記号処理 40－3
（1987. 1. 13）

EVLISマシンのための Parallel-Lispコンパイラとその実現

安田弘幸，坂口寿和，三野雅仁，山田真一，斎藤年史，安井 裕
（大阪大学・工学部）

Lispの高速処理を目的として，すでに開発したマルチプロセッサによるLisp並列処理マシン — EVLISマシン — のためのコンパイリングシステムについて述べる．

並列処理においては，プロセスの各プロセッサに対する割付けとプロセスの生成・消滅によるオーバヘッドとが問題となり，とくにオーバヘッドは，コンパイリングシステムではインタプリタシステムに比べて処理速度に対する影響が大きい．この問題を解決するため，ソースプログラム上で並列処理を指定した関数呼び出しの部分に，プロセスの制御を詳細に指示できる〈評価コントロールリスト〉や，関数の再帰呼び出しなどによって起こるプロセス数の爆発的な増大を抑制する機構を導入した．

これらの機能を実現できるコンパイラをEVLISマシン上にインプリメントし，ベンチマークプログラムを用いて動特性を測定した結果，オーバヘッドの減少と効率の良い負荷分散とを確認した．

PARALLEL-LISP COMPILER FOR EVLIS MACHINE

Hiroyuki YASUDA, Toshikazu SAKAGUCHI, Masahito MINO,
Shinichi YAMADA, Toshifumi SAITO and Hiroshi YASUI

Department of Applied Physics, Faculty of Engineering, Osaka University
2-1, Yamadaoka, Suita, Osaka 565, JAPAN

This paper describes the compiling system for EVLIS machine, multiprocessor system developed in our laboratory.

In parallel processing, both the overhead time for process creating and the distribution of tasks to these processors affect the system performance. Particularly, the overhead is critical in compiling system. To solve these problems, we introduce "Evaluation Control List" which controls creation of processes in detail, and one construct which reduces the number of processes created in recursive function calls. "Evaluation Control List" is placed in the part of function call of source programs.

We implement the Parallel-Lisp compiler, and the dynamic measurements show some reduction in the overhead and the efficient load-distribution to processors.

1 はじめに

我々は，リスト処理言語 Lisp の高速処理を目的とした Lisp の並列処理マシン ― EVLIS マシン[1][2][3][4] ― の試作・研究を1978年より行っている．このマシンは，複数台のリスト処理専用プロセッサ ― EVAL-Ⅱプロセッサ ― と，各プロセッサに共有されるメインメモリを中心にしたマルチプロセッサシステムである．すでに，複数台の EVAL-Ⅱプロセッサ用Parallel-Lisp インタプリタ[5]と,プロセッサ単体用Lispインタプリタ[6]，コンパイラ[7]とがインプリメントされ稼働している．これらの処理系は，ベンチマークプログラムなどにおいてそれぞれ十分な速度を得ていることがすでに報告された．現在，さらに高速な処理を目的として，出力されるオブジェクトが並列に実行される Parallel-Lispコンパイラ[11](以下，並列コンパイラという)の開発を行っている．

本稿では，まず２章において EVLISマシン における並列処理方式について述べ，並列コンパイラにおける実行効率改善のための新たなアイデアを示す．３章においてこのコンパイラに与えるプログラム上の表現について述べ，４章においてこれらの機能を実現する並列コンパイラについて説明し，５章で実行結果を示して考察する．

2 EVLISマシンにおける並列処理

この章では，すでに我々の提案した並列処理の方式を述べ，並列処理でのいくつかの留意点と，これらの手段のコンパイリングシステムへの適用について示す．

2.1 インタプリタでの並列処理

Lisp の処理を並列に実行する場合，並列化の対象をどこにおくかが問題となる．我々は，インタプリタ内での関数 evlis の動作がその第１引数の要素の数にプロセスとして分割でき，かつ evlis がインタプリタ実行中に頻繁に使用されることに着目した．そこで，Parallel-Lisp インタプリタでは，EVLIS マシンの名前にあるように，関数 evlis での第１引数の各々の評価を複数台のプロセッサで並列に実行している．このことは，Lisp 関数の引数を並列に評価することになる．ここで分割された評価のためのそれぞれの処理をプロセスと呼ぶ．処理の対象となる Lispプログラムは，pure Lisp に限定せず，副作用を伴う関数の使用も許し，副作用の伝幡についてはインタプリタが自動的にリカバーし解決する．すでに，並列処理の効率向上を目的として，並列処理の選択を可能とする関数タイプを導入し成功した[8]．

