ハイパフォーマンス
コンピューティング 72−10
（1998. 8. 6）

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

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

近年マイクロプロセッサの性能の向上はめざましく、回路規模も増大している。回路規模の増大にともなってCAD(Computer Aided Design)ツールの高速化が望まれている。並列処理による高速化はこのような要求に答える有効な手段の一つである。本研究では、タイミング解析を行なうプログラムを並列オブジェクト指向言語A-NETLにより作成し、標準メッセージパッシングライブラリPVMを使用し、ワークステーションクラスタ上で実行しその評価と並列化による効果を調べた。プログラムを高速化するために、メッセージ転送量の削減とノード稼働率の向上を目的として6つの改良を行なった。

素子数145の論理回路を用い、ワークステーション3台で実行した場合、改良を行なった結果、17.612秒から2.492秒と約7.07倍の速度向上が得られた。また、素子数837の論理回路を用い、ワークステーション2台で実行した場合、実行速度は、4.778秒となった。

Parallel timing analysis of logic circuits using a workstation cluster

Ryuta TSUCHIYA , Kanemitsu OOTSU ,
Tsutomu YOSHINAGA and Takanobu BABA

Department of Information Science, Faculty of Engineering,
Utsunomiya University

As the size of large-scale integrated (LSI) circuits grows, the computation time for timing analysis is becomming a critical issue in verifying circuit design. Parallel processing is a promising solution for this pr oblem. We have designed and implemented a parallel timing analysis algorithm, using a parallel object-oriented language called A-NETL. A-NETL programs are compiled to C code which utilizes the PVM message passing library, and then executed on a workstation cluster.

Starting from an initial implementation, we applied 6 techniques for utilizing the parallelism among multiple workstations and for reducing the amount of messages exchanged among them. Experimental results show that the application of these techniques reduces the execution time from 17.612 (sec) to 2.492 (sec) (a speedup of 7.1) for a logic circuit with 145 elements on a cluster of 3 workstations, and that further implementation improvement attains an execution time of 4.778 (sec) for a circuit with 837 elements on a cluster of 2 workstations.

1 はじめに

近年集積回路技術の性能の向上はめざましく、それにつれて回路規模も増大している。現在、数百万ゲート規模のLSIも存在する。シミュレーション、配置配線など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でてきている。CAD(アプリケーション)において、データの大きさをnとするとき処理アルゴリズムの手数が$O(n)$で済むことは稀であり、一般に$O(n^2)$の割合で処理時間が増大していく[1]。

したがって、回路規模の増大にともなってCADツールの高速化が望まれている。この解決手段として種々のハードウェアアクセラレータが開発されているが、これらのアクセラレータは特定のアプリケーションのみを高速化するものであり、LSI開発全体のスループットの向上に果たす役割は十分とはいえない。また、単機能であるが故に導入コストが割高であり、広く使われるに至っていない。そのため、汎用並列計算機を使用した高速化やプログラマブルなアクセラレータによる高速化が試みられている。

本研究の目的は、タイミング解析に対する並列化の有効性を示すことである。そのために、タイミング解析を行なうプログラムを、我々の研究室で設計・試作した並列オブジェクト指向言語A-NETL[2]により作成した。

A-NETLは、いったんCプログラムに翻訳し、これを研究室の複数のワークステーション(以後WS)環境で実行する。メッセージ通信にはPVM[3]を用いる。

2 タイミング解析

2.1 タイミング条件

タイミング解析とは、論理回路によって信号が素子を通過する際に生じる素子遅延と配線遅延を計算し、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なわれるかどうかをチェックする事である。チェックの対象項目としては種々あるが、本研究では、基本的な項目として、論理回路で結ばれたFFのクロックを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。

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、(1) の単一クロック回路を対象とする。

図 1に 2 つのフリップフロップ FF1、FF2 と組合せ回路からなる回路図とそのタイミングチャートを示す。

図中< 1 >のクロック信号の立上りエッジで、FF1 の出力が更新される。この変化は，組み合わせ回路を伝播して行き、FF2 の入力まで到達する。この伝播のなかで、もっとも遅いものが FF2 へ到達してから < 4 >のクロック信号の立上りまでの時間が、セットアップ時間より長ければセットアップ時間の要件は満たされる。

