並列分散処理及び多並列計算を用いた大規模線形回路網の高速過渡解析に関する研究

静岡大学　博士論文

並列分散処理及び多並列計算を用いた

大規模線形回路網の高速過渡解析に関する研究

2011 年 6 月

大学院　自然科学系教育部

情報科学専攻

井上　雄太

目 次

第1章　序論

1.1　背景

近年の回路実装技術の進歩により，集積回路やパッケージ，PCBの高密度集積化は著しい．加えて，回路の動作周波数が高速になるにつれて，立ち上り時間と立下り時間が短くなり，回路内の信号波はさまざまな高調波を含む高周波信号となっている．その結果として，チップやパッケージ上では高周波の影響による配線上の信号の遅延や反射，配線間のクロストーク，グランドバウンスなどの予期しない現象が発生し，回路の誤動作を引き起こすことになる．これは，製造した回路で所望の動作を得られ無いことを意味し，対策を講じた回路の再設計，再製造を必要とする．しかしながら，所望の動作を得られるまで何度も試作を繰り返し，検証することは現実的ではない．そのため，回路設計の段階で高周波による影響を考慮した動作検証，対策が必要とされている．この回路の動作検証には，古くから回路シミュレータと電磁界シミュレータが用いられて来ている．回路シミュレータでは，高周波の影響は相互インダクタンスと相互キャパシタンスによって

表され，各要素が強く結合された等価回路網を用いることになる．一般的に，この等価回路網はCADモデルから抽出ツールを用いて作成される．そして，抽出対象として回路の配線をモデル化した場合には，素子の数は百万を超える．回路シミュレータとして標準的に用いられているSPICE[1]系シミュレータは行列演算に基づいており，行列に各要素をスタンプしていくと密結合に近い行列になる．そのため，行列演算に必要とする計算時間が膨大となり，実用的な時間で解析結果を得ることが困難である．一方で，電磁界シミュレータでは，高密度実装に伴う微細構造を再現するため，非常に細かいメッシュによって再現された物理モデルを解析することになり，実際に数十億のメッシュを用いることになる．そのため，代入計算のみで電磁界の解析結果を得るFDTD法[2]を用いても，回路シミュレーションと同様に実用的な時間で解析結果を得ることが困難である．このような背景の下，従来のシミュレータと比べて，高速に解析が可能なシミュレーション技術に基づくシミュレータが求められている．

高速なシミュレーション技術の実現のためには，二種類のアプローチにより高速化が試みられている．一つは従来よりも高速なアルゴリズムを開発する試みであり，もう一つはハードウェアアクセラレータを用いた高速化である．回路の過渡解析に限定した場合では，前者は連立一次方程式の解法の高速化[3, 4, 5]や緩和法に基づく手法[6, 7]，回路縮小法[8]，また，新たな定式化手法[9, 10, 11]に

よる高速化が提案されている．後者においては，行列演算と非線形素子のモデル評価の並列化 [12, 13, 14, 15, 16, 17, 18, 19, 20, 21] ，特定のハードウェアの適用 [22, 23, 24, 25, 26] が提案されている．しかしながら，いずれか一方の高速化には限界があり，両方共のアプローチを取り入れた手法も提案されている．すなわち，新たに開発した手法の並列化である [27, 15, 28].

このようにいくつかの従来よりも高速，かつ，並列化された手法が提案されているが，いずれの場合であっても十分に高速であるとは言いにくい．これは並列化における欠点として，未並列化時よりもアルゴリズムが複雑になることなどのオーバーヘッドの出現，ハードウェアによる制限を受けることが挙げられる．並列化によるオーバーヘッドや制限は，過渡解析では各時間ステップや各反復処理中の更新手順の変化や前処理，通信，同期処理といった逐次処理，それらに必要とする実行時間として現れる．これらは並列化における高速化のボトルネックであり，このようなオーバーヘッドは極めて少なくしなければならない．そのため，オーバーヘッドの発生が少ない，より並列化に適しているアルゴリズムを用いる必要性がある．加えて，既存の環境にある高速なハードウェアへの展開も考える必要がある．これは，画像処理用の演算装置であるGPUが急速に性能を伸ばしており，近年では浮動小数点の演算も可能となった．そのため，GPUを利用することによりCPUで実行した時と比較して数倍から数十倍高速なアプリケーションの

開発が可能となっている．従って，高速な回路シミュレータの開発を行うためには，高速で並列化に適したアルゴリズムに対して，並列計算機やGPUを適用することが必要である．

1.2　並列化による高速回路シミュレーション手法

工学分野では，微分方程式により記述された多くの問題がある．しかしながら，これらのほとんどは解析解を持たず，通常は数値計算によって解かれる．そのため，数値計算の目的は微分方程式を数値的に解くことである．回路の過渡解析の場合には，これは非線形常微分方程式として記述される．回路シミュレータとして標準的に用いられているSPICE系のシミュレータでは，回路素子とそれらの接続情報が記述されたネットリストを基に修正節点解析法 (MNA : Modefied Nodal Analysis) で定式化される [1]．そして，後退オイラー法や台形公式などの陰的数値積分法を用いて非線形代数方程式に離散化し，ニュートン・ラフソン法を用いて線形代数方程式とする．この結果として，各時間ステップやニュートン・ラフソン法の各反復は，ヤコビアン行列の計算を行うことで得られる係数行列 $\mathbf{A}$ を持つ線形代数方程式 $\mathbf{Ax} = \mathbf{b}$ を解く問題へと帰着する．この線形代数方程式は，通常，LU分解法を用いる事で方程式の解を得る．そのため，一連の処理において，ニュートン・ラフソン法での各反復でヤコビアン行列を更新するモデル評価とLU分解法

の二つにシミュレーション中の計算時間のほとんどが集中する．そして，高速な回路シミュレーションを実現するためにこれらの並列化を含む高速化手法が提案されている．

モデル評価ではニュートン・ラフソン法により，各時間ステップ中の各反復でヤコビアン行列の各要素のモデル評価を行う．これは，回路規模が中規模までの間では解析に必要とする計算時間の多くを占める．そのため，テーブルルックアップ [29] や，各要素の計算処理を一変数ごとに独立して更新できることに注目した並列化 [12, 15, 19, 20] が行われている．加えて，モデル評価部分のみを FPGA[25] や GPU[21] によって高速化する手法も提案されている．

線形代数方程式の解法の一つである LU 分解法は，寄生素子を多数含む回路網では最悪の場合 $O(n^3)$ の計算量を必要とし，大規模な回路網の場合には膨大な計算時間を必要とする．しかしながら，SPICE 系シミュレータでは，多くの場合で回路行列は疎行列で表され，疎行列の性質を利用したいくつかの高速化手法を用いることができる．その中の一つが，行列の fill-in の発生を抑える行列構造へのリオーダリングであり，更に，回路分割法による縁付きブロック対角 (BBD: bordered block diagonal) 行列への変形 [5] や，縁付きブロック対角行列を再帰的に行い部分回路行列ごとの潜在性を利用した動的分割技法 [30] が挙げられる．そして，この縁付きブロック対角行列の並列化手法が提案されている [13, 14, 15, 18]．これは，

縁付きブロック対角のLU分解で発生するfill-inがブロック対角行列内に制限され，他のブロック対角行列とは独立して解を得られることに注目している．すなわち，対角成分のみ配置されたブロック対角行列であった場合には，各ブロック対角行列が独立して解析結果を得られることを意味している．

ここで，縁付きブロック対角行列を用いた場合の並列化によるオーバーヘッドについて簡単に述べると，まず，線形代数方程式 $\mathbf{Ax} = \mathbf{b}$ を $\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}$ として解くことになる．ここで，

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{BB_1} & & & & \mathbf{A}_{1c} \\ & \mathbf{A}_{BB_2} & & & \mathbf{A}_{2c} \\ & & \ddots & & \vdots \\ & & & \mathbf{A}_{BB_k} & \mathbf{A}_{kc} \\ \mathbf{A}_{c1} & \mathbf{A}_{c2} & \cdots & \mathbf{A}_{ck} & \mathbf{D} \end{bmatrix}, \hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \\ \vdots \\ \mathbf{x}_{1k} \\ \mathbf{x}_2 \end{bmatrix}, \hat{\mathbf{b}} = \begin{bmatrix} \mathbf{b}_{11} \\ \mathbf{b}_{12} \\ \vdots \\ \mathbf{b}_{1k} \\ \mathbf{b}_2 \end{bmatrix}$$

であり，$\mathbf{A}_{BB_n}(n = 1, 2, \ldots, k)$ は $m_n \times m_n$ の行列，$\mathbf{A}_{nc}$，$\mathbf{A}_{cn}$，$(n = 1, 2, \ldots, k)$，$\mathbf{D}$ は要素の接続関係を示す行列である．ここで注目するのは，各ブロック対角行列 $\mathbf{A}_{BB_n}$ は独立して解を得られるが，右下の部分回路行列 $\mathbf{D}$ は節点分割を行ったことによる接続情報が含まれる．そのため，全てのブロック対角行列の情報を基に計算処理が行われる．また，この縁付きブロック行列を解くために，Sherman-

Morrison-Woodbury の公式 [31],

$$\left(\mathbf{B}+\mathbf{C}\mathbf{R}^{T}\right)^{-1}=\mathbf{B}^{-1}-\mathbf{B}^{-1}\mathbf{U}\left(\mathbf{I_{cc}}+\mathbf{R}^{T}\mathbf{B}^{-1}\mathbf{U}\right)^{-1}\mathbf{R}^{T}\mathbf{B}^{-1} \tag{1.2.1}$$

を用いる．この公式を用いるため，行列 $\hat{\mathbf{A}}$ は $\hat{\mathbf{A}}=\mathbf{B}+\mathbf{C}\mathbf{R}^{T}$ と分割される．このとき，

$$\mathbf{B}=\begin{bmatrix}\mathbf{A}_{BB_1} & & & & \\ & \mathbf{A}_{BB_2} & & & \\ & & \ddots & & \\ & & & \mathbf{A}_{BB_k} & \\ \mathbf{A}_{c1} & \mathbf{A}_{c2} & \dots & \mathbf{A}_{ck} & \mathbf{D}\end{bmatrix},\mathbf{C}=\begin{bmatrix}\mathbf{A}_{1c}\\ \mathbf{A}_{2c}\\ \vdots \\ \mathbf{A}_{kc}\\ \mathbf{0}\end{bmatrix},\mathbf{R}^{T}=\begin{bmatrix}\mathbf{0} & \mathbf{I_{cc}}\end{bmatrix}$$

である．公式中には逆行列が存在するが，逆行列を求めずに各ブロック対角行列ごとで解を得ることになる．すなわち，ブロック対角行列とそれ以外の行列の順にLU分解法の前進代入，後退代入を用いることで並列化が可能となる．ここで，各PEに割り当てられる回路行列は，

$$\mathbf{B}_n=\begin{bmatrix}\mathbf{A}_{BB_n} & \mathbf{0}\\ \mathbf{A}_{cn} & \mathbf{D}\end{bmatrix},\mathbf{C}_n=\begin{bmatrix}\mathbf{A}_{nc}\\ \mathbf{0}\end{bmatrix},\mathbf{R}^{T}=\begin{bmatrix}\mathbf{0} & \mathbf{I}_{cc}\end{bmatrix},\hat{\mathbf{x}}_n=\begin{bmatrix}\mathbf{x}_{1n}\\ \mathbf{x}_2\end{bmatrix},\hat{\mathbf{b}}_n=\begin{bmatrix}\mathbf{b}_{1n}\\ \mathbf{b}_2\end{bmatrix},$$
$$(n=1,2,\dots,k)$$

である．すなわち，SPICE系シミュレータの場合には，ブロック対角行列である$\mathbf{A}_{BB_n}$以外の行列の要素と更新処理の複雑化，リオーダリングがオーバーヘッドとして現れることになる．縁付きブロック対角行列を用いた手法では，並列性はブロック対角行列の数によって制限される．そのため，縁付きブロック対角行列を入れ子構造にする入れ子縁付きブロック対角行列 (Nested BBD: nested bordered block diagonal) により，ブロック対角構造を増やす手法 [14, 15, 16]，LU分解自体にも並列化を施した並列BBD行列の解法 [19] が提案されている．しかしながら，これらの手法では劇的に早くなることは無く，並列性の確保とオーバーヘッドのトレードオフとなる．そのため，回路構造によっては並列化の効果を十分に得られない場合もある．

このように従来手法の並列化は，更新処理が複雑になり，同期や通信処理といったオーバーヘッドの発生が問題となる．このようなオーバーヘッドの発生と，並列性の確保が同時に行える行列構造は，行列構造が素子をスタンプした段階で対角行列，またはブロック対角行列となる場合である．これは，対角行列，またはブロック対角行列の場合には，対角成分の任意の要素，またはブロックで行列を分割することができ，分割した部分行列ごとに独立して解を得られる．従って，並列化による高速化では，オーバーヘッドの発生が少ない並列性の高い新たな手法，特に回路行列をスタンプした際に既に対角行列，またはブロック対角行列となる

手法への適用を行うことが求められる．

1.3　本論文の目的と構成

本論文では，従来手法よりも並列化に適している手法に基づき，投入する計算機資源に応じた高速化が得られる回路シミュレーション技術を確立することを目的とする．また，電子回路の設計では，抵抗とキャパシタンス，インダクタンスといった線形素子のみではなく，ダイオードやトランジスタなどの非線形素子を含んだ回路網の解析を行うことが必要となる．しかしながら，本論文では，特に線形素子のみで記述された電源/グランドプレーンの高速化に論点を絞るものとする．

本論文では，2章でクラウドコンピューティングを用いて並列計算機を構築し，高速に回路の過渡解析ができる手法の一つであるLatency Insertin Method (LIM)[32]に並列化を施した並列分散型LIM[33, 34, 35]を用いることで，LIMの並列性の高さとクラウドコンピューティングによる大規模並列計算機の有効性について述べる[36, 37]．3章では，近年，浮動小数点の演算処理を行うことができるようになったGPUを用いたLIMの高速化について述べる[38, 39, 40]．これは，数値計算分野の用途でも大きく注目されるようになったGPUに対して，CPUとは異なるプログラミングモデルに基づく適用を行ったものである．そのため，最適化に対するアプローチもCPUの時とは異なる．そこで，GPUに基づいてLIMを高速化す

るための最適化についても示す．4章で，LIMを寄生素子を含む回路網を解析できるように拡張したアルゴリズムであるブロックLIMの並列化による高速化について述べる[41, 42]．これは，並列分散型LIMと同様に並列化を行うことができ，かつ，回路構造に依存するが利用したPEの数に応じた高速化が可能であることを示す．最後に，5章で本論文の総括を行う．

第2章　クラウドコンピューティングと並列分散型LIMによる高速過渡解析

2.1　概要

従来，並列計算機資源の獲得には，サーバ機器類の購入と設置，保守，電気代が必要であり，維持するだけでも多くの費用が発生する．特に十台以上の計算機をネットワークで接続した場合には，ある一台の故障により並列計算機が利用できない状態も発生する．そのため，研究室などの規模の小さな組織で，並列化したプログラムの実行のために大規模な並列計算機資源を獲得することは悩ましい問題の一つである．

近年，計算機資源の提供方法に大きな変化があり，新たな提供方法はクラウドコンピューティングと呼称されている．クラウドコンピューティングという言葉自体は明確な定義を持たず，言葉どおり雲をつかむような存在である．しかしなが

ら，計算機資源のアウトソーシングと考えた場合には，必要とする計算機資源の規模を任意に変更でき，かつ，導入コストを必要としない点は，普段，研究室内で利用するのは困難な大規模な並列計算機の構築を可能にする．これは，市販されているPCをネットワークで接続して構築するPCクラスタ[43]のことである．また，必要とする金額は利用した計算機資源，つまり，サービスの種類とインスタンス数，利用時間に比例した費用のみであり，利用の仕方によっては従来の計算機資源の導入と比べて非常に効果的であると考えられる．しかしながら，クラウドコンピューティングを計算機資源と捕らえて科学技術計算を行った場合にどのように用いるのが効果的であるかは述べられていないため，並列化に対応したアプリケーションを実行することで検証を行う．

並列化に対応したアプリケーションには，Latency Insertion Method (LIM) [32]に並列分散処理化を施した並列分散型LIMを用いる[33, 34, 35]．LIMは陽的な差分法の一つであるleapfrogアルゴリズムに基づく手法であり，従来手法と比べて数十倍から数百倍高速に回路の過渡解析を行える手法である[32, 33, 34, 44, 45, 46]．また，小規模な並列計算機環境では高い並列性が得られている．しかしながら，大規模な並列計算機環境下での並列化の効果が議論されておらず，その並列化の効果を検証する必要がある．

本章では，クラウドコンピューティングを用いた並列計算機を構築し，その上

で並列分散型LIMによる性能評価を行う[36, 37]．ここでは，Amazon EC2[47]によって提供されているサービスを用いる．また，検証では他の並列計算機システムとの速度向上の倍率を比較することで，クラウドコンピューティングによる並列計算機システムと並列分散型LIMの評価を行う．最終的に，クラウドコンピューティングによって構築した並列計算機が大規模並列計算機資源として効果的であることを示す．

2.2　クラウドコンピューティング

クラウドコンピューティングは近年頻繁に耳にするバズワードの一つであり，明確な定義はない．そのため．本論文中ではPCクラスタを構築するために一時的に利用できる計算機の提供サービスとする．このクラウドコンピューティングでは，インスタンスの種類と利用数，利用時間によって利用料金が決定される．インスタンスは，CPUやメモリ容量などの計算機の構成によっていくつかの種類が提供されており，利用者が任意に選択できる．また，利用するインスタンスの規模，すなわち利用する計算機の台数についても任意の時刻に任意の規模に変更できる．このインスタンスの種類や料金などはサービスの提供業者によって異なるため詳細は述べない．

本章では，クラウドコンピューティングを用いてPCクラスタによる並列計算機

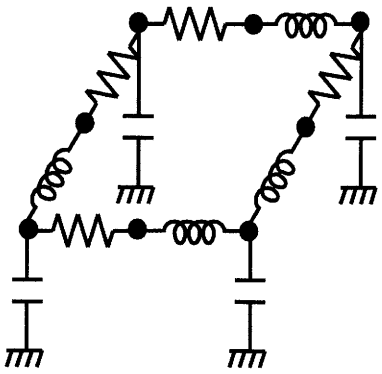
を構築する．この並列計算機は，サービス提供業者が用意しているコマンドを用いることで，直ちに計算機の規模を任意に変更することができる．すなわち，計算機資源を大量に必要とするときには新たなインスタンスを立ち上げることで規模を増加させ，必要としない時には利用しないインスタンスを終了することで規模を縮小させられる．また，クラウドコンピューティングを用いた場合には，利用料金の関係から必要とするときに必要な規模のインスタンスを適切に用いることが求められる．

2.3 Latency Insertion Method (LIM)

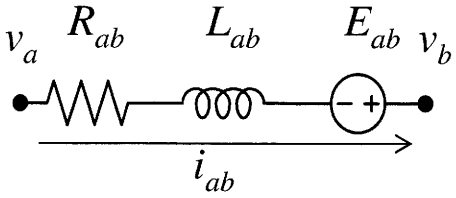
2.3.1 Original LIM

LIMは回路の過渡解析のためのアルゴリズムの一つであり，leapfrogアルゴリズムに基づいた手法である．従来のSPICE系シミュレータは行列演算を用いるため，大規模回路に対して非常に多くの計算時間を必要とする．一方，LIMでは行列演算を必要としないため，従来のSPICE系シミュレータと比較して非常に高速に大規模回路網の過渡解析を行える．

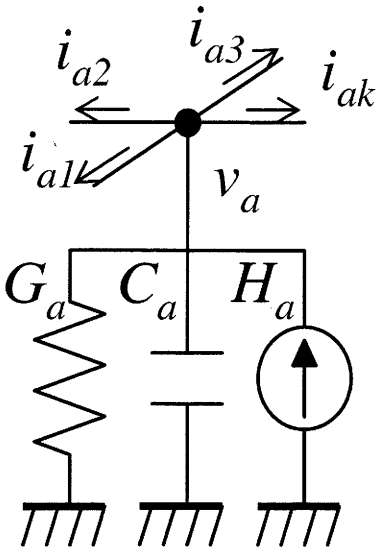
しかしながら．LIMは回路の過渡解析を行うためには特定の回路構造を必要とする．その特定の回路構造を図2.1に示す．LIMの解析対象は図2.1(a)で示す単位



(a) 単位セル.



