論理型言語向き並列計算機 KPR のプロセス管理方式†

平田博章††* 柴山潔†† 萩原宏††

現在開発中の並列計算機 KPR は，論理型言語の高速 OR 並列実行を目指して設計されている．本論文では，(1) KPR のプロセス制御ユニット (PCU) のハードウェア構成方式，(2) PCU による動的負荷分散方式，(3) PCU のプロセス・スケジューリング方式，について述べる．PCU は，プロセスのスケジューリング，動的負荷分散の制御，ローカル・メモリの管理，プロセス (プロセッサ) 間通信の制御などのプロセス管理機能を効率良く実行するハードウェア機構を備えている．さらに，KPR プロセッサ間ネットワークは自動負荷分散機構を備えている．負荷割り付け戦略としては，負荷の表現方式を工夫することで，割り付け規則の簡単化を図った．また，PCU は，プロセスを深さ優先的にスケジューリングするが，探索を深めてゆく推論木の枝の数を制限し，それらの中では幅優先的なスケジューリングを行う．その数は，KPR のハードウェア構成で実現される並列処理能力に応じて決定される．

1. はじめに

われわれは論理型言語の高速実行を目指した並列計算機 KPR を開発中である[1),2)]．論理型言語の持つ並列性には，AND 並列性，OR 並列性，引数間並列性などがあるが，並列型の論理型言語は，AND 並列性と OR 並列性のどちらを積極的に利用するかで，AND 並列型言語と OR 並列型言語の２種類に大別できる．KPR が対象とする論理型言語 KPR-L は，OR 並列型である．１つの定義体中の各節は並列に評価され，また，１つの節本体のゴール列は左から右へと順に評価される．

KPR の実行モデルである「並列リダクション・モデル (PR モデル)[1),15)]」では，KPR-L プログラムの実行を AND-OR の探索過程とみなし，この AND-OR 木の各ノードに１つのプロセスを割り当てている．

本論文では，KPR のプロセス管理方式について述べる．すなわち，KPR のプロセス制御ユニットのハードウェア機構，動的負荷分散方式（プロセスの空間的配置）およびスケジューリング方式（プロセスの時間的配置）について詳述し，これらのプロセス管理機能の観点から他の並列処理方式との比較を試みる．

† The Process Management of a Logic Programming Language-Oriented Parallel Machine KPR by HIROAKI HIRATA, KIYOSHI SHIBAYAMA and HIROSHI HAGIWARA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室
* 現在 松下電器産業(株)
Matsushita Electric Industrial Co., Ltd.

2. KPR の概要

2.1 並列リダクション・モデル (PR モデル)

KPR の実行モデルである PR モデルは，KPR-L に内在する並列性を反映しており，OR ノードに対応するものには Or(O) プロセスと Database(D) プロセスの２種類がある．Oプロセスは推論定義体中の OR 関係にある節頭部の単一化を並列に行い，複数の子プロセスを同時に起動する．Dプロセスはデータベース定義体における事実の検索を並列に行う．一方，AND ノードに対応する Stream(S) プロセスでは，節本体の AND 関係にあるゴール列を左から右へと評価する．KPR-L の OR 並列性によって，一つのゴールに対して複数個の解が返されてくるため，ゴール列評価の過程で複数本の枝分かれしたストリームが得られる．S プロセスは，さらに，先 (左) のゴール呼び出しに対して返されてきた解を用いて次 (右) のゴール呼び出しを行う「サブプロセス」に細分される．

また，プロセス間通信のためのメッセージは，(i) プロセスの起動を要求する invoke デマンド，(ii) １つの解を返す success イベント，(iii) 解が (もうこれ以上) ないことを知らせる fail イベント，の３種類である．

原則的には，AND-OR プロセス木の構造を反映して，Sプロセスと，OプロセスまたはDプロセスとが交互に起動され，また，プロセス間通信は親子間でのみ行われる．しかし，PR モデルでは，プロセス間通信の高速化を図るため，付録に示すような最適化戦略を採用している．

なお，これらの最適化の可能性は，すべて，プログラムの実行前に静的に検出することができる．

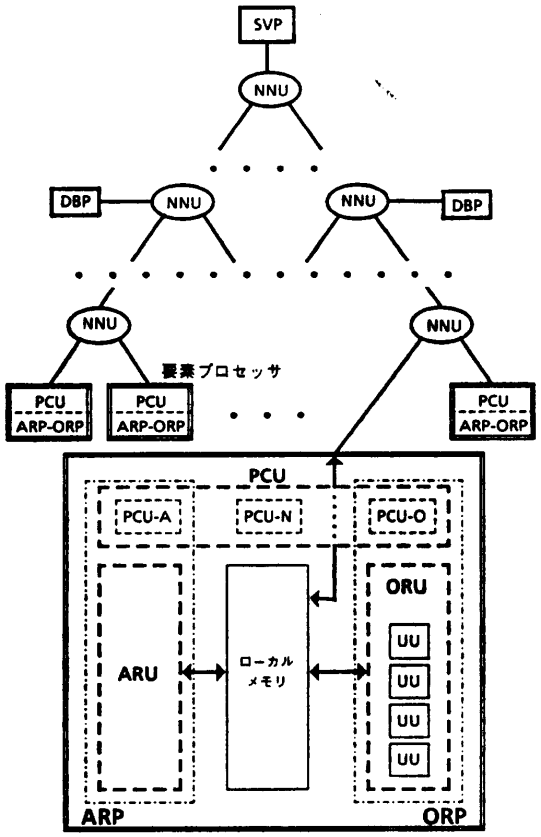


図 1 KPR のシステム構成
Fig. 1 System organization of KPR.

