

データフローマシンを用いた 故障シミュレーションシステム

中田 恒夫, 藤田 昌宏, 田中 英彦, 元岡 達
(東京大学 工学部)

§ 1. はじめに

論理回路の大規模化、複雑化に伴い回路の試験・診断が重要視されるようになってきている。故障シミュレーションは試験入力データの有効性判定、故障辞書作成など試験・診断において重要な位置を占めるが、その計算時間はゲート数 n に対し $O(n^2) \sim O(n^3)$ であり、従来のノイマン型計算機上のソフトウェア・シミュレータを用いる限り、VLSI時代に対応することは難しいと考えられている。これに対し、論理回路の動作の並列性に注目しマルチプロセッサにより論理/故障シミュレーションを高速に行なう方法が考えられ、専用マシンが近年いくつか提案あるいは作製されている〔⑦, ⑧, ⑨〕。

本研究では、論理回路の動作の並列性に注目するという点では他の専用マシンの考え方と同様であるが、論理回路とデータフローグラフの類似性に着目しデータフローマシン上で故障シミュレーションの並列処理を行なうことを考える。すなわち、汎用の並列計算機を用いて処理を高速化することを目的としている。

我々は既に、マルチマイクロプロセッサシステムTOPSTAR-II〔①〕上に故障シミュレーションの並列処理システムを実装している〔②〕。ここではシステムの動作解析を通じて性能評価・検討を行なった結果を報告する。

§ 2. 故障シミュレーションの並列処理法

故障シミュレーション・アルゴリズムとして、

- ①パラレル法
- ②ディダクティブ法
- ③コンカレント法〔⑤, ⑥〕

の3方式が知られている。本システムでは、上記の3方式のうち、柔軟性、拡張性などの点で最も優れた方式であるコンカレント法をもとにしている。

本システムでは、並列処理の基本単位として、一論理素子に対して行われる処理を考え、論理回路における動作の並列性を十分に引き出す。また、複数の入力データに対し故障シミュレーションを行なうことを想定し、これらをパイプライン的に処理することによってマルチプロセッサを十分に活用し高速化を図る。

なお、一論理素子に対する処理の実体は、いくつかの故障回路に対応した素子の評価であり、これも並列処理が可能であるが本システムではこの並列性は利用していない。

2. 1 Concurrency

シミュレーションの対象となる回路のモデルとして図2.1のように組合せ回路部と記憶回路部に分離されたものを考える。これらについて以下に示すようにして回路をデータフローグラフに変換し並列性を引き出す。

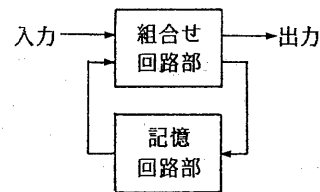
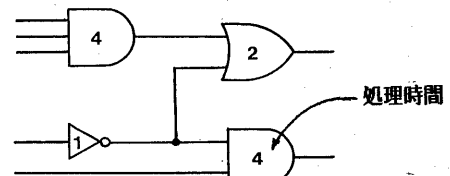


図2.1 回路のモデル

①組合せ回路部

基本的にはランク・オーダによる0遅延のシミュレーションを行なう。同一のランクにある素子に対する処理は並列に行ない得るが素子により処理時間にばらつきがある場合は並列性を十分引き出しているとはいえない。一般に各素子に対する処理を始めるには各素子の入力データがすべて揃っていればよい。従って各素子をデータフローグラフにおけるノードに対応させデータフロー制御を行なうことにすると素子の処理順序が自然に整えられ回路の持つ並列性が最大限に生かされることになる(図2.2)。



総処理時間 { ランク・オーダ 8
 データフロー 6

図2.2 ランク・オーダとデータフロー

②記憶回路部

単一遅延のシミュレーションを行なう。組合せ回路部の場合と同様に論理素子とノードを1対1に対応させるが、通常のデータフローでは記憶を表現することができないため特に記憶ノードなるものを設ける。記憶ノードとは出力から入力へフィードバックするアークを持ち、かつ、入出力間に遅延を有するものである(図2.3)。

