TR-0776

マルチPSIにおける並列処理とその評価
ー小粒度高並列オブジェクトモデルに基づくパラダイムについてー

瀧　和男、市吉　伸行

August, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

——招待論文——

マルチPSIにおける並列処理とその評価——小粒度高並列オブジェクトモデルに基づくパラダイムについて——

正員 瀧 和男† 非会員 市吉 伸行††

Parallel Processing on the Multi-PSI and Its Evaluation —— About a Programming Paradigm Based on a Small-Grain Highly Concurrent Object Model ——

Kazuo TAKI †, *Member and* Nobuyuki ICHIYOSHI ††, *Nonmember*

あらまし マルチPSIは，64台のプロセッサをもつ分散メモリ型MIMD計算機である．マルチPSIが対象とする知識処理の問題は，計算の粒度が動的で均質さが低く，良好な負荷バランスを実現するためには，問題を多くの部分問題に分割することが重要である．これは一方で通信オーバヘッド増大の危険性をもつ．そこで本論文では，通信オーバヘッドを抑えつつ良好な負荷バランスを実現するための，上記問題領域向きプログラム方法論を提案する．この方法論では，問題を多数の通信し合うオブジェクトとして定式化し，負荷割付けの自由度を確保する．一方負荷割付けに際しては，システムの性能とオブジェクトの性質から通信オーバヘッドを見積もって，それを許容値に入れるよう処理の粒度を調節する．本プログラム方法論を最短経路問題，LSI配線問題，論理シミュレーションの3種のプログラム開発に適用し，いずれも高い効率を実現した．例えば，26万個の小粒度オブジェクトを64台のプロセッサ上に分散配置した最短経路プログラムでは，約75％の効率を得た．また12,000個のオブジェクト(＝ゲート)からなる論理シミュレーションでは，64プロセッサで48倍のスピードアップと99Kイベント/秒の高い絶対性能を実現した．これらの結果から，本プログラム方法論の有効性を確認できたと共に，分散メモリ構造の並列計算機においても，ある程度小粒度の並列処理で高い効率が実現できることを示した．

キーワード マルチPSI，MIMD型並列計算機，プログラム方法論，並列オブジェクトモデル，負荷バランス，通信オーバヘッド，性能評価

1. まえがき

マルチPSIは，分散メモリ構造のMIMD型並列計算機で，64台のプロセッサを2次元格子ネットワークで接続した構造をもつ．第5世代コンピュータプロジェクトの一環として開発され，プロジェクトの最終成果となるべき並列推論マシンPIMの，実験機にあたる計算機システムである[18],[19]．マルチPSIは，ハードウェア研究用の計算機であると同時に，並列論理型言語KL1の効率の良い処理系と，並列オペレーティングシステムPIMOSを装備し，並列ソフトウェアに関する数々の研究とプログラム開発に本格利用することを目指したシステムである[21],[30],[31]．マルチPSIのシステム構成を図1に示す．

分散メモリ構造の並列計算機は，一般にプロセッサ間の通信コストが高いと言われており，プログラム設計においてはプロセッサ間通信を抑制すること，ないしは処理の粒度を高めることに多くの努力が払われてきた．

ところが第5世代コンピュータプロジェクトが主な応用領域として目指してきた知識処理では，動的で均質さの低い計算を発生する問題が多く現れる．この性質は，SIMD型計算機の得意とするデータ並列処理，すなわち均質なデータに対する均質で同期的な処理とは対照的である．動的で均質さの低い処理の場合，プロセッサ間の負荷バランスのために，実行前にあらかじめ問題をプロセッサ数に等しく均等分割して各プロセッサに割り付けるような手法*はとることができない．

† 財団法人新世代コンピュータ技術開発機構，東京都
Institute for New Generation Computer Technology, Tokyo, 108 Japan
* 現在，(株)三菱総合研究所

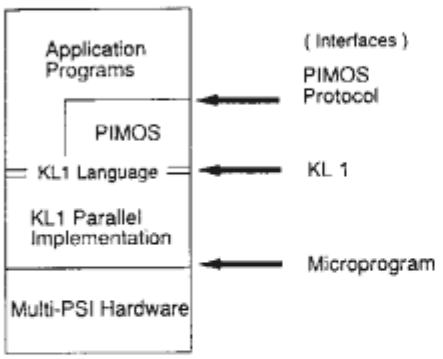


図1 マルチPSIのシステム構成
Fig. 1 System structure of the Multi-PSI.

代わりに，(a)問題をより小さな部分問題に分割して動的にプロセッサに割り付ける† か，(b)部分問題の数をプロセッサ数より十分に多く用意して，それを例えばランダムにプロセッサに割り付ける†† などして，負荷の均等化を図ることが必要となる．つまり負荷バランスのためには，問題の分割単位を小さくすることがデータ並列処理の場合に比べて余計に重要となった．

すなわち通信コストが高いと言われる分散メモリ型並列計算機上で，ある程度小粒度の並列処理を効率良く実現できるか否かが課題となったわけである．マルチPSIにおいては，この問題を解決するために，言語実装およびOS核機能まで含めた，プログラマから見える通信コストをできる限り小さくするように努力を払った．

一方で動的で均質さの低い問題は，プログラム構造の複雑さや負荷分散の面で，データ並列処理に比べてプログラミングがより難しくなる傾向をもつ．これに対処するため，適当なプログラム方法論を確立することが重要であるが，この領域における体系立った研究はほとんど行われていないのが現状である．

そこで本論文では，上記の問題領域を対象として比較的小粒度の並列処理を扱いつつ，分散メモリ型並列計算機上で高い効率を実現することができるプログラム方法論の一つとして，「小粒度高並列オブジェクトモデルに基づくパラダイム」を提案する．対象とする問題領域は，問題が互いに通信し合いながら並行動作する非常に多数のオブジェクトとして定式化でき，オブジェクトの属性の数は多くなく，オブジェクト間の通信にはある程度の局所性があり，計算はオブジェクト間で不均質に発生するという種類のものである．このような領域は，データ並列計算に含まれず，大量計算でかつ記述が可能なものの一つである．またこのパラダイムの負荷分散方式は，前記(b)の負荷割付け方法の一種であるが，システムの通信性能を考慮しつつ，通信オーバヘッドが顕著にならない範囲で小さな粒度を選択するという点に特徴をもつ．多くの問題がこの方法論に適合するか否かはいまだ明らかでないが，本論文では数種のプログラム開発例を用いて，その適用性と具体的処理性能を報告し，提案した方法論の有効性を示す．また，分散メモリ型並列計算機上である程度の小粒度並列処理により高い効率が実現できることを実証する．

以下ではまず2.で，マルチPSIハードウェア，KL1言語，言語の実現方式の概要を述べると共に，提案するプログラム方法論にとって重要な，言語と言語実装の特徴を説明する．併せてプロセッサ間通信等のシステム性能の高さについて具体的数値を示すことにより，以後の議論の準備を行う．3.では，論文の主題である「小粒度高並列オブジェクトモデルに基づくパラダイム」，すなわちプログラム方法論を提案する．4.では，本論文の第2の主題である具体例の提示を行う．本手法に基づいて作られた3種類の並列プログラムについて，プログラムの設計過程と，性能および動特性の計測評価結果を報告し，本手法の有効性を示す．これは同時に，分散メモリ型並列計算機上である程度の小粒度並列処理により高い効率が実現できることの実証でもある．最後に5.で結論をまとめる．

2. マルチPSIシステムの特徴

本章では，マルチPSI，KL1言語および言語処理系について概要を述べると共に，次章のプログラム方法論にとって重要となる言語と言語実装の特徴を説明し，併せてプロセッサ間通信等のシステム性能の高さについて具体的数値を示す．これらにより以後の議論の準備を行う．なお，詳細は文献を参照されたい[10],[11]．

2.1 マルチPSI

マルチPSI[111]は，分散メモリ構造をもつMIMD型

† 例えば文献(6)．
†† 例えば文献(5)のScattered decomposition方式．
††† マルチPSIには，6台のプロセッサを接続したマルチPSI/V1と64台を接続した改良版のマルチPSI/V2がある．本論文では，単にマルチPSIといえば後者を指すものとする．

* 例えば文献(5)のDomain decomposition方式．

表1　マルチ PSI のハードウェア諸元

プロセッサ	タグアーキテクチャ（8ビットタグ・32ビットデータ）
	水平型マイクロプログラム制御（53ビット×16 K 語 WCS）
	サイクル時間：200 ns
メモリ	プロセッサ当り最大80 MByte（16 MW）
	キャッシュ（オフチップ）4 KW
ネットワーク	最大8×8メッシュ／全系同期（200 ns）
	ウォームホールルーチング（worm-hole routing）
	チャネル：プロセッサ間全2重 8ビット並列（片方向5 MByte/s）
フロントエンド	逐次型推論マシン PSI II（最大4台接続可能）

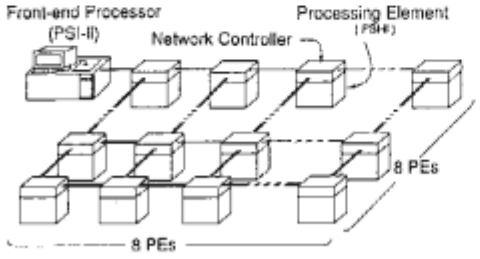


図2　マルチ PSI のハードウェア構成
Fig. 2　Hardware structure of the Multi-PSI.

