Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

時間依存性のない線形素子による定数伝播を用いた電子回路シミュレータ SPICE3 の高速化

篠塚 研太[†1] 富永 浩文[†1]
中村 あすか[†1] 前川 仁孝[†1]

本稿では，電子回路シミュレータ SPICE3 の連立方程式求解における演算数を削減することで高速化する手法を提案する．SPICE3 は，LU 分解法を用いて，連立方程式を複数回求解することで過渡解析を行う．SPICE3 の過渡解析では，行列要素の値が変化しないため，連立方程式求解時に行列要素が一定の値となる時間依存性のない線形素子がある．値が一定である要素同士の演算は，演算結果を行列の各要素に伝播することで，過渡解析中に複数回実行する必要がない．そこで，提案手法は，時間依存性のない線形素子に対応する行列要素の値を，連立方程式求解処理において行列の各要素に対して伝播することで，演算数を削減する．評価の結果，提案手法である演算数を削減した SPICE3 は，従来の SPICE3 に比べ，シミュレーション処理が約 1.6 倍高速化することを確認した．

Speed up of Electronic Circuit Simulator SPICE3 using Linear Devices without Time Dependence.

Kenta Shinozuka ,[†1] Hirobumi Tominaga ,[†1]
Asuka Nakamura [†1] and Yoshitaka Maekawa[†1]

This paper proposes a speed-up scheme of electronic circuit simulator SPICE3 by reducing number of operands in solving circuit equations. SPICE3 performs transient analysis by solving circuit equations using LU factorization. SPICE3 treats linear devices without time dependence with unchange parameter. Therefore, solving process of circuit equations treats these devices as constant element. Operation results using constant elements are unchange in simulation. So performing constant propagation in matrix elements, operations using constant elements solve only one time in transient analysis. Therefore, the proposed method using constant propagation reduces operations in solving process of circuit equations. The performance evaluation shows the modified SPICE3 gives us about 1.6 times speed-up compared with SPICE3.

1. はじめに

SPICE3(Simulation Program with Integrated Circuit Emphasis 3)[1),2)] は，回路の特性解析を行う電子回路シミュレータとして広く利用されている．近年，半導体技術の進歩による VLSI の集積度の上昇によって解析回路が大規模化するにつれ，SPICE3 によるシミュレーションに多くの時間が必要となっている．SPICE3 によるシミュレーションでは，過渡解析に最も多くの時間がかかるため，過渡解析の高速化が求められている[3),4)]．

SPICE3 による過渡解析は，解析回路を，回路方程式と呼ばれるランダムスパースな係数行列を持つ非線形連立微分方程式に定式化する．非線形連立微分方程式を解くためには，修正節点解析法[5)] を用いて，数値積分法による時間差分化および Newton-Raphson 法による線形化を行い，線形連立方程式に変換し，求解する必要がある．連立方程式の求解は，シミュレーション中に繰り返し行う必要があり[6)]，SPICE3 による過渡解析時間の約 9 割を占めることが報告されている[7)]．このため，SPICE3 の過渡解析の高速化には，連立方程式求解処理の高速化が重要となる．

SPICE3 は，解析可能な回路の構造や種類を限定せずにシミュレーションするために，直接法を用いて連立方程式を求解する[8)]．SPICE3 がシミュレーション中で求解に用いる係数行列は，修正節点解析法により，スタンプする値が Newton-Raphson ループのイタレーションごとに変化する．このため，SPICE3 は，Newton-Raphson ループ中の LU 分解を行う前に係数行列を再計算する．一方，係数行列には，シミュレーション中にスタンプする値が変化しない要素も存在する．値が変化しない要素同士の演算結果は，シミュレーションを通して同じ値になる．このため，時間発展ループの前で値が変化しない要素同士の演算を行い，この値を係数行列にスタンプする．これにより，Newton-Raphson ループ中では値が変化する要素を含む演算のみを実行することで，連立方程式求解時の演算数を削減できる．ただし，演算数を削減するためには，Newton-Raphson ループ中で係数行列の各要素に対し，値が変化する要素，または値が変化しない要素を判定する処理を追加する必要が生じる．

直接法によるスパースな連立方程式求解の高速化手法として，コード生成法の考え方を用いて，求解に必要な演算のみを抽出し，実行する研究がされている[9)–11)]．本手法は，非零要素を含む演算をコードとして生成し，実行することで零要素を含む演算を省き，連立方

†1 千葉工業大学 情報工学科，Department of Computer Science, Chiba Institute of Technology.

Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

程式求解に必要な演算のみを実行する．また，生成したコードは，係数行列の非零要素の位置が同じであれば，連立方程式の求解に繰り返し利用することで，求解時間の削減が期待できる．そこで，本稿では，コード生成法の考え方を用いて，シミュレーション過程において値の変わらない行列要素に対する演算を削減し，SPICE3 の過渡解析を高速化する手法を提案する．提案手法では，シミュレーション過程で値が変化する要素を参照する演算をコードとして記述し，コードを実行するだけで Newton-Raphson ループ中の演算の必要性を判定する処理を行わずに不必要な演算を削減できる．提案手法において，シミュレーション過程に値が変化しない要素を検出するためには修正節点解析法のスタンプ情報を用い，生成するコード数を削減するためにはリオーダリングおよび定数伝播を用いる．

以降の章では，まず，第 2 章で，SPICE3 による過渡解析処理手順について述べる．次に，第 3 章で，定数伝播を用いた連立方程式求解における実行演算数の削減方法について述べる．第 4 章で，提案手法である演算数を削減した SPICE3 のシミュレーション速度に関する性能評価を行うことで提案手法の有効性を示し，第 5 章でまとめる．

2. SPICE3 による過渡解析処理手順

電子回路の過渡解析は，回路方程式と呼ばれる非線形連立微分方程式を解くことで，時刻変化による回路の状態変化をシミュレーションする．一般的な非線形微分方程式の求解手順は，まず，非線形連立微分方程式を，時間積分法を用いて時間的に差分化することで非線形連立方程式に置き換える．次に，非線形連立方程式を，Newton-Raphson 法によって，線形化することで線形連立方程式に変換し，これを繰り返し求解する．

SPICE3 の過渡解析では，修正節点解析法を用いて解析回路の定式化を行う．図 1 に，SPICE3 の過渡解析のシミュレーション処理フローを示す．修正節点解析法は，解析回路中の各素子の接続構造から，あらかじめ時間的な差分化および線形化済みの値を行列要素にスタンプすることで，時間発展ループと Newton-Raphson ループ内で求解する線形連立方程式の係数行列を作成する[5]．このため，同一シミュレーション内で生成される連立方程式は，非零要素が同位置に存在するランダムスパースな係数行列を持つ．ただし，解析回路中に非線形素子が存在する場合，係数行列中で同位置にある要素に対して，異なる値をスタンプする必要が生じる．このため，SPICE3 では，Newton-Raphson ループを繰り返すたびに，修正節点解析法による係数行列の再計算を行う．

SPICE3 の過渡解析では，連立方程式の求解に，LU 分解法の一つである外積ガウス法を用いる．LU 分解法は，連立方程式 $Ax = b$ の係数行列 A を下三角行列 L と上三角行列 U に分解し，前進代入によって $Ly = b$ を，後退代入によって $Ux = y$ を解くことで，連立方程式を求解する．三角行列 L, U は，係数行列 A の各要素 a_{ij} を式 (1) および式 (2) によって更新することで生成する．

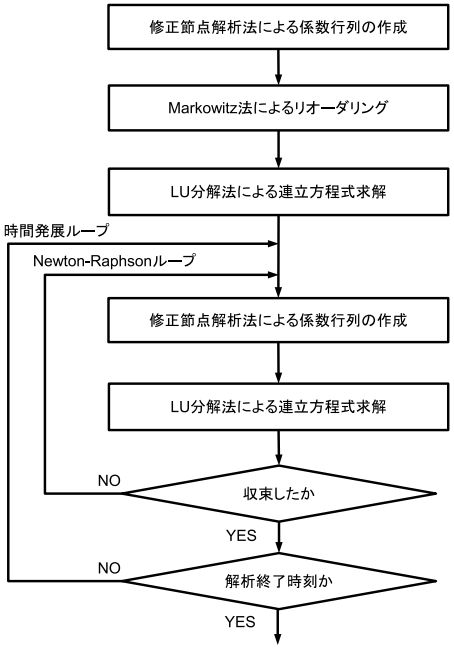

図 1 SPICE3 のシミュレーション処理フロー
Fig. 1 A simulation processing of SPICE3.