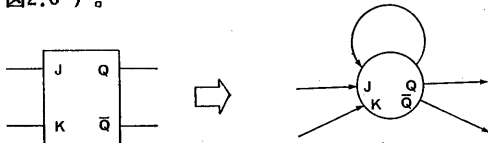


図2.3 フリップフロップと記憶ノード

2.2 Pipeline

前節で論じた並列性は大規模な回路では非常に大きな値になるが、実行中常に大きな値が保たれるとは限らない。多数のプロセッサを有効に利用するため、本システムでは複数の入力データを次々とデータフロープログラムに送りこみ、十分な並列性を確保する。

8.3. 実験システムの構成

実験システムの構成を図3.1に示す。図中で本システム独自のものは、次の4つである。

- ・システム・プログラム
- ・テーブル生成コンパイラ
- ・故障シミュレーション・プログラム
- ・IPLプログラム

以下でそれぞれについて簡単に説明する。

①システム・プログラム

TOPSTAR-II [付録参照] 上でデータ駆動によるデータフロー処理を実現するプログラムである。本プログラムはZ-80アセンブリ言語により記述されている。

②テーブル生成コンパイラ

制御テーブルには、プロシージャとノードの対応関係、ノード間の接続情報が記載されており、データフロー処理を行なう際に参照される。ユーザはテーブル記述言語をもちいてこれらを記述しコンパイラにかけることにより、マシンで用いるテーブルを得る。

③故障シミュレーションプログラム

論理素子における処理の中核を記述したプログラムで、C言語で書かれている。

処理の主要部はすべての論理素子について共通であるため、ユーザは論理関数をC言語で記述し本プログラムとリンクさせて、ノードに対応するプロシージャを作り出すようにしている。

なお、メモリ容量の制約からシミュレータの機能を以下のように定めている。

- ・論理値は、0, 1の2値
- ・単一縮退故障を対象とする。
- ・ゲートレベル、機能レベルいずれも可。
- ・遅延モデルは0遅延または単一遅延のみ。

④IPLプログラム

実行に必要なオブジェクトプログラムはインテル形式でフロッピーディスクに収められている。プログラムを実行する際は、IPLプログラムによりオブジェクトを各CM・PMに送り適当な合図とともに実行を開始する。

IPLプログラムはZ-80アセンブリ言語で記述されている。

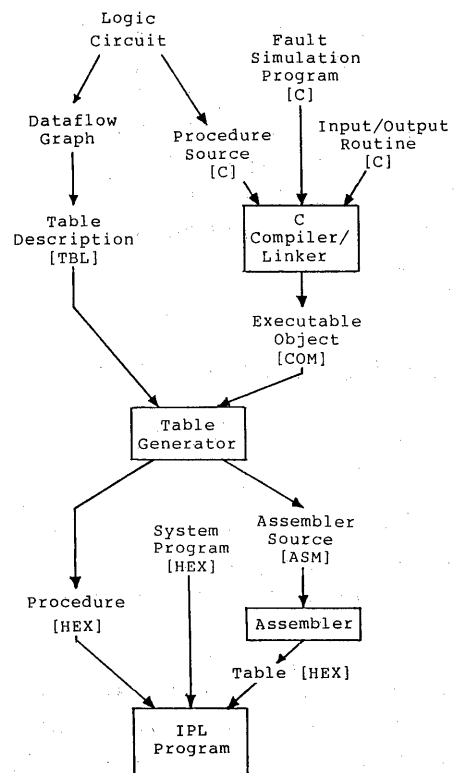


図3.1 実験システムの構成

§ 4. システムの動作解析

本章では図4.1のBCD加算器のシミュレーション例をもとに動作解析を行なう。

4. 1 並列処理度の測定

本システムは並列処理による高速化を目標としているため、評価基準として並列処理度を用いる。並列処理度とは、逐次処理における処理速度に対する、並列処理における処理速度の比のことであるが、ここでは次の定義とする。

