入選

並列回路シミュレーションマシン Cenju†

中田登志之††　田辺記生†††　梶原信樹††
松下智††　小野塚裕美†††
浅野由裕††††　小池誠彦††

Cenjuはモジュール分割法に基づく並列回路シミュレーションのアルゴリズムを効率よく実現することを目的として設計された並列マシンである．本システムでは，モジュール分割法に基づいてシミュレーションの対象となる回路を非線形素子を含む複数個の部分回路群と部分回路群を結合する接続回路網とに分け，部分回路ごとの計算と全体の接続回路網の計算を収束するまで交互に繰り返す．このモジュール分割法により，回路シミュレーションの90%以上の演算量を占める部分回路群におけるモデル評価ならびに行列演算を，各部分回路ごとに独立に演算することが可能となる．Cenjuでは階層バス構成を採用し，64台のプロセッサエレメント(PE)を多段接続網で結合された8本のクラスタバスに8台ずつ接続している．要素プロセッサとしてはMC 68020とMC 68882 (20 MHz)に4 MBの主記憶をもたせた．さらに浮動小数点演算機能を強化するためにWeitek社のWTL 1167を搭載している．Cenjuを用いて回路シミュレーションを実行したところ，トランジスタ数1,688個の小さい回路で64台時に1台のときの15倍，トランジスタ数6,974個の回路で1台のときの15.8倍程度の速度向上を達成した．さらに並列処理の隘路となる逐次部分のLU分解を並列化することにより，27.8倍程度の速度向上を達成するめどが得られた．

1. はじめに

超LSI回路の開発に不可欠な回路シミュレーションは倍精度浮動小数点演算を多用するため，回路の規模が大きくなるにつれて，そのシミュレーション時間の増加が問題になってきている．特にメモリ回路の設計などでは，回路シミュレーション以外に有効な設計手法が確立されていないので，各種パラメータを変えながらシミュレーションを繰り返し，回路の最適化を行う方法が採られている．

回路シミュレーションは科学技術計算の範疇にあるが，非線形素子を含む大規模なスパースマトリックスの微分方程式を解く必要があり，ベクトル長が短くなるので従来型のベクトル計算機がそれほど有効に働かないことが指摘されている．したがって，回路シミュレーションを高速化するためにはベクトル型よりむしろMIMD型の並列マシンによるアプローチが向いていると考えられる[1]．

MIMD型の並列マシンで回路シミュレーションを並列化した例としては緩和法が有名である[2]．今まで最も速度向上を得た例としては緩和法の一種である反復法を用いて，NOT回路を50個直列に結合したような理想的な回路を対象として，1台のときの20倍程度の速度向上を得たものが報告されている[3]．しかしながら緩和法には以下のような問題点が存在する．

（1） 大きなフィードバックを含むような回路あるいはネットに接続する素子間の相互作用が強い回路では反復回数が多くなり，精度や収束性が保証されない．

（2） 対象として扱えるテクノロジがMOSのディジタル回路に制限されており，バイポーラLSIやBICMOS LSIに適用できない．

（3） 今後アナログASICが普及するようになれば，大規模アナログ回路のシミュレーションに対応する必要があるが，緩和法では対応できない．

そこで，筆者らはモジュール分割法に基づく並列回路シミュレーションのアルゴリズムを提案し，その有効性を4台構成のプロトタイプシステムで確認した[4]．モジュール分割法は，並列処理に適しているうえ，精度，収束性が従来の逐次型のシミュレータと同様に保証される．プロトタイプシステムでの評価の結果，提案しているアルゴリズム及び方式は数十台規模の並列マシンにおいても有効であることが確認できた

† Cenju: A Multiprocessor System for Modular Circuit Simulation by TOSHIYUKI NAKATA (C & C Systems Research Laboratories, NEC Corporation), NORIO TANABE (VLSI CAD Engineering Division, NEC Corporation), NOBUKI KAJIHARA, SATOSHI MATSUSHITA (C & C Systems Research Laboratories, NEC Corporation), HIROMI ONOZUKA (VLSI CAD Engineering Division, NEC Corporation), YOSHIHIRO ASANO (NEC Scientific Information System Development Ltd.) and NOBUHIKO KOIKE (C & C Systems research Laboratories, NEC Corporation).