一方、< 2 >のクロック信号の立上りエッジで、FF1 の出力はさらに更新されるので、この変化も組み合わせ回路内を伝播し、FF2 の入力へ到達する。この伝播のなかで、もっとも速いものが < 4 >のクロック信号からホールド時間以内に到達すると、ホールド時間の要件が満たされず FF2 への書き込みが正しく行われない。

図 1に示すような回路の場合、タイミング条件式は以下のようになる。

- Setup 条件式

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

- Hold 条件式

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

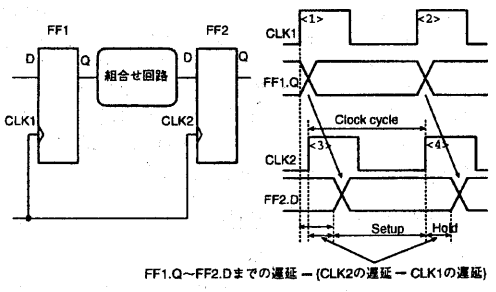


図 1: セットアップ、ホールド条件

2.2 遅延計算

遅延モデルには、$t = t_0 + KCL \times C$(t を各ゲート間の遅延) を用いている。

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、いずれも各素子に固有の値である。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。

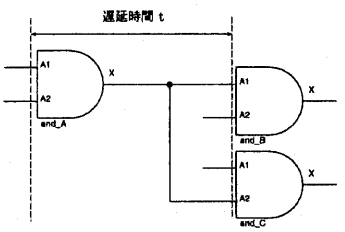


図 2: 遅延の計算

図 2のような回路の場合、点線間の遅延 t は次のようになる。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。

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

3 タイミング解析の並列化

3.1 オブジェクトの構成

プログラムは、1 つの main オブジェクトと複数の subcircuit オブジェクトから構成される。subcircuit は A-NETL のインデックストオブジェクトとして静的に生成される。インデックストオブジェクトは、同一のソースコードで記述したオブジェクトを同時に複数個の異なるオブジェクトとして定義するものであり、それらのオブジェクトはインデックス番号で区別される。

以下に各オブジェクトの機能を説明する。

- main
  全体の処理の流れの制御、タイミング解析の開始点やネットリストなどのデータの subcircuit への転送を行なう。
- subcircuit
  トレースおよびタイミングチェックを並列に行なう複数のオブジェクトである。

3.2 データ構造

3.2.1 入力データ

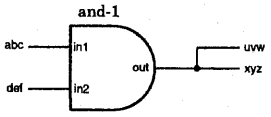
データの入力には、PARTHENON[4] のネットリスト記述言語 NLD (Net List Description) で記述されたネットリストファイルを使用する。

NLD では、等電位表現を採用している。これは、「接続されている端子をすべて並べて表す」形式である。図 3(a) に示した回路を NLD で表現したものが図 3(b) である。

図 3(b) 中、2 行目の (def-pin abc type input) は、abc と言う名前の外部入力端子を使うことを表している。6 行目の (def-comp and--2 and-1) は、 and--2 という部品種名の部品を構成要素として，and-1 という固有名で使うことを表している。また、9 行目の (def-net uvw xyz and-1.out) は、外部端子の uvw と xyz そして and-1 の out という端子がすべて接続されていることを表している。

3.2.2 内部データ構造

上で述べたネットリストはプログラム内部では、A-NETL の辞書型データ構造を用いて実現する。辞書は見出し語 (キー) とその値から構成され、見出し語の重複は



(a)

```
 1:  (def-module test
 2:      (def-pin abc type input)
 3:      (def-pin def type input)
 4:      (def-pin xyz type output)
 5:      (def-pin uvw type output)
 6:      (def-comp and--2 and-1)
 7:      (def-net abc and-1.in1)
 8:      (def-net def and-1.in2)
 9:      (def-net uvw xyz and-1.out)
10:  )
```

(b)