(b) 枝部の構成要素.



(c) 節点部の構成要素.

図 2.1: LIMの構成要素.

セルが複数接続された構造であり，図2.1(b)と図2.1(c)に示すように枝部と節点部により構成される．枝部は直列に接続された抵抗，インダクタンス，電圧源で構成され，同様に，節点部は並列に接続されたコンダクタンス，接地キャパシタンス，電流源で構成される．そして，枝部と節点部にはそれぞれインダクタンスとキャパシタンスが存在することが解析を行う際の条件である．そのため，インダクタンスとキャパシタンスが存在しない場合には微小な値の素子を挿入することで解析を行うことができるようにする．LIMでは，枝部で電流，節点部で電圧を求めるが，電流と電圧の時間ステップがそれぞれ半時間ステップずつ異なる時間に配置される．そのため，電流と電圧を交互に更新することで過渡解析が行われる．

ここで，任意の節点間に流れる枝の電流をi_{ab}，任意の節点の電圧をv_aとし，時間ステップをn，時間刻み幅をΔtとすると，図2.1(b)とKVL（Kirchhoff's Voltage Law）から式（2.3.1）が，図2.1(c)とKCL（Kirchhoff's Current Law）から式（2.3.2）が得られる．

$$v_a^{n+1/2} - v_b^{n+1/2} = L_{ab}\left(\frac{i_{ab}^{n+1} - i_{ab}^{n}}{\Delta t}\right) + R_{ab} i_{ab}^{n} - E_{ab}^{n+1/2} \tag{2.3.1}$$

$$-\sum_{k=1}^{M_a} i_{ak}^{n} = C_a\left(\frac{v_a^{n+1/2} - v_a^{n-1/2}}{\Delta t}\right) + G_a v_a^{n+1/2} - H_a^{n} \tag{2.3.2}$$

このとき，未知変数が式（2.3.1）ではi_{ab}^{n+1}，式（2.3.2）では$v_a^{n+1/2}$であり，式

(2.3.1)，(2.3.2) を未知変数について整理するとLIMの更新式 (2.3.3)，(2.3.4) が得られる．

$$i_{ab}^{n+1} = \frac{L_{ab} - \Delta t R_{ab}}{L_{ab}} i^n + \frac{\Delta t}{L_{ab}} \left(v_a^{n+1/2} - v_b^{n+1/2} + E_{ab}^{n+1/2} \right) \tag{2.3.3}$$

$$v_a^{n+1/2} = \frac{C_a}{C_a + \Delta t G_a} v_a^{n-1/2} + \frac{\Delta t}{C_a + \Delta t G_a} \left(- \sum_{k=1}^{M_a} i_{ak}^n + H_a^n \right) \tag{2.3.4}$$

ここで，式 (2.3.3)，(2.3.4) の時間刻み幅である Δt は取りうる値に制限があり，最大値は解析対象の回路中にある最小のインダクタンスとキャパシタンスの素子値に依存する．時間刻み幅の最大値の条件式は式 (2.3.5) で与えられる [48].

$$\Delta t_{max} < \sqrt{2} \min_{j=1}^{N_n} \left(\sqrt{\frac{C_j}{N_b^j} \min_{k=1}^{N_b^j} (L_{j,k})} \right) \tag{2.3.5}$$

ここで，N_n は回路中の総節点数，N_b^j は節点 j に接続している枝の総本数，$L_{j,k}$ は節点 j に接続している枝構造に含まれるインダクタンス，C_j は節点 j の接地キャパシタンスである．

式 (2.3.3)，(2.3.4) より，電流，電圧の更新では共に自身の過去の値と，半時間ステップ前の電流または電圧の値を参照する．そのため，全て既知の値を用いて更新処理を行うことになる．そのため，電流，電圧は1変数ごとに独立して更新を行える．また，更新に必要な要素が隣接する要素のみであるため，多数のメ

モリ参照を必要としない．このことは，LIMが並列計算に適していることを示唆している．

2.3.2　並列分散型LIM

LIMは，FDTD法と同様にleapfrogアルゴリズムに基づいており，FDTD法を並列化した場合には利用した計算機資源に比例した速度向上が得られる[49]．そのため，LIMの場合でも同様の速度向上が得られることが期待でき，並列化を行った並列分散型LIMが提案されている[33, 34]．しかしながら，大規模な並列計算機環境での並列化の効果は確認されておらず，大規模な計算機環境での速度向上の検証が求められている．

並列分散型LIMは通常のLIMとは異なり，複数のPEを用いて過渡解析を行う手法である．そのため，通常のLIMで行われる前処理に加えて計算領域の分割処理が加わり，過渡解析の更新処理にオーバヘッドが現れる．これは，更新手順の複雑化と同期，通信処理が加わることを意味する．LIMの解析対象を6×5セルのプリント基板の電源/グランドの等価回路とし，PE数を2，すなわち，二分割した時の解析領域の領域分割を図2.2と図2.3に，過渡解析のフローチャートを図2.4に示す．図2.2と図2.3に示すように，各部分回路は均等な計算領域となるように分割する．計算領域の分割で注意する点として，境界となる部分の電流変数と電圧変数

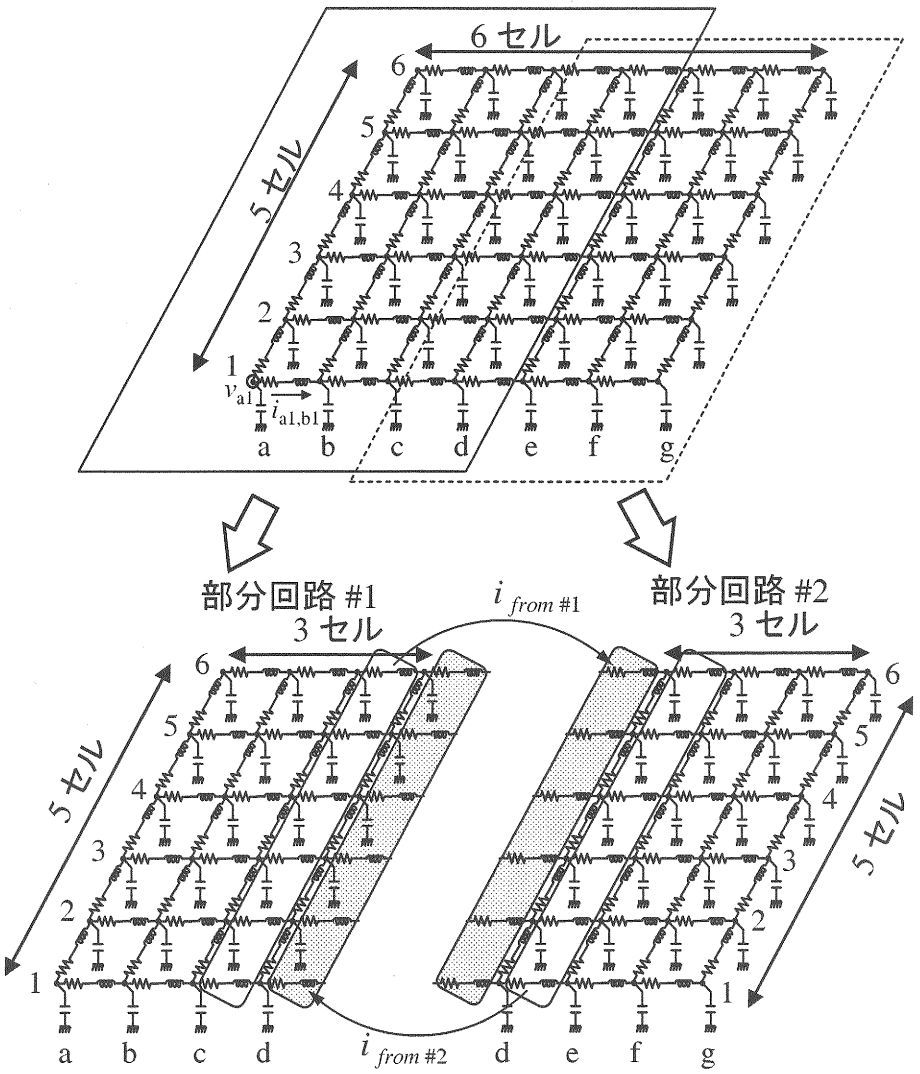


図 2.2: 並列分散型 LIM の領域分割.

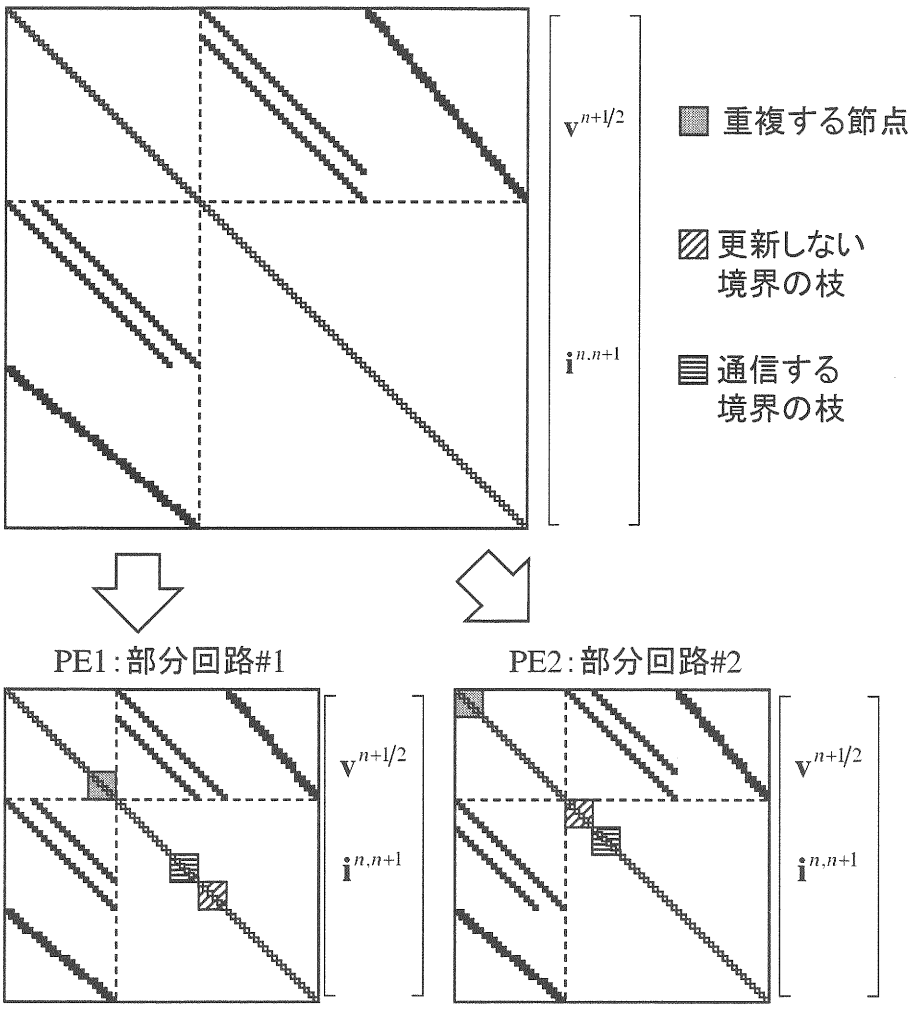


図 2.3: 行列で表した領域分割.

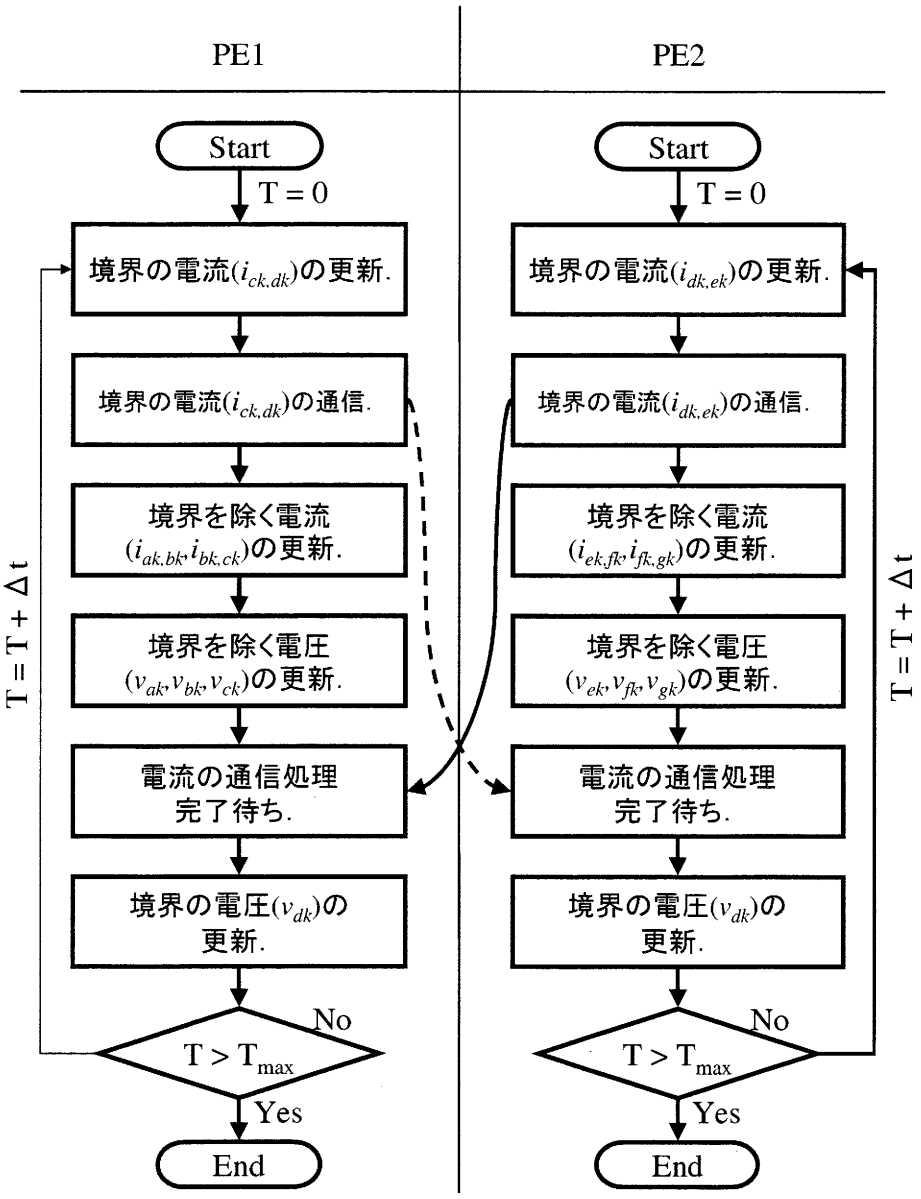


図 2.4: 並列分散型LIMのフローチャート.

を重複して保持することである．ここでは，電流変数$i_{ck,dk}, i_{dk,ek}(k = 1, 2, \cdots, 6)$と電圧変数$v_{dk}(k = 1, 2, \cdots, 6)$が重複する変数である．これらは，通信処理に非同期処理を用いて計算処理と重複させるために用いる[50]．一般的に，並列計算ではプログラム中に存在する逐次処理を最小にしなければ性能を得られない．ネットワークを介した通信処理は，CPUでの計算処理に比べて多くの時間を必要とし，そのままでは多くの時間を必要とする逐次処理として現れる．そのため，通信処理と計算処理を重複することで逐次処理として現れる通信時間を最小にしている．

並列分散型LIMは図2.4に示すように，先ず，境界に位置する電流変数の更新を行う．ここでは，PE1は$i_{ck,dk}$がPE2は$i_{dk,ek}$が対応する．そして，行列では横線で囲まれた部分の係数を利用する変数である．更新した電流変数の値は，非同期関数を用いて各PE間で通信を行う．すなわち，PE1では$i_{dk,ek}$をPE2では$i_{ck,dk}$の計算処理を通信処理に置き換えることになる．これは，斜線で囲まれた部分の係数を利用する要素が計算処理を経ずに更新されることになる．その後，重複して保持されている電流変数と電圧変数以外の更新を行い．最後に，電流変数の通信が終了するのを待った後にv_{dk}の更新を行い1時間ステップの更新が完了する．並列分散型LIMでは，この一連の処理を解析終了時刻まで繰り返して行うことで過渡解析が行われる．

ここで図2.3に示す行列を用いてLIMの係数行列の分割を再考すると，LIMの

アルゴリズムでは各要素の接続関係を示す接続行列が更新時には右辺側に配置され，結果として対角行列を解くことが挙げられる．対角行列のみであるため，任意の要素での行列の分割を実現でき，かつ，任意の数のPEを用いて並列計算を行うことができる．加えて，各PEが担当する領域のみを保持すればよいため，必要なメモリの量に応じた並列計算機環境を提供することによって，単一のPEでは解析できない対象であっても解析を行うことが可能となる．

2.4　計測結果

ここでは，LIMがSPICE系のシミュレータよりも高速であることの確認，クラウドコンピューティングを用いた並列計算機でプログラムを実行した時の効果，そして，並列計算機の規模を大規模化した時の性能について順に検証していく．ここでは，図2.5に示す電源/グランドプレーンの等価回路網を用いる．また，全ての解析で，入力波形に遅延0.2nsec，立ち上がり0.1nsec，立下り0.1nsec，パルス幅1.0nsec，振幅0.05Aの電流を入力し，観測点として右下に位置する節点の電圧を測定する．

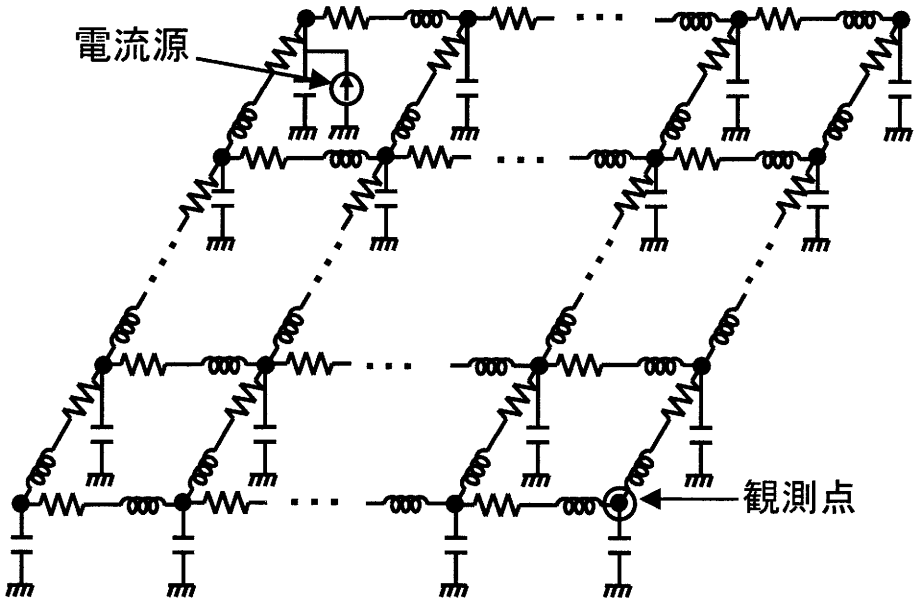


図 2.5: 電源/グランドプレーンの例題回路.