†† 日本電気(株) C & C システム研究所
††† 日本電気(株)超 LSICAD 技術本部
†††† 日本電気技術情報システム開発(株)ソフトウェア事業部

ため，実際に 64 台構成のマルチプロセッサシステム Cenju を開発した．64 台のプロセッサを用いて 6,974 トランジスタからなるダイナミック RAM の制御回路のシミュレーションを実行したところ，1 台のときの 15.8 倍程度の速度向上を達成した．さらに隘路となる逐次部分を並列化することにより 1 台のときの 27.8 倍程度の速度向上を達成する見とおしを得た．

本稿では Cenju のマシンアーキテクチャ，並列アルゴリズム及び評価結果について報告する．

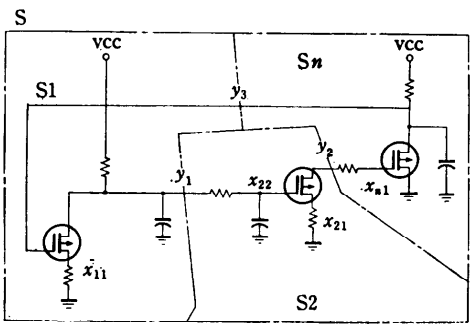
2. モジュール分割並列回路シミュレーションアルゴリズム

図-1 に過渡解析における従来の直接解法とモジュール分割並列解法の違いを図式的に示し，**図-2** にモジュール分割法の詳細アルゴリズムを示す．従来法は一つの大規模・非線形・ランダム・スパースな常微分方程式を解く．一方，本方式ではシミュレーション対象となる回路を n 個の部分回路に分割し，解析処理を

①複数の非線形で小規模な部分回路の常微分方程式群の疑似解の求解（並列実行フェーズ 1），

従来方式：

$S(x_{11}, x_{21}, x_{n1}, \dot{x}_{11}, \dot{x}_{21}, \dot{x}_{n1}, y_1, y_2, y_3, \dot{y}_1, \dot{y}_2, \dot{y}_3)=0$



本方式：

ステップ 1　（部分回路の内部計算）

$S1(x_{11}, \dot{x}_{11}, y_1, y_2, \dot{y}_1, \dot{y}_2, t)=0$

$S2(x_{21}, x_{22}, \dot{x}_{21}, \dot{x}_{22}, y_1, y_2, \dot{y}_1, \dot{y}_2, t)=0$

$S3(x_{n1}, \dot{x}_{n1}, y_2, y_3, \dot{y}_2, \dot{y}_3, t)=0$

ステップ 2　（部分回路間の共通ネットワークの計算）

$N(y_1, y_2, y_3)=0$　線形方程式求解

ステップ 3　（部分回路の線形求解）

$N1(x_{11}, y_1, y_2)=0$

$N2(x_{21}, x_{22}, y_1, y_2)=0$

$N3(x_{n1}, y_2, y_3)=0$

図-1　直接解法とモジュール分割並列解法の違い
Fig. 1　Overview of parallel modular circuit simulation.

②接続ネットワークに相当する境界変数に関する線形方程式の求解（単一実行フェーズ），

③線形計算による，境界変数の解を用いた部分回路ごとの更新解の求解（並列実行フェーズ 2），

の 3 段階を繰り返すことにより，実行する．

処理を 3 段階に分けたことにより，並列処理可能な部分が増し，並列処理の効果が期待できるが，従来法に比べプロセス間同期処理，疑似解の生成，内部変数の再計算，接続ネットワークの計算などがオーバヘッドとして加わる．しかし，大半の処理は全プロセッサで並列に処理されるので並列処理の効果のほうがはる

親プロセスの処理	子プロセスの処理
①$t_{n+1}=t_n+d_t$ とし後退積分の行列係数を求める，近似解を与え，後退積分を行いチャージ電流を計算．	
②独立なソースの値を解ベクトルにロード．	
③親と子プロセス群の同期．	
④結合ネットワークの境界変数を子プロセスに送出．	
	親プロセスより境界変数を入力し解ベクトルにロード．
⑤親と子プロセス群の同期，続いてモデル評価，マトリックスのセットアップ	
⑥	内部変数の求解，ノートン等価値を生成，親に送出
ノートン等価値群をマージし接続ネットワークの境界変数を求解，解を子プロセス群に送出	
収束判定	親プロセスより境界変数を入力し解ベクトルにロード，内部変数の修正，収束判定．
⑦親と子プロセス群の同期	
⑧全回路が収束したか？ 収束しなければ⑤へ， 収束すれば⑨へ．	
⑨親と子プロセス群の同期．	
⑩後退積分式の更新，積分誤差を計算．	
⑪全回路の最大積分誤差を求め，新しい時間刻みと積分次数を決定，子プロセス群に連絡．	
⑫最終時刻に到るまで①からの処理を繰り返す．	

図-2　モジュール分割法の詳細アルゴリズム
Fig. 2　Detailed algorithm of parallel modular circuit simulation.

かに高くなるので問題とはならない.

ただし,②の逐次処理が非並列な処理として残り,部分回路の分割数が多くなった場合に無視できなくなる可能性が存在する.

良く知られているアムダールの法則[5]によると,全処理に対して単一プロセッサでしか処理できない部分が p $(0 \leqq p \leqq 1)$ 存在し,全体で n 台のプロセッサが存在する場合1台に対する速度向上は

$$1/(p+(1-p)/n) \qquad (1)$$

で表される.したがってたとえば単一実行処理の部分が全体の5%しかない場合でも,得られる速度向上は,n を無限大にしても高々 20 となってしまう.

