ABCL/EM-4:データ駆動並列計算機上の並列オブジェクト指向言語処理系の実装と評価

八 杉 昌 宏† 松 岡 聡†† 米 澤 明 憲†††

並列オブジェクト指向計算は，計算モデルとしての強力な表現力と自然なMIMD的並列性を持っている．しかしながら，従来のMIMD並列計算機では，(1)リモート通信に要するコスト，(2)オブジェクト間のコンテキストスイッチに要するコスト，が非常に高く，並列オブジェクト指向計算はその性能を十分に発揮できないという問題があった．我々の提案したソフトウェア/ハードウェアアーキテクチャ（**ABCL/EM-4**）は，RISC的アプローチ，すなわち，ハードウェアは簡潔で高速化可能なものとし，複雑な処理は最適化されたソフトウェアで行うというアプローチにより，リモートメッセージパッシングやコンテキストスイッチのコストを，逐次の手続き呼び出しに匹敵するオーダーに削減することを可能にした．本稿では，データ駆動並列計算機EM-4をターゲットとして開発したABCL/STコンパイラを用いて，EM-4実機による処理系の評価を行ったので報告する．

ABCL/EM-4: An Implementation and Evaluation of a Concurrent Object-Oriented Language System on a Data-Driven Parallel Computer

Masahiro Yasugi,† Satoshi Matsuoka††
and Akinori Yonezawa†††

Concurrent object-oriented computing provides modeling power and natural MIMD parallelism through concurrency of objects. Unfortunately the high costs of inter-node message passing and intra-node scheduling make the implementation of concurrent object-oriented languages inefficient. To overcome these problems, we have proposed a new software/hardware architecture (ABCL/EM-4) which realizes efficient parallel execution of programs based on a concurrent object-oriented computation model. Our ABCL/EM-4 achieved high performance with a combination of simple and fast hardware mechanisms and sophisticated software design, where the cost of a remote message-passing and/or a context-switch can be almost comparable to that of a sequential procedure call. This paper shows the evaluation results with the developed ABCL/ST compiler on the data-driven parallel computer EM-4.

1. はじめに

並列オブジェクト指向計算モデル[1),12)]は，(1)問題の自然な表現，(2)プログラムのモジュラリティ(3)MIMD的並列性，といった利点や特徴を兼ね備えた強力な計算モデルである．しかしながら，従来，並列オブジェクト指向計算は，実際の計算機においてその性能を十分に発揮できないという問題があった．従来のMIMD並列計算機に関していうと，その主な原因には，(1)リモート通信に要するコストが非常に高いこと，(2)オブジェクト間のコンテキストスイッチに要するコストが高いこと，が挙げられる．

例えば，並列オブジェクト指向計算での返答付きリモートメッセージパッシングでは，(1)ノード間メッセージ送信・転送・受信（2回），(2)コンテキストスイッチ（2回）のコストが，メソッドルックアップ・メソッド実行のコストに追加されたオーバーヘッドとなる．これらのコストを担う機能は逐次言語にはないため，あらたに追加する必要があり，従来は，OSやライブラリの提供する機能を利用することが多く，オーバーヘッドを大きくしてきた．例えば，メソッドルッ

† 神戸大学工学部情報知能工学科
Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University
†† 東京工業大学大学院情報理工学研究科数理・計算科学専攻
Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
††† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, The University of Tokyo

クアップ・メソッド実行のコストは通常 100 クロック(EM-4[6),8)] では $8\mu s$)以下と考えられ,ノード間メッセージ送信・転送・受信(1 回)や,コンテキストスイッチ(1 回)のコストが $8\mu s$ 以上であれば,合わせると 80%以上がオーバーヘッドということになる.

実際,OS のソケットやプロセス(あるいはカーネルスレッド)の機能を用いて,メッセージ送信・転送・受信やコンテキストスイッチを実現した場合は,そのコストは数 $100\mu s$ を超えることが多い.また,OS よりは高速なユーザレベルのライブラリとして,メッセージパッシングライブラリやスレッドライブラリを用いて,メッセージ送信・転送・受信やコンテキストスイッチを実現した場合でも,例えば,AP1000+上の MPI の送受信の遅延は約 $20\mu s$, SparcServer1000 における Solaris Thread Library でのコンテキストスイッチのコストは約 $45\mu s$ といったように,多くの場合,$15\mu s$ を超えるコストが必要とされる.

我々の目標は,(1) 逐次の手続き呼び出しの 10 倍とも 100 倍ともいわれたリモートメッセージパッシングのコストを,逐次の手続き呼び出しに匹敵するオーダーにまで削減すること,(2) 実行時間の大部分を占めていた並列オブジェクトのノード内スケジューリングに要するコストを,オブジェクト本来の計算と比較して無視できる程度に抑えること,である.

そのためには,ハードウェアアーキテクチャの進歩が重要であるとともに,その特性を引き出すソフトウェアアーキテクチャもまた重要である.我々の提案するソフトウェア/ハードウェアアーキテクチャ(**ABCL/EM-4**[11),13),15)])は,RISC 的アプローチ,すなわち,ハードウェアは簡潔で高速なパケット(データ)駆動アーキテクチャとし,ソフトウェアについては最適な通信コードやコンテキスト管理コードをコンパイラに直接生成させる,というアプローチにより,メッセージ送信・転送・受信についてはメソッド起動処理と合わせても 3 ワードのメッセージで 95clocks($7.6\mu s$)以下のコスト,コンテキストスイッチについては 16clocks($1.3\mu s$)に最低限のレジスタの保存・復帰に必要なコストを追加したコスト,を達成した.

我々の得た結果は,並列オブジェクト指向計算モデル/言語が,実用的な並列システムの実現に極めて適していることを示している.

