分散マルチプロセッサの高速コンフィギュレーション技術

田村　修 †

分散マルチプロセッサ環境において，言語処理系をシステムに内蔵し，ソースプログラムをプロセッサ別にコンパイル，配信するしくみを提案する．開発環境の主要部分をシステムに取り込むことにより，多種多様なマイクロコントローラから構成されるシステムの内部アーキテクチャを隠蔽し，迅速な開発と長期的な保守管理を可能とする．

A Rapid Configuration Method for Distributed Multiprocessor

Osamu Tamura †

A built-in language processor compiles and distributes the source program for distributed multiprocessor system．　Its internal architecture is abstracted by the implemented compiler．　This method enables rapid development and long-term maintenance．

1．はじめに

コンピュータ応用機器の高度化に伴い，多数のプロセッサから構成されるシステムが増えている．自動車，産業用ロボット，センサーネットワークなどでは，疎結合された多数のプロセッサが連携動作する．これらのシステムでは一般に，多様な機能モジュールを個別に開発し，通信プロトコルで疎結合して統合管理する構造となっている[1)2)]．

従来，システム機能の高度化に際しては，強力なCPU にリアルタイム OS を搭載し，多数のジョブを集中処理させる構成が一般的であった．これは，組込みCPUの高性能化とともに扱い易いRTOSが普及したことにも拠るが，複数 CPU の共存による開発効率の悪化が敬遠されたことも一因と考えられる．しかしながら，例えば人間型ロボットなどでは，

・制御が単一 CPU の処理能力を超えている
・集中型では柔軟な拡張が困難
・配線の輻輳による信頼性の低下

などの問題が表面化してきており[3)]，処理の分散化は避けられない課題となりつつある．システムの分散プロセッサ化には

・負荷分散によるリアルタイム性の向上
・共通バス上の通信による，配線の引き回し削減

などの効果もあり，今後確実に採用が増加すると予想される．

多数のプロセッサから構成されるシステムを効率的に開発するアプローチとしては，システム内プロセッサの統一構成がある．すべての機能モジュールに同一あるいは同一アーキテクチャのマイクロコントローラを用いることで，各 CPU プログラムの開発環境を一本化するものである[3)]．しかしながら，例えば人間型ロボットでマルチプロセッサによる機能分散を図る場合，視覚処理，姿勢制御から，末端のセンサ入力，アクチュエータの位置決めまでには処理規模に大きな開きがあり，また必要とされる周辺機能も異なるため，無駄が大きくなり易い難点がある．

マルチプロセッサシステムにおいては，各 CPU へのプログラムのロード方法も問題となる．実験的なシステムでは，ハードウェアを構築したものの多数の CPU へのプログラム配信が課題となる例[4)]がしばしば見られる．昨今はインシステムプログラミングなどが容易になってきてはいるが，個々のプロセッサごとに接続し書き込む手間は大きい．接続を切り替えることなく任意のプロセッサの不揮発性メモリにプログラムをロードするしくみは，保守の面からも望ましいといえる．

さらには，複数プロセッサの協調動作を支援するプログラミング環境も重要となる[5)]．疎結合型の分散プロセッサでは，厳しいタイミングでの同期が必要とされる場合は少ないが，各プロセッサに共通の処理や固有の処理を明瞭に記述できるしくみは有用と考えられる．

本稿では，分散マルチプロセッサシステムにおいて
(1) 個々の CPU アーキテクチャをソフトウェアで隠蔽
(2) 複数プロセッサ向けプログラムをコンパイル配信する機能をシステムに内蔵

† 有限会社リカージョン
Recursion Co., Ltd.

することにより，多種多様な CPU で構成されるシステムへの迅速なコンフィギュレーションが可能となることを示す．また，この技術がローエンドのマイコンによる極めて小規模の分散プロセッサシステムにおいても適用可能であることを実例で確認する．

