を対象として、その並列化について検討し、近年安価に入手可能である共有メモリ型マルチプロセッサのWS（ワークステーション）クラスタ上で静的タイミング解析プログラムを実装する。静的タイミング解析とは、設計した回路の遅延時間を計算し、回路中のFF（フリップフロップ）間が、正常な動作をするかを解析するLSI-CADアプリケーションの一種である。また、マルチスレッドライブラリとMPI(Message Passing Inteface)ライブラリを併用する事により、各ライブラリの効率的な使用方法を提案する。並列化によって、2つのプロセッサを持つマルチプロセッサマシン2台で約2.6倍の台数効果を得た。

Parallel processing of static timing analysis using MPI and multiple threads

RYUTA TSUCHIYA , KANEMITSU OOTSU ,
TSUTOMU YOSHINAGA and TAKANOBU BABA

Department of Information Science, Faculty of Engineering,
Utsunomiya University

Recently, the performance and the circuit scale of LSI have been highly increased. The LSI-CAD(Computer Aided Design) applications are expected to be high performance,because processing time of this application has been increased. We studied about parallel processing for the static timing analysis which specially needs processing time. And we have built a static timing analysis program on Work Station(WS) Clusters with shared memory. Static timing analysis program is a kind of LSI-CAD applications,that analyse behavior of FF(flip-flop) by calculating logic circuit delay. Also,we propose a way to effectively use both multi threading library and MPI(Message Passing Interface) library. Based on experimental results,we show that about 2.6 times speed-up is achieved on 2 hosts with 2 processers.

1 はじめに

集積回路技術の発展はめざましく、数百万～数千万ゲートの回路も登場している。そのため、論理シミュレーションや配置配線などのLSI設計CAD(Computer Aided Design)アプリケーションでは、このような大規模な問題を扱う必要が生じている。

LSI設計においては、論理機能とタイミング（性能や信号遅延時間）の検証が必要となる。本研究は、後者のタイミング解析を高速化することを目的とする。静的タイミング解析（STA）の高速化に対しては、回路の遅延を近似によって求める手法が提案されている[1][2]。しかしながら、近似の精度が十分でないと、回路によっては誤差が問題となる場合も生じる。我々は、必要となる検証精度を確保しながら、その計算量とメモリ消費量に対応する手法として、ワークステーション（WS）クラスタ環境による分散並列処理を導入する。また、近年では共有メモリ型マルチプロセッサマシンの普及も進んでいることから、マルチプロセッサマシンを含むWSクラスタ上でのSTAの高速化について検討する。

2 タイミング解析処理

2.1 タイミング条件

タイミング解析は、信号が論理回路を通過する際に生じる遅延（『遅延』は素子固有の値を用いて計

算されたもの)を計算し、フリップフロップ(以後FFと表記する)への書き込みが正常に行われるかどうかを検証することである。具体的には、図1に示すように組合せ回路を通してつながったFF1とFF2のクロックをそれぞれCLK1、CLK2としたとき、CLK1とCLK2が「同期関係にある」ときのセットアップ、ホールド条件のチェックを行なうことである。

CLK1とCLK2が「同期関係にある」とは、

(1) CLK1とCLK2が同一のクロックである時
(2) CLK1とCLK2のどちらかがマスタークロックで、一方がそれに属するスレーブクロックの時
(3) CLK1とCLK2が同一のマスタークロックに属するスレーブクロックの時

のいずれかの場合である。本研究では、研究の初期段階として(1)の完全同期回路のみを対象とする。

セットアップ条件は、FF1からFF2まで到達する信号の中で最も遅いものが、<4>のクロック信号の立上りに対してFF2のセットアップ時間より前に確定すれば満たされる。また、ホールド条件は、FF1からFF2まで到達する信号の中で最も早いものが、<4>のクロック信号の立上りに対してFF2のホールド時間より後に確定すれば満たされる。

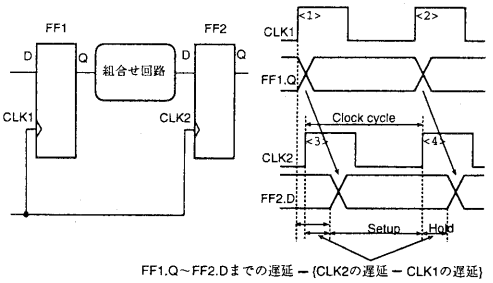

図 1: セットアップ、ホールド条件

以下にタイミング条件式を示す。

- Setup条件式

$$FF2\text{の}Setup\text{時間} <$$
$$Clock\_cycle - \{FF1.Q \sim FF2.D\text{の遅延}$$
$$-(CLK2\text{の遅延} - CLK1\text{の遅延})\}$$
$$\Downarrow$$
$$FF1.Q \sim FF2.D\text{の遅延} <$$
$$Clock\_cycle + CLK2\text{の遅延} - CLK1\text{の遅延}$$
$$-FF2\text{の}Setup\text{時間}$$