以下,2節では我々の並列オブジェクト指向計算/プログラミングモデルについて述べる.3節では EM-4[6),8)] のアーキテクチャの概略を述べる.4節ではソフトウェアアーキテクチャの実現方法について述べる.5節ではコンパイラについて述べる.6節では性能評価結果を示す.7節では我々のソフトウェア/ハードウェアアーキテクチャと従来の方式とを比較し,議論する.

2. 並列オブジェクト指向言語 ABCL/ST

並列オブジェクト指向言語 ABCL/ST[10)]は,データ駆動型並列計算機 EM-4 を対象マシンとする処理系 ABCL/EM-4 のプログラミング言語として設計された言語で,ABCL/1[12)]をベースとして部分型を含む静的な型付けなどを導入している.ABCL/ST そのものはマシン独立な言語となっている.

我々の計算/プログラミングモデル ABCL[12)]では,並列の単位となるのは,独自のメモリと計算能力を持つ多数のオブジェクトであり,これらはメッセージパッシングにより互いに協調しながら並列に計算を行う.また,オブジェクトは動的に生成できる.

ABCL のオブジェクトは状態変数とスクリプト(メソッド)を持つ.オブジェクトは休眠,待機,活性のいずれかのモードにあり,休眠モードでメソッド実行可能な要求メッセージを受け取ると,活性モードとなってメソッドを実行し,メソッド実行の完了により休眠モードに戻る.また,メソッド実行中に別のメッセージが必要な場合は,そのメッセージを受け取って実行を再開するまでの間,待機モードになる.

ABCL におけるメッセージ送信には,past 型(非同期メッセージ送信,文法上は,`[Obj <= Message]`)と now 型(非同期メッセージ送信+返答メッセージ待ち,文法上は,`[Obj <== Message]`)がある.非同期に送られた要求メッセージは,受け手オブジェクトでメッセージキューに入れられ,順番に処理される.また返答メッセージは,返答メッセージ待ちオブジェクトの実行を再開させる.

クラス定義は次の形式を用いており,オブジェクトはクラスのインスタンスとして生成される.

```
[class <クラス名> <インタフェース> <生成時引数リスト>
  <状態変数リスト>
  <ボディ>]
```

ただし,ABCL/ST では,ABCL/1 と異なり<ボディ>部に任意の形式を記述でき,通常は`script` 形式を記述して休眠モードでのメッセージ受信を行う.

3. EM-4 のアーキテクチャ

データ駆動並列計算機 EM-4[6),8)]は,電子技術総合研究所で開発された.EM-4 プロトタイプ[6)]は,12.5 MHz で動作する 80 台の要素プロセッサ(PE)から構成されており,1990 年から稼働している.EM-4

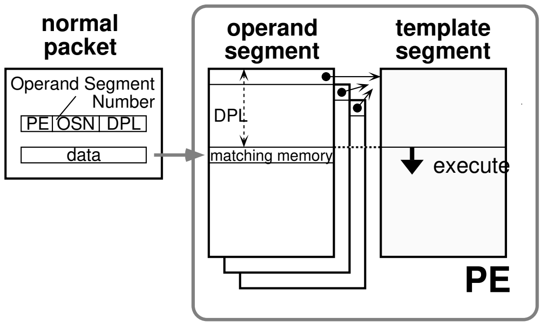

図 1 パケット駆動による命令列の実行
Fig. 1 Packet-driven execution of instruction sequence

のアーキテクチャは，パケット（データ）駆動データフローアーキテクチャと従来のコントロールフローアーキテクチャをパイプラインレベルで巧みに結合したハイブリッド・アーキテクチャとなっていて，パケット駆動された命令列をレジスタファイルを用いて RISC 的に効率良く実行する．

パケットは 2 ワードの固定長で，それぞれ 1 ワードのアドレス部とデータ部からなり，ハードウェア FIFO キューから取り出されながら，順次，処理される．命令列の実行はそれ自身が実行を終えるまで，続くパケットにより割り込まれることはない．命令列はあるテンプレートセグメント内のあるディスプレースメント（DPL）からのメモリに格納される．パケットによりその命令列の実行を開始させるには，オペランドセグメントというメモリセグメントをアロケートして，そのテンプレートセグメントにリンクしておく（図 **1**）．1 入力の場合，そのオペランドセグメント番号（OSN）と DPL をアドレス部に持つパケットにより，リンクされたテンプレートセグメント内の命令列の実行が（パケットのデータ部を最初の命令のオペランドデータとして）開始される．2 入力待ち合せの場合は，オペランドセグメント内のオフセット DPL のメモリワードを用いて高速な待ち合せを行う．

このような通常のパケットに加えて，10 種類以上の特殊パケットがある．特殊パケットは，オペランドセグメントを必要とせず，リモート資源割り当ての他，1 ワードデータのリモート書き込みなどにも利用される．

パケット送信については，パケット送信命令を実行（1 クロックを消費）するだけで，レジスタから直接 EM-4 の高速なネットワーク（FIFO 保証サーキュラオメガネットワーク）にパケットを送出できる．

4. ソフトウェアアーキテクチャ

EM-4 のパケット駆動による命令列の実行というアーキテクチャの特徴は，一見すると，メッセージの受信とメソッド起動をパケット駆動に対応させ，一つのメソッドの実行を一つの命令列の実行に対応させられそうである．しかしながら，EM-4 ハードウェアのこれらの特徴は，直接用いるには低レベル過ぎ，並列オブジェクト指向計算モデルをすべて実現するには，適切なソフトウェアアーキテクチャを必要とする．例えば，メソッドの実行中に続くパケットを受信するには，一旦，命令列の実行を終了しなくてはならず，ソフトウェアアーキテクチャは続くパケットの受信後に正しくメソッド実行が再開されるようにする必要がある．

我々の提案する ABCL/EM-4 のソフトウェアアーキテクチャが並列オブジェクト指向計算のために提供しなくてはならない機構として，以下では (1) （リモート）オブジェクトの生成，(2) （リモート）非同期メッセージパッシング，(3) ノード内スケジューリング，(4) 並列オブジェクトのガーベジコレクション，について述べる．