並列計算機で，64台の要素プロセッサを高速の2次元格子ネットワークで接続したものである[17]．マルチ PSI の構成を図2に示す．要素プロセッサは，逐次型推論マシン PSI II[12] の CPU と同じハードウェアであり，KL1 用のマイクロプログラムを搭載する．タグアーキテクチャをとり，記号処理言語の実行に適した機能を備えている．ネットワークは，可変長メッセージの取扱いと経路制御機能をもち，指定された行先プロセッサまで自動的にメッセージを送り届ける．プロセッサとネットワークの間は FIFO バッファで接続されている．マルチ PSI の諸元を表1に示す．

ほかにハードウェアの特徴として，プロセッサ性能に比べて相対的に大容量のメモリを挙げることができる．計算の均質さの低い問題の場合，プロセッサの稼動率を確保するためには，より多くの仕事をプロセッサに割り当てる必要があり，そのためのメモリが必要となる．更に KL1 は単一代入言語で動的メモリ割当てを必要とし，このことも大容量メモリを必要とする理由である．

2.2　KL1 言語

KL1 は，並列処理用の論理型言語である．言語の基本機能はフラット GHC[20],[16] に基づき，OS 実現のための支援機能，負荷割付けやスケジューリング制御の機能，効率向上のための組込み機能などを追加している[21]．実用性を意識しながら，単一代入規則を厳格に守っている数少ない言語の一つである．

KL1 は，動的で均質さの低い大量計算問題の並列処理を効率良く実現するのに必要な，次のような性質をもっている．

- 構造の複雑な並列プログラムでも記述できること
- 通信や同期に関するバグが入りにくいこと
- 負荷バランスがプログラムとして記述できること
- 負荷割付けおよびスケジューリングの変更に対して柔軟性が高く，性能調整がしやすいこと

これらの性質は，SIMD 型計算機が得意とする均質で同期的な計算の場合に比べて，とりわけ重要性が大きい．

KL1 は，上記の性質を実現するのに次のような機能を備える．はじめの3項目は，論理的並列性あるいは並行性（concurrency）の記述に関するものであり，4番目の項目は，物理的並列性（parallelism）ないし実際のプロセッサ上での並列実行制御に関するものである．分散メモリ構成の計算機ではこれらを区別して扱わないことも多いが，KL1 では言語の中の概念として区別している．

（1）　データフローに基づく暗黙の同期　　KL1 プロセス間の通信と同期は，データフローモデルに基づきすべて暗黙のうちに行われる．プログラマはそれらを陽に記述することはなく，バグ混入の機会を著しく減少させている[1]．

（2）　小粒度並列プロセス　　KL1 プログラムにおける並列実行の単位は，節定義の中のボディゴールの1個1個である．各ゴールは，小粒度のプロセスとみ

なすことができ，KL1の実行は多数の小粒度プロセスの並列実行であるとも言える．

（3） 非決定性(indeterminacy)　KL1は，節選択に関する非決定性を有する．データフロー同期と非決定性を組み合わせて用いることにより，見込み計算(speculative computation)を許したプログラムや，動的負荷分散プログラムなどの，非決定的な動きをするプログラムの記述が可能となる．この種のプログラムは，非決定性をもたない単一代入言語では記述が困難である．

（4） 論理的並列性の記述と物理プロセッサ上の並列実行制御の記述を分離　プラグマ(pragma)は，プロセッサへのゴール割付けとゴールの実行優先度を指定するための記法である．プラグマは，プログラムの意味には影響を与えずに，実行時の並列性と実行効率を制御する．プラグマは通常，問題を解くためのアルゴリズムにかかわるプログラム記述(論理的並列性の記述)およびデバッグが完了した後に，ゴールに対して付加するものである．

（a） 負荷分散プラグマ-ゴールのプロセッサへの割付けは，@node(X)というプラグマで指定する．Xはプログラム中で計算可能である．

（b） 実行優先度制御プラグマ-ゴールの実行優先度は，@priority(Y)というプラグマで指定する．現状で4096の優先度レベルを実現している．ゴール間の厳格ではない実行順制御に使用し，並列処理効率の調整に有効である．

（5） その他の特徴　OS支援機能として，ゴール(プロセス)の集合をタスクとして一括管理・制御する「荘園」機能がある．また処理効率を高めるための組込みデータ型や組込み機能を，論理型言語の性質(GHCのレベルの)を壊さない形で取り込んでいる．

2.3 KL1言語の実装

2.3.1 言語実装の概要

分散メモリ構造のハードウェア上に，一つのKL1システムを実装しており，プロセッサごとに独立のKL1システムを載せているのではない．論理変数やプログラムコードについて大域名前空間を実現している．プロセス間の通信や同期は，たとえ別々の計算ノード間であっても暗黙のうちに行われる．小粒度の並列プロセスを効率良く実現している．また複数のちり集め方式を組み合わせて使用し，メモリの管理・利用効率を高めている．

KL1言語の実装にあたっては，前節に述べたKL1言語の特徴ある機能を可能な限り効率良く実現することを目指した．それらの機能のほとんどは，いわゆるOS核機能に関係するため，OS核機能を言語実装の中に取り込み，低いオーバヘッドで利用できるように努めた．すなわち，メモリ管理，プロセス管理とスケジューリング，通信と同期，大域名前空間，通信メッセージの組立て・送受などの機能を，言語実装の中で実現している[14),18)]．

一方PIMOSは，プログラミング環境やユーザインタフェースをはじめとする上層のOS機能を実現している．

2.3.2 暗黙の通信とその効用

プロセッサ間での暗黙の通信機能は，論理的並列性の記述と物理プロセッサへの仕事割付け記述を分離するのになくてはならないもので，次章のプログラム方法論を成立させた背景の中の最も重要な要素である．以下では暗黙の通信とその効用について簡単に述べる．

KL1プロセス間の通信と同期は，プロセス間で共有された論理変数に対するユニフィケーションとして暗黙のうちに実行される．これらはプロセッサ内でも，プロセスが別々のプロセッサに置かれた場合でも，プログラマから見ると共通になるように実装した．

プロセス(ゴール)は，負荷分散プラグマ@node(X)を付加するだけでプロセッサ間を移動する．このときプロセスが参照ポインタをもっていたならば，異なるノード間での遠隔参照が自動的に生成される．これらの遠隔参照を用いて，遠隔プロセス間での通信と同期が実現されている．

この方式により，ハードウェアの分散メモリ構造は，プログラマにとってほとんど見えないものとなっている．プログラマは，プロセス間の通信と同期を設計するのに，それらのプロセスがどこに置かれるか(同一プロセッサ内か別か)を意識する必要がなくなる．

すなわち，論理的並列性の記述の段階(コンカレントプログラミング)では，プログラマはハードウェア構造を意識しないでよく，プラグマを用いて負荷割付けやスケジューリングなどの並列実行の制御を記述する段階(パラレルプログラミング)ではじめて，プロセッサの接続構造等のハードウェア構造を意識すればよくなる．

2.4 マルチPSIにおけるPE間処理コスト

マルチPSIでは，暗黙の通信・同期や小粒度並列プロセスを効率良く実現しているが，それらの具体的な処理コストに関しての測定を行ってきた[11),16),19)]．後の議論に関係の深いものをまとめる．

[処理時間の単位] 以下の処理コストの議論では，処理時間の単位として，リストのappendプログラムにおける1リダクションの時間を用いる．以下ではこれをALI (time of Append Logical Inference) と呼ぶ．マルチPSIでは，1 ALI=7.8 μs (39マイクロ命令ステップ) である．ALIという相対表現を用いると，プロセッサ性能とは独立に処理コストの大きさの議論ができる[†]．

[通信と同期] KL1プロセス間の通信と同期は，どちらもユニフィケーションにより実現され，実行メカニズム上は区別がない．単にプログラマがどちらのつもりで使うかにより呼び方を変えるだけである．

[プロセッサ内コスト] プロセッサ内での通信・同期のコストは，1.8 ALIである．この値から，プロセッサ内におけるプロセス間通信周期が決まれば，そのオーバヘッドが計算できる．1.8 ALIの時間計算して必ず1回通信するプロセスでは，オーバヘッドは50 %となる．通信周期を粒度だとすると，小粒度のプロセスが低いオーバヘッドで使えると言える．

[プロセッサ間コスト] プロセッサ間での通信・同期のコストは，リストをメッセージストリームとして用いる場合では，プロセッサ負荷として現れる部分 (送信側，受信側の和) が23 ALIで，全コストのほとんどを占めている．ネットワークハードウェアの転送時間は隣接プロセッサ間で約1 ALI，ネットワーク上の最も遠隔のプロセッサ間でも3 ALI未満しかなく，マルチPSIのネットワーク通信性能は，ほとんどプロセッサ性能で決まっていることがわかる．

[コスト比] プロセッサ内とプロセッサ間の通信コストの比は，上記の値から約13~14倍であることがわかる．プロセッサ間通信コストがこの程度に小さいことから，プロセッサ間にまたがって比較的粒度の小さい通信し合うプロセスを配置できると言える．

[ゴール移動コスト] 次に，KL1のゴールを別プロセッサへ移動するコストを考える．これは別プロセッサへの，遠隔プロセス呼出しのコストとも言える．3引数のゴールで，そのうち1個がアトム，2個が遠隔参照となるゴールの場合，プロセッサ負荷として現れるゴール移動のコストは，27 ALIである．隣接プロセッサ間のネットワーク転送時間は1.7 ALIであった．