- Hold条件式

$$FF2\text{の}Hold\text{時間} <$$
$$FF1.Q \sim FF2.D\text{の遅延}-$$
$$(CLK2\text{の遅延} - CLK1\text{の遅延})$$
$$\Downarrow$$
$$FF1.Q \sim FF2.D\text{の遅延} > CLK2\text{の遅延}-$$
$$CLK1\text{の遅延} + FF2\text{の}Hold\text{時間}$$

2.2 遅延計算

各ゲート間の遅延tは、$t = t_0 + KCL \times C$ の式で計算する。t_0、KCLは素子の種類に固有の値であり、t_0は素子の基本遅延、KCLは負荷による遅延の増加係数を表している。$KCL \times C$は配線遅延と負荷を合わせた値である。また、Cは接続先の入力負荷の合計と配線負荷の和である。入力負荷もまた、素子固有の値であり、入力端子ごとに設定されている。配線負荷の値は、接続先のゲートの個数に応じた値が予め設定されている。

例えば、図2の遅延時間は

$$t = (and\text{の}t_0\text{の値}) + (and\text{の}KCL\text{の値})$$
$$\times\{(and.A1\text{の入力負荷})+$$
$$(and.A2\text{の入力負荷}) + (2\text{ゲート接続時の配線負荷})\}$$

となる。ここで、and_A、and_B、and_Cは素子固有名で、素子の種類はandであるとする。

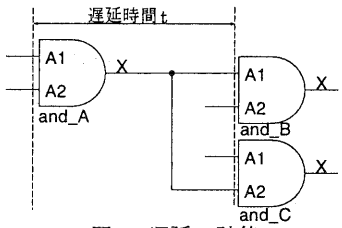

図 2: 遅延の計算

3 実装

3.1 全体の処理

全体の処理の流れを図3に示す。

前処理

内部データ構造への変換は、テキストデータであるVerilog-HDLネットリストを数値化し、等電位表現へ変換する。数値化は、素子固有名を番号付けすることにより、タイミング解析の実行時間に大きく影響を及ぼす回路検索にかかる時間を短縮することである。また、等電位表現への変換は、マルチプロセスによる並列化に対する回路分割のために必要となる。また、本タイミング解析プログラムでは、MPIを用いた並列実行において自プロセスが保持する部分回路に接続先が無い場合にメッセージ転送を行なう。そのため、複数の出力がある素子(加算器など)の一部が使われていない場合、全ての部分回路で接続先が見つからず、無限にメッセージ転送を行なってしまう。この問題を解決するために、ダミーの出力端子を挿入する処理を内部データへの変換中に行う。

タイミング解析の開始点の抽出は、内部データ構造への変換と同時にFFの出力端子をリストに追加することで行う。

クロックパスの遅延計算はタイミング解析ルーチンをもちいてクロックパスを検索し、その遅延を計算する。前処理では、内部(回路)データ、タイミング解析の開始点、クロックパスの遅延を共有メモリに書き込み、本体処理に渡す。ここまでの処理は逐次的に実行する。

本体処理

タイミング解析は開始点のリストからタイミング解析の開始点を取り出し、回路のトレース、遅延計算、タイミングチェックを行なう。開始点のリストが空になった時点で結果を収集し処理を終了する。本研究はこれらの処理の内、最も処理時間を要するタイミング解析を並列化する。

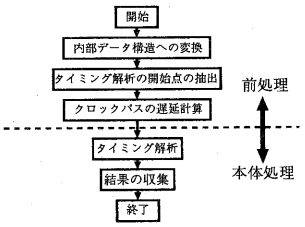

図 3: 全体の処理の流れ

3.2 並列化の方針

本研究では図4に示すように、ホスト毎に回路を分割する。そして、同一ホスト内では共有メモリ上に回路データを保持し、マルチスレッド [4] を用いて並列化を行なう。また、異なるホスト間でのみ MPI[3] を用いた並列化を行う。