4.1 オブジェクトの表現

ABCL/EM-4 では，一つのノードに複数個のオブジェクトが存在し，各オブジェクトは図 **2** のようにいくつかのコンポーネントから構成される:

Object Segment: オブジェクトの状態を司る．状態変数や一時変数などオブジェクトの状態を保持し，EM-4 のオペランドセグメントを用いて実現される．メッセージキューを管理するポインタ，テンポラリボックスのリスト（スタックの代わり）を管理するポインタもこの上におかれる．
Object Segment のアドレスは PE 番号と合わせてシステム全体でユニークなオブジェクトの名前（ID）として用いられる．
また，パケットの DPL 値に対応するいくつかのエントリが存在し，後述のパケット交換手順を実現する（エントリの詳細は 11)，15) 参照）．

Reactive Text: オブジェクトのパケットに対するハンドラを司る．上で述べたエントリに送られたパケットに対して，リアクティブに（パケット駆動的に）実行される命令列をストアする．Object Segment にリンクされたテンプレートセグメントとしてすべてのオブジェクトに共通であり，同じノード内のオブジェクトに共有される．実際の実装では，この Reactive Text を複数個用意し，オブジェクトがメッセージ待ちかどうかに応じて Object Segment からのリンクを切替えることにより，待ち状態チェックなし命令列による高速なメッセージハンドリングを実現している．

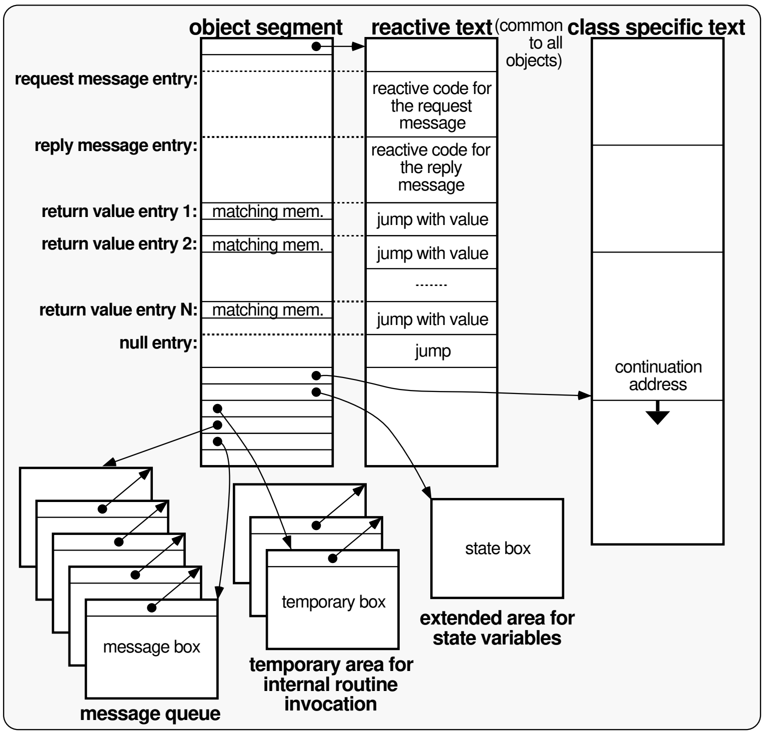

図 **2** オブジェクトの表現
Fig. 2 Representation of an object

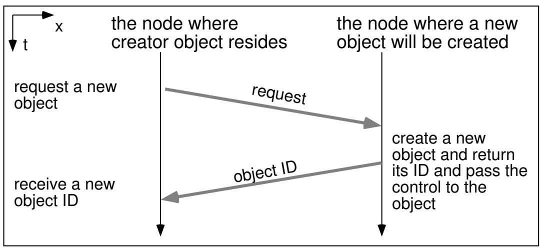

図 **3** オブジェクト生成のためのパケット交換
Fig. 3 Packet exchange for object creation

Class-Specific Text: オブジェクトの逐次実行を司る．オブジェクト内部の逐次実行プログラムである．(1) オブジェクトの初期化，(2) メッセージの取り出し，(3) メソッドルックアップ，(4) メソッド実行といった処理を行う．複数個の命令列から構成され，同じクラスに属するオブジェクトに共有される．逐次実行を中断して命令列の実行を終了する際には，必要なパケットの受信後 Reactive Text から Class-Specific Text 内の継続（continuation）アドレスにジャンプして実行再開できるよう，オブジェクトの状態をセットする．

4.2 オブジェクト生成

（リモート）オブジェクトの生成は，生成したいクラスの Class-Specific Text アドレスと返答アドレスを指定した特殊パケットを，オブジェクトを生成したいノードのハンドラ（Object Creator）に送信することで行われる（図 **3**）．Object Creator は，(1) 新たな Object Segment を割り当て，(2) 割り当てたオブジェクトの ID を含むパケットを返答アドレスに送信し，(3) 新しく生成したオブジェクトに制御を渡し，Class-Specific Text 内の初期化のコードを実行させる．図には示していないが，オブジェクトのクラスが生成時引数をとる場合は，この後さらに，新たな Object Segment に対して引数の書き込みを 4.3節のメッセージパッシングと同様の手順で行っている．

4.3 メッセージパッシング

（リモート）メッセージパッシングにはメッセージボックスの予約という方法を用いる．その理由は，パケット長が 2 ワード固定であり，大きなメッセージを送るには，複数個のパケットから受け手ノードでメッセージを再構成しなくてはならないからである．そこで，個別に通信されるパケットをまとめあげるため，メッセージボックス（mbox）というメッセージの入れものを介してメッセージパッシングを行う．メッセージボックス（32 ワード固定）は，フリーリストにより効率的に管理される．

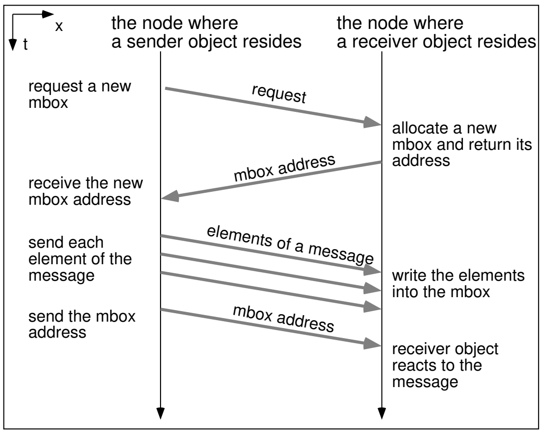

図 4 要求メッセージ送信のためのパケット交換
Fig. 4 Packet exchange for a request message send

要求メッセージ送信は次の手順で行われる(図 4).