2.2 コンパイラと並列処理

並列コンパイラにおいても，まず，関数の引数評価に並列処理を適用することを主体とした．コンパイリングシステムでは，プロセスの処理において並列処理のオーバヘッドの割合がインタプリタのそれに比べて大きく，めだつ可能性がある．いま，ある関数を並列に実行するとき，ｎ個のプロセスに分かれるとする．プロセスｉの処理時間をＴpi，プロセス生成に要する時間，つまり並列処理のためのオーバヘッドをＴoiとする．この関数をＮ台のプロセッサで処理するときに各プロセッサにつねに仕事が与えられたと仮定すると，この関数の実行時間Ｔは，

$$T = \frac{1}{N}\sum_{i=1}^{n}(Tpi + Toi)$$

と表される．インタプリタとコンパイラとを比べたとき，Ｔoi の大きさにはあまり差がないが，Ｔpi の値は通常コンパイラの方が小さい.つまり相対的にＴoiが大きくなることになり，コンパイリングシステムでのオーバヘッドの割合が大きくなる．Ｔoiの値を抑えるためには，比較的大きな処理をまとまって実行するプロセスを，限定して生成すればよい．この制御を行うために，我々はプロセスに対してレベルの概念を与え，これをもって生成過程を制御した．このレベルはプロセスの親子関係を示す順位であり，新しく子プロセスが生成されれば，親プロセスのレベルに１を加えたものが子プロセスに与えられる．そして，プロセスの生成段階においてレベルをチェックし，あらかじめ定められたレベル制限値以上のときは新たなプロセスの生成を制止する．本稿ではこの制御を，レベルによる並列化の制御という．

2.3 並列評価の際の負荷の分散

一方，インタプリタ，コンパイラを問わず引数の並列評価を行う場合に考えなければならないことがある．すべての Lisp 関数の引数を並列評価することは，オーバヘッドのコストが高く処理効率の低下をまねくおそれがある．そこで，2.1 において述べたように，エディタなどによって Lisp プログラム上で任意に指示ができ，並列処理の選択を可能とする関数タイプを導入した．コンパイラにおいても同様に，並列処理を行う関数を選択できるようにすることによって，処理効率が改善されることが期待できる．

しかし，'*'を付加することによって得られる図１に示す並列評価関数を考える．

```
(*FOO (CAR X) (BAR X) (BAR (CDR X)))
```

図１．アンバランスなプロセスを生み出す関数呼び出しの例

関数 BAR の処理が CAR に比べて大きければ，プロセ

スの大きさにアンバランスが生じる．つまりこの例の場合，３つのプロセスを生み出すが，(CAR X) の処理を行うプロセスは他の２つのプロセスに対し非常に小さくなり，各プロセッサに対する負荷の分散に不均衡が生じる．これはアイドル状態のプロセッサが多く存在することとなり，並列処理の効率が悪くなる．

コンパイラは インタプリタ と違い，あらかじめ Lisp プログラムを解析しコードを生成するので，静的に解決可能な処理はコンパイル時に行うことができる．並列処理の制御に関していえば，並列化の手段が種々存在していても，コンパイリング中に適用されるならば実行時には負担にならない．そこで本コンパイラでは，引数評価の優先順位やプロセスの生成についての制御方式を詳細に設定できるように設計した．

本研究で作成した並列コンパイラは，2.2,2.3 において示したように，特に並列処理のためのプロセス生成に関する制御をインタプリタに比べ詳細に指定できるように設計し，インプリメントした．実際に種々のプログラムを与えその動きを測定することにより，並列指示の自動化に対する指針ともなる．

③ 並列処理指示の記述法

この章では，並列コンパイラのために新しく追加された<評価ｺﾝﾄﾛｰﾙﾘｽﾄ>とともに，引数の並列処理を期待する関数呼び出しの記述法，ならびにその具体例をあげて説明する．

3.1 言語仕様

並列コンパイラで使用する Lisp 言語は，基本的には Lisp 1.5 に準拠した EVLIS マシンの Lisp を使用する．②で示した機能を実現するために，引数の並列評価を期待する関数呼び出しは，Parallel-Lisp インタプリタでの記述に加え，次に示すものとなる．

(* <評価ｺﾝﾄﾛｰﾙﾘｽﾄ> <関数名> <引数>)