図 3: 回路例

ないが値は重複する場合がある。セルの生成、値の参照の操作が可能である。

この辞書型を用いて、辞書の見出し語には端子、その値には見出し語の端子の先に接続されている端子のリストを代入する。なお、プログラム内部のデータは、値参照の高速化のために全て数値化している。見出し語と値を交互に参照することにより、回路のトレースを行う。

3.3 アルゴリズム

全体の処理の流れを図 4に示す。

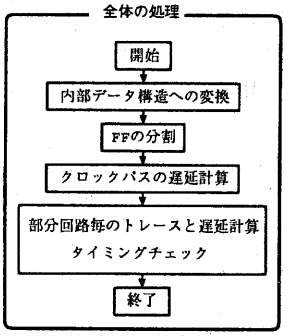


図 4: 全体の処理の流れ

FF の分割とは、回路中の FF の出力端子 (タイミング解析の開始点) を分割し、それを各インデックストオブジェクトに割り当てることである。

タイミング解析部のアルゴリズムを以下に述べる。

(1) バッファから FF の出力端子を取り出す。

(2) 取り出した FF の出力端子を開始点として回路のトレースを行なう。

(3) 回路が分岐していればスタックにそこまでの遅延時間と分岐点を記録し、更にトレースを続ける。

(4) トレース中、FF に達した時はタイミング条件を算出し、タイミングチェックを行なう。

(5) バッファが空になるまで (1)〜(4) を繰り返す。

これらの処理を各インデックストオブジェクトが独立して行うことにより、並列にトレースおよび遅延計算が行なう。

3.4 高速化の手法

我々は、3.1から 3.3を実現したプログラムを試作 [5] した後、これに以下に示す 7 つの改良を加えた。

3.4.1 トレース済パスの記憶 (改良 1)

本プログラムでは、FF の出力ピンから FF の入力ピンまたは外部端子までのタイミング解析を行なう。この場合、異なる FF から始まるタイミング解析で、同じ回路をトレースする場合がある。この場合、一度トレースし遅延計算を行なったパスを、再びトレースし計算することは無駄である。そこで、一度トレースし遅延計算を行なったパスをノード毎に記録しておき、そのパスが再び計算される時には、記録した値を使うことで高速化を図れる。この時記録する情報は、接続点とその間の遅延時間である。

この改良は、今回評価を行なった全てのプログラムの前提となっている。

3.4.2 辞書の分割 (改良 2)

本タイミング解析プログラムでは、ネットリストの検索に辞書型を用いている。タイミング解析における処理は主に、ネットリストの検索およびタイミングチェックを行なうが、タイミングチェックは、簡単な四則演算と比較のみなので、相対的に、検索にかかる時間に実行性能は大きく左右される。

A-NETL on PVM では辞書を逐次的に検索している。そこで、辞書を分割することにより、高速化を図る。

3.4.3 メッセージ転送量の削減 (改良 3)

従来のプログラム (高速化の手法を行なっていないもの) では、あらかじめネットリストを等分割して、それぞれの部分回路 (分割された論理回路) を各インデックストオブジェクトに割り当てることによって並列にタイミングチェックを行なっていた。そこで、トレースの過程で接続先が自らのインデックストオブジェクトに無い場合、他のインデックストオブジェクトへのデータの受渡しを行なう。

しかし、

(1) メッセージ転送の負荷が増加してしまう。

(2) ネットリストには、FF がまとまって出力されているため、結果が偏りやすい。

(3) タイミング解析の終了を自動化する為には、各インデックストオブジェクトの動作状況の確認の為のメッセージを何回も送らなければならない。

などの問題がある。

そこで、各インデックストオブジェクトが全てのネットリストを持つ事によって、各インデックストオブジェクト内でタイミング解析処理が閉じる事が可能となる。それにより、トレースの開始点である FF が、どのインデックストオブジェクトにでも割り当てる事が可能となる。そこで、FF を各インデックストオブジェクトに均等に割り当てる事で結果を分散させる事が可能となる。

