計算機アーキテクチャ　112－8
（１９９５．　６．　２３）

並列オブジェクト指向トータルアーキテクチャA-NET
－ノードプロセッサの実装とその評価－

吉永　努　　澤田　東　　馬場　敬信

宇都宮大学・工学部・情報工学科
E-mail: yoshi@infor.utsunomiya-u.ac.jp

並列オブジェクト指向計算モデルに基づく高並列計算機のノードプロセッサを設計し，プロトタイプを試作した．このノードプロセッサは，メソッドを実行するPEと，メッセージの送受信を行うルータからなる．PEは，メッセージ送受信命令などの高機能命令セットをマイクロプログラムで実行する．また，同期機構や動的データ型付けなどをサポートするためのタグ付きアーキテクチャを採用し，これをタグ処理ユニットによりハードウェア支援する．TPUの効果を加算命令を例に調べた結果，TPUを使用しない場合より実行時間（マイクロステップ数）を約44%高速化できることが分かった．プロトタイプを30MHzで動作させてメッセージの転送時間を実測した結果，35バイトのメッセージを隣接ノードに転送するのに約5.2 μ s，1ホップ当たりの遅延時間が約1.0 μ sであることが確認できた．

A Parallel Object-Oriented Total Architecture A-NET
– Implementation and Evaluation of the Node Processor –

Tsutomu YOSHINAGA, Akira Sawada, and Takanobu BABA
Department of Information Science
Faculty of Engineering, Utsunomiya University
2753 Ishii-machi, Utsunomiya-shi, 321 Japan

We have developed a node processor for a multicomputer based on a parallel object-oriented computation model. The node processor consists of a processing element (PE) and a router. The prototype PE has a high-level instruction set which includes message sending and receiving. It also supports tagged data structure for a synchronization mechanism and dynamic data typing. The special hardware unit for tag processing speeds up about 44% of its execution time for an add instruction.

The router has been designed as independent to the network-topology. It takes approximately 5.2 μ s to transfer a 35-byte message to an adjacent node. When it transfers a message over multiple nodes, it takes approximately 1.0 μ s per hop.

1 はじめに

並列処理との親和性の良さから，オブジェクト指向計算モデルに基づいて言語とマシンを統合して開発する試みがある．POOMA は，ネットワークトポロジ独立な通信プロセッサで静的，動的なルーティングをサポートした点に特徴があった [2]．J-Machine は，通信や同期機構を統合したMDP と呼ぶメッセージ駆動プロセッサを用いて，並列オブジェクト指向ばかりでなく，複数の並列プログラミングモデルを効率よくサポートすることを設計方針としている [3]．

我々の研究室では，並列オブジェクト指向トータルアーキテクチャA-NET の開発を進めている [1]．これまでに，記述言語として A-NETL[8] を設計し，メッセージパッシング型並列プログラムの記述とシミュレーションによる動作検証を行ってきた．A-NETL のメッセージ送受信においては，例えばPVM[5] で必要な送信バッファの初期化やメッセージ引数のパック／アンパックなどをユーザプログラムで記述する必要がなく，簡潔にプログラムが記述できる．このような，高レベルの言語インタフェースを提供する場合，ユーザにとっては並列プログラムの記述が容易になる反面，実行時のメッセージ送受信や同期のオーバヘッドが大きくなりがちである．我々の目標は，ユーザにとってはできる限り自然に並列処理を記述できる言語インタフェースを提供しながら，それを必要最小限のハードウェアで効率的に実行する高並列マシンのノードプロセッサを開発することにある．A-NET ノードプロセッサの設計方針は，以下の通りである．

1. 中粒度の並列処理を対象とする．
2. ネットワークトポロジを静的に可変とする．
3. できるだけ簡単な構成の A-NETL 専用プロセッサとする．

以上の設計方針に基づき，PE とルータからなるノードプロセッサを設計し，そのプロトタイプを試作した．PE は，メッセージ送受信命令などの高機能命令セットをマイクロプログラムにより実行する．また，可変長命令のフェッチ／デコードを支援する命令前処理ユニットや，同期機構／動的データ型付けを支援するタグ処理ユニットなどに特徴がある．ルータは，メッセージの経路選択ロジックをプログラマブルにすると共に，適応制御が可能となっている．