表 2.1: HSPICE と LIM の実行時間の比較.

セル数	実行時間 (sec)	
	HSPICE	LIM
400	4.68	0.39
10,000	935.88	5.78

2.4.1　LIM と SPICE 系シミュレータの比較

LIM と SPICE 系シミュレータとの比較では，代表的な SPICE 系シミュレータの一つである HSPICE と比較することで検証する．まず，LIM と HSPICE で 400 セルと 10,000 セルでモデル化した電源/グランドプレーンの等価回路の解析し，計算時間と出力波形の比較を行う．図 2.6 に 400 セルでモデル化された電源/グランドプレーンの等価回路網を解析したときの出力波形と表 2.1 に HSPICE と LIM の計算時間を示す．この計測では Sparcv9 1GHz を搭載した計算機を用いた．解析結果と計算時間の比較により，LIM は HSPICE と同様の出力波形を得ることができる．また，計算時間は 10,000 セルを解析したときには 160 倍以上高速に解析する事ができ，LIM が非常に効果的な手法であることが分かる．

2.4.2　クラウドコンピューティングと既存の並列計算機の比較

クラウドコンピューティングを用いて構築した並列計算機の有効性を検証するため，既存の並列計算機環境と両方で並列分散型 LIM を実行することで検証する．

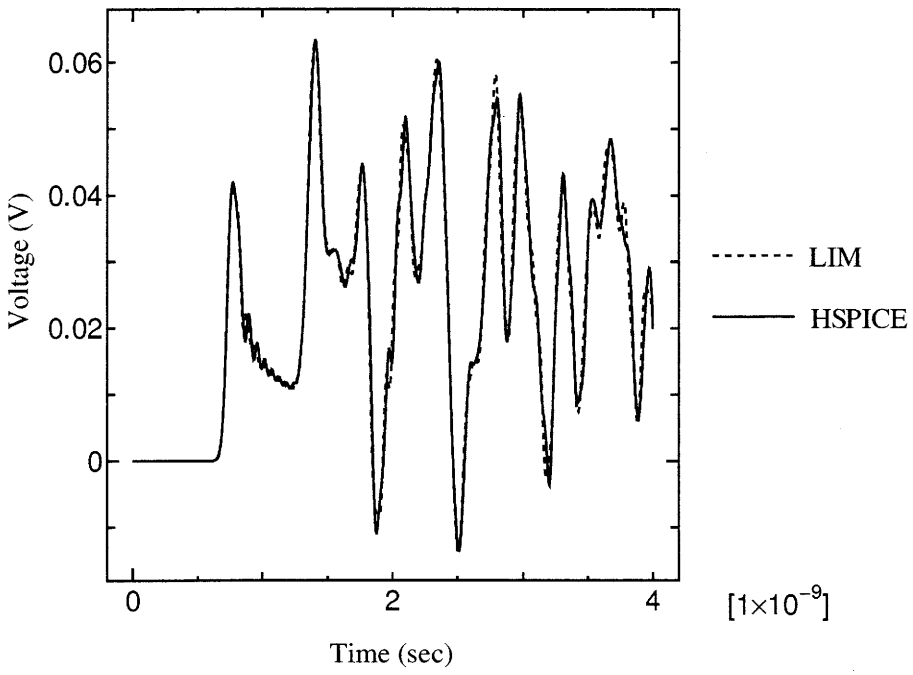


図 2.6: 出力波形.

並列分散型LIMはMPI[51]を用いて並列化を行っている．すなわち，計算機環境によるプログラムの差はない．例題回路として，1,000,000セル，4,000,000セル，9,000,000セルでモデル化した回路網を用い，1000時間ステップまでの実行時間を計測を行った．クラウドコンピューティングは，Amazon EC2[47]のエクストララージインスタンスを2インスタンス用いて分散メモリ型の並列計算機を構築した．エクストララージインスタンスの1インスタンスの構成を表2.2に示す，本来なら，各インスタンス間のネットワーク構成が問題となるが，仮想化されたネットワークであるため単にネットワークで接続された構成であるとだけ述べる．また，分散メモリ型の並列計算機とは，並列計算機システムを構成するそれぞれの計算機が独立したメモリ領域を持っており，異なる計算機同士では互いのメモリ領域を参照できない構成である．ここでは，各インスタンス間でのメモリ参照ができないことを意味している．比較対象とする既存の並列計算機環境は，SGI Altix4700を用いた．Altix4700の構成を表2.3に示す．加えて，クラウドコンピューティングとは異なり，Altix 4700は共有メモリ型の並列計算機である．共有メモリ型の並列計算機とは，分散メモリ型のときとは違い並列計算機全体で一つのメモリ領域を共有している構成である．また，クラウドコンピューティングの場合では，割り当てられたプロセスはコアによって計算が行われるのに対して，Altix 4700ではCPUによって計算が行われる．これは，計算機のアーキテクチャの違いによる．

表 2.2: エクストララージインスタンスの計算機構成.

CPU 周波数	2.3 GHz
CPU 数	2
コア数	8 コア
メモリ	15 GByte

表 2.3: SGI Altix 4700 の計算機構成.

CPU	Itanium 1.6 GHz
CPU 数	16
メモリ	32 GByte
ネットワーク	NUMA

そのため，Altix 4700 は多数の CPU によって構築された並列計算機であるといえる．表 2.4 に CPU 数とコア数の比較について示す． 表 2.7 に，各並列計算機での PE 数を増やしていった速度向上の倍率の比較を示す.

Altix 4700 で実行した場合には，PE の数を増やすにつれて速度が向上していくが，クラウドコンピューティングを用いた場合には PE 数が 8 以上では速度の向上を得ることができないことがわかる．これは，計算機のアーキテクチャの違いによって生じていると考えることができる．すなわち，Altix 4700 では各 CPU がメモリを参照するためのバスを占有できるのに対して，クラウドコンピューティングを利用した場合には，一つの CPU からメモリへのバスを複数のコアで共有している．そのため，各 CPU へ 2 個以上のプロセスが割り振られた場合には，メモリ

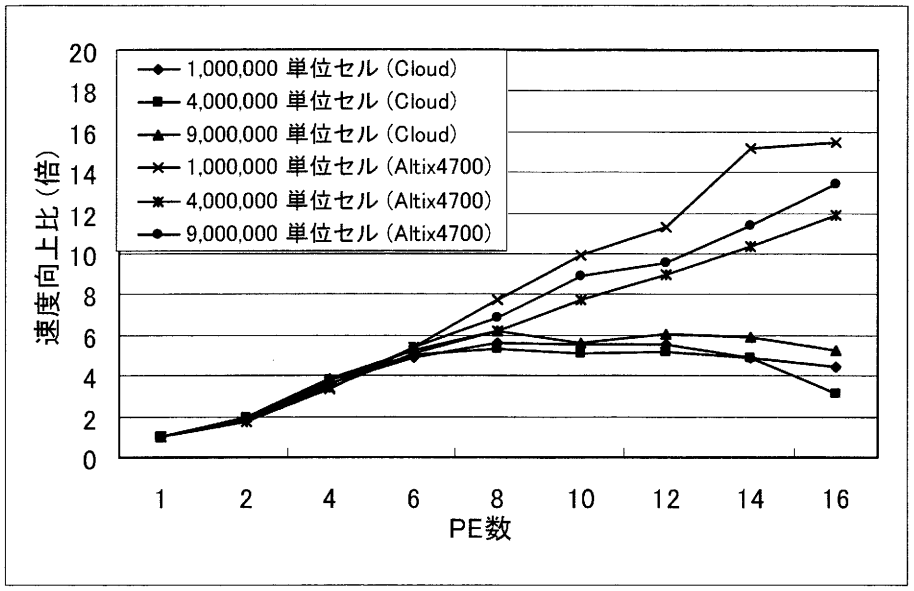


図 2.7: 速度向上の倍率の比較.

表 2.4: CPU数とコア数の比較.

	SGI Altix4700	Cloud Computing System
CPU数	16	4
Core数	-	16

とコアの間のデータ転送が計算に対して転送量が不足しているためであると考えられる．これにより，並列分散型LIMは，ハードウェアによる制限を受けなければ高速に回路の過渡解析を行うことができる手法であると考えられる．

2.4.3　大規模並列計算機での性能検証

次に，クラウドコンピューティングを利用し，大規模並列計算機を構築して検証を行う．ここではエクストララージインスタンスを16インスタンス用いて構築を行った．これは，CPU数では32個，コアについては128個からなる並列計算機である．前節でCPU数の2倍程度のPE数であれば速度の向上が得られることを確認したため，64PEまで順にPEの数を増やして計測を行う．図2.8に実行時間の推移を図2.9に速度向上の倍率を示す．速度向上については，32PE付近までの間は順調に増加し，1PEの時と比べて25.75倍の高速化を得られた．この結果は，非常に高い並列化の効果を得られていることがわかる．速度向上の倍率とプログラム中のどれほどまでを並列化することができたかを示すアムダールの法則を式

(2.4.1) に示す.

$$S = \frac{1}{(1-r) + \frac{r}{p}} \tag{2.4.1}$$

ここで，S は並列計算実行時の速度向上の倍率，r は並列化可能な部分と並列化不可能な部分の比である．式 (2.4.1) より，並列分散型LIMは99%以上の部分を並列化することが可能であることを示している．しかしながら，実行時間が32PEまでは減少し，それ以降は実行時間の減少が飽和した状態になる．すなわち，システムのオーバヘッドが現れていると考えられる．先ほどまでの計測結果により，Altix 4700では16PEまでの間は順調に速度の向上を確認できていた．そのため，クラウドコンピューティングでは32PE以上を利用した場合にはCPUとメモリ間のデータ転送が飽和状態にあると考えられる．そのため，クラウドコンピューティングを用いる場合には，得たい速度向上の倍率によってCPUの数を選択すればよいことが分かる．すなわち，更に速度が欲しい場合には多数のインスタンスを用いて，プログラムを実行すればよい.

2.5　本章の総括

本章では，クラウドコンピューティングを用いてPCクラスタを構築し，並列分散型LIMによる大規模線形回路網の解析を行った．並列分散型LIMは，電源/グランドプレーンの等価回路網を解析するのに適した手法であり，99%以上の部分を

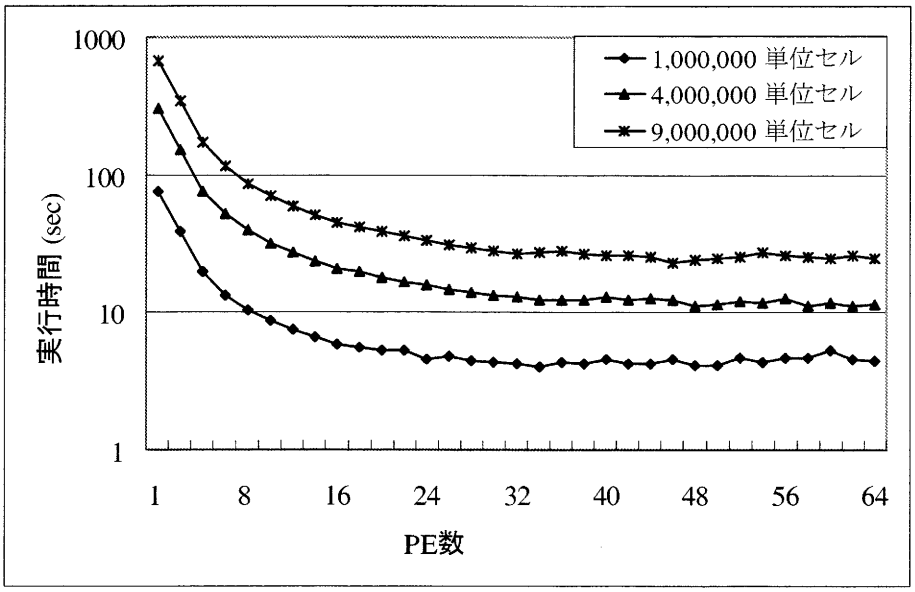


図 2.8: 実行時間.

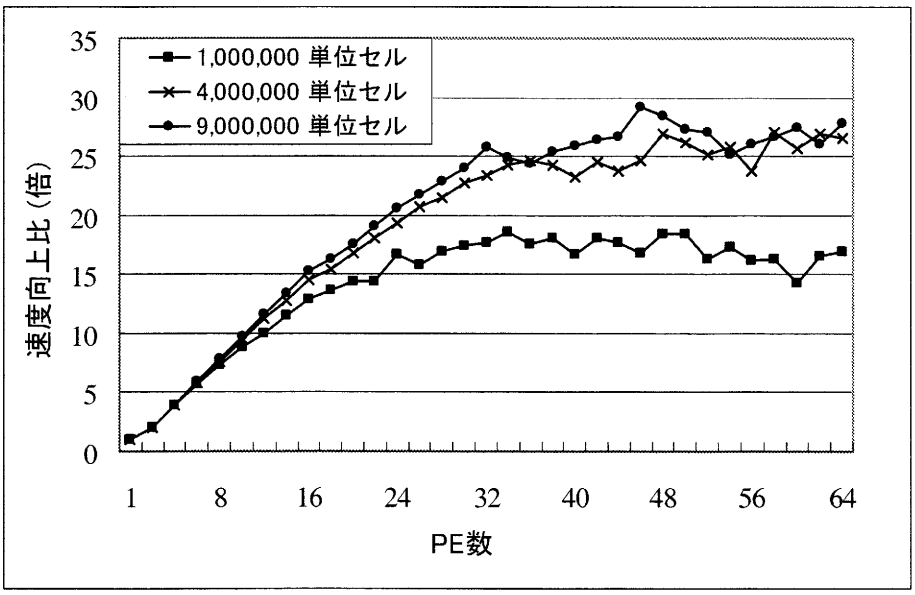


図 2.9: 速度向上の倍率.

並列化できる高い並列性を有していることを示した．クラウドコンピューティングを用いた並列計算機では，並列分散型LIMの速度向上の倍率からCPU数と同数までのPEであれば理想的に速度向上を得られることを示した．これは，解析規模に応じたCPU数を持つインスタンスを用いる事により，利用者が必要とする計算速度の向上を得られることができることを示している．

また，大学内で共有している大型計算機の場合には稼動率の関係から，多数の利用者が共同で利用する．そのため，実行ジョブの管理には多くの場合にバッチ方式が用いられている．このことは，利用者からはバッチシステムに登録した自らの実行ジョブがいつ実行されるかを管理することができないことを意味する．対して，今回のようにクラウドコンピューティングを用いた場合には，自由に計算規模に応じた計算機を占有することができる．しかしながら，利用料金が発生するため，常に並列計算機を構築した状態にしておくのではなく．必要とするときに必要なだけのインスタンスを用いて並列計算機の構築を行うことが求められる．

第3章　GPGPUに基づくLIMの高速過渡解析

3.1　概要

GPU(Graphics Processing Units) はCPU とは異なり，SIMT(Single Instruction, Multiple Thread)[52] と呼ばれるアーキテクチャで設計されている．これは，CPU が複雑な制御機能を処理できることに重点が置かれているのに対して，GPU はメモリ参照と同時に大量のスレッドによる演算処理に重点が置かれているためである．そして，画像の陰影処理のためにGPU 上に浮動小数点演算ユニットが実装され，画像処理以外の分野でGPU を活用する「GPU を用いた汎用計算 (GPGPU : General Purpose computing on Graphics Processing Units)」が活発に行われている [53, 54, 55]．これは，複雑な制御を必要とする処理はCPU で行い，それ以外の計算量を必要とする処理でGPU を用いる手法である．これには，特別なGPU を購入する必要は必ずしも無い．すなわち，一般的なPC に搭載されているGPU をハードウェアアクセラレーションとして利用することが可能である．

2章で，並列計算機での高速化について述べたが，市販されているGPUは最新のものであれば100個以上の演算装置が搭載されている．そのため，GPUによる高速化は多並列計算と考えることができ，また，一つのグラフィクスカード上に搭載されているメモリを用いるため共有メモリ型の並列計算機と考えることができる．しかしながら，前述のように制御処理が不得手であるため，従来のSPMD(Single Program Multiple Data)[50]のプログラミングモデルを用いた場合には性能を得ることができない．そのため，ストリームプログラミングモデル[53]に基づいたプログラミングを行う必要がある．

2章で並列分散型LIMが並列化に適しており，投入した計算機に比例した高速化の効果を得られることを確認している．そのため，GPUを用いた場合には，CPUと比べて非常に高速に解析を行うことが期待できる．そこで，本章ではGPUをハードウェアアクセラレータとして用い，GPUを使う上での最適化とLIMの高速化について述べる．

3.2 GPGPU (General Purpose computing on Graphics Processing Units)

近年，GPUによる浮動小数点演算が可能になり，グラフィックスカード上のメモリとGPU間のメモリバンド幅がCPUと比べて非常に早いハードウェエアとなった．この浮動小数点演算性能とメモリバンド幅を利用する，GPUを用いた汎用計算 (GPGPU: General Purpose computing on Graphcs Processing Units)[53] が注目を集めている．GPGPUでは，計算アルゴリズムが並列化に適している場合，高性能なGPU一つを搭載した計算機で計算機数十台分の性能をまかなうことができる．そのため，並列計算機を構築することよりも，廉価に高性能な計算資源を手に入れることができる．さらに，C言語拡張された開発環境であるCUDA (Compute Unified Device Architecture)[52] の登場により，従来のプログラムからGPUを利用したプログラムへの移植が容易となり，N体問題や粒子流体の解析にGPGPUが利用されている [53, 54, 55]．これはCPUでは実現が難しかったリアルタイムシミュレーションの実現が可能となり，N体問題のシミュレーションで利用されるGRAPE[56] といった専用ハードウェアがより容易に用いることができるとも言える．