3. 小粒度高並列オブジェクトモデルに基づくプログラミングパラダイム

3.1 小粒度高並列オブジェクトモデル

はじめにモデルを定義する．

多数の小粒度オブジェクトが互いに通信し合いながら並行動作しつつ問題の解を求めるモデルである．オブジェクトの属性の数は多くなく，各オブジェクトの動きは非同期的である．計算は，オブジェクト間でのメッセージの流れに応じて不均質に発生する．問題解法のアルゴリズムは，センタを置かない分散アルゴリズムを基本とし，各オブジェクトは受け取ったメッセージに応じて自律的に動作する．オブジェクト間には，メッセージの交換に関して何らかの局所性があるものとする．

上記説明中，粒度とは，通信頻度の逆数と定義する．すなわち通信頻度が高いと粒度は小さい．同一プロセッサ内でのプロセス間通信頻度の逆数，または通信発生の平均周期をPE内粒度と呼ぶことにする．またPE間通信の平均周期をPE間粒度と呼ぶ．

このモデルの特徴のうち，属性ないし性質のばらつきが大きくはない多数のオブジェクトの集まりとして問題をモデル化する点は，データ並列処理 (均質なデータを扱う意味で) に近い．一方，計算の発生は非同期的で不均質である．この点はデータ並列処理 (同期的で均質な処理の意味で) と異なる．オブジェクトの属性の数が少ないものに限定したのは，その数が多いと，プログラムとして記述しきれなくなるからである．

このような形に定式化できる問題が，たくさん存在するか否かはまだ明らかでないが，少なくともデータ並列処理以外の新しい問題領域を対象としている．データ並列よりは自由度の高いモデルであることから，適用範囲は広いと期待している．

3.2 プログラム方法論の概要

本プログラム方法論は，次の4段階からなる．更に効率を向上させるために第5段階目の追加が考えられるが，これは後ほど述べる．

[第1段階：問題の定式化] 前節の並列オブジェクトモデルに基づいて問題を定式化する (し直す)．重要なのはオブジェクトの個数が多いことであり，粒度の小ささは必要条件ではない．オブジェクトの個数を確保することにより，プロセッサ数の多い並列計算機上での負荷バランスを容易にすることを目指す．

[第2段階：分散アルゴリズムの設計] 上記定式化に基づき，問題解法の分散アルゴリズムを設計する．分散

† 言語処理系の最適化の過程で，appendプログラムの性能向上の割合が他のプログラムのそれより大きくなることがある．従ってALI値を用いた相対性能の比較は，厳密な比較を必要としないときにだけ用いる．

アルゴリズムとは，計算順序の制御をどこか1箇所で行うのではなくて，オブジェクトがメッセージ交換により自律的に動き，オブジェクト間のメッセージの連鎖が解を導く方式のことである．オブジェクトが自律的に並行して動く分，並列度が高い．

[第3段階：プログラム化] オブジェクトを，通信し合う軽量プロセスとしてプログラム化する．この段階で，プログラムの論理的並列性の設計が完了する．KL1では，この状態での擬似並列実行が可能であり，問題解法の分散アルゴリズムにかかわるバグの除去は，この段階で行ってしまう．

[第4段階：仕事の割付け] 並列計算機への仕事の割付けを設計する．負荷バランスと通信オーバヘッドの削減を同時に一定水準以上の良さで実現するための，現実的な方法論を以下に提案する．PE間処理コストの測定値から，PE間粒度の最小値の制約を求め，粒度を制御するという考え方に基づいている．

3.3 仕事割付けの方法論

本方法論では，はじめは次の仮定を置く．

(1) 問題は十分に大きく，プロセッサ1台に割り付けられるオブジェクトの個数も十分に多い．

(2) 通信性能は，通信にかかわるプロセッサ処理の性能で決まり，裸のネットワーク転送性能にはよらない．

一つ目の仮定は，問題さえ大きくできれば無理はない．二つ目の仮定は，ハードウェアと言語処理系の性能により成立しないこともあるが，マルチPSIでは成り立つ．

(1)の仮定があると，例えばランダムな割付けにより，完全な負荷バランスが得られる．このとき，並列処理効率 E は次式で表される．

$E=1-(ECP/ECI)$

ECP：通信1回当りの平均CPU時間(秒)

ECI：平均通信間隔(秒)＝PE間粒度

提案する仕事割付けの方法は，先に目標とする並列処理効率 E を与えてやり，そこから必要とされるPE間粒度 ECI を求める．ここで ECP は，通信の種類とデータ長が決まればシステム性能から一意に求まる．粒度を ECI の計算値以上にすれば，処理効率も E 以上を得ることができる．本方式では，粒度を許される下限付近に設定することによって，粒の個数を最大限に確保することに努め，それらの粒をランダムにプロセッサに割り付けて，負荷バランスを実現するものである(便宜上，循環的割付けも用いる)．

現実には次の手順による．

(a) プログラムで実現されたプロセスの粒度(PE内粒度＝ICI)を予想する．これは，一つのメッセージ受信に対してプロセス内でどの程度の時間の計算を行うかの見積りである．

(b) 発生するプロセス間通信の種類と平均的データサイズを予想し，これがプロセッサ間に渡った場合の ECP の値を求める．これを計算するのに必要な言語処理系とハードウェアの性能値は，あらかじめ測定済みとする．

(c) 目標とする処理効率 E を決める．

(d) E と ECP から必要とされる ECI の値を計算する．

(e) PE内粒度 ICI を，必要とされるPE間粒度 ECI の計算値と比較する．

(f-1) $ICI \geq ECI$ ならば，プロセスをそのまま負荷割付けの単位とし，ランダムにプロセッサへ割り付ける．仮定(1)と(2)が成り立つ限り，これだけで目標の効率 E を実現できる．

(f-2) $ICI < ECI$ ならば，粒度が小さすぎる．そこでローカルに通信し合うプロセス複数個をグループとし，グループの粒度(グループ外との通信頻度の逆数) GCI が，$GCI \geq ECI$ となるような大きさのグループを作る．このグループを単位として，ランダムにプロセッサへ割り付ける．やはり仮定(1)と(2)が成り立つ限り，これで目標の効率 E を実現できる[†]．

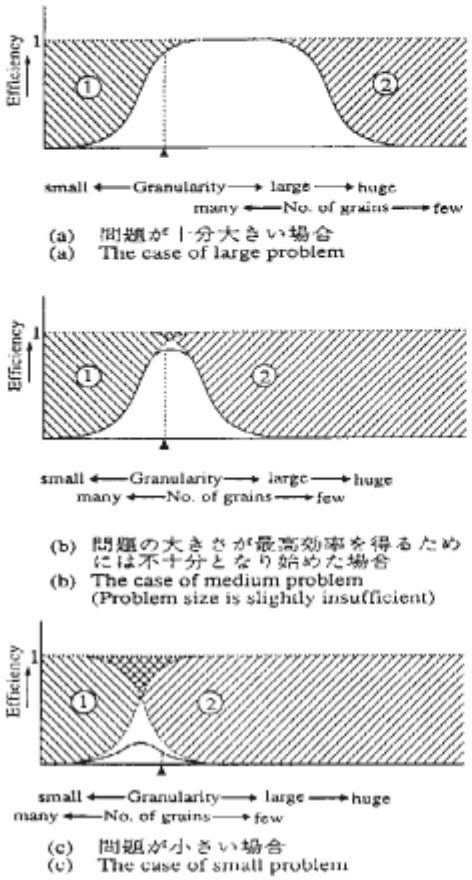
3.4 問題が小さい場合の考察

前節で仕事割付けの方法論を提案したときの強い仮定を緩め，問題の大きさが十分でなく，プロセッサ1台に割り付けられるオブジェクトの個数が負荷バランスをとるのに不十分な場合について考察する．これは前節の仮定(1)を置き換えるものである．問題の大きさが不十分なことは，現実の応用ではしばしば起きる．前節の(2)の仮定はそのまま用いる．

それでは図を用いて，問題の大きさを変えた場合の粒度(ここではPE間粒度のこと)と効率の関係を定性的に説明する．併せて3.3の仕事割付けの方法論における粒度制御の方針と図との対応関係について述べる．

図3は，問題の大きさを大，中，小と3種類に固定したそれぞれの場合の，並列処理効率とPE間粒度の関係を示す．縦軸は並列処理効率を示し，横軸はPE

† (f-1)および(f-2)の計算では，厳密には ECI の代わりに $(ECI - ECP)$ を用いるのが正しい．但し E が1に近いときはもとの計算式が近似解を得るのによい．



(a) 問題が十分大きい場合
(a) The case of large problem

(b) 問題の大きさが最高効率を得るためには不十分となり始めた場合
(b) The case of medium problem
(Problem size is slightly insufficient)

(c) 問題が小さい場合
(c) The case of small problem

図3 問題の大きさを変えたときの粒度と効率の関係
Fig. 3 Relationship between granularity and efficiency on the cases of different problem size.