このように実装する理由は、共有メモリを構築するにあたり、マルチプロセスよりもマルチスレッドで構築した方が実装が容易であり、コンテキストスイッチが軽量であることによる。また、プロセス間通信には移植性の観点から MPI を利用する。

更に、タイミング解析の開始点を共有変数とし、各スレッドがそれを取り出して処理することで、スレッド間で負荷を動的に分散させることが容易となる。また、回路を分割する理由は、増大する回路規模に対応するためと、回路分割により、回路検索の範囲が狭まり、検索時間が短縮するためである。

3.3 マルチスレッドによる並列化

マルチスレッドによる並列化は図4の処理を図5のようにスレッドに分割することにより行なう。手順を次に説明する。

(1) 前処理で共有メモリ上に回路データ、開始点のリストおよびクロックパスの遅延を書き込む。

(2) タイミング解析スレッドを利用可能なプロセッサ台数分起動する。

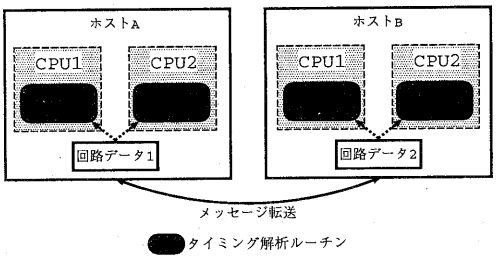

図 4: 並列化の方針

(3) 各タイミング解析スレッドはタイミング解析の開始点(別のプロセスから送られてきた接続点を含む) を開始点のリストから取り出す。

(4) 各スレッドは回路データおよびクロックパスの遅延を参照して回路のトレース、遅延計算、タイミング解析を行なう。

(5) 開始点のリストが空になり次第、結果を収集して処理を終了する。

このように実装することで、各スレッドは1つの開始点から始まる処理を完了次第、共有しているリストから開始点を取り出すことになるため、各スレッド間で自動的に負荷分散が行われる。

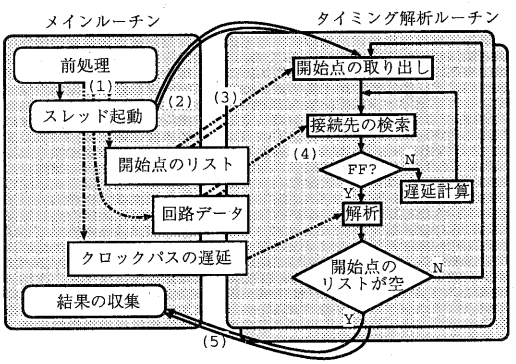

図 5: マルチスレッドによる並列化

マルチスレッドにおいて共有メモリの変更を行なう場合、共有データの破壊を防ぐために相互排除する必要がある。マルチスレッドライブラリでは、アプリケーション内で有効な mutex を1つ持ち、それを取得・解放することで相互排除を行なう。本プログラムにおいて、開始点のリストは各スレッドから変更が生じるため、相互排除が必要となる。ここで、mutex の取得と解放は1回につき約 2〜3μ秒であり、たとえ100万回の相互排除が発生しても、たかだか2〜3秒しかかからない。本プログラムでは、全実行時間から見ると誤差の範囲でしかなく(リストの個数が約8万の場合、実行時間は約1万秒)、台数効果

を阻害しているものは純粋に相互排除が起こっている時間である。そのため、相互排除によりアクセス出来ない時間を減らすことが台数効果の向上につながる。

そこで、開始点のリスト用mutexをタイミング解析スレッド数分用意し、各スレッドが所持しているリストが空になった場合に同一プロセス内の別スレッドが所持しているリストを参照できるようにする。このようにすれば、開始点のリストが空でない場合には相互排除を除くことができ、リストが空になれば動的負荷分散が図れる。

3.4 マルチプロセスによる並列化

ここでは、図4に示した並列化の方針に従って、マルチスレッドによる並列化を複数のホストに拡張したMPIを用いた並列化について述べる。

まず最初に、各ホスト内の開始点のリストの処理が終わるまでメッセージを蓄積し、開始点のリストが空になった時点でメッセージ送受信する実装(図6(A))について考える。この実装の場合、あるホストで開始点のリストが空になりメッセージ受信命令を発行しても、送信側のホストがタイミング解析を実行している。そのため、受信側ではメッセージ受信待ちとなり、タイミング解析処理がアイドル状態となる。