ここでは，CUDAに対応したGPUのアーキテクチャとCUDAのプログラミングモデルについて述べる．

3.2.1 CUDA (Compute Unified Device Architecture)

CUDAはGPUを利用するための開発環境の一つであり，他にもOpenCL[57]やBrook[58]といった開発環境がある．CUDAの利用にはCUDAに対応したGPUが必要になり，これはCPUとは異なる特殊なハードウェアになる．そのため，CUDAでは特殊な計算処理単位が用いられ，この計算単位をスレッドと呼ぶ．スレッドはプログラムの実行単位の一つであり，複数のスレッドが実行されている場合には，同じ命令の処理がスレッドごとに行われる．CUDAでは同時に実行可能なスレッド数が非常に多く，これが高速な計算を可能としている．このようにCUDAは特殊な環境であるため，CPUを用いた時のプログラミングモデルとは異なるストリームプログラミングモデル[53]を用いる．そのため，CPUでのループ処理を展開した特殊なプログラミングを行うことになる．そして，このプログラミングモデルでは，カーネルによって処理が，参照するデータの配列をストリームとして管理される．

CUDAでは，CPUのことをHost，GPUのことをDeviceと呼び．Deviceで実行される関数のことをカーネルと呼ぶ．また，Deviceで処理を実行する際，Hostからグリッド情報と共にカーネルが発行されることでDeviceで処理が行われる．グリッドとは，Deviceで実行されるスレッドを管理する情報をまとめたものである．グリッドは任意の数のブロックにより構成され，更にブロックは任意の数のスレッ

ドによって構成される．図3.1にグリッドとブロック，スレッドの関係を示す．

そして，CUDAを利用した計算では，メモリアクセスの速度が演算性能に大きな影響を与える．そのため，同じ計算アルゴリズムであってもメモリアクセスの方法によっては，大きく計算性能が変化する．CUDAを利用して高い計算性能を導き出すにはGPU上に実装された共有メモリやレジスタを利用した効率的なメモリアクセスが必須となる．

3.2.2　CUDA対応GPUアーキテクチャ

CUDA対応のアーキテクチャはSingle-Instruction, Multiple-Thread [52]と呼ばれるアーキテクチャによって設計されている．これは，同時に多数のスレッドによって命令を実行することができるアーキテクチャである．ここでCUDA対応GPUの一つであるGTX280のブロックダイアグラムを図3.2に示す．GPUはいくつかのTPC（Texture/Processor Cluster）からなり，1個のTPCは3個のSM（Streaming Multiprocessor）とTexture Unitにより構成される．そして，SMは8個のSP（Streaming Processors cores）により構成される．このSPが実際に計算を行うプロセッサになる．そのため，GPU上に実装されている総SPの数でGPUの計算能力が決定される．各SMではSM内でのみ参照可能な16KByteの共有メモリとコンスタントメモリ，テクスチャメモリと8KByteのレジスタを備えてお

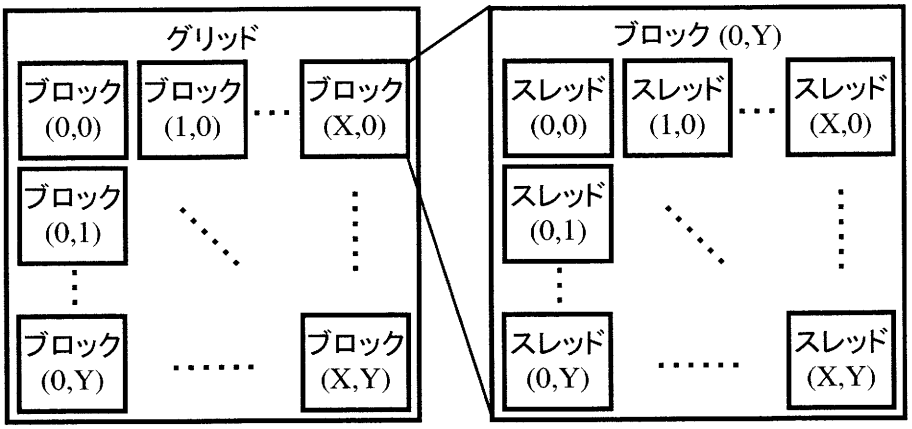


図 3.1: グリッドとブロック，スレッドの関係.

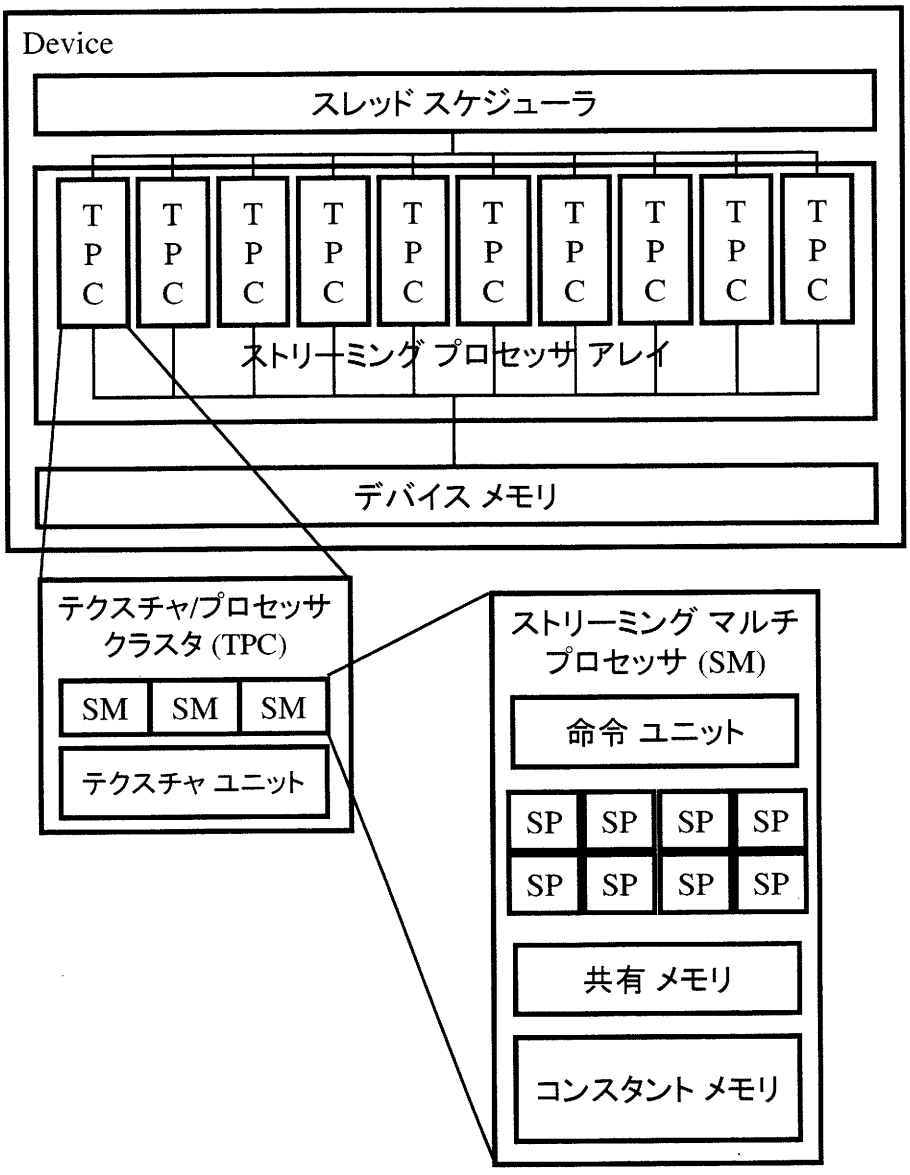


図 3.2: GTX280のブロックダイアグラム.

り，これらのメモリを効率的に用いることが性能向上の鍵となる．

3.3　GPGPUに基づくLIM (GPGPU-LIM)

GPGPU-LIMは，LIMで最も計算量を必要とする電流と電圧の更新処理をCPUではなくGPUを用いる手法である．GPGPU-LIMでは，通常のLIMと同様に電流と電圧を交互に更新するが，電流変数と電圧変数の更新は変数毎に割り当てられたスレッドにより更新される．LIMでは式（2.3.3），(2.3.4）より，電流または電圧の各変数の更新処理において，同時刻に更新される値を参照しない．そのため，各変数に割り当てられたスレッドは同時に電流または電圧の更新処理を行える．すなわち，GPGPU-LIMでは，更新処理を多数のスレッドによる電流または電圧の同時に多数の並列計算を行うことで高速化が実現される．

LIMの解析対象を6×5セルとした時の例題回路を図3.3に示す．図3.3中に，節点は7×6個，枝は$7\times5+6\times6$個存在する．LIMでは，節点に電圧変数が枝に電流変数が定義されるため，電圧変数を7×6個，電流変数を$7\times5+6\times6$個用いて過渡解析を行うことになる．従って，通常のLIMで解析対象が$n\times m$セルの場合に，電圧変数は$(n+1)\times(m+1)$個，電流変数については$(n+1)\times m$個と$n\times(m+1)$個の二次元の変数配列の確保し，メモリ参照時には順に参照を行う．一方，GPGPU-LIMでは全ての変数に対して$(n+1)\times(m+1)$個の一次元の変

数配列を確保し，これをある一定の大きさで分割してカーネル内でストリームとして扱う．すなわち，ストリームにメモリ参照処理が置き換えられる．同時に一次元配列で確保を行うのは，メモリの参照処理を効率化するためでもある．そのため，GPGPU-LIMでは，一次元配列で確保された変数配列を一定の領域ごとに領域分割を行う．この分割は電流変数と電圧変数で異なる領域分割の仕方を行う．

3.3.1　枝電流の領域分割

電流変数の領域分割の様子を図3.4に示す．電流変数に対しては，256変数毎に1つのブロックが割り当てられ，その中の1スレッド毎に1個の電流変数の更新が割り当てられる．図3.3と式（2.3.3）によれば，枝電流の更新を行う際に隣接する枝同士で参照する節点電圧が重複している．GPGPUでは，メモリアクセスの方法で性能が大きく左右される．そのため，この重複する値へのアクセス回数を少なくすることで性能を向上させている．すなわち，データの局所性を利用することで効率的なメモリアクセスを実現する．枝電流の更新処理では，最初にこの重複する節点の電圧変数の値を共有メモリに格納する．共有メモリに格納されたデータは，ブロック内の各スレッドで共有して参照することができるメモリである．値の格納後に共有メモリから節点電圧の値を参照し，枝電流の更新処理を行う．このように枝電流の更新処理では，節点電圧の参照へメモリ参照が集中する

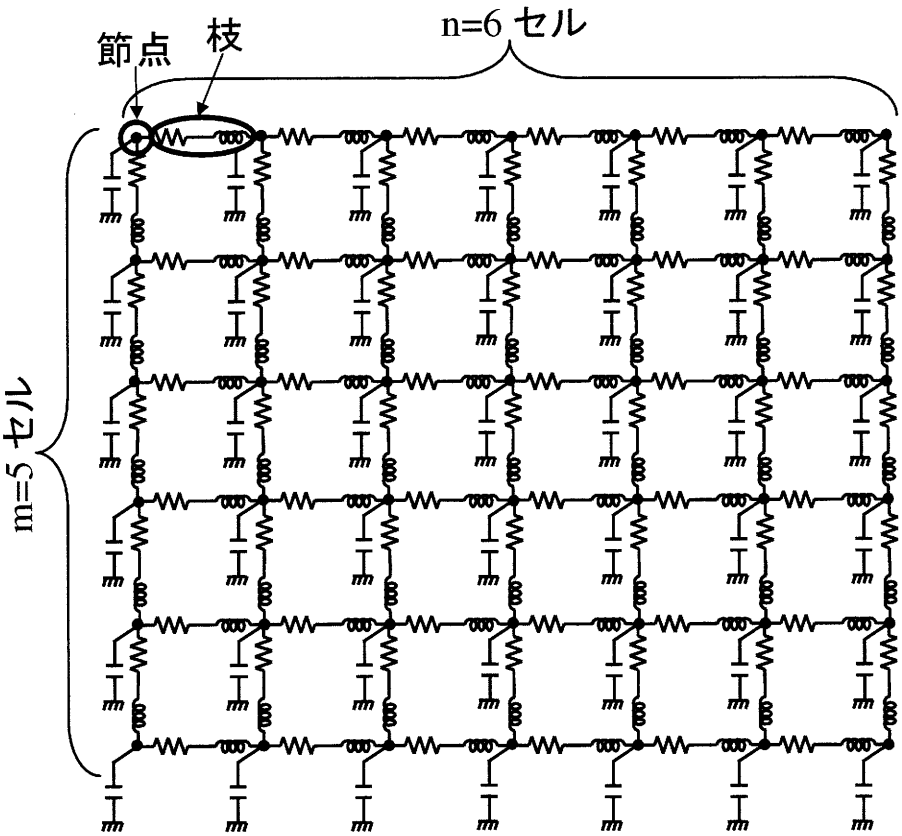


図 3.3: LIM の解析対象とする例題回路.

のを共有メモリを用いることにより，デバイスメモリへのアクセス回数を減少させている．

3.3.2　節点電圧の領域分割

次に，電圧変数の領域分割を図3.5に示す．電圧変数では，256変数×8列の領域が1つのブロックに割り当てられる．そのため，各スレッドが8回節点電圧の更新処理を行う．図3.3と式（2.3.4）によれば，節点には最大4個の枝が接続されており，また，節点電圧の更新の際，隣接する節点同士では重複して参照される枝電流の値が存在する．それ故，枝電流の更新処理と同様に，共有メモリやレジスタを用いてデバイスメモリへのアクセス回数を低減できる．レジスタは各スレッドに割り当てられ，割り当てられたスレッドからのみ参照することができるメモリである．図3.6に電圧の更新処理中に参照する枝電流へのアクセスを示す．1列目の更新処理では行方向で重複して参照される枝電流の値を共有メモリへ格納し，それ以外の枝電流の値をレジスタへ格納する．図3.6中では，レジスタに格納されるのはi_A，i_Bの領域にある枝電流の値，共有メモリに格納されるのは破線で囲まれたi_{S1}の領域にある枝電流の値である．格納後，共有メモリとレジスタからのみ枝電流の値を参照して更新処理を行う．次に，2列目以降の更新処理では，1列前の節点電圧の更新処理でレジスタへ格納した枝電流1つが今回の更新処理で参照

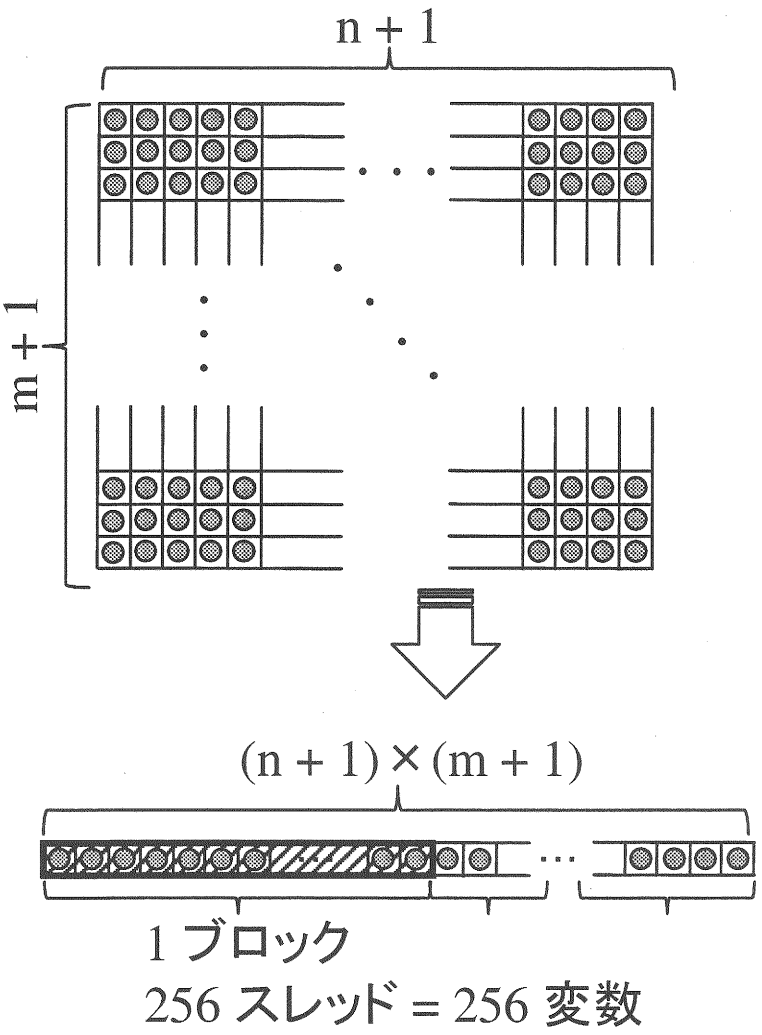


図 3.4: 枝の電流変数の領域分割.

する枝電流と重複する．図3.6中の2列目の更新処理ではi_Bが重複する枝電流である．そこで，i_Bの値を保持し，i_Cの値を新たにレジスタへ格納する．共有メモリにはi_{S2}の値を格納する．この時，i_B以外の1列目の更新処理で格納した共有メモリとレジスタの値は破棄される．格納後，1列目の時と同様に更新処理を行い，8列目までこの処理を繰り返す．このように，節点電圧の更新処理では共有メモリとレジスタを利用し，ブロックまたはスレッドごとのデータ共有によりデバイスメモリへのアクセス回数を低減させている．これは，メモリアクセスを処理の順番に沿って適切なローカルメモリへと格納，破棄，参照を明示していることになる．

3.3.3　GPGPU-LIMの更新手順

GPGPU-LIMでの更新手順を図3.7に示す．本論文では，GPGPU-LIMをCUDAを用いて実装したため，CPUをHost，GPUをDeviceと定義している．GPGPU-LIMでは，枝電流と節点電圧の更新処理をHostがグリッド情報とともにカーネルを発行することで，Deviceで実行される．そして，時刻の管理や出力処理をHostで行う．そのため，解析結果の出力はHostへ伝達しなければならない．この解析結果の伝達はDeviceのメモリからHostのメモリにメモリ情報を転送することを意味する．GPGPU-LIMでは，1時間ステップの更新処理終了後に出力処理のために必要なメモリ情報をDeviceからHostへ転送することで，出力処理を可能に

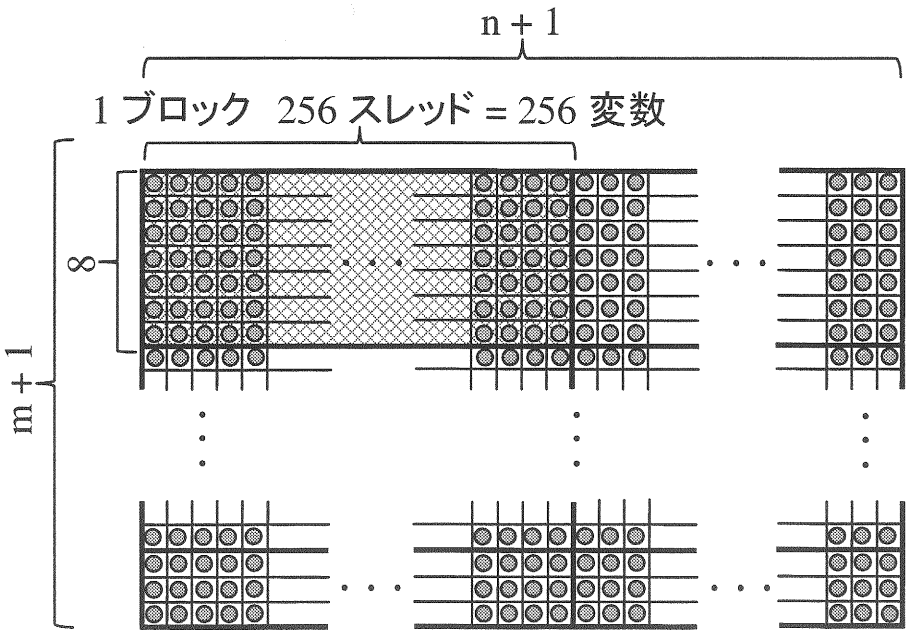


図 3.5: 節点の電圧変数の領域分割.

