EM-Cによる共有二分決定グラフの並列処理

6M−4

甲村康人† 佐藤三久‡ 児玉祐悦‡ 坂井修一‡ 山口喜教‡
†三洋電機(株) ‡電子技術総合研究所
(†E-mail: koumura@tk.ic.rd.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すでに実用的な処理系が提供されつつある。

より複雑な論理関数を操作する必要性から、より大規模なSBDDを現実的な時間で処理するための研究として、データ駆動型並列計算機EM-4上でのアセンブリ言語による実現についての発表を既に行なった[2]。

本稿ではデータ駆動型計算機EM-4上の並列言語であるEM-Cの上でのSBDDの処理系の実現と、種々の最適化手法について述べる。

2 共有二分決定グラフ

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。

一般にSBDDの処理は関数を表現するグラフを再帰的にたどることによって行なわれる。しかしSBDDのグラフは再収れんを持つため、計算木を単純に展開すると同じ部分計算を複数回実行することになり、効率が悪い。このためメモ法による演算結果の再利用が本質的と言える。また、同じ関数を表現する節の一意性が保証されている必要があるため、演算結果として新たな節を生成する時点で、既に同じものがシステム中に存在しないことを検査しなければならない。

SBDDの処理の並列化の1つの戦略は、計算木を展開してゆく上で、その部分木を並列に実行することである。ただし上に述べたようなSBDDの制約条件から、(1)新たな節の生成を並列に実行しようとしたときに、一意性をどのように保証するか、(2)同じ部分計算の検出と、結果の再利用をどのようにして行なうか、が鍵となる。

さらに、並列計算機によってSBDDを扱う意義を考えた場合、処理速度を向上できると言うことだけではなく、計算機のプロセッサ数に応じてより大規模なSBDDを扱えることが期待される。

本稿では、問題のサイズに関するスケーラビリティを得るためにSBDDの節情報を各プロセッサに分割して持つことにし、節情報を、それを必要とするスレッドが存在するプロセッサに転送することは極力避け、求める節情報が存在するプロセッサにスレッドを移動すると言う戦略をとる。

3 EM-C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高並列計算機である。1000台規模の並列処理の実現を目指しており、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[3][4]。PEを接続するネットワークとしてPE数Nに対し最大距離が $O(\log N)$ であり、演算とは独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/スレッド生成及びスレッドのコンテクスト切替え等において非常に高い性能が実現されている。

EM-CはこのEM-4の特徴を活かし、C言語にユーザが明示的に並列化の制御を指示できるような拡張を行なった並列プログラミング言語である[5]。スレッドの制御をデータのあるプロセッサに移動するというスタイルをも支援しているため、本稿の手法を実現することは容易である。

EM-CによればSBDDの演算は次のように記述できる。以下の記述でparallelは複数の文を並列に実行するためのコンストラクタであり、whereは、PEアドレスまで含んだグローバルポインタを与えて、そのデータのあるPEに制御を移すためのコンストラクタである。以下の関数andで論理関数の表現は節へのグローバルポインタであり、

Processing of Shared Binary Decision Diagrams using the Parallel Programming Language EM-C.

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

whereによって着目する節の存在するPEに処理を移している。ここでf->varは節fの変数を表すラベル、f->s0, f->s1はfのそれぞれのサクセサである。

```
sbdd global *and(f, g) sbdd global *f, global *g; {
  sbdd global *rr;
  if (isLeaf(f) || isLeaf(g)) return leafAnd(f, g);
  if (isValid(rr = queryMemo(f, OP_AND, g)))
    return rr;
  if (f->var < g->var) swap(f, g);
  where (f) dowith (f, g, rr) {
    sbdd global *g0 = g, global *g1 = g;
    sbdd global *r, global *r0, global *r1;
    if (f->var == g->var) g0 = g->s0, g1 = g->s1;
    parallel {
      r0 = and(f->s0, g0);  r1 = and(f->s1, g1);
    }
    r = get(f->var, r0, r1);
    if (rr == INVALID_UNDEF)
      registerMemo(f, OP_AND, g, r);
    return r;
} }
```