本方式で高速化を実現するためには,単一実行フェーズ処理時間の短縮及び並列実行フェーズでのおのおののプロセッサの負荷の均衡化をはかることが重要となる.

3. 並列回路シミュレーションに適したアーキテクチャと Cenju

3.1 並列回路シミュレーションに適したアーキテクチャ

2.でまとめたアルゴリズムを並列システム上で実現する場合,プロセッサ間通信は主に以下の二つの時点で生じる.

①並列実行フェーズ1の終わりで,各部分回路を担当するプロセッサが,ノートン等価値を生成して親のプロセスを担当するプロセッサに送出するとき.

②単一実行フェーズの終わりで,親プロセスを担当するプロセッサが,線形方程式を解いた結果得られた境界解の値を各部分回路を担当するプロセッサに分配するとき.

特に良く用いられる単一レベルのモデルの場合,①の通信は n 台のプロセッサから1台のプロセッサへの書込み,②は1台のプロセッサから n 台のプロセッサへの書込みとなる.

したがって回路シミュレーションに適した並列プロセッサのアーキテクチャを構成する上での指針としては以下の4点があげられる.

a) ハイパキューブのように多数のプロセッサが低いスループットで同時に多数のプロセッサと通信できる形態よりは,たとえ,同時には数少ないプロセッサとしか通信できないにせよバス結合のように高いスループットで結合されている形態のほうが有利である.

b) ①の通信では,n 台のプロセッサからのデータを1台のプロセッサが受けることになる.この場合,メッセージ通信の形態をとると,データを受け取る側のプロセッサが隘路になる可能性が存在する.したがって直接相手のメモリに書き込むことが可能な共有メモリに基づくプロセッサ結合形態のほうが有利である.

c) 一方,商用のバス結合型マルチプロセッサシステムのように,システムバスに共有メモリを配置して各プロセッサがそのメモリにアクセスする形態では,並列実行フェーズ1ならびに並列実行フェーズ2で,共有メモリが隘路となり,各プロセッサでの並列処理を妨げる結果となる.したがって各プロセッサがローカルにメモリを有する形態が望ましい.

d) プロトタイプ[4]で用いたバス結合方式は以上のアーキテクチャに適合するが,1本のバスに結合できるプロセッサ台数は高々 10 数台に制限される.

そこでわれわれは,以下のようなアーキテクチャの選択を行った.

(1) 分散共有メモリ方式の採用

プロセッサ間でデータを効率良く転送するためにメモリ共有機構を設ける.おのおののローカルメモリにはシステムにユニークなアドレスを与え,各プロセッサはローカル及びリモートメモリのすべてをアクセス可能とする.ただし,自ローカルメモリへはグローバルなバスを経由せずローカルなバスを介してアクセス可能とし,グローバルバスのトラフィックの増大を抑制する.

(2) グローバル階層ネットワーク—ローカルバス方式

プロセッサを8から 10 数台ごとにクラスタにまとめ,クラスタ内はグローバルバスで結合し,さらにクラスタ間の結合に新たな結合網を用いる.ただしクラスタ内・クラスタ間の通信はハードウェアならびにシステムソフトウェアで吸収し,シミュレーションプログラムにはシステム全体をユニークなアドレスでアクセス可能にする.

3.2 Cenju のハードウェア

図-3 に Cenju のシステム構成図を示す.

Cenju は,64 台のプロセッサから構成される.各プロセッサは8台ごとに1本のクラスタバスで結合される.これら8本のクラスタバス間は蓄積型多段結合網で結合される.

クラスタ間結合網を設計するにあたっては,共有メ

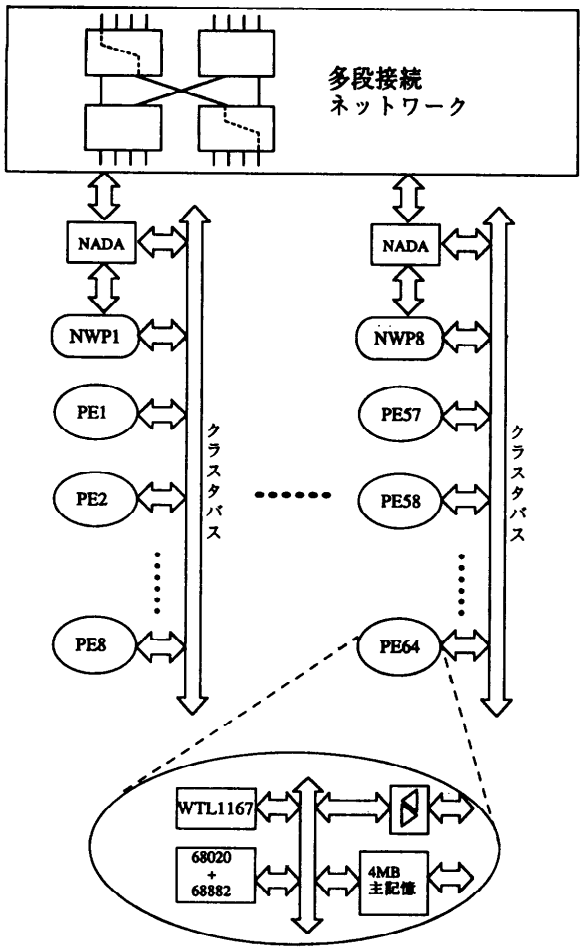


図-3 Cenju のシステム構成図
Fig. 3 System organization of Cenju.