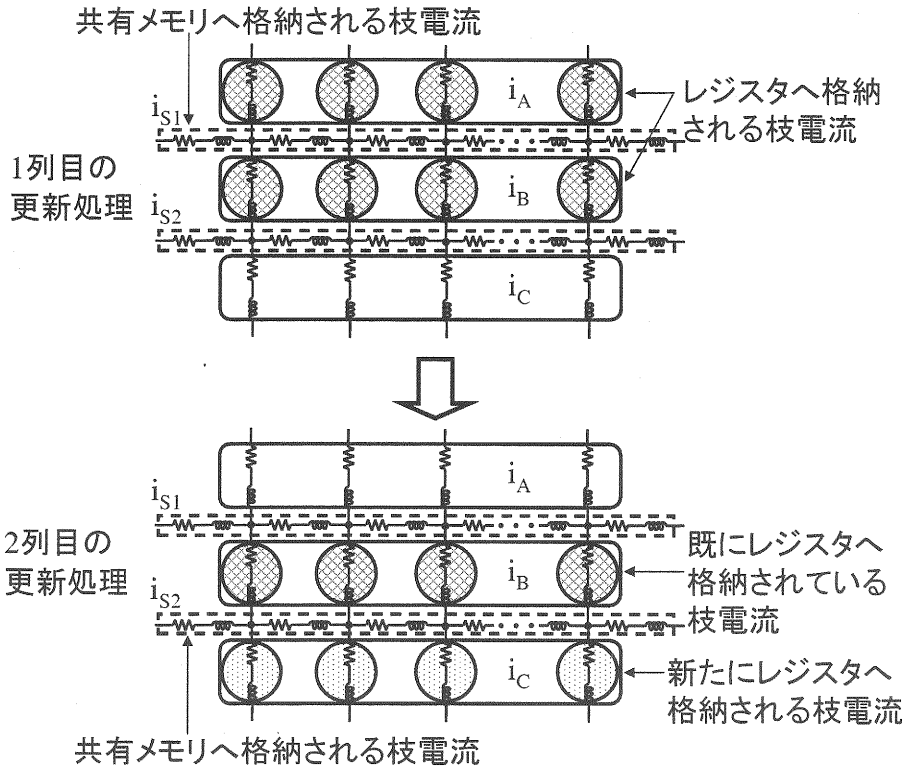


図 3.6: 枝電流へのアクセス.

している．

3.4　計測結果

電源分配回路網の電源/グランドプレーンは，図3.8のように抵抗とインダクタンス，接地キャパシタンスによりモデル化が行われる[59]．そこで，図3.8で示す電源/グランドのプレーン等価回路を解析対象とし，観測点で電圧を測定した．解析対象の一辺の長さを同じとし，セル数を200×200セルから3000×3000セルまで解析領域の一辺につき200セルずつ順に増加させた15パターンを，解析区間[0秒, 4n秒]とし，電源に遅延0.2n秒，立ち上がり，立下がりが共に0.1n秒，パルス幅1.5n秒，電圧5.0Vの電圧波形を入力し，観測点として右下に位置する節点の電圧を測定した．各素子の値は，R_{in}を$10[\Omega]$，Rを$1.68\times10^{-2}[\Omega]$とするが，L，Cはセル数によって単位セルの長さと幅が変わり，素子値も変化する．そのため，200×200 Cellのとき，Lを6.3×10^{-11}[H]，Cを4.43×10^{-16}[F]として解析を行っている．

LIMが従来のSPICE系シミュレータと比べて高速であることは2章で述べたため，ここでは，出力波形の比較を行うために解析対象を200×200の時の解析結果をHSPICEで取得し，比較する．そして，GPGPU-LIMとLIMの実行時間の比較を行った．

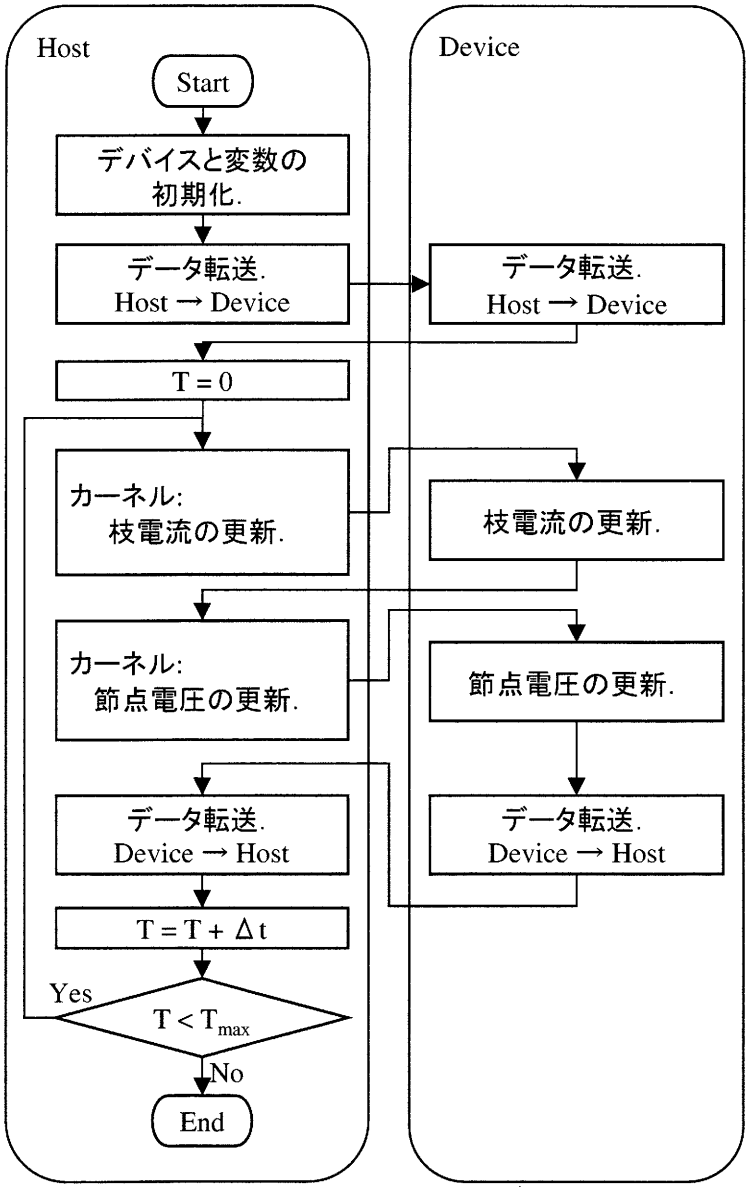


図 3.7: GPGPU-LIM のフローチャート.

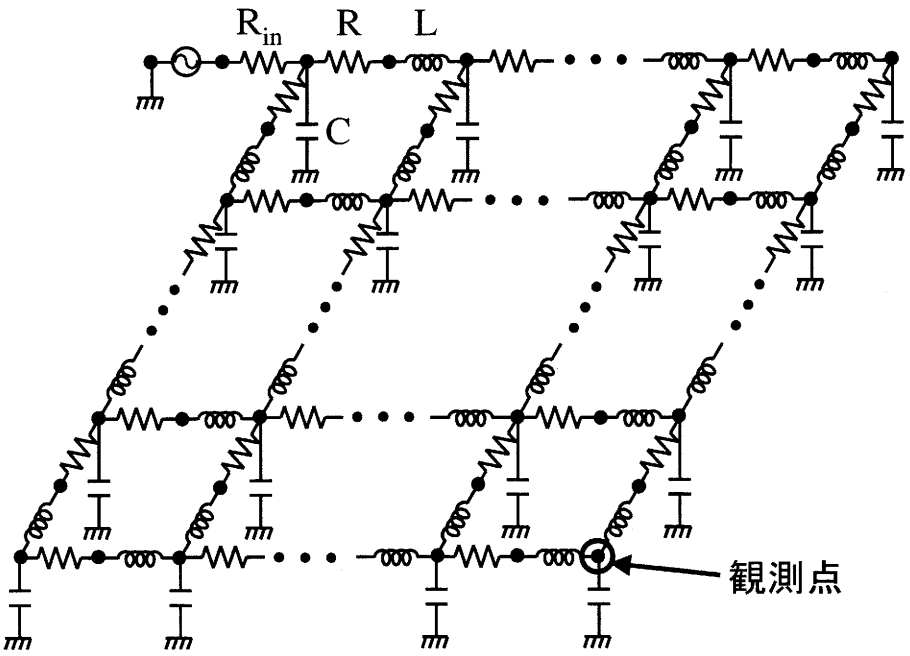


図 3.8: 例題回路（電源/グランドのプレーン等価回路）.

計測環境として，GPUはGeforce GTX295をCPUはXeon 3.2GHzを使用して，実行時間の計測を行った．Geforce GTX295は，一つのグラフィックカード上に二つのGPUを持つ構成になっている．Geforce GTX295に搭載されているGPUはそれぞれがStreaming Multiprocessor（SM）を30個内蔵，つまり240個のStreaming Processor（SP）から成り，メモリを896MB搭載した構成である．今回は搭載されているGPUの内一つのみを利用して計測を行った．また，GPUは単精度浮動小数点型で過渡解析を行い．CPUでは倍精度浮動小数点型で解析を行っている．これは，GPUは単精度浮動小数点での演算において多数スレッドでの演算が可能なためである．

解析対象を200 × 200セルとして，LIM，HSPICE，そしてGPGPU-LIMで解析した出力波形を図3.9に示す．

図3.9より，GPGPU-LIMとLIM，HSPICEにおいて同様の出力波形を得られている．セル数を変更した場合でも同様に等価な出力波形を得ることができた．これは単精度浮動小数点型であってもGPGPU-LIMによる解析が十分な精度で解析できることを意味している．次に，セル数を変えて解析を実施した際の実行時間を図3.10に，その速度向上の倍率を図3.11に示す．図3.10と図3.11から，GPGPU-LIMはLIMより解析にかかる実行時間が劇的に削減されていることが分かる．特に600 × 600セル以上の解析において，30倍以上の高速化を実現している．規模

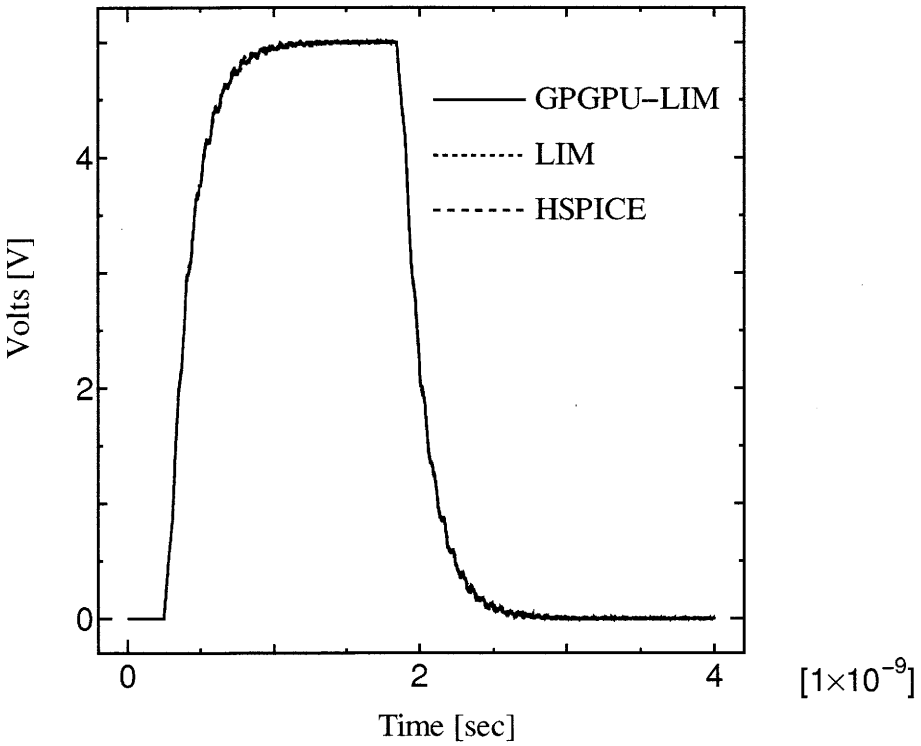


図 3.9: 200 × 200 セルでの過渡解析結果.

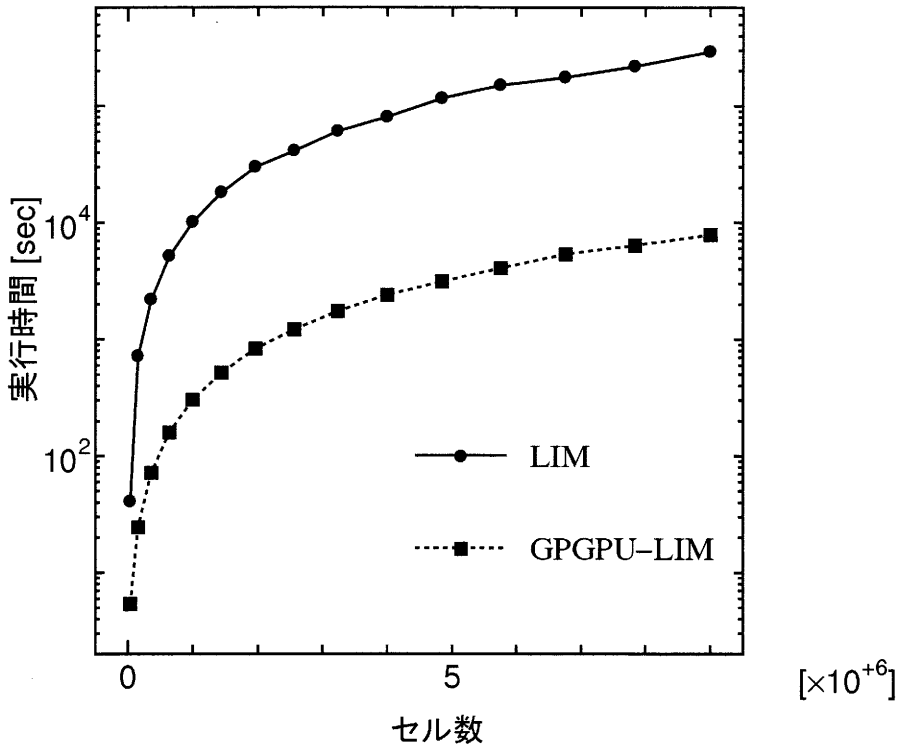


図 3.10: 実行時間の比較.

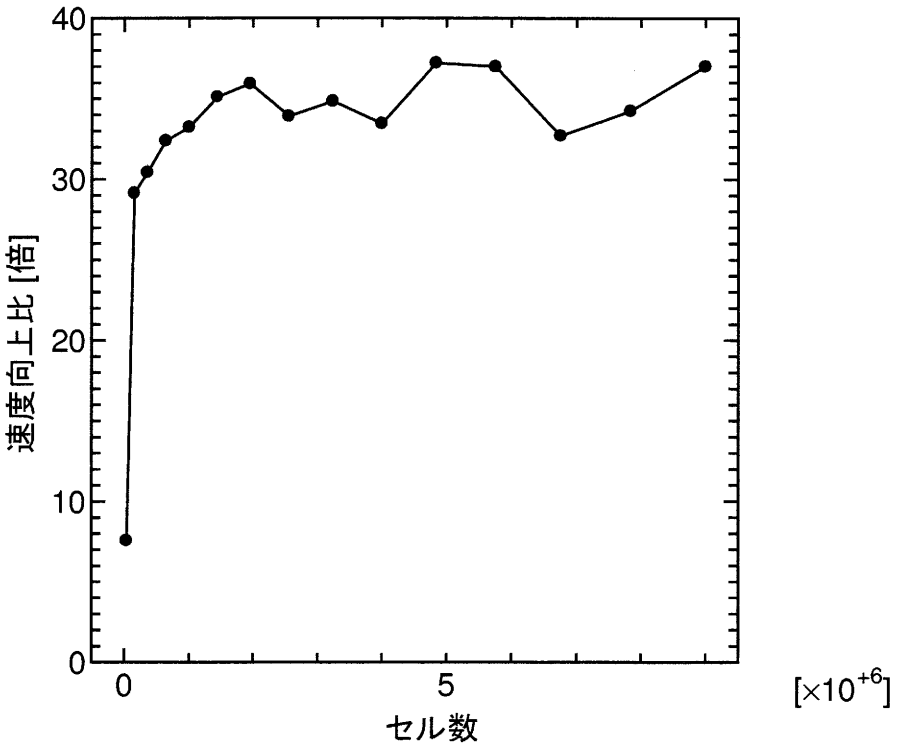


図 3.11: 速度向上の倍率.