2. 基本構成

2. 1 アーキテクチャの仮想化

様々なアーキテクチャの CPU が混在するシステムでは，CPU ごとに開発環境を使い分ける必要が生じる．プログラム配信と保守を容易にするため，ここでは個々の CPU アーキテクチャをソフトウェア的に均質化する方法を用いた．各 CPU は，簡単な構造の中間コードインタープリタを実装することにより，仮想スタックマシンとして動作する．このスタックマシン用の中間コードを生成するコンパイラを用意すれば，すべての CPU へのプログラミング環境が共通化される．CPU の交換や追加に際しては，それぞれの CPU にインタープリタ部分のみを移植する．インタープリタ動作により実行効率は 1/2～1/10 程度に低下する．性能限界での処理を行う場合や低消費電力化が必須の応用では，インタープリタ部分にネイティブコードでの拡張が必要となる．とはいえ昨今のローエンドマイコンでは数～数十 MIPS の性能を持つものが普及し始めており，多くの応用においてはこの性能低下を許容し得る状況にあるとみられる．

2. 2 言語処理系の内蔵

仮想スタックマシン向けにコード生成するコンパイラであれば，どのようなプログラミング言語でもよいが，分散環境に依存した拡張がなされることや，長期的な保守を考えた場合，システム自体に処理系が内蔵されると都合がよい．分散マルチプロセッサシステムでは，システムの構成に応じたコンパイルやライブラリのリンク，あるいはプロセッサへのコード転送が必要となる場合が多い．これらの処理をシステム内で管理すれば，クロス環境との同期を取る必要がなく，プロセッサ増設や機能追加などの拡張に際しても，制御プログラムへの影響を軽減できる．言語処理系の内蔵は，マルチプロセッサ化に伴う個別のアーキテクチャ依存処理をシステム内に隠蔽する仮想化とも考えられる．また，CPU ごとに用意される PC 上の開発環境は，自身のバージョンアップや OS の進化なども含め，現状を長期にわたって正確に維持し難い問題がある．言語処理系をシステム側に移すことにより，長期に渡り，また場所を選ばずに保守を行うことが可能となる．

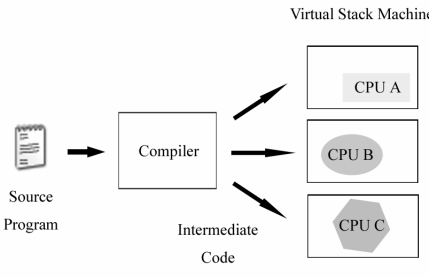


図 1 仮想マシン化によるコンパイラ共有
Fig. 1 The Unified Compiler for Virtualized Machines

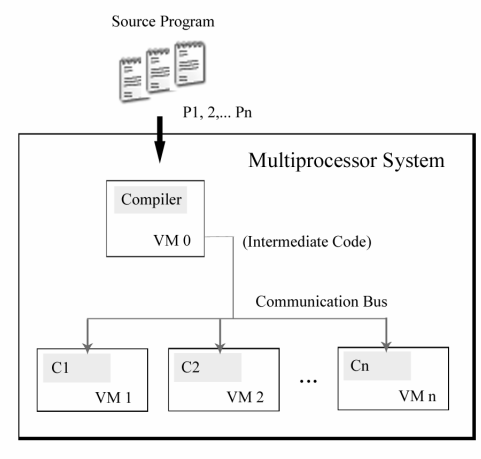


図 2 コンパイラの組込み
Fig. 2 Embedded Compiler

2. 3 コンパイラの拡張

分散型のマルチプロセッサシステムでも演算能力の高性能化を目的とするものでは，効率的な負荷分散や同期のしくみなどが重要となるが，ここではロボットなど，末端処理の分散により異なる機能を持った CPU モジュールが多数，協調動作するものを想定する．このようなシステムでは個々のプロセッサに固有のプログラムを用意する必要があり，プログラムテキスト中で指定されるプロセッサへのコード生成と配信が順に行われるだけでなく，共通の処理をまとめて記述できるものが扱い易い．ここでは独自開発した極小規模の組込み向け言語処理系を拡張し，個々のプロセッサへのプログラム記述を仕分けしてコンパイル，配信する機能を追加した．この差分配信機構付きコンパイラごと実機に組込み，分散プロセッサの効率的なコンフィギュレーションに有用な技術を検討する．

3. 差分配信のための言語処理

組込み機器開発用の言語処理系としては標準化が進み普及したCやJavaが望ましい. TinyCC, Wavaのように言語処理系を軽量化する試みもあり, 搭載メモリやCPUの処理能力が十分であれば検討に値する. ここでは搭載メモリが数Kバイト程度のローエンドマイコンを実装対象としたことや, 標準仕様の定着した言語に極端なサブセットや方言は好ましくないとの考えにより, 組込みコンパイル向けに独自設計した軽量言語 ForCy を用いた [6]. これは実装コストが小さく拡張性が高い FORTH をベースにフロー記述の見難さを改善しポインタ的操作を禁止するなどしたもので, 中間コードを 16bit の仮想スタックマシンとなるインタープリタで実行する. C 言語の制御構文と演算子の多くに対応し, PostScript に似た記述を特徴とする. データとして変数, 配列, 文字列を扱える。自己記述によるコンパイラの中間コードサイズは約 1.7K バイト, インタープリタは CPU により 1～2K バイト程度で実装できる. ワードを処理単位とし, 定義(関数)を積み上げてプログラムする.

