異機種分散環境上での Dcaml バイトコードコンパイラの設計と実現†

6 E − 8

牛嶋 哲　浅野 貴史　脇田 建　佐々 政孝

東京工業大学

1. はじめに

異機種分散環境で動作するソフトウェアの構築は困難な作業である。それは異機種分散環境に起因するさまざまな障壁—アーキテクチャ・データ表現の違い、適切な通信プロトコルの設計・実現など—を克服するための支援が十分ではないからである。

これらの障壁を克服するための支援として、われわれは分散言語 Dcaml を開発している。Dcaml は以下の特徴を持つ [1]。

- Dcaml 言語は関数型言語 Objective Caml[3] を拡張したものであり、関数閉包を一級の値として扱うことができる。
- ソフトウェア分散共有記憶 (S-DSM) および関数閉包を遠隔起動するための基本操作によって分散透明なプログラミングモデルを提供する。
- 2 種類のコンパイラを提供する。実行効率が重視される実用的なアプリケーションにはネイティブコンパイラを用いる。バイトコードコンパイラはより柔軟な設定を支援するために用いる。

バイトコードをネイティブコードと共存させることができればこれら 2 種類のコンパイラを有効に活用することができる。本稿では Dcaml においてバイトコードをネイティブコードと共存させるために要求される機能およびその実現方法を述べる。

また、現在 Dcaml アプリケーションは基本的に同一のソースセットからコンパイルされたプログラムが分散環境上の複数ノードで動くことを前提としているが、より緩い制限の設定を支援するための構想についても述べる。

2. 共通外部表現の必要性

異機種分散環境において S-DSM を介してデータを共有するためには、データの外部表現を定める必要がある。ノード間でデータを送受信する際、送信ノードでデータの内部表現を外部表現に変換して送り、受信ノードでその外部表現を内部表現に変換することによって、内部表現の違いを吸収する。

内部表現の違いは大きく 2 種類に分類できる。一つは基本データの表現形式の違いである。ネイティブコードの場合基本データの表現形式は一般に機種ごとに異なる。一方バイトコードの場合これは機種に依存しない。もう一つは複合データ (配列、レコード、関数閉包など) の内部構造の違いである。バイトコード同士またはネイティブコード同士では内部構造は機種によらず共通であるが、バイトコードとネイティブコードとでは関数閉包の内部構造が異なる。

バイトコードをネイティブコードと共存させる上では、関数閉包をどのように共有するか、つまり関数閉包の外部表現をどのように定めるのが問題となる。これについて次節で述べる。なお、その他のデータの共有はネイティブコードの場合と同様の手法によって実現すればよい [2]。

3. 関数閉包の共有

関数閉包を共有するためには、以下の相違点を吸収するような外部表現を定める必要がある。

- 関数のコードが機種・コンパイラによって異なる。
- 関数閉包の内部構造がコンパイラによって異なる。

3.1 関数コードの外部表現

関数のコード (命令列) は機種・コンパイラによって異なる。したがって関数のコード自体をそのまま外部表現として用いることはできない。この問題に対処するために、Dcaml では関数のコードはすべてのノードに既に存在することを仮定する。この仮定によって、コードの代りにコードへの参照 (コードの位置) を外部表現として用いることが可能となる。

関数コードへの参照としてそのアドレスを使うことができれば好都合であるが、これは機種・コンパイラによって異なる。そこで関数コードのアドレスに対してこれと 1 対 1 に対応する ID (関数 ID) を割り当て、関数 ID 自身は機種・コンパイラに依存しないようにする。すると関数 ID を関数コードの外部表現として用いることができる。

関数 f のコードに割り当てられる関数 ID I_f は (1) f が属するモジュールの名前および (2) f に割り当て

---

†Design and implementation of the Dcaml bytecode compiler
T. Ushijima, T. Asano, K. Wakita, and M. Sassa
Tokyo Institute of Technology

られた、モジュール内で一意な番号で構成される。このような構成になっているのは、モジュールごとの分割コンパイルおよび後述する非 SPMD 設定を可能にするためである。Dcaml コンパイラは関数コードのアドレスと関数 ID の間の相互対応表をコンパイル時に生成し、これを実行形式ファイルの中に埋め込むようにしている。