そこで、このアイドル時間が発生することを解消するために、タイミング解析スレッドがノンブロッキング送信で送信命令を発行する実装(図6(B))について考える。この実装の場合、あるホストで開始点のリストが空になりメッセージ受信命令を発行すると、すぐにメッセージの受信処理を開始する。しかし、受信処理を行う間は開始点のリストが空であるため、タイミング解析処理はアイドル状態となる。更に、送信側のホストから出された複数の送信命令に対して複数の受信命令を発行して受信処理を行なう必要があり、1回の受信処理の時間が増える。すなわちタイミング解析処理のアイドル状態が長くなる。

一方、メッセージ送受信を別スレッドで行なう実装の場合(図6(C))、メッセージ送受信処理はタイミング解析処理を行う間に実行されるため、開始点のリストが空になる事態が発生しにくい。また、開始点のリストが空でない限り、いずれかのプロセッサはタイミング解析処理を行うため、タイミング解析処理のアイドル時間が減少する。更に、送信用バッファに蓄積されたメッセージを転送する構造とすることで、次節で述べるメッセージを一括して転送し総メッセージ転送時間を短縮する実装が容易となる。このように実装することで、受信待ち状態の解消と、受信処理時間の短縮が図れる。

そこで本研究では、各ホストにホスト間の通信を行なうメッセージ転送スレッドと、メッセージ転送用バッファを用意することで並列化を行う。タイミング解析スレッドはそのメッセージ転送スレッドを介してプロセス間通信を行なう。

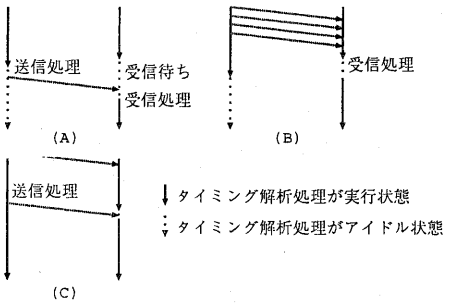

図 6: タイミング解析処理のダイアグラム

3.4.1 一括転送による総転送時間の削減

100BASE-Tのスイッチングハブで接続されたUltraSPARCII(300MHz)のWS 2台で、MPIライブラリを用いてメッセージ転送を行なった場合、2048bytesのメッセージを送るのには約0.7msの時間がかかる。また、4096bytesのメッセージを送るのには約0.9msの時間がかかる。このことから、2048bytesのメッセージを2回送るより、1回の4096bytesのメッセージで送る方がはるかに総転送時間が減少する。そこで、複数のメッセージを一括して転送することで総転送時間を削減する。

3.4.2 メッセージ転送ルーチンの実装

本プログラムでは、各スレッドが共有メモリにある開始点のリストから開始点を取り出し並列に解析を行う。そのため、異なるプロセス間の通信は、共有メモリ上にある送信用バッファから開始点のリストへとスレッドの区別をせずに行なえる。そこで、送受信スレッドは図7のように実装した。図中の送信条件は、送信用バッファに規定数以上のメッセージが蓄積されるか(**条件1**)、規定数回(α)→(β)のループを経過した場合(**条件2**)に満たされる。

条件1はメッセージがある程度蓄積してから転送することで、一括転送を効率的に行なうためである。また、規定数以上のメッセージが蓄積された場合には、即座に転送し送信用バッファを溢れさせない。**条件2**は、受信側ホストのアイドル時間を減らすためにメッセージを転送する。

ここから転送処理の手順について説明する。まず、メッセージ転送スレッドが起動すると、ノンブロッキング受信を発行し、他ホストからのメッセージ転送に備える(1)。次に、複数のタイミング解析スレッ

ドが蓄積していく送信用バッファをチェックし(2)、送信条件を満たした場合、送信用バッファに蓄積された分全てを一括転送する(3)。メッセージを受信した場合(4)、送られて来たメッセージを開始点のリストに加え(5)、ノンブロッキング受信を再発行する(γ)。

ノンブロッキング受信は、受信手続きを行ったあと別の処理を行うことができるため、メッセージを受信完了していない場合にも送信用バッファのチェックやメッセージ転送の処理が可能となる。

