1T－4 Committed-Choice 型言語 Fleng における静的負荷分割の PIE64 上での実装および評価

村上聡, 荒木拓也, 中田秀基, 小池汎平, 田中英彦
東京大学 工学部

1 はじめに

我々は, Committed-Choice 型言語 Fleng において, コンパイル時に通信量を低減させる静的負荷分割手法を提案し [1], その静的解析手法を開発してきた [2]. 本稿では, 並列推論エンジン PIE64 における静的負荷分割の実現方法について, 実装に重点を置いて報告する.

2 並列推論エンジン PIE64

PIE64 は, 推論プロセッサ UNIRED, 通信プロセッサ NIP, 管理プロセッサ MP を搭載した推論ユニット (IU) を, 相互結合網で接続した構成になっている. この相互結合網は自動負荷分散接続機能を持ち, 自動的に負荷最小の IU を選択して接続することができる.

3 静的負荷分割

Fleng プログラム中で負荷分散を行ないうるのは, ヒープを確保する箇所およびゴールを生成する箇所である. これらについて次のような負荷分散戦略を指定する.

戦略 A 負荷最小の IU を選択する.

戦略 B ローカル IU を選択する.

戦略 C ゴールの引数に与えられたポインタの指す IU を選択する.

戦略 D 同じクローズ内の他の負荷分散ポイントにおいて, 戦略 A によって選択された IU を選択する.

図 1 に, 静的負荷分割の行なわれた append のプログラムを示す.

```
append([A | X], Y, Z @ on(1)) :-                    (a)
        Z = [A | Z1 @ any(2)] @ to(1),              (b)  (c)
        append(X, Y, Z1) @ to(2).                   (d)
append([ ], Y, Z) :- Y = Z.
```

図 1: append のプログラム

図中の下線部は負荷分散戦略を指定するアノテーションであり, それぞれ次のような意味を持つ.

- 下線部 (b) は戦略 A を表し, 変数 Z1 を負荷最小の IU に確保することを意味する.
- 下線部 (d) は戦略 D を表し, 変数 Z1 を確保したのと同じ IU にゴールをフォークすることを意味する.
- 下線部 (a) は戦略 C で参照されるデータを表す. また下線部 (c) は戦略 C を表し, ゴールの引数に与えられている変数 Z と同じ IU にリストセル [A|Z1] を確保することを意味する.

4 リモートヒープアロケーション

我々が提案している静的負荷分割手法を実装する上で, リモート IU 上のヒープをいかにして確保するかがポイントになる. PIE64 では NIP が受信バッファの自動割当て機能を持っており [3], リモートヒープアロケーションはこの機能を使って実現されている.

NIP のこの機能は, UNIRED からは次の命令で利用できる.

```
1) nipwritel   size, src_reg, result_reg
2) nipwritem   size, src_reg, dstIU_reg, result_reg
3) nipwritegl  size, src_reg
4) nipwritegm  size, src_reg, dstIU_reg
```

nipwritel では src_reg の内容が指すメモリ領域から size ワード分を負荷最小の IU に転送する. 受信側 IU では, データを受信すると自動的にヒープを確保して受信したデータを書き込み, 書き込んだアドレスをリプライとして result_reg に返す.

nipwritem では src_reg の内容が指すメモリ領域から size ワード分を dstIU_reg で示される IU に転送する. 受信側 IU の動作は nipwritel と同様である.

nipwritegl では src_reg の内容が指すメモリ領域から size ワードのゴールフレームを負荷最小の IU に転送する. 受信側 IU では, データを受信すると自動的にヒープを確保して受信したデータを書き込んでゴールフレームを作成し, MP にゴールのエンキュー処理を依頼する.

nipwritegm では src_reg の内容が指すメモリ領域から size ワードのゴールフレームを dstIU_reg で示される IU に転送する. 受信側 IU の動作は nipwritegl と同様である.

5 静的負荷分割の実装

Fleng コンパイラは, プログラムに付加されたアノテーションを解釈し, それに対応した UNIRED 用アセンブラコードを出力する.

ここでは, 構造体 (ベクタ) を確保する場合, およびゴールをフォークする場合の, 各負荷分散戦略に対応するコードを示す.

なお, N 番レジスタを %rN と表記している. またオペランドは転送元, 転送先の順になっている.

Implementation and Evaluation of a Static Load Partitioning Method for Committed-Choice Language Fleng on PIE64
Satoshi MURAKAMI, Takuya ARAKI,
Hidemoto NAKADA, Hanpei KOIKE, Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

5.1 データの確保

サイズAのベクタを確保する場合のコードを以下に示す. データをリモートに転送する場合には, あらかじめ転送元(ローカル)IUの一時領域に転送するデータを作成しておいてから, NIPを用いてリモートに転送する. リストを確保する場合もこれに準ずる.