によって計算速度の向上にばらつきがあるのは，領域分割ではSMの数を上回る数の部分領域に分割される．そのため，部分領域をSMに割り当てて更新を行わせるGPU上のスケジューリングによるものと考えられる．今回の計測ではCPUにXeon 3.2GHzを用いているが，最新のCPUでは理論性能が高いCPUが存在する．それらと比較を行った場合には，この速度向上の倍率は30倍よりも低くなる．しかしながら，その演算性能は今回用いたCPUと比べて倍程度であり，一桁以上高速ではない．そのため，GPGPU-LIMはHSPICEと比べて三桁以上高速に回路の過渡解析を行うことできる．

3.5　本章の総括

本章では，GPU上でのLIMの実装とその性能評価について述べた．GPGPU-LIMとLIMを電源分配回路網の電源/グランドプレーンに適用し，HSPICEの結果と比較することにより，出力波形の一致を確認した．プログラム中でGPGPU-LIMは単精度浮動小数点を用いて，LIMは倍精度浮動小数点を用いている．そのため，GPGPU-LIMはLIMと比べて誤差が蓄積されやすいが，同精度での解析を行うことができることを示している．実行時間については，HSPICEとLIMでは160倍近くLIMが高速に過渡解析を行うことができ，更にGPGPU-LIMではLIMよりも30倍以上高速に解析を行うことができる．CPUの性能も向上しているた

め，常に30倍以上高速に解析を行えるわけではないが，GPUはCPUと比べて一桁以上高速なアプリケーションを開発できるハードウェアでと言える．そのため，GPGPU-LIMではHSPICEと比べて三桁以上の高速化が可能である．

並列分散処理で計測で用いたのと同じ性能のCPUを用いた場合には，同様の効果を得るためには30個以上のCPUが必要となり，実際に物品を購入してGPUと同様の効果を得るのが困難である．一方で，GPUについては更なる性能の向上が計画されている．このことから電源分配回路網の高速過渡解析にGPGPU-LIMが極めて有効であり，今後のハードウェアを用いた高速化では必須になると考えられる．

第4章　並列分散型ブロックLIMによる高速過渡解析

4.1　概要

電源分配回路網やバスシステムを表したCADモデルを抽出ツールを用いて等価回路網の作成を行うと，相互インダクタンスと相互キャパシタンスにより相互結合された強結合伝送線路としてモデル化が行われる．このような回路網をSPICE系のシミュレータを用いて解析した場合，相互結合素子によるfill-inの増加により膨大な計算時間を必要とする．そこで，強結合伝送線路を高速に解析するため，LIMを拡張したブロックLIMが提案された[60]．ブロックLIMは，相互インダクタンスと相互キャパシタンスによって相互結合された部分をブロックとして扱い，このブロックの更新処理にのみ行列演算を用いる．ブロック化された部分は回路網全体から見ると局所的であり，局所的陰的手法であると言える．そのため，回路網全体で行列演算を用いるSPICE系の解析手法と比べて，非常に高速に回路の過渡解析を行うことが期待される．

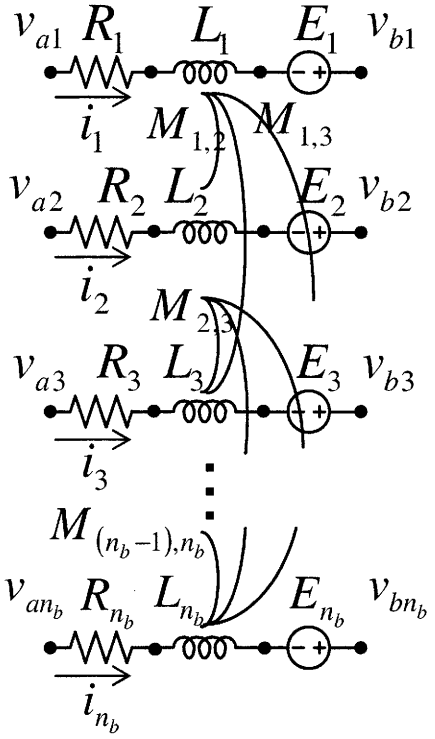
ブロックLIMの更新処理は，LIMと同様に電流と電圧が交互に更新される．そのため，LIMのように並列化に適した手法であると考えられる．本章では，ブロックLIMに対してMPIを用いて並列化を行い，並列計算機環境でその性能を検証する．

4.2　ブロックLIM

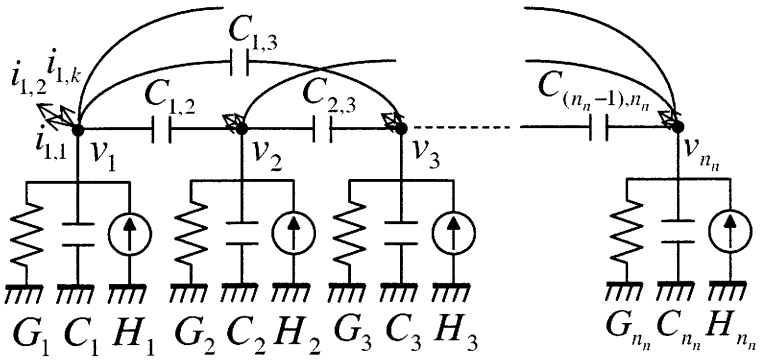
4.2.1　Originalブロック LIM

近年，相互インダクタンスと相互キャパシタンスによって相互結合された回路網を解析することができるブロックLIMが提案された[60]．ブロックLIMが解析対象とする回路では，LIMの枝構造と節点構造に加えて，図4.1に示す枝ブロック構造と節点ブロック構造が複数個含まれている．枝ブロック構造はb_n個の枝構造によって構成され，それぞれの枝構造に含まれるインダクタ同士が全て相互インダクタンスによって結合している．同様に，節点ブロック構造はn_n個の節点構造によって構成され，それぞれの節点構造に含まれるキャパシタ同士が全て相互キャパシタンスによって結合している．

ブロックLIMもLIMの更新式と同様の導出を行う．すなわち，図4.1(a)に示す枝ブロック構造において，全ての枝構造へKVLを適用すると，任意の枝ブロック



(a) 枝ブロック部の構成要素



(b) 節点ブロック部の構成要素

図 4.1: ブロック LIM の構成要素.

構造についての式は連立一次方程式の形になり，

$$\mathbf{v}_{ab} = \mathbf{R}_{ab} \cdot \mathbf{i}_{ab} + \mathbf{L}_{ab} \cdot \frac{d}{dt}\mathbf{i}_{ab} - \mathbf{e}_{ab} \tag{4.2.1}$$

と書ける．ここで$\mathbf{R}_{ab}$と$\mathbf{L}_{ab}$，$\mathbf{i}_{ab}$，$\mathbf{v}_{ab}$，$\mathbf{e}_{ab}$は4.1(a)に依存する行列とベクトルであり，

$$\mathbf{R}_{ab} = \begin{bmatrix} R_1 & 0 & \cdots & 0 \\ 0 & R_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{n_b} \end{bmatrix}, \mathbf{L}_{ab} = \begin{bmatrix} L_1 & M_{1,2} & \cdots & M_{1,n_b} \\ M_{1,2} & L_2 & \cdots & M_{2,n_b} \\ \vdots & \vdots & \ddots & \vdots \\ M_{1,n_b} & M_{2,n_b} & \cdots & L_{n_b} \end{bmatrix},$$

$$\mathbf{i}_{ab} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_{n_b} \end{bmatrix}, \mathbf{v}_{ab} = \begin{bmatrix} v_{a1} - v_{b1} \\ v_{a2} - v_{b2} \\ \vdots \\ v_{an_b} - v_{bn_b} \end{bmatrix}, \mathbf{e}_{ab} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_{n_b} \end{bmatrix},$$

である．式(4.2.1)に，leapfrogアルゴリズムに基づく有限差分法を適用すると，

$$\mathbf{v}_{ab}^{n+\frac{1}{2}} = \mathbf{R}_{ab} \cdot \mathbf{i}_{ab}^{n} + \mathbf{L}_{ab} \cdot \frac{1}{\Delta t}\left(\mathbf{i}_{ab}^{n+1} - \mathbf{i}_{ab}^{n}\right) - \mathbf{e}_{ab}^{n+\frac{1}{2}} \tag{4.2.2}$$

が得られる．このとき，式(4.2.2)の未知変数ベクトル$\mathbf{i}_{ab}^{n+1}$について整理すると，

$$\mathbf{L}_{ab}\cdot\mathbf{i}_{ab}^{n+1}=(-\Delta t\mathbf{R}_{ab}+\mathbf{L}_{ab})\cdot\mathbf{i}_{ab}^{n}+\Delta t\left(\mathbf{v}_{ab}^{n+\frac{1}{2}}+\mathbf{e}_{ab}^{n+\frac{1}{2}}\right) \tag{4.2.3}$$

となる．

同様に，図4.1(b)に示す節点ブロック構造において，全ての節点構造へKCLを適用すると，任意の節点ブロック構造についての式は連立一次方程式の形になり，

$$-\mathbf{i}_a=\mathbf{G}_a\cdot\mathbf{v}_a+\mathbf{C}_a\cdot\frac{d}{dt}\mathbf{v}_a-\mathbf{h}_a \tag{4.2.4}$$

と書ける．ここで$\mathbf{G}_a$と$\mathbf{C}_a$，$\mathbf{v}_a$，$\mathbf{i}_a$，$\mathbf{H}_a$は4.1(b)に依存する行列とベクトルであり，

$$\mathbf{G}_a=\begin{bmatrix}G_1 & 0 & \cdots & 0\\ 0 & G_2 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & G_{n_n}\end{bmatrix},\mathbf{C}_a=\begin{bmatrix}\alpha_1 & -C_{1,2} & \cdots & -C_{1,n_n}\\ -C_{1,2} & \alpha_2 & \cdots & -C_{2,n_n}\\ \vdots & \vdots & \ddots & \vdots\\ -C_{1,n_n} & -C_{2,n_b} & \cdots & \alpha_{n_n}\end{bmatrix},$$

$$\mathbf{v}_a = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n_n} \end{bmatrix}, \mathbf{i}_a = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n_n} \end{bmatrix}, \mathbf{h}_a = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_{n_n} \end{bmatrix},$$

である．ここで，α_m と β_m は次のように定義する．

$$\alpha_m = \sum_{k=1}^{m-1} C_{k,m} + C_m + \sum_{k=m+1}^{n_n} C_{m,k}$$
$$\beta_m = \sum_{k=1}^{M_m} i_{m,k}, (m = 1, 2, \ldots, n_n)$$

式(4.2.4)に，leapfrogアルゴリズムに基づく有限差分法を適用すると，

$$-\mathbf{i}_a^n = \mathbf{G}_a \cdot \mathbf{v}_a^{n-\frac{1}{2}} + \mathbf{C}_a \cdot \frac{1}{\Delta t}\left(\mathbf{v}_a^{n+\frac{1}{2}} - \mathbf{v}_a^{n-\frac{1}{2}}\right) - \mathbf{h}_a^n \tag{4.2.5}$$

が得られる．この時，式(4.2.5)の未知変数ベクトル $\mathbf{v}_a^{n+frac12}$ について整理すると，

$$(\Delta t\mathbf{G}_a + \mathbf{C}_a) \cdot \mathbf{v}_a^{n+\frac{1}{2}} = \mathbf{C}_a \cdot \mathbf{v}_a^{n-\frac{1}{2}} + \Delta t\left(-\mathbf{i}_a^n + \mathbf{h}_a^n\right) \tag{4.2.6}$$

となる．

ブロックLIMでは，式(4.2.3)，(4.2.6)を解く時のみ行列演算を必要とする．す

なわち，ブロック化された枝と節点の更新には式(4.2.3)と(4.2.6)を，ブロック化されていない枝と節点については式(2.3.3)と(2.3.4)を用いて更新を行う．このとき，電流と電圧を交互に更新することで，回路の過渡解析を行う．

4.2.2　並列分散型ブロックLIM

ブロックLIMは，回路全体の枝ブロック構造と節点ブロック構造の電流と電圧を交互に更新することで過渡解析を行う．並列分散型ブロックLIMも同様に電流と電圧を交互に更新するが，回路全体をいくつかの枝ブロックと節点ブロックを含む部分回路に分割する．そして，各部分回路に1つのPEを割り当て，各部分回路は割り当てられたPEのみで更新処理を行う．この時，各部分回路は節点ブロックが隣接するPE同士で重複して更新されるブロックとなる．そのため，分割処理はこの重複するブロックを加えたうえで，各PEの部分回路の大きさが均等となるように分割を行う．式(4.2.3)と(4.2.6)より，枝または節点ブロックの更新処理は，接続された電流または電圧の値を必要とする．したがって，更新処理の最中で隣接する部分回路同士で情報の通信が発生する．

図4.2に5個の節点ブロック構造と4個の枝ブロック構造からなる強結合伝送線路を2個のPEによって分割した時の回路の分割を示し，行列での分割の様子を図4.3に，更新手順を図4.4に示す．N_1，N_2，N_3は節点ブロックをB_a，B_bは枝ブ

ロックを表し，$\mathbf{v}_1$，$\mathbf{v}_2$，$\mathbf{v}_3$ は節点ブロック N_1，N_2，N_3 の電圧ベクトルの値，$\mathbf{i}_a$，$\mathbf{i}_b$ は枝ブロック B_a，B_b の電流ベクトルの値に対応する．そして，$\mathbf{v}_2$ が重複して更新される節点ブロック N_2 の電圧ベクトルの値である．並列分散型ブロックLIMは，最初に重複して更新されるブロックに隣接するブロックの電流または電圧の値が更新される．すなわち，図4.2において，PE1で $\mathbf{i}_a$ が，PE2で $\mathbf{i}_b$ が式(4.2.3)により更新される．次に，隣接する各PE間で直前に更新された電流ベクトルの値を互いに送受信しあう．PE1はPE2に向けて $\mathbf{i}_a$ を送信し，$\mathbf{i}_b$ の情報を受信する．同様に，PE2はPE1へ $\mathbf{i}_b$ を送信し，$\mathbf{i}_a$ の情報を受信することで，各PEは式(4.2.6)における $\mathbf{v}_2$ の更新処理に必要な電流ベクトルの値が更新されたことになる．すなわち，計算処理を通信処理に置き換えている．さらに，この通信中に重複して確保されていないブロックの更新処理を行う．これは，各PEでは $\mathbf{v}_2$ と $\mathbf{i}_a$，$\mathbf{i}_b$ を除くすべての部分回路の電流と電圧の更新処理が通信処理と並行して行われることになる．このように，通信処理と更新処理を同時に実行するのは，通信時間に必要とする時間は演算処理に比べて多くの時間を必要とする．そのため，互いの処理を重複して実行することで実行時間全体に対して逐次処理として現れる通信時間を減少させるために行う[50]．通信完了後，各PEで重複して更新される $\mathbf{v}_2$ を式(4.2.6)を用いて更新する．並列分散型ブロックLIMは，この一連の処理により，1時間ステップの更新処理が行われる．結果として，ブロックLIMと並

列分散型ブロックLIMでは全く同じ結果を得ることができる．

先ほどまでは，実行するPEが2個の時について述べたが，2個より多い場合であっても同様の回路分割法と更新処理により，並列化を行うことができる．その際，回路全体の枝と節点のブロック数と実行PEの数について考える必要がある．すなわち，枝と節点のブロック数が実行PEの数よりも共に多くなければならない．これは，枝と節点のブロック数が実行PEの数と同等か少ない場合，各PEに割り当てられる部分回路が重複して更新されるブロックのみとなり，通信処理にかかる時間が各PEの実行時間に対して大きくなるためである．また，負荷分散が均等になるようにもしなければならない．これは，重複する節点ブロックの更新処理の前に各PEの通信処理が終了しているかの判定を行う．この際，負荷分散が不均等な場合には，通信終了待ちのPEが存在し，全体の性能を下げることになる．そのため，並列分散型ブロックLIMで並列化の効果を得るためには，部分回路中に重複しないブロックが複数個割り当てられ，かつ，各PEに割り振られる部分回路の大きさが均等になるようにするのが望ましい．

ここで並列分散型ブロックLIMの係数を行列として配置した場合を考えると，図4.3に示す構造の行列となる．節点と枝の係数行列は密行列であり，この行列は，密なブロック行列が対角上に配置されたブロック対角行列と類似している．この行列をLU分解法などで解を得る場合には，左下と右上に各要素の接続関係を表

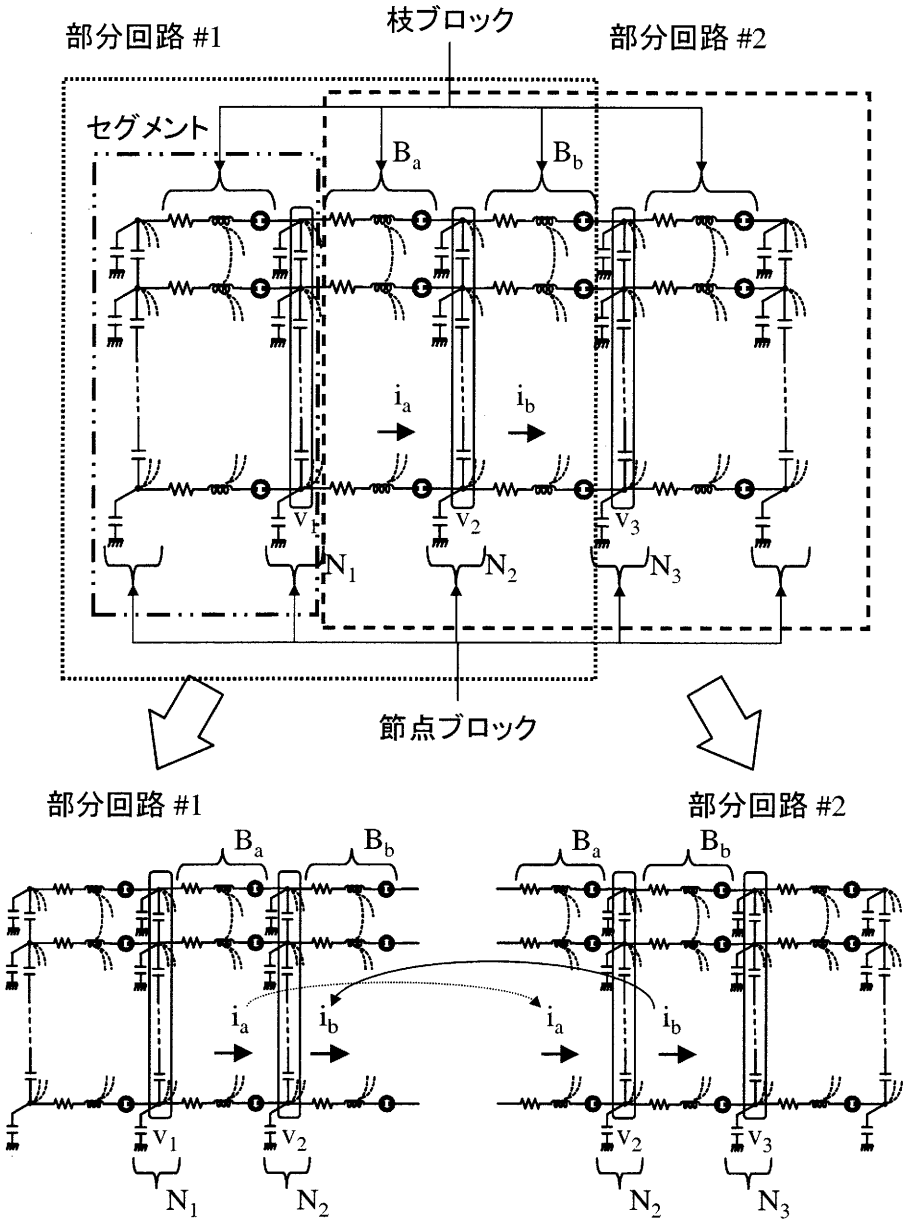


図 4.2: 並列分散型ブロック LIM の領域分割.