2.2 KPR のシステム構成

KPR のシステム構成は，**図1**に示すような，マルチプロセッサ構成方式を採用している[1]．並列リダクション・モデルにおけるS，O，Dの各プロセスは，それぞれ ARP (And Reduction Processor), ORP (Or Reduction Processor), DBP (DataBase Processor) と名付けられた専用プロセッサで処理される．したがって，KPR はヘテロジニアスな機能分散処理システムとみなすことができる．

要素プロセッサ (PE) は，一対の ARP と ORP とが密結合する形で構成される．すなわち，ARP が AND リダクション操作を行う ARU (And Reduction Unit) を，ORP が OR リダクション操作を行う ORU (Or Reduction Unit) をそれぞれ中心とし，両者をプロセス管理専用ハードウェア・ユニットである PCU (Process Control Unit) によって管理する構成方式が採用されている．また，ORU は，その内部に4つの UU (Unification Unit) を持ち，節頭部の単一化処理を最大4個まで同時に並列実行することができる．

各 PE は，多重バス構成をとる2分木状のプロセッサ間ネットワークによって結合される．ネットワークの中間ノードには NNU (Network Node Unit) が配置され，その中のいくつかには DBP が接続される．さらに，ネットワークの根ノードには，システム全体の管理を行う SVP (SuperVisory Processor) が接続される．

2.3 プロセス制御ユニット (PCU) のハードウェア構成

プロセス制御ユニット (PCU) は，ARU や ORU で実行するプロセスのスケジューリング，子プロセスの管理，動的負荷分散方式の実現，PE 内のローカル・メモリ管理，プロセス（プロセッサ）間通信の制御などを行う専用ユニットである．したがって，PCU 自体は，論理プログラムの実行と直接的な関係はなく（ただし，実行モデルには依存する），むしろ機能的には，従来の汎用計算機システムにおけるオペレーティング・システム機能を支援する専用ハードウェア・ユニットと捉えることができる．

PCU においては，①ARU や ORU が効率良く動作するように制御する，②ネットワークにおけるバスの無駄な占有時間を最小に抑えるようにプロセッサ間通信の管理を行う，の2点が重要な課題となる．時間的には，PCU での処理が ARU や ORU での処理と完全にオーバラップするくらいの性能が PCU に要求される．

並列計算機におけるプロセス制御ユニットとしては，既存の汎用マイクロプロセッサなどを使用するのが普通であるが，KPR ではそのような方式をとらず，PCU もその機能に応じて専用化（ハードウェア化）を図る方式を採用した．この理由としては，（1）ARP や ORP が専用化されたヘテロジニアス・システム構成をとっていることによるユニット間インタフェースの問題，（2）既存の汎用マイクロプロセッサを PCU として用いるには多くの不必要な機能を有していること，（3）汎用マイクロプロセッサをソフトウェアによって専用化し，PCU として用いた場合，専用化されている ARU や ORU との良好な速度バランスを獲得することが難しくなること，などが挙げられる．

PCU の専用化においては，まず，**図2**に示すように，PCU の機能を PCU-A, PCU-O, PCU-N の3つのサブユニットに分散した．PCU-A, PCU-O はそれぞれ ARP, ORP におけるプロセス制御を行い，同様

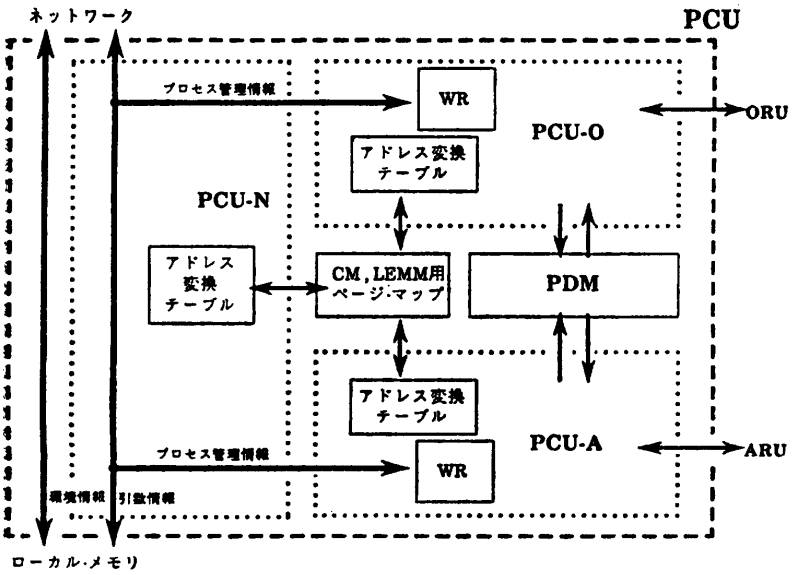


図 2 PCU のハードウェア構成
Fig. 2 Hardware organization of PCU.

のハードウェア構成とする．また，PCU-N はネットワークとのインタフェースであり，主にプロセッサ間通信を管理する．これら 3 つのユニットは，PDM (Process Description Memory) やページ・マップ，WR (Working Register) を共有し，互いに密結合された構成をとっており，サブユニット間通信におけるオーバヘッドを軽減している．PDM は，容量 8 K語（1 語=32 ビット）のメモリであり，その PCU 自身が管理しているプロセスのプロセス記述子を記憶している．

論理プログラム実行時には，各 PE 内の PCU がそれぞれ，KPR システム全体の管理機能の一部を分担・協調して遂行する．KPR システムにおけるプロセスの生成は，PCU 内では，新しいプロセス記述子を作成し，これを PDM に登録することに対応する．

PCU-N では，主に，ネットワークを介したプロセッサ間通信を管理するので，通信データ用と演算データ用にそれぞれ専用のバス（両者の間にはバイパスを設ける）やレジスタを設けるなどの工夫を施し，例えば，受信完了後に改めて受信データをレジスタ間で転送するなどのオーバヘッドが生じないような構成としている．

PCU における処理機能の単位をあまり小さくすると，管理や実行の上でのオーバヘッドが問題になる．また，逆に，処理単位を大きくしすぎると，次に掲げるような要素を満たすことができなくなる．

（a） 1つのプロセスに対して複数個連続して送られてくるメッセージ（例えば，fail イベント）を処理しなければならない．

（b） ARU や ORU で，あるプロセスを実行している時に，PCU では，そのプロセスの管理と並行して，次にその ARU や ORU で実行するプロセスの準備を行わなければならない．

（c） ARU や ORU で1つのプロセスを実行した結果，その解が複数個得られることもあり，PCU は，このような場合にも効率良く対処できなければならない．

そこで，PCU-A/O では，外部からの割り込みが発生すると，PCU 内において1つのタスクを生成し，これを単位として処理を進める「マルチタスク方式」をとっている．PCU は，プロセス間通信メッセージとして授受されるプロセス管理情報をもとに，子プロセスの管理を行う．PCU-N からのメッセージ受信通知，対の PCU-O/A からの通信要求，自らの管理する ARU/ORU からの種々のサービス要求などの割り込みによって，PCU-A/O では，（1）ページ・フォールト処理，（2）ARU/ORU の起動，（3）ARU/ORU での実行結果の登録，（4）メッセージ送出，（5）PDM 操作やメッセージの作成，などのタスク処理を行う．