$$\text{並列処理度} = \frac{\text{プログラムの実行時間の総和}}{\text{TOPSTAR-IIでの処理時間}}$$

4. 2 並列処理度の測定

PM台数、queue容量、割り付けで用いるCM台数を変え並列処理度を求めた。結果を図4.2に示す。

また、他の応用例での測定結果を図4.3に示す。

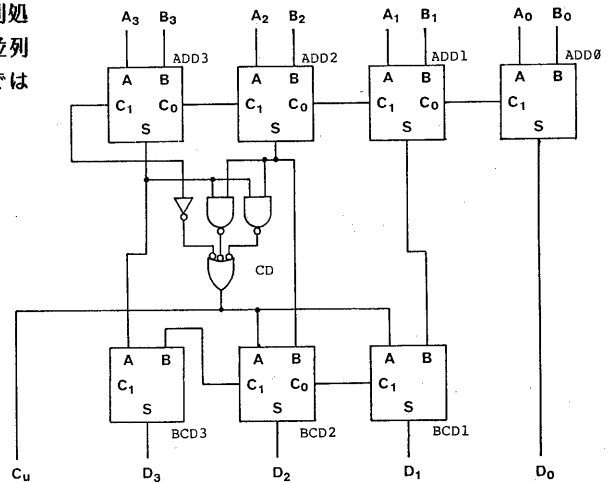
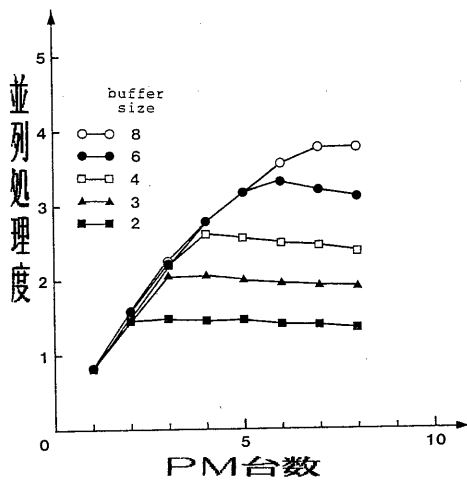
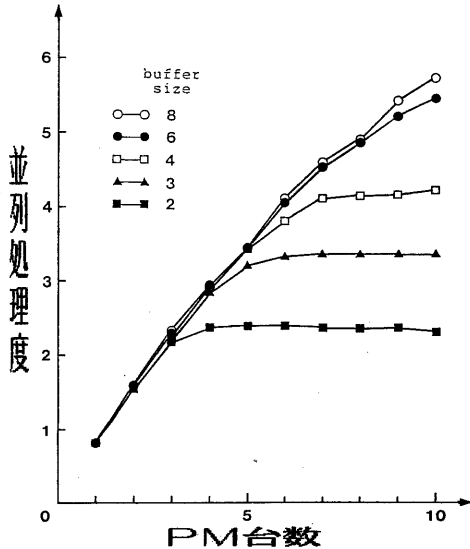