単一のテキストファイル中に, 各プロセッサに共通な処理部分とプロセッサごとに固有の差分が記述されたプログラムを入力とし, 各プロセッサに対し(共通部分＋固有部分)を中間コード化して配信する(図3). プログラムのコンパイルと配信は逐次的に行われる. 共通部分では例えば通信プロトコルなど, 固有部分では CPU ごとのセンサやアクチュエータの個別動作などを記述する.

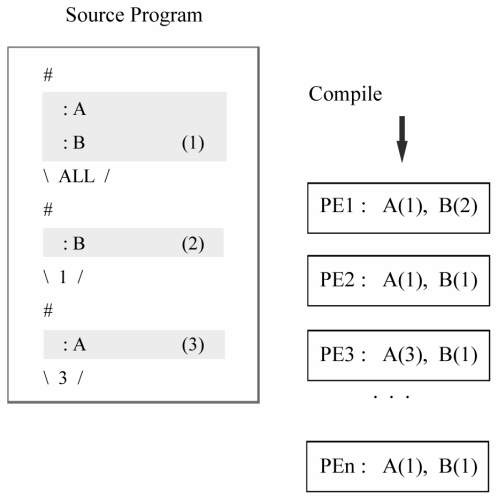


図 3 差分コンパイルと配信
Fig. 3 Differencial Compile and Distribution

固有部分の記述には,
(a) 共通部分にない関数を追加定義
(b) 定義済みの関数を再定義(オーバライド)
の 2 通りがある. (b)は, ほとんどのプロセッサで共通な処理となる関数がある場合や, 各プロセッサで異なる動作だが上位の呼び出し関数が共通な場合などに有用となる. ForCy では実行時の関数参照を間接的に行う中間コードを生成するため, コンパイル中に再定義が発生した時点で関数テーブルのポインタを入れ替える. このとき, 共通部分の参照されなくなった関数処理は破棄する. その他には, 最初にコンパイルされる共通部分の退避とコピー処理などを追加した.

図 3 において, '#'～'＼'の範囲がコンパイル区間, :A, :B がユーザ定義された関数であり, '/'でコード生成処理を完結して配信する. "ALL"指定されたコードが共有部分となり, 以降のコンパイル区間の開始時にリロードされる. "ALL /" ですべての, "1 /" でID番号1のプロセッサへ中間コードが配信される.

4. 実装

最も簡単な実例として, 欧米で学習教材として普及しているムカデ型ロボット Stiquito[7]に実装した. これは電流発熱型形状記憶合金により6本の足を動かすもので, オリジナル版では TI 社のマイクロコントローラ MSP430F1122 を用いて制御している. ここでは試みに各足にプロセッサ1個を割り当て, さらにコンパイル配信用プロセッサを持った, 7プロセッサ搭載型に改造して分散配信の実験を行った. コンパイル配信にはルネサス R8C/Tiny (R5F21154SP, 以降 R8C), 足制御にはマイクロチップ PIC12F683(以降 PIC) を用いた. プログラムはPCからRS-232C通信により R8C に送られ, プロセッサごとにコンパイルされた中間コードは各 PIC に共有バス経由で配信される.

R8C のプログラム領域には, インタープリタおよび 0.6K バイトの差分配信機能を加えたコンパイラの中間コードを格納する. 2K バイトのプログラム書き換え可能データ領域には, コンパイル中の共通コードおよび個別コードが書き込まれる. 各プロセッサへの配信には, アドレス指定式の通信プロトコルをパラレルポート上に実装した.