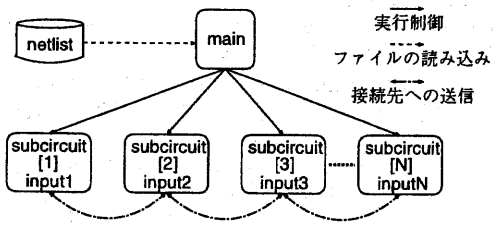


図 5: 従来のプログラム

更に、従来のプログラムでは、先にも述べた通り、回路のトレースにおいて部分回路内に接続先が見つからない場合、メッセージ転送を行なう。しかし、この改良により接続先が見つからない可能性が無くなるため、接続先の有無を判定する条件分岐が必要無くなる。しかし、問題として、メモリの使用量が増える、トレース済パスの記憶のヒット率が下がる、の2点が考えられる。

前者は、回路を分割していないため、分割数倍(これ以後、断りが無ければ「分割数」はインデックストオブジェクトの総数を表す)のメモリを使用する事となる。後者の理由は、本プログラムでは、改良1(トレース済パスの記憶)で同じパスをトレースする際に同じ計算の手間を省いているが、この改良で、回路を分割せずにFFを分割しているので、パスの記憶も各インデックストオブジェクトに分散してしまう。そのため、トレース済のパスがもう一度使われる確率が低くなり、パスの記憶のヒット率は、おおよそ(1/分割数)になると考えられる。

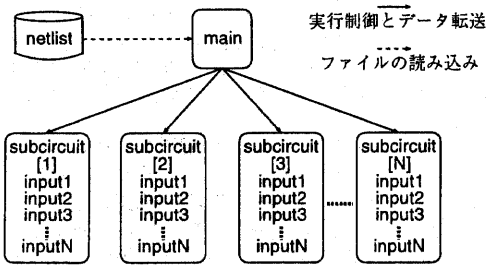


図 6: メッセージの転送量を減らしたプログラム

3.4.4 データ変換の並列化 (改良 4)

内部データへの変換は、ファイルからネットリストを読み込み、素子に連番を付け、それを辞書型変数に代入する。この処理は、インデックストオブジェクト間での相互参照を必要としない。つまり、データ変換実行中にメッセージの送受信が必要無く、各インデックストオブジェクト内で処理が閉じている。以上の理由により、並列化によるメッセージ転送の負荷を上回る実行性能の向上が見込める。

3.4.5 タイミング解析部の改良 (改良 5)

改良3(メッセージ転送の削減)によってトレースにおける各インデックストオブジェクト間のメッセージ転送が必要無くなった。そこで、それを省いたプログラムを作成した。

3.4.6 入力データの分散 (改良 6)

当初、1つのホストのハードディスクから、ネットワークを介してデータの読み込みを行なっていた。しかしこの方法では、ネットワークの負荷が増大し、更にデータの読み込みが、一つのホストに集中する為、データ変換の並列化における効果が半減してしまう。そこで、入力データを各ホストに分散させることにより、ネットワークの負荷の減少および並列性の向上を図る。

3.4.7 マシン性能に応じた負荷分散 (改良 7)

本研究では、異なる性能を持つWSを結合したWSクラスタ上で実行する事を前提としているので、性能の異なるマシンでの実行でもノード稼働率を上げる改良が必要である。そこで、まず最初に本プログラムで最も処理に時間のかかるトレース及びタイミングチェックにおけるノード稼働率の向上を考えた。

タイミングチェックの仕事量は、基本的に割り付けられたFFの数に依存する。そこで、マシンの速度比に応じてタイミング解析の開始点を分散させることにより、ノード稼働率の向上を図る。

具体的には、あらかじめ、各マシンの性能を元に速度比を設定する。式(1)に示すように、インデックストオブジェクトが載っているマシンの速度を全てのマシン速度の和で割った分を上限として割り振った。

なお、ここでは、CPUのクロックをもってマシンの速度としている。

$$\text{割り当てる } FF\text{の数} = \frac{\text{マシンの速度} \times FF\text{の総数}}{\text{使用するマシンの速度の和}} \quad (1)$$

4 実験結果

改良1(トレース済パスの記憶)と改良2(辞書の分割)は全実験結果の前提となっている。

また、評価に使用した論理回路は、『従来のプログラムとの比較』では、素子数145、FF数22である。それ以外の結果では、素子数837、FF数47の回路を用いている。