本稿では，ノードプロセッサの構成とプロトタイプの性能評価について述べる．

2 ノードプロセッサ

2.1 命令セットアーキテクチャ

(1) 高機能命令セット

A-NETL 指向の高機能命令セットを定義し，これをマイクロプログラムで実行することとした．これは，主にプロトタイプマシンのチューニングに伴って必要となる設計変更に対して，コンパイラや OS などのソフトウェアに影響を与えることなく，ファームウェアで柔軟に対応できるようにするためである．

命令セットは，10 種類，79 命令からなり，通常のデータ移動，算術論理演算，分岐などの他，メッセージ送信，構造体操作，データ型操作などがあり，基本的に A-NETL のメッセージ式に一対一に対応する．2 項演算や配列／リスト操作などのプリミティブメソッドは，直接 PE の機械命令に翻訳されて，マイクロプログラムで実行される．また，メッセージ送信についても機械命令で実行し，ライブラリのコールやシステムコールは行わない．

例として，表 1 に幾つかの機械命令と，対応する A-NETL の記述を示す．A-NETL はデータ型定義の必要ない言語であるため，機械命令もデータ型によらない総称的なものとなっている．オペランドは一般に，即値，メソッドの一時変数，メッセージ引数，リテラル，オブジェクトの状態変数，分岐のための相対番地のいずれかである．即値と分岐番地以外は，ローカルメモリ上の位置がそれぞれのベースレジスタからのオフセット値で表される．オペランドデータの記憶場所をローカルメモリにした理由は，PE の設計方針としてプロセッサと小容量メモリが複雑な階層を持つことなく密に結合した構成を仮定したことによる．以上のような命令形式を採ることにより，オブジェクトをコンパクトにコンパイルして，ローカルメモリ上のワーキングセットを減らすと共に，メッセージ受信に伴うコンテクスト切り替えを高速化する．メッセージ送受信命令については，3章で詳しく述べる．

(2) タグ付きデータ構造

メッセージによる同期や動的データ型付け，ガーベジコレクションなどをサポートするために，タグ付きデータ構造を採用した．ローカルメモリ上のデータ毎に，データの存在を示すフラグやデータ型を持たせている．未来型メッセージの返値の格納先に指定された変数をリターンメッセージによる値の決定前に読み出そうとした場合，フラグのチェックによ

表 1: 命令セットと A-NETL の対応（一部）

種類		ニーモニック		相当する A-NETL 記述
データ転送		mv	des so	des = so
型変換		chgType	des sol so2	des = sol change: so2
フィールド操作	(挿入)	ins	des sol so2 so3	des insert: (sol, so2, so3)
	(抽出)	ext	des sol so2 so3	des = sol extract: (so2, so3)
算術演算	(加算)	add	des sol so2	des = sol + so2
論理演算	(論理積)	blAnd	des sol so2	des = sol && so2
配列操作	(参照)	at	des sol so2	des = sol at: so2
	(変更)	atPt	sol so2 so3	sol at: so2 put: so3
リスト操作	(参照)	fst	des so	des = so first
	(追加)	addFst	sol so2	sol addFirst: so2
メッセージ送信	(過去型)	sndP	obj sel sol so2 ...	obj sel: (sol, so2, ...)
マルチキャスト	(過去型)	sndPM	obj sel sol so2 ...	@obj sel: (sol, so2, ...)
メッセージ受信		rcv	sol so2 so3	sol receive: (so2, so3)

