計算機アーキテクチャ 137−14
ハイパフォーマンス 80−14
コンピューティング
（2000. 3. 2）

配列間接アクセスを用いないコード生成法による電子回路シミュレーションの高速化とその並列処理

間中邦之† 刑部亮†
前川仁孝†† 笠原博徳†

本稿では、電子回路シミュレーションにおけるランダムスパースマトリクス処理に伴う配列間接アクセスを除去したループフリーコードを生成することにより、WS、PC等の単一プロセッサシステム上で電子回路シミュレーションを高速化すると共にその並列化を行なう手法を提案する。電子回路シミュレータSPICEでは直接法を用いたスパース行列求解の高速化のために、コード生成法により非零要素のみの計算を列挙したループフリーコードを生成している。しかしその際スパース行列処理のための配列間接アクセスが並列化を含め処理高速化を阻害する要因の一つになっている。そこで本論文では配列間接アクセスを用いないループフリーコードを生成する回路シミュレータを作成し、加算器等の5例の回路に対してWS及びPC上で性能評価を行なった。その結果、過渡解析をSPICE3f.4より2倍から65倍高速に行なえることが確かめられた。また、上記の手法に加えて、回路分割により粗粒度タスクを生成し、分割回路を主記憶共有型マルチプロセッサ(SMP)アーキテクチャ用標準APIであるOpenMPを用いて並列化する方法についても述べる。

Evaluation and Parallel Processing of Electronic Circuit Simulation using Code Generation without Array Indirect Access

KUNIYUKI MANAKA,† RYO OSAKABE,† YOSHITAKA MAEKAWA††
and HIRONORI KASAHARA †

This paper proposes fast sequential circuit simulation scheme using loop free code without the array indirect access. It allows us several tens of times speed-up compared with SPICE version 3f.4 on a WS and a PC. Electronic circuit simulator SPICE uses loop free code by the code generation method to speed-up random sparse matrix direct solution. However, array indirect access for sparse matrix handling used in SPICE is one of factors that obstruct speed-up of processing including parallel processing. Therefore,this paper proposes the circuit simulation scheme using loop free code whitout array indirect access which allow us to speed-up transition analysis. The performance evaluation shows the proposed scheme gives us 2 to 65 times speed-up compared with SPICE on a WS and a PC. Moreover,the generated code is processed in parallel on a SMP machine using OpenMP.

1. はじめに

近年の半導体技術の進歩と共にVLSIの集積度は上昇し、回路の設計と検証に多くの時間とコストが必要になっている。中でも電子回路のシミュレーション[1]に要する時間の短縮はチップ開発期間短縮のための重要な課題の一つである。そのために、従来ベクトルスーパーコンピュータあるいはマルチプロセッサなどを用いた高速化も試みられているが[2),3)]、電子回路シミュレーションは科学技術計算の中で最もベクトル化並列化しにくい計算の一つである[4)]。

電子回路シミュレーションの過渡解析では大きく分けて直接法と反復法の2つの方法が使用される。ウェブフォーム、ノンリニアリラキゼーション等[5),6)]の反復法は容易に高い並列性が得られるが、CMOS回路でないと収束性が悪く適用可能回路が限定されるという問題点がある。これに対し、直接法を用いた手法は回路の種類や構造、また回路の動作速度によらずどのような回路も解析することは可能であるが[7)]、大きな記憶領域を必要とし[8),9)]、さらに回路シミュレーションで要求されるランダムスパースな係数行列を持つ連立一次方程式求解の並列処理が非常に難しいために、高速化が困難であるという問題点がある。

従来より直接法の電子回路シミュレータとして広く用いられているSPICE[8)]ではこのシミュレータ時間短縮のためスパース行列の直接法を用いた求解に、コード生成法を用いループフリーコードを生成していた。しかし、スパース行列の格納のために用いている配列間接アクセスが、並列化を含めそれ以上の処理高速化を阻害する要因の一つになっている。

そこで本論文では、この配列間接アクセスを用

† 早稲田大学 理工学部 電気電子情報工学科, Dept. of Electrical, Electronics and Computer Engineering, Waseda Univ.
†† 千葉工業大学 情報工学科, Dept. of Computer Science, Chiba Institute of Technology.

