数 値 解 析 17－5
（１９８６．７．４）

マルチプロセッサ・システムにおける回路解析コードの並列処理

福井　義成（東芝）　　大吉　哲郎（日本クレイ）
加藤　毅彦（日本クレイ）渡辺　庸一（日本クレイ）

我々は，回路解析コードの高速化を多方面から行ってきた。ここでは，CRAY X-MPにおいて，回路行列を構成する部分にマルチタスキングを適用した結果について述べる。マルチタスキングを適用した部分については，２CPUの場合，約２倍の高速化が得られた。

ここで，マルチタスキングとは，１つのユーザで同時に複数個のCPUを使用し，各々異なった計算を並列処理するものである。この並列処理は，スカラー・ベクトル処理の双方で可能であり，アルゴリズムやデータ構造などの点からベクトル化不可能なプログラムでも，各々が独立した計算であれば適用可能である。

PARALLEL PROCESSED CIRCUIT ANALYSIS BY THE MULTI-PROCESSOR SUPERCOMPUTER

Yoshinari Fukui*, Tetsuro Oyoshi**
Takehiko Kato**, Youichi Watanabe**

*TOSHIBA CORPORATION, **Software department, Cray Research Japan Ltd.

The authors have been searching the ways to reduce execution time of a circuit analysis code from all the angles. This time, utilization of multitasking capability of CRAY X-MP with two CPUs was applied to the circuit matrix generation part of a code and the performance was improved by the factor of two.

The multitasking is the technique to utilize multi-CPUs in one job in order to process independent calculations in parallel. By this parallel processing very substantial performance improvement is expected to even programs which can not be vectorized because of the applied algorithm or data structures since it is effective to not only vector processing parts but also scalar processing parts.

On this paper, a summary of the multitasking technique as well as its application to the circuit analysis are discussed.

１章　はじめに

我々は，回路解析コードＳＰＩＣＥ－ＧＴの高速化を多方面から行ってきた。ここではＣＲＡＹ　Ｘ－ＭＰ上におけるプログラミング上の手法について報告する。ＳＰＩＣＥ－ＧＴは，ＳＰＩＣＥ２Ｇ．６をベースに東芝で改良・機能強化したものである。

プログラム的手法による高速化の観点から，回路解析コードの計算は次の３つの部分に分類することができる。

（１）回路行列の構成

（２）求められた回路行列を解く

（３）入出力や全体の制御など

ＳＰＩＣＥ－ＧＴをＣＲＡＹ　Ｘ－ＭＰにインプリメントするときに（２）の回路行列の求解の部分の高速化を行った。手法としてはコード生成を行い、ＦＯＲＴＲＡＮ版の求解ルーチンに比べて約２６倍の高速化を実現した。この結果，回路行列の求解に要するＣＰＵ時間は，１つの例では，ＦＯＲＴＲＡＮ版では約７６％であったものが，コード生成では約１６％になった。この結果，コード生成版では，（１）の回路行列の構成に約７８％のＣＰＵ時間を必要とするようになった。

この回路行列を構成する部分を高速化するために，複数個のＣＰＵを同時に使用するマルチタスキングを適用することとした。以下２章では，マルチタスキングの特徴について，３章では，ＳＰＩＣＥ－ＧＴへの適用について述べ，そして４章では，マルチタスキングの効果について示す。また５章において，今後の展望について述べる。

２章　マルチタスキング

マルチプログラミング環境下では，ＣＰＵの有効利用のため，１つのＣＰＵを複数のジョブが共有する。マルチプロセッシング方式によれば，複数のジョブがジョブ単位に複数のＣＰＵを共有する。さらに１つのジョブを複数のタスクに分割し，タスク単位にＣＰＵを割当て，並列化を高め，システム全体のパフォーマンスを改善する“マルチタスキング”と称す試みがなされた。（図１）