間粒度を表す．効率の定義は次式のとおりであり，真の計算時間とは通信処理を含まない意味である．

$$\text{並列処理効率} = \frac{\sum_{\text{PE}} (\text{PE における真の計算時間})}{\text{実行時間} \times \text{PE 数}}$$

[問題が十分大きい場合] 図3(a)は，問題が十分大きい場合である．問題の大きさとは総計算量である．実線が，粒度を変化させたときの効率のグラフである．①の領域は，PE 間粒度を小さくしすぎたために CPU が通信処理を行う時間が相対的に増加し（通信オーバヘッドが顕著になり），効率を低下させる部分である．②の領域は，粒度を大きくしすぎたために粒の個数が不十分となり，負荷の不均衡で CPU の遊ぶ時間が発生して効率を低下させる部分である．問題が十分に大きいと，①と②の領域は離れており，中間部分には粒度を変化させても高い効率を維持できる領域が広く存在する．すなわち問題が大きいと，仕事の割付けに苦労しなくても性能を得やすい．

[問題の大きさがやや不十分な場合] 図3(b)にそれを示す．問題が小さくなると，粒度を固定した場合の粒の個数は少なくなり，負荷バランスはとりにくくなる．すなわち粒度を増加させた場合の負荷の不均衡は，3(a)の場合より早く起こり，グラフの②の領域は左に移動する．図3(b)では，①と②の領域がちょうど重なり始めるような問題の大きさを示す．これより問題が小さくなると，もはや1に近い効率を得ることはできない．

3.3 の仕事割付け方式は，図中の▽印，すなわち通信オーバヘッドが出始める付近に，粒度をもってこようとするものである．この点に粒度を設定すると，問題の大きさが図3(b)の場合以上あれば，いつでも高い効率を得ることができる．すなわち，最も広範囲の問題の大きさの変化に対して高い効率を維持することが可能となる．

[問題が小さい場合] 図3(c)は，更に問題が小さい場合で，①と②の領域が大きく重なっている．重なった部分では，通信オーバヘッドと負荷の不均衡の両方が原因で効率が低下し，総合的な効率は実線のように低いものとなる．この場合に限っては，3.3 の仕事割付け方式に従って三角印のところに粒度を設定しても，最高効率を得ることにはならない．

しかしながらこのように問題の小さい場合は，効率最高の点を見つけるには，粒度と仕事割付けを慎重に調整する必要があり，それが必要なときは三角印の点を調整の出発点として用いるのがよかろう．

このように効率最良の点を試行錯誤ででも求めたい場合というのは，大きさがほぼ等しく性質も同様の問題を繰り返し計算する必要のあるときで，かつ，プロセッサ台数の多い並列計算機を独占することができ，効率が低くても1個1個の解を少しでも早く求めたいようなときに限られる．

1台の並列計算機を複数人で使う場合，このように小さい問題では使用プロセッサ数を減らすべきで，そうすると負荷バランスは改善でき並列処理効率も上がり，システム全体の有効利用を図ることができる．

[関連する別方式について] ここで扱った仕事割付け方式では，通信のオーバヘッドが許容値を超えない範囲で，最も小さい粒度を選択するという方針をとったが，全く逆の考え方として，負荷バランスが一定値以上を

実現できるための必要最小限の粒の数を用意するべく，なるべく大きい粒度を選択するという考え方も成り立つ．この場合は，図3(a)のグラフの右肩部分に粒度を設定することになる．

ここで取り扱ったのは静的な仕事割付けであったが，動的負荷分散における問題の大きさ，部分問題の粒度，効率に関する考察と測定結果を別論文で報告している[6]．

3.5 ネットワーク性能不足の場合の考察

3.3の仮定(2)を置き換えるものとして，通信性能がネットワーク転送性能で決まるような場合を考える．このとき仮定の(1)は存続するものとする．

この場合，通信頻度を上げるとネットワークバンド幅の制約でネットワークの閉そくが発生し，プロセッサが待たされて効率が低下する．通信にかかわるプロセッサ処理は，効率上無視できるとする．このときPE間粒度の下限は，ネットワーク通信量の許容値で抑えられる．裸のネットワークバンド幅を B，複数のプロセッサがネットワークを使用するときに1台のプロセッサに許される単位時間通信量の最大値を $f(B)$ (但し $B > f(B)$) とする．f はネットワークトポロジーと通信パターンにより変わるが，概略の $f(B)$ を求めることは可能であろう．このときPE間粒度の制約は次式で表される．

$f(B) \geq EDT/ECI$

EDT：通信1回当りの平均データ転送量 (bytes)

ECI：平均通信周期 (s)－PE間粒度

この場合の仕事割付けの方法は次のように変わる．まず EDT を見積もり，ネットワーク閉そくを起こしにくいように余裕をみて $f(B)$ の値を決め，次に $f(B)$ と EDT からPE間粒度 ECI の下限を求める．以後は3.3の手順の(e)以下と同様である．

ネットワーク転送性能とプロセッサによる通信処理性能が拮抗する場合，応用プログラムによっていずれの制約が強くなるかが変わるので，前節の方法と使い分けることになる．

3.6 プロセッサ内の最適化 (第5段階)

提案した仕事割付け方式は，割付けの単位を小粒度のオブジェクト (プロセス) としているため，割付けの変更，負荷バランスの改善に関する自由度は高い．一方，1台のプロセッサが多数の小粒度並行プロセスを担当するため，プロセス切換え／スケジューリングのオーバヘッドや，プロセス実現のメモリコストを無視できない場合が出てくる．このような問題に対する改善の可能性を示す．

提案したプログラム方法論に基づくプログラムは，次のような構成をもつ．言語実行系の役割も記す．

・オブジェクトの記述 (データの記述，メッセージに対応した処理の記述)

・オブジェクト生成とオブジェクト間の通信路生成の記述

・プロセッサへのオブジェクト割付けの記述

・実行優先度指定の記述

・KL1言語実行系の役割：プロセス (オブジェクト) の管理，スケジューリング，実行

これをプログラム変換して，プロセッサ内でのプロセス切換えをなくすプログラムを生成する．基本的な考え方は，オブジェクトを，オブジェクトの動作を規定するプログラムとオブジェクトの状態を表現するデータに分離する．そして1台のプロセッサ内では，あらゆるオブジェクトのプログラム部分だけをまとめ，更にデータ部分だけをまとめて1個のプロセスで管理する．このプロセスはオブジェクトの動作を模擬するシミュレータであり，オブジェクト間で交換されるメッセージのスケジュール，メッセージに対応するオブジェクト動作の実現 (対応するオブジェクトの状態データの更新と，必要に応じメッセージの生成) を行う．変換後のプログラムはおおよそ次のような構成をもつ．

・オブジェクト動作を模擬するシミュレータの記述

- メッセージに対応する処理の記述
  オブジェクトの状態データ管理の記述
- メッセージスケジューラの記述

・シミュレータ間の通信路の記述とプロセッサに1個ずつのシミュレータの生成

・各シミュレータへのオブジェクトデータの割付け

・実行系の役割：実行のみ (オブジェクト管理とスケジューリングは上記プログラムで)

このような変換を行うことにより，プロセッサ内でのプロセス切換え／スケジューリングのオーバヘッドを削減すると共に，多数のプロセスを実現するのに使用したメモリの量を減らすことが可能となる．

このプログラム変換は，KL1プログラムからKL1プログラムへの変換であってもよいし，更に機能レベルの低い言語への変換でもかまわない．後者の場合には，KL1がサポートしている小粒度の並列実行のためのサポートや暗黙の同期機構のうち，プログラム中で必要としないものは積極的にそれらのコードを省略するような最適化が可能であろう．これにより，更なる処理性能の向上が期待できる．

4. プログラム例と計測評価

提案したプログラム方法論を実際のプログラム開発に適用した例を紹介し，性能測定と分析結果を報告する．

4.1 最短経路問題

これはシステムの評価とソフトウェアの研究・実験を目的に作られた初期のプログラムである．これを具体例として，3.に示したプログラム作成の手順を再度追ってみる．なお，問題解法の分散アルゴリズム，負荷分散方式の詳細に関しては，既に報告がある[22],[23]．

4.1.1 問題の定義

最短経路問題を定義する．

n 個の点と e 個の辺からなる有向グラフを考える．一つの辺は，2個の点と向きで与えられている．すべての辺には，非負の実数値が与えられている．これを辺のコスト，または長さと呼ぶ．$l+1$ 個の点 $v_0, v_1, \cdots, v_l$ において，すべての $i\,(=0,1,\cdots,l-1)$ に対して点 v_i から v_{i+1} への辺 e_i が存在するとき，辺の列 $e_0, e_1, \cdots, e_{l-1}$ を点 v_0 から点 v_l への経路と呼ぶ．ある経路に含まれる辺のコストの和をその経路のコストと呼ぶ．グラフのある2点間を結ぶ経路のうち，コスト最小のものを最短経路と呼ぶ．図4に，グラフと最短経路（Shortest path）の例を示す．このグラフの例では，2点間の辺のコストは，両方向ともに等しい．

このとき，与えられたグラフ上の出発点と呼ぶ特定の1点から，他のすべての点の各々への最短経路を求める問題を，1点から全点への最短経路問題と呼ぶ．

4.1.2 プログラム作成手順

[第1段階：問題の定式化]　最大限の並列性を実現するため，グラフの点1個1個をオブジェクトとしてモデル化する．辺で結ばれたオブジェクト同士は通信路をもち，最短経路探索のためのメッセージを交換し合う．この問題の場合，モデル化は第2段階の分散アルゴリズムの設計と不可分である．

[第2段階：分散アルゴリズムの設計]　1点から全点への最短経路を求める分散アルゴリズムを設計した．

点オブジェクト v_i が受け取るメッセージはコスト経路情報 $cp(\mathrm{cost}, v_i)$ である．cost は，出発点からメッセージが通過してきた経路のコスト，v_i は直前の点の情報である．点オブジェクトはその状態として，これまでに到着したコスト経路情報のうち最小コストのもの $cp(\mathrm{min\_cost}, v)$ を保持する．点オブジェクトはメッセージが到着すると，次のように振る舞う．cost ≧ min_cost の場合，何もしない．cost < min_cost の場合，新着メッセージの方が短い経路を通ってきたことになるので，(1)自らの状態を新着のコスト経路情報に書き換え，(2)隣接のオブジェクトに対してコスト経路情報 $cp(\mathrm{cost}+\mathrm{ck}, v_i)$ を生成してメッセージ送出する．ck は隣接オブジェクトへ至る辺のコストである．図4のグラフの f 点におけるメッセージ処理の様子を図5に示す．

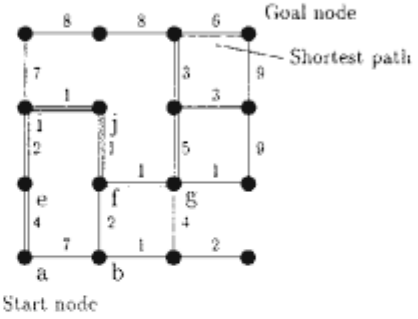


図4　グラフと最短経路の例
Fig. 4　An example of a graph and the shortest path.