いずループフリーコードを生成することにより、WS(WorkStation) や PC(PersonalComputer) などの単一プロセッサシステム上で、回路シミュレーションを SPICE3f.4 より数十倍以上高速に行なうことを可能としたので、その結果について報告する。

さらに本手法を回路分割[10] を用いて、主記憶共有型マルチプロセッサ (SMP) アーキテクチャ用の標準 API である OpenMP[11] により粗粒度並列処理 [12),13)] する手法を提案し、SMP マシン Ultla Enterprise 3000 上での粗粒度並列処理の性能評価を行う。

本稿では、第 2 章で、直接法を用いた回路シミュレーションの高速化手法について概説する。第 3 章で、単一プロセッサ上での回路シミュレーションの性能評価について述べる。第 4 章で、回路分割を用いた粗粒度並列処理の概要について述べる。第 5 章で SMP マシンを用いて並列性能評価した結果について述べる。

2. 直接法を用いた回路シミュレーションの高速化手法

本章は、直接法を用いた電子回路シミュレーションの求解手法について述べる。

修正節点解析法を用いて回路方程式のモデル化[14] を行うと、電子回路の動特性は非線形連立微分方程式

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, t) = 0, \quad \mathbf{x}(t=0) = \mathbf{x}_0$$

で表せる。ここで、$\mathbf{x}$ は節点電圧などに使われる解ベクトルである。直接法を用いて上式を解く場合には、非線形連立微分方程式を解くためのインプリシット積分、非線形方程式を解くための Newton-Raphson 法、線形方程式の求解が必要となる。

電子回路の非線形連立微分方程式は特性が急に変化するスティフな系となる事から、本研究ではインプリシット積分法として、スティフな系に強い可変ステップ可変次数の BDF (Backward Differential Formula) 法[15] を、SPICE の解析手法である台形積分法、Gear 法をインプリメントしている。直接行列求解には SPICE と同様にクラウト法を用い、ループフリーコードで生成する。本手法の特徴は、ユーザが SPICE 形式の回路リストを入力すると、SPICE の高速化を難しくしていた配列間接アクセスを排除し、さらに定数伝搬などの最適化を行った Fortran 言語で記述されたコードを生成する所である。これによりユーザはこの生成されたコードを使用する任意のマシンのネイティブ逐次 Fortran コンパイラあるいは OpenMP コンパイラでコンパイルし、回路シミュレーションを行うことができる。

以下では、本手法の特徴である配列間接アクセス、定数伝搬、生成されるループフリーコード、そして本手法の回路シミュレーション構成について述べる。

2.1 配列間接アクセス

本節では、配列間接アクセスを用いない手法について述べる。直接法行列求解の回路方程式の係数行列はランダムスパース性が高いため、SPICE では非零要素の保持のため配列間接アクセスを用いて係数行列の演算を行う。このため、配列を参照する時に、メモリアクセス回数が多くなり、これが処理速度向上を阻む原因の一つとなっている。またその並列処理においては配列間接アクセスに関するコンパイル時、データ依存解析が困難なためその十分な並列性を抽出するのが難しかった。

そこで本手法では、ループフリーコード生成の際、全ての変数、定数、一時変数全てに対して配列の間接参照を一切用いずスカラ変数のみでコードを生成することにより、データ記憶領域も最小限の大きさで良く、配列の添字計算、間接アクセスを無くし値を参照する時にメモリを直接参照することで、単一プロセッサ上での処理の高速化、並列化コンパイラによる並列性の解析の容易化を実施する。