なお、送受信スレッドは所々にCPUの実行権を譲渡するsched_yield()関数を挿入して、送受信スレッドのCPU利用率を下げている。

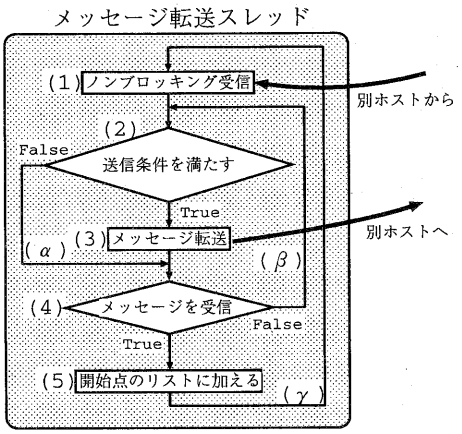

図 7: メッセージ転送の詳細

3.4.3 部分回路への分割

回路分割は、素子番号が近い(ネットリストにおいて素子情報が近い)ものを同一の部分回路に保持する。これにより、回路の接続先が見つからない事態(メッセージ転送が発生する)が減少することが予想される。そこで、本研究では辞書型データの素子番号によって振り分ける。例えば、素子数60000の回路を2つのホストで実行するのであれば、素子番号0から30000をホスト1に割り当て、素子番号30001から60000をホスト2に割り当てる。

更に、図8(a)の左側にあるように、部分回路の切れ目に複数のパスがまたがることにより、メッセージ転送数が増大することを防ぐための改良を施した。改良の方法として、単純に図8(a)の右側にあるように入力と出力の間で回路分割する方法があるが、今回は回路の接続情報を考慮して分割する。

具体的には、出力先が自プロセスにある端子の個数をI、他プロセスにある個数をOとすると、$O-I >$ 閾値となる場合に、出力先の等電位部分を他プロセスに送る(図8(a))。

このような改良を行うと、出力先の端子が自プロセスに多くある場合(図8(b))と、出力先が少ない場合(図8(c))に出力先の等電位部分を自プロセスに保持することでメッセージ転送数が抑えられる。

なお、本研究では閾値を5としている。

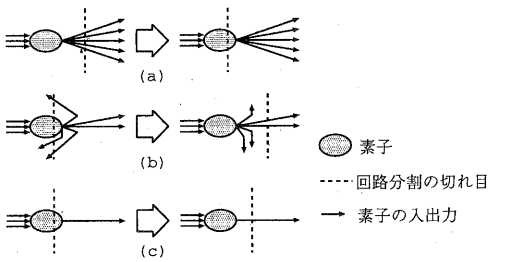

図 8: 部分回路への分割

4 評価結果

4.1 各実装の評価

図9に素子数58000の回路を用いた結果について示す。実験には、SunEnterprise(UltraSPARCII 400MHz×8、MM:3GB)を用いた。そして、2プロセッサ、4プロセッサのホストと2プロセッサのホストを2台用いたクラスタ環境を想定して実験を行った。図中Aは逐次プログラムの結果であり、C、Eは開始点のリストを共有させなかった場合の結果である。また、Gは接続情報を考慮した回路分割を行っていないものであり、Hは一括転送を行わず個々のメッセージに対して送信を行なったものである。また、2スレッド2ホストとは、タイミング解析スレッドが合計4つということであり、2プロセッサのホスト2台構成のクラスタ環境を想定したものである。なお、逐次プログラムは専用のものを使用した。

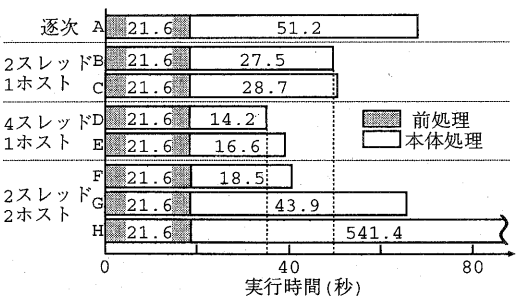

図 9: 評価結果

この結果から、マルチスレッド化により、本体処理部分の逐次から2スレッド(A→B)と2スレッドから4スレッド(B→D)で共に約1.9倍、逐次から4スレッド(A→D)でも約3.6倍となりリニアに近

い台数効果が得られ、マルチスレッドによる並列化は高速化に有効な手段であるとわかる。また、開始点のリストを共有させる動的負荷分散の効果は2スレッド(B→C)では目立った効果はないが、4スレッド(D→E)では大幅な効果があることがわかる。このことから、4プロセッサマシンが普及した場合、非常に有効な技術であるといえる。

