データ駆動計算機 EM-4 における共有二分決定グラフの並列処理について

3D−5

甲村康人† 児玉祐悦‡ 山口喜教‡
†三洋電機(株) ‡電子技術総合研究所
(†E-mail: koumura@edden.hirakata.sanyo.co.jp)

1 はじめに

共有二分決定グラフ (Shared Binary Decision Diagrams、以下 SBDD) [1] は Directed Acyclic Graph (以下 DAG) による論理関数の一表現形式であり、多くの実用的な論理関数を比較的小さな領域で表現でき、種々の演算を実際的な時間で実行できるため、論理設計支援のための様々な応用で用いられている。

本稿では SBDD を分散メモリ方式の並列計算機上で処理するための手法について述べる。本手法では、SBDD の逐次処理に対するオーバーヘッドを最小にしつつ、十分な並列性を引き出すことを目指している。

さらに本手法を我々が開発している高並列データ駆動計算機 EM-4 プロトタイプ上に実現し、評価を行なった。

以下、2節では共有二分決定グラフの並列処理手法について、3節では EM-4 を概観した上で本手法の実現について述べる。4節では本手法の評価を行なう。

2 共有二分決定グラフの並列処理

まず、準備として SBDD を定義する。真偽値を $B = \{0,1\}$ とすると n 入力論理関数全ての集合 F_n は次のように帰納的に与えることができる (関数の Curry 化)。

$$F_0 = B$$
$$F_{k+1} = (B \to F_k)$$

すると節の集合が F_n と同型であるような DAG G_n を次のように得ることができる。

節のラベルが l で、b によってラベル付けられた有向辺が v_b を指すような節を $v(l, \{\xrightarrow{b} v_b, \ldots\})$ と書くことにし、ϕ_k および G_k を次のように定義する。

$$\phi_0(b \in B) = v(b, \emptyset)$$
$$G_0 = \{\, \phi_0(b) \mid b \in B \,\}$$
$$\phi_{k+1}(f) = \begin{cases} \phi_n(f(0)) & \text{if } f(0) = f(1), \\ v(k+1, \{\xrightarrow{0} \phi_k(f(0)), \xrightarrow{1} \phi_k(f(1))\}) & \text{otherwise.} \end{cases}$$
$$G_{k+1} = \{\, \phi_{k+1}(f) \mid f \in F_{k+1} \,\}$$

すると $f \in F_n$ が $f(0)$ と $f(1)$ とで特徴付けられることから明かに G_n は写像 ϕ_n によって F_n と同型となる。

このようにあらゆる n 変数論理関数は G_n 中の節と1対1に対応する。ここで SBDD はシステムが各時点で必要とする関数に対応する節を含み、サクセサを得る演算について閉じている G_n の部分グラフのうちの最小のものであると定義できる。SBDD では、必要/不要となった関数に対応する G_n の節が SBDD 中に動的に生成/消去されながら処理が進む。

SBDD の処理は一般的にグラフを再帰的にたどることによって行なう。従って処理の構造は木構造となるが、グラフは再収れんし得るので単純に処理を展開すると同じ処理が複数回実行されるため効率が悪い。そこでメモ法を用いて演算結果を再利用することが必須となる。もちろん、全ての演算結果を記憶しつづけることは非現実的なので何らかの戦略で管理する必要がある。

従って、典型的な SBDD の処理として SBDD 中の2つの関数 f と g の論理積をとる処理は次のようになる。f か g が葉の場合の処理は自明であるため省いてある。

```
and(f, g)
  if (r == queryMemo(f, and, g)) return r;
  if (f.l > g.l)
    r = get(f.l, and(f.s0, g), and(f.s1, g));
  else if (f.l < g.l)
    r = get(g.l, and(f, g.s0), and(f, g.s1));
  else
    r = get(g.l, and(f.s0, g.s0), and(f.s1, g.s1));
  registerMemo(f, and, g, r);
  return r;
```