2.2 定数伝搬

本手法では求解コードの生成時に過渡解析実行時の計算量を減らすために、定数伝搬を行い、コード内で定数化可能な変数は定数化し、生成されるコード中に Newton-Raphson ループのあるいは最外側の時間発展ループの外側へ移動する。従来のコード生成法では配列内に存在した変数あるいは不変式もそのまま最内側の Newton-Raphson ループで繰り返し計算されていたのに対し、本スカラ変数を用いたループフリーコードの生成では、これらのループ不変式を Newton-Raphson ループだけでなく、最外側の時間発展ループの外側へ移動することが可能となる。またコード生成システム中で、計算可能なステートメント部分についてはループフリーコード生成時に内部で先行評価 (事前計算) を行っている。

さらに係数行列の LU 分解では、計算を左上の要素から右下の要素へと行うので、ある要素の計算を行う際は、その要素の同一列上部及び同一行左部の要素が計算に利用される。そこで定数をできるだけこの部分に割り付けるように、Markowitz 法[16] と併用して係数行列の定数の部分を行列の左上、変数を右下に持っていくようにリオーダリングを施すことで、計算を行うべき要素が定数になる可能性が高くなり、その結果、定数伝搬の対象領域を広げている。

2.3 ループフリーコード

本節では、本電子回路シミュレータが生成するループフリーコードについて述べる。この線形連立方程式の求解コードは、逐次計算において最高速であることが知られているコード生成法[17] を用いる。これは通常の Fortran 言語で記述されたクラウト法による求解より数十倍も高速であることが知られている[2]。生成したループフリーコードの例として、図 1に 3章で評価に使用する SPICE3f.4 付属のサンプル回路 cascaded rtl inverters における係数行列の LU 分解部分を示す。この回路の過渡解析で生成される全体のコードサイズ

は約 900 行である。

```
C     #  LU decomposition
      T63= v206 + v207 + v210
      v34= T63 + S9
      T66= v207 + v210
      v35= T66 * c28
      T68= v209 - v207
      v37= T68 * ( c2 / v34 )
      T69= v37 * v35
      T70= T69 * c28
      T71= v207 + v208
      T72= T70 + c96
      v38= T71 + T72
      v41= S5 * ( c2 / v38 )
      T76= v41 * S5
      T77= T76 * c28
      T79= T77 + c96
      v42= c45 + T79
      v43= c47 * ( c2 / v42 )
      T81= v43 * c46
      T82= c48 - T81
      v44= T82 + c94
      v50= S2 * ( c2 / v44 )
      T89= v236 + v237
      v47= T89 + S12
      v48= v238 - v236
      T93= v236 + v239
      T94= T93 * c28
      v51= T94 * ( c2 / v47 )
      T96= v50 * S2
      T97= v51 * v48
      T98= T97 + T96
      T99= T98 * c28
      T100= v235 + v236 + v239
      T101= T99 + c94
      v52= T100 + T101
```

図 1　生成コード例

図 1において、例えば係数行列の要素 a_{77} はスカラ変数 v34 へと変換する。同様にスカラ変数の頭文字、c を定数、v を変数、T を一時変数、S を定数伝搬した定数として定義している。その後に続く数字は配列に対して 1 対 1 対応になるように順に割り振られる。また、1 つ 1 つのステートメントは算術代入文からなるスカラ計算で記述される。

2.4　回路シミュレーションの構成

本手法の回路シミュレーション構成を図 2に示す。本シミュレータは Fortran コンパイラのプリプロセッサとして実装することにより、生成されたコードは WS や PC などの単一プロセッサマシンのネイティブ Fortran コンパイラや、SMP マシン用の OpenMP コンパイラ、さらにマルチグレイン並列処理を行う自動並列化コンパイラ OSCAR Fortran コンパイラ [12] を用いてコンパイルし、WS や PC、各種並列マシンで実行することが可能である。