表-1 Cenju のメモリマップ
Table 1 Memory map of Cenju.

PE 番号	グローバルアドレス
0	0x80000000−0x803fffff
1	0x80800000−0x80bfffff
…	……
63	0xbbc00000−0xbbffffff

モリアクセスを実現するために，ハードウェア量と性能とのトレードオフを考慮して，クラスタ間データ書込みをハードウェアで実現する一方，クラスタ間データ読出しをハードウェアとソフトウェアの分担作業で実現することとした．このために，クラスタ間結合網をアクセスするためにネットワークアダプタ (以下 NADA) とネットワークプロセッサ (以下 NWP) を接続した．

プロセッサ (以後 PE) は，MC 68020 (20 MHz)，MC 68882 ならびに 4 MB のメモリから構成される．さらに浮動小数点演算機能を強化するために，Weitek 社の浮動小数点演算器 (WTL 1167) (ピーク 1.6 MFLOPS (倍精度)) を搭載している．

分散共有メモリ方式を実現するために各 PE のメモリを2ポート化した．各 PE は片方のポートを用いておのれのメモリに対して高速にアクセスすることが可能である．このアクセスをローカルメモリアクセスと呼ぶ．ローカルメモリはすべての PE にて，0x00000 番地から 0x3fffff 番地に番地付けされている．

各メモリのもう一方のポートはクラスタバスに接続されている．したがって各 PE はこのポートを用いてクラスタバスを経由して他の PE のメモリにアクセスすることが可能である．このアクセスをグローバルアクセスと呼ぶ．

NWP は浮動小数点演算器を除いては PE と同様の構成をとる．NADA は，ネットワークアクセスを管理するハードウェアであり，NWP に対しメッセージ通信ポートを提供する．さらに NADA は，バスを監視しクラスタ間の書込みアクセスを検知し，相手のクラスタに対するパケットを生成するオートマトンも含む．

グローバルアクセス時には各 PE のメモリに対して表-1 に示すようにユニークな番地が割り付けられている．

同一クラスタ内の他の PE のメモリにアクセスする場合は，バスアクセスのオーバヘッド程度で比較的高速にアクセスすることが可能である．

一方，ある PE が他のクラスタに属する PE のメモリにアクセスしようとした場合はアクセスの種類によって，次のように動作が異なる．

(1) アクセスが書込み操作の場合，NADA はクラスタ外への書込みアクセスを検出すると，そのデータとアドレスをパケット化しネットワークに転送する．Cenju ではクラスタ間の書込みアクセスをハードウェア化とパイプライン化している．この結果，書込みアクセスは1データの格納が終了する前にアクセ

スを終了することが可能なので，アクセス速度は数μ秒要するが，スループット自体はクラスタ内アクセスと遜色ない速度が得られる．

ただし，転送経路の途中のFIFOが溢れそうになった場合には書込み操作を行ったPEの属するNADAが，そのPEに対してバスエラーを引き起こす．この場合は書込み要求を行っていたPEは一定時間待った後，再び書込み要求を出す．

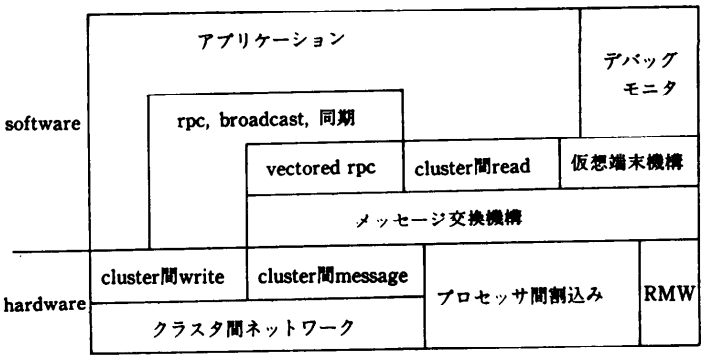


図-4 システムソフトウェアの階層
Fig. 4 Software hierarchy.