りトラップを発生してコンテクストをサスペンドさせる．データ型については，ユーザモードで不用意にそれを破壊しないよう，データ型コードの取り出しや型変換の命令を定義している．データ型に対して誤ったプリミティブメソッドを実行しようとした場合，マイクロプログラムによりエラーが検知され，OS の例外処理から実行エラーが通知される．また，世代更新方式のガーベジコレクションが PE 内でローカルにサポートされており，構造体データはヘッダにコピーフラグや世代を表すフィールドを持つ．

2.2 ハードウェア構成

図 1に，ノードプロセッサのブロック図を示す．

(1) P E

PE は，1 語 76 ビットの水平型マイクロ命令によって制御される．32 ビットの ALU, FPU の他，タグ処理のオーバヘッドを軽減するため，8 ビットのタグ処理ユニットを設けている．タグ付きアーキテクチャを反映して，メモリとファームウェアで使用するレジスタは，1 語 40 ビット構成である．その他，可変長命令のプリフェッチとデコードを支援するための命令前処理ユニットや，システム／ユーザそれぞれのベースレジスタなどの実行環境を格納する 2 セットの特殊レジスタなどに特徴がある．すでに述べたように，オペランドデータはローカルメモリ上に記憶されるため，メモリのアクセスはレジスタと同程度に高速であることが理想であるが，プロトタイプでは大学の研究室での実装を考慮してできるだけハードウェア構成を簡単化している．この方針のもとに，オペランドのアドレス計算は通常の ALU を用いることとし，データのキャッシングも行っていない．

メンテナンス回路としては，ホストとのデータ転送を行うポート，ノード番号の比較器，メンテナンスコマンドのデコーダを設けている．

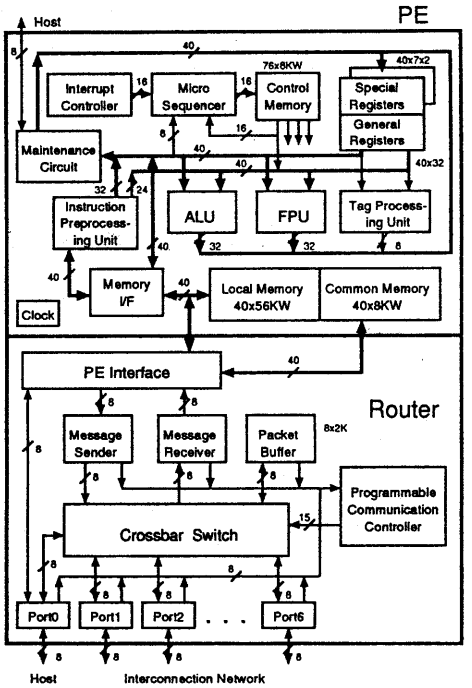

図 1: ノードプロセッサの構成

(2) ルータ

ルータは，ホスト用ポートと隣接ルータ間との結合用の 6 個のポートを持ち，これらとメッセージセンダ／レシーバ，パケットバッファ(PB) をクロスバスイッチ (CS) により結合する．メッセージの経路選択と CS の設定は，プログラマブル通信制御装置 (PCC) が行う．PCC は，CS につながったハードウェアブロックからラウンドロビン方式でメッセージの宛先を入力し，レシーバや送出可能なポートをプログラマブルなロジックで決定することにより，

トポロジ独立な制御を達成している．また，出力候補ポートから利用可能なものを選択して経路を適応制御する．CS の設定は，PCC が経路選択結果に基づいて，CS の制御フリップフロップを書き換えることで実現する．CS につながったハードウェアブロック間の接続は，他のブロック間でのデータ転送中にも可能であり，ルータ内で同時に複数メッセージを送受信できる．PB は，バーチャルカットスルールーティングを行うために設けてある．メッセージの最短経路上の出力候補ブロックがすべて使用中の時，そのメッセージは 1 回のラウンドロビンの間はその場で待たされるが，2 回目の順番が回った時にも転送できなければ PB に一時退避される [9]．PE インタフェースはホストからオブジェクトをロードする時と，メッセージ送受信時にメモリとセンダ／レシーバとのインタフェースとして使用する．

