

異機種分散環境上での Dcaml ネイティブ コンパイラの設計と実現

6 E - 7

浅野 貴史 牛嶋 哲 脇田 建 佐々 政孝

東京工業大学 理学部 情報科学科

1. はじめに

分散プログラミング言語 Dcaml は、プログラミング言語 ML の方言である Objective Caml (Ocaml) [1] を異機種分散環境に対応させた分散プログラミング言語である。Dcaml の特徴は、ソフトウェア分散共有記憶 (S-DSM) によって実現される分散透明性と、関数閉包の遠隔起動である [2]。

本稿では、Dcaml のネイティブコンパイラの異機種分散環境における実装について述べる。Dcaml ネイティブコンパイラの実装にあたっては、Dcaml のすべてのデータに分散透明にアクセスできることを保証した。このため、Dcaml の実行時システムで実現されるソフトウェア分散共有記憶は、関数閉包を含めたすべてのデータをアーキテクチャの異なる計算機の間で共有できるように設計されている。さらに、実行時システムのデータ表現に遠隔ポインタを加えることにより、書き換え可能 (mutable) データの一貫性を保証した。

以降では、Dcaml の実装方法、およびその性能評価実験について述べる。

2. ソフトウェア DSM

ソフトウェア DSM を異機種間上で実現するためには、データの一貫性を保ちつつ、データ送受信に伴うデータ変換が必要である。

2.1 データの一貫性

データの一貫性の維持は、各ノードに分散しているデータが、同一内容であることで保証される。ML には、mutable データと書き換え不可能 (immutable) データがある。immutable データは、データの一貫性維持に影響がないので、複製を作ってもよい。mutable データは複製を作ると、データへの書き込みが起こったときに複製を所有している各ノードに変更を反映することが必要である。完全に反映される前にデ

ータへのアクセスが起こるとデータの不整合が生じてしまう。よって、データの変更がすべてのノードにとって同期的に起こるシステムが、もっとも信頼性が高い。

Dcaml 実行時システムでは信頼性の高いデータの一貫性を保証するために、mutable データは複製を作らずに、データの実体はひとつのノードだけが保持していることにする。他のノードでは、そのデータは実体を保持しているノードへの遠隔ポインタとする。

2.2 遠隔ポインタ

ここでは、遠隔ポインタの Dcaml 実行時システム内部での実装について述べる。mutable データは、送信側でそのデータを遠隔参照表に登録する。実際に送信するのは、表の登録した場所を識別する番号である。受信側で、送信してきた参照テーブルの登録番号と送り主ノードの組を遠隔ポインタのデータ表現として確保する。このデータへのアクセスが起こると、遠隔ポインタのデータ表現から、データ保持ノードを判別し、そのノードとメッセージ通信を行い、データの書き換え、参照を行う。

2.3 データの変換

異機種分散環境で Dcaml のデータを送受信するために、Dcaml ネイティブコンパイラ間で統一の外部データ表現を定義した。通信する際に、送信側でそのデータを外部データ表現に変更し、受信側でアーキテクチャ依存の内部データ表現に逆変換する。

3. 関数閉包の遠隔起動

Ocaml の実行時システムにおいて、関数閉包は、閉包を実装する命令列の先頭番地と環境の組からなる。環境は、自由変数の値をあらわすデータ列として表現されているので 2 節で述べた方法で変換を行えば良い。しかし、異なるアーキテクチャ間では命令列のアドレスが異なるため、そのアドレスをそのまま送信することはできない。そこで、Dcaml ネイティブコンパイラは、命令列のアドレスとそのアドレスに対応する関数識別番号 (関数 ID) の対応表 (関数表) を用意

し [3]、Dcaml 実行時システムは、この関数表を利用して、命令列のアドレスの外部表現として関数 ID を使う。こうして異機種間での関数閉包の受け渡しが可能となる。関数閉包は送信側で上述のデータ変換を行い、目的ノードに送信される。そして、受信側で新たなスレッドとして起動される。