(1) 送り手は,受け手ノードにメッセージボックスを予約するために,返答アドレスを含む特殊パケットを,受け手ノードに送信する.
(2) 受け手は,フリーリストから新たなメッセージボックスを割り当て,そのメッセージボックスアドレスを含むパケットを返答する.
(3) 送り手は,メッセージの引数を評価し,そのデータを直ちにパケット送信する.そのデータは,受け手ノードの予約したメッセージボックスの適切な位置に書き込まれる.
(4) 送り手は,受け手オブジェクトにそのメッセージボックスアドレスを送信する.
(5) 受け手オブジェクトは,メッセージのキューイングまたはメソッド起動を行う.

送り手では,メッセージの引数の評価とそのデータのパケット送信がパイプライン的に処理されるので,本メッセージ送信をパイプライン送信と呼ぶ.ネットワークは FIFO 性を保証しているので,メッセージボックスアドレス到着時には引数の書き込みは完了している.メッセージ引数のワード数を s(典型的には 2~4 程度)とすると,合計 $s+3$ 個のパケットが交換される.パイプライン送信の利点は 7節で議論する.

now 型など返答を求めるメッセージ送信の際は,返答先として ⟨返答先オブジェクト, 返答メッセージボックス⟩ のペアを指定して送信する.返答先に対する返答メッセージ送信は,すでに割り当てられた返答メッセージボックスを用いるので,予約は省略される.

4.4 オブジェクトのスケジューリング

3節で述べた通り,EM-4 のハードウェアは,ハードウェア FIFO パケットバッファをスケジューリングキューとして命令列を高速にスケジューリングする.オブジェクトのスケジューリングを実現するには,このハードウェアによる基本スケジューリングの上に,継続アドレスを含むコンテキストの保存/復帰のための処理を追加すれば良い.サスペンドの手順としては,(1) 継続アドレス以外のコンテキストを保存,(2) 必要なら Reactive Text の設定,(3) 継続アドレスの保存,を行った後,命令列を終了してハードウェアにより次のパケットをスケジュールさせる.また,実行再開の手順としては,ハードウェアがパケットをスケジュールして対応する Reactive Text 内のハンドラを起動させた後に,(4) 継続アドレス((3) で保存したもの)から実行再開,(5) 必要なら Reactive Text の設定,(6) 継続アドレス以外のコンテキストの復帰,を行う.

コンテキストスイッチは,(a) リモートのオブジェクトの生成やメッセージボックスの予約の結果を受けとる場合や,(b) 要求メッセージや返答メッセージをを待ち合わせる場合に必要となる.それぞれの場合に応じて継続アドレスの保存/復帰方法を特化することで,処理の高速化を行う.特に (a) においては継続アドレスはリターンパケットとの二入力待ち合せを行うことでハードウェアで処理ができる.

(1) や (6) のコンテキストの保存/復帰についてはコンパイラは生きている変数を解析して,保存するレジスタ数を最小限に抑える.

4.5 ガーベジコレクション

並列オブジェクトの Garbage Collection(GC)には,マーキングアルゴリズムを用いる.GC ルーチンはコンパイラによってユーザプログラムに埋め込まれる.現在までに並列オブジェクトの GC の実装と評価[14] を行った.6節での性能評価は,GC を行わないコードを用いて行った.

5. コンパイラ

ABCL/ST コンパイラは,ABCL/ST で記述されたソース言語を EM-4 のアセンブリコードにコンパイルする.コンパイラのもっとも重要な仕事は 4節で述べたソフトウェアアーキテクチャを実現するコードを生成することである.ABCL/ST と EM-4 のアセンブリコードの抽象レベルの差を埋めるために,コンパイラは三段階の抽象レベルの異なる中間言語を用いている.コンパイラの詳細は 7),10) で述べている.

現在のところ,中間コード最適化のうち,複写伝搬はまだ実装中であるため,コンパイラによる大域レジスタ割り付けで spill がやや多めに起きてしまう.このため,6で述べる性能評価では 11),13),15) で報告

```
[class Counter-with-Create (obj [Int (@ Int)]) ()
  (script
   (==> [n]
        (if (<= n 1)
            !1  ; returns 1
            ![(new Counter-with-Create) <== [(- n 1)]]))))]

[class Counter-after-Create (obj [Int (@ Int)])
   ((Int n) ((@ []) r))
  (state (Counter-after-Create next-obj))
  (if (<= n 1)
      [r <= []] ; ACK for the completion of object chain
      [next-obj := (new Counter-after-Create (- n 1) r)])
  (script
   (==> [n]
        (if (<= n 1)
            !1
            ![next-obj <== [(- n 1)]]))))]

[class Main (obj []) ((Int n))
  (state (Counter-with-Create counter-obj1)
         (Counter-after-Create counter-obj2)
;;; creation + now message passing
  [counter-obj1 := (new Counter-with-Create)]
  [counter-obj1 <== [n]]
;;; creation only
  (sync r        ; waiting ACK for object chain completion
    [counter-obj2 := (new Counter-after-Create n (@ r))])
;;; now message passing only
  [counter-obj2 <== [n]]
  (exit 0)]
```

