ＡＩＰ－ＬＩＳＰ：（６）第２版の概要

7P-6

中村明　河込和作　宮本出　星野康夫
（株）東芝

０．はじめに

我々は、ＲＩＳＣ風アーキテクチャを持つ独自開発プロセッサＡＩＰ上に、Ｃｏｍｍｏｎ　Ｌｉｓｐに準拠し実行専用システムを指向したＬｉｓｐ処理系、ＡＩＰ－Ｌｉｓｐ第１版を開発した。

現在、ＡＩＰ－Ｌｉｓｐ第１版は動作中であり、性能評価を行った結果、当初の性能目標を達成していることを確認した。

我々は、次のステップとして、第１版開発の経験に基づいて、プログラム開発環境を充実したＡＩＰ－Ｌｉｓｐ第２版の開発を計画している。

本報告では、ＡＩＰ－Ｌｉｓｐ第２版に実装を計画している機能について、その概要を報告する。

１．ＡＩＰ－Ｌｉｓｐ第１版の概要

ＡＩＰ－Ｌｉｓｐ第１版は、既存のＬｉｓｐアプリケーションを、高速に動作させることを主目的にした、実行専用システム指向の処理系であり、次のような特徴を持っている。

１）　クロスコンパイラによるスタティックリンク方式

ＡＩＰ－Ｌｉｓｐ第１版では、実機であるＡＩＰ上にコンパイラは存在せず、ホスト計算機上の既存Ｃｏｍｍｏｎ　Ｌｉｓｐ処理系（ＣＬＩＳＰ）で動作するクロスコンパイラによって、ＡＩＰ標準形式のオブジェクトファイルを作成する方式を採用した。

オブジェクトファイルは、クロスリンカによって単一のロードモジュールにまとめられ、ＡＩＰにダウンロードされて実行される。

クロスコンパイラ自体は、Ｃｏｍｍｏｎ　Ｌｉｓｐで記述されており、ＡＩＰ－Ｌｉｓｐに固有の実行時システムの存在を全く前提にしていないので、優れた移植性を持っている。

２）　オブジェクト性能を重視した最適化コンパイラ

当初の性能目標を達成するため、コンパイラは、コンパイラ自体の動作速度よりも、生成されるオブジェクトコードの性能を重視して設計されている。

関数呼び出しは、同一ファイル内で定義されていなくても、コンパイル時に呼び出し先が確定している限り、シンボルの関数セルを通した間接呼び出し（ｓｌｏｗ－ｌｉｎｋ）ではなく、ｃａｌｌ命令による直接分岐（ｆａｓｔ－ｌｉｎｋ）を行うオブジェクトコードが生成される。

オブジェクト性能やメモリ効率とトレードオフの関係にあるデバッグ支援機能については、オブジェクトコードの速度性能を優先した結果、全く組み込んでいない。

３）　モジュール化された実行時システム

ＡＩＰ－Ｌｉｓｐの実行時システムは、最も低レベルのＩ／Ｏ処理、例外処理等を除き、全てＬｉｓｐで記述されている。そのため、Ｃ言語等のいわゆる手続き型言語で実行時システムを記述するのと違って、統合性／保守性に優れた構造を実現している。

実行時システム内部は、階層化されており、機能ごとに独立性の高いモジュールの集まりになるように設計されているので、再構成が容易である。

２．コンパイラのネイティブ化と高速化

ＡＩＰ－Ｌｉｓｐ第１版では、前述のように、コンパイラはホスト計算機上で動作するクロスコンパイラであったが、ＡＩＰ－Ｌｉｓｐ第２版では、コンパイラをＡＩＰに移植し、実機上で動作するネイティブコンパイラとする。

ネイティブコンパイラとすることによって、コンパイル速度の大幅な向上が期待できる。これは、プログラム開発工程の短縮に大きく貢献する。

また、第１版において、コンパイル時環境と実行時環境の食い違いによって発生し得る不具合を回避するために存在していた、トップレベル形式の記述制限も解除されると考えられる。

我々は、コンパイル速度向上のため、コンパイラのネイティブ化以外にも、次のような変更を計画している。

１）　パス構造の見直し・統合化（クリーンナップ）
２）　アセンブルフェーズのインコア化

３．インクリメンタルローダの実装

現在のＡＩＰ－Ｌｉｓｐ第１版は、スタティックリンク方式を採用しているので、アプリケーションプログラムを作成するユーザはＣ言語等で記述されたプログラムと同様な手順で、コンパイル・リンクを行う必要がある。

ＡＩＰ－Ｌｉｓｐ第１版は、処理対象として、既に他のＣｏｍｍｏｎ　Ｌｉｓｐ処理系上で開発され充分にテストが行われたプログラムを前提としていたので、スタティックリンク方式を採用することができたが、第２版において充実させようとしているプログラム開発環境では、コーディング・コンパイル・テストを数多く繰り返す必要があり、一般的なＬｉｓｐ処理系が持っているインクリメンタルロード機能の実装が、実用上不可欠である。

インクリメンタルロード機能は、関数定義やデータの内容をローディング可能な形式（ｆａｓｌ形式）でファイルに書き出すダンパ（ｄｕｍｐｅｒ）、ｆａｓｌ形式ファイ

Outline of AIP-LISP Version Two
Akira NAKAMURA, Kazusaku KAWAGOME, Izuru MIYAMOTO, Yasuo HOSHINO
TOSHIBA Corp.