図4.1 BCD加算器

(a) CM1台



(b) CM3台



(c) CM8台

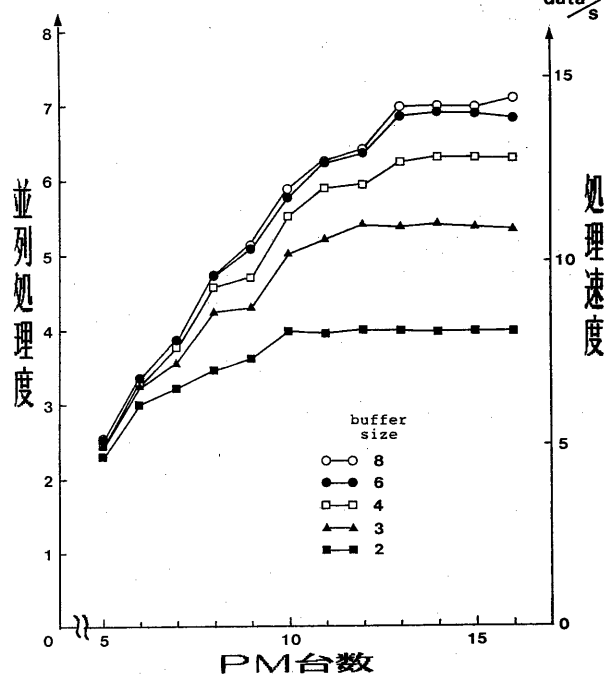


図4.2 PM台数と並列処理度

circuit	1	2	3	4
simulation level	function	gate	gate	gate
number of gates	35	31	31	46
number of nodes	8	31	31	46
mean time of procedure execution (ms)	49.28	30.76	43.23	137.87
execution time on TOPSTAR-II (s)	6.87	41.62	42.47	165.72
execution time (sequential) (s)	49.23	346.95	483.39	1655.73
parallelism	7.16	6.13	9.38	9.87

1. BCD ADDER
2. 10-TO-4 PRIORITY ENCODER
3. BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS
4. 9-BIT PARITY GENERATORS/CHECKERS

図4.3 種々の回路例での測定結果

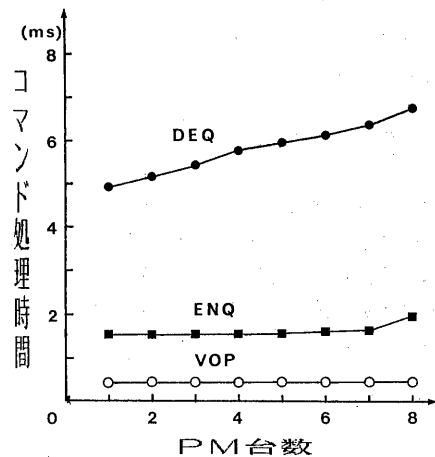


図4.4 各コマンドの平均処理時間 (CM)

4.2 システム・オーバヘッドの測定

TOPSTAR-IIでは、CMとPMの動作は以下のように分類される。

- | | | |
|----|---|----------|
| CM | { | 割り込み待ち |
| | | 割り込み処理 |
| | | 各コマンドの処理 |
| PM | { | 割り込み順番待ち |
| | | 各コマンドの処理 |
| | | プログラムの実行 |

この中でシステム・オーバヘッドに相当するのは、PM側の割り込み順番待ちと各コマンドの処理である。ただしPMはCMに割り込みをかけた後CMからの応答を待つため、CM側の割り込み処理、各コマンドの処理も考慮に入れる必要がある。

BCD加算器の例においてCM1台の割り付け方式の下でオーバヘッド時間の測定を行なった。なおCMにおける割り込み処理時間、PMにおける割り込み順番待ち時間はそれぞれCM、PMのコマンド処理時間の中に含めて考える。測定結果を図4.4、図4.5に示す。

図より明らかなようにCMのコマンド処理時間はPM台数によらずほぼ一定で、値自体もプログラムの平均実行時間に比べて十分小さい。これに対しPMのコマンド処理時間はPM台数の増加とともに急速に増大しプログラムの処理時間と同程度になる。

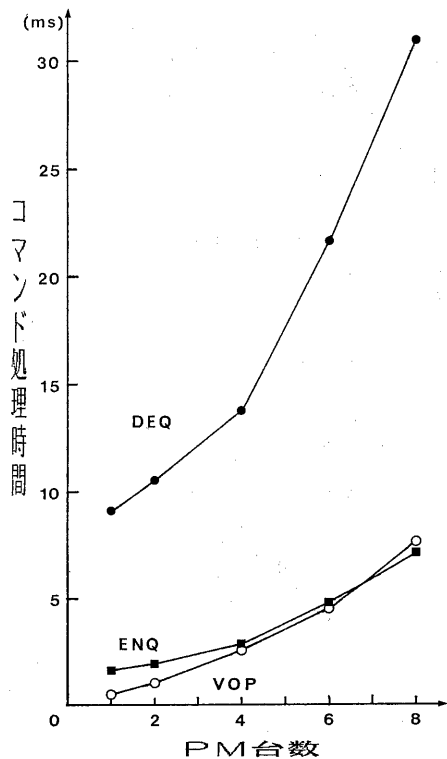


図4.5 各コマンドの平均処理時間 (PM)