図 5 遅延の評価のためのカウンタプログラム
Fig. 5 Counter program for latency evaluation

した一部ハンドコンパイルしたコードの性能評価結果より，3 割ほどその性能が低下している．しかしながら複写伝搬の実装により解決される問題と考えている．

6. 性能評価

測定は，開発した ABCL/ST コンパイラを用いて，EM-4（80PEs，クロック 12.5MHz）上で行った．

6.1 遅延の評価

リモートのオブジェクトの生成と，now 型リモートメッセージパッシングの遅延を測定するために，ネットワーク上の隣のノードにオブジェクトを生成し，そのオブジェクトに now 型メッセージパッシングを行うプログラム（図 **5**）を用いた．`Counter-with-Create` クラスの各オブジェクト O_iは，O_{i-1}から now 型のメッセージ M_{i-1}を受信すると，カウント値が 1 より大きい限り，次の処理を行う．

(1) 隣のノードに新しいオブジェクト O_{i+1}を生成
(2) O_{i+1}にカウント値をデクリメントして now 型でメッセージ送信（M_i）
(3) O_{i+1}からの返答メッセージ R_{i+1}をそのまま，M_{i-1}に関する返答メッセージとして O_{i-1}に返答メッセージ送信（R_i）

このとき，now 型のメッセージ送信のため，メッセージ送信は順次行われる．つまり，任意の時点においてシステム全体でメッセージは一つしか存在しない．一方，`Counter-after-Create` クラスの各オブジェクトは，オブジェクトの生成を先に行い，オブジェクトのチェインを作成後，now 型メッセージの処理を行う．

表 **1** オブジェクト生成/メッセージ処理の時刻詳細（クロック数）
Table 1 Details of timetable (clock cycles) for object creation/message processing

t	events
(1) Timetable for object creation and request message:	
0	Reception of request message M_{i-1} from object O_{i-1}
9	Resuming execution of O_i
27	**Start of object creation** (O_{i+1})
37	Sending of request packet for creation to $node_{i+1}$
53	Reception of O_{i+1}'s ID packet from $node_{i+1}$
57	Sending of starting packet to O_{i+1}
59	**Start of request message send** (M_i) **to** O_{i+1}
64	Sending of mbox request packet for M_i to $node_{i+1}$
94	Reception of mbox address from $node_{i+1}$
126	Sending of the mbox address packet (M_i) to O_{i+1}
131	Reception of M_i by O_{i+1}
(2) Timetable for reply message:	
0	Reception of reply message R_{i+1} from object O_{i+1}
7	Resuming execution of O_i
15	**Start of reply message send** (R_i) **to** O_{i-1}
26	Sending of the reply mbox address (R_i) to O_{i-1}
35	Reception of R_i by O_{i-1}
(3) Timetable for request message:	
0	Reception of request message M_{i-1} from object O_{i-1}
9	Resuming execution of O_i
28	**Start of request message send** (M_i) **to** O_{i+1}
45	Sending of mbox request packet for M_i to $node_{i+1}$
60	Reception of mbox address from $node_{i+1}$
92	Sending of the mbox address packet (M_i) to O_{i+1}
97	Reception of M_i by O_{i+1}

実際には，図 5 のプログラムに実行時間測定のためのコードを追加して測定を行った．`Counter-with-Create` クラスでの測定では，1 オブジェクト生成+1 now 型メッセージパッシングあたりの平均実行時間は 13.12μs（164 clocks）であった．また，`Counter-after-Create` クラスでの測定では，1 オブジェクト生成あたりの平均実行時間は 3.92μs（49 clocks），1 now 型メッセージパッシングあたりの平均実行時間は 10.64μs（133 clocks）であった．表 **1** には時刻の内訳を示した．表の (3) に示す通りメッセージボックスの予約には 15 clocks を必要とする．ただし，表の (1) では，メッセージボックスを要求された時点では，その直前に生成したオブジェクトに関する処理を続けているため，メッセージボックスの予約に 30clocks を要している．

6.2 スループットの評価

80 台の PE からなる EM-4 システム全体として，どの程度の処理能力があるのかを測定するために，木構造にオブジェクトを生成し，その上で now 型リモートメッセージパッシングを行うプログラム（図 **6**）を

```
[class Fib (obj [Int (@ Int)]) ()
  (script
   (==> [n]
        (if (<= n 2)
            !1
            !(match
                 ;; synchronize two now message passings
                 (sync2 r1
                   [(new Fib) <= [(- n 1) (@ r1)]]
                   [(new Fib) <== [(- n 2)]])
               (=> [t1 t2]
                   (+ t1 t2))))))]

[class Main (obj []) ((Int n))
  [(new Fib) <== [n]]
  (exit 0)]
```

図 6　スループット評価のためのフィボナッチ数プログラム
Fig. 6　Fibonacci program for throughput evaluation

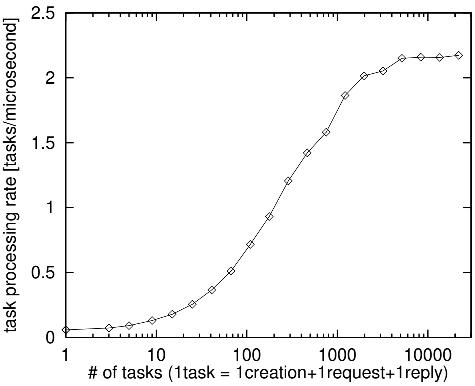

図 7　スループット評価プログラムの結果
Fig. 7　The result of the throughput measuring program

用いた．各オブジェクトは now 型メッセージを受けて，次の処理を行う．

(1)　新しいオブジェクトを二つ生成（サーキュラオメガネットワーク上に二つある隣のノードにそれぞれ生成する）
(2)　生成した 2 オブジェクトに now 型マルチキャスト
(3)　now 型マルチキャストの返答を足し合わせて，返答メッセージ送信