(2) アクセスが読出しアクセスの場合，読出し操作を行ったPEの属するクラスタのNADAが，そのPEに対してバスエラーを引き起こす．読出し要求を出したPEのモニタのバスエラーハンドラは，スタックフレームを解析して読出し要求のメッセージを生成し，そのクラスタのNWPを介してアクセスすべきメモリのあるクラスタのNWPへメッセージを送る．要求を受けたNWPは読出し処理を代行し，その結果得られたデータを要求したPEに直接返す．このため，クラスタ間のデータ読出しは遅くなる．

回路シミュレーションではこの点を考慮して，プロセッサ間通信を行う場合は，可能なかぎりProducer-Consumerモデルに基づき，データの生成者がデータの消費者にデータを転送するように工夫している．

プロセッサ間通信のためのハードウェアサポートとしては，上記したメモリアクセスの他に各PEが特定の番地にアクセスすることにより，任意のPEに割込みをかけるPE間割込み機能を装備している．

3.3 カーネル・ソフトウェア[6]

前節で述べたメモリアクセス機能と，PE間割込みを用いて，図-4に示すような階層に沿ってシステムソフトウェアを構築した．階層化することにより，ソフトウェアの信頼性・稼働性の向上を目指すとともに，この階層構造の上に効率の良いカーネルを作成することを，目標とした．

Cenjuではプロセッサ間で生じる非同期事象を統合的に取り扱うために，最下層では，呼び出される手続きのアドレスをシステムのベクタエリアに登録しておき，この番号により遠隔手続きを指定し呼び出すベクタ付きrpcを用いている．また，ベクタ付きrpcに排他性を組み合わせることで，Hoareのモニタ[7]的なアクセスを実現した．Cenjuではこのベクタ付きrpcを用いて，プロセスの起動，プロセスの停止，ユーザレベルの遠隔手続き呼出し(rpc)，プロセッサ間のデータ転送などを実現している．

4. 評 価

4.1 通信の評価

表-2に本システムでメモリ階層に従った平均メモリアクセスタイムを示す．3.でも述べたようにクラスタ間の書込みアクセスは，ハードウェア化されパイプライン化されている．このため，クラスタ間の書込みではデータがメモリに実際に書き込まれるまで5.2μsを要するが，スループットとしてはクラスタ内アクセスと遜色ないアクセス速度800 nsが得られる．ただし，ネットワークからバスへのアクセス変換のためデータ量が多くて，転送路の途中のFIFOが溢れかけたときは前述したように，プロセッサに対してバスエラー割込みが生じる．この場合の平均転送間隔は2.8μsとなる．

これに対して，クラスタ間読出しのオーバヘッドはきわめて大きい．これは，バスエラースタックフレーム生成のオーバヘッドを初め，ハングアップ防止のタイムアウト機構，メッセージのキューイングの排他制御，及びNWPのメッセージハンドリングのオーバヘッドなどによる．

表-2 平均メモリアクセスタイム
Table 2 Average memory access time.

ローカル	クラスタ内グローバル	クラスタ間書込み(アクセス)	クラスタ間書込み(スループット)	クラスタ間読出し
250 nsec	750 nsec	5.2 μsec	800 nsec	240 μsec

ただし，回路シミュレーションでは先ほども述べたように基本的にプロセッサ間通信を write 操作で実現しているため，性能上は問題ない．さらにどうしてもデータの読出しが必要な場合には次に述べる遠隔手続き呼出し (rpc) を用いることでオーバヘッドを軽減することが可能である．

4.2 遠隔呼出しの評価

Cenju ではユーザ用の遠隔呼出し (rpc) として，ブロック型 rpc (手続きの結果が相手の PE から帰ってくるまで実行をブロックするもの)，ならびに非ブロック型 rpc (手続きの結果が相手の PE から帰ってくるのを待たずに実行を続行するもの)，の2種類の rpc を提供している．

1引数の関数をブロック型 rpc で呼んだ場合を実測した結果，1回の呼出しに 640 μs を要するとの値を得た．したがって，3回以上 (N 回とする) のクラスタ間 read を，一つの rpc によって起動される write に変更することで速度向上が図られる．その場合，速度向上比は，

(N 回のクラスタ間読出し)/(1回の rpc+N 回のクラスタ間書込み)

$=(240/(0.8+640/N))$ (小量転送時) から

$(240/(2.8+640/N))$ (大量転送時で接続網が詰まったとき)

の間の値になる．

このような rpc を用いた変更は，前述の性能向上法に比べアルゴリズム的な変更が小さく，有用である．回路シミュレーションの場合は，前述したようにシミュレーションフェーズでは，書込みアクセスしか用いていないが，シミュレーションの前処理のリンクフェーズでやむを得ず読出しを多用していたものを上記の方式に変更している．

4.3 並列度の評価

表-3 に本回路シミュレーションの評価で用いた回路の諸元を示す．いずれもダイナミック RAM の制御回路である．回路1は Cenju で実行するには小さな回路であり，Cenju で実行しても 16 台程度で速度向上は飽和すると予想される．一方，回路2は Cenju の仕様を設定したときに想定した (5,000～10,000 トランジスタ) 程度の大きさの回路である．部分回路数は 191 である．このように部分回路数がプロセッサ数より多い場合は Cenju ではなるべく負荷を均等にするように部分回路をプロセッサに割り当てる．