ここで f.l は節 f のラベル、f.s0 は 0-successor、f.s1 は 1-successor である。queryMemo は以前に同じ演算が行なわれており、しかもそれを覚えている場合にその結果を返す関数、registerMemo は必要に応じてメモへの登録を行なう関数である。get は現在の SBDD 中に対応する節が存在すればそれを返し、なければ新たな節として SBDD に加える関数である。

ここで、並列化は処理の木を深さ優先で展開するのではなく幅優先で展開することによって得られる。上の例では再帰的に呼び出されている2つの and を並列に実行することに対応する。ただし、グラフが再収れんを持つことから

1. 同じ G_n の節を同時に生成しようとする複数の get 関数同士の排他制御、および
2. メモへの参照と書き込みの間に他のスレッドが同じ演算を開始しないための排他制御

が必要となる。ただし、1. は正しい処理を行なうために不可欠であるが、2. は厳格に行なわれなくとも処理効率に影響するのみで結果の正しさは変わらない。

3 EM-4 による並列化 SBDD の実現

EM-4 はデータ駆動モデルを拡張した強連結枝モデルに基づいた1000台規模の並列処理を目指した高並列計算機であり、現在80台の要素プロセッサ (Processing Element、以下 PE) からなるプロトタイプが稼働している [2][3][4]。PE をつなぐネットワークとしては PE 数を N とした時最大距離が $O(logN)$ であり、演算とは独

Processing of Shared Binary Decision Diagrams on the Highly Parallel Machine EM-4.

Yasuhito KOUMURA†,
Yuetsu KODAMA‡, Yoshinori YAMAGUCHI‡
†SANYO Electric Co., Ltd., ‡Electrotechnical Laboratory

立にパケット転送が行なわれる多段ネットワークであるサーキュラーオメガ網を用いている。クロックは12.5MHzでRISCアーキテクチャにより1クロック1命令の処理が可能。各PEは2クロックに1パケットの出力が可能であり、関数の起動はパケットの到着によってデータ駆動方式で行なわれるため、PE間のデータ転送/関数呼び出し/スレッド生成において非常に高い性能が実現されている。

この優れたアーキテクチャにより前節で述べた並列処理方式を次のような方針で容易に実現することができた。

まずG_nの節集合を各PEに分割(partition)する。SBDD中のある節に対する処理の要求は対応するG_n中の節を担当するPEにPE間関数呼びだしとして実行される。2項演算のように、2つの節の情報が必要な処理については節のラベルの大きい側の節でその処理を行なう。これにより、get関数の排他制御はPE内で行なえば済むため効率的に実現できる。

また、演算結果のメモの空間($G_n \times G_n \times O$、ここでOは演算の集合)についても各PEに分割する。メモ空間はハッシュテーブルによる固定領域とする。メモを参照してそこに求める結果がない場合、そのエントリをロックし、同じ演算を求める他の関数からの参照を結果が書き込まれるまで待たせる。これにより、全く同じ演算を並列に実行してしまうことによる余分な命令実行を防いでいる。

4 評価

前節で述べた方針に従って、EM-4上にSBDDの処理系を実現し、評価した。評価は80台のPEを持つEM-4プロトタイプを用いて行なった。プログラムは並列実行を行なうもの(para版)と行なわないもの(seq版)の2種類について評価した。

主に問題の並列性がどれだけ抽出できるかの評価をおこなうため、比較的性質の理解が容易な次の問題について実験を行なった。

radder(n,m): n-bit加算器($x_{n-1}\cdots x_0 + y_{n-1}\cdots y_0$)のcarry出力関数を求める問題。ここで入力変数の順序は

$$x_0 x_1 \cdots x_m y_0 x_{m+1} y_1 \cdots x_{n-1} y_{n-m-1} y_{n-m} \cdots y_{n-1}$$