§ 5. 評価・検討

5. 1 PM台数と並列処理度

PM台数と並列処理度の関係で注目すべき点を列挙する。

①PM台数が少ないうちは両者はほぼ比例する。

②PM台数の増加とともに並列処理度の伸びが鈍り、ついには飽和してしまう。queue容量をふやすと伸び具合は良くなるが、いずれ飽和に至ることには変わらない。

③CM台数が少ない割り付け方式の下でPM台数をふやすと並列処理度はむしろ低下してしまう。

以下で②、③の原因について考える。

5.1.1 並列処理度飽和の原因

並列処理度を飽和させる原因として考えられるのは次の5つである。

- ①処理対象の並列度の限界
- ②システム・プログラムのオーバーヘッド
- ③割り付け方式……クリティカル・ノード
- ④両端のPMの影響
- ⑤処理開始・終了時の無駄

これらのうち②、③については、それぞれ5. 2、5. 3で詳しく検討し、この2つが主な原因であることを明らかにする。

5.1.2 並列処理度低下の原因

データ駆動型処理システムではPMがCMに“仕事”を要求する方式をとっているが、CM内の“仕事”の量が常に少ないような例では“仕事”のないPMがふえ、次々とCMにDEQをかける。CM側では、無駄なDEQの処理に忙殺され真に必要な処理、すなわち“仕事”を与えるDEQ、ENQ、VOPに対する応答が遅れてしまう。このような状況下ではPMがふえるほど“仕事”のないPMができ真に必要な処理がますます遅れることになる。これが並列処理度低下の原因である。

現在の制御方式、すなわちPMがCMに“仕事”を要求する方式である限り以上のような並列処理度の低下は避けることができないが制御の高速化および分散化によりほとんど無視できる程度に引き下げることができる。

5. 2 システム・オーバーヘッドの影響

システム・オーバーヘッドが並列処理度に与える影響を、オーバーヘッドを考慮に入れた並列処理度の上限を算出することによって定量化する。4. 2で示したようにPM台数がふえるとオーバーヘッドが急激に増大するが、このうち処理を進めていく上で最低限必要な分を不可避なオーバーヘッドと考え並列処理度の上限の計算に組み入れる。なお、不可避なオーバーヘッド時間は図4.4、図4.5でCM1台、PM1台の場合のコマンド処理時間に相当する。

BCD加算器の例で、オーバーヘッドが各CM、PMにうまく分配されたとすると、並列処理度の最大値は10.8となる。

5. 3 クリティカル・ノードの影響

クリティカル・ノードとは処理ネックとなるノードである。クリティカル・ノードにはデータフロープログラムの構造に依存するものと、実行環境に依存するものの2種に分類できるが、前者はパイプライン化により悪影響は残らない。従ってここでは後者のみ考える。

TOPSTAR-IIにおいては各CMに直接結合しているPMは8台であるためループがない限り各ノードの処理は同時に最大8台のPMで実行され得る。しかし実行結果をCMに送り返す都合上、実際にノードの処理に携わることのできるPMは少なくなる場合がある。このようなノードは処理が滞りがちになり処理ネックとなる。

BCD加算器の例でCM8台の割り付け方式の場合、クリティカル・ノードを考慮に入れた並列処理度の上限は11.8となる。オーバーヘッドの影響も合わせて考えた場合、最大並列処理度は7.9となる。

5. 2、5. 3の結果をまとめると以下の通り。

理論的上限	16.0
システム・オーバーヘッドを考慮した上限	10.8
クリティカル・ノードを考慮した上限	11.8
両者を考慮した上限	7.9
実測における上限	7.2

また、これらを並列処理度のグラフに重ねたものを図5.1に示す。

以上の議論から並列処理度を飽和させる主要因はシステム・オーバーヘッドとクリティカル・ノードであると言える。

5.4 割り付け方式

ノードの割り付け方式としては、実行前に固定的に割り付ける静的割り付けと実行中に割り付けが変化する動的割り付けが考えられる。一般に動的割り付けの方が優れた割り付けを実現しやすいが、処理時間を測定しながら割り付けを変更するためオーバーヘッドが大きくパイプライン効果があがらないという難点がある。そこで以下では静的割り付けのみを検討する。