PCU-A/O は，前述したタスクの頻繁な切り替えに高速に対処する必要がある．なぜならば，PR モデルに基づく論理プログラムの実行では，ARP-ORP 間の通信が頻繁に行われ，特に，メッセージ通信の局所性を優先した動的負荷分散を採用した場合，対となる ARP-ORP 間の通信量はかなり大きくなるものと予想されるためである．したがって，KPR では，ARU と ORU がローカル・メモリを共有し，PCU-A と PCU-O とが PDM を共有して密結合するという構成方式によって，このプロセッサ間通信によるオーバヘッドを極力軽減している．例えば，ARP が対の ORP へデマンドを送る場合，PCU-A 側で新しいプロセス記述子を作成し，割り込みと共に PCU-O へそのプロ

セス記述子のエントリを通知するだけで良い．

PCU の機能は，(i)通信やプロセスの効率が良くなるようにスケジューリングする，(ii)要素プロセッサにおけるプロセス・スイッチを高速に行う，(iii)動的負荷分散を実現する，などであった．これらを実現するために，PCU の各ユニットはさらに，優先度に基づくプロセス・スケジューリング，タスク・キューの高速処理，高速割り込み処理，の各機能をハードウェア化している．

例えば，論理プログラム実行におけるプロセス・スイッチは，ARU/ORU での実行と並行して PCU-A/O が次のプロセスの準備を行うことで高速に行うことができるが，場合によっては，割り込みに対する応答の速さが問題となることもある．したがって，PCU-A/O では，①高速コンテクスト・スイッチ機構を装備し，②タスクの高速ディスパッチ機能をハードウェアで実現している．

①については，(i) PCU-A/O 内部のタスクが使用する汎用レジスタは，大容量 RAM で構成することによって，論理的にフレーム化する，(ii)専用レジスタは，個々に退避領域を設ける，などの方法によって，1マシン・サイクルで各レジスタの内容を並列に退避・回復することを可能にしている．これらのレジスタはウィンドウ方式で管理されており，論理的な1つのタスクに対し，物理的な1つのレジスタ・ウィンドウを割り付けている．したがって，ウィンドウ番号を変更するだけでタスク・スイッチを行うことができる．仮想記憶方式を採用している一般の汎用計算機においてこのような方式を採用した場合，タスク数が用意されているウィンドウ数を上回ると，レジスタあふれなどの場合に対する特別な機能が必要となり，それがオーバヘッドとなる可能性がある．KPR の要素プロセッサは仮想記憶方式を採用せず，個々にバックアップ記憶を持たないため，十分なウィンドウ数を用意する方式によって，このようなオーバヘッドを回避している．

②は，タスクのディスパッチング時のオーバヘッドを抑え，また，割り込み不可状態の時間の短縮に貢献する．タスクのスケジューリングにはマルチレベルの FIFO キューを用いて行う．PCU-A/O では物理的に8レベルの優先度をサポートし，次に実行すべきタスクをハードウェアで自動的に選択するほか，キュー要素のつなぎ替えも高速に行えるような構成となっている．この構成では，特に汎用化を指向せず，KPR の実行モデルに適合したスケジューリング方式にやや柔軟性を加える程度に留めることで，ハードウェア機構を簡素化している．

なお，PCU の各サブユニットは，各々が独立した水平型マイクロプログラムによって制御される．マイクロプログラム方式を採用した理由としては，(i) PCU ではステータスのセットやリセットが比較的多く，これらの大半が並列に操作可能である，(ii) AND 並列型言語を処理・実行する場合には実行モデルも変更する必要があり，そのような場合に備えて柔軟性を確保したい，(iii) RISC などでは命令パイプラインによって処理速度を稼いでいるが，PCU では割り込み処理を高速に行う必要性から，多段の命令パイプラインを採用するのは好ましくない，(iv) KPR システムの評価やデバッグ用ツールを装備しやすい，などが挙げられる．

2.4 動的負荷分散のためのハードウェア機構

KPR のシステム規模 (PE 数) が大きくなると，1台の特別なユニット (プロセッサ) がシステム全体の状況を集中管理しながら負荷分散を行うのは現実的でない．そこで，KPR では各 PCU と NNU とが互いに局所的な情報を用いて協調することにより，システム全体としての動的負荷分散を実現する方式をとっている．

KPR が2分木状のネットワークを採用していることを利用し，各 NNU は，自分を根とする右部分木，左部分木，自身より上位の部分という3つの単位で負荷の大きさを管理する (**図3**参照)．各 NNU は隣接する NNU (または PCU) との間で互いの負荷情報を専用線を用いて受け渡しすることにより，システム全体の状態を反映した負荷分散を行う．各 PCU から知らされる PE の負荷情報は，いくつかの NNU を介して，間接的に他の PCU (PE) に伝えられる．し

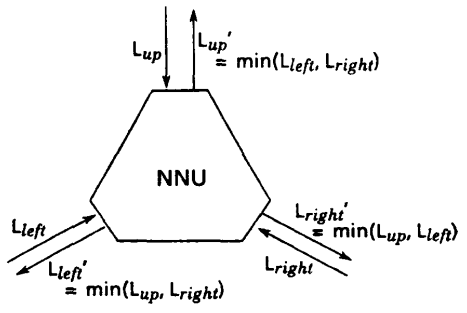


図 3 NNU の負荷量伝播機構
Fig. 3 Load propagation mechanism of NNU.

たがって，各 PE が他のすべての PE 個々の負荷を知ることはできないが，少なくとも，自分よりも負荷の小さい PE が他に存在するかどうかの判断を下すことはできる．PCU は，この負荷情報をもとに，ネットワークへデマンドを送出するか否かを決定する．

このように，2分木の構造を利用して負荷分散を行う方式は，Utah 大学の AMPS[3] でも用いられているが，KPR では，NNU のハードウェア・コスト軽減のために負荷分散機能を NNU と PCU とに分散させている点で，KPR の NNU に相当するユニットのみが負荷分散を行う AMPS の方式とは異なる．

NNU は負荷伝播線による負荷管理と連動する高速バス・スイッチ機構を備えており，PCU が発したデマンドの送出先は，複数の NNU が各 NNU 間のバスを順次切り替えていく方式により，通信の局所性を保ちながら，負荷量が最少の PE へと動的に決定される．ネットワーク・トポロジは異なるが，このような自動負荷分散機能をネットワークに付加したアーキテクチャは，東京大学の PIE[4),5)] や電子技術総合研究所の SIGMA-1[6)] でも採用されている．