また、MPIによる並列化では、逐次から2スレッド2ホスト(A→F)では、本体処理で2.77倍となり、この回路規模では回路検索範囲が狭まる効果よりもメッセージ転送のオーバーヘッドが大きくマルチスレッドの方(A→D)が速い。GとHではメッセージ転送のオーバーヘッドが膨大なものとなり、一括転送や接続情報による部分回路の作成が必須であることがわかる。

4.2 回路規模による台数効果の変化

ここでは、回路規模を変化させた場合の台数効果を比較する。表1に実験に用いた各回路の規模と逐次版の前処理、本体処理および全体の実行時間とその比率を示し、図10に各回路の逐次版(前処理+本体処理の実行時間)を1として、回路を変化させた場合の台数効果の推移を示す。

表 1: 各回路の規模と逐次版の実行時間[sec]

	素子数	FF数	前処理	本体処理	全実行時間
#1	7748	1248	2.9	2.0	4.9
比率			59%	41%	100%
#2	15460	2496	4.9	4.8	9.7
比率			51%	49%	100%
#3	58080	9386	21.6	51.2	72.8
比率			30%	70%	100%
#4	93107	15015	47.9	131.8	178.9
比率			27%	73%	100%
#5	465617	75075	1694.3	9904.9	11549.2
比率			15%	85%	100%

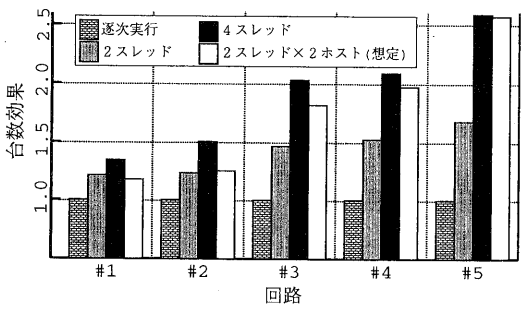

図 10: 回路規模の変化による台数効果の推移

この結果から、回路規模が増大するのに伴い、全実行時間に対して、逐次で行う前処理にかかる時間の割合は減少し、台数効果が向上している。

また、4スレッドと2スレッド2ホストは、使用するプロセッサの総数は同じであるが、2スレッド2ホストの方ではメッセージ転送のオーバーヘッドが存在する。しかし、回路規模が増大することにより、タイミング解析の処理量が増えるために回路検索範囲が狭まる効果が表れ、台数効果が向上している。

5 おわりに

本稿では、共有メモリマルチプロセッサマシンの特徴を活かし、マルチスレッドとMPIを用いてタイミング解析プログラムを作成し評価を行った。マルチスレッドによる並列化では2プロセッサで1.9倍、MPIを用いた並列化では、2プロセッサ×2ホストのクラスタ環境を想定した結果で3.6倍の台数効果を得た。(いずれも本体処理)

今後の課題として考えられることは、回路分割における閾値やメッセージ転送ルーチンのメッセージ転送条件(規定数以上のメッセージあるいは規定数回のループ)の最適な値を実験的に求めることである。

謝辞

本研究において、親切な指導や適切なアドバイスをくださり、関連資料の提供をしてくださった富士通キャドテック(株)の湯浅克男部長・柴田雄司部長・鈴木康弘課長・大川正一氏・町田光之氏に感謝する。また、日頃討論を頂く馬場研究室の方々に感謝する。本研究は、一部文部省科学研究費 基盤研究(B)課題番号10558039、奨励研究(A)課題番号11780190の援助による。

参考文献

[1] Y. Kukimoto,W. Gosti,A. Saldanha,R. K. Brayton : "Approximate Timing Analysis of Combinational Circuits under the XBD0 Model",IEEE ,0- 89791- 993- 9/ 97

[2] 平田、近藤、小野寺、田丸:"抵抗分を含む負荷を駆動するCMOS論理回路のゲート遅延計算手法",情報処理学会論文誌, Vol.40,No.4,pp1679-1686,1999.

[3] MPI日本語訳プロジェクト,"MPI:メッセージ通信インターフェース標準(日本語訳ドラフト)",(1996)

[4] Steve Kleiman,Devang Shah,Bart Smaalders著 岩本信一訳,"実践マルチスレッドプログラミング", 株式会社アスキー(1998)ISBN4-7561-1784-8

[5] 土屋竜太、大津金光、吉永努、馬場敬信 "WSクラスタを用いた並列論理回路タイミング解析の高速化", 情報処理学会研究報告 , Vol.98,No.72 , pp.55-60 (1998)