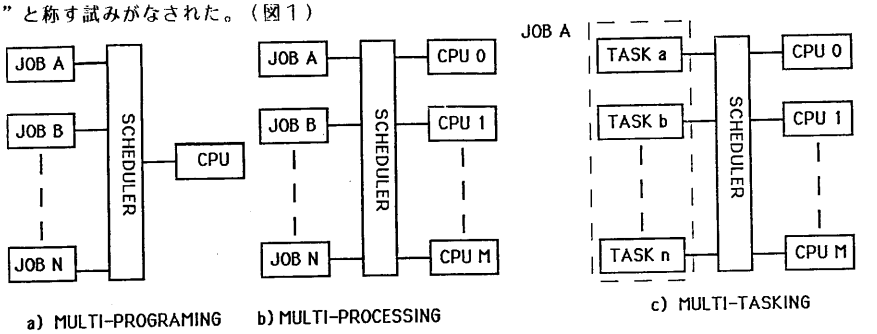

図１　ＣＲＡＹ　Ｘ－ＭＰにおける並列多重処理

ＣＰＵの並列使用は以下の５つのレベルに分類できる。

１° ジョブレベル

２° ジョブステップレベル

３° ルーチンレベル

４° ループレベル

５° ステートメントレベル

ジョブレベル，ジョブステップレベルの並列化は従来のマルチプロセッシング方式の並列処理であるが，マルチタスキングはルーチンレベルに並列処理を適用した。

マルチタスキングジョブを単独で実行した場合，ＣＰＵの同期のためのオーバーヘッドを無視すれば，２ＣＰＵで２倍，４ＣＰＵで４倍のスループットの向上が得られる。これに対し，マルチプログラミング環境下（バッチ環境）で実行した場合，マルチタスキングジョブの１タスクは他のジョブとＣＰＵを競合するため，単独実行のようなパフォーマンスは得られない。しかし大メモリーを要求するジョブへのマルチタスキングの適用は，システム全体の大幅なパフォーマンスの向上につながる。例えば，４ＣＰＵ，主記憶装置８メガワードの計算機上で７．５メガワードの

メモリーを用いるジョブが実行されているとする。このとき，０．５メガワード以下のジョブが投入されるまで３つのＣＰＵは遊休状態にある。もしこのジョブが４つのＣＰＵを用いるマルチタスキングジョブであれば，４つのＣＰＵは常に実行状態にあり，このジョブのシステム滞在時間は１／４に減少し，システム全体のパフォーマンスの向上につながる。

前述のようにマルチタスキングはルーチンレベルでタスクが定義され並列処理が行われる。タスクの発生，タスク間の同期等は全てＦＯＲＴＲＡＮプログラムからライブラリを呼ぶことにより行われる。マルチタスキングはスカラー処理，ベクトル処理双方に適用され，ベクトル化が最内側ループ周辺においてのみ適用されるのに対し，マルチタスキングはより外側のループ，さらに独立したサブルーチンに対しても適用される。

以下にマルチタスキングをサポートするハードウェア資源について述べる。図２に２ＣＰＵを有するＣＲＡＹ　Ｘ－ＭＰの概略を示す。メインメモリー上のマルチタスキングプログラムは２つのＣＰＵを並行に使用する。タスクの発生・消滅，タスク間の同期には２ＣＰＵ間の通信が必要となるが，これにはいずれのＣＰＵからもアクセス可能なＣＰＵコミュニケーションレジスタが使われる。これらのレジスタはタスクの発生・ウエイト・終了のシグナル交換に用いられる。

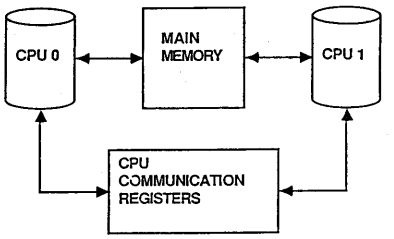

図２　ＣＲＡＹ　Ｘ－ＭＰ２のハードウェア構成

マルチタスキングをサポートするソフトウェア資源は以下の３つである。

１° 再入可能なコードを生成するＦＯＲＴＲＡＮコンパイラ

２° タスクの発生・消滅・同期を制御するＦＯＲＴＲＡＮライブラリ

３° タスクに対してＣＰＵを割り当てるスケジューラ

ＣＲＡＹ・ＦＯＲＴＲＡＮコンパイラ（ＣＦＴ）は複数のＣＰＵで同一のサブルーチンを実行できる再入可能なコードを生成する。実行形式のＦＯＲＴＲＡＮプログラムはコードエリア，データエリア，さらにデータエリアは各ルーチンに局所的なローカルデータエリア，各ルーチンに共通なコモンデータエリアに分類できる。複数のＣＰＵで同一のサブルーチンを実行する場合，ローカルデータエリアは各々のタスクで独立でなければならない。各ルーチンのローカルデータエリアはタスクの発生時に動的にメモリー内に割りあてられる。さらにコモンデータエリアはタスク間で共有しうるもの，独立なものとに分類できるが，後者をサポートするのがタスクコモンである。通常のコモンエリアがプログラムローディング時に割当てられるのに対し，タスクコモンデータエリアは，ローカルデータエリアと同様，タスク発生時に動的にメモリー内に割当てられる。（図３）にマルチタスキングプログラムのメモリー構成図を示す。