3. 動的負荷分散方式

3.1 割り付け者主導方式 対 被割り付け者主導方式

動的負荷分散を実現する際に，まず考えられる方式は，アイドル状態となる PE からの割り付け要求を契機として，負荷割り付けを行う方式（被割り付け者主導方式）である．この場合には，不必要に負荷を他の PE に割り付けることはなく，したがって，無駄な通信処理を避けることができる．しかし，割り付け要求を行うためには，他の PE へのブロードキャスト，またはこれと同等の通信機能が要求され，KPR のような高並列計算機の場合には，適当ではない．また，論理プログラムの実行では，PE がいつアイドル状態になるかを予測することは不可能であり，一度アイドル状態となると，次の処理が割り付けられるまでアイドル状態のままで待たなければならない．そこで，KPR では，新プロセス生成者側の PE がその時点での負荷分散の状況を考慮して負荷割り付けの行動を起こす方式（割り付け者主導方式）を採用した．

KPR では，ネットワークに PE の負荷量を伝えるための専用線を物理的に設けており，割り付け要求のための通信は不要である．また，ネットワークの持つ専用ハードウェア機能によって，負荷量を他の PE に伝えるためにわざわざ通信を行う必要もない．しかし，割り付け者主導方式では，不必要に負荷を他の PE に割り付けてしまう危険性もあり，そのための通信がオーバヘッドとなる恐れがある．そこで，KPR では，割り付け戦略と負荷の表現法によって，これらの問題の解決を図る．

3.2 負荷割り付けのコスト

OR 並列型言語の処理方式における1つの問題は，多重の変数束縛関係など，OR 並列環境下における環境データの管理方式である．KPR では，PR モデルの自然な実現方式として，環境データをプロセスごとにコピーする方式を採用した．このため，マシン命令において多重束縛の管理を行う必要はないが，プロセス間での構造データの共有は不可能になる．

プロセスを他の PE に割り付ける際には，その環境データを単純にコピー（通信）すれば良く，環境データに対して特別な操作を施す必要はない．また，プロセスを自分自身の PE に割り付ける場合でも，それぞれの環境データは共有されないため，環境データのコピーが必要である．したがって，この点に関して，負荷を自分自身に割り付ける場合と他の PE に割り付ける場合とでは，そのコストに大きな差はない．ただし，ネットワークにおける通信の状況を考慮する必要はある．

逆に，この方式では，常に環境データのコピーが必要であり，このため，KPR では，環境データのコピーと論理プログラムの実行とを時間的にオーバラップさせることによって，環境コピーのオーバヘッドを実質的になくすことを狙っている（5章で詳述）．

3.3 負荷の表現方式

負荷分散を考える場合，一般には，各 PE の待ち行

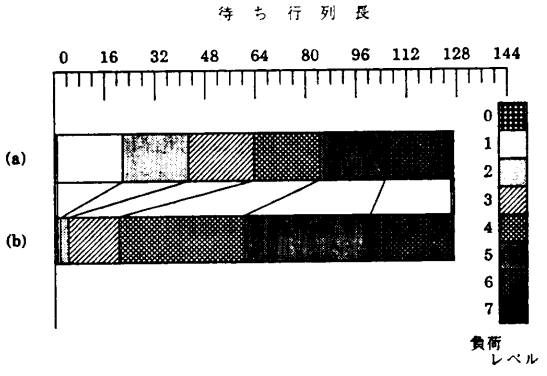


図 4 待ち行列長による負荷の表現方式
Fig. 4 Load levels represented by length of ready queue.

列長または局所メモリの使用率を「負荷」の度合いとすることが多い．KPR では，基本的に，負荷を待ち行列長によって表現する．ORP では，これはOプロセスの数であるが，ARP では，Sプロセスの数ではなく，サブプロセスの数とする．なお，実際にプロセッサが稼働状態であるか否かをネットワークに知らせることは重要であるため，実行中の（サブ）プロセスの分も待ち行列長に含めて数えるものとする．

負荷分散を行う際に，あまり細かく負荷の比較を行うと，僅少差で負荷の少ないところにデマンドが集中するという現象が生じうる．ハードウェア・コスト（負荷伝播線のビット数）の点からもできるだけ少ない負荷レベル数で有効な負荷分散を実現したい．そこで，待ち行列長そのものを用いて負荷量の比較を行うのではなく，より粗く負荷量を表現する．

プログラム実行の初期の段階では全 PE を速やかに稼働状態にする必要があり，このため，待ち行列長が0か否かは重要な判断基準となる．また，プロセスの実行と次に実行するプロセスのための準備（環境コピーなど）とを時間的にオーバラップさせるためには，待ち行列中に常にいくつかのプロセスが存在している必要がある．

以上の点を考慮し，待ち行列長と PE の負荷量を示す負荷レベルとの対応関係を次のように定めるものとする．すなわち，待ち行列長が0の時負荷レベルも0とし，待ち行列長の短いところでは負荷レベルも細かく設定する．その後，次第に1つの負荷レベルの範囲を大きくしていき，待ち行列長が長いところでは，各負荷レベルの範囲がほぼ均等となるように設定する（図4（b）参照）．

なお，負荷分散機構が必要とする負荷情報とは，他の PE に自分があとどれくらいの仕事を請け負うことができるかを提示する性質のものでもあるため，単純に待ち行列長を負荷と定義するだけでは不十分である．そこで，PE 内のメモリの未使用領域が少なくなってきた時には，待ち行列長とは無関係に，自分自身の PE の負荷レベルを最大レベルに設定する．

3.4 シミュレーションによる検討

KPR の動的負荷分散方式を検討するため，KPR のプロセス・レベルのシミュレータを用いて実験を行った．ただし，シミュレーションの対象とした応用プログラムはいずれも DBP を使用するほどの大量のデータベース節を持たないので，このシミュレーションでは，DBP の設置を仮定せず，Dプロセスは ORP で実行するものとした．

負荷分散戦略には，以下に示すA～Fの6つを選定した．

[負荷レベル数]

- 戦略A：128 レベル．
- 戦略B，C，D，E，F：8レベル．

[ARP/ORP の負荷レベル]

- 戦略A：待ち行列長．
- 戦略B：待ち行列長が0の時負荷レベルを0，最大長（127）の時をレベル7として，その間を均等に区分し，各区分にレベル1～6を割り当てた（図4（a）の方式）．
- 戦略C，D：3.3 節の表現方式を採用（図4（b）の方式）．
- 戦略E，F：設定せず．

[PE の負荷レベル]

- 戦略A，B，C，D：ARP と ORP の負荷レベルの大きいほうの値．
- 戦略E：ARP と ORP の待ち行列長の和に対して，その値が0の時負荷レベルを0，最大値の時をレベル7として，その間を均等に区分し，各区分にレベル1～6を割り当てた（図4（a）の方式）．
- 戦略F：ARP と ORP の待ち行列長の和に対して，3.3 節の表現方式を採用（図4（b）の方式）．