表-4 に回路1，回路2のシミュレーションをプロセッサ数を変化させて実行させたときの実行時間と速度向上比を示す．(ただし回路2の場合は，容量の関係で8台未満では実行できなかった．この部分ではまだあまり並列性が飽和していないので8台のときの速度向上を1台のときの 6.25 倍として計算している．) なおプロセッサ数を変化させたときも部分回路の個数は変化させていない．

表-3 シミュレーションの評価で用いた回路の規模
Table 3 Size of circuit used for evaluation.

	回路 1	回路 2
節点数	約 1,000	約 4,000
トランジスタ数	1,688	6,974
部分回路数	64	191

表-4 実行時間と速度向上
Table 4 Execution time.

台数	回路 1	回路 2
1	13 hr 44 min (1.00)	—
2	7 hr 36 min (1.81)	—
4	4 hr 07 min (3.33)	—
8	2 hr 19 min (5.93)	10 h 35 min (6.25)
16	1 hr 26 min (9.58)	6 h 39 min (9.94)
32	1 hr 18 min (10.56)	4 h 47 min (13.82)
64	56 min (14.7)	4 h 11 min (15.81)

回路1，回路2ともに速度向上としては 15～16 倍程度の速度向上が得られた．この値は文献3)で報告されている 20 倍には少し劣る．しかしながら文献3)で用いた回路が NOT 回路を直列につないだベンチマーク的な回路であったことを考慮すると，ダイナミック RAM の制御回路という実用的な回路に対して，このような値が得られたことは大きな意義があると思われる．

当初の予想に反して回路1でと回路2であまり速度向上に差がなかった理由としては，

(1) 回路2の部分回路数が 191 個と PE 台数 64 の3倍程度となっており，その分接続ネットワークの行列自体が大きくなったこと．

(2) 回路2の DRAM のモデルの特性のために，接続ネットワークの行列の非零要素率が 20% 程度と予想していたより大きくなったこと

があげられる．このために，回路2では1台で実行する場合，逐次部分の全体に占める率が約4%弱になると推定される．

4.4 逐次部分の並列化

4.4.1 LU 分解の並列化

前節では，部分回路数が増加すると高並列システムになればなるほどアムダールの法則が効いてきて，単一実行フェーズにおける逐次部分のオーバヘッドが大きいことが判明した．逆に逐次部分が少しでも並列化できれば並列化の効果が効いてくる．そこで，本節では 2. で述べたアルゴリズムの逐次部分に関して並列処理が適用できるか否かについて検討する．

逐次部分に相当する単一実行フェーズの処理の大半は境界解に関する線形方程式の求解処理である．本システムではこの処理を LU 分解によって行っている．LU 分解（あるいはガウス消去）の並列化については文献 8) などでも発表されている．しかしながら以下の理由から，通常は実際の応用では使われていない．

（1） 密行列でもピボット行のブロードキャストのオーバヘッドのために高々数倍程度の速度向上しか得られない．

（2） スパース行列の場合は，プロセッサ間の負荷の不均衡のためにますます不利となる．

ところがわれわれの場合は以下の理由から，あえて検討してみることとした．

（1） 部分回路の接続ネットワークの境界解の場合，非零要素率は 20% 程度と比較的高い．（部分回路の内部解では数%程度）

（2） この処理の部分は本来なら逐次処理となってしまう部分であり，たとえ 2 倍でもこの部分を高速化できればたとえば回路 2 の場合，全体の速度向上としては 24 倍になり大きな効果が期待できる．

そこで，まずは妥当性の検討を行う意味から，全体のプログラムとはリンクせずにデータ構造などはまったく回路シミュレーションのものと同じに保ちながら，LU 分解の部分だけを取り出し，評価を行うことにした．

4.4.2 LU 分解の並列化のアルゴリズム

LU 分解の並列化の方法としては，外積法[9] の LU 分解に対して，各行をプロセッサに**図-5** に示すように割り付ける方法を採用した．

説明を簡単にするために，行列を密行列の場合として以下に並列化の手順を示す．

外積法の LU 分解は分解される行列を Aik で表すと**図-6** のようになる[9]．ここで（3）と（4）を逆にして，（4）と（1）を一緒にすると，並列 LU 分解のプログラムは**図-7** に示すもののようになる．

図-7 で（a）は Pivot 行の全 PE への放送であり，並列処理を実現するためのオーバヘッドとなる．（b）において，$Aik=0$ の場合はその行に関しては演算を行う必要はない．