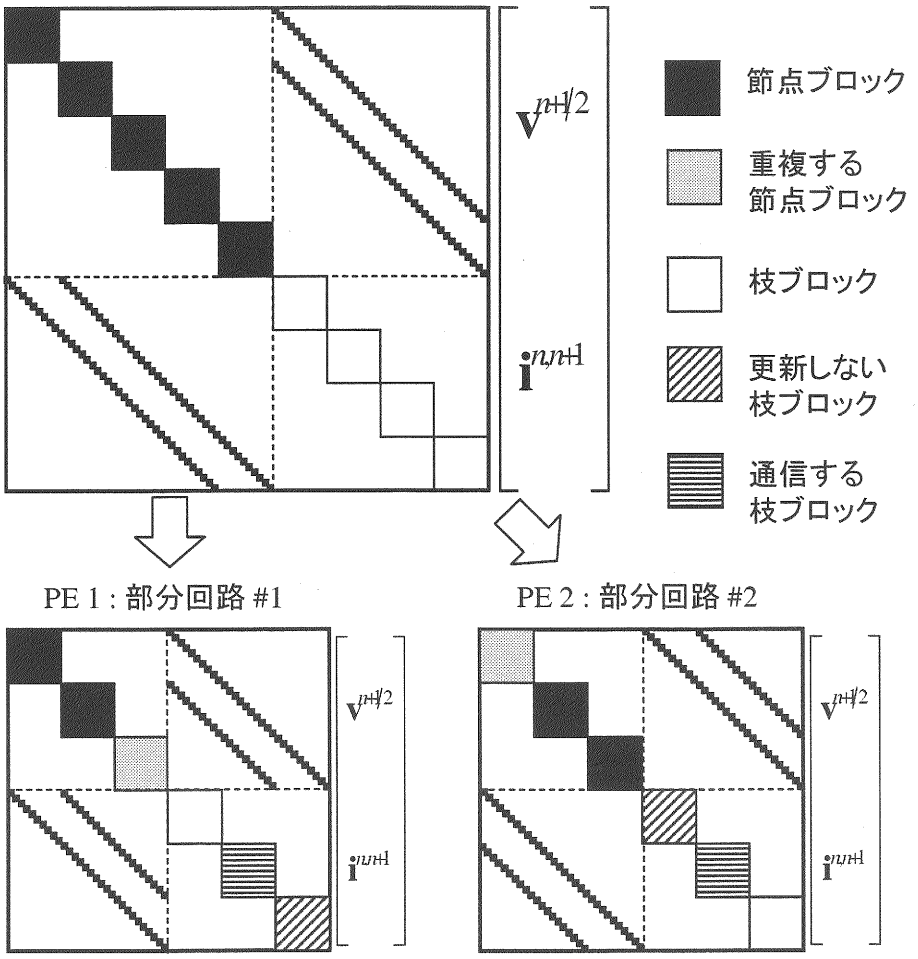


図 4.3: 行列で表した領域分割.

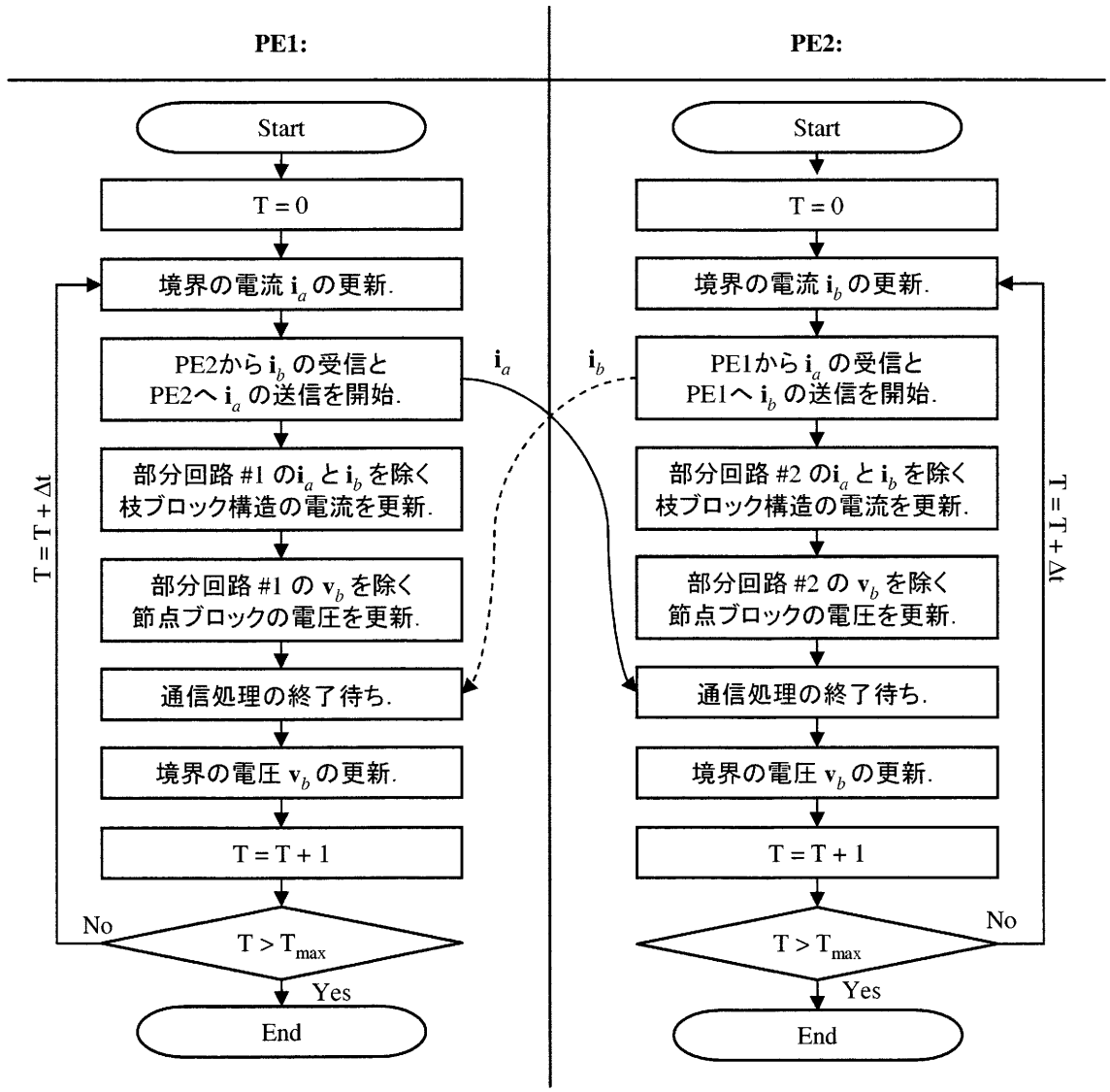


図 4.4: 並列分散型ブロック LIM のフローチャート.

した接続行列により，大量のfill-inが発生するが，ブロックLIMのアルゴリズムにより，接続行列が右辺側に配置される．そのため，更新時には電流と電圧のそれぞれについてブロック対角行列を用いて解を求めていると考えることができる．これにより，回路網の構造に応じて，任意のブロック対角行列で領域分割をすることができ，各ブロック対角行列に対してPEを割り当てることが可能となる．

4.3　計測結果

並列分散型ブロックLIMの性能を検証するため，図4.5に示す160本の伝送線路が相互インダクタンスと相互キャパシタンスによって相互結合された例題回路を用いて解析を行った．電源に遅延1n秒，立ち上がり，立下りが共に0.1n秒，パルス幅9.9n秒，内部抵抗10Ω，電流0.1Aの電流波形を入力し，時間刻み幅を1p秒とし，入力の反対側に位置する節点の電圧を測定した．ここでは，節点ブロック2個と枝ブロック1個つで構成された部分をセグメントと定義する．計測では，セグメントを5個繋げた例題回路に対して，HSPICEとブロックLIMでの比較とブロックLIMと並列分散型ブロックLIMで，セグメントを5個，10個，20個，50個繋げた例題回路の実行時間の計測を行った．計算機環境は，HSPICEとブロックLIMの比較は，Sparcv9 1GHzを搭載した計算機で，ブロックLIMと並列分散型ブロックLIMは表4.1に示すT2Kオープンスーパーコンピュータの仕様[61]に

表 4.1: 並列計算機の構成.

OS	RedHat Enterprise Linux AS V4
CPU	Quad-Core AMD Opteron 2.3 GHz
CPU数	4
コア数	16
メモリ	32 GByte
ネットワークカード	InfiniBand DDR

表 4.2: HSPICEとBlock-LIMの実行時間の比較.

線路数	セグメント数	要素数	実行時間 [sec]	
			HSPICE	ブロックLIM
160	5	244,429	27,014.95	245.59

沿って構成された並列計算機を用いた．また，この計算機の構造を図4.6に示す．そして，計算機の仕様により，ブロックLIMは1コアを利用して実行され，並列分散型ブロックLIMは各コアによって実行される．すなわち，2PEの場合には2コアを用いて実行され，最大で16コアまでの間で実行することができる．

それぞれの手法で過渡解析を行った出力波形を図4.7に示す．そして，HSPICEとブロックLIMの実行時間の比較を表4.2に示す．さらに，ブロックLIMを1PEで実行し，並列分散型ブロックLIMで実行するPEの数を順に増やした時の実行時間の比較を表4.3に示す．表4.3中で,「-」としている個所は，計測を行っていない個所である．これは，実行PE数を増やしても重複するブロックが増えるのみで並列化の効果を得ることができないためである．

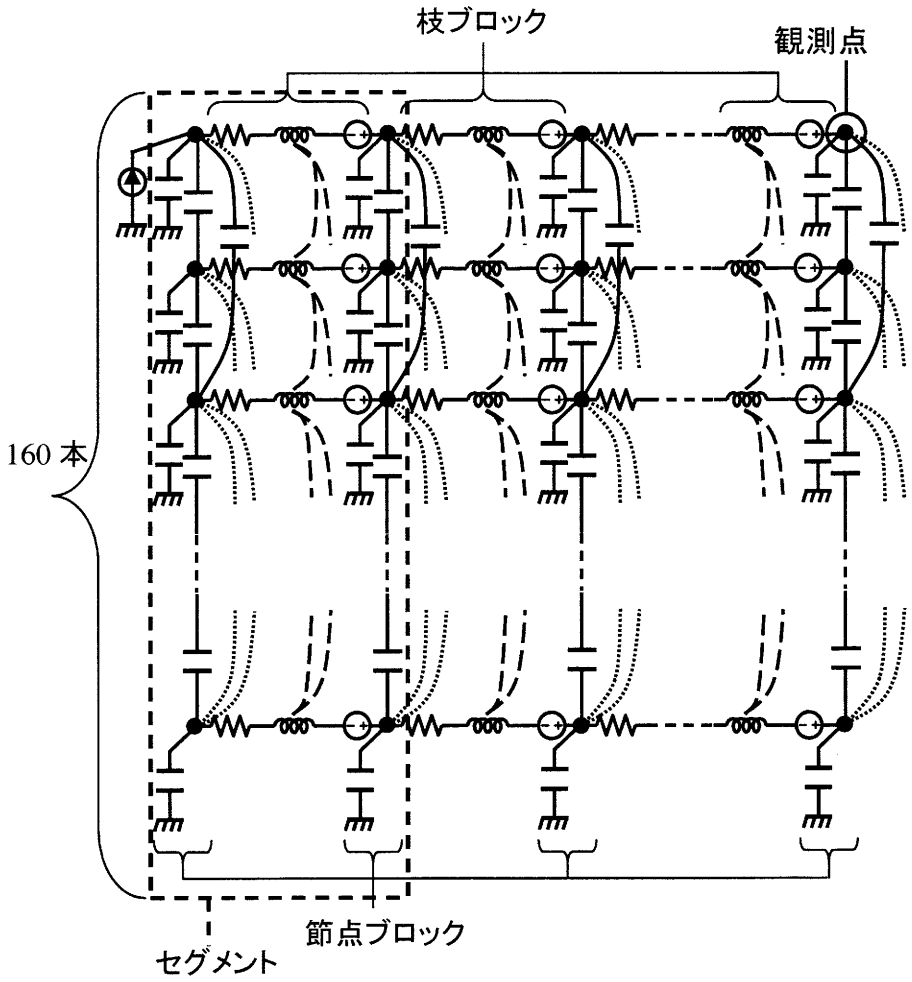


図 4.5: 例題回路.

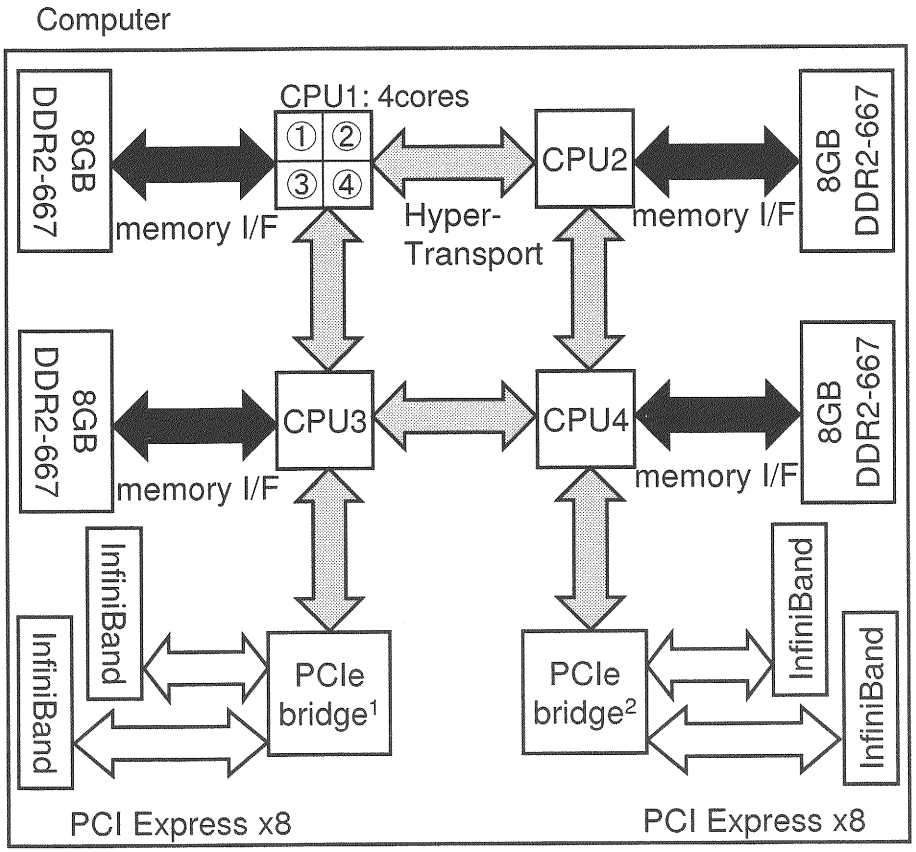


図 4.6: T2K の仕様に基づく計算機.

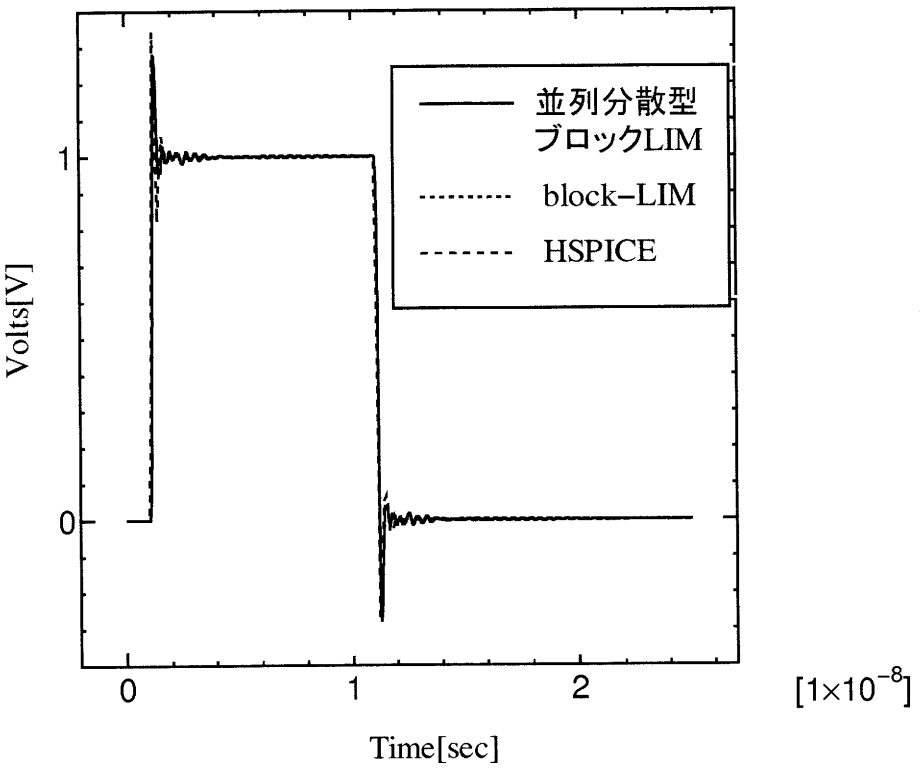


図 4.7: 出力波形.

表 4.3: ブロックLIMと並列分散型ブロックLIMの実行時間の比較.

		PE数	セグメント数			
			5	10	20	50
実行時間 [sec]	ブロックLIM	1	118.35	259.12	529.94	1,345.01
	並列分散型ブロックLIM	2	79.25	136.15	272.91	679.50
		4	-	93.79	152.69	372.67
		6	-	-	123.66	263.65
		8	-	-	96.5	208.67
		10	-	-	69.22	156.65
		12	-	-	-	153.22
		14	-	-	-	128.53
		16	-	-	-	126.83

図4.7より，各手法の出力波形は同様の出力波形を得られる．表4.2より，ブロックLIMはHSPICEと比較して100倍以上高速に過渡解析を行うことができることが確認される．更に，表4.3と図4.8より，2PE使用時には倍近く高速化することができ，セグメント数50の時には14PE以上を使用することで10倍以上高速に解析可能であることが確認できる．速度向上が10PE以上を使用した場合に飽和した状態になるのは負荷分散の問題である．すなわち，最も時間のかかるPEに割り当てられたセグメントの数が同じとなるためである．この場合では，10PEと12PEでは6セグメント，14PEと16PEでは5セグメントが割り当てられる．ここで，説明を簡単にするためにセグメントをやめ，ブロックの個数で説明を行う．50セグメントで構成される強結合伝送線路の回路網は51個の節点ブロックと50個の枝ブロックによって構成され，各ブロック160次元の密行列をLU分解法を

用いて解いている．すなわち，各ブロックの計算量は均等であり，最も多くのブロックが割り当てられたPEによって並列化の効果が制限される．LU分解法を用いて更新を行うブロックの個数は，10PEと12PEでは11個，14PEと16PEの場合では9個となり，それぞれで得られる速度向上の倍率は最大で9倍と11倍である．この最大の速度向上が得られた場合に実行時間は，10PEと12PEでは149.45秒，14PEと16PEでは122.27秒となる．そのため，理論値よりも最大で7.2秒遅いが，その遅延は5%程度であり本手法が非常に効果的である事がわかる．

4.4　本章の総括

本章では並列分散型ブロックLIMを用いて強結合伝送線路の解析を行った．計測結果から，並列分散型ブロックLIMはブロックLIMに対して，16PE使用時に10倍以上高速に過渡解析を行うことができることを確認した．ブロックLIMは，HSPICEと比べて100倍以上高速に過渡解析を行うことができるため，並列分散型ブロックLIMではHSPICEと比べて1000倍以上高速に過渡解析ができる．

並列分散型ブロックLIMの並列性は回路網を構成しているセグメントの数に依存するため実行PE数が制限されるが，より大規模な解析対象が与えられた場合には更にPE数を増やして解析する事が可能である．

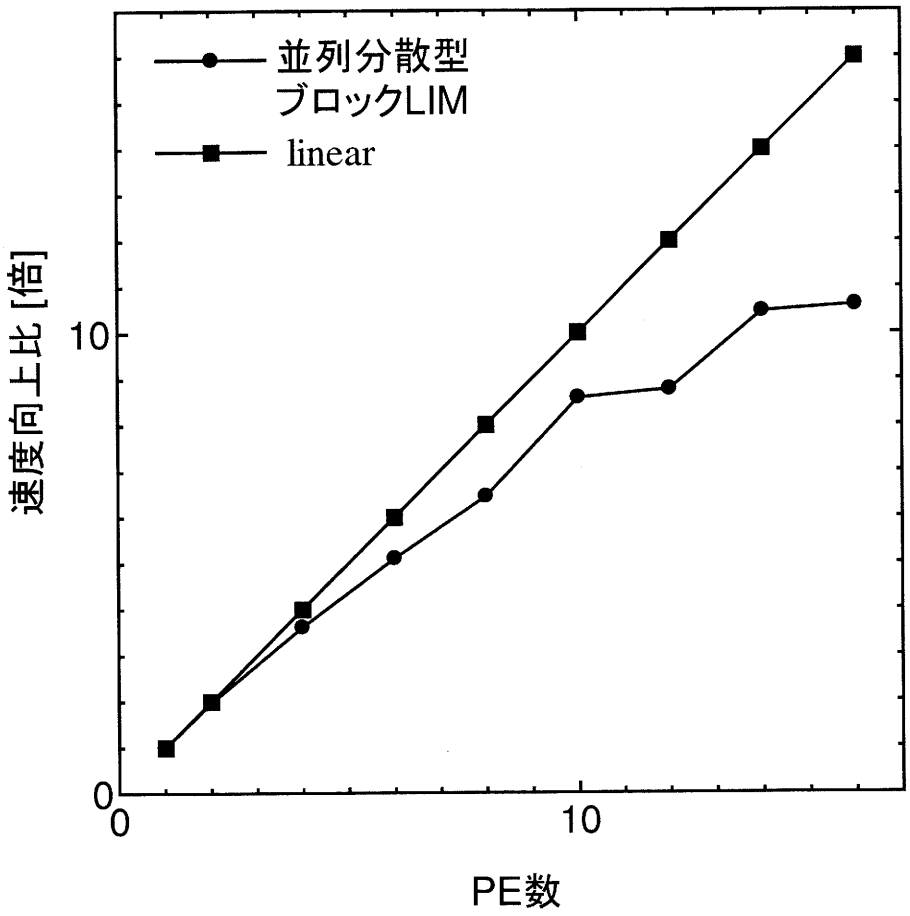


図 4.8: 速度向上の倍率.