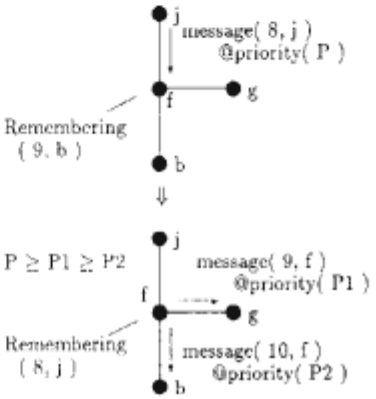


図5　f 点におけるメッセージ処理
Fig. 5　A message handling on vertex f.

任意のオブジェクト v_i の初期状態は，$cp(+\infty, v_i)$ である．実行は，出発点 v_s にメッセージ $cp(0, v_s)$ を流し込むことにより始まる．出発点の状態は $cp(0, v_s)$ に更新され，隣接オブジェクトへのコスト経路情報が生成されてメッセージ送出される．メッセージはおおむね，出発点から近隣のオブジェクトへと波面状に広がっていく．オブジェクトが構成するネットワーク上からすべてのメッセージが消滅すると，アルゴリズムは終了

する．このとき，各オブジェクトの状態として保持されるコスト経路情報が，出発点からそのオブジェクトへの最短経路情報である(詳細は文献を参照のこと[22])．各点オブジェクトは，直前の点の情報だけを保持するため，最短経路を求めるには，その点から逆向きに出発点までたどることが必要である．

このアルゴリズムにおいて，小さいコストのメッセージから先に処理されるようメッセージ処理の優先度を制御することができれば，計算の複雑さはDijkstraのアルゴリズム[2]と同じ$O(n^2)$になる．これを行わなければ，計算の複雑さは悪化する．

[第3段階：プログラム化] オブジェクトをKL1のプロセスとしてプログラム化した．オブジェクト間には通信用のストリームを張り，与えられたグラフと同一の接続トポロジーを実現する．オブジェクトが扱うメッセージとして，コスト経路情報のほかに，最短経路の問合せ，終結処理などを用意した．すべてのメッセージが消滅したことを検出する方法として，(1)ショートサーキット法[16]，(2)メッセージの取扱いを独立の荘園内に閉じ込め，荘園の終了検出機能を利用する方法[15]，の2種類で試作し，最終的な性能のやや高い後者を使用した．

計算の複雑さを低く保つために，小さいコストのメッセージから先に実行することを厳密に実現しようとすると，すべてのメッセージを優先度付き待ち行列で集中管理する必要が生じる．これは高い並列性を実現する目的と矛盾する．そこで，KL1ではゴールごとに実行優先度をきめ細かく(現状では4096レベル)指定できることを利用し，メッセージ送出を行うゴールの実行優先度をメッセージ中のコスト情報に基づき変化させる方式をとった．すなわちコストの小さいメッセージの送出を担当するゴールほど，高い優先度が与えられる．実行優先度に基づくゴールの実行順制御は，プロセッサ単位で管理実現されているため，複数プロセッサによる実行では，小さいコストのメッセージから実行することは必ずしも厳密には実現されない．これにより，厳密な実行順制御の場合に比べて，本来必要のないメッセージに関する計算が発生し，計算量は少し増加する(最大50％程度の増加が認められた[23])．その一方で並列性は確保される．

[第4段階：仕事の割付け] 仕事の割付けを決めるため，種々の時間の見積りを行う．正確な見積りは容易ではなく，また概略の目安を得ることが重要であるため，以下で扱う数値の精度は1けた前後である．

(a) プロセスの粒度 *ICI* の見積り　コスト経路情報のメッセージが届いたとき，メッセージがすぐに消滅するときの計算量は，プログラムを分析すると4リダクションであった．一方メッセージに含まれるコストが小さく，メッセージが隣接のプロセスに再送出されるときの計算量は，14リダクションであった．発生率は前者の方が高いので，それをプロセスの粒度としておく(厳密には，出現頻度で加重平均をとるべき)．小規模の最短経路問題を1プロセッサで実行することにより，1リダクションの平均の重さ(時間)が求まっており，それは5 ALIである．このことから，最短経路プログラムの *ICI*〔4リダクション〕は，20 ALIとなる．

(b) 通信当りのCPU時間 *ECP* の見積り　通信はすべて，5引数(数値定数4個と変数1個)のゴール送出として発生する(メッセージ送出するためのゴールを送出している)．そこでこの重さは，2.4の値をそのまま用い，27 ALIとしておく．

(c) 処理効率の目標：80％に決める．

(d) PE間粒度 *ECI* の計算：

$0.8=1-(27\text{ALI}/ECI)$，$ECI=135\text{ALI}$

(e) *ICI* と *ECI* の比較　プロセス粒度 *ICI* (20 ALI)は必要とされるPE間粒度 *ECI* (135 ALI)より小さい．従って隣接プロセスをグループ化し，PE間粒度を大きくする必要がある．

(f-2) グラフの点をグループ化するとき，グループの内と外の間の通信量は，グループの境界を横切るグラフの辺の数に比例すると考えられる．

例題では，2次元格子状のグラフを用いる．格子点がグラフの点である．通信比率を(グループ内外の通信量)/(グループ内の点の数)とする．このときグラフから正方形の領域を切り出してグループにすると，通信比率は，グループ内の点の数の平方根に逆比例することが，簡単な計算によりわかる．すなわち，正方形の領域をなす隣接プロセスをグループ化することにより，グループ内のプロセス数の平方根に比例してグループの粒度 *GCI* を増大させることができる．

$GCI \geq ECI$ (=135 ALI)とするために，$ICI=20$ ALIのプロセスを7の2乗個集めて，$GCI=20\times7=140$ を得ることができる．すなわち，隣接プロセスを49個以上集めたグループを単位として，プロセッサへランダム割付けすることにより，問題サイズが十分であるときには，目標とする80％の効率が得られることになる．

4.1.3 性能測定と考察

点の数 16,384 からなる 2 次元格子グラフ（1 辺 128 点からなる正方形）を測定に用いた．各辺のコストは，乱数を用いて 0 から 99 の範囲の整数を生成して与えた．プログラムは 64 プロセッサのマルチ PSI 上で実行した．

（1） プロセッサ稼動率と通信オーバヘッド

仕事割付け単位のグループ 1 個に含まれる点の数を 3 種類選び，64 プロセッサで実行した場合のプロセッサ稼動率と通信オーバヘッドを測定した．具体的には，2 次元格子グラフを多数の小さな正方形領域に等分し，それぞれの領域に含まれる点をグループとして，グループ 1 個 1 個を 64 台のプロセッサに循環的に割り付けた．図 6 にプロセッサ稼動率と通信オーバヘッドのグラフを示す．縦軸は稼動率を，横軸は経過時間を示す．棒グラフ 1 本 1 本は，0.25 秒ごとの全プロセッサの平均稼動率および平均の通信オーバヘッドを示す．通信は，送信と受信に分けて表示されている．（a）はグループ内の点の数が 64 個，（b）は 16 個，（c）は 4 個の場合である．計測と表示は，マルチ PSI の OS である PIMOS に組み込まれたツールを用いた[9]．

各グラフとも，中央で稼動率が高く両端で低いのは，実行の開始，終了のタイミングが 0.25 秒刻みの途中にきていること，実行の開始直後および終了前はメッセージ数が少なく計算量が少ないことによる．前節の計算から，グループ内の点の数を 49 より少し多い 64 個に設定することにより，通信のオーバヘッドは 20 % より幾分小さい値となることが期待された．図 6（a）では，compute，send，receive の和に対する send，receive の和の比率，すなわち通信オーバヘッドは，約 20 % で期待した値に近いものとなった．

グループ内の点の数を 16 個，4 個と少なくするに従い，問題サイズが一定であるためにグループの個数は次第に増加し，負荷バランスは次第に良くなる．これは平均の idle 時間の減少という形でグラフに現れている．一方，通信オーバヘッドは次第に増加することがグラフから読み取れる．並列処理の効率は，compute の部分の % 値を平均して得られることから，グループ内の点の数 16 個のときが最高効率を達成したことがわかる．

このような特性を示すのは，3.4 の説明および図 3 に従えば，問題が小さい場合に相当する．すなわち PE 間粒度を注意深く調整しないと効率最高の点を見出すことは難しく，その点においても得られる効率は高くはない．

（2） スピードアップ

次に使用プロセッサ数を変化させて実行時間を測定し，プロセッサ 1 台の実行時間に対する比の逆数，すなわちスピードアップを求めた．図 7 は，グループ内の点の数が 1，4，16，64，256 の場合のスピードアップのグラフである．グループ内の点の数 16 のときに最高のスピードアップを示し，点の数が増えても減ってもスピードアップは悪化している．このことは，先の効率に関する考察とも一致している．

プロセッサ数の少ない場合（グラフの左端）では，グループ内の点の数が多くても（グループの個数が少なくても）負荷バランスは良好となるため，通信オーバヘッドの最も小さいグループ内点数 256 の場合が，最も高

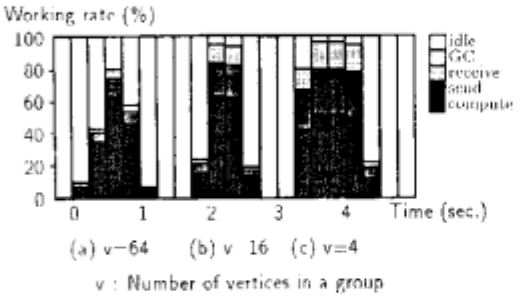


図 6　最短経路問題（16,384 点）におけるプロセッサ稼動率
Fig. 6　Processor working rate on the shortest path problem (16,384 vertices).