$$l_{ij} = (a_{ij} - \sum u_{ik} \times l_{kj})/l_{ii} \quad (i < j) \tag{1}$$

$$u_{ij} = a_{ij} - \sum u_{ik} \times l_{kj} \quad (i \geq j) \tag{2}$$

式 (1)，式 (2) のように，三角行列 L, U の生成における l_{ij}, u_{ij} の導出では，更新要素 a_{ij} の同列，および同行に存在する要素を参照する．図 2 に SPICE3 における三角行列 L, U 生成過程の例を示す．図中の例は，a_1 を基点とした更新を行うところであり，外積ガウス法

Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

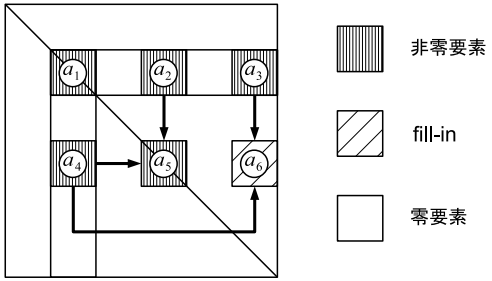

図 2 SPICE3 における LU 分解処理例
Fig. 2 An example of LU factorization of SPICE3.

の参照領域中の非零要素は a_2 と a_3 だけとする．このとき，SPICE3 では，零の含まれる乗算および減算の結果が必ず零になることを利用し，参照要素に零が含まれる演算を行わない．よって，図 2 では，a_4 と a_2 および a_4 と a_3 を参照する積差演算のみを行う．ここで，更新要素 a_6 は，LU 分解前には零要素であるが，LU 分解中に本演算によって非零要素となるため，fill-in が生じる．

fill-in の発生により係数行列内の非零要素数が演算実行前より増加すると，fill-in を参照要素とする演算が発生するため，連立方程式求解における実行演算数が増加する．このような演算数の増加を防ぐために，SPICE3 では，Markowitz 法[12] によるリオーダリングを行うことで fill-in の発生を抑制する．Markowitz 法は，同一行と同一列の非零要素数の積が小さい非零要素を選択し，係数行列の軸位置に移動することで，fill-in 発生を抑えるリオーダリングを行う．ただし，シミュレーション過程で生成される係数行列は同位置に非零要素が存在するため，SPICE3 がリオーダリングを行うのは求解過程で一度だけである．また，LU 分解中に，対角要素を更新する積差演算が行われる場合，対角要素の値が零になる可能性がある．そのため，SPICE3 は，LU 分解中に対角要素が零になると，非零要素を対角に配置するために，部分ピボッティング[6] を行う．

3. 演算数削減による連立方程式求解の高速化手法

SPICE3 は，連立方程式の求解をシミュレーション中に繰り返し求解するため，連立方程式求解処理を高速化する必要がある．非線形連立微分方程式求解における Newton-Raphson ループ中で値が変化しない行列要素は，時間発展ループの外側で一度だけ演算し，この値を係数行列にスタンプすることで，演算を省くことができる．しかし，演算数の削減には，Newton-Raphson ループ中で係数行列の各々の要素に対し，値が変化する要素であるか変化しない要素であるか判定する必要がある．Newton-Raphson ループ中の連立方程式求解で要素の判定処理を追加することにより演算数の削減ができるが，判定処理時間が増加する．そこで，コード生成法[8),13)] の考え方を用いて，値が変化する要素のみを含む演算を，コード生成部でコードとして生成し，コード実行部で生成したコードを繰り返し実行する．図 3 に，Newton-Raphson ループで実行する演算を削減するために，提案手法を用いた SPICE3 のシミュレーション処理フローを示す．本稿では，シミュレーション中に値が変化しない行列要素を定数，値が変化する行列要素を状態変数として定義する．図 3 では，まず，修正節点解析法により係数行列を作成し，より多くの演算を削減するために，定数を考慮したリオーダリングを行う．次に，係数行列の各要素に対して，定数である要素，状態変数である要素の判定を行う．次に，定数同士の演算である場合は演算を実行し，状態変数を含む演算である場合はコード生成部で演算をコードとして生成する．最後に，コード実行部で Newton-Raphson ループ中に生成部で生成したコードを繰り返し実行することで連立方程式の求解を行う．