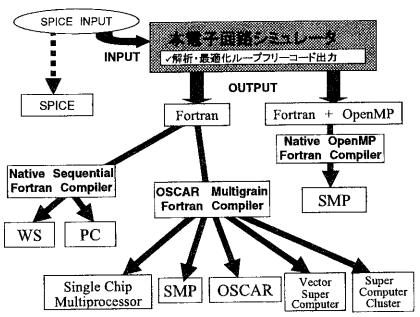


図 2　本回路シミュレーションの構成

3.　回路シミュレーションの性能評価

ここでは、本手法の回路シミュレーションの性能を単一プロセッサの WS 及び PC 上で評価した結果について述べる。

3.1　評価環境

本評価は SPICE3f.4 付属のサンプル回路及び幾つかの小規模回路の過渡解析を例として、単一プロセッサでの過渡解析に要する時間の測定を行い比較した。

評価マシンは WS 及び PC で行う。WS は、Sun Microsystems,Inc.Ultra 60 で、CPU は UltraSPARC-II 360MHz、オンチップ L1Cache は I-Cache・D-Cache 共に 16KB、L2Cache は 4MB、Memory Size は 512MB、OS は Solaris 2.6 である。Fortran コンパイラは Sun の Fortran77 SC4.2 を用い、コンパイラオプションはマシン最適化、浮動小数点演算最適化、最適化数学ライブラリのインライン展開等を行う '-fast' を使用する。PC は、CPU は Pentium-II 266MHz、オンチップ L1Cache は 32KB、L2Cache は 512KB、Memory Size は 128MB、OS は Red Hat Linux 6.1J である。Fortran コンパイラは GNU の Fortran77 を用い、コンパイラオプションは浮動小数点演算などのマシン最適化を行う '-O3' を使用する。

評価の比較対象として代表的な回路シミュレーションである SPICE3f.4 (Univ. of California, Berkeley) を使用する。この比較では、本提案手法で生成された Fortran コードをコンパイルした実行形式ファイルを実行させた時間と、SPICE3f.4 の ICL(Interactive Command Language) を使用し計測した動特性解析時間を用いた。また、インプリシット積分法を共に台形積分法および Gear 法へと変更した場合の性能評価を行う。

3.2　評価結果

最初に WS 上の評価結果を示す。図 3は SPICE の台形積分法と本シミュレータの BDF 法、台形積分法、図 4は SPICE の Gear 法と本シミュレータの Gear 法について、各評価回路での過渡解析に要する時間を評価し、SPICE を 1 とした場合の速度向上率をグラフとして示したものである。グラフ中、左から 8bit 加算器、3 入力 nand 回路、2bit 乗算器、ecl compatible schmitt trigger(schmitt ckt)、cascaded rtl inverters(rtlinv ckt) の評価結果を示す。生成された回路行列サイズは順に 754×754,378×378,114×114, 19×19,12×12 である。

図 3に示すように台形積分法を用いた SPICE と本シミュレータでは、平均で約 25 倍、最大で rtlinv ckt の場合は SPICE で 90ms 要するのに対して本手法で 1.62ms と約 56 倍処理を高速化できることが確認された。また、台形積分法を用いた SPICE に対する BDF 法を用いた本シミュレータを比較した場合 (図 3参照) は、平均で約 30 倍、最大で SPICE の rtlinv ckt に 90ms 要したのに対し、本シミュレータでは 1.72ms と約 52 倍処理を高速化できることが確認された。さらに、図 4に示すように共に Gear 法を用いた場合でも SPICE に対し本手法は平均で約 29 倍、最大で SPICE の場合 rtlinv ckt の 110ms に対して本手法は 1.73ms と約 63 倍の処理高速化が可能であることが確認された。