静的割り付けで考慮すべき点は次の2点である。

- ①通信のオーバーヘッドを減らすこと
- ②うまく負荷を分散させること

図5.2 (a) のデータフローグラフの各ノードをCM 4台に割り付ける場合、①、②を重視すると、それぞれ図5.2 (b), (c) の割り付けが得られる。

①を重視する方式の利点は、割り付けが簡単でCM、PM台数がふえてもオーバーヘッドがあまり大きくなり、割り付けの難しさも変わらないことであり、欠点としては負荷が特定のPMに偏りやすいこと、処理開始、終了時に多くの遊びPMを生じることなどが挙げられる。

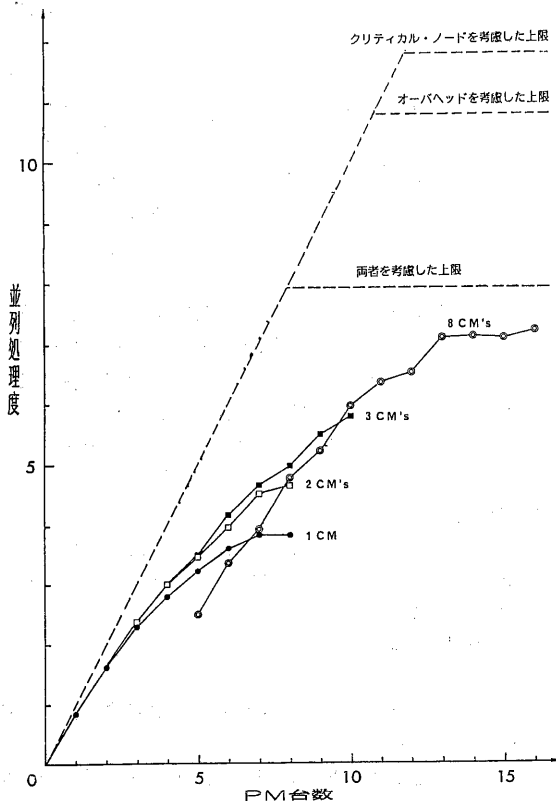


図5.1 並列処理度の上限

なお、②を重視する割り付けでは上の利点・欠点が裏返しになる。

図4.3 の3番目の応用例に対し①、②を重視した割り付け、およびランダムな割り付けに対し並列処理度を測定した結果を下に示す。

通信のオーバーヘッド軽減を重視	9.53
負荷分散を重視	9.39
ランダム	7.90

この例で見る限り両方式の差は小さく、データフローグラフ上で近いノードはTOPSTAR-IIの上でも近いCMに置くことが自然であることから、まず通信のオーバーヘッドの軽減を重視し、その中で負荷分散を図るべきであろう。