<評価ｺﾝﾄﾛｰﾙﾘｽﾄ>::=<順位評価ﾘｽﾄ> | <empty>
<順位評価ﾘｽﾄ>::=(<順位ﾌﾟﾛｾｽ並び>)
<順位ﾌﾟﾛｾｽ並び>::=<順位ﾌﾟﾛｾｽ> |
<順位ﾌﾟﾛｾｽ並び> <順位ﾌﾟﾛｾｽ>
<順位ﾌﾟﾛｾｽ>::=<引数番号> | <並列評価ﾘｽﾄ>
<並列評価ﾘｽﾄ>::=(<並列ﾌﾟﾛｾｽ並び>)
<並列ﾌﾟﾛｾｽ並び>::=<並列ﾌﾟﾛｾｽ> |
<並列ﾌﾟﾛｾｽ並び> <並列ﾌﾟﾛｾｽ>
<並列ﾌﾟﾛｾｽ>::=<引数番号> | <順位評価ﾘｽﾄ>
<引数番号>::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | ····

<順位ﾌﾟﾛｾｽ並び> は記述されている<順位ﾌﾟﾛｾｽ>に左から順に優先順位がついていることを示す．したがって，副作用はこの順に作用される．

<並列ﾌﾟﾛｾｽ並び> は記述されている<並列ﾌﾟﾛｾｽ>がすべて並列に評価され，そのプロセス間の優先順位はない．

<引数番号> は，与えられた引数に対して，左から右へ 1,2,3,··· と順につけた番号である．ひとつの <評価ｺﾝﾄﾛｰﾙﾘｽﾄ> の中に現れる <引数番号> は重複してはいけない．また，存在しない <引数番号> を記述してはいけない．

記述されていない引数番号に対応する引数は，事前に逐次的に処理される．この処理のための新たなプロセスは生成しない．

<評価ｺﾝﾄﾛｰﾙﾘｽﾄ> が <empty> のときは，すべての引数に対して左から右への優先順位がついているとみなす．

3.2 ４引数の関数呼び出しの例

具体的に４引数の関数に <評価ｺﾝﾄﾛｰﾙﾘｽﾄ> を与えた例を示す．

(* (((2 4) 3)) fn arg1 arg2 arg3 arg4)
図２．４引数の関数呼び出しの例

並列評価の手順は，<評価ｺﾝﾄﾛｰﾙﾘｽﾄ> が示している．ここではまず，表記されていない引数 arg1 が先に評価される．そののち２つのプロセスが生成され，ひとつは arg2,arg4 の引数がこの順に優先順位をもって評価されるもの，ひとつは arg3 が評価されるものである．この２つのプロセスは，優先順位と関係なく並列に処理される．全ての値がそろった後に関数 fn が適用(apply)される．arg2,arg4 それぞれと arg3 との引数評価に優先順位はない．

以上のように，ユーザが関数に与えられる引数の評価順序を任意に決めることが可能である．

④ EVLIS マシンと並列コンパイラ

この章では，並列コンパイラが出力するオブジェクトを実行する Lisp 並列処理マシン ― EVLISマシン ― について簡単にふれ，コンパイリングシステムにおける動作環境，ならびにコンパイラの概要，そのインプリメンテーションについて説明する．

4.1 EVLISマシンの構成

EVLISマシンのハードウェア構成について，簡単に説明する．（図３）

・EVAL-Ⅱプロセッサ[3][4]

マイクロプログラム制御方式の高速リスト処理専用プロセッサである．EVLISマシン には現在２台実装し

ている．リスト処理ではオペランドとしてポインタが頻繁に取り扱われることを考慮してバス上ではデータとアドレスを区別していない．さらに，リストを手繰る CARCDR 演算機能を備え，ALU 演算，分岐演算と同時に１命令内で実行可能であるなど，リスト処理を高速に行うアーキテクチャを採用している．したがって，既製のマイクロプロセッサ等はいっさい使用していない．ハードウェアスタックなどに使用可能なスクラッチパッドメモリ(SM)は 21ビット幅 4K語，制御記憶(WCS)は 48ビット幅 8K語ある．WCS は I/Oプロセッサを通して書換え可能である．

・メインメモリ(MM)

リストデータ，スタックフレーム等の共有データを格納している．プロセッサ間の通信用にメイルボックスとしても使用している．現在 ８バンク 実装しており，総容量は 40ビット幅 32K語である．並列処理における排他制御を行うためのバンクロック機構を備えている．

・q-buffer[9]

各プロセッサ共有の高速 FIFO メモリで，処理待ちプロセスへのキューを保持している．総容量は 21ビット幅 4K語である．

・I/Oプロセッサ[9]