次に PC 上の評価結果を示す。図 5は SPICE の台形積分法と本シミュレータの BDF 法、台形積分法、

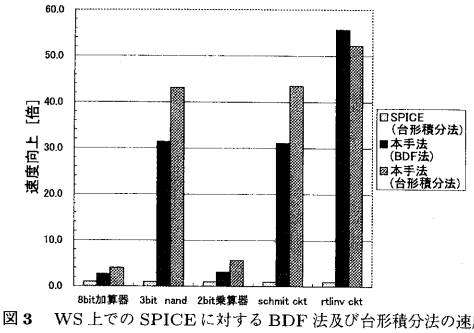


図3 WS上でのSPICEに対するBDF法及び台形積分法の速度向上率

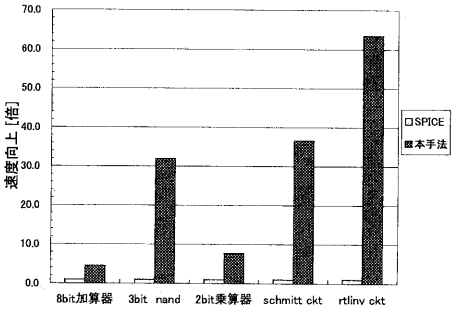


図4 WS上でのSPICEに対するGear法の速度向上率

図6はSPICEのGear法と本シミュレータのGear法について、各評価回路での過渡解析に要する時間を評価し、SPICEを1とした場合の速度向上率をグラフとして示したものである。グラフの見方はWSと同様である。

図5に示すように台形積分法を用いたSPICEと本シミュレータの場合は、平均で約8.5倍の処理高速化が得られ、最大では3入力nand回路でSPICEが3140ms要するのに対し、本シミュレータでは210msと約15倍処理を高速化できることが確認された。また、台形積分法を用いたSPICEに対するBDF法を用いた本シミュレータを比較した場合には、平均で約11.5倍、最大で同じく3入力nand回路の時、SPICEが3140msに対して本手法は90msと約35倍処理を高速化できることが確認された。さらに、図6に示すように共にGear法を用いた場合でもSPICEに対し本シミュレータは平均で約9.6倍、最大で3入力nand回路の時、SPICEが3540msに対して本シミュレータは230msと約15倍処理を高速化できることが確認された。

以上の結果より、本手法により回路シミュレーションが単一プロセッサであるWS及びPC上で大幅に高速化できる事が確認できた。

4. 回路シミュレーションの並列処理手法

前章までの手法により、単一プロセッサ上での高速化を実現した。しかし大規模回路の解析において更な

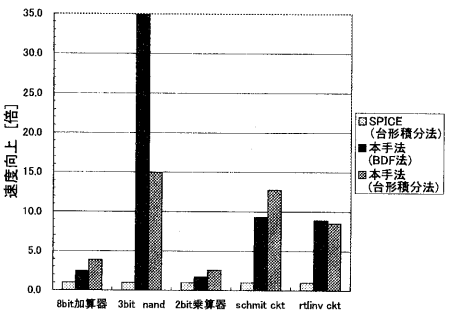


図5 PC上でのSPICEに対するBDF法及び台形積分法の速度向上率

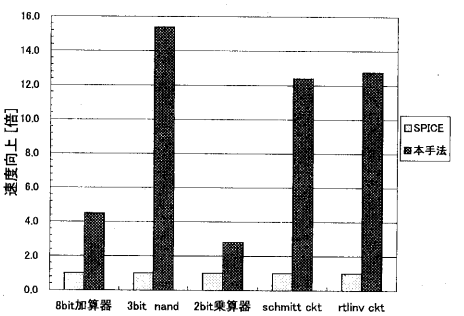


図6 PC上でのSPICEに対するGear法の速度向上率

る高速化のためにマルチプロセッサシステム上での並列処理が有効と考えられる。しかし、現在パソコンレベルサーバからスーパーコンピュータまで広く使用されているSMP用自動並列化コンパイラではループ並列処理のみを行っているため、本稿のようなループフリーコードに対して並列処理性能を伸ばすのは困難である。

