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

6 E − 6

脇田 建 牛嶋 哲 浅野 貴史 佐々 政孝
東京工業大学 数理・計算科学専攻

1. はじめに

ネットワーク技術の進展にともない、コンピュータネットワークの利用者数が爆発的に増加しつつある。分散環境で動作する良質のソフトウェア(分散システム)を大量供給することは、ネットワーク社会が期待する、簡易で効率的なサービスを提供するために不可欠である。しかし、分散システムの構築は依然として困難であり、優秀な技術者の経験に依存するところが大きい。

われわれの目標は、分散プログラミング言語を開発し、それによって分散システムの開発を通常のものと同程度に容易にすることである。Dcaml は既存のMLの処理系 Ocaml [L97]を分散環境に対応させたものであり、分散共有記憶と関数閉包の遠隔起動に基づく通信機構を特徴とする。前者は関数閉包を含めた任意のデータに対して分散透明な仮想記憶空間を実現し、後者はその上でマルチスレッドと同じインタフェイスを提供する。Dcaml は Ocaml の豊富なライブラリ資産をそのまま受け継ぎ、高効率のコンパイラ、バイトコードコンパイラなどの処理系を提供する。

2. 異機種分散システムの開発

異機種分散環境で動作するソフトウェアの開発を困難にする要因として以下が挙げられる。

異機種性 分散環境は、アーキテクチャが異なるさまざまな計算機から構成される。アーキテクチャの異なる計算機間で情報を共有するためには、相互にデータを変換することが不可欠である。

ソフトウェア階層 分散プログラミングでは、システムコール、並行プログラミング、通信ライブラリなどに関する各種ソフトウェアの詳細な知識が要求される。

通信プロトコル 通常、分散システムでのデータ授受にはメッセージ通信が用いられる。適切なメッセージ通信のためには、送受信の際のプロトコル、すなわちメッセージの送受信の順序とデータ型を正確に一致させる必要がある。通信プロトコルの静的な検査は困難なため、プロトコルバグを取り除くために、しばしば試行錯誤が求められる。

これらの問題点は、次節で述べる Dcaml の設計方針によって解決される。

3. Dcaml システム

われわれは、Ocaml のコンパイラと実行時システムを変更することにより、分散言語 Dcaml を開発している。

3.1 プログラミングモデル

Dcaml の実行時システムは、ソフトウェア分散共有記憶(S-DSM)を実現する。異機種性に伴うデータ変換はすべて S-DSM が適宜処理する[AU+98]。Dcaml アプリケーションは S-DSM 上で SPMD 的に動作し、S-DSM を通して関数閉包(closure)を含めたすべての ML データを共有している。

Dcaml は分散プログラミングのために関数閉包の遠隔起動を基礎とした二つのプリミティブ spawn と rcall を提供する。いずれの関数もノードと関数閉包を引数に取り、ノードに関数閉包を送り、それを新たなスレッドとして実行する。spawn が非同期的にスレッドを起動するのに対して、rcall スレッドの実行終了と同期し、その結果を rcall の値とする。両者のインタフェイスは以下の通りである。

spawn: node → (unit → α) → unit
rcall: node → (unit → α) → α

S-DSM 上で関数閉包の遠隔起動を用いたプログラミングの利点は、明示的なデータ変換が不要である点と送受信プロトコルの実現に悩まされることがない点である。後者は、特に、プログラム全体の型を静的に検査できる点で重要である。

S-DSM と関数閉包の遠隔起動はマルチスレッドプログラミングモデルのセマンティクスを異機種分散環境上に実現していると見なせる。なぜならば、前者はメモリ空間を、後者はスレッド起動を分散環境で実装すると見なせるからである。

S-DSM は Ocaml の実行時システムのセマンティクスを完全に維持している。このため、Dcaml は Ocaml とソースレベル互換である。したがって、Ocaml で提供されるデータ型(配列、参照、関数閉包、オブジェクトなど)とライブラリ(正規表現、Unix システムコール、Tk、DBM、パーザー生成器など)がそのまま Dcaml で利用できる。

3.2 処理系

Dcaml の処理系として、ネイティブコンパイラ[AU+98]とバイトコードコンパイラ[UA+98]を提供する。ネイティブコンパイラはアセンブリコードを出力する。このため、実用アプリケーションに求められる高い実効性能が得られる。一方、バイトコードコンパイラは特定のアーキテクチャに依存しないバイトコードを出力する。このバイトコードはバイトコードインタプリタにロードされ、実行される。Java インタプリタと同様に、実行時にバイトコードのロードができるため、ネイティブコンパイラを用いた場合に比べ、動的なアプリケーションの開発が可能となる。

Dcaml: a system for development of heterogeneous distributed applications
K. Wakita, T. Ushijima, T. Asano, and M. Sassa
Tokyo Institute of Technology,
Ken.Wakita@is.titech.ac.jp