[デマンド割り付け戦略]

- 戦略A，B，C：デマンドを発した ARP/ORP は，デマンドの宛先候補である自分自身あるいは対の ARP/ORP の負荷レベルとネットワークの負荷レベルとを比較して，小さいほう（等しい時は，自分自身または対の ARP/ORP）に割り付ける．ただし，ORP については，自分の待ち行列長が1で，かつ，割り付けるプロセスの種類がOプロセスの時は，負荷量の比較を行わずに自分自身に割り付ける．
- 戦略D：ARP は必ず対の ORP に割り付ける．ORP については，戦略A，B，Cと同じ．
- 戦略E，F：ARP については戦略Dと同じ．ORP については，自分が構成要素である PE の負荷レベルとネットワークの負荷レベルとを比較して，小さいほう（等しい時はその PE）に割り付ける．

戦略Aは比較基準の設定のために選定したもので，最も細かく負荷を表現する場合に相当する．戦略Bと戦略Cとは，負荷レベル数を同一にして，その負荷表現の違いに対する効果を調べるためのものである．戦

略A～Cは負荷割り付けをプロセス単位で行うものであるが，これに対して，戦略D～Fでは，負荷割り付けの論理的な単位をより大きく設定している．これに伴って，戦略E，Fでは，負荷の表現方法にも変更を加えている．

6-queen プログラム，3ビット加算器の論理シミュレーション・プログラム，DCG 構文解析プログラム（トップダウン・パーサ）を応用プログラムとした場合のシミュレーション結果をそれぞれ**図5**，**図6**，**図7**に示す．負荷分散を考える時には，応用プログラムの並列度とシステム構成規模（PE 数）との兼ね合いについて考慮する必要がある．すなわち，(a)システム構成規模に対して並列度の小さい応用プログラムでは，負荷を分散させると，各 PE の負荷量はあまり大きくならず，自然に負荷量のばらつきも小さくなる．しかし，この場合は，負荷の広げすぎによる通信処理などのオーバヘッドの増加に注意しなければならない．逆に，(b)応用プログラムの並列度がシステム構成規模を上回る場合には，負荷のレベル数や表現法の影響が大きく現れる．

いずれの応用プログラムでも，戦略Aは戦略C，Dと比べてあまり大きな差は見られない．これによって，細かく負荷を表現し，比較しても，それほどの有効な結果が得られるとは限らないことが分かる．また，戦略E，Fは，他と比べてあまり良い結果を示していないが，これは，ARP と ORP の待ち行列長の和を用いて負荷を表現したため，負荷割り付け時の対のプロセッサに対する配慮を一切していないことによるものと考えられる．

負荷の表現方式については，戦略BとC，EとFを比較すると，図5，6では明らかに，それぞれC，Fのほうが良い結果を示している．一方，図7では，PE 数が8台以下の時には戦略Bが最も良い結果を示している．これは，計算の初期に各 PE に負荷が配られた後は自然に負荷が均衡したためで，DCG 構文解析プログラムの性質に負うところが大きいと判断される．しかし，その差は戦略Cと比べて大差なく，以上より，3.3 節で示した負荷表現方式が有効であることが確かめられた．

なお，論理シミュレーション・プログラムの場合，多くの解を受け取るSプロセスがいくつか存在する．図6で戦略B，Eが特に悪い結果を示しているのは，そのようなSプロセスが1台の PE に集中して割り付けられたことにより，著しい負荷の不均衡を生じたためでもある．システム側がプロセスの性質を予測できない以上，他の戦略においても，このような状況を常に回避できるとは限らない．この点からも，負荷量の少ない時に他の PE から負荷を受け取る機会を多くする負荷表現方式（3.3 節に示した）が重要である．

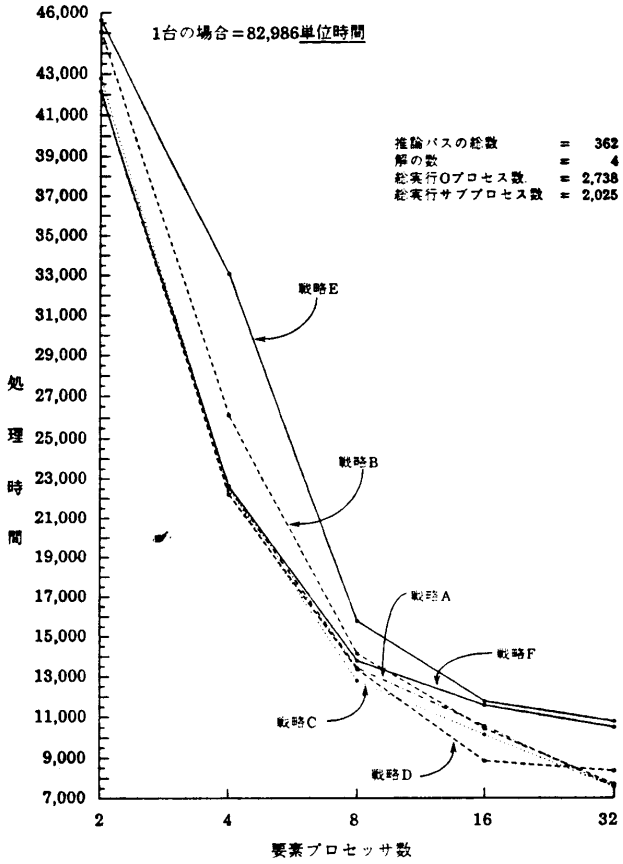


図5 シミュレーション結果 (6-queen プログラム)
Fig. 5 Simulation result (6-queen program).