そこで本論文は本手法に加えて、回路分割[10]を用いて粗粒度並列処理を行う手法を実現した。

本手法では、分割回路を粗粒度タスクに対応させ、ユーザはSPICE形式の回路リストを入力すると、Open-MPディレクティブを挿入したFortran言語で記述された並列化コードを生成する。これによりユーザはこの生成されたコードを使用する各SMPマシンのOpen-MPに対応したFortranコンパイラでコンパイルする事で、専門技術がなくとも簡単に回路シミュレーションを並列処理できる。

以下では、本手法の特徴である回路の分割手法、マクロタスク生成、生成される並列化コードについて述べる。

4.1 回路の分割手法

本節では、回路の粗粒度並列性を利用するための回路の分割手法について述べる。

従来より、回路分割の代表的な手法の一つであるKernighanとLinの考案したアルゴリズム(KL法)[18]を基に様々な方法が提案されているが、本手法ではその中でも代表的なFudicciaとMattheyesが考案した

アルゴリズム (FM 法)[19] を採用した。このアルゴリズムは KL 法を改良したもので、2 組のブロックの間でセルを交換するのではなくセルを 1 つずつ移動させることにより分割を最適化する。また、移動候補を選ぶ際には cutset の増減 (gain) を計算して gain でソートした Gain Vector と呼ばれる配列を用い、KL 法に比べ計算量を大幅に削減している。セルの数が n とすると、KL 法では候補を選ぶ際の時間計算量が $O(n^2)$ となるのに対し、FM 法では $O(n)$ であるが、収束回数が増えるために $O(n^{1.2})$ から $O(n^{1.5})$ 程度の計算量になる。

4.2 回路分割によるシミュレーション計算手法

本節では回路分割を適用した際の粗粒度並列処理におけるシミュレーション計算手法について述べる。

以下に、本手法で採用した Hajj 等によって提案された節点分割 [20] を行なった場合の計算手法について説明する。回路方程式を作ることで得られた連立一次方程式の係数行列に対して、分割する節点にあたる変数を行列の縁側にリオーダリングする事により図 7 のような縁付ブロック対角行列が得られる。

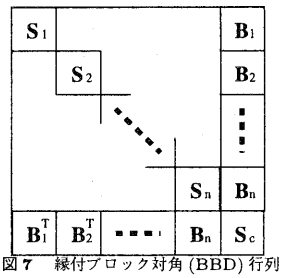
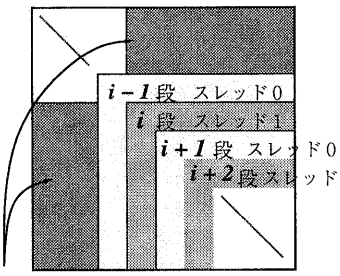


図 7 縁付ブロック対角 (BBD) 行列

このような行列をクラウト法を用いて解くことを考えると、$S_k(k=1,2,\cdots,n)$ は並列に LU 分解でき、また S_k の計算が終了すれば、B_k, $B_k^T(k=1,2,\cdots,n)$ も並列に LU 分解することができる。また、S_c は密行列となり、この部分の LU 分解は、S_k, B_k, B_k^T の LU 分解が全て終了した段階で計算可能になる。本手法では S_c の LU 分解はプロセッサ台数を p とした時、p 段同時分解 [21] を行う。図 8にプロセッサ 2 台を用いた場合の例を示す。通常、i 段の分解計算は $i-1$ 段の分解終了後に行うため、並列効果がほとんどない。そこで、i 段の分解を割り当てられたスレッドは $i-1$ 段の分解が終了していなくても、$i-2$ 段までの要素のみを参照した部分分解計算を行うことができる事より、密行列部分を参照する計算以外は全て並列に計算を行うことができる。

また、前進代入部における計算は、S_k, B_k, B_k^T を一つのグループとする部分行列を考えると、S_c を除いて全て他の部分行列の計算結果を必要としないので、S_c 以外は並列に計算できる。S_c については、自分のブロック以外の全ての解を必要とするので、各部分行列の前進代入の計算が全て終り次第、計算可能となる。