第5章　結論

5.1　総括

本研究では，leapfrogアルゴリズムに基づくLIM系のアルゴリズムを特殊なハードウェアに適用させることにより，高速な回路の過渡解析を実現した．LIMは，電流と電圧の時間配置が半時間ステップずつ異なり，更新には既知の値と隣接する変数の値を用いてのみ更新処理を行う．これは係数を行列で表した場合に，電流と電圧の更新時に変数同士の接続関係を示す接続行列が右辺に移項し，対角行列の解を求めることになる．そのため，任意の要素で行列を分割した部分行列に対してPEを割り当てることと，重複する領域を設けた通信処理の非同期処理化が可能であり，並列化に適したアルゴリズムであることが分かる．また，並列化をしたときに現れる同期処理，通信処理に必要とする時間の削減を行え，更新処理の複雑化も少ない効果的な並列計算処理を実現できることを示した．以下に本研究で得られた成果を総括する．

1章では，従来手法の並列化においてオーバーヘッドが発生した場合に効果を十

分に得られない問題点を示し，現在求められている回路シミュレータについて述べた．

2章では，クラウドコンピューティングを用いて大規模なPCクラスタを構築し，並列分散型LIMによる性能の検証について述べた．並列分散型LIMは並列化に適しており，PE数が32の時に25倍以上の高速化が可能な手法である．これは並列分散型LIMが99%以上の処理を並列化可能であり，高い並列性を有していることが分かる．加えて，PCのようなメモリ容量が少ない場合には解析できない対象を与えられたとしても，並列分散型LIMでは各PEが自分の担当する計算領域のみを保持すればよいため，解析を行うことが可能である．そして，LIMが従来手法と比べて100倍以上高速に解析できる事を考えると並列分散型LIMは2500倍以上高速に解析が可能である．また，クラウドコンピューティングを用いた並列計算機では，CPU数と同数のPEまでであれば計算速度を向上させられることを確認した．今回は，CPU数を32までとして検証を行ったが，この数を更に増加させることでより速度の向上を得られると考えられる．クラウドコンピューティングでは，CPU数を左右するインスタンスの数は任意に変更可能である．そのため，利用者がより大規模な並列計算機資源を構築し，並列化したアプリケーションを直ちに実行できる．そのため，必要なインスタンス数の管理と利用時間を適切に用いた場合には，機器の購入などの導入コストを必要としない非常に効果的な計算

機資源の獲得方法である．

3章では，一般的なPCにも搭載されているグラフィック処理用の演算装置であるGPUを用いた高速化について述べた．GPUを利用するには，CPUとは異なるプログラミングモデルを用いて，いくつかの最適化を行う必要がある．この最適化は電流変数と電圧変数それぞれに対して手法が異なり，それぞれの領域分割方法と計算を行うスレッドの割り当て方法を述べた．結果として，GPUを利用することでCPUで計算を行う時よりも一桁以上高速に解析する事ができ，かつ，LIMが単精度浮動小数点を用いても同様の出力波形を得られる．加えて，GPUは倍精度浮動小数点にも対応し，今後も更なる性能の向上が計画されている．このことから，GPUを用いるのに適したアルゴリズムである場合には，劇的な速度向上を常に得られることが期待できる．そのため，ハードウェアアクセラレータとしてGPUを用いることは必須の技術となると考えられる．

4章では，LIMを拡張したブロックLIMアルゴリズムの並列化について述べた．回路構造に依存したPEを用いることになるが，解析対象が50セグメントで構成された強結合伝送線路の場合には，14PE以上を用いる事で10倍以上高速に解析を行うことができることを示した．ブロックLIMがHSPICEと比べて100倍以上高速であるため，並列分散型ブロックLIMは従来手法と比べて1000倍以上高速に解析することが可能である．

本論文で検討したLIM系アルゴリズムの並列化は，並列計算機環境とGPUの利用のいずれの場合で高速に解析する事が可能である．そして，最新のGPUでは倍精度浮動小数点にも対応し，性能の向上も著しい．また，クラウドコンピューティングではGPUを搭載したインスタンスも登場し，今後の大規模並列計算機への適用も期待することができる．このように，計算機環境については，回路の集積化技術の進歩とソフトウェアの充実によって多岐にわたるアプリケーションの高速化への糸口を見せている．そのため，今後もこれらの特殊なハードウェアを適切に用いていくことが必要であると考えられる．

参考文献

[1] A. Vladimirescu, *The SPICE book.* John Wiley & Sons, Inc., 1994.

[2] A. Taflove and S. C. Hagness, *computational electrodynamics: the finite-difference time-domain method*, 3rd ed. Norwood, MA 02062: Artech House, Inc., 2005.

[3] R. D. Berry, "An optimal ordering of electronic circuit equations for a sparse matrix solution," *IEEE Trans. Circuit Theory*, vol. CT-18, no. 1, pp. 40–50, Jan. 1971.

[4] H. Y. Hsieh, "Pivoting-order computation method for large random sparse system," *IEEE Trans. Circuits Syst.*, vol. CAS-21, no. 2, pp. 225–230, Mar. 1974.

[5] I. N. Hajj, "Sparsity consideration in network solution by tearing," *IEEE Trans. Circuits Syst.*, vol. 27, no. 5, pp. 357–366,, May 1980.

[6] E. Lelarasmee, A. E. Rueli, and A. L. Sangiovanni-Vincentlli, "The waveform relaxation method for time-domain analysis o large scale integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-1, no. 3, pp. 131–145, Jul. 1987.

[7] A. R. Newton and A. L. Sangiovanni-Vincentlli, "Relaxation-based electrical simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-3, no. 4, pp. 308–318, Oct. 1984.

[8] A. Odabasioglu, M. Celik, and L. T. Pileggi, "PRIMA: passive reduction-order interconnect macromodeling algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 8, pp. 645–654, Aug. 1998.

[9] H. Ji, A. Devgan, and W. Dai, "Ksim: A stable and efficient RKC simulator for capturing on-chip inductance effect," in *Proc. ASP-DAC 2001*, pp. 379–384, Yokohama, Japan, Jan. 2001.

[10] T.-H. Chen, C. Luk, and C. C.-P. Chen, "INDUCTWISE: Inductance-wise interconnect simulator and extractor," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 7, pp. 884–894, Jul. 2003.

[11] Y. Tanji, T. Watanabe, H. Kubota, and H. Asai, "Large scale RLC circuit analysis using RLCG-MNA formulation," in *Proc. IEEE DATE 06*, Munchi, Germany, Mar. 2006.

[12] P. Sadayappan and V. Visvanathan, "Circuit simulation on shared-memory multiprocessors," *IEEE Trans. Comput.*, vol. 37, no. 12, pp. 1634–1642, Dec. 1988.

[13] C.-P. Yuan, R. Lucas, P. Chan, and R. Dutton, "Parallel electronic circuit simulation on the iPSC® system," in *Proc. IEEE CICC '88*, pp. 651–654, May 1988.

[14] D. C. Yeh and V. B. Rao, "Partitioning issues in circuit simulation on multiprocessors," in *Proc. IEEE ICCAD '88*, pp. 300–303, Nov. 1988.

[15] R. A. Saleh, K. A. Gallivan, M.-C. Chang, I. N. Hajj, D. Smart, and T. N. Trick, "Parallel circuit simulation on supercomputers," *Proc. IEEE*, vol. 77, no. 12, pp. 1915–1931, Dec. 1989.

[16] P.-Y. Chung and I. N. Hajj, "Parallel solution of sparse linear system on a vector multiprocessor computer," in *Proc. IEEE ISCAS '90*, vol. 2, pp. 1577–1580, LA, USA, Jun. 1990.

[17] P. F. Cox, R. G. Burch, D. E. Hocevar, P. Yang, and B. D. Epler, "Direct circuit simulation algorithm for parallel processing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 10, no. 6, pp. 714–725, Jun. 1991.

[18] T. Kage, F. Kawafuji, and J. Niitsuma, "A circuit partitioning approach for parallel circuit simulation," *IEICE Trans. Fundamentals*, vol. 77-A, no. 3, pp. 461–466, Mar. 1994.

[19] T. Kage and J. Niitsuma, "A parallel BBD matrix solution for MIMD parallel circuit simulation," *IEICE Trans. Fundamentals*, vol. 78-A, no. 1, pp. 88–93, Jan. 1995.

[20] P. M. Lee, S. Ito, T. Hashimoto, J. Sato, T. Touma, and G. Yokomizo, "A parallel and accelerated circuit simulator with precise accuracy," in *Proc. IEEE VLSID '02*, Jan. 2002.

[21] K. Gulati, J. F. Croix, S. P. Khatri, and R. Shastry, "Fast circuit simulation on graphics processing units," in *Proc. ASP-DAC 2009*, pp. 403–408, Yokohama, Japan, Jan. 2009.

[22] P. Agrawa, S. Goil, S. Liu, and J. A. Trotter, "Parallel model evaluation for circuit simulation on the PACE multiprocessor," in *Proc. VLSI Design 1994*,

pp. 45–48, Jan. 1994.

[23] D. M. Webber and A. Sagiovanni-Vincentlli, "Circuit simulation on the connection machine," in *Proc. 24th ACM/IEEE Design Automation Conf.*, pp. 108–113, Jun. 1987.

[24] Z. Feng and P. Li, "Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms," in *Proc. ICCAD 2008*, pp. 647–654, San Jose, CA, USA, Nov. 2008.

[25] N. Kapre and A. DeHon, "Accelerating SPICE model-evaluation using FPGA," in *Proc. IEEE FCCM '09*, pp. 37–44, Apr. 2009.

[26] ——, "Parallelizing sparse matrix solve for SPICE circuit simulation using FPGAs," in *Proc. FPT '09*, pp. 190–198, Dec. 2009.

[27] D. Dumlugöl, P. Odent, J. P. Cockx, and H. J. D. Man, "Switch-electrical segmented waveform relaxation for digital MOS VLSI ansd its acceleration on paralllel computers," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-6, no. 6, pp. 992–1005, Nov. 1987.

[28] E. Z. Xia and R. A. Saleh, "Parallel waveform-newton algorithms for circuit simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 4, pp. 432–442, Apr. 1992.

[29] B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS – an MOS timing simulator," *IEEE Trans. Circuits Syst.*, vol. CAS-22, no. 12, pp. 901–910, Dec. 1975.

[30] M. Nishigaki, N. Tanaka, and H. Asai, "Hierarchical decomposition and latency for circuit simulation by direct method," *IEICE Trans. Fundamentals*, vol. E75-A, no. 3, pp. 347–351, Mar. 1992.

[31] G. H. Golub and C. F. V. Loan, *Matrix Computation*, 3rd ed. The Johns Hopkins Univerisity Press, 1996.

[32] J. E. Schutt-Ainé, "Latency insertion method (LIM) for the fast transient simulation of large networks," *IEEE Trans. Circuits Syst. I*, vol. 48, no. 1, pp. 81–89, Jan. 2001.

[33] T. Watanabe, Y. Tanji, H. Kubota, and H. Asai, "Prallel distributed time-domain circuit simulation of power disribution networks with frequency-

dependent parameters," in *Proc. ASP-DAC 2006*, pp. 832–837, Yokohama, Japan, Jan. 2006.

[34] ——, "Fast transient simulation of power distribution networks containing dispersion based on parallel-distributed leapfrog algorithm," *IEICE Trans. Fundamentals*, vol. J90-A, no. 2, pp. 388–397, Feb. 2007.

[35] 井上雄太, 関根惟敏, 浅井秀樹, "並列分散型LIMに基づく高速回路シミュレーション," 第23回エレクトロニクス実装学会春季講演大会, pp. 9–10, 横浜, Mar. 2009.

[36] Y. Inoue, T. Sekine, T. Hasegawa, and H. Asai, "Fast circuit simulation based on parallel-distributed LIM using cloud computing system," in *Proc. ITC-CSCC 2009*, pp. 845–846, Jeju, Korea, Jul. 2009.

[37] ——, "Fast circuit simulation based on parallel-distributed LIM using cloud computing system," *JSTS*, vol. 10, no. 1, pp. 49–54, Mar. 2010.

[38] 井上 雄太, 関根 惟敏, 浅井 秀樹, "GPGPU-LIMに基づく回路シミュレーションとその評価," 信学技報, CAS, vol. 109, no. 199, pp. 37–42, 広島, Sep. 2009.

[39] Y. Inoue, T. Sekine, and H. Asai, "GPGPU-based latency insertiion method : Applycation to PDN simulations," in *Proc. IEEE EDAPS 2009*, Hong Kong, China, Dec. 2009.

[40] 井上 雄太, 關根 惟敏, 浅井 秀樹, "GPGPU-LIM を用いた電源分配回路網の高速過渡解析," 信学論 (C), vol. J93-C, no. 11, pp. 406–413, Nov. 2010.

[41] 井上 雄太, 関根 惟敏, 浅井 秀樹, "並列分散型ブロック LIM による強結合伝送線路の高速過渡解析," 信学技報, CAS, vol. 109, no. 300, pp. 25–30, 名古屋, Nov. 2009.

[42] Y. Inoue, T. Sekine, and H. Aasai, "Parallel-distributed block-LIM-based fast circuit simulation of tightly coupled transmission lines," in *Proc. IEEE ECTC 2010*, pp. 657–662, Las Vegas, USA, Jun. 2010.

[43] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer, "BEOWULF: a parallel workstation for scientific computation," in *Proc. 24th Int. Conf. on Parallel Processing*, pp. 11–14. CRC Press, 1995.

[44] H. Kubota, Y. Tanji, T. Watanabe, and H. Asai, "Generalized method of the time-domain circuit simulation based on LIM with MNA formulation," in *Proc. IEEE CICC 2005*, pp. 282–292, San Jose, CA, Sep. 2005.

[45] H. Asai and N. Tsuboi, "Multi-rate latency insertion method with RLGC-MNA formulation for fast transient simulation of large-scale interconnect and plane networks," in *Porc. IEEE ECTC 2007*, pp. 1667–1672, Reno, NV, May 2007.

[46] T. Sekine and H. Asai, "CMOS circuit simulation using latency insertion method," *IEICE Trans. Fundamentals*, vol. E92-A, no. 10, pp. 2546–2553, Oct. 2009.

[47] "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2/.

[48] S. N. Lalgudi, M. Swaminathan, and Y. Kretchmer, "On-chip power-grid simulation using latency insertion method," *IEEE Trans. Circuits Syst.*, vol. 55, no. 3, pp. 914–931, Apr. 2008.

[49] C. Guiffaut and K. mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas Propag. Mag.*, vol. 43, no. 2, pp. 94–103, Apr. 2001.

[50] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[51] "MPI forum," http://www.mpi-forum.org/.

[52] "CUDA 3.2 programming guide," http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf.

[53] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, pp. 80–113, Mar. 2007.

[54] W. mei W. Hwu, Ed., *GPU Computing gems*, emerald ed., ser. Applications of GPU Computing. Morgan Kaufmann, 2011.

[55] "CUDA ZONE," http://www.nvidia.com/object/cuda_home_new.html.

[56] D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura, "A special-purpose computer for gravitational many-body prblems," *Nature*, vol. 345, pp. 33–35, May 1990.

[57] "OpenCL," http://www.khronos.org/opencl/.

[58] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: Stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, Aug. 2004.

[59] L. D. Smith, T. Anderson, and T. Roy, "Power plane SPICE models and simulated performance for materials and geometries," *IEEE Trans. Adv. Packag.*, vol. 24, pp. 277–287, Aug. 2001.

[60] T. Sekine and H. Asai, "Block-latency insertion method (block-lim) for fast transient simulation of tightly coupled transmission lines," *IEEE Trans. Electromagn. Compat.*, vol. 53, no. 1, pp. 193–201, Feb. 2011.

[61] H. Nakashima, "T2K open supercomputer: Inter-university and interdisciplinary collaboration on the new generation supercomputer," in *Proc. Intl. Conf. Informatics Education and Research for Knowledge-Circulating Society 2008*, pp. 137–142, Kyoto, Japan, Jan. 2008.

謝辞

本研究を進めるにあたり，終始適切なご助言，ご指導を賜りました静岡大学工学部の浅井秀樹教授に深く感謝の意を表します.

そして，本論文の査読をしていただきました静岡大学工学部の大坪順次教授，三浦憲二郎教授，静岡大学電子工学研究所の川人祥二教授に深く感謝の意をあらわします.

また，本論分の作成に当たって多くのご助言を賜りました静岡大学情報基盤センターの長谷川孝博准教授に深く感謝の意を表します.

最後に，これまでの研究において活発な議論に応じてくれた關根惟敏氏をはじめとする浅井研究室のメンバーの皆様にお礼申し上げます.