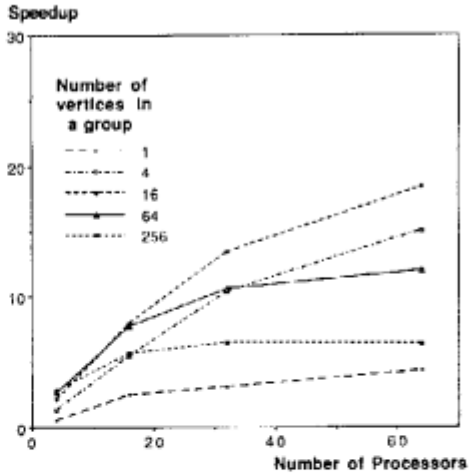


図 7　最短経路問題（16,384 点）におけるスピードアップ
Fig. 7　Speedup on the shortest path problem (16,384 vertices).

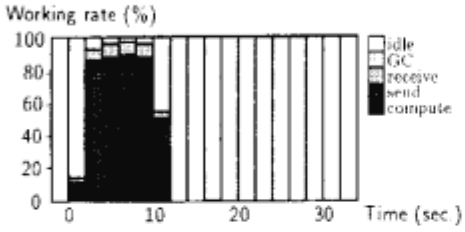


図8 規模の大きい最短経路問題(262,144点)におけるプロセッサ稼動率
Fig. 8 Processor working rate on the large shortest path problem (262,144 vertices).

い効率を示している。

(3) 問題を大きくした場合

前節で最高でも20倍弱のスピードアップしか得られていないのは，問題が小さいためであった。ここではグラフの点の数を16倍の262,144にして，プロセッサ稼動率と通信オーバヘッドを計測した。グループ内の点の数は64，プロセッサ数も64である。

結果を図8に示す。通信オーバヘッドは，図6(a)の場合とほぼ同じで，idleの割合が十分減少し，グラフの中央部分で75%を超える効率を実現することができた。これ以上の大きさの問題では，PE間粒度を同一にしておくといつも同程度の高効率が実現できる。

4.2 LSI配線問題

二つ目の適用例はLSI配線問題である。マルチPSI上の実用的な応用問題として開発を行い，アルゴリズム，プログラム設計，性能評価に関して既に報告している[13][14]。ここでは，本プログラム方法論の適用という面から説明を加え，新規の性能測定結果を報告する。

4.2.1 問題の説明

LSI配線設計の中でも，図9に例示するようなブロック間配線を対象とした。縦線用，横線用の二つの配線層を用い，あらかじめ定められた配線格子上で配線を行う。指定の端子間(図中では同番号の端子同士)をなるべく短い距離ですべて接続する配線経路を求めるのが問題である。

4.2.2 プログラム作成手順

[第1段階：問題の定式化] 小粒度高並列オブジェクトモデルとの親和性，計算効率の良さ，結線率の高さなどを考慮し，配線の基本アルゴリズムとして，線分探索法の一種である予測線分探索法[7]を使用することにした。この基本アルゴリズムでは，線分を配線の単位としている。そして接続目標の端子との近さを評価値として，評価値が高くなる方向へ配線を決めていく。

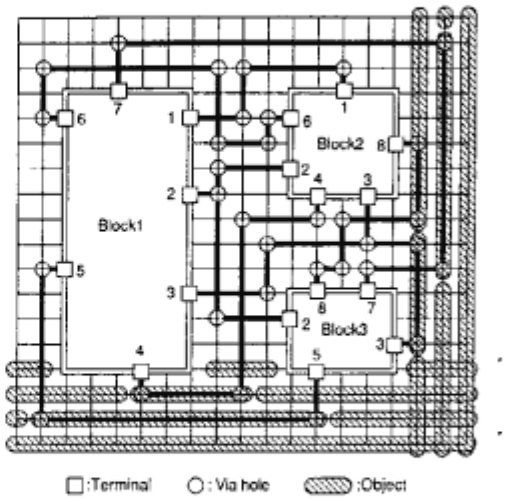


図9 LSI配線問題と並列オブジェクトに基づくモデル化
Fig. 9 LSI routing problem and its modeling based on concurrent objects.

そこで線分が単位であることに着目し，図9に示すように配線格子上のあらゆる既配線，未配線の線分をオブジェクトとしてモデル化した。図では一部のオブジェクトのみ，斜線で強調してある。交差するオブジェクトは，メッセージの通信路をもち，互いにメッセージ交換しながら配線経路を決めていく。

[第2段階：分散アルゴリズムの設計] 予測線分探索法の逐次アルゴリズムを分散アルゴリズムに設計し直した。

おおそのイメージを述べる。最初は，配線禁止領域のブロック，接続すべき端子，配線に使える未配線の線分オブジェクトが与えられる。次に，配線開始端子に隣接する線分オブジェクトに，接続目標端子を引数とした配線要求メッセージを与える。オブジェクトは直行するオブジェクトに評価値計算要求メッセージを出し，それらからの応答を待つ。評価値がすべて戻ってくると，最良の評価値を返したオブジェクトに接続するように配線線分を決定し，その線分オブジェクトの状態を既配線とする。そして接続されたオブジェクトに対して，再び配線要求メッセージを送る。これを繰り返して，目標端子までの配線を行う。配線が進むにつれて，1本の未配線オブジェクトは既配線と未配線の短い線分オブジェクトに分割されていく。配線できなかったときは，配線の引きはがし，バックトラックなどもメッセージ交換で行う。

本方式では，2種類の並列性を実現した．一つは評価値の計算を複数のオブジェクトで並行して行うこと，もう一つは，異なる端子対の間での配線を並行して進めることである．メッセージに対するオブジェクト内での処理は，原則として不可分のものとして行われ，複数メッセージがオブジェクトに到着すると，それらはオブジェクトの入り口で順序化される．

[第3段階：プログラム化] オブジェクトをKL1のプロセスとしてプログラム化することは，最短経路問題のときと同様である．メッセージの種類とそれに対する処理内容は，最短経路問題に比べはるかに多様で複雑である．

線分オブジェクトに対応するラインプロセスは，実行時に動的に分割・併合され，それらへの通信路の適切な張替えが必要である．これを実現するために，縦または横の配線格子1本に対して一つずつマスターラインプロセスと呼ぶものを配置し，同一配線格子上のラインプロセスを管理させると共に，それらのラインプロセスへのメッセージを中継させている．

直交するラインプロセス間の通信路をすべて独立に生成すると，プロセッサ間を渡る通信路の数があまりに増加し，それらを管理するメモリ領域の確保が難しくなる．これを減少させるように通信路の張り方を工夫している．

[第4段階：仕事の割付け]

（a） プロセスの粒度 *ICI* の見積り　メッセージ1個に対するマスターラインプロセスの処理は，最低で3リダクション，15 ALI，ラインプロセスの処理は，最もよく現れる軽い処理のメッセージについて，6リダクション，30 ALIである．

（b） 通信当りのCPU時間 *ECP* は27 ALIとする．

（c） この問題の場合，粒度を調節するためのオブジェクトのグループ化について，あまり良い方法が見つからなかったので，次のような妥協を行った．

まず，マスターラインプロセスの処理15 ALIを単独でプロセッサ割付け単位として良いかを検討した．この値は *ECP* =27 ALIに比べ小さすぎ，27/(27+15)=0.64つまり64％の通信オーバヘッドが出る．これは許容できないので，簡単な方法として，マスターラインプロセスとその直接の管理下にあるラインプロセスをグループ化することを考える．マスターラインが受けたメッセージは，通常はそのままラインに転送され処理される．グループが受け取ったメッセージに対する処理の高さは，15+30=45 ALIである．これを割付け単位とすると，通信オーバヘッドは，27/(27+45)=0.38すなわち38％となる．この値はまだ大きいが，通信の局所性を高める適当なグループ化方法が他に見当たらなかったため，この値を許容値として認めてしまうことにする．

すなわち，1個のマスタラインプロセスとその管理下にあるすべてのラインプロセスをまとめてグループとし，マスタラインの座標の順に各プロセッサへ循環的に割り付けることにした．

4.2.3 性能測定と考察

格子の大きさが524×424，ネット数1088のテストデータについて配線を行った場合の，プロセッサ稼動率と通信オーバヘッドを測定した．このデータは，8分の1の大きさの実LSIデータをコピーして作ったテスト用のものである．オブジェクトの数は，配線完了時で1万前後となる．プロセッサ台数を変えた場合のスピードアップについては，別に報告している[6]．