以下では，Newton-Raphson ループにおける値が変化しない要素を特定するための時間依存性のない線形素子の検出方法について述べる．次に，定数伝播の対象領域を広げるためのリオーダリング方法について述べ，シミュレーション中に値が変化する要素を含む演算のみを実行するための演算コードを用いた連立方程式求解方法について述べる．

3.1 Newton-Raphson ループにおける定数要素の検出

Newton-Raphson ループで演算数を削減するために，提案手法では，修正節点解析法でスタンプする値がシミュレーション過程で値が変化しない要素を検出する必要がある．SPICE3 における修正節点解析法では，固定抵抗器および独立電圧源のように時間依存性のない線形素子において，スタンプする値はシミュレーション過程で変化しない．このため，提案手法では，シミュレーション過程において同じ値を係数行列に書き込むスタンプを定数として扱い，定数のみでスタンプされる要素を定数要素とする．また，時間依存性のある素子からスタンプされる要素の中にも，スタンプされる値がシミュレーション中に変化しない要素があるため，このような要素も定数として扱う．

定数要素の検出するためには，修正節点解析法によるスタンプ時に，各要素にスタンプされる値が定数かどうかを判定する必要がある．このため，定数要素の検出は，修正節点解析法のスタンプ時に行う．また，コード生成部でこの情報を参照するために，係数行列中の各非零要素に定数か変数かを判別するフラグ情報を付加する．

Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

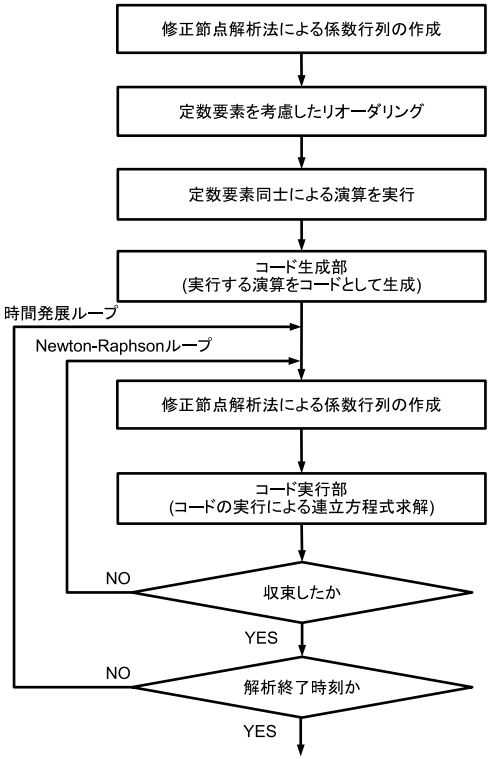

図 3 演算数を削減した SPICE3 のシミュレーション処理フロー
Fig. 3 A simulation processing of SPICE3 by reducing operations.

3.2 定数を考慮したリオーダリング

LU 分解は，式 (1)，式 (2) のように，計算を左上の要素から右下の要素へと行うので，要素の更新の際は，その要素の同一列上部，および同一行左部の要素を参照する．このため，定数伝播の対象領域広範囲化による LU 分解での実行演算数の削減は，定数要素を行列の左上に配置し，変数要素を右下に配置することで，LU 分解で多くの更新要素が定数となるようにリオーダリングを行う．そこで，Markowitz 法によるリオーダリングにおいて，係数行列の各行，各列の非零要素をカウントするのではなく，各行，各列の状態変数要素をカウントし，(行カウント − 1) × (列カウント − 1) が最小となる行と列を選んで係数行列の軸位置に移動させる．この方法では，状態変数要素のみカウントするため，定数要素が多く存在する行，列が行列の軸位置に移動した場合，fill-in が多く発生する可能性がある．しかし，行列の左上に定数要素を集めることで，fill-in により生じた非零要素が定数となる可能性が高くなる．fill-in により生じた要素が定数である場合，この要素を用いた演算は，他の定数と同様に Newton-Raphson ループの外側で実行することで，繰り返し行う必要がない．