3 メッセージの送受信機構

我々は，ハードウェアの開発前にシミュレーションを繰り返し行い，性能とハードウェアコストのトレードオフから現在の実行モデルに到達した．ここでいうメッセージとは，単純なリモート変数のリード／ライトのように細粒度のものではなく，メッセージによって受け渡される引数や返答時の宛先，セレクタなどの実行環境を用いて一連の命令列であるメソッドの実行を依頼するものであり，またその返答である．

3.1 メッセージ送信

A-NETL による過去／未来型メッセージでは，オブジェクトは他のオブジェクトに何らかの処理を依頼して処理を継続する．したがって，送信処理が PE の実行をブロックするのは好ましくない．このため，PE-ルータ間のメッセージ送信用バッファとして FIFO を設けた．PE は FIFO にメッセージを格納することで送信を完了し，ルータは FIFO にメッセージが書き込まれた信号によって読み出しを開始する．PE から FIFO へのメッセージデータの格納は，実行中のユーザプログラムからコンテクスト切り替えが発生しないよう，ファームウェアによって実現した．メッセージ送信命令を実現するマイクロプログラムは，機械命令のオペランドによって指定される宛先オブジェクト ID，メッセージセレクタ，

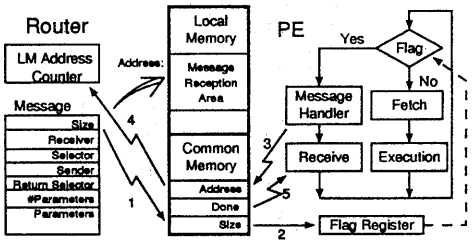

図 2: メッセージ受信

引数を FIFO に格納し，FIFO のポインタを更新する．PE 側の FIFO ポインタライトによってルータに割り込みが発生し，ルータが同一のアドレスを読み出すことで割り込みはクリアされる．

マルチキャストは，ファームウェアによる送信の繰り返しで実現する．したがって，ルータはマルチキャストに対する特別なハードウェアを必要としない．ルータのメッセージ送信は，すべてハードウェアのステートマシンによって制御される．

3.2 メッセージ受信

A-NETL ではメッセージ受信に関して，到着数，到着順序，引数のデータ型とサイズなど実行時に決定される要因が多く，Active Message のように受信前に到着データの領域を獲得しておくのは困難である [4]．また，メソッド起動に関しては，単純な FIFO 順序の処理に加え，複数メッセージの待ち合わせや，メッセージ引数とオブジェクトの状態変数の比較によりメソッド起動を遅延させる機能など，柔軟なスケジューリングモデルを設けている．

以前は，PE-ルータ間共有メモリ上に受信 FIFO を設け，PE 上でのメソッド実行とルータによる FIFO への到着メッセージの格納をオーバラップできるようにしていた．しかし，このような方法では受信メッセージを FIFO からユーザ領域にコピーするためのオーバヘッドが大きい [6]．そこで図 2 に示すように，ルータが直接 PE のローカルメモリ (LM) 上にメッセージを格納する方式に変更した．

1. ルータが，メッセージヘッダに含まれるサイズを共有メモリ (CM) 上にライトすることでメッセージの到着を通知する．
2. メッセージ到着が PE 上のフラグレジスタに反映され，機械命令のフェッチサイクルでローカル OS のメッセージハンドラにトラップする．