このデータを64プロセッサを用いて配線した場合の実行時間は36秒，配線率は99％～100％（並列処理で複数の配線間の処理タイミングが毎回微妙に変わり，配線率が変化する），1プロセッサの実行と比べたスピードアップは24倍であった．

図10に，プロセッサ稼動率と通信オーバヘッドのグラフを示す．実行開始直後に稼動率が高くすぐ低下するのは，その時期に短い配線（正確には折れ曲りの少ない配線）のほとんどを処理し終わるためである．実行終了前に稼動率が低下しているのは，非常に長い配線およびバックトラックを伴う配線のわずかのものが，最後まで完了せずに残るためと考えられる．

通信オーバヘッドは，予想していた38％よりは低く，約20％であった．これはメッセージ1個に対する処理の重さの平均が，前節での見積りよりも大きかったためと考えられる．このようにPE間粒度を正しく見積もることは難しい場合も多いが，粒度の下限が正しく見積もられていれば，通信オーバヘッドの上限も間違いなく抑えられることになる．

並列処理効率は，グラフから見る限り平均約50％程度あるように見られる．一方スピードアップ値24から計算される効率は，24/64=0.38すなわち38％である．両者の違いは，並列処理するとプロセッサ1台の実行に比べ複数の配線処理が同時進行する割合が増し，このため配線同士の干渉，領域の取り合いなどで計算量自体が増加しているためと考えられる．すなわち計算量増加分だけスピードアップは悪化する．

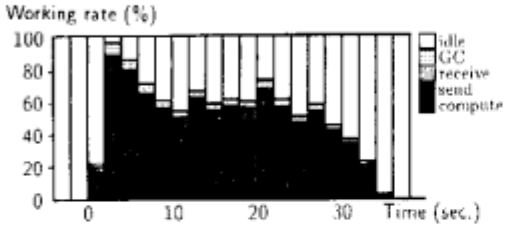


図10 LSI配線問題(524×424格子, 1,088ネット)における プロセッサ稼動率
Fig. 10 Processor working rate on the LSI routing problem (524×424 grids, 1,088 nets).

更に2倍程度大きい問題について，プロセッサ稼動率と通信オーバヘッドを測定した．この場合，稼動率は約10％上昇し，通信オーバヘッドは変わらなかった．

4.3 論理シミュレーション

三つ目の適用例は論理シミュレーションである．これもマルチPSI上の実用的な応用問題として開発を行い，アルゴリズムと性能評価に関して既に報告している[8],[9]．ここでは，3.6に述べたプロセッサ内の最適化の面から説明を加える．

4.3.1 問題概要

ゲートレベルで記述された論理回路の動作をシミュレートする．順序回路，同期回路だけでなく，非同期回路も対象とする．信号値はHi，Lo，X(不定)の3値モデルを用い，各ゲートの遅延時間に単位時間の整数倍を割り当てるノンユニット遅延モデルとする．これを並列イベントシミュレーションの問題として扱う．

4.3.2 第4段階までのプログラム化

[問題の定式化と分散アルゴリズムの設計] イベントシミュレーションでは，各ゲートがオブジェクトとしてモデル化され，オブジェクトがメッセージを交換しながらシミュレーションを進める．メッセージは，信号変化に対応するイベント情報とその生起時刻(タイムスタンプ)をもつ．

並列イベントシミュレーションで最も重要となる時刻管理機構として，高い並列性を実現可能なバーチャルタイムに基づく方法を用いた．時刻管理は各ゲートで分散して行われる．メッセージはタイムスタンプ値の小さい順(正しい順)にゲートに到着するとの仮定のもとにすべてのシミュレーション処理を進め，各ゲートはメッセージと状態の履歴を保存する．メッセージ順序の不正を発見すると，履歴を巻き戻し(ロールバック)，処理のやり直しを行う．

[プログラム化] ゲートオブジェクトをKL1のプロセスとして実現し，プロセス間にメッセージ通信路を張る．ロールバックを削減するため，プロセッサ内では小さいタイムスタンプをもつメッセージから順に処理されるようにスケジューラを置く．不要な履歴を解放するため，大域的な時刻合せを低い頻度で行う．

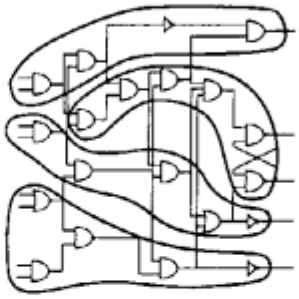


図11 縦割り指向の回路分割戦略
Fig. 11 "Cascading-oriented" circuit partitioning strategy.

[仕事の割付け] ゲートオブジェクトを図11に示す例のようにグループ化し，グループ単位にプロセッサへランダムに割り付ける．このグループ化は，縦割り指向戦略と呼んでおり，連結したゲートをなるべく同一プロセッサに置いてPE間粒度を高めると共に，複数ファンアウト部分で並列性を抽出しようとするものである．粒度の調整は，グループ内の連結したゲートの数を変えて行う．具体例は性能測定の項で述べる．

ここまでの段階で，いったん，動作するプログラムを作成した．

4.3.3 第5段階の最適化

すべてのゲートをプロセスで実現した場合の頻繁なプロセス切換えとメモリ消費とを削減するのが第5段階の効率化である．3.6で述べたプログラム変換による方法はまだ準備できていないため，手作業で3.6の変換内容に従って，KL1プログラムからKL1プログラムへの書換えを行った．

その結果，各プロセッサに1個ずつ，オブジェクトの動作をシミュレートすると共にオブジェクトの状態をデータとして管理するプロセスを設置することになった．意味上はオブジェクトがメッセージを交換し合うが，実際は上記プロセスの間でメッセージが往来し，また同プロセスはメッセージのタイムスタンプに応じてメッセージのスケジューリングを行う．各オブジェクトにおける履歴は，オブジェクトの状態データの一部として管理し，大域的な時刻合せと履歴解放も同プ

ロセスが行う．

4.3.4 性能について

ゲート数 11,965 個の論理回路に対して，マルチ PSI 64 プロセッサによる実行で，1 プロセッサに比較して 48 倍のスピードアップ，99K イベント/秒のシミュレーション性能を得た．ゲート 1 個のシミュレーション処理の粒度は 55 ALI で比較的大きく，各ゲートの平均 fan-in，fan-out はそれぞれ 1.6 程度であった．通信当りの CPU 時間を 27 ALI とすると，ゲート 4 個ずつでグループを構成するときの通信オーバヘッドが約 20 % と見積もられた．バーチャルタイムの特性として，メッセージの流れる向きになるべく直列に長く連なったゲートをグループ化する方が，ロールバック頻度を抑えるのに有効であるため，グループ 1 個当りのゲート数は，上記の実行例の場合には，12～32 個となるように自動生成した．

またマルチ PSI 上に他の 2 種類の代表的時刻管理方式でシミュレーションプログラムを作成し性能比較を行った結果，本方式が群を抜いて高い性能であることが確認された[8],[9]．

第 5 段階の効率化を適用することによる性能の向上は，プロセッサ 1 台当りのシミュレーション性能で比較して 1.5 倍程度であった．この程度の性能差であるのは，マルチ PSI の KL1 処理系が，もともと多数の小粒度プロセスの並行処理を効率良く扱えるためであるとも言える．

4.4 汎用計算機との性能比較

厳密な議論は難しいが，同様の問題を汎用計算機で処理する場合との性能比較・考察を試みる．

LSI 配線では，ほとんど同一のアルゴリズムによる逐次プログラムを汎用計算機で実行した場合との性能比較を行った[4]．マルチ PSI 64 プロセッサによる実行時間は，IBM 3090/400（約 15 MIPS）上の FORTRAN プログラムに比べて同等ないし少し短いという結果を得た．また小規模配線問題をマルチ PSI の 1 プロセッサで実行すると 111 秒，IBM マシンで実行すると 7.45 秒であり，15 倍の開きが出た．マルチ PSI はマイクロ命令の実行周期が 200 μs であり，これを CISC 命令の性能に換算して 2～3 MIPS と仮定すると，IBM マシンの 15 MIPS との性能比は 7.5～5 倍となるべきところであるが，実測では 15 倍の開きがあった．すなわち性能比で 2～3 倍分の損失が存在しており，これが並列オブジェクトモデルに基づくプログラミングの損失と KL1 の実装オーバヘッドを合わせたものであると考えられる．但しこのプログラムでは，KL1 が得意とする（効率よく実現している）通信・同期機能を多用するため，後者のオーバヘッドは小さいと期待される．

論理シミュレーションの最大性能は，マルチ PSI 64 プロセッサで 99 K イベント/秒であったが，次にこの値について考察する．汎用計算機上の論理シミュレーションの性能について正確なデータは得ていないが，近年のワークステーション上の高性能なシミュレータの性能として，10 MIPS の RISC プロセッサの場合に 40 K イベント/秒と仮定する．マルチ PSI の 1 プロセッサでバーチャルタイム法を用いると，約 2 K イベント/秒の性能であるが，逐次実行に適したタイムホイール法では最大で 5 K イベント/秒程度が得られている．RISC 命令換算のマルチ PSI の性能を 5 MIPS とする．シミュレーション性能を MIPS 値で正規化すると，マルチ PSI は 1 K イベント/秒・MIPS，ワークステーションは 4 K イベント/秒・MIPS となり，マルチ PSI の方が約 4 分の 1 である．この場合，両者とも同一モデルのプログラムを仮定しているので，この性能上の損失は，KL1 の実装オーバヘッドと考えることができる．すなわち通信や同期の必要ない単一プロセッサ上の処理では，KL1 が得意とする機能をほとんど使わない．つまり KL1 で実現している単一代入，暗黙の通信・同期，小粒度並列プロセス等のサポートメカニズムが，この程度のオーバヘッドとして現れると考えられる．このオーバヘッドは，3.6 で述べた第 5 段階の最適化によって，KL1 から KL1 への変換ではなく手続き型言語への変換をうまく行うことにより，大きく削減できる可能性をもつ．