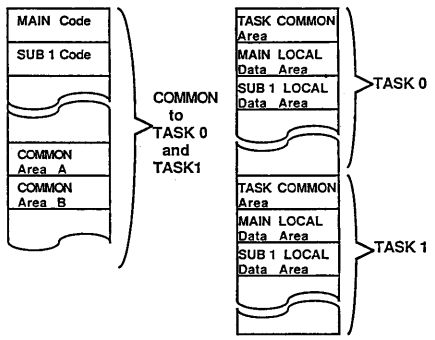

図３　マルチタスキングプログラムのメモリー構成

次にマルチタスキングをサポートするＦＯＲＴＲＡＮライブラリルーチンについて述べる。これらのルーチンは以下の３つに分類できる。

１° タスクの発生・消滅を制御するタスク制御ルーチン

２° タスク間で同期をとるイベント制御ルーチン

３° タスク間のコモンデータを他のタスクから保護するロック制御ルーチン

（表１）に以下のルーチンの呼びだし形式を示す。

タスク制御ルーチンには子タスクを発生させる“ＴＳＫＳＴＡＲＴ”，及び子タスクの実行が終了するまで親タスクの実行を待たせる“ＴＳＫＷＡＩＴ”の２つがある。（図４）

イベント制御ルーチンは，並行して実行しているタスク間

表１　マルチタスキング・ライブラリルーチン

TASK CONTROL ROUTINE	CALL TSKSTART(TASKID,Subname,[Arg])
	CALL TSKWAIT(TASKID)
EVENT CONTROL ROUTINE	CALL EVPOST(EVENT)
	CALL EVWAIT(EVENT)
	CALL EVCLEAR(EVENT)
LOCK CONTROL ROUTINE	CALL LOCKON(LOCK)
	CALL LOCKOFF(LOCK)

で同期をとるために用いられる。並行して実行しているタスクに対しイベントを発行する“ＥＶＰＯＳＴ”，イベントが発行されるまで実行を中断する“ＥＶＷＡＩＴ”，及びイベントを受けた後それを解除する“ＥＶＣＬＥＡＲ”の３つがある。（図５）

ロック制御ルーチンは複数のタスク間のコモンデータエリアを他のタスクから保護するために用いられる。以後に続く実行文を“ＬＯＣＫＯＦＦ”が出現するまで他のタスクからロックをかける“ＬＯＣＫＯＮ”，および“ＬＯＣＫＯＮ”でかけたロックを解除する“ＬＯＣＫＯＦＦ”からなる。（図６）

