早稲田大学大学院理工学研究科

博 士 論 文 概 要

論 文 題 目

並列論理型言語処理系の
分散メモリ環境における実装方式の研究

申 請 者

六 沢 一 昭
Kazuaki ROKUSAWA

1997 年 11 月

本論文は, 並列論理型言語処理系を分散メモリ環境へ実装する方式を述べたものである. まず, 実装する上での技術的課題を示し, それらを解決する方式を検討した. そして実際に分散メモリ環境に処理系を実装し評価を行なった.

分散メモリ環境と並列論理型言語

複数のプロセッサ (以後 PE と略す) を何らかの形で接続したものが並列計算機である. 並列計算機を使って処理を行なうには複数の PE にデータを共有させなければならない. 同期を取らせることも必要である. **分散メモリ環境**とは, 共有メモリの存在しない並列計算機環境である. この環境ではデータを物理的に共有することがない. **メッセージ通信のみ**によってデータの共有と同期を行なう.

並列論理型言語は並列記号処理を記述するための言語である. プログラムは節からなる. 実行単位はゴールであり, これは手続き型言語の関数やサブルーチンに相当する. ゴールの実行とは, そのゴールに対応する節を選び, 選ばれた節に記述されているユニフィケーション群とゴール群を実行することである. **ユニフィケーション**は論理型言語特有の処理であり, ふたつの変数を結び付けるものである. 一方が未定義 (値が書き込まれていない状態) で, もう一方が値を持つ場合は代入操作を行ない, 両方とも値を持つ場合は値の同一性をチェックする. 両方とも未定義の場合は一方から他方へ参照を張る (一方への参照を他方へ書き込む).

ゴール群は**共有変数**を介して通信する. あるゴールが共有変数に値を書き込み, その他のゴール群がその共有変数から値を読み出すのである. 未定義状態の共有変数の値を読もうとしたゴールは実行を中断し, 別のゴールによって値が書き込まれるのを待つ. これによりゴール間の同期がなされる.

技術的課題

分散メモリ環境における並列論理型言語処理系を考える. この環境では各 PE にゴールが分散し並列に実行される. 実行においては, 他 PE のゴール群との間でデータの共有や同期処理を行なう.

共有メモリがないため「他 PE への参照」が出現する. この参照を**外部参照**と呼ぶことにする. 異なった PE に存在するゴール群は, 外部参照を介して共有変数にアクセスし, データの共有と同期を行なう. これを実現するためには処理系は以下の 3 つの機能を持つことが必要である. **外部参照管理** — 外部参照の生成消滅などの管理. **参照値の読み出し** — 外部参照の指す値を読み出す. **分散ユニフィケーション** — 外部参照に係わるユニフィケーション.

この環境には,「PE の状態」が PE の数だけ,「メッセージの内容」がメッセージの数だけ存在する. これらの集合を**大域状態**と呼ぶ. プログラムの開発及び実行では, この**大域状態を検出/制御**することが必要になる. これを実現するために処理系は以下の 3 つの機能を持つべきである. **終了検出** — すべての PE でゴールが終了しメッセージも存在しない状態を検出する. **強制終了** — すべてのゴールを終了させ, メッセージも存在しないことを確認する. **静止状態検出** — すべてのゴールが中断し, メッセージもなく, そのままでは何の変化も起きない状態 (静止状態) を検出する. デッドロックは静止状態のひとつである.

以上述べた 4 つ (外部参照管理, 参照値の読み出し, 分散ユニフィケーション, 大域状態の検出と制御) が並列論理型言語処理系実装における技術的課題である.

解決する方式

o **外部参照管理**　大域的な処理を行なうことなく PE 間の参照を管理する方式を提案した. 各 PE は**輸出表**というアドレス変換テーブルを持ち, 他 PE から参照されるデータセルを登録する. 外部参照は「PE 番号と輸出表エントリ番号の組合せ」で表現する. PE 内の GC (ゴミ集め) によってデータセルのアドレスが変化しても外部参照には影響しないので, PE 毎の独立した GC が可能である.