図 1 Dcaml アプリケーションのアーキテクチャ

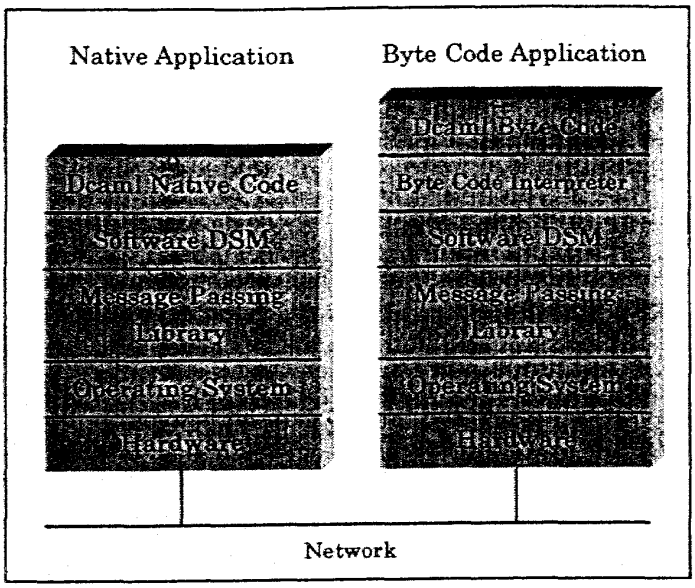

図 2 分散 Mandelbrot 計算

```
exception Done
let index = ref 0   (* next column to be computed *)
let new_job () =
  let i = !index in
  if i < size then (incr index; fun () -> calc_Mandelbrot i)
  else raise Done
let master node =
  try while true do
      let column = rcall node (new_job ()) in
      draw_column column
    done
    with Done -> ()
  let main nodes =
    foreach node in nodes do
      (fun node -> spawn current_node
                   (fun () -> master node))
    done
```

図1は Dcaml 処理系のソフトウェアアーキテクチャを示している。Dcaml アプリケーションは S-DSM 上で実行する複数のマルチスレッドプロセスより構成される。S-DSM 層で、アーキテクチャ依存性とネイティブコードとバイトコードの差異を吸収し、アプリケーションコードの特定の実行環境への依存性を排除する。S-DSM では主に、アプリケーション内部のデータと外部表現の間の変換、関数閉包の遠隔起動に伴うスレッド起動、遠隔データアクセスの処理を実装する。ノード間通信には、MPI や PVM などの標準的な通信ライブラリを用いている。

4. 記述例

図2は Dcaml を用いた記述例として Mandelbrot 図形を計算するプログラムのドライバルーチンを示している。この計算は二次元配列の各要素の値を計算するものである。各要素の計算は互いに独立であり、容易に並列性を抽出できる。図2では、配列の各列(i)をノードに配布し、計算した結果をもとに画面表示を行っている。各ノード用のマスタースレッドを起動するのに spawn を、ノードに計算する指示を与えるために rcall を用いている。

5. 関連研究

ML の分散並列実装に関する研究としては ParaML [BN+94]、dML [OK94]などが知られている。ParaML は Dcaml と同様に SPMD 環境での分散実行モデルであるが、異機種分散環境には対応していない。また、メッセージ送信に対する静的な型検査ができない等の問題を抱えている。dML は、非 SPMD 環境における通信とそれに対する型システムを提案している。遠隔手続き呼び出しをサポートしているが、バックコールをするために実行性能に問題が残る。

6. 現状と今後の課題

現在、Dcaml のネイティブコンパイラが SUN SPARCstation (UltraSPARC, Solaris 2.5.2)と DEC AlphaStation (Alpha 21164, Digital UNIX V 4.0B)から構成されたワークステーションクラスタ上で利用できる。現在、SGI O2 (R10000, IRIX 6.3)、PC (Pentium, Linux with Linux thread)上への移植を試みている。バイトコードコンパイラについては設計を終え、実装を開始している。

Dcaml コンパイラは、アプリケーションが SPMD 的に動作すること、および、アプリケーションがすべてのノードで同時に実行を開始することを仮定している。前者はコードを送らずに関数閉包の遠隔起動を実装する上で必要であり、後者は MPI1 の制限に由来する。しかし、これらの仮定は実用的なシステムの構築には制約が強すぎ、クライアントサーバモデルの分散システムを実装する上で障害となる。現在、PVM の spawn を利用することで動的にプロセスを生成することを検討している。さらに、SPMD の制約を緩め、同一モジュールを持つノード間での通信を許す枠組みを設計し実装している[UA+98]。

参考文献

[AU+98] 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、IPSJ 56-6E-07、(1998).

[BN+94] Baily, Newey, Sitsky, and Stanton, "Supporting Coarse and Fine Grain Parallelism in an Extention of ML," in CONPAR '94, pp. 693-704, (1994).

[L97] X. Leroy, "The effectiveness of type-based unboxing," workshop on types in compilation, (1997).

[OK94] Ohori and Kato, "Semantics for Communication Primitives in a Polymorphic Language," POPL 93, pp. 99-112, (1993).

[UA+98] 牛嶋、浅野、脇田、佐々．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、IPSJ 56-6E-08、(1998).