Lisp の入出力を担当している．また，マシンのコントロールプロセッサともなる．

他に，EVLISマシンには，各 EVAL-Ⅱプロセッサ 毎にハードウェアのデバッグ支援 となる Diagnostic Interface(DI)[3][4] が実装されており，I/Oプロセッサを通してプロセッサの状態を知ることができる．メインメモリの競合を測定する装置[10]もハードウェアで組み込まれている．

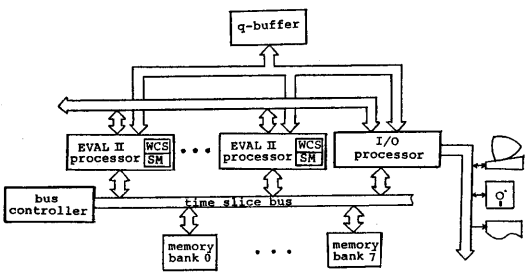

図３．EVLISマシンの構成

4.2 並列コンパイラとそのオブジェクト

(1) コンパイラの特徴

並列コンパイラは，Lisp で記述されている．既に稼働しているプロセッサ単体用 Lisp コンパイラ[7]に，並列処理の機能を追加した．ピープホールオプティマイザを含めて約4,000行である．また，オブジェクト用の実行時のサブルーチンとして，マイクロアセンブラで記述されたものがソースコードで約3,800行である．以下にこのコンパイラの概要を述べる．

・一括コンパイル

処理高速化のため，コンパイルする関数から呼ばれる関数は，すべて一括してコンパイルする．これにより，CALL する関数はすべて SUBR タイプとなっており，実行時に関数の形態などを調べなくてもよい．

・並列実行環境の保存

並列処理における多重環境を実現するために，並列コンパイラではひとつのプロセスの実行環境をひとつのスタックフレーム（以下単にフレームという）に保存している．このフレームは共有メモリである MM 上に作られる．フレームの詳細については，(3)で述べる．

・変数の束縛，保存

並列コンパイラは，すでに稼働している並列インタプリタとの互換性をはかるため，ダイナミックスコーピングを採用している．ローカルな変数は，その値を高速アクセスの行える SM 上のスタックに保存している．グローバルな変数は，並列実行環境においてはその変数をアクセスするプロセスの数だけの動作環境が存在することがあり，変数の束縛においても多重環境を持たせなければならない．本コンパイラにおいては，グローバル変数の束縛にリスト構造を採用することによって多重環境を実現している．それぞれのプロセスはその環境であるリストへのポインタを保持している．並列処理のためのプロセスを生成した場合，スタックの内容を新しいスタックフレームへ書き出すことによって，環境を伝播する．

・レベルによる並列化の制御

レベルによる並列化としてのプロセス生成の制御はコンパイラへのパラメータとして関数ごとに与え，コンパイラをレベル制御モードにしておく．このモードのとき，オブジェクトにレベル制御用のコードが付加される．レベル制限値の設定は，実行時に処理系へコマンドで与える．

・メインメモリアロケーション

現在リストデータ用に４バンク(16Kセル)，フレーム用に３バンク(768 フレーム)，ベースレジスタやその他の作業用に１バンク使用している(図４)．これは，システム立ち上げ時に簡単に変更可能である．

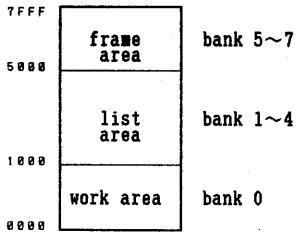

図４.メインメモリアロケーション

現在の構成を図４に示す．なお，排他制御のためのバンクロックの関係上，これら３種を同一バンク上に設定することはできない．

(2) コンパイラの動作

コンパイラは，関数の引数並列評価を指示した記述を見つけると〈評価ｺﾝﾄﾛｰﾙﾘｽﾄ〉を解析し，その中の一つの〈並列評価ﾘｽﾄ〉に対して，一つの親フレームと
その中に現れる〈並列ﾌﾟﾛｾｽ〉の数に対応する子フレームとを MM 上に作成するオブジェクトを作り出す．ここで，コンパイラに対しあらかじめレベルによる並列化の制御が指定してある場合，コンパイラは同時にレベルチェック用のオブジェクトと逐次処理を行うオブジェクトをも出力する．現在のところ副作用のリカバー処理に関してはインプリメント中であり，〈順位評価ﾘｽﾄ〉に現われるプロセスは逐次処理を行っている．