自然数 n を与えたとき，結果として，ちょうどフィボナッチ数列の第 n 項が得られるような木構造についてのスループット（タスク処理率）を測定した．ただし，1 タスク $=$ (1 リモートオブジェクト生成$+$ 1 now 型リモートメッセージパッシング) とする．もちろん，このプログラムは容易に大きな並列度が得られるときの性能を評価するためにある．図 7 の N1 に，$2 \leq n \leq 21$ と変化させたとき，総タスク数に対するタスク処理率の変化を示す．このグラフから，総タスク数の増加とともにタスク処理率は飽和していくが，システム全体（80PEs, クロック周波数 12.5 MHz）として，1 μs の間に 2.15 タスク以上の並列処理性能を持ち得ることがわかる．

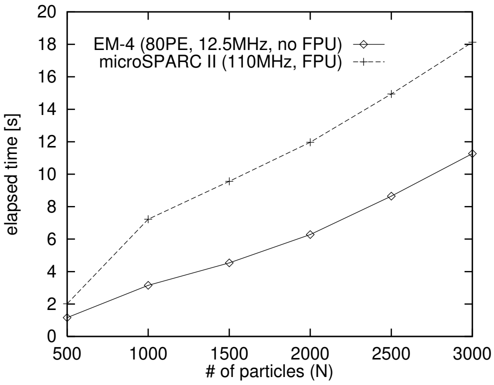

図 8　三次元 N 体問題の実行時間
Fig. 8　The elapsed time of the N-body program

6.3　三次元 N 体問題による評価

N 体問題とは，N個の質点が互いに遠隔力を及ぼしながら運動しているニュートン力学系の状態をシミュレートする問題である．近似を用いることで計算量を減らす手法として提案されてきたものには J. Barnes と P. Hut による木構造を用いた $O(N \log N)$ のアルゴリズム[2]がある．我々がすでに発表した N 体問題の並列オブジェクト指向数値計算アルゴリズム[10]は，J. Barnes と P. Hut の逐次アルゴリズムを拡張したもので，その時間複雑度は $O(N)$ 個の並列オブジェクトを用いて $O(\log N)$ となり，非常に高い並列度が得られることがわかっている．

三次元 N 体問題アプリケーションによる性能測定には，以下の測定条件を用いた:

(1)　近似条件: 近似セルのサイズ/距離 $< 1/0.95$
(2)　$N(=500, 1000, \ldots, 3000)$ 質点を均等に分布
(3)　一ステップ（木構造の生成と力の計算）の時間を測定

EM-4 との比較のために，SUN のワークステーション SS5（microSPARC II，110MHz）を用いた．そのための実行コードは，同一の ABCL/ST プログラムから ABCL/ST コンパイラの C コード生成機能と gcc2.6.3(-O4) を用いてコンパイルした．

以上の条件下での性能測定結果は，図 8 のようになった．80 台の PE を持つ EM-4 で，SS5 の単一プロセッサの 1.6 倍から 2.3 倍程度の性能が得られている．EM-4 は，浮動小数点演算ユニットは持っていないため，浮動小数点演算は整数命令の組合せで実行する．浮動小数点演算を多用するため，SS5 の単一プロセッサ性能を EM-4 のそれの 20 倍程度とすると妥当

表 **2** 他の並列オブジェクト指向言語処理系とのコストの比較
Table 2 Cost comparison with other language systems

language	machine	[MHz]	remote msg. [cycles]	remote msg. [μs]	context switch [cycles]	context switch [μs]
ABCL/ST	EM-4	12.5	72	5.8	16	1.3
CST[4]	J-Machine	20	76	3.8	110	5.5
ABCL/f [13]	AP1000	25	223	8.9	240	9.6
Concert[5]	CM-5	33	252	7.7	232	7.0
Ocore[17]	Paragon	50	2450	49.0	210	4.2
A-NETL[16]	A-NET	6	1558	260.0	3269	544.8

な数字と考えられる．実際，本プログラムの主要な計算の一つ（SS5 で実行時間の 5%程度）である万有引力の法則による加速度の計算に，EM-4 は SS5 の 61 倍の時間を要する．なお，EM-4 の各 PE のメモリ量は 0.5MB であるため台数効果は測定できなかったが，N=500 程度と問題サイズが小さい場合においても，80 台の PE すべてを有効に利用できていた．

7. 議　　論

本節では，我々のソフトウェア/ハードウェアアーキテクチャが CST[4]/J-Machine[3]などにおける従来の方式と比べてどのようにしてオーバーヘッドを削減しているのかについて考察する．

7.1 関 連 研 究

表 **2** に，主要な並列オブジェクト指向計算システムにおけるリモートメッセージパッシングやコンテキストスイッチのコストを示す．

リモートメッセージパッシングについては，メッセージ送信からメソッド起動までの時間を示した．ABCL/EM-4 は CST/J-Machine に次ぐ性能を示した．CST/J-Machine はこの表では一番良い性能を得ているが，7.2節で示す問題がある．これら二つでは受信に応じて命令列が駆動されるハードウェアを用いている．ABCL/AP1000[13]は，アクティブメッセージ[9]のように，AP1000 のサーキュラバッファからメッセージを取り出すためのハンドラをコンパイラがメッセージのフォーマット毎に生成して，受信側では送信側の指定したハンドラを単に呼び出すことで高速な受信を可能としている．これら三つではコンパイラが生成したコードで送信を行う．Concert[5], Ocore[17]ではライブラリを用いて送受信をし，ライブラリで受信したメッセージに対してアクティブメッセージ的に処理を行う．A-NET[16]ではメッセージ送信命令のマイクロコードによる実行，OS による受信を行うが，OS のオーバーヘッドが大きい．