シミュレーションの結果を総合的に判断すると，負荷レベル数の多い戦略Aを除けば，戦略CおよびDが安定して良い結果を示している．しかし，前述したイベント（解）に対する問題にも配慮すると，より調整の機会の多い戦略Cが適当であると判断できる．

4. プロセス・スケジューリング方式

4.1 幅優先的 対 深さ優先的スケジューリング

OR 並列型の論理プログラムの処理は，一般に，幅優先あるいは深さ優先の2通りの探索方法を用いて実現することができる．各プロセスを幅優先的にスケジューリングした場合，プログラムの持つ並列性を最大

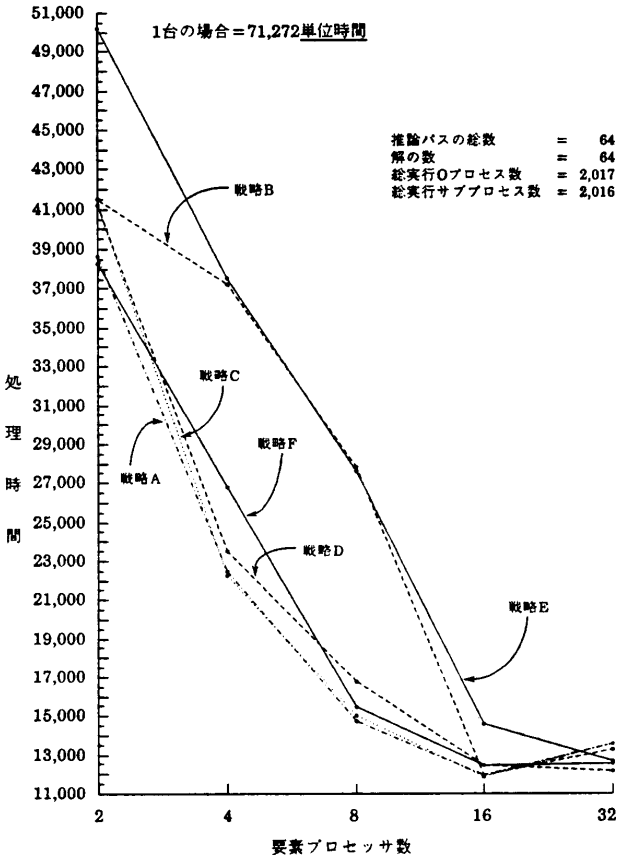


図 6 シミュレーション結果（論理シミュレーション）
Fig. 6 Simulation result (logic simulator program).

限に引き出すことが可能である反面，(i)単一解探索の場合でも全解探索に要するのとほぼ同じだけの処理時間を要する，(ii)並列性の爆発を引き起こす可能性がある，などの問題がある．これに対して，深さ優先的なスケジューリングを行うと，これらの問題点を解決できる．

並列処理効果の活用と並列性による爆発の防止との両立を考えた場合，まず，次のような方法が考えられる．すなわち，すべての PE が稼働状態となるまでは，幅優先的に負荷を分散し，それぞれの PE 内では，深さ優先でプロセスを実行する，つまり，逐次実行するという方法である．

しかし，KPR の場合，次のような点を特別に考慮する必要がある．すなわち，(1) 1台の PE 内においても ARP と ORP との並列実行が可能であり，また，SプロセスとOプロセスとは原則的に交互に呼び出される．また，(2)プロセス木の最も深いレベルのノードに対応するプロセスが新たにプロセッサに割り付けられた時に，すぐにそのプロセスを他のプロセスよりも優先して実行しようとすれば，プロセスの実行と次実行プロセスのための準備とをオーバラップして行うことが不可能となる．したがって，KPR の各プロセッサの PCU は，プロセス木の1つの推論パスに対してのみ深く探索していくのではなく，探索を深く進めていくパスを少数本に制限して，その中で幅優先的な探索を行うスケジューリング方式をとっている．

4.2 スケジューリング方式

各プロセッサが深さ優先的なスケジューリングを行っているものとすれば，新しく生成されたプロセスは，その時点で，プロセス木において最もレベルの深いノードに対応するものの1つであると考えることができる．そこで，このような性質を利用すれば，LIFO を用いて容易に深さ優先的なスケジューリングを行うことが可能である．

図 8 に，各プロセッサで用いるスケジューリング・キュー（厳密には，PCU-A/O 内におけるプロセス・スケジューリング用のキュー）の構造を示す．①自分が構成要素である PE 内で生成された処理要求と，②他の PE から送られてきた処理要求とを区別し，それぞれ別のキュー Q_2, Q_3 によって LIFO で管理する．このような区別を行うのは次のような理由による．すなわち，①については前述した性質が比較的満たされていると考えられるのに対し，②についてはネットワークによる通信の遅延などもあり，前述した性質が必ずしも満たされているとは限らないからである．特に，各 PE で処理している枝の深さに大きなばらつきがある場合には，Q_3 内の要素が，必ずしも推論木におけるレベルの深いものから順に並んでいることは保証されない．

実際にディスパッチの対象となるのはキュー Q_1 中の先頭の要素である．Q_1 中の要素数は最大2～3個に制限し，FIFO で管理する．これは，前節(2)の対応策となっている．また，述語呼び出しごとに，その回数をカウントしておき，Q_1 に要素を加える時には，Q_2 と Q_3 のそれぞれ先頭の要素で，呼び出し回数の大きいほうを選択する．これにより，自分自身のプロセッサで処理する枝の深さの調整を図る．

5. 他の方式との比較

論理型言語処理システムのマルチプロセッサ上での実現に対して，AND-OR 木の各ノードにプロセスを

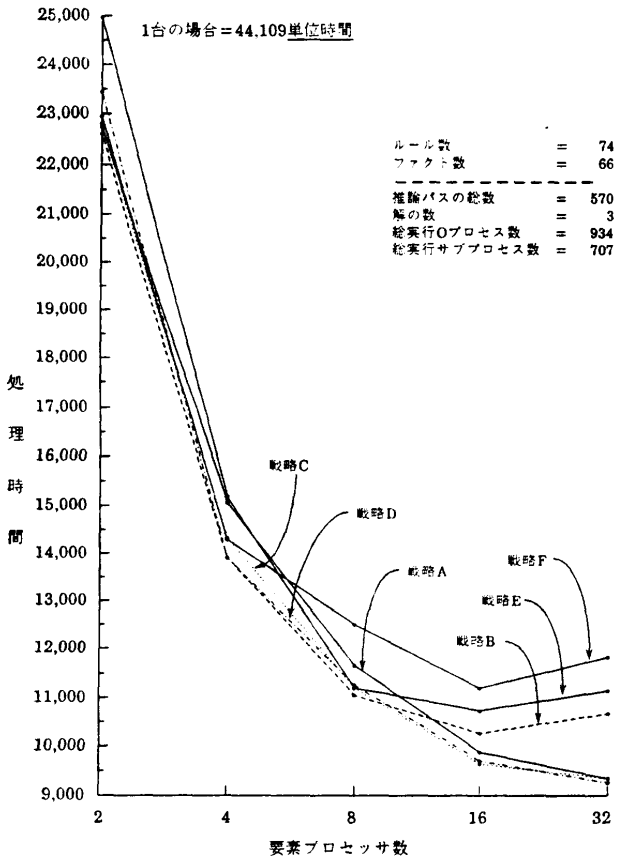


図 7 シミュレーション結果 (DCG 構文解析)
Fig. 7 Simulation result (DCG parser program).