4.1 実行環境

実験に使用したWSの仕様を表1に示す。

表 1: ホストの仕様

ホスト名	CPU	clock(MHz)	memory(MB)
S60	SuperSPARC	60	96
S75	SuperSPARC	75	128
U167	UltraSPARC	167	128
DU200	UltraSPARC 2	200	320
DU300	UltraSPARC 2	300	512

4.2 メッセージ転送量の削減 (改良 3) の効果

4.2.1 従来のプログラムとの比較

『従来方式』は、3.4章の「高速化の手法」を行なっていないものである[5]。また、『改良方式』は、『従来方式』に対して、改良1(トレース済パスの記憶)、改良2(辞書の分割)と改良3(メッセージ転送の削減)を行なったものである。

FFの分割(メッセージ転送量削減)による効果

ここでは、FFを分散した事による効果を示す。

『従来方式』と『改良方式』で、タイミング解析の出力結果の総数の比較を表2に示す。

ネットリストには、FFがまとまって出力されているため、ネットリストを等分割している『従来方式』は、結果が偏ってしまう。

FFを分散した事によって、結果の総数が0～75から24～37となり、負荷分散を達成している。

表 2: 出力結果の総数 (個)

オブジェクト番号	1	2	3	4
従来方式	0	19	35	75
改良方式	37	24	37	31

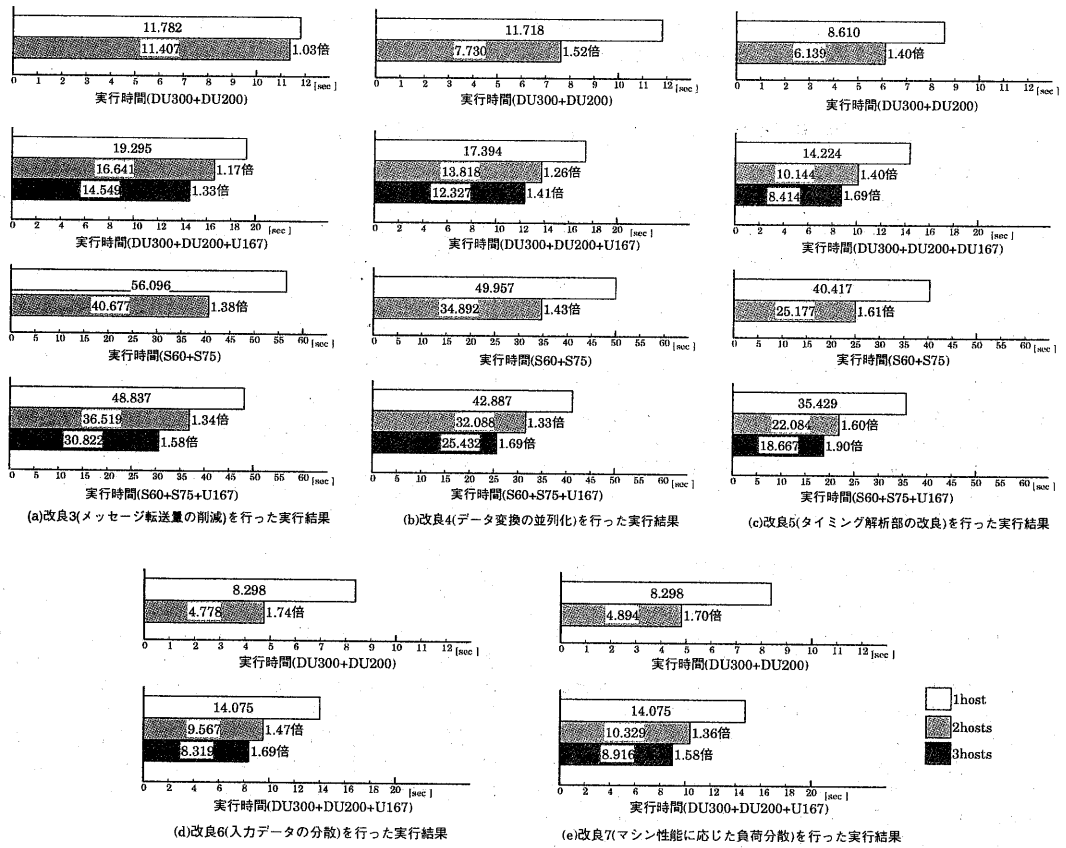


図 7: 実行結果

実行速度の比較

表 3に示す様に、約 7 倍の速度向上がみられた。「従来方式」では、各インデックストオブジェクト間の独立性に乏しく、頻繁に回路接続の為のメッセージ転送を行なっていた。そのため、メッセージの衝突が起こりやすく、メッセージが停滞してしまう。更に、前に述べた通り、タイミング解析の負荷の偏りが生じてしまう。「改良方式」では、この 2 つの問題を解決し、負荷分散の効果もあり、大幅な性能向上を得た。

表 3: 実行時間の比較 (秒)

	従来方式	改良方式
実行結果	17.612	2.492

4.2.2 台数効果と実行速度

ここからは、素子数 837、FF 数 47 の回路を用いて、実験を行なった。

図中の棒グラフ内の数値は、実行時間を表しており、2 台以上では、棒グラフの右に 1 台に対する台数効果を示す。