並列効果を妨げる要因としては（a）のオーバヘッド

Row 1	-> PE 0
Row 2	-> PE 1
.....	
Row n	-> PE $n-1$
Row $n+1$	-> PE 0
Row $n+2$	-> PE 1
.....	
Row $2n$	-> PE $n-1$
Row $2n+1$	-> PE 0
.....	
Row m	-> PE k

図-5 並列 LU 分解におけるプロセッサの割当て
Fig. 5 Processor allocation in LU factorization.

```
for k := 1 to n do
  for i := k+1 to n do·····(1)
    Aik := Aik/Akk·····(2)
end.
  for j := k+1 to n do·····(3)
    for i := k+1 to n do·····(4)
      Aij = Aij−Aik*Akj·····(5)
    end.
  end.
end.
```

図-6 LU 分解のアルゴリズム
Fig. 6 Algorithm for LU factorization.

```
for k := 1 to n do
  if (PeContainsPivotRow) {
        Broadcast Akj (j=k to n)…(a)/*ピボット行の放送*/
  }
  Barrier ()/*全 PE で同期*/
  for All i in k+1 to n which concerns this PE do…(b)
    Aik := Aik/Akk
    for All j in k+1 to n do
      Aij = Aij−Aik * Akj
    end.
  end.
  Barrier ()
end.
```

図-7 並列 LU 分解のアルゴリズム
Fig. 7 Algorithm for parallel LU factorization.

ならびに(b)に起因する PE 間の負荷のばらつきによるものが存在する．さらに，回路シミュレーションの他の部分と比べるとこの部分での並列性の粒度は比較的細かいものとなり，同期のオーバヘッドなどが目立つことになる．

なお LU 分解に適したプロセッサ数と全体の回路シミュレーションに適したプロセッサ数は異なることも予想される．そこで，この LU 分解での同期処理は，通常のプロセッサ間同期とは異なるルーチンを用いている．したがって，たとえば全体のシミュレーションは 64 台のプロセッサを用いて行い，LU 分解の部分だけを8台のプロセッサで実行することも可能であるようにした．

4.4.3 性能評価

表-5 の回路1と回路2(1)の欄に 4.3 と同じ回路を対象として，プロセッサ数を1台から 10 台まで変化させたときの1回の LU 分解に要した時間を msec 単位で計測したものを示す．なお比較のために実行時間の横に，プロセッサ数が1台のときの速度を1としたとき，ならびに逐次版の処理速度自体を1としたときの相対速度を示した．

逐次版のときと，プロセッサ数が1台のときの処理時間は同期などのオーバヘッドが0となるため，本来一致するはずである．しかしながら，本プログラムでは，ⅰ）必ずピボット行を一度テンポラリ領域にコピーしていること，ⅱ）一部逐次版で用いている高速化のための構造体を利用できなくなっていること，のために並列版のプログラムのほうが3割ほど遅くなっている．

回路1と回路2の場合を比較すると，次数が高くてかつ非零要素率が高い回路2のほうが速度向上率が高い．いずれにせよプロセッサ数が6～8で速度向上は飽和しており，最大で，1台のときの3.6倍，逐次版に比べて2.9倍程度の速度向上が得られている．プロセッサ数をそれ以上増やすとかえって処理時間が長くなっているが，これはピボット行の放送のオーバヘッドのためであると思われる．

図-7 のプログラムを変更して1度に2行ずつ並列に分解させることが可能である．1度に2行ずつ並列に分解させると，ピボット行のコピー時のオーバヘッドが少し増えるが，その反面，いずれかの行の Aij の要素が非零のときに該プロセッサに対する仕事があるために，プロセッサ間の負荷の不均衡を吸収することが可能となる．さらに同期の回数が半減することにより，同期のオーバヘッドも減ると期待される．

表-5 の回路2(2)の欄に回路2の行列に対して2行ずつ分解させた場合の結果を示す．予想に反して2行ずつ分解したほうが，1行ずつ消去した場合より，処理時間が少しだけ長くなっている．したがって，本質的にピボット行の放送のオーバヘッドによって並列度が決定されると判断できる．

以上の実験により，LU 分解を並列化することにより，従来の逐次部分の処理の大半を占める LU 分解を逐次版と比較して，1.7倍から2.9倍程度高速化できることが判明した．

この値は小さいと思えるかもしれないが，アムダールの法則が示すとおり，その効果は大きい．その点について回路2の場合で考察してみる．

全体の仕事を W として，1台のときに単一実行フェーズに相当する逐次部分の占める割合が4％と仮定する．

64 台のときには逐次部分の処理時間は $0.04W$，並列部分の処理時間は処理のばらつきによるオーバヘッドを 33% と仮

表-5 LU 分解に要する時間
Table 5 Time spent in LU factorization.