5.5 システムの大規模化の検討

TOPSTARが現在のアーキテクチャを保ったまま各モジュールを横方向に延ばして大規模なシステムを構成し、大きな回路を扱った場合の本システムの性能を考える。

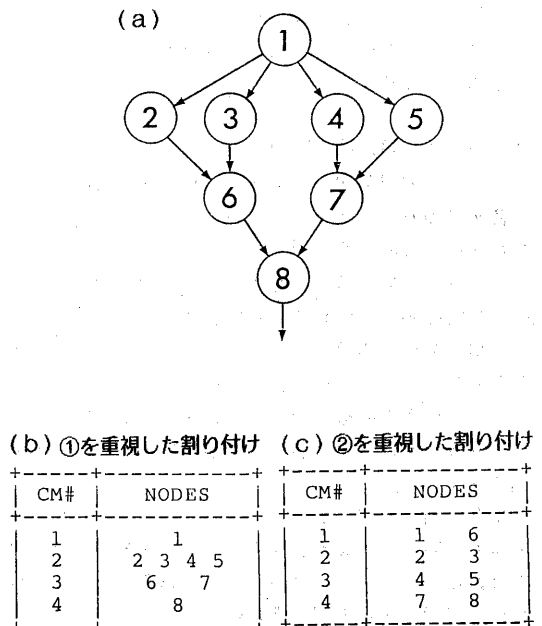


図5.2 データフローグラフと割り付け

5.5.1 処理能力

回路規模の増大は一般に並列性を高めることにつながり性能の向上が期待できる。この際、問題となるのはTOPSTARシステムの大規模化に伴う通信のオーバーヘッドの増大である。すなわち遠くにあるノードとの通信が多く必要となりリレーのオーバーヘッドが処理能力を抑える可能性がある。

しかし論理回路の規模増大は論理深度を深めるか、あるいはモジュール化を進めることにつながり、いずれの場合も通信のローカリティは保証され、オーバーヘッドはそれほど大きくならないと考えられる。

5.5.2 割り付け

回路のモジュール化が進めばモジュール内の各論理素子の割り付けをローカルに行ない、その後でモジュール間の配置を考えればよい。論理深度の深まりは、5.4の手法をそのまま適用することで対応できる。むしろ問題はテーブル生成コンパイラの負担増であり並列処理による高速化などを検討すべきである。

以上のように本システムにおける基本構成を大きく変更することなく大規模化が可能であり、その際にも十分特徴が発揮されて高速なシミュレーションができると考えられる。

§ 6. 結論

論理回路とデータフローグラフの類似性に着目しデータフロー制御を利用して、故障シミュレーションを並列に処理する方式を提案し、データフローマシンTOPSTAR-II上に実験システムを実装した。

作製したシステムを実際の回路に適用して性能を測定した結果プロセッシングモジュール16台で逐次処理の10倍程度の処理速度を実現していることがわかった。また動作解析から処理速度を抑える主要因がシステム・オーバーヘッドとクリティカル・ノードにあることを示すとともに、ノードの割り付け手法についても検討を加え論理回路のローカリティを生かして通信のオーバーヘッドを軽減する方式で十分な性能が得られることを示した。

シミュレータの機能拡張、ノード割り付けの自動化などは今後の課題である。

《 参考文献 》

- ①Suzuki, T. et al.: Procedure Level Data Flow Processing on Dynamic Structure Multimicroprocessors, JIP, Vol. 5, No. 1, pp.11-16, 1982
- ②中田、田中、元岡: “データフローマシンTOPSTAR-IIによるコンカレント故障シミュレーション”
情報処理学会電子装置設計技術研究会資料11-3
1980年10月
- ③中田、田中、元岡: “データフローマシンによる故障シミュレータの性能評価”
情報処理学会第25回全国大会2F-3, 1982
- ④中田、田中、元岡: “データフローマシン“TOPSTAR-II”におけるノード割り付け手法”
情報処理学会第26回全国大会3P-4, 1983
- ⑤Abramovici, M. et al.: Concurrent Fault Simulation and Functional Level Modeling, Proc. of 15th DAC, pp.128-137, 1978
- ⑥Ulrich, E. G. and Baker, T.: The Concurrent Simulation of Nearly Identical Digital Networks, Proc. of 10th Design Automation Workshop, pp.145-150, 1973
- ⑦Pfister, G. F.: The Yorktown Simulation Engine, Proc. of 19th DAC, pp.242-249, 1982
- ⑧Abramovici, M. et al.: A Logic Simulation Machine, Proc. of 19th DAC, pp.65-73, 1982
- ⑨Sasaki, T. et al.: HAL; A Block Level Hardware Logic Simulator, Proc. of 20th DAC, pp.150-156, 1983

《付 録：TOPSTAR-IIのシステム構成》

(1) アーキテクチャ

TOPSTAR-IIは、プロシージャレベルのデータフローによって分散制御されるマルチマイクロプロセッサシステムである。

アーキテクチャの特徴としては以下の3点が挙げられる。

①モジュール構成(図0.1 参照)