$i-1$ 段の分解計算が終らなくても
i 段の部分的分解計算が参照できる要素
図 8 密行列 S_c の二段同時分解

最後に行なわれる後退代入部の計算は、S_c に属する解の計算が終らないと、各部分行列の計算ができない。そのため、始めに S_c に関する計算を行なった後に、この値を他の部分行列に転送することにより、残りの各部分行列を並列に計算できるようになる。

4.3 並列化コード生成

本節では、本電子回路シミュレータ生成する並列化コードについて述べる。スレッド生成には OpenMP の!$omp parallel sections ディレクティブを用いる。プロセッサ数分のスレッドを最初に生成し、最後まで同じスレッドを用いる事でスレッド生成のオーバヘッドを最小限におさえている。

生成コードは OpenMP コードを加えた Fortran 言語のみで書かれているため、使用 SMP マシンの OpenMP 対応ネイティブ Fortran コンパイラでコンパイルし、並列処理を行う事が可能である。

5. 並列処理手法の性能評価

本章では，本手法の回路シミュレーションの粗粒度並列処理性能を SMP マシン上で評価した結果について述べる。

5.1 評価環境

本評価は 3bit 入力 nand 回路の過渡解析を例として、SMP マシン上でプロセッサ数を 1,2 とした時の過渡解析に要する時間の測定を行い比較した。評価マシンは Ultra Enterprise 3000、2 台の Ultra SPARC(167MHz) と 128MB のメモリを搭載したプロセッサボードが 3 枚実装され、6 台の CPU と 384MB の共有メモリが 1 本のバスで接続された SMP マシンである。Fortran コンパイラは Sun WorkShop 6 Fortran95 Early Access #4 を用い、コンパイラオプションはマシン最適化、OpenMP 対応、並列処理等を行う '-O3 -xtarget=native -xparallel -explicitpar -mp=sun,openmp' を使用する。

5.2 性能結果

現在上記コンパイラを用いて本手法の並列性能の評価を行っている。2 分割を行った回路に対して、本手法の BDF 法でプロセッサ数を変化させた時の過渡解析に要する時間を評価した所、プロセッサ 2 台で約

1.3 倍処理を高速化出来ることが確認された。今回の評価であまり大きな高速化がされない原因としては、評価回路の係数行列サイズが 400x400 と小さいため、OpenMP のスレッド生成や同期のオーバーヘッドが相対的に大きいためと考えられる。

現在の所、Sun OpenMP Early Access #4 コンパイラではより大きな回路に対してのコンパイルが困難であるため、今後コンパイラの改良を待ち、大規模な回路問題に提案手法を適用し、その有効性を評価したいと考えている。

6. おわりに

本稿では、直接法を用いた回路シミュレーションの高速化のため配列間接アクセスを用いないループフリーコードを生成する手法の提案を行なった。この手法により単一プロセッサシステムである WS 及び PC 上で SPICE3f.4 と比較して、シミュレーション時間を 2 倍から 63 倍高速化できる事を確認できた。

また、更なる高速化のため、回路分割による粗粒度並列処理組み合わせる手法の提案を行なった。またその並列化では OpenMP を利用することで多くの SMP マシン上での並列処理を行うことを可能とする。

今後、高速化の手法として、使用する変数を事前にキャッシュに置くプレロードの適用、より大規模な回路に適用するため、分割回路間の粗粒度並列処理に分割回路内のステートメントレベルの近細粒度並列処理を組み合わせたマルチグレイン並列処理[12),23)]をシングルチップマルチプロセッサ[22)]上での並列化する手法を開発していく予定である。