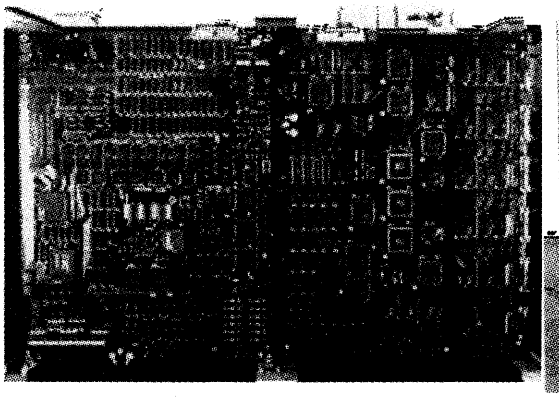

図 3: ノードプロセッサの写真

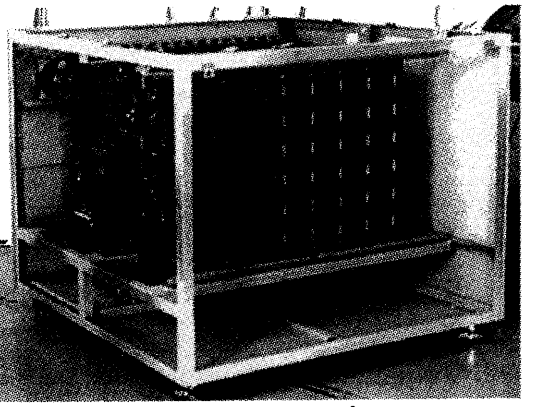

図 4: A-NET マルチコンピュータ

3. メッセージハンドラは，LM 上にメッセージの書き込み領域を獲得して，その先頭アドレスを CM にライトする．
4. ルータがアドレスカウンタをセットしてメッセージを DMA ライトする．
5. ローカル OS は，CM 上の変数 Done によりルータの書き込み完了を認知して，メソッドの起動，または返値の格納を行う．

この方式では，メッセージディスパッチ用の特別のハードウェアやソフトウェアによる FIFO 管理は必要ない．メッセージの到着は，ファームウェアにより機械命令境界でチェックされるため，優先順位の異なる複数の FIFO[3] やユーザプログラムにメッセージ到着を確認する命令を埋め込むことなく速達メッセージの実現が可能である．

4 プロトタイプ

図 3にノードプロセッサの写真を示す．左が PE，右がルータである．基板サイズは共に 500mm×375 mm で，内層に GND, VCC を持つ 4 層基板となっている．配線は，2.54mm のピン間に 3 本の設計ルールで行った．

PE，ルータ両基板は，160 ピンと 140 ピンのコネクタで直接結合する．ルータは，通信のためにこれらのコネクタを通して，PE 基板上の LM と CM をアクセスできる．

図 4に 8 ノードを実装した A-NET マルチコンピュータの写真を示す．プロトタイプでは，ハードウェアデバッグのためにホストの VME インタフェースと各ノードの PE を 40 芯のケーブルでバス結合する．これを用いて，ホスト側から，制御記憶，LM，CM，プログラムカウンタ，レジスタなどの読み書きとマイクロプログラムの実行を制御できる．制御記憶の読み書きやマイクロシーケンサへのアドレスセットなどはホストからのコマンドをメンテナンス回路が直接デコードして制御する．メモリやレジスタの読み書きについては，デバッグ用のマイクロプログラムをステップランすることで実現する．各 PE 上でノード番号をディップスイッチに設定し，ホストから送られるコマンドの対象ノードと比較して応答することで，ホストから 1 対 1，1 対多の制御ができる．

ホスト-ルータ間は，40 芯と 5 芯のケーブルでディジーチェーン接続しており，アプリケーションがホスト-ノード間の通信路として利用する．ホストからは，指定ノード，またはすべてのノードへのメッセージ転送やオブジェクトのロードが可能である．ルータからホストへのメッセージ通信は，各ルータのホスト用ポートにより衝突がないよう調停される．

ルータ-ルータ間は，ルータ基板のエッジに配置した 6 ポートの 26 ピンコネクタを介してケーブルで相互接続する．接続の仕方は設定したいトポロジに応じて静的に可変とし，PCC と併せて種々のトポロジでの実行を可能とする．

4.1 ハードウェア実装

表 2に PE, およびルータ基板に実装した部品の一覧を示す．

PE は，シーケンサ，ALU，FPU，レジスタファイルに AMD29300 シリーズの 32 ビット LSI ビルディングブロック（PGA）を用いた．PLCC は，マイクロデコーダ等に使用した Complex PLD（CPLD）と PE-

