C-011

命令フェッチコストを考慮した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

Improving the Instruction Fetch Cost of the Code Generation Method in the Circuit Simulation

根本 和宜 † Kazuyoshi Nemoto
宮崎 収兄 † Nobuyoshi Miyazaki

1. はじめに

近年における半導体需要の増加とともに，大規模かつ複雑な回路をより高速に開発，設計することが求められている．これに伴い，電子回路シミュレータの高速化が大きな課題となる．従来より，電子回路シミュレータの高速化手法として，方程式を高速に求解する手法の研究[1]が行なわれており，なかでもコード生成法の研究は，効果的な手法として多くの提案，改良がなされてきた[2)3)]．しかし，コード生成法により生成されるループフリーコードは演算を実行するたびにメインメモリから命令をフェッチする必要があり，この命令フェッチコストが高速実行の妨げとなっている．そこで，中間言語的な命令コードを生成し，これをデータパスを通してロードし，命令キャッシュに収まるサイズのプログラムで繰り返し解釈しながら実行することで高速化する．

2. 電子回路シミュレーションとコード生成法

電子回路シミュレーションは，非線形連立微分方程式で与えられる電子回路の動特性をインプリシット積分やNewton-Raphson 法などを用いて線形化し，線形連立方程式を LU 分解法などを用いて解く．電子回路シミュレーションにおける回路方程式の係数行列の多くは，ゼロ要素を多く含むスパース行列であり，解析中は行列の構造がほとんど変化しないという特長を持つ．コード生成法はこの特長を利用し，LU 分解及び前進代入，後退代入で実行する以下の式 (1)，式 (2) から構成されるコードを生成し実行する．

$$op1 = op1 - op2 \times op3 \quad (1)$$

$$op1 = op1/op2 \quad (2)$$

コード生成法により生成されるコードは，ゼロ要素を含む四則演算のような無駄な演算を除いた，繰り返し演算のない算術代入演算のみのループフリーコードである．

3. コード生成法の問題点と解決策

コード生成法ではループフリーコードを生成するので，実行時に使用する演算命令は一命令につき一度しか実行されない．このため，演算命令を実行する際に命令キャッシュは有効利用されず，毎回メモリから命令をフェッチする必要がある．また，CISC アーキテクチャの汎用プロセッサの多くは，命令をフェッチする際にデコード処理を行うが，このアーキテクチャでは，演算命令をフェッチする度に毎回デコード処理を行う必要がある．つまり，CISC アーキテクチャの汎用プロセッサ上でループフリーコードを実行した場合，メモリからの頻繁な命令フェッチとそれに伴うデコード処理が高速化の妨げになっている．

†千葉工業大学, Chiba Institute of Technology

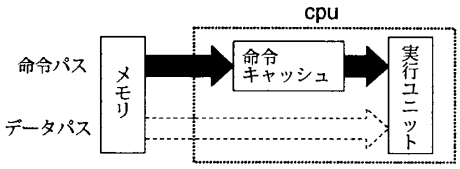

図 1: 命令パスを使用した演算命令の流れ

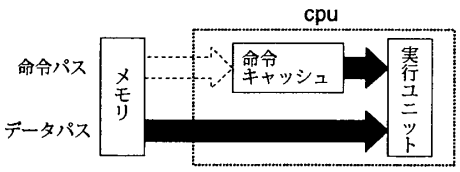

図 2: データパスを使用した演算命令の流れ

そこで本手法では，図 1 に示すように命令パスから命令をフェッチするのではなく，図 2 に示すようにデータパスから実行する命令の情報を持ったデータをロードし，命令キャッシュに収まるサイズのプログラムで解釈しながら実行する．これにより，命令フェッチコストを削減し高速化を図る．本手法は命令キャッシュからの命令フェッチコストの割合が小さいアーキテクチャ上では，より高速に処理が可能であると考えられる．しかし，実行する命令の情報をデータパスからロードするため，データパスに流れるデータの量は従来のコード生成法より多くなる．また，ロードしたデータを解釈する必要があるため，実行する命令の総数も従来の方法より多くなる．このため，命令パス全体のフェッチコストの中で命令キャッシュからの命令フェッチコストの割合が大きければ処理速度が低下する．

4. コードの生成及び実行

本手法では，実行する演算命令をデータパスからデータとしてロードするために，中間言語的な命令コード (以後，中間コードという) を生成する．

中間コードの大部分は，式 (1) 及び式 (2) の演算を示すオペレータとその演算対象を示すオペランドで構成される．例えば，図 3 の係数行列において非ゼロ要素 $mat[9]$ を更新する場合，$mat[9] = mat[9] - mat[8] \times mat[4]$ の演算を行なう．この式は式 (1) と同じ演算式である．そこで，まず生成する中間コードに演算式が式 (1) であることを示すオペレータを格納する．次に，この演算式が必要とするオペランドは $mat[9], mat[8], mat[4]$ である