3.2 関数閉包の内部構造

Dcaml が扱う関数閉包の内部構造は、バイトコードとネイティブコードとで異なる。これは関数の部分適用を実現するための方法が異なる、という事実を反映している。

バイトコードの場合、引数を n 個 ($n \geq 1$) 取り、値が $v_1, \ldots, v_k$ であるような k 個 ($k \geq 0$) の自由変数を持つ関数 f を m 個 ($0 \leq m < n$) の引数 $a_1, \ldots, a_m$ に適用した結果として得られる関数閉包 F_m^n は

$$F_m^n = \begin{cases} (e_f, v_1, \ldots, v_k) & m = 0 \\ (e_f - 1, F_0^n, a_1, \ldots, a_m) & m > 0 \end{cases}$$

とあらわされる。ここで e_f は f のコードのアドレスをあらわすものとする。ただしバイトコードの場合関数には入口が 2 つあり、部分適用によって得られる関数閉包をさらに適用する場合には 1 つ手前の命令から実行する。この命令は既に与えられた引数をスタックに積み直す (引数はスタック経由で渡される)。

一方ネイティブコードの場合 F_m^n は

$$F_m^n = \begin{cases} (e_f, n, v_1, \ldots, v_k) & n = 1 \\ (c_m^n, n, e_f, v_1, \ldots, v_k) & n > 1, m = 0 \\ (c_m^n, 1, a_m, F_{m-1}^n) & n > 1, m > 0 \end{cases}$$

とであらわされる。ここで c_m^n はコンパイラが生成する補助関数のアドレスである。c_m^n ($m < n-1$) は F_m^n から F_{m+1}^n を作り、c_{n-1}^n が e_f を実行する。

3.3 関数閉包の外部表現

上記 2 項から、任意の関数閉包を構成するための本質的な情報は $(I_f, n, (v_1, \ldots, v_k), (a_1, \ldots, a_m))$ という 4 つ組であることがわかる。そこで関数閉包をあるノードから他のノードに送る場合には、送信ノードでは関数閉包からこの情報を直列化して外部表現に変換した上で送り、受信ノードではその外部表現を元にしてそのノード用の内部表現を用いて関数閉包を構成する。

4. 非 SPMD 設定の支援

Dcaml アプリケーションは基本的に SPMD (single program/multiple data) 風に動作する。つまり、同一のプログラムが複数ノード上で動く、という前提で作成される。これは主として等価な関数コードがすべてのノードに存在していることを保証したいからである。

しかし実際にある関数閉包を遠隔起動するには、その関数およびその関数から直接・間接に呼び出される関数について自ノードと遠隔ノードに同一のソースコードからコンパイルされた等価な関数コードが存在していれば十分である。そこで各ノードでは部分的に異なるプログラムが動いていてもよいことにする代りに、必要な関数コードが存在するかどうかの検証を実行時に行なうことが考えられる。

Dcaml ではこのような等価性の検証はモジュール単位で行なう。基本的な方針としては、コンパイル時に各モジュールのソースコードのダイジェスト (たとえば MD5 による) を算出しておき、実行時に各ノードが持つダイジェストを比較することによってコードの等価性を判断することにする。関数閉包を遠隔起動する際には、実行される関数定義を含むモジュール、およびそのモジュールが直接・間接に参照しているモジュールについて上記の検証を行なう。

5. まとめ

Dcaml においてバイトコードをネイティブコードと共存させるための関数閉包の共有手法について述べ、非 SPMD 設定を支援するための枠組を示した。ネイティブコードとの共存については実装はほぼ完了しており、非 SPMD 設定を支援するための機構を実装している段階である。

さらに、動的リンク機能を提供することによってより柔軟な分散アプリケーションを構築することが考えられる。この場合にはリンクする際に上記の等価性検証を行なう。動的リンク機能を導入するためには、どのようにしてセキュリティを提供するを考慮することが重要になる。これは今後の課題である。

参考文献

[1] 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. 情報処理学会第 56 回全国大会講演論文集 (1998).

[2] 浅野, 牛嶋, 脇田, 佐々. 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. 情報処理学会第 56 回全国大会講演論文集 (1998).

[3] Leroy, X. Objective Caml. Available at http://pauillac.inria.fr/ocaml/ (1997).