表 2: 実装部品数

	PE		Router	
PGA		6		3
PLCC	44pin CPLD	15	84pin CPLD	9
	8K×16 SRAM	3		
DIP	8K×8 SRAM	10	256×9 FIFO	9
	64K×4 SRAM	12	2K×9 FIFO	1
	24pin PLD	6	24pin PLD	12
	20pin PLD	12		
	TTL	165	TTL	106
	Switch	2	Switch	2
	R-PACK	9		
SIP		6		25
LED		39		25

ルータ間共有メモリ用のデュアルポート RAM である．DIP の内訳は，ローカルメモリ用 256K SRAM, 制御記憶用 64K SRAM, PLD, TTL, 集合抵抗, およびノード番号設定用スイッチである．デバッグのため，マイクロアドレスとプログラムカウンタを 7 セグメント LED を用いて表示すると共に，フラグなどを LED で表示する．

ルータは，クロスバスイッチに TI-SN74ACT8841（PGA）を使用し，各ブロックの制御部を CPLD（PLCC）で構成した．DIP の内訳は，ポートやメッセージセンダ／レシーバで経路選択の間一時的にメッセージを格納する 256×9 ビット FIFO, 出力ポートの衝突時にメッセージを退避する 2K×9 ビット FIFO, ネットワークトポロジに応じてポート未使用時に設定するスイッチ，その他，PLD，TTL からなる．また，SIP（抵抗）は主に基板外とつなぐ信号のターミネータとして使用している．LED は，デバッグのため FIFO の状態やパリティエラーを表示する．

5 評価

ノードプロセッサの単体性能，およびそれによって構成したマルチコンピュータの性能バランスを評価するために，プロトタイプノードプロセッサを使用した実験を行った．

5.1 命令セットの評価

表 3に，代表的な命令のマイクロステップ数を示す．これらは，実際にプロトタイプ上で動作を確認したマイクロプログラムの値である．オペランドが，即値かメモリ上のデータかによりステップ数が異なるため，最小と最大のステップ数を示す．基本的なレジスタ演算は 1 ステップで終了するが，プロトタイプでは，ローカルメモリからのデータのロードや総称的命令を実現するためのデータ型検査を実行時に行うため，1 命令に必要なステップ数が多くなっている．メッセージ送信命令 sndP のステップ数は，引数により大きく異なり，最小で 217 サイクル，30MHz 動作時に約 35.8 μ s を必要とする．なお，全マイクロプログラムの制御記憶容量は，2623 ワードである．

表 3: 命令毎のマイクロステップ数

命令	最小	最大
mv	8	10
chgType	8	10
ins	27	43
ext	24	41
add	13	21
blAnd	12	20
at	21	23
atPt	22	24
fst	11	12
AddFst	31	33
sndP	217	
rcv	43	

表 4: タグ処理ユニットの効果（加算の場合）

ステップ数	TPU あり	TPU なし	比
整数+整数	17	25	1.47
整数+小数	21	30	1.43
小数+小数	19	27	1.42
平均	19	27.3	1.44
サイズ	20	32	1.60

5.2 タグ処理ユニットの効果

タグ処理ユニット (TPU) は，実行時のデータ型や同期フラグ検査のオーバヘッドを緩和するために設けた．この効果を調べるために，加算命令を例にとり TPU がないと仮定した場合のマイクロプログラムを作成し，TPU を使用するマイクロプログラムとの比較を行った．TPU を使用しないプログラムでは，タグ処理も ALU を使用するものとする．結果を**表 4**に示す．整数同士の加算に要するステップ数の内訳は，加算マイクロプログラムへの分岐に 2 ステップ，データのロードに 4×2 ステップ（即値の時は 2×2 ステップ），データ型検査に 3 ステップ，レジスタ-レジスタ加算に 1 ステップ，結果の格納に 3 ステップである．データのロード／ストアの占める割合が大きいため，TPU の効果はそれほど顕著でないが，TPU がないとステップ数が平均で 1.44 倍，制御記憶容量が 1.6 倍になることが分かる．