外部参照と輸出表の GC は**重み付け参照カウント方式**によって行なう. これは参照される側 (輸出表エントリ) だけでなく参照する側 (外部参照) もカウント (**重み**) を持つものである.「あるエントリの重み」と「それを指す外部参照の重みの合計」を等しく保つことにより,「エントリの重みゼロ」は「そのエントリを指す外部参照が存在しない」ことと同値になる. 外部参照を生成する時には外部参照とエントリに同じ重みを付け, コピーを作る時は, 重みを分割し一方をコピーに付ける. 外部参照を解放した時は重みを輸出表へ返却し, 返却された重みはエントリから引く. 引いた結果ゼロになったならばそのエントリは解放する.

o **参照値の読み出し**　読み出し遅延を隠蔽し外部参照のループに対処できる方式を提案した. 読み出し処理は以下の通りである. **開始** — 外部参照の指す PE へ `%read` メッセージを送信する. `%read` は参照値の返信先を引数として持つ. 参照値を必要としたゴールは実行を中断し外部参照に記憶する. ここで実行ゴールが切り替わる. プログラムの実行は中断せず, 読み出し遅延が隠蔽される. **`%read` の受信/参照値の返信** — 参照先が値を持つならば直ちに値を返信する. 外部参照ならば, それが指す PE へ `%read` を転送する. 未定義の場合は参照先に返信先を記憶し, 値が書き込まれたら, 記憶していた返信先へ送信する. **参照値を受信** — 外部参照を参照値で書き換え, 記憶されていた中断ゴールの実行を再開する.

ループが存在する場合は `%read` を単純に転送すると処理が終わらなくなる. カウンタを `%read` に持たせて転送の上限を設定することにより終了を保証した.

o **分散ユニフィケーション**　ループの生成を防ぎ終了を保証する方式を提案した. ユニフィケーションを依頼する `%unify` メッセージを外部参照の参照先 PE へ送ることによって, ユニフィケーションを行なう.

ユニフィケーションによって外部参照を書き換えることのできない言語ではループを作ってはいけない. `%unify` がループに飛び込むと転送が繰り返され処理が終わらないからである. 未定義変数から外部参照へ参照を張ることを禁止すると, ループは生まれないが, 処理の終了する保証もなくなってしまう. 向き (PE 番号の大小関係) に係わる属性 (**Safe/Unsafe**) を外部参照に持たせることにより,

ループの生成防止と終了の保証を実現した.

∘ **大域状態の検出と制御**　重みを用いて終了検出を行なう **WTC (Weighted Throw Counting) 方式**を提案した. この方式では, 終了を検出するプロセス (検出プロセス) と PE 及びメッセージのそれぞれが重みを持つ. 検出プロセスの重みは負の整数, 他のふたつは正の整数である. そして「すべての重みの合計がゼロ」を保つ. 送信するメッセージには正の重みを付け, 自分の重みはその分だけ減らす. メッセージを受信したら, 付いていた重みを自分の重みに加える. PE 内のゴールがすべて終了したら重みを検出プロセスに返却する. これらの操作を行なうと, すべての PE でゴールが終了しメッセージも存在しない時にのみ, **検出プロセスの重みがゼロ**になる.

上述した WTC 方式をベースに強制終了と静止状態検出を実現した. **強制終了**は, メッセージを PE に送ることによってゴールを終了させ, WTC 方式によって, メッセージの存在しないことを確認する. **静止状態検出**は, PE 内のすべてのゴールが中断した時にも重みを返却することにより, 終了検出と同様に実現できた.

実現した処理系

並列論理型言語 KL1 の分散処理系を, 性質が異なるふたつの並列計算機上に実現し評価を行なった. ひとつは記号処理専用機であり, もうひとつは汎用計算機である. いずれの環境においても処理方式の有効性を確認することができた.

∘ **Multi-PSI 及び PIM/m 上の KL1 分散処理系**　Multi-PSI と PIM/m は分散メモリ型の記号処理専用並列計算機である. Multi-PSI は 64 台の, PIM/m は 256 台の PE を持つ. 処理系の実現においては, タグを用いて様々な新しいデータタイプを定義するなど, **専用機ならではの最適化**を多数行なった.

数種類の実験的/実際的な応用プログラムを実行して WTC 方式と外部参照管理方式を評価した. WTC 方式では充分少ないメッセージによって終了が検出されること, 及び, 充分低いオーバヘッドで外部参照の管理できることが確認できた.

∘ **分散 KLIC 処理系**　KLIC は汎用 KL1 言語処理系である. KL1 プログラムは C 言語に変換され UNIX の下で実行される. この処理系は,「性能は高いが移植性が極めて低い」という Multi-PSI, PIM/m 上の処理系の欠点の解消を目的として開発したものである. 実現においては性能だけでなく**移植性**も重視した.

処理系プログラムの計算機依存部分は全体の 5% 以下であり移植性は高い. 実際に数多くの計算機に移植がなされている. 代表的なベンチマークと実験的/実際的な応用プログラムを実行した結果, 並列実行時の高い性能が認められた. 分散処理系であるが故の部分のオーバヘッドもわずかであることが確認できた.

分散メモリ環境への処理系実装には「大域的な処理を局所的な処理の集合で実現する」「大域的な状態を局所的な視点から検出/制御する」という困難さがあった. 本論文で述べた実装方式はこれらを充分に解決したと考えられる.