TOPSTAR-IIは、CM (Communication/Control Module) および、PM (Processing Module) の2種類のモジュール多数から構成されている。現在のシステムでは、CM8台、PM16台となっている。各モジュールは、CPUとしてZ-80を搭載している。

②オーバラップした部分結合

TOPSTARシステムでは多数のCMが両方向に伸び、その下に多数のPMが結合することを想定しているが、その際、CM・PM間の結合を完全結合とすることは難しい。そこでTOPSTARシステムでは処理対象を、結合に局所性のあるデータフローグラフに限定しモジュール間結合にも局所性を持たせている。即ち各CMはいくつかの“近い”PMに、各PMはいくつかの“近い”CMに直接結合させている。

③DMAによる高速データ転送

TOPSTAR-II上でプロシージャレベルのデータフロー処理を実現する場合、CM・PM間で比較的多量のデータ転送が行われる。データ転送が処理ネットワークにならないようにTOPSTAR-IIでは、DMAによりCM・PMのローカル・メモリ間的高速データ転送を実現している。転送レートはDMA準備時間を除き $1.6\mu\text{sec}/\text{バイト}$ である。

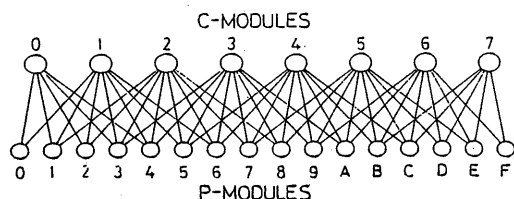


図0.1 TOPSTAR-IIの構成

(2) 制御方式

TOPSTAR-IIの制御方式の特徴は次の3点に要約できる。

- ①データ駆動による分散制御
- ②PMの自由競争による負荷分散
- ③追い越しを許すパイプライン処理

これらの特徴を具体例を挙げて説明する。

TOPSTAR-IIの処理対象は、プロシージャをノードとするデータフロープログラムである。各CMには前もっていくつかのノードが割り当てられ、ノードごとに入力データを蓄えるバッファ (queue) が用意されている。PMは結合範囲にあるCMに割り込みをかけ“仕事”を求めるコマンド (DE Queue) を送る。CM側では実行可能な“仕事”があるかどうかを調べ、あれば入力データとプロシージャを送り、なければその旨知らせる。ここである“仕事”が実行可能であるとは、次の2条件が共に満たされたことを指す。

①プロシージャを実行するのに必要な入力データがすべて揃っている。

②結果の送り先ノードすべてについてqueue が少なくともひとつずつは空いている。

“仕事”を与えられなかったPMは次のCMに割り込みをかけ“仕事”を求めるが、与えられた場合、まずqueue がひとつ空いたことをそのノードを出力先とするノードすべてに知らせる (VOP)。

次に、PMはプロシージャを実行し、実行結果をアウトセットのノードに送る (ENQ) ことにより、一連の動作を完了する。その結果PMはアイドルとなり再び“仕事”を求めてCMに割り込みをかける。

図0.2 は以上の一連の動作を図式化したものである。図では (a) のデータフロープログラムでノードN0の処理をP5が行なう場合を想定している。なお、図中の破線はP5とC0の間に結合がないためにVOPが届かないことを意味するが、このVOPは後でP3、P4などが行なうことになる。

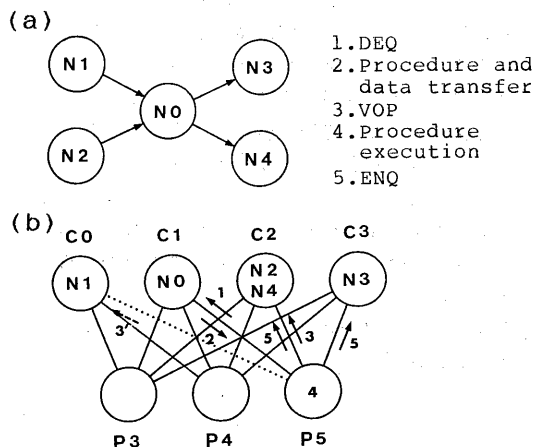


図0.2 TOPSTAR-IIの動作