	回路 1	回路2(1)	回路2(2)
次数	204	228	同左
非零要素数	7,238(17.3%)	13,755(26.4%)	同左
逐次版	658 1	2,211 1	2,211 1
(PE= 1)	971(1) (0.68)	2,932(1) (0.75)	2,951(1) (0.75)
(PE= 2)	579(1.68) (1.14)	1,615(1.82) (1.36)	1,632(1.81) (1.35)
(PE= 3)	475(2.04) (1.39)	1,180(2.48) (1.87)	1,254(2.35) (1.76)
(PE= 4)	415(2.33) (1.58)	993(2.95) (2.23)	1,030(2.87) (2.15)
(PE= 5)	395(2.45) (1.66)	904(3.24) (2.45)	929(3.18) (2.38)
(PE= 6)	382(2.54) (1.72)	832(3.52) (2.65)	871(3.39) (2.54)
(PE= 7)	389(2.49) (1.69)	781(3.75) (2.83)	862(3.42) (2.56)
(PE= 8)	387(2.51) (1.70)	791(3.70) (2.80)	811(3.64) (2.73)
(PE= 9)	394(2.46) (1.67)	763(3.84) (2.90)	822(3.59) (2.70)
(PE=10)	403(2.40) (1.63)	780(3.75) (2.84)	791(3.73) (2.80)

定すると $(0.96/64)*1.33W=0.02W$ となる．したがって 64 台時の逐次部分の処理時間は全体の 66% を占めることになる．仮に LU 分解の処理に要する時間が逐次部分の 90% を占めるとすると，LU 分解の部分を並列化することにより，逐次部分の処理時間は $(0.04W*0.1+0.04W*0.9/2.9)=0.016W$ になる．すると回路2全体の処理時間は $0.016W+0.02W=0.036W$ となり，このときの速度向上は1台のときの 27.8 倍 (1/0.036) 程度になり，その効果は大きいと言える．

5. おわりに

回路シミュレーションを効率よく実現するモジュール分割アルゴリズムに基づく並列回路シミュレーションマシン Cenju の構成，システムソフトウェアの概要および性能評価結果について述べた．実用規模の 6,974 トランジスタのダイナミック RAM の制御回路で，64 台構成時に，1台のとき 15.8 倍程度の速度向上を得ることができた．さらに速度向上を妨げる逐次部分の LU 分解を並列化することにより，1台のときの 27.8 倍程度の処理速度の向上を得られる見とおしがたった．

今後は実際にこの LU 分解の並列アルゴリズムをシミュレーションプログラムに組み込み，速度向上を評価する必要がある．

また，Cenju は並列回路シミュレーション用に設計したが，アーキテクチャは汎用であり，粗い粒度の並列性を有する他の応用にも適用することが可能である．

われわれは現在，故障シミュレーション[10]などの CAD の応用やモンテカルロシミュレーションなどの数値処理の応用にも Cenju を適用して，評価を開始している．

今後はこれらの評価を元に Cenju のアーキテクチャをより汎用な並列マシンに適したものに発展させていきたいと考えている．

謝辞 本研究の機会を与えていただき，また有益な示唆をいただいた当社半導体事業グループ柳川技師長，超 LSICAD 技(本)晴山部長，黒部部長，北村部長代理，C&C システム研究所石黒所長，森野所長代理，大野主管研究員ならびにハードウェアの製作に多大な貢献をしていただいた C&C システム研究所三野輪一氏に深謝いたします．

参考文献

1) Deutsch, T. et al.: Parallel Computing for VLSI Circuit Simulation, VLSI Systems Design pp. 46-52 (July 1986).
2) Deutsch, J. T. and Newton, A. R.: A Multiprocessor Implementation of Relaxation-based Electrical Circuit Simulation, Proc. 21st DA Conf. pp. 350-357 (1984).
3) Jacob, G. K., Newton, A. R. and Pederson, D. O.: An Empirical Analysis of the Performance of a Multiprocessor-Based Circuit Simulator, Proc. 23rd DA Conf. pp. 588-593 (1986).
4) Nakata, T., Tanabe, N., Onozuka, H., Kurobe, T. and Koike, N.: A Multiprocessor System for Modular Circuit Simulation, Proc. ICCAD 87, pp. 364-367 (1987).
5) Amdahl, G. M.: Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities, Proc. SJCC, Vol. 30 (Apr. 1967).
6) 松下ほか：並列シミュレーションマシン，情報処理学会第 37 回全国大会予稿集，pp. 97-98 (1988).
7) Hoare, C. A. R.: Monitors: An Operating System Structuring Concept, Comm. ACM, Vol. 17, No. 10, pp. 549-557 (Oct. 1974).
8) 星野 力：PAX コンピュータ，オーム社，東京 (1985).
9) 津田孝夫：数値処理プログラミング，岩波講座ソフトウェア科学 9 (1988).
10) 中田ほか：統合 DA 用専用並列マシン MANYO における並列故障シミュレーション，情報処理学会第 36 回全国大会予稿集，5Y-8, pp. 1911-1912 (Mar. 1988).

(平成元年8月30日受付)