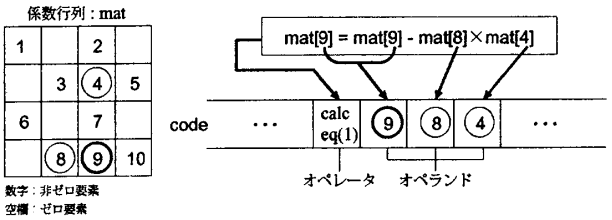

図 3: 中間コード生成例

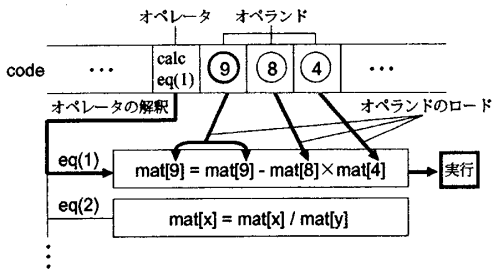

図 4: 中間コードの解釈及び実行例

ので，この要素位置をオペレータの直後に格納する．このように一度の LU 分解及び前進代入，後退代入で実行される全ての演算を中間コードに格納する．また，電子回路シミュレーションでは回路を繰り返し計算するために計算精度が問題となる．このため，対角要素の計算後に要素の値をチェックする必要がある．そこで，対角要素の演算後に要素の値をチェックし，異常がある場合には要素の場所を返すオペレータを対角要素の演算後に格納する．この他，高速化手法として対角要素を逆数化して除算を削減するオペレータ及び繰り返し参照する要素をスカラ化し参照を高速化するオペレータを適切な部分に格納する．

生成した中間コードは命令キャッシュより小さなループ構造のプログラムで解釈及び実行する．図 3 で生成した中間コードを実行する例を図 4 に示す．まず，実行する命令を決定するためにオペレータを読み込む．この場合は式 (1) の演算命令と解釈される．次にオペランドを読み込む．これで演算式 $mat[9] = mat[9] - mat[8] \times mat[4]$ と解釈され，演算が実行される．これを中間コードの最後まで繰り返す．

5. 性能評価

3 節で述べたように，本手法は命令パスのアーキテクチャにより性能が左右されると考えられる．そこで，命令パスのアーキテクチャが異なる汎用プロセッサ，Intel Pentium III 1GHz 及び Pentimun4 3.4GHz で評価する．Pentium III は命令キャッシュがデコーダの前にあり，CISC 命令が格納されるためにキャッシュからの命令フェッチコストが大きい．Pentium III の L1 命令キャッシュ及び L1 データキャッシュは共に 16KBytes，L2 キャッシュは 256KBytes である．このマシンのメインメモリは 512MBytes である．これに対し，Pentimun4 は命令キャッシュがデコーダの後にあり，マイクロオペレーションと呼ばれる RISC 命令が格納されるためにキャッシュからの命令フェッチコストが小さい．Pentimun4 の L1 命令

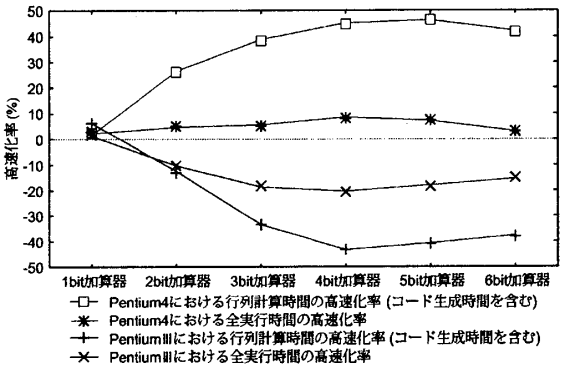

図 5: SPICE3 における高速化率

キャッシュは 12Kμops，L1 データキャッシュは 16KBytes，L2 キャッシュは 1MBytes である．このマシンのメインメモリは 1GBytes である．両マシンとも OS は Linux 2.6.11，コンパイラは gcc 3.4.3，コンパイルオプションは-O2 である．評価には代表的な電子回路シミュレータである SPICE3 を用いた．評価回路には 1bit から 6bit までの NAND ゲートバイナリ加算器を使用した．

図 5 に SPICE3 に対し本手法を実装した場合の高速化率を示す．図 5 から Pentimun4 の行列計算で約 40%，全実行時間で約 10%の高速化が確認できる．これは，本手法により命令フェッチコストが削減されたためと考えられる．これに対し，Pentium III では行列計算で約 20%，全実行時間で約 40%の速度低下がある．これは，Pentium III の命令キャッシュがデコーダの前にあるためと考えられる．また，低 bit の加算器でほとんど高速化されていないのは，全処理に対する行列計算の割合いが小さいために本手法の効果が明確に出なかったと考えられる．

6. おわりに

本稿では，電子回路シミュレーションの回路方程式求解手法として用いるコード生成法において，命令フェッチコストを削減することで高速化する手法を示した．評価の結果，命令キャッシュからの命令フェッチコストが小さなアーキテクチャにおいて，行列計算で約 40%，全実行時間で約 10%の高速化が確認できた．

参考文献

[1] F.Yamamoto, A.Takahashi: Vectorized LU Decomposition Algorithms for Large-Scale Circuit Simulation, IEEE Transactions on Computer-Aided Design, Vol. 4, No. 3, pp. 232-239, (1985).

[2] F.G.Gustavson, W.Liniger, R.Willoughby: Symbolic Generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations, Journal of the Association for Computing Machinery, Vol. 17, No. 1, pp. 87-109, (1970).

[3] Y. Fukui, H. Yoshida, S. Higono: Supercomputing of Circuit Simulation, Proc. Supercomputing '89, pp. 81-85, (1989).