4. メッセージ通信

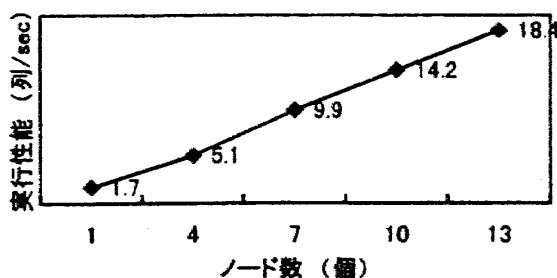
関数閉包の遠隔起動を含む Dcaml アプリケーションを実行する各ノードでは、複数のスレッドが動くことになる。そのため、複数のスレッドからメッセージ通信を利用できれば、メッセージ通信にかかる時間を少しでも減少出来る。Dcaml 実行時システムにおいては、関数閉包の遠隔起動の要求と遠隔ポインタ参照のときにメッセージ通信が起こる。

しかし、我々が現在利用しているメッセージ通信ライブラリである MPI、PVM では、スレッドにメッセージを送信することが出来ない。さらに、受信はそれぞれのメッセージの種類—Spawn、Rcall、遠隔ポインタアクセスの要求—を区別し、種類に応じた処理をしなければならない。これらの問題点の簡単な解決策として、メッセージ受信は各ノードごとに常時起動している1つのスレッド（メッセージ受信スレッド）に任せる。メッセージ受信スレッドはメッセージを受信し、メッセージの種類に応じて、Spawn（Rcall）ならば起動要求された Dcaml スレッドの起動、遠隔ポインタの参照ならば要求発行側が参照したいデータを返信し、更新ならば要求データを更新する。問題点としては、ある一つのノードに他のノードからのメッセージが集中した場合でもメッセージ受信スレッドは一つなので、メッセージが直列化してしまう。このため、このノードがボトルネックになってしまう。

5. 性能評価

Dcaml ネイティブコンパイラの性能評価のために、フラクタルの一つである Mandelbrot 集合を計算するプログラムを用いた ([2]の図2)。15 ノードの Sparc-Station クラスタ (Solaris2.5.1) 上で実行した結果を示す (図1)。このプログラムは列の計算をノードに配分している。2 次元配列のサイズは 500×500 である。グラフの横軸は計算を分散させたノード数、縦軸は単位時間あたりに計算した列の数から割り出した実行性能である。この結果は、良好な台数効果が得られている。

図1 分散 Mandelbrot 評価実験の結果



6. おわりに

Dcaml ネイティブコンパイラにおける S-DSM について述べた。この DSM の実現法は効率面を考えていない素朴な実装である。そのため、mutable データの数が多いほど、メッセージ送信が起こる回数が多くなる。メッセージ通信は計算に比べ非常にコストが大きいので、通信回数が多いプログラムでは高速化が望めなくなってしまう。今後は mutable データの多寡に左右されないように、DSM の工夫が課題である。また、Dcaml が使いやすく十分な記述性を向上する得るために、ParaML [4]のような明示的なデータパレルプリミティブ、分散排他制御機構などを提供することを検討している。

謝辞

実験環境を提供してくださった本学松岡研究室のみなさんに感謝いたします。この論文の草稿を読んでくださった大島芳樹さんに感謝いたします。

参考文献

- [1] Xavier Leroy. "The effectiveness of type-based unboxing" Workshop types in Compilation, 1997.
- [2] 脇田、牛嶋、浅野、佐々. 異機種分散環境上のアプリケーション開発環境 Dcaml システムの構想, IPSJ 56-6E-06, 1998.
- [3] 牛嶋、脇田、浅野、佐々. 異機種分散環境上での Dcaml バイトコードの設計と実現, IPSJ 56-6E-08, 1998.
- [4] P.Bailey, M.Newey, D.Sitsky & R.Stanton "Supporting Coarse and Fine Grain Parallelism in an Extension of ML" in CONPAR, pp. 693-704, 1994.