なお，TPU に使用したハードウェアは，タグ移動

のための32to8データセレクタと8ビットバッファ×2及びパリティジェネレータ，タグ検査と設定のための32to16セレクタと24ビットPLD×2である．

5.3 命令前処理ユニットの効果

命令前処理ユニット(IPU)は，2本の命令レジスタ(IR)を使用して命令1語を先読みすると共に，フェッチした可変長命令から命令コードやオペランドを切り出す処理を支援する．オペランド処理としては，ベースレジスタ指定部とオフセット部の分離や即値の符号拡張を行う．IPUによって切り出された命令コードやオペランドのベースレジスタ部は，マイクロシーケンサに入力され，それぞれのマイクロプログラムへの多方向分岐に利用される．

IPUの効果を調べるために，IPUを使用せずに，汎用レジスタによりプログラムカウンタ(PC)とIRを代用するマイクロプログラムを作成し，その場合のステップ数を調べた．その結果，命令のフェッチに3ステップ，命令コードの処理に6ステップ，オペランド1つにつき10ステップ余分に必要であった．例えば，IPUを使用した整数加算は17ステップで終了するところ，IPUを使用しないと$17+3+6+10\times3=56$ステップかかる．以上のことから，IPUはA-NETL指向高機能命令のフェッチとデコードに大きく貢献しているといえる．

なお，IPUに使用したハードウェアは，19ビットPC，16ビットプリフェッチ用カウンタ，40ビット命令レジスタ×2，オペランド処理とIPUのコントローラに44ビットPLD×3などである．

5.4 メッセージ送受信性能

ルータによるメッセージの転送時間を実測した結果，送信ノードにおいてメッセージの先頭がネットワークに出力されるまで約47クロック，中継ノードで約30クロック，受信ノードでメッセージをローカルメモリに格納してローカルOSに引き渡すまでに約$38+2n$クロック必要なことが分かった(nはメッセージのバイト数)．値が決まらないのは，PCCによる経路選択要求がラウンドロビンで受付られるためである．30MHz動作時にはそれぞれ，約1.6, 1.0, (1.3+0.067n) μ sとなる．受信ノードの消費時間には，3.2節で述べたルータによるメッセージの書き込み完了を示す変数Doneの更新までを含む．したがって，メッセージを無衝突で距離D転送する時間は次式で表される．

$$\begin{aligned} T_d &= 1.6+1.0(D-1)+1.3+0.067\times n \\ &= 1.9+D+0.067\times n \quad (\mu s) \end{aligned}$$

この式から，A-NETLにおける最小のメッセージ35バイトを隣接ノードに転送するのに約5.2 μ sかかり，転送距離がのびる毎に約1.0 μ sの遅延が発生することが分かる．文献[9]で予備評価した時よりも高速となっているが，この理由は，クラスを予めすべてのノードに割り付け，動的オブジェクト生成に伴うオブジェクトコード転送を不要にして，ルータのロジックを簡単化したことによる．

5.5 ノード内処理と通信の関係

以上の議論のもとに，典型的なA-NETLプログラムにおけるメッセージ送受信を図示すると，図5のようになる．図中の数値が下限値で示してあるのは，引数の構造とサイズ，転送距離により増加することを表す．また，実際にはルータがメッセージをローカルメモリに格納する前にOSによる領域獲得が必要であるが，簡単化のためここではその時間をメッセージ格納後のローカルOSの処理に含めて考える．マイクロからローカルOSへのトラップは機械命令境界で発生するため変動する．仮に，メソッド起動処理の時間を文献[7]と同じ1472ステップと見積もると，約243 μ s(1サイクル165ns)を要する．今回評価したサンプルプログラムにおける，1メッセージ当たりのユーザの実行は平均2064.6サイクル，340.7 μ sであったことから，相対的にローカルOSの前処理が重いことが分かる．この理由として，A-NETLではメッセージの引数や同期モデルに自由度が大きく，実行時の処理が多いことが挙げられる．ただし，メッセージ引数のアンパック，受信処理をユーザが記述するのではなく，ローカルOSがサポートしているためにシステム／ユーザ実行時間の比が大きくなっているもので，ローカルOSが無駄な処理をしている訳ではない．