PIC にはインタープリタ部分(約 1. 5K 語)のみ実装し, 受信した中間コードは256バイトのEEPROM領域に格納する. 6本の I/O は, 1本を通信用, 2本を自由度2の足制御, 他を前後のプロセッサとの情報交換などに用いる. 歩行時には R8C は休止し, リング接続された

6CPU が交互に足の前進と引き上げ復帰を繰り返す．形状記憶合金の伸縮には，ソフトウェア生成した PWM パルスを加えている．

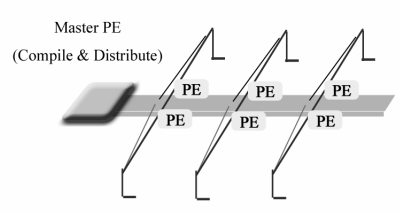
当初，各 PIC が隣接 CPU と半周期ずれる動作となる簡単な分散アルゴリズムを実装した．すべての CPU を同一プログラムで駆動できるが，起動時の交互動作への移行に手間取り，プログラムに工夫を要することが分かった．つぎに，最前列の足対が互いに反転動作するようにし，他の足は前方の足に反転追従するようプログラムした．自足の動作決定に参照する足が異なるため，この部分のみを差分プログラムした．処理が類似した CPU 多数を効率的にプログラミングできることを確認するとともに，分散プログラムにおいても処理の共通化が有効であることが分かった．

5. まとめ

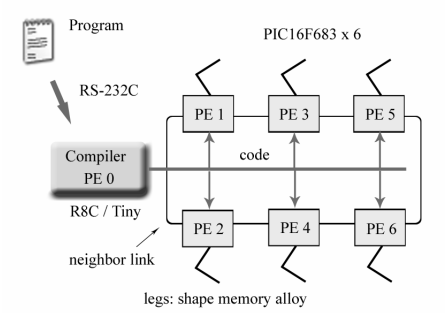
分散マルチプロセッサ環境において，言語処理系をシステムに内蔵し，外部からのソースプログラムをプロセッサ別にコンパイル，配信するしくみを提案し実装した．実際に搭載メモリの少ないマイコンを複合したマルチプロセッサシステムを1週間余りで構築し，多様なプロセッサより構成されるシステムの開発を迅速に行うとともに，保守を容易にする技術としての実用性を示した．

差分配信機構によるプログラミングでは差分記述の有用性を確認できたが，今後，既存のプログラミング言語で実現されている本格的な継承機構が，共通処理の多い分散マルチプロセッサ環境のプログラミングにも適用可能か検討したい．

アーキテクチャを仮想化するとともに言語処理系などのシステム構築技術を可能な範囲で機器側に移す発想は，システム補修を長期にわたり確実に行う技術としての発展も期待できる．今後の課題としては，より現実的なシステムでの実証を行うこと，分散環境で必要とされる言語処理系や OS 機能の要件を絞り込み開発の枠組みを整備していくことなどが挙げられる．



(a) overview



(b) connection

図4　分散プロセッサによるムカデ型ロボット
Fig. 4　Multiprocessored Stiquito

なお，本開発は 2005 年度 IPA 未踏ソフトウェア創造事業で支援されている．

参考文献

[1] 服部 博行，森川 聡久，“需要が拡大する自動車制御 OS を知る”，Design Wave Magazine，CQ 出版，pp. 97-98 (Dec 2004).

[2] 於久 健太郎，清水 健二，松坂 要佐，小林 哲則，“マルチモーダルロボット用マルチプロセッサアーキテクチャ”，情報処理学会計算機アーキテクチャ研究報告，150-10 (2002).

[3] 松井 俊浩，比留川 博久，石川 裕，山崎 信行，加賀 美聡，堀 俊夫，金広 文男，斎藤 元，稲邑 哲也，“ヒューマノイド・ロボットのための実時間分散情報処理”，信学技法， CPSY2003-45.

[4] 御牧 義，南沢 正之，上坂 達生，“ワンチップマイコンを用いたマルチプロセッサ・システムの試作”，情報処理学会マイクロコンピュータとワークステーション研究報告，54-1 (1989).

[5] 馬場 公光，甘田 苗，“疎結合マルチプロセッサシステムのためのシステム記述言語の設計”， 情報処理学会論文誌，Vol. 27，No. 1，pp121-123，(1986).

[6] 田村 修，“組込み機器のスクリプト制御技術”，組込みシステム技術に関するサマーワークショップ 2005 予稿集，pp.31-34，(2005)

[7] Conrad, J.M. :Stiquito Controlled!, IEEE CS Press, 2005.