(3) オブジェクトの実行時の動き

コンパイラによって出力されたオブジェクトの実行時の動きを 図２ の例をもとに説明する．全体の処理の流れを 図５a,b に示す．

関数 fn を含むプロセスを実行していたプロセッサは，この関数 fn に与える引数の評価シーケンスに入ったとき，最初に逐次評価引数 arg1 の評価を行い，その値をスタックに退避する．

次に，並列評価のプロセスを生成するため，フレームを作成する．まず，現在の環境を保存しておくための親フレームを作る．このフレームへ，次に作成する２つの子フレームへのリンク情報と，現在の環境やスタックの内容，処理再開番地などを格納する．次に，２つのプロセス（arg2,arg4 の評価プロセスと arg3 の評価プロセス）にそれぞれ対応する２つの子フレームを作成する．このフレームは，子プロセスの処理開始番地，親フレームへのリンク情報，ローカル変数の値やグローバル変数リストへのポインタなどを格納する．このときのフレームの状態を図６に示す．

作成されたこれら２つの子フレームへのポインタを q-buffer へ登録し，自分自身は別の処理待ちプロセス

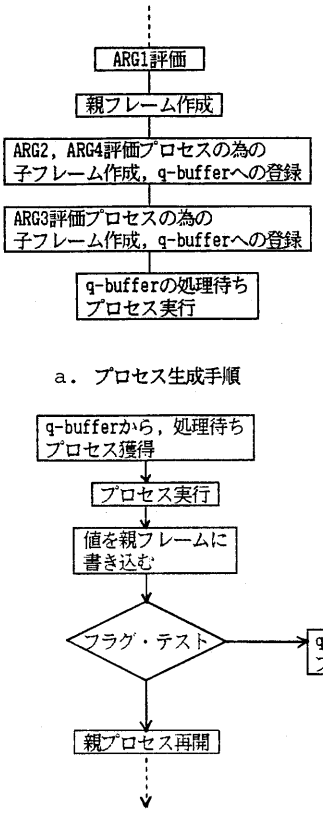

図５.引数並列評価時の処理の流れ

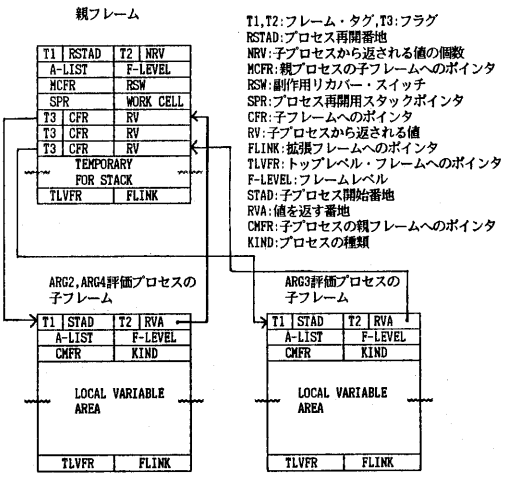

図６.フレームの構造

の獲得にいく．

q-buffer から子プロセスを獲得したプロセッサの動きは，まず，このプロセスのフレームから処理を行うために必要な環境などを，スタック，レジスタ等に取り込む．そののち，評価シーケンスへ移る．値が求まればそれを上述の親フレームの RV の位置に書き込む．ここで，子プロセスの処理は終了する．このとき，値を返すと同時に親フレームのフラグ(T3)をチェックし，このプロセスから生成されたすべての子プロセスの値が返っていれば，このプロセッサ，つまり最後に値を返したプロセッサが親プロセスの処理を再開する．この例では，親フレーム中の３つの値とスタックから arg1 の値合計４つをレジスタに転送し，また，親フレームのスタック退避エリアからスタックの内容を復帰し，関数 fn を呼ぶ．

以上が一例としての引数並列評価を伴う関数呼び出し処理の流れである．

レベルによる並列処理の制御を行うモードにおいてコンパイルされたオブジェクトの場合，arg1 の処理にはいる前に現在のプロセスのレベルをチェックする．このときの値がシステムにあらかじめ設定されている値に等しければ，逐次処理用オブジェクトへ分岐し，いっさいフレームを作らずに逐次的に引数評価を行う．