また，対角要素が変数である場合，シミュレーション中に値が零になる可能性があるため，LU 分解中に部分ピボッティングを行う必要がある．しかし，対角要素が定数である場合，要素の値が変化しないため，対角要素は常に非零要素となり部分ピボッティングを行う必要がない．そこで，上記のリオーダリング方法を用いる際，対角要素が定数要素となるようにリオーダリングを行う．

3.3 定数伝播による連立方程式求解の演算数削減

連立方程式求解の演算数削減は，定数同士の演算結果を，係数行列の各要素に伝播することで行う．定数伝播を行うために，まず，図 3 のように，リオーダリング処理後に定数要素同士の演算を定数情報を参照して実行する．この演算で得た値は，Newton-Raphson ループ中の係数行列作成処理で係数行列の初期値として用いる．定数要素同士の演算は，係数行列作成の段階で演算結果を行列に反映することで，定数伝播を行うことができる．これにより，定数要素同士の演算はシミュレーション中に再度実行する必要がない．

状態変数要素含む演算は，コード生成部でコードとして生成する．図 4 に，コード生成部での積差演算コードの生成例を示す．式 (1)，式 (2) に示す LU 分解で行う計算は，更新要素を a_{ij} とすると，式 (3)，式 (4) に示す 2 種類からなる．

$$a_{ij} = a_{ij}/a_{ii} \tag{3}$$

$$a_{ij} = a_{ij} - a_{ik} \times a_{kj} \tag{4}$$

式 (3)，式 (4) より，LU 分解で行う演算は，除算と積差演算のみであるため，コードには除算，積差演算を判定するためのオペレータおよび演算に用いる要素を格納する．図 4 では，まず対角要素 a_{ii} から同行に存在する参照要素 a_{ik} と同列に存在する a_{jk} を抽出し，コードを生成する．生成するコードの先頭には，積差演算と識別するためのオペレータを格納する．オペレータの後方には，積差演算に必要な更新要素 a_{ij} と参照要素 a_{ik}，a_{kj} を格納する．除算の場合も同様に，コードの先頭から，オペレータ，更新要素，参照要素の順に格納する．対角要素が状態変数であるとき，値が変化することで対角が零となる可能性がある．そこで，対角要素が状態変数の場合，GESP[14] を用いて LU 分解を行う．GESP は，対角要素が状態変数なら実行し，定数なら実行する必要がないため，コード生成部において，対

Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

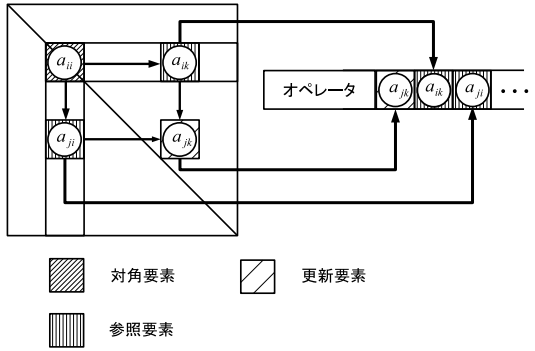

図 4 コードの生成例
Fig. 4 An example of code generation.

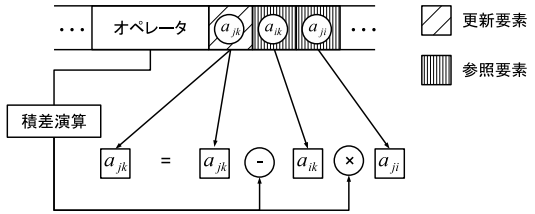

図 5 コードの実行例
Fig. 5 An example of code execution.