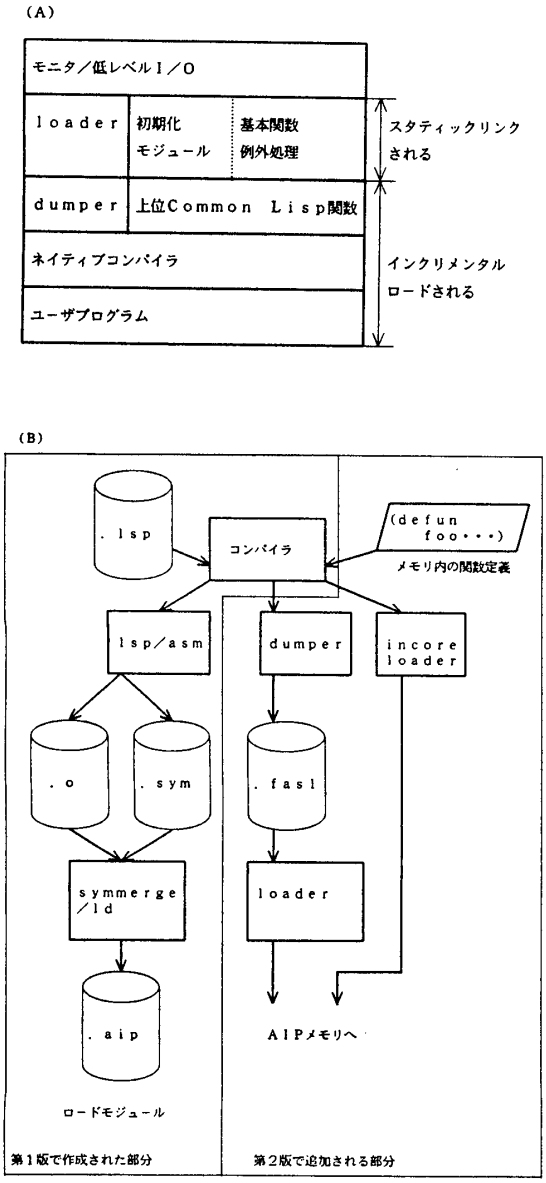
ルを読み込み、メモリ上に関数定義やデータを配置するローダ（ｌｏａｄｅｒ）からなる。

ローディング時の外部参照解決は、外部のリンカを使用せず、インコアにて行う予定である。

我々は、第１版が持っている性能を維持するため、ｃａｌｌ命令による関数呼び出しを使ったまま、関数の再定義を可能とすることを前提にして、リンク方式の検討を進めている。

インクリメンタルローダ実装後は、現在、全てスタティックにリンクされている実行時システムの大部分は、起動後にロードされることになり、システム構成は図１に示すようなものとなる。

図１　　ＡＩＰ－Ｌｉｓｐ第２版のシステム構成



４．デバッグ機能の向上

前述した、インクリメンタルロード機能と並んで、プログラム開発環境で重要なのは、プログラムのデバッグを支援するツール群の機能と操作性である。

ＡＩＰ－Ｌｉｓｐ第１版は、トレーサ、ステッパ、インスペクタのような、独立性の高いデバッグ支援機能と、エバリュエータの構造に依存しない小規模なデバッガ、トップレベル上から、マシンレベルのデバッグを支援する機能等の基本的なデバッグ手段を実現している。

第２版では、本格的なプログラム開発に充分なデバッグ機能を実現するために、コンパイラやエバリュエータを見直し、高機能なデバッガの実装を目指す。現在のところ、次のような機能アップを検討している。

（イ）　コンパイラは、少なくとも標準の設定では、実行時に次の様な情報を保持できるようなコード生成を行うようにする。

１）　関数の呼び出し履歴
２）　局所変数（引数）名と格納場所
３）　パラメータ名と受取り形式

（ロ）　エバリュエータ（ｅｖａｌ）は、評価対象となる形式の呼び出し履歴・引数・局所変数情報を保持して、デバッガから参照可能にする。

５．まとめ

以上、我々が実現を検討しているＡＩＰ－Ｌｉｓｐ第２版の構想のうち、主として、プログラム開発機能の向上に関わる部分について報告した。

我々は、実行性能についても、実行時システムのチューニングや、現在のＡＩＰ－Ｌｉｓｐ第１版ではまだ充分に行われていない、大域的最適化、型情報による最適化等の実装によって向上させることを計画している。

よりバランスの良いシステムの構築を目指して、今後とも研究を継続する予定である。

参考文献

［１］斉藤　他：“ＡＩワークステーションの開発思想”第１回人工知能学会全国大会（１９８７）

［２］宮本　他：“ＡＩＰ－ＬＩＳＰの実現　（Ｉ）”第３６回情処学会全国大会（１９８８）

［３］G.L.Steele Jr.:"Common Lisp the Language", Digital Press,1984