(4) ＧＣ

並列コンパイリングシステムでは，リストセルとフレームの２種のデータに対してＧＣを行う必要がある．現在，リストセルは，フリーリストが底をついた時点で開始するいわゆる STOP & MARK 方式で行っている．これは I/O プロセッサをコントローラとして複数台の EVAL-Ⅱプロセッサ がマーキング，コレクティングをそれぞれ並列に実行する．フレームＧＣは，親フレームの処理を再開するプロセッサが，その子フレームと親フレームとを即時回収する方式である．

⑤実行結果および考察

この章では，この並列コンパイラによって実際にコンパイルし実行した結果について示す．また，いくつかの問題についてプロセッサの稼働状況を測定し，考察を行う．

5.1 実行時間

並列コンパイラによってコンパイルした結果の例として，Lisp コンテスト[12]等の問題についての実行時間を表１に示す．ここで，単体とあるのはプロセッサ単体用 Lisp コンパイラにおいてコンパイルを行ったものであり，並列とあるのは Parallel-Lisp コンパイラにおいてコンパイルを行ったものである．この並列での実行時間は，一例としての<評価ｺﾝﾄﾛｰﾙﾘｽﾄ>を伴った並列化指示での例である．

表１.単体および並列指示における実行時間の比較

関　数	単体(S)	並列(P)	比率(S/P)
Tarai-5	1415.7	710.4	1.99
List-Tarai-4	155.0	98.3	1.58
Fibb-20	114.6	59.5	1.93
Srev-6	11.1	8.4	1.32
TAK-18-12-6	295.8	151.1	1.96
8-Queen[13]	467.7	377.7	1.24
Bit-A-7	25.6	24.1	1.06

測定時のEVAL-ⅡおよびＭＭのクロックはそれぞれ100ns,61ns である．

5.2 プロセッサの動特性

本研究での並列コンパイラは，オーバヘッドの低減とプロセスの効率的な配分を目的に作成した．この目的が達成されているかどうかを調べる.ここで，プロセッサの処理内容を次の４つに分ける．

(1)エバリュエーション(EVALUATION)
…リスト処理状態

(2)メモリ競合(MC)
…メモリ競合による待状態

(3)オーバヘッド(OH)
…並列環境管理のための処理を行っている状態

(4)アイドル(IDLE)
…処理するプロセスがない状態

以下，これら４つの状態をそれぞれ EVALUATION,MC, OH,IDLEで示す．

明らかに，EVALUATION の割合が大きければ処理効率がよいといえる．IDLE の割合が大きければ，分割されたプロセスのバランスが悪く負荷がうまく分散されていないことになる．OH が大きければ，小さなプロセスを多く作り過ぎオーバヘッドがめだったことになる．MC は共有メモリを使用する限りにおいては避けることはできないが MM へのアクセス頻度が少なければ小さな値を示す．オブジェクトを実行したとき，これら４つの状態がどの様に割り振られているかを測定すれば当初の目的が達成できたかどうかを知ることができる．

測定は，Diagnostic Interface とメモリ競合測定装置および内蔵のタイマで行った．これらは共にハードウェアで実装されており，測定に際しての影響はない．

まず，<評価ｺﾝﾄﾛｰﾙﾘｽﾄ> の効果を調べるために，種々の関数にいくつかの <評価ｺﾝﾄﾛｰﾙﾘｽﾄ> を与えて測定した．ここでは，List-Tarai 関数および Tarai 関数(図７)の結果として，実行時間を表２に，動特性を

```
(DE TARAI (LAMBDA (X Y Z)
    (COND ((GREATERP X Y)
           (* <評価コントロールリスト> TARAI
                        (TARAI (SUB1 X) Y Z)
                        (TARAI (SUB1 Y) Z X)
                        (TARAI (SUB1 Z) X Y)))
          (T Y)
))
(DE LIST-TARAI (LAMBDA (X Y Z)
    (COND ((LESSP (CAR X)(CAR Y))
           (* <評価コントロールリスト> LIST-TARAI
             (LIST-TARAI (COPY (CDR X)) Y X)
             (LIST-TARAI (COPY (CDR Y)) Z X)
             (LIST-TARAI (COPY (CDR Z)) X Y)))
          (T Y)
))
```

図７.List-Tarai,Tarai 関数の定義

表２.種々の<評価コントロールリスト>による実行時間の比較

<評価コントロールリスト>	Tarai-4	List-Tarai-4
① ((1 2 3))	402.2ms	539.3ms
② (((1 2) 3))	404.5	469.8
③ ((1 (2 3)))	333.9	467.5
④ (((1 3) 2))	380.7	529.3

測定時のEVAL-ⅡおよびMMのクロックはそれぞれ200ns,122ns である.