角要素が状態変数であるときのみ GESP に関する演算をコードとして生成する．

コード実行部では，生成したコードを Newton-Raphson ループ中で実行する．図 5 に，コード実行部での積差演算コードの実行例を示す．コードの実行は，図 5 に示すように，まずコードの先頭からオペレータを抽出することで，演算子を特定する．次に，コードから演算に用いる要素を抽出し，オペレータに対応した演算子を用いて演算を行う．この処理をコード実行部で繰り返し行うことで，連立方程式を求解する．

4. 演算数削減による SPICE3 高速化手法の性能評価

提案する手法の有効性を示すために，従来の SPICE3 と演算数を削減した SPICE3 の過渡解析時間を比較する．本評価で過渡解析を行う回路は，NMOS 型トランジスタおよび PMOS 型トランジスタを含み，SPICE3 における係数行列サイズが，それぞれ 179×179, 236×236, 592×592, 650×650 となる．評価に用いたマシンは，CPU が Intel Core 2 Duo E6600(2.40GHz)，メモリが 2GB である．また，コンパイラは GCC Version 4.6.1 を用い，最適化オプションは-O2 を使用する．評価は，従来の SPICE3，提案手法である演算数を削減した SPICE3 について行う．また，リオーダリングとして Markowitz 法を用いた提案手法，定数考慮は定数要素を考慮したリオーダリングを行う提案手法について評価する．

表 1 過渡解析時間
Table 1 A processing time of transient analysis.

行列サイズ	SPICE3[s]	演算数削減手法 [s]	定数を考慮したリオーダリング [s]	高速化率 [$times$]
179×179	32.15	25.33	19.96	1.61
236×236	118.37	114.58	110.41	1.07
592×592	14.578	12.347	10.419	1.39
650×650	9.85	9.43	9.43	1.04

表 1 に，従来の SPICE3 と演算数を削減した SPICE3 による各評価回路の過渡解析に要する処理時間を示す．ただし，高速化率は式 (5) により導出する．

$$\text{高速化率} = \frac{\text{従来の SPICE3 での処理時間}}{\text{演算数を削減した SPICE3 での処理時間}} \tag{5}$$

表 1 より，演算数を削減した SPICE3 は，全ての評価回路において，従来の SPICE3 に比べて過渡解析時間が短く，最大約 1.6 倍高速化できることが確認できた．これは，179×179 の評価回路は，独立電圧源が多く含まれているため，係数行列内に定数が多く存在するので，他の評価回路に比べて多くの演算を削減できたと考えられる．また，演算数を削減した SPICE3 で Markowitz 法によるリオーダリングを行う場合に比べ，定数を考慮したリオーダリングを行う方が全ての評価回路で処理時間が短いことがわかる．以上のことから，定数同士の演算数を考慮したリオーダリングは，提案手法による過渡解析高速化への影響が大きいといえる．

次に，定数を考慮したリオーダリングによる高速化への影響を調べるために，各評価回路において，演算数を削減した SPICE3 で Markowitz 法を用いた場合と定数を考慮したリオーダリングを用いた場合での演算中に発生した fill-in の数を比較する．また，提案手法による高速化の要因を調べるために，各評価回路において，従来の SPICE3 による連立方程式求解の演算数と演算数を削減した SPICE3 による連立方程式求解の演算数を計測する．表 2 に，各リオーダリング手法を用いた際に発生した fill-in の数，表 3 に，各評価回路の連立方程式求解時に実行した除算数と積差演算数の和を示す．ただし，表 3 では，式 (6) を

Vol.2011-ARC-197 No.3
Vol.2011-HPC-132 No.3
2011/11/28

表 2 LU 分解で発生した fill-in 数
Table 2 Number of fill-ins in LU factorization.

行列サイズ	Markowitz	定数を考慮したリオーダリング	増加した fill-in
179 × 179	102	120	18
236 × 236	376	405	29
592 × 592	380	406	26
650 × 650	208	208	0

表 3 連立方程式求解の実行演算数
Table 3 Number of operands in solving circuit equations.