5.1.1 戦略B: ローカルに確保

```
1.    allc      VEC, A+1, %r2
2.    stim      A, [%r2]
3.    ..      /* ベクタの要素を書き込む */
```

これはベクタをローカルヒープに確保し, そのアドレスを2番レジスタに入れる場合のコードである. ローカルヒープにベクタ用の領域を確保し(1行), その要素を書き込む(2,3行).

5.1.2 戦略A: 負荷最小のIUに確保

```
1.    sub.p     %sp, A+1, %sp
2.    mov       %sp, %r1
3.    stim      A, [%r1]
4.    ..      /* ベクタの要素を書き込む */
5.    nipwritel A+1, %r1, %r2
```

これは, ベクタを負荷最小のIUに確保し, そのアドレスを2番レジスタに入れる場合のコードである. まず初めにスタックにベクタ用の領域を確保する(1,2行). そしてその領域にベクタの要素を書き込み(3,4行), それをnipwritel命令で負荷最小のIUへ転送する(5行).

5.1.3 戦略C,D: 指定IUに確保

```
1.    jtag      %r0, ATOM, atom
2.    sub.p     %sp, A+1, %sp
3.    mov       %sp, %r1
4.    jmp       do
    atom:
5.    allc      VEC, A+1, %r1
    do:
6.    stim      A, [%r1]
7.    ..      /* 構造体の要素を書き込む */
8.    jtag      %r0, ATOM, local
9.    nipwritem A+1, %r1, %r0, %r2
10.   jmp       end
    local:
11.   mov       %r1, %r2
    end:
```

これは, ベクタを0番レジスタの指すデータと同じIUに確保し, そのアドレスを2番レジスタに入れる場合のコードである. この場合は, その参照するデータがポインタ型かどうかを調べ(1行), ポインタ型であったならばスタックにベクタの領域を確保する(2,3行). ポインタ型でなければリモートへの転送は行なわないので, ローカルヒープに領域を確保する(5行). 次に, 確保した領域にベクタの要素を書き込み(6,7行), リモートへ転送する場合にはnipwritem命令で指定したIUへ転送する(9行).

5.2 ゴールのフォーク

ゴールをフォークする場合のコードを以下に示す. ゴールをリモートに転送する場合には, 転送元(ローカル)IUのヒープにゴールフレームを作成し, それをNIPを用いてリモートに転送する. なお, ゴールの転送は非同期に行なわれリプライは受け取らない.

5.2.1 戦略B: ローカルIUへフォーク

```
1.    allc      VEC, A+1, %r1
2.    stim      [%r1], A
3.    ..      /* ゴール名, 引数を書き込む */
4.    fork      %r1, mp
```

これは, ゴールをローカルIUへフォークする場合のコードである. ローカルヒープにゴールフレームを作成してから(1-3行), ローカルIUのMPへゴールのエンキュー処理を依頼する(4行).

5.2.2 戦略A: 負荷最小のIUへフォーク

```
1.    allc      VEC, A+1, %r1
2.    stim      [%r1], A
3.    ..      /* ゴール名, 引数を書き込む */
4.    nipwritegl A+1, %r1
```

これは, ゴールを負荷最小のIUへフォークする場合のコードである. この場合は, ローカルヒープにゴールフレームを作成してから(1-3行) nipwritegl命令を使って負荷最小のIUへ転送する.

5.2.3 戦略C,D: 指定IUへフォーク

```
1.    allc      VEC, A+1, %r1
2.    stim      [%r1], A
3.    ..        /* ゴール名, 引数を書き込む */
4.    jtag      %r0, ATOM, local
5.    nipwritegm A+1, %r1, %r0
6.    jmp       end
    local:
7.    fork      %r1, mp
    end:
```

これは, ゴールを0番レジスタの指すデータと同じIUへフォークする場合のコードである. まず初めにローカルヒープにゴールフレームを作成する(1-3行). 次に参照するデータがポインタ型かどうかを調べ(4行), ポインタ型であったならばnipwritegm命令を使って指定したIUへ転送する(5行). ポインタ型でなければローカルIUへフォークする(7行).

6 おわりに

この実装法では, 同じIUへ転送するデータであってもそれぞれ別個に転送しているので, 今後の課題として, 転送するデータを一括して転送することで, 通信のオーバーヘッドをさらに削減するという手法が考えられる.

参考文献

[1] 日高, 小池, 舘村, 田中. 実行プロファイルに基づくコミッティッドチョイス型言語の静的負荷分割手法. 情処論, Vol.32, No.7, pp.807-816, 1991.

[2] H.Nakada, T.Araki, H.Koike, H.Tanaka. A Fleng Compiler for PIE64. Proceeding of PACT '94, 1994.

[3] T.Shimizu. Design Specification of Network Interface Processor (Volume1). Technical Report TRIE-92-1, 東大・情報工学, 1992.