図８に示す．ここでは，レベルによる並列評価の制御を行っていない．

Tarai の場合，プロセスを３つ生み出す①に比べて③の実行時間が２割ほど速くなっている．図８から，<評価コントロールリスト> によるプロセス制御によってオーバヘッド(OH)が減少していることがわかる．④では，OH の時間は③と大差ないが，IDLE 時間が増加している．つまり，プロセスのプロセッサへの配分が現在のマシン構成にフィットしていないことがわかる．List-Tarai においても問題の性質上 Tarai と同様の傾向を示すが，微妙な点で違いがある．これは，オブジェクトの大きさが違いプロセスの配分に差があったためだと思われる．<評価コントロールリスト> での並列化制御による処理効率向上は，与える問題にかなり依存し，静的に類似したものにおいても，場合によっては非常に違う傾向を示す．実行時に与える引数によっても効率が変化することもあり，自動的に効率のよい並列化の指示を行う場合には，そのプログラムに対して与えられる予定である引数の情報などをユーザがコンパイラに対し示す必要がある．

つぎに，レベルによる並列処理の制御の効果について種々の問題についていくつかのレベルでの実行時間の変化と動特性を測定した．ここでは,8-Queen [13] および List-Tarai-4 の関数についてその結果を図９,図１０に示す．

レベルによる並列化の制御を行うことにより，実行

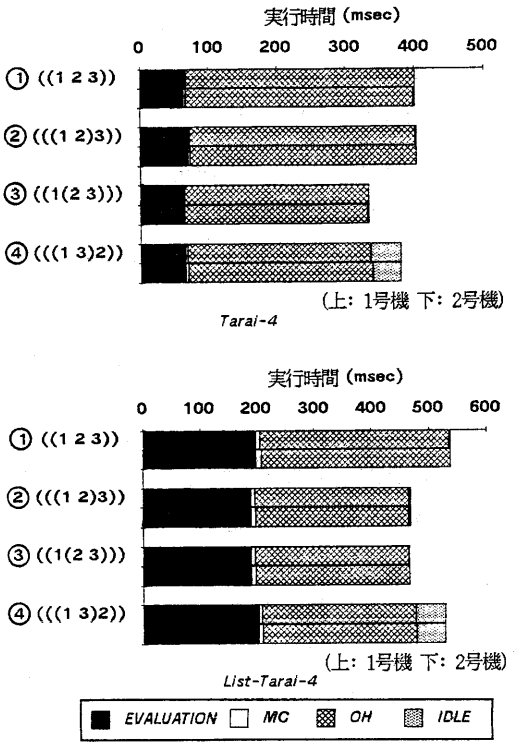

図８．Tarai-4,List-Tarai-4 における プロセッサの動特性

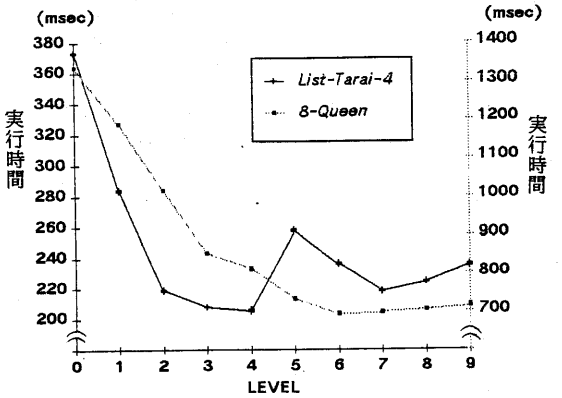

図９.レベル制御を行ったときの実行時間の変化

測定時のEVAL-ⅡおよびMMのクロックはそれぞれ200ns,122ns である.