行列サイズ	SPICE3	演算数削減手法	定数を考慮したリオーダリング	演算削減率 [%]
179 × 179	2560	2408	2348	8.29
236 × 236	8624	8531	8484	1.63
592 × 592	7944	7832	7702	8.69
650 × 650	6866	6864	6862	0.06

用いて，従来の SPICE3 に対する演算数を削減した SPICE3 の演算削減率を算出する．

$$\text{演算削減率} = \left(1 - \frac{\text{演算数を削減した SPICE3 の演算数}}{\text{従来の SPICE3 での演算数}}\right) \times 100 \qquad (6)$$

表 2 より，全ての評価回路において，定数を考慮したリオーダリングは Markowitz 法と比べて fill-in の数が増加した．これは，定数を考慮したリオーダリングでは，係数行列内で状態変数の数が最小の行，列を軸位置に移動することで，定数による演算時に fill-in の数が多く発生したためであると考えられる．しかし，表 3 より，定数を考慮したリオーダリングは，Markowitz 法と比べて演算数を多く削減するため，リオーダリングによって fill-in の数が増加しても，演算数が多く削減できるため，処理時間が短縮することが確認できた．

5. おわりに

本稿では，Newton-Raphson ループ中で実行する演算数を削減するために，時間依存性のない線形素子に対応する行列要素を Newton-Raphson ループ中で定数として扱い，定数伝播を行うことで SPICE3 の過渡解析を高速化する手法を提案した．提案する手法の有効性を確認するために評価を行った結果，演算数を削減した SPICE3 は，従来の SPICE3 に比べて最大約 1.6 倍高速にシミュレーション可能なことを確認した．また，提案手法は，定数が多い回路ほど効果的に演算数を削減し，SPICE3 を高速化することが確認できた．

参 考 文 献

1) Thomas, L.: THE SPICE3 IMPLEMENTATION GUIDE, Electronics Res. Lab., Mem. No.UCB/ERL M89/44, Univ. of California, Berkeley (1989).
2) 三浦道子，名野隆夫，盛健次：回路シミュレーション技術と MOSFET モデリング，リアライズ理工センター (2003).
3) 西原明法, 鹿毛哲郎, 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, Vol.182, No.1, pp.47–54 (1999).
4) Fukui, Y. and Yoshida, H. and Higono, S.: Supercomputing of Circuits Simulation, *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, pp.81–85 (1989).
5) Ho, C. W. and Ruehli, A. E. and Brennan, P.A.: The Modified Nodal Approach to Network Analysis, *IEEE Transaction on Circuits and Systems*, Vol.22, No.6, pp. 504–509 (1975).
6) 浅井秀樹, 渡辺貴之 : 電子回路シミュレーション技法, 科学技術出版 (2002).
7) Kapre, N. and Dehon, A.: Parallelizing Sparse Matrix Solve for SPICE Circuit Simulation using FPGAs, *IEEE International Conference on Fiele Programmable Technology* (2009).
8) Duff, I. S. and Erisman, A. M. and Reid, J. K.: Direct Method for Sparse Matrices, *Oxford Univ. Press* (1986).
9) 黒田亮，木村啓二，笠原博徳：配列間接アクセスを用いないコード生成法を用いた電子回路シミュレーション手法の性能評価，情報処理学会研究報告計算機アーキテクチャ, Vol.2005, No.7, 2004-ARC-161, pp.1–6 (2005).
10) 間中邦之，刑部亮，前川仁孝，笠原博徳：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，情報処理学会研究報告計算機アーキテクチャ, Vol.2000, No.23, 1999-ARC-137, pp.77–82 (2000).
11) 根本和宜，佐田宏史，前川仁孝，伊與田光宏，宮崎収兄：命令キャッシュを考慮したコード生成法による方程式求解の高速化手法，情報処理学会研究報告計算機アーキテクチャ, Vol.2004, No.80, 2004-ARC-159, pp.67–72 (2004).
12) Markowitz, H. M.: The Elimination Form of Inverse and Its Application to Linear Programming, *Management Science*, Vol.3, pp.255–269 (1957).
13) Gustavson, F. G. and Liniger, W. and Willoughby, R.: Symbolic Generation of an Optimal Crout Algorithm for Sparce Systems of Linear Equations, *Journal of the ACM*, Vol.17, No.1, pp.87–109 (1970).
14) Xiaoye, S. L. and James, W. D.: Making sparse Gaussian elimination scalable by static pivoting, *Proceedings of the 1998 ACM/IEEE conference on Supercomputing* (1998).