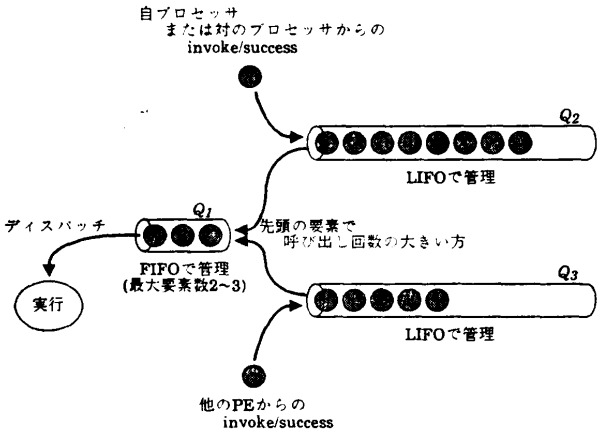


図 8 スケジューリング・キューの構造
Fig. 8 Structure of scheduling queue.

割り付けるという考え方を最初に示したのは，Conery らである．PR モデルも，基本的には彼らの示した AND/OR モデル[7] に従ったものであるが，プロセス管理方式については，AND/OR モデルでは深さ優先的なプロセス・スケジューリングを行うことがモデルに機能として含まれているが，PR モデルではスケジューリングに関する要素を含んでいない，という点で両者は異なっている．

Biswas らの制限付き OR (LOR) 方式[8] は，OR 並列処理に関しては，AND/OR モデルと本質的に同じである．彼らはモデル・レベルでのシミュレーションによって，PR モデルと同様のモデルで幅優先的スケジューリングを行った場合と，LOR 方式とを比較し，LOR 方式のほうが優れていることを確かめている．LOR 方式では，子プロセスからの解を，それが必要になるまで受け取らない．これに対して KPR で行う深さ優先的スケジューリングは，モデルの実現レベルにおける機能であり，S プロセスは，その必要性の有無にかかわらず，すべての解を受け取る．しかし，その解に対する処理（サブプロセス）は，不必要に実行されるわけではない．結局，LOR 方式も KPR で行うスケジューリング方式も，解を親子のどちら側のプロセスで蓄積するかの違いであるが，KPR のほうが別解を要求するための通信は不要であり，また，モデルの実現レベルでスケジューリングを行う点で，やや柔軟性を持っている（例えば，不必要に OR 並列実行を制限することがない）と考えられる．

以上のようなアプローチとは別に，論理型言語を OR 並列処理する場合には，PR モデルのプロセスよりもさらに処理粒度を粗く設定することができる．そのようなシステムの例として，例えば，Utah 大学の Boplog システム[9]，富士通らの株分け方式[10]，Gigalips プロジェクトの Aurora[11] などが挙げられる．本論文の以降の議論では，このようなアプローチによる処理方式を「マルチ・シーケンシャル方式（MS 方式）」と呼ぶことにする，

MS 方式では，各 PE は，論理プログラムを逐次実行し，1 台の PE が同時に 2 つ以上の仕事を実行することはない．すなわち，完全な深さ優先探索を行う．これに対して，PE 単位では，幅優先的なスケジューリングを行う．つまり，他の PE に負荷を移動させる時には，できるだけ推論木の根に近いノード（厳密には，そのノードを根とする部分木）の処理を選択する．このような方法を採用する理由としては，次のようなことが挙げられる．

（1） 負荷割り付けのコスト：Aurora のような共

有メモリ・システムでは，負荷割り付け時にメモリ領域をロックする必要がある．これは，処理が進むにつれて，推論木の形状が動的に変化するためである．各PEは，プログラム実行の各時点で推論木の葉ノードに対応する部分を処理しているが，そこから仕事をアイドル状態のプロセッサに分け与えようとすると，実行を一時的に中断しなければならない．また，株分け方式のような分散メモリ・システムでは，推論木の根に近い部分で仕事を分割するほうが，環境データの転送量が少なくて済む．

（2） 処理粒度：分け与えられた仕事の処理量が少なければ，そのプロセッサは少しの時間経過の後に，再びアイドル状態となり，負荷割り付けを要求する（被割り付け者主導方式）．一般には，推論木の葉よりも，根に近いノードから仕事を分割したほうが，そのノードを根とする部分木が大きく育つことが期待できる．

KPRでは，推論木の葉の部分で仕事（プロセス）を分割することに問題はない．環境データは述語呼び出しごとにコピーされるが，そのデータ量は必要最小限のものであり，それまでの履歴のすべてをコピーする必要はない．また，他のPEが探索した部分をトレースしておく必要もない．したがって，1回の負荷割り付けにかかるコストはMS方式よりもKPRのほうが少ない．しかし，処理粒度が細かいため，環境コピーの回数がMS方式に比べて多くなり，特に，非常に多量の構造データを扱う場合には，そのコピーに要する時間でKPRの性能が押さえられてしまう可能性もある．したがって，KPRでは，PCUとARU/ORUの処理を並行して行う機能分散方式を採用し，さらに環境コピーのためのハードウェア機構をARU内に装備している[1]．

また，MS方式では，仕事を分割すれば，そこから解が返されてくることはないが，KPRでは子プロセスから解が返されてくるため，通信回数は多くなる．ただし，Sプロセス宛に返されてくる解に対して推論木の深さの調整を図ることにより，システム全体として，MS方式よりもさらに深さ優先的な探索が可能になる．

早稲田大学の階層型挟み打ち探索法[12]もMS方式の1つと考えられる．この方式では，OR木の枝を左右両側から挟み打ちすることにより，単一解探索の場合にPE台数以上の速度向上を達成できる場合があることを確かめている．これは，解の位置がOR木の右側に片寄っている場合に可能となる．KPRでは，Oプロセスが子プロセスを生成する場合，その生成（OR木の探索）順序は特に定まっておらず，UUにおける単一化の終了したものから動的に決定される．また，各子プロセスの割り付けられたプロセッサの実行状況によって，推論木のどの枝から探索が進められるかが決定される．

6. おわりに

本論文では，KPRのプロセス管理方式について，特に，プロセスのスケジューリング方式，動的負荷分散方式について重点的に述べた．

プロセスの同期はKPRの実行モデルで既に実現されており，これを管理するPCU内においては，資源の割り当て以外には，タスク間での同期問題は存在しない．PCUの構成では汎用性を特に狙わず，逆に，実行モデルに依存して構成を単純化することにより，ハードウェア機構の簡素化を図っている．しかし，PCUのハードウェア化に伴うコストについて，特にアーキテクチャ的に他のユニットとの速度バランスがとれているかどうか十分な評価を加える必要があろう．