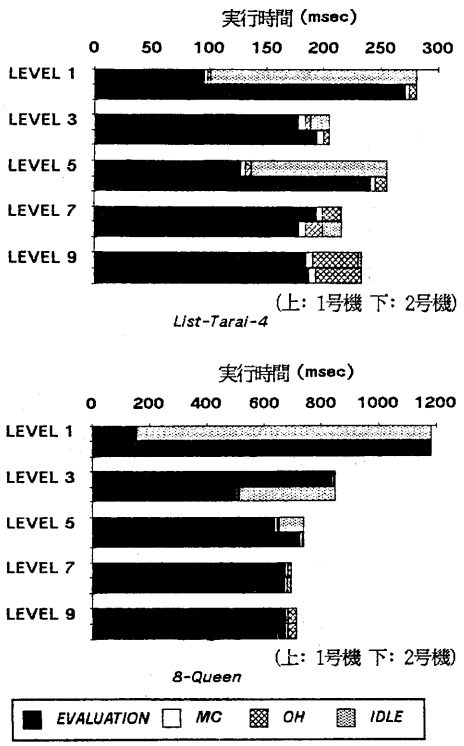

図１０.List-Tarai-4,8-Queen における
プロセッサの動特性

時間にかなりの変化がみられる．レベル制限値が小さい場合，プロセスの配分が悪くプロセッサのアイドル時間(IDLE)の割合が多い．レベル制限値が大きくなると多くのプロセスが生成されるため，プロセッサへの配分はよくなり IDLE は減少するが，かわりにプロセス生成･消滅のためのオーバヘッド時間(OH)が増加する．レベル制限値の最適値は実行する関数にかなり依存するが，並列コンパイラではコンパイル時にレベル制限モードにしておき，実行時に制限値を調整することによって比較的容易に最適値を選ぶことが可能である．

6. まとめ

本研究によって，コンパイリングシステムにおいても，十分な並列度を得ることが可能であることがわかった．とくに我々が提案，導入した〈評価ｺﾝﾄﾛｰﾙﾘｽﾄ〉による制御とレベルによる並列化の制御により，より並列度の高いコンパイリングシステムとしての目的を達成できた．本稿で述べたシステムにおいては，並列化の制御はすべて静的な状態で行っている．現在，実行時での動的な状態による並列プロセス生成制御，例えばアイドルプロセッサの数やキューにある処理待ちプロセスの数などをパラメータとした実行時においての制御についても研究を行っている．

今後，ユーザのこれら処理系へのパラメータ指示がなくても，自動的に高効率なオブジェクトを生成するプリコンパイラの開発などを行ってゆく．

EVLIS マシンには EVAL Ⅱプロセッサがすでに２台実装されているが，現在３台目を追加すべく制作中であり，３台のプロセッサでの並列処理の動特性をも測定し，それをもとに，より一層のスピードアップをはかる．またプロセッサの台数などのパラメータをあらかじめ与えて，その状態での最適なコードを生成する，より柔軟性のある並列コンパイラの作成についても研究中である．

［参考文献］

(1)安井　裕 他：LISPでの並列処理における動特性とEVLISﾏｼﾝの構成,情処,記号処理資料10-4(1979)
(2)前川博俊 他：試作 EVLIS ﾏｼﾝのEVAL Ⅱと開発支援機能,情処,記号処理資料17-1(1982)
(3)前川博俊 他：高速LISPﾏｼﾝとﾘｽﾄ処理ﾌﾟﾛｾｯｻEVAL Ⅱ,情処論文誌,Vol.24,No.5,pp683-688(1983)
(4)Maegawa,H. et al.：Fast LISP Machine and List Evaluation Processor EVAL-Ⅱ,JIP,Vol.8,No.2,pp121-126(1985)
(5)西川　岳 他：EVLISﾏｼﾝの並列処理ｱﾙｺﾞﾘｽﾞﾑとそのｲﾝﾌﾟﾘﾒﾝﾃｰｼｮﾝ,情処第23回全大,4H-7(1981)
(6)斎藤年史 他：高速LISPﾏｼﾝとﾘｽﾄ処理ﾌﾟﾛｾｯｻEVAL Ⅱ,情処論文誌,Vol.24,No.5,pp689-695(1983)
(7)高橋俊樹 他：EVLISﾏｼﾝにおけるLISPｺﾝﾊﾟｲﾗ,情処第32回全大,4G-11(1986)
(8)西開地秀和 他：EVLISﾏｼﾝのLISP並列処理における動特性,情処第29回全大,6B-1(1984)
(9)土井俊雄 他：EVLISﾏｼﾝの管理とI/Oﾌﾟﾛｾｯｻ,情処第23回全大,4H-8(1981)
(10)西開地秀和 他：LISP並列処理ﾏｼﾝ－EVLIS ﾏｼﾝ－の動特性測定と評価,情処,記号処理資料31-9(1985)
(11)安田弘幸 他：EVLISﾏｼﾝにおける並列Lispｺﾝﾊﾟｲﾗの実現,情処第33回全大,2E-5(1986)
(12)奥野　博：第3回Lispｺﾝﾃｽﾄおよび第1回Prologｺﾝﾃｽﾄ報告,情処,記号処理資料13-4(1985)
(13)中西正和：Lisp入門,近代科学社