5. むすび

マルチ PSI は，分散メモリ構造の MIMD 型並列計算機で，64 台のプロセッサを 2 次元格子ネットワークで接続した構造をもつ．小粒度プロセスの取扱いとプロセッサ間に渡る暗黙の通信・同期を効率よく実現した KL1 言語処理系を実装している．

マルチ PSI が対象とする知識処理の問題は，計算の性質が動的で均質さが低いという特徴をもつ．このような問題の並列実行では，良好な負荷バランスを実現するために，問題をプロセッサ数より十分多くの部分問題に分割する必要が生じるが，このことは分散メモリ構造の並列計算機にとっては，プロセッサ間通信オーバヘッドの増加の危険性を招く．

本論文では，上記の問題領域を対象として，分散メモリ構造の並列計算機上で，通信オーバヘッドを抑えつつ良好な負荷バランスを実現するための「小粒度高並列オブジェクトモデルに基づくプログラム方法論」を提案した．この方法論では，問題を十分に多数の通信し合うオブジェクトとして定式化することにより負荷割付けの自由度を確保する．一方通信オーバヘッドを許容値以下に抑えるため，通信に関して局所性をもつオブジェクト同士をグループ化して負荷割付け単位とし，実効的な粒度の大きさを実現する．このとき，オブジェクトの特性とシステムの通信性能を用いて通信オーバヘッドの概略値を見積もり，これが許容値に入るように処理の粒度を調節するという方式である．

また問題の大きさと効率に関する考察を行い，上記の粒度設定方針を用いることにより，問題の大きさの変化に対して最も広い範囲で高い効率が得られることを示した．また，プロセッサ内のオブジェクト切換えオーバヘッドを削減するためのプログラム変換についても述べた．

本プログラム方法論を適用した具体例として，人工的な最短経路問題，LSI配線問題，論理シミュレーションの3種のプログラムを示し，その計測評価結果から高い効率を実現できたことを報告した．例えば，26万個の小粒度オブジェクトを64台のプロセッサ上にグループ化して分散配置した最短経路問題では，約75％の効率を実現した．また12,000個のオブジェクト(＝ゲート)からなる論理シミュレーションでは，64台のプロセッサを用いて1台に比べ48倍のスピードアップとなり，絶対性能も99Kイベント/秒で，ソフトウェアによるシミュレーション性能として良好な結果を得た．これらの結果から，本プログラム方法論の有効性を確認することができた．また分散メモリ構造の並列計算機上で，ある程度小粒度の並列処理を行っても，粒度制御を適切に行うことにより高い効率が実現できることを示した．

一方，マルチPSI上で本方式を適用したプログラムと，汎用計算機上で手続き型言語で書いたプログラムを比較した場合，ハードウェア性能が等しいときには，小粒度オブジェクトの実現コストとKL1言語の実装オーバヘッドにより，マルチPSIの方が2分の1から3分の1の性能となることが，配線問題で確認された．この損失は，高い並列性を引き出すための代償とも言える．この損失の割合は，KL1が得意とする通信・同期の処理が減少する場合(例えば論理シミュレーションプログラムの場合)ほど大きくなる．

本プログラム方法論の問題点についてはまだ十分な検討を行っていないが，一つには高い効率を得るために問題の大きさをある程度大きくする必要があり，そのため大きなメモリを必要とすることが検討課題と考えている．これは並列計算機のハードウェア設計に関係するため，定量的な検討を加える予定である．

謝辞 本研究にあたり，プログラム開発，性能計測，データ提供に協力されたICOT第1，第2，第7研究室の研究員諸氏，ならびに関連メーカーの研究員諸氏に感謝する．特に大西論氏に感謝する．また研究の機会を与えて下さり，常に励まし続けて下さったICOT研究部長の内田俊一氏，研究所長の渕一博氏に感謝する．

文　献

(1) Chikayama T.: "Operating System PIMOS and Kernel Language KL1", Proc. Int. Conf. on Fifth Generation Computer Systems, FGCS'92 (June 1992).
(2) Dijkstra W. E.: "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik 1, pp. 269-271 (1959).
(3) 伊達　博，大里能久，滝　和男："並列オブジェクトモデルに基づくLSI配線プログラム"，情処学論，33, 3, pp. 378-386 (1992-3).
(4) Date H., Matsumoto Y., Hoshi M., Kato H., Kimura K. and Taki K.: "LSI-CAD Programs on Parallel Inference Machine", Proc. FGCS'92 (July 1992).
(5) Fox G., Johnson M. A., Lyzenga G. A., Otto S. W., Salmon J. K. and Walker D. W.: "Solving Problems On Concurrent Processors", 1, Prentice Hall (1988).
(6) Furuichi M., Taki K., and Ichiyoshi N.: "A multilevel load balancing scheme for or-parallel exhaustive search programs on the Multi-PSI", Proc. 2nd ACM SIGPLAN Sympo. PPOPP'90, pp. 50-59 (1990).
(7) 北沢仁志："高配線率線分探索の一手法"，情処学論，26, 11, pp. 1366-1375 (1985-11).
(8) 松本幸則，滝　和男："バーチャルタイムによる並列論理シミュレーション"，情処学論，33, 3, pp. 387-396 (1992-3).
(9) Matsumoto Y. and Taki K.: "Parallel logic Simulator based on Time Warp and its Evaluation", Proc. FGCS'92 (June 1992).
(10) Nakajima K., Inamura Y., Ichiyoshi N., Rokusawa K. and Chikayama T.: "Distributed Implementation of KL1 on the Multi-PSI/V2", Proc. Sixth Int. Conf. on Logic Programming, pp. 436-451 (June 1989).
(11) Nakajima K. and Ichiyoshi N.: "Evaluation of Inter-processor Communication in the KL1 Implementation on the Multi-PSI", ICOT Technical Report, TR-533 (1990).
(12) Nakashima H. and Nakajima K.: "Hardware Architecture of the Sequential Inference Machine: PSI

-II". Proc. 1987 Symposium on Logic Programming, pp. 104-113 (1987-09).

(13) Nitta K., Taki K. and Ichiyoshi N. : "Experimental Parallel Inference Software". Proc. FGCS'92 (June 1992).

(14) Onishi S., Matsumoto Y., Nakajima K. and Taki K. : "Evaluation of the KL1 Language System on the Multi-PSI". in Proc. Workshop on Parallel Implementation of Languages for Symbolic Computation (July 1990). Also ICOT Technical Report, TR-585.

(15) Rokusawa K., Ichiyoshi N., Chikayama T. and Nakashima H. : "An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems". Proc. 1988 ICPP, I. Architecture, pp. 18-22 (Aug. 1988).

(16) Shapiro E. : "The Family of Concurrent Logic Programming Languages". Comput. Surv., 21, 3, pp. 413-510 (1989).

(17) Takeda Y., Nakashima H., Masuda K., Chikayama T. and Taki K. : "A Load Balancing Mechanism for Large Scale Multiprocessor Systems and its Implementation". Proc. FGCS'88 (Nov. 1988).

(18) Taki K. : "The Parallel Software Research and Development Tool : Multi-PSI System". Programming of Future Generation Computers, ed. Fuchi K. and Nivat M., pp. 411-426, Elsevier Science Publishers B. V., North Holland (1988).

(19) Taki K. : "Parallel Inference Machine PIM". Proc. FGCS'92 (June 1992).

(20) Ueda K. : "Guarded Horn Clauses : A Parallel Logic Programming Language with the Concept of a Guard". ICOT Technical Report, TR-208 (1986).

(21) Ueda K. and Chikayama T. : "Design of the Kernal Language for the Parallel Inference Machine". Comput. J., 33, 6, pp. 494-500 (Dec. 1990).

(22) 和出久美子，市吉伸行："マルチPSI上の最短経路問題の実現と評価"，先記計算機アーキテクチャ研資，89-79，pp. 83-91 (1989-11).

(23) Wada K. and Ichiyoshi N. : "A study of mapping locally message exchanging algorithms on a loosely coupled multiprocessor". ICOT Technical Report, TR-587 (1990).

（平成4年5月12日受付，5月22日再受付）

瀧　和男

昭51神戸大・工・電子卒，昭54同大大学院修士課程システム工学了，工博，同年(株)日立製作所入社．同社大みか工場にて制御用計算機システムの設計に従事．昭57(財)新世代コンピュータ技術開発機構に出向，以来逐次型および並列型推論マシンと並列応用プログラムの研究開発に従事，現在第1研究室室長．並列マシンのアーキテクチャ，並列プログラミング，LSI CADなどに興味をもつ．情報処理学会，IEEE，ソフトウェア科学会各会員．

市吉　伸行

昭54東大・理・情報科学卒，昭56同大大学院修士課程了，昭57(株)二菱総合研究所入社，昭59より1年間米国パネル研究所にて人工知能を研修．昭62より平3まで(財)新世代コンピュータ技術開発機構へ出向，論理型言語処理系，大規模並列プログラムの研究開発に従事．情報処理学会会員．