AND並列型言語KL1を対象としているマルチPSI[13),14)]などでは，ユーザ指定による負荷分散方式を提案している．しかし，KPRの対象言語であるKPR-Lは，OR並列型の言語であり，ユーザにプロセスを特に意識させることはない．したがって，ユーザにプログラムの並列実行下の状況把握を強いるのは無理な場合が多く，また，述語呼び出しの際の変数に対して入出力のアクセス・モードの制限はないので，述語の呼び出し方によって，プロセス木の構造は全く異なったものとなる．このような理由により，現時点では，負荷分散プラグマの使用を考えていないが，KPRの対象言語をAND並列型にまで拡張した場合には，このようなプラグマの利用を含めた負荷分散方式について考慮することが必要となろう．

謝辞 本研究の一部は，文部省・科学研究費補助金によっている．

参考文献

1) 柴山ほか：論理型言語向き並列計算機KPRのアーキテクチャ，*Proc. of the 6th Logic Programming Conf. '87*, ICOT, pp. 183-192 (1987).
2) 柴山ほか：論理型プログラミング言語向き並列

計算機 KPR の並列処理方式，並列処理シンポジウム JSPP '89 論文集，情報処理学会など，pp. 91-98 (1989).
3) Keller, R. M. et al.: A Loosely-Coupled Applicative Multi-Processing System, *Proc. of National Computer Conf.*, pp. 613-622 (1979).
4) 坂井ほか：動的負荷分散を行う相互結合網の構成，情報処理学会論文誌，Vol. 27, No. 5, pp. 518-524 (1986).
5) 山内ほか：並列推論マシンにおける負荷分散方式の評価，情報処理学会技術研究報告，Vol. 88-ARC, No. 71, pp. 125-132 (1988).
6) 平木ほか：並列計算機におけるネットワークを用いた動的負荷分散機構，電子情報通信学会論文誌，Vol. J 69-D, No. 2, pp. 180-189 (1986).
7) Conery, J. S. and Kibler, D. F.: Parallel Interpretation of Logic Programs, *Proc. of the Conf. on Functional Programming Languages and Computer Architecture*, ACM, pp. 163-170 (1981).
8) Biswas, P. et al.: A Scalable Abstract Machine Model to Support Limited-OR (LOR)/Restricted-AND Parallelism (RAP) in Logic Programs, *Proc. of the 5th Int. Conf. and Symp. on Logic Programming*, pp. 1160-1179 (1988).
9) Tinker, P. and Lindstrom, G.: A Performance-Oriented Design for OR-Parallel Logic Programming, *Proc. of the 4th Int. Conf. on Logic Programming*, pp. 601-615 (1987).
10) 増沢ほか：株分け並列推論方式とその評価，*Proc. of the Logic Programming Conf. '86*, ICOT, pp. 193-200 (1986).
11) Calderwood, A. et al.: The Aurora OR-Parallel Prolog System, *Proc. of the Int. Conf. of Fifth Generation Computer Systems*, ICOT, pp. 819-830 (1988).
12) 甲斐ほか：階層型挟み打ち探索による PROLOG OR 並列処理手法，情報処理学会論文誌，Vol. 29, No. 7, pp. 647-655 (1988).
13) 宮崎：並列論理型言語 KL 1 の実現方式と並列 OS の記述，電子情報通信学会論文誌，Vol. J 71-D, No 8, pp. 1423-1432 (1988).
14) Takeda, Y. et al.: A Load Balancing Mechanism for Large Scale Multiprocessor Systems and Its Implementation, *Proc. of the Int. Conf. of Fifth Generation Computer Systems*, ICOT, pp. 978-986 (1988).
15) 柴山ほか：論理プログラミング指向並列リダクション・マシン KPR のアーキテクチャ，電子通信学会技術研究報告，EC 85-70, pp. 43-54 (1986).

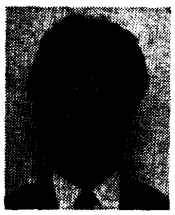
付録 プロセス間通信の高速化のための最適化戦略

（1） 単一の子しか持たず，単にデマンドの通過点でしかない冗長なプロセスを除去する．具体的には，節本体にゴール・リテラルが1つしか存在しない場合，そのOプロセスは子に当たるSプロセスを除去し，直接孫プロセス（OまたはDプロセス）を起動する．

（2） 解を返すプロセスとその解を実際に必要としているプロセスとの間に多くのプロセスが介在する場合は，中間のプロセスを飛び越して，直接，祖先のプロセスにイベントを返す．具体的には，(i)Sプロセスにおける最右端のゴールに対応して起動された子プロセスからの，そのSプロセスへの success イベント，(ii)Oプロセスが起動した子プロセスからの，そのOプロセスへの success イベント，(iii)ただ1つの子プロセスしか起動しないOプロセスへの，その子プロセスからの fail イベント，が飛び越しの対象となる．

（平成元年 6 月 19 日受付）
（平成元年 11 月 14 日採録）

平田 博章（正会員）
昭和 38 年生．昭和 62 年京都大学工学部情報工学科卒業．平成元年同大学院修士課程情報工学専攻修了．同年松下電器産業(株)に入社．現在同社情報通信研究センター情報通信関西研究所に勤務．在学中 KPR 開発プロジェクトに参画．



柴山 潔（正会員）
昭和 26 年生．昭和 49 年京都大学工学部情報工学科卒業．昭和 54 年同大学院博士課程単位修得退学．同年同大学工学部情報工学教室助手．昭和 61 年同助教授，現在に至る．工学博士．計算機システム，計算機アーキテクチャなどの教育・研究に従事．電子情報通信学会，人工知能学会，IEEE, ACM 各会員．ICOT・WG 委員．昭和 61 年度本学会論文賞受賞．

萩原　宏（正会員）

大正15年生．昭和25年京都大学工学部電気工学科卒業．NHKを経て，昭和32年京都大学工学部助教授，昭和36年同教授，現在に至る．工学博士．情報理論，パルス通信，電子計算機などの研究に従事．昭和31年度稲田賞受賞．昭和50，62年本学会論文賞受賞．昭和56～58年度本学会副会長．著書「電子計算機通信1～3」「マイクロプログラミング」など．電子情報通信学会，ACM，IEEE各会員．