コンテキストスイッチについて，ABCL/EM-4 は一番良い性能を得た．ただし，表中，ABCL/EM-4 は「返答メッセージ待ちコンテキストスイッチ」のコストで，後の 7.2節で述べるコンパイラによる保存分は含まない．CST, A-NET では ABCL/EM-4 では必要ないコンテキストフレームの割り当てが必要で，CST/J-Machine で未来トラップを用いる場合はさらに 50cycles が必要となる．OCore についてはスイッチのみのコストを示した．ABCL/f, Concert については要求メッセージがすぐに処理できない時のコンテキストスイッチで，フレーム割り当てやメッセージキュー操作を含むコストを示した．

ABCL/EM-4，CST/J-Machine ではハードウェアのパケットキュー，メッセージキューをそれぞれスケジューリングキューとして用いる．これによりキュー操作をハードウェアに任せるとともに，リモートのメッセージに対して小さな遅延で処理を行うことができる．他の処理系では，むしろ，ノード内のローカルなメッセージパッシングに対して高性能が得られるようにしており，ソフトウェアのスケジューリングキューを併用する．特に，ABCL/AP1000 では，更にスタックを併用し，実行をブロックせずにノード内の処理が可能な場合には，フレームをスタックに取ってスケジュールすることで，2.3μs での処理が可能である．

7.2 リモートメッセージ送信のオーバーヘッド削減

EM-4 は，パケットを通信の単位としているので，別々に送信されたパケットからメッセージを構成するためには，4.2節で述べたようにメッセージデータを送信する前にメッセージボックスを予約しておく必要がある．メッセージボックスを前もって予約する我々の方法は，J-Machine[3]のようにメッセージを通信の単位とし，メッセージをサーキュラバッファを用いて受信する方法と比較して，一見予約に要する時間だけ損しているように見える．しかし，EM-4 のようなリモート通信が高速なアーキテクチャでは通信遅延はあまり問題にならないばかりでなく，実は，(a) 受け手側では，フリーリストでメッセージのバッファを管理できる．(b) 送り手側では，メッセージ引数の評価・送信のパイプライン化が可能，という点から以下のような理由で性能向上をもたらす．

(a) は，受け手ノードが予約なしでメッセージを受けるためには，特殊なハードウェアを用いるにしてもサーキュラバッファ以外の方法は複雑過ぎる．我々の方法では，予約を用いるため，複雑なハードウェアを必要としないばかりでなく，ソフトウェアによるフリーリストを用いたバッファ管理ができる．

サーキュラバッファと比較すると，フリーリストを用いる我々の方式には，キューイングされたメッセージ

のコピーが不要，という大きな利点がある．一方，サーキュラバッファでは，(1) 着順優先以外のスケジューリングが必要な場合や，(2) 到着したメッセージがすぐ処理できない場合に，メッセージをサーキュラバッファ外の領域にコピーしてサーキュラバッファから取り除かなくてはならない．(1) については，探索問題などがこれに当たり，(2) については，例えば，あるオブジェクトが now 型のメッセージパッシングをして特定の返答メッセージ M_Rを待っているとき，M_Rより先に到着したメッセージをコピーする必要がある．

(b) は，予約を用いるため，送り手は実際のメッセージの送信前にメッセージボックスアドレスを得ることができる．このため，メッセージ引数の評価・送信のパイプライン化（パイプライン送信）が可能になる．

パイプライン送信は，メッセージ送信の遅延を削減するとともに，メモリアクセス回数の削減をもたらす．なぜなら，引数の評価後，直ちにアドレスを指定して送信可能であることは，あるメッセージを送信中に別のメッセージを送信可能であることを意味し，その結果，中断される可能性のあるメッセージ送信の評価済み引数データの退避が不要となる．例えば，

[*ObjA* <= [:*MsgA* (- n 1) (*Fun* x y)
　　　　　　　　　　[*ObjB* <== [:*MsgB*]]]]

のように，(1) 演算，(2) 関数コール，(3) メッセージパッシング，の評価結果を *ObjA* へのメッセージの引数とするような計算を考えると，ABCL/EM-4 では，*ObjA* へのメッセージの引数データ（例えば (1) の演算結果）は，その評価が済んだ時点で，メモリを介することなく直接パケット送信することが可能であり，*ObjB* への送信などのために引数データを一旦メモリに退避する必要がない．

7.3 ノード内スケジューリングのオーバーヘッド削減

EM-4 ハードウェアはパケットを受けてから命令を実行するまでの遅延が小さい上に，パケットの処理と命令の処理はパイプライン化されている．例えば，データの書き込みのための特殊パケットの処理は，パイプラインとしてみると一つのパケットにつき 2 クロックしか必要としない．

いくつかの計算機ではレジスタコンテキストの自動保存をサポートし，そのオーバーヘッドを削減するため汎用レジスタの数を少なくしてきた．例えば，J-Machine では，汎用レジスタは 4 個しかなく，それらは未来トラップで自動的に保存される．一方，EM-4 は，10 個の汎用レジスタを持ち，大部分の計算を RISC 的なレジスタベースで効率良く実行する．しかも，コンテキストスイッチが起こる位置は命令列の終了地点と決っているので，レジスタ上の必要なデータはハードウェアではなくプログラム自身で保存するため，コンパイラの解析により保存を最低限に抑えて，オーバーヘッドの非常に少ないコンテキストスイッチが実現できる．これは，逐次型言語において，caller がレジスタを保存するのと同様に実現できる．

8. おわりに

従来，並列オブジェクト指向計算は，実際の計算機においてその性能を十分に発揮できないという問題があった．我々は，パケット（データ）駆動アーキテクチャという EM-4 ハードウェア上に，(1) オブジェクトのパケット駆動的表現，(2) フリーリストによるメッセージバッファ管理，(3) パイプラインメッセージ送信，(4) コンテキスト最小化，といったソフトウェアアーキテクチャ技術を用いることで並列オブジェクト指向計算を効率良く実行できることを示してきた．

本稿では，EM-4 をターゲットとして開発した ABCL/ST コンパイラを用いて，遅延，スループットの性能評価，実アプリケーションによる性能評価を行った．我々の得た結果は，並列オブジェクト指向計算モデル/言語が，適切なハードウェア・ソフトウェアのサポートのもとに実用的な並列システムとして実現可能であることを示している．