上の queryMemo は同一の部分計算が進行中、あるいは過去に行なわれたことを検査するための関数であり、registerMemo はその演算結果の登録を行なうための関数である。この計算結果を記憶しておくためのテーブルは各PEに分散したハッシュテーブルとして次のように実現できる。

```
sbdd global *queryMemo(f, op, g)
  sbdd global *f, global *g; {
  memo global *h = calcMemoHash(f, op, g);
  where (h) dowith (h, f, op, g) {
    if (h->f == f && h->op == op && h->g == g) {
      lock(h->lock);
      unlock(h->lock);
      return h->r;
    }
    if (locked(h->lock)) return INVALID_LOCKED;
    else { lock(h->lock);
      h->f = f, h->op = op, h->g = g;
      return INVALID_UNDEF;
} } }

sbdd global *registerMemo(f, op, g, r)
  sbdd global *f, global *g, global *r; {
  memo global *h = calcMemoHash(f, op, g);
  where (h) dowith (h, r) {
    h->r = r;
    unlock(h->lock);
} }
```

上のコードでは適切なハッシュ関数によって演算結果を検索する処理が各PEに分散されることを期待している。既に同一の部分計算が他のスレッドで進行中の時は、対応するテーブルのエントリがロックされているので、その結果を待つことで同一の部分計算の再利用を行なう。

節の一意性を保証しつつ必要な節を確保するための関数getは、やはり節を各PEに分散したハッシュテーブルによって管理する。

```
sbdd global *get(v, s0, s1)
  sbdd global *s0, global *s1; {
  sbdd global **hash = calcNodeHash(v, s0, s1);
  if (s0 == s1) return s0;
  where (hash) dowith (hash, v, s0, s1) {
    hash に登録された節の中から (v,s0,s1) を検索
    if (節が見つかった) return その節;
    else {
      (v, s0, s1) をキーに持つ新たな節を生成し、
      hash に登録する
      return その節;
} } }
```

ここでは、ハッシュ関数によって、節情報が各PEに平均して分散されることを期待している。ハッシュ関数のキーとなるのは、節の一意性を保証するために必要な情報である。本手法ではハッシュ衝突によって同じハッシュエントリに登録される節は必ず同じPE上に存在することを仮定している。EM-C ではユーザが明示的にプロセッサを手放すことを指定するか、関数呼び出しを行なわない限り、他のスレッドに制御を奪われることがないため、排他制御のための操作は特に行なう必要がない。

4 最適化手法

本稿の手法に従った実現では、(1) ある種の問題は内在する並列性が大き過ぎ、単純な実現では計算機の資源を使い果たす、(2) 負荷のアンバランスにより台数効果が得られない、(3) 粒度が比較的小さいため通信のオーバーヘッドが顕著となる、等の問題点が考えられる。

(1) に関しては、十分なアクティビティが各PEに存在する場面では計算木の展開を幅優先で並列にではなく、深さ優先で逐次に行なう必要がある。この場合各PEにアクティビティが十分存在することをどのように検出するかが課題である。

(2) に関しては、節を均等に分散して PE に持たせるために適切にハッシュ関数を選択する必要がある。また、他の節から多くのリンクを持つ節はアクセスが集中する可能性が高いので、このような節を選択的に他のPEにコピーして持つことができれば特定のPEへの処理の集中をある程度防げる。

(3) に関しては、粒度を大きくするために節情報を親の節が存在するPEにコピーしてスレッド制御のPE間移動を抑えることが考えられる。ただし解き得る問題のサイズとのトレードオフとなる。

5 おわりに

並列プログラミング言語EM-Cによる共有二分決定グラフの処理系について、その並列化手法を中心に述べた。現在、本稿で述べた手法に基づいて、実際にEM-4上に処理系を構築している。今後はEM-Cで記述したことによる抽象度の高さおよび修正の容易さを活かして、アセンブリ言語では困難であった種々の最適化手法を採り入れ、評価してゆく計画である。

謝辞 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] 甲村, 児玉, 山口. 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. 情報処理学会第44回全国大会論文集, 3D-5, (1992).

[3] 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.

[4] 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.

[5] 佐藤, 児玉, 坂井, 山口. 並列計算機EM-4の並列プログラミング言語EM-C 情報処理学会第46回全国大会論文集, 6M-1, (1993).