まず、改良 3(メッセージ転送の削減) を実装したもので、実験を行なった。その台数効果と実行時間を図 7(a) 示す。なお、ネットリストの内部データへの変換は逐次的に行なっている。この結果より、DU300+DU200(以後、この組合せを高速マシン対と呼ぶ) を使用したものでは、1.03 倍の台数効果しか得られていない。しかし、そのマシン構成以外 (以後、この組合せを低速マシン群と呼ぶ) では、2 台で 1.17 倍～1.38 倍、3 台でそれぞれ 1.33 倍、1.58 倍の台数効果が得られている事が分かる。その理由は、高速マシン対では、プロセッサの性能が他のマシン構成に比べて優れており、逐次的に行なっているデータ変換処理 (ディスク上のデータの読み込み) のオーバーヘッドが相対的に大きいためである。

実行時間は、対象とするマシンの台数が 2 台から 3 台に増えると、1 台での実行時間でも大幅な差が生じている。この理由は、高速マシン対、低速マシン群、共に 3 台目の速度差が大きく (3 台目のマシン U167 の 1 台での実行時間は 34.320 秒) なっているためである。

4.3 データ変換の並列化 (改良 4) の効果

ここでは、改良 3(メッセージ転送量の削減) に加えて、改良 4(データ変換の並列化) を実装したもので、実験を行なった。この改良による、台数効果と実行時間の変化を図 7(b) に示す。

台数効果については、前節の結果に比べて、高速マシン対では 1.03 倍から 1.52 倍と大幅な台数効果の向上がみられる。その理由は以下の通りである。

データ変換処理はディスクからネットリストを読み込み、処理しているので、マシンの計算能力によらずほぼ一定の時間がかかる。これにより、マシンの計算能力が高ければ高い程、タイミング解析部の処理時間に比べて変換処理にかかる時間が相対的に大きくなる事が言える。そのた

め、マシンの計算能力が比較的高いものほど、データ変換の並列化による効果があらわれ、台数効果の向上が大きくなる。

4.4 タイミング解析部の改良 (改良 5) の効果

ここでは、改良 4(データ変換の並列化) に加えて改良 5(タイミング解析部の改良) を実装したもので実験を行なった。その台数効果と実行時間を図 7(c) に示す。

データ変換の並列化を行なったものに比べて、低速マシン群では、台数効果の向上がみられた。その理由を以下に述べる。

タイミング解析部が高速化された事によって、その部分の処理時間が減少する。そのため、タイミング解析部の処理時間の偏りがこの改良する前に比べて相対的に減少する。しかし、高速マシン対では、前述した変換処理のオーバーヘッドが相対的に増し、その変換処理部分のノード稼働率の偏りが大きな影響を与えることとなり、高速マシン対では、台数効果が減少した。

4.5 入力データの分散 (改良 6) の効果

ここでは、改良 5(タイミング解析部の改良) に加えて改良 6(入力データの分散) を実装したもので実験を行なった。その、台数効果と実行時間を図 7(d) に示す。