で与えられる(左が高位の変数)。ただし、部分問題としてのn-1 bit加算器のcarry出力関数はすでに求まっているものとする。直感的にはこの問題の結果のグラフはmが小さい時高さがnに比例し、幅が2^mに比例する。つまり問題のサイズが$n2^m$程度で、問題の持つ並列性は概ね2^mであるような問題とみなすことができる。

nを50とし、mを0から5まで変化させた時のpara版/seq版の実行時間と、その比を図1に示す。また、Sun4/330(SPARC25MHz, 主記憶32MBytes)による実行時間とEM-4のpara版との比も示した。問題の並列性に応じて並列化による高速化が順調に行なわれていることがわかる。m=7の場合については問題の並列性が大き過ぎ、PEの資源を使い尽くしてしまったため実行できなかった。

5 おわりに

本稿で述べた方式において、並列化による高速化の隘路となる可能性があるのは次の点である。

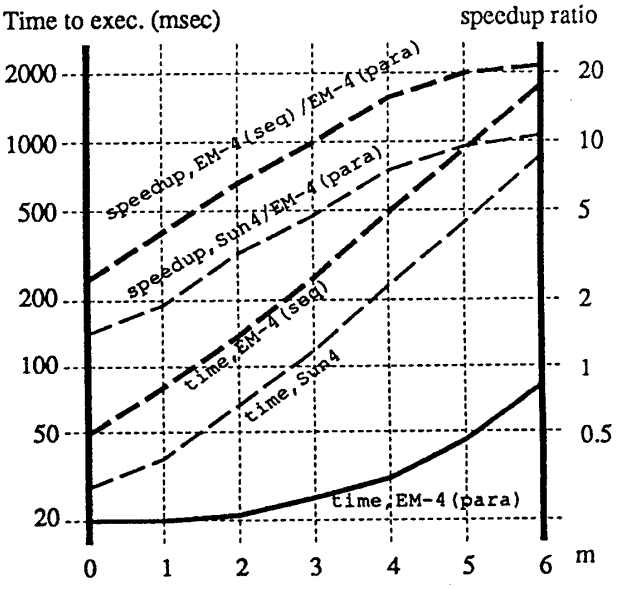

図1: 並列化による速度向上度

- アクセスが集中する節が存在すると1つのPEが隘路となる。このような節はその入力次数が大きいことから検出できるため節の情報を他のPEに複製して持たせるなどの対処が比較的容易に行なえる。
- 演算結果のグラフの幅が大きくない場合、すなわちもともとの問題に並列性がないような場合には本手法では本質的に高速化は望めない。しかしこのような問題の処理時間がシステム全体に占める割合はわずかである。必要ならば、複数の問題を並列に解くことで全体の並列性をさらに引き出すことが可能である。

さらに、現在の実現では、必要以上の並列性を持つ問題に対してはプロセッサの資源を使い切ってしまい結果が得られない。並列性を押えるための管理をいかに実現するかが今後の課題である。

本研究を遂行するにあたりご指導、ご討論頂いた電子技術総合研究所の弓場情報アーキテクチャ部長、島田計算機方式研究室長ならびに研究室の皆様方に感謝します。

参考文献

[1] 湊, 石浦, 矢島. 論理関数の共有二分決定グラフによる表現とその効率的処理手法. 情報処理学会論文誌, Vol.32, No.1, (1991), 77-85.

[2] Yamaguchi,Y., Sakai,S., Hiraki,K., Kodama,Y. and Yuba,T. *An Architectural Design of a Highly Parallel Dataflow Machine.* Proc. of IFIP 89, (1989), 1155-1160.

[3] 児玉, 坂井, 山口. データ駆動型シングルチッププロセッサEMC-Rの動作原理と実装. 情報処理学会論文誌, Vol.32, No.7, (1991), 849-858.

[4] 佐藤, 児玉, 坂井, 山口. 並列計算機EM-4における分散データ構造を用いたマルチスレッドプログラミング. 情報処理学会第84回計算機アーキテクチャ研究会, (1992).