現在，EM-4 の次の世代として，EM-X がすでに電子技術総合研究所で開発されている．今後，EM-X 上にも処理系の移植を進めていく予定である．

謝辞　電総研の山口喜教，坂井修一（現筑波大学），佐藤三久（現 RWCP），児玉祐悦の各氏には，EM-4 の使用に際して技術的な点を含む貴重な御助言をいただいた．ここに深謝する．

参 考 文 献

1) Agha, G.: *Actors : A Model of Concurrent Computation in Distributed Systems*, The MIT Press (1987).
2) Barnes, J. and Hut, P.: A Hierarchical $O(N \log N)$ Force-Calculation Algorithm, *Nature*, Vol. 324, pp. 446–449 (1986).
3) Dally, W. J. et al.: The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms, *IEEE MICRO*, Vol. 12, No. 2, pp. 23–39 (1992).
4) Horwat, W.: Concurrent Smalltalk on the Message-Driven Processor, Technical Report MIT-AI-TR 1321, MIT AI Lab. (1991).
5) Karamcheti, V. and Chien, A.: *Concert*—

Efficient Runtime Support for Concurrent Object-Oriented Programming Languages on Stock Hardware, *Proc. of Supercomputing'93*, pp. 598–607 (1993).

6) Kodama, Y., Sakai, S. and Yamaguchi, Y.: A Prototype of a Highly Parallel Dataflow Machine EM-4 and its Preliminary Evaluation, *Proc. of InfoJapan '90*, pp. 291–298 (1990).

7) Matsuoka, S., Yasugi, M., Taura, K., Kamada, T. and Yonezawa, A.: Compiling and Managing Concurrent Objects for Efficient Execution on High-Performance MPPs, *Parallel Language and Compiler Research in Japan* (Bic, L. F., Nicolau, A. and Sato, M.(eds.)), Kluwer Academic Publishers, pp. 91–125 (1995).

8) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Dataflow Single Chip Processor, *Proc. of ISCA*, pp. 46–53 (1989).

9) von Eicken, T., Culler, D. E., Goldstein, S. C. and Schauser, K.E.: Active Messages: a Mechanism for Integrated Communication and Computation, *Proc. of ISCA*, pp. 256–266 (1992).

10) Yasugi, M.: A Concurrent Object-Oriented Programming Language System for Highly Parallel Data-Driven Computers and its Applications, PhD Thesis, Department of Information Science, The University of Tokyo (1994).

11) Yasugi, M., Matsuoka, S. and Yonezawa, A.: ABCL/onEM-4: A New Software/Hardware Architecture for Object-Oriented Concurrent Computing on an Extended Dataflow Supercomputer, *Proc. of ICS*, pp. 93–103 (1992).

12) Yonezawa, A.(ed.): *ABCL: An Object-Oriented Concurrent System*, The MIT Press (1990).

13) Yonezawa, A., Matsuoka, S., Yasugi, M. and Taura, K.: Implementing Concurrent Object-Oriented Languages on Multicomputers, *IEEE Parallel & Distributed Technology*, Vol.1, No.2, pp. 49–61 (1993).

14) 八杉昌宏: データ駆動並列計算機上の分散並行ガーベジコレクションの評価, 並列処理シンポジウム(JSPP'97)論文集，pp. 345–352 (1997).

15) 八杉昌宏，松岡聡，米澤明憲: ABCL/onEM-4：データ駆動計算機上の並列オブジェクト指向計算システムの高性能実装, 並列処理シンポジウム(JSPP'92) 論文集, pp. 171–178 (1992).

16) 岩本善行, 吉永努, 馬場敬信: 言語レベルからみたA-NET マルチコンピュータのメッセージパッシング性能, 情処研報 96-ARC-119, pp. 1–6 (1996).

17) 小中裕喜, 石川裕, 前田宗則, 友清孝志, 堀敦史: 超並列オブジェクトベース言語 Ocore の並列計算機上での実装, 情報処理学会論文誌, Vol. 36, No. 7, pp. 1520–1528 (1995).

(平成 8 年 9 月 16 日受付)
(平成 9 年 7 月 4 日採録)

八杉　昌宏（正会員）

1967 年生．1989 年東京大学工学部電子工学科卒業．1991 年同大学大学院電気工学専攻修士課程修了．1994 年同大学大学院理学系研究科情報科学専攻博士課程修了．1993～1995 年日本学術振興会特別研究員（東京大学，マンチェスター大学）．1995 年より神戸大学工学部情報知能工学科助手．博士（理学）．並列処理，言語処理系などに興味を持つ．日本ソフトウェア科学会，ACM 会員．

松岡　　聡（正会員）

1963 年生．1986 年東京大学理学部情報科学科卒業．1989 年同大学大学院博士課程中退．同年情報科学科助手，同大学情報工学専攻講師を経て，1996 年 10 月より東京工業大学情報理工学研究科助教授．理学博士．オブジェクト指向言語，言語処理系，並列システム，自己反映言語，ユーザ・インタフェースなどの研究に従事．ECOOP'97 のプログラム委員長をはじめ，各種学会のプログラム委員を歴任．ACM，IEEE-CS 各会員．

米澤　明憲（正会員）

1947 年生．1977 年 Ph.D. in Computer Science（MIT）．1989 年より東京大学理学部情報科学科教授．超並列・分散ソフトウェアアーキテクチャ，などに興味を持つ．共著書「算法表現論」「モデルと表現」（岩波書店），編著書「ABCL: An Object-Oriented Concurrent System」（MIT Press）などがある．4 年間ドイツ国立情報処理研究所（GMD）科学顧問，ACM Transaction on Programming Languages and Systems 副編集長，IEEE Parallel & Distributed Technology 編集委員などを歴任，現在日本ソフトウェア科学会理事長．