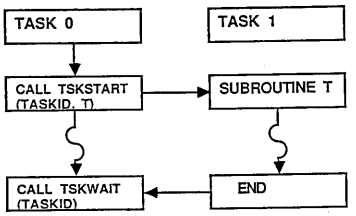

図４　タスク制御ルーチン

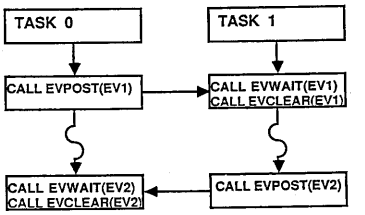

図５　イベント制御ルーチン

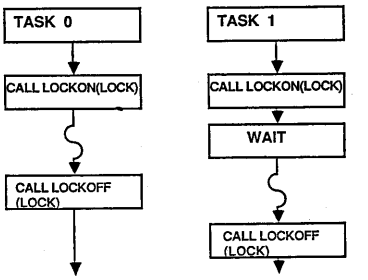

図６　ロック制御ルーチン

図７に行列の積を求めるプログラムにマルチタスキングを適用した例を示す。この例では最外側ループを２つに分割し，２つのＣＰＵで処理している。

最後にマルチタスキングをサポートするスケジューラーについて述べる。マルチプログラミング環境下では，スケジューラーは各ジョブに対しＣＰＵの割り当て・解放を行う。実在するＣＰＵを物理的ＣＰＵ，スケジューリングされジョブに割り当てられたＣＰＵを論理的ＣＰＵと称す。マルチタスキング環境下では，スケジューラは各タスクに対し論理的ＣＰＵの割り当て・解放を行う。マルチタスキングジョブのタスクの状態は，実行状態，他のタスクの割り込みにより実行中断されたサスペンド状態，および実行中断を解除されたウエイト状態の３つである。マルチタスキングライブラリはスケジューラに対しタスクの状態遷移，及び論理的ＣＰＵの割り当て・解放を要求する。タスク発生ルーチン（ＴＳＫＳＴＡＲＴ）はスケジューラに対しタスクのウエイト状態への遷移，および論理的ＣＰＵの割り当てを指示する。実行タスクの割り込みルーチン（ＴＳＫＷＡＩＴ，ＥＶＷＡＩＴ，ＬＯＣＫＯＮ）は実行タスクのサスペンド状態への遷移，および論理的ＣＰＵの解放を指示する。割り込み解除ルーチン（ＥＶＰＯＳＴ，ＬＯＣＫＯＦＦ）はタスクのサスペンド状態からウエイト状態への遷移，論理的ＣＰＵの解放を指示する。タスクの消滅はＴＳＫＳＴＡＲＴで指定したルーチンがＲＥＴＵＲＮ文を実行したとき起こり，割り当てられた論理的ＣＰＵは解放される。（図８）

```
      COMMON  A(100,100), B(100,100), C(100,100)
      EXTERNAL  T
      CALL TSKSTART(TASKID,T)
      DO 4800 I=1,99,2
      DO 4500 J=1,100
        C(I,J)=0.0
        DO 4200 K=1,100
          C(I,J)=C(I,J)+A(I,K)*B(K,J)
4200    CONTINUE
4500  CONTINUE
4800  CONTINUE
      CALL TSKWAIT(TASKID)
      STOP
      END
      SUBROUTINE T
      COMMON  A(100,100), B(100,100), C(100,100)
      DO 4800 I=2,100,2
      DO 4500 J=1,100
        C(I,J)=0.0
        DO 4200 K=1,100
          C(I,J)=C(I,J)+A(I,K)*B(K,J)
4200    CONTINUE
4500  CONTINUE
      RETURN
      END
```