前節の結果に比べて、高速マシン対では大幅な台数効果の向上がみられた。その理由を以下に述べる。

前節にも述べた通り、高速マシン対ではネットリストのデータ変換に時間がかかる。それは、ネットワークを介してディスクのアクセスを行なっているためと、1 つのディスクに集中して読み込みを行なっているためである。そこで、ディスクのアクセスを各ホストに分散させることにより、台数効果の大幅な上昇がみられる。

この結果では、前節の改良からの全体的な実行性能の向上は、1 台の場合では平均で 1.03 倍にとどまっており、2、3 台では平均 1.12 倍となっている。これは、この改良を行なう前の実行では、1 台の時にネットワークの衝突が起きないが、2 台以上では複数のホストから 1 つのホストにディスクのアクセス要求が出されるため、ネットワークの衝突が起こる。この改良を行なう事によって、ディスクアクセスにおけるネットワークの衝突が解消された事がわかる。

4.6 マシン性能に応じた負荷分散 (改良 7) の効果

ここでは、改良 6(入力データの分散) に加えて改良 7(マシン性能に応じた負荷分散) を実装したもので実験を行なった。その台数効果と実行時間を図 7(e) に示す。

台数効果と全体的な実行性能は、共に前節の改良 6(入力データの分散) から比べて減少している。これは、この実験で用いた回路規模では、タイミング解析処理にかかる時間が短く、相対的な負荷分散処理時間が長くなる。そのため、2 台以上の場合でも実行速度の上昇がみられなかった。しかし、回路規模が大きくなれば、実行性能の向上が見込める。その理由を以下に述べる。分散処理にかかる時間は、FF 数とインデックストオブジェクト数に依存するため、分散処理はデータ量を n とすると、$O(n)$ の割合で処理時間が増大していく。しかし、一般にタイミング解析は $O(n^2)$ の割合で処理時間が増大していく。そのため、回路規模が大きくなれば負荷分散処理に比べて、タイミング解析部の処理量が大幅に上昇する。

5 おわりに

本稿では、A-NETL によるタイミング解析プログラムの作成を行ない、PVM を用いた WS クラスタ上で実験を行った。5 種類のプログラムを作成し、マシンの構成を変えて実行した。従来のプログラムから実速度で約 7 倍の速度向上がみられ、マシン数が 1 台から 2 台で、最大で 1.74 倍の速度向上を実現した。

今後の課題としては、マシン性能の差が大きい場合における実行性能の低下の改善、より大規模な論理回路を用いて実験し評価を行なう、などが挙げられる。前者を実現するために、現在均等に分割している入力データを各マシンの処理能力に応じて適正に分割することが考えられる。後者の目的は、マシン性能に応じた負荷分散による効果 (4.6節参照) で述べた回路の大規模化に対する実行性能を明らかにする事である。

謝辞

本研究において、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) 湯浅克男部長・鈴木康弘課長・大川正一氏・町田光之氏に感謝する。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 (C) 課題番号 09680324、基盤研究 (B) 課題番号 10558039、 奨励研究 (A) 課題番号 09780237 の援助による。

参考文献

[1] 高山、広瀬、下郡、庄司、岩下 : "大規模論理装置に対する高速設計検証システム", 情処技法,95-ARC-110-2(or 95-DA-73-2),1995.

[2] T.Baba,T.Yoshinaga and T.Furuta:Programming and Debugging for Massive Parallelism:The Case for a Parallel Object-Oriented Language A-NETL,Proc.OBPDC'95,Lecture Notes in Computer Science 1107,pp.38-58(1996).

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck and V. Sunderan : "PVM3 USER'S GUIDE AND REFERENCE MANUAL", ORNL/TM-12187,Sep 1994.

[4] 小栗 清、名古屋 彰、野村 亮、雪下 充輝 : "始めての PARTHENON", CQ 出版 (1995), http://www.kecl.ntt.co.jp/car/parthe/index_j.htm

[5] 清川勇司: "並列オブジェクト指向概念に基づくタイミング解析の並列化", 宇都宮大学工学部情報工学科卒業論文 , No17, 1996.3.