参考文献

1) Sadayappan, P. and Visvanatan, V.: Circuit Simulation Shared Memory Multiprocessors,IEEE Trans. Computers, Vol.C-37 ,No.12,pp.1634-1642 (1988).
2) Fukui, Y., Yoshida, H. and Higono, S.: Supercomputing of Circuit Simulation, Proc. Supercomputing'89, pp.81-85 (1989).
3) Yamamoto, F. and Takahashi, S.: Vectorized LU Decomposition Algorithms for Large-Scale Circuit Simulation, IEEE Trans. Computer Aided Design of Integrated Circuits and Systems, Vol.CAD-4, No.3, pp.232-239 (1985).
4) Lynn Pointer: PERFECT REPORT:1, CSRD Rpt. No.896 (1989).
5) White, J. and Sangiovanni-Vincentelli, A.: RELAX2: A Modified Waveform Relaxation Approach to the Simulation of MOS Digital Circuits, Proc. ISCAS*83, pp.756-759 (1983).
6) Saleh, R. A., Newton, A. R.: Iterated Timing Analysis in SPLICE1, Proc. ICCAD*83, pp.139-140 (1983).
7) 西原明法, 鹿毛哲朗, 奥村万規子, 山村清隆, "ポスト SPICE 回路シミュレータ", 電子情報通信学会誌, Vol82, No.1, pp47-54, 1999
8) Cohen, E.: Program Reference for SPICE2, Electronics Res. Lab., Mem. No.ERL-M592, Univ. of California, Berkeley(1976).
9) Newton, A. R.: The Simulation of Large Scale Integrated Circuits, IEEE Trans. Circuits and Systems, Vol.CAS-26, pp.741-749 (1979).
10) 鹿毛, "回路シミュレーション技術の動向", 電子情報通信学会技術研究報告, VLD90-44(1990-9)
11) L.Dagum and R.Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming", IEEE Computatinal Science and Engineering., Vol.6, Num.9, pp.943-962, Jan 1998
12) 笠原, "並列処理技術", コロナ社 (1991).
13) 鹿毛, "VLSI 回路シミュレーション", 電気学会論文誌 C 分冊, No.6,1987
14) Ho, C. W., Ruehli, A. E. and Brennan, P. A.: The Modified Nodal Approach to Network Analysis, IEEE Trans. Circuits and Systems, Vol.CAS-22, No.6, pp.504-509 (1975).
15) Brayton, B. K., Gustavson F. G. and Hachtel G. D.: A New Efficient Algorithm for Solving Differential Algebraic Systems Using Implicit Backward Differential Formulas, Proc. IEEE, Vol.60, No.1, pp.98-108 (1978).
16) Markowitz, H. M.: The Elimination Form of Inverse and Its Application to Linear Programming, Management Science, Vol. 3, pp.255-269 (1957).
17) Duff, I. S., Erisman, A. M., Reid, J. K.: Direct Method for Sparse Matrices, Oxford Univ. Press (1986).
18) B.W.Kernighan, S.Lin, "An Efficient Heuristic Procedure for Partiotioning Graphs", Bell Syst. Tech. J., Vol.49, pp.291-307, Feb.1970
19) C.M.Fiduccia, R.M.Mattheyses, "A Linear-time Heuristic for Improving Network Partiotions", Proc. 19th Design Automat. Conf., pp.175-181, 1982
20) I.N.Hajj, "Sparcity Considerations in Network Solution by Tearing", IEEE Trans. Circuits and Syst., CAS-27, 5, pp.357-366, May 1980
21) O.Tanabe.: LU Decompoaition on Distributed Memory Machines. IPSJ SIG Notes,pp.55-60 (1995).
22) 早稲田大学, "シングルチップマルチプロセッサ", 平成 11 年 特許出願第 363702 号, 平成 11 年 12 月 22 日.
23) 前川 仁孝, 高井 峰生, 伊藤 泰樹, 西川 健, 笠原 博徳, " スタティックスケジューリングを用いた電子回路シミュレーションの粗粒度/近細粒度階層型並列処理手法", 情報処理学会論文誌, Vol.37, No.10, Oct.1996.