図７　マルチタスキングを適用したプログラムの例

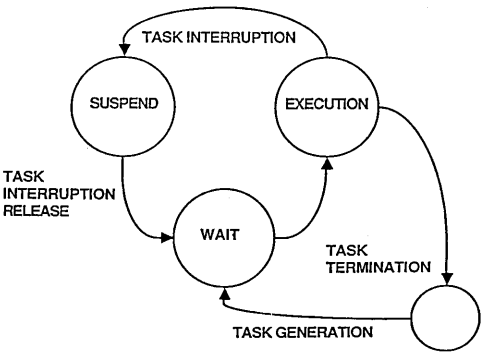

図８　タスクの状態遷移

3章　回路解析の特徴とＳＰＩＣＥ－ＧＴへのマルチタスキングの適用

回路解析コードで行う解析には

（１）直流解析

（２）交流解析

（３）過渡解析

等がある。中でも過渡解析は，計算時間が長くなるため，最も高速化を必要とする解析である。

過渡解析も，１章で述べたように，次の３つの部分に分けて考えることができる。

（１）回路行列の構成

（２）求められた回路行列を解く

（３）入出力や全体の制御など

（２）については，前に述べたように，コード生成を行い高速化を実現した。

ある典型的な例では，（１）～（３）のＣＰＵ使用時間の分布は，次のようになっている。

	（１）	（２）	（３）
ＦＯＲＴＲＡＮ版	２１．８％	７５．９％	２．３％
コード生成版	７８．３％	１５．８％	５．９％

（３）の部分は，全体に占める割合いも小さく，入出力が主な内容であるため，当面は高速化の対象とはしない。全体に占める割合いが，大きくなった時点で考えることとする。

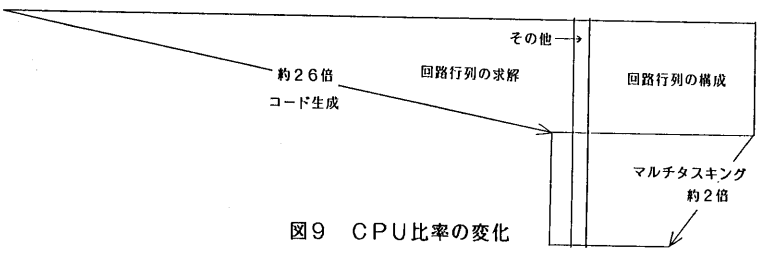

図９　ＣＰＵ比率の変化

回路解析に現れる行列は，ランダム・スパース行列となり，スパース率も高い。そのため，ＳＰＩＣＥでは回路行列を記憶するためのデータ構造としてリスト方式を採用している。その結果，（１）の回路行列を表現するリストをたどって初めてデータを得ることができる。このため，必然的に数値的な計算よりも，条件判定とリスト構造をたどることに多くのＣＰＵ時間を費しているので，ベクトル演算は，簡単に行うことはできない。この部分に，マルチタスキングを適用して高速化を図ることにした。

回路行列の構成の部分をさらに分けると

（ａ）各行列要素の計算（部分行列）

（ｂ）計算された行列要素の全体行列への加算

となる。（ａ）のステップは，各行列要素ごとに独立に計算可能であり，並列処理向きである。（ｂ）のステップは，同一メモリへの複数回の加算があるため，逐次処理になる。しかし，幸いなことに，回路行列においては，部分行列の計算が複雑なため，（ｂ）に必要とされるＣＰＵ時間は，（ａ）に必要とされるＣＰＵ時間に比べて無視することができる。そのため，複数ＣＰＵによる並列処理の効果が大きい。（図１０）

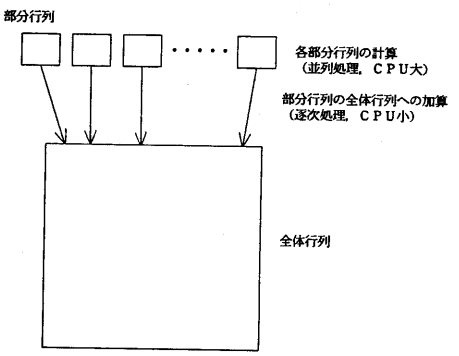

図１０　全体行列の構成

4章　マルチタスキングの効果

単独実行の環境でのマルチタスキングの効果を表2に示す。CPUは2つである。8ケースの中で，ケース（2）～（6）はCPU時間も短く，並列計算の効果は期待できないが，ケース（1）と（8）はCPU時間が長く，最も並列計算の効果が出てほしい問題である。

表2：マルチタスキングの効果

ケース	経過時間（秒）			CPU時間（秒）		
	シングルタスク	マルチタスク（2）	比率	シングルタスク	マルチタスク（2）*	比率
1	373	253	0.68	371	418	1.13
2	55	37	0.67	54	65	1.20
3	55	40	0.73	48	70	1.46
4	47	31	0.66	45	57	1.27
5	47	29	0.62	45	53	1.18
6	47	30	0.64	47	54	1.15
7	169	107	0.63	168	191	1.14
8	354	231	0.65	354	398	1.12

＊ 2つのCPUの合計時間

8ケースのプログラム全体の経過時間の比率は，0.63～0.73倍となっている。特にCPU時間の長いケース（1）と（8）は各々0.68倍，0.65倍と効果的である。オーバーヘッドによるCPU時間の増加も1割程度である。

5章　まとめ

マルチタスキングにより，回路解析コードの高速化が実現できた。ベクトル化が難しいが，並列計算可能な部分を持つプログラムにマルチタスキングが有効であることが実証できた。

一方，ベクトル化されたプログラムも，並列処理なしに，現在の100～1000倍以上の性能を出すことは困難であり，今後の方向としては，複数のCPUによる並列処理が不可欠であろう。並列計算にも次の2つのタイプがある。

（1）汎用の並列計算機 { 各CPUはスカラーCPU
　　　　　　　　　　　 { 各CPUがベクトルCPU

（2）特定目的のための専用型並列計算機

今回報告したマルチタスキングは，汎用の並列計算機の方向に位置する。汎用の並列計算機が有効に利用できるためには，次のことが大切であろう。

（1）並列プログラムの書きやすさ
　　並列アルゴリズムが記述しやすい言語

（2）計算の再現性・検証のしやすさ
　　複数のCPUを使用すると単独CPUではまったく問題とならない，タイミングのズレ等により計算の再現性が困難になる。

並列計算機は，今後必ず出現するため，並列計算機をうまく利用できるような言語・環境を整えることが大切である。