6 むすび

並列オブジェクト指向トータルアーキテクチャA-NETプロジェクトの一環として開発されたノードプロセッサの構成と評価について述べた．ノードプロセッサのアーキテクチャは，並列オブジェクト指向実

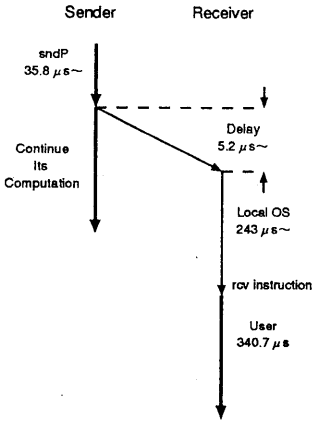

図 5: メッセージ送受信の時間的関係

行モデルを素直に実現したものであり，動的データ型付けやメッセージによる種々の同期機構など，並列プログラミングを容易にするための高レベル言語インタフェースをコンパイラ，ローカル OS，ハードウェアの協調作業によりサポートしている．また，プロトタイプ実装という位置付けから，PE はマイクロプログラム制御とした．このようなアプローチをとったことにより，OS やファームウェアのメッセージ通信における役割分担などは比較的柔軟に行える．

現在は，各ノード単位でのデバッグを終了し，5 ノードでの動作確認を行っている．今後，16 ノードにハードウェアを拡張し，A-NETL アプリケーションによるネットワークトポロジ独立性の評価などを予定している．

謝辞　ハードウェアの試作に関してご協力頂いた日本電気（株）C & C 研究所，(株) 東芝研究開発センター，(株) 日立製作所日立研究所，(株) 大晶電子，富士精密（株），日本 AMD（株）の各社に感謝する．また，試作に携わった岩本善行君をはじめとする研究室の各位に感謝する．

本研究は，一部，文部省科学研究費補助金（試験研究 (B) 課題番号 04555077，重点領域研究「超並列処理」課題番号 04235104，一般研究 (c) 課題番号 07680334，奨励研究，課題番号 07780225）の補助による．

参考文献

[1] 馬場敬信，吉永努:“並列オブジェクト指向トータルアーキテクチャA-NET における言語とアーキテクチャの統合”，信学論 (D-I) 招待論文，**J75-D-I**, 8, pp.563–574(1992).

[2] America P.H.M., Hulshof B.J.A., Odijk E.A.M, Sijstermans F., Van Twist R.A.H, and Wester R.H.H.: “Parallel Computers for Advanced Information Processing”, IEEE Micro, pp.12–15, 61–75(Dec. 1990).

[3] Dally W.J., Chien A., Fiske S., Horwat W., Keen J., Larivee M., Lethin R., Nuth P., and Wills S.: “The J-Machine: Fine-Grain Concurrent Computer”, Proc. IFIP Congress, pp.1147–1153(1989).

[4] Eicken T., Culler D.E., Goldstein S.C., and Schauser K.E.: “Active Message: a Mechanism for Integrated Communication and Computation”, Proc. 19th ISCA, pp.256–266 (1992).

[5] Geist A., Beguelin A., Dongarra J., Jiang W., Mancheck R., and Sunderam V.: “PVM 3 User's Guide and Reference Manual”, Oak Ridge National Laboratory, ORNL/TM-12187(1994).

[6] 堀江健志，小柳洋一，今村信貴，林憲一，清水俊幸，石畑宏明：“メッセージ通信の分散メモリ型並列計算機性能への影響 –通信と演算のオーバラップと直接メッセージ受信の効果–”，情処学論, **35**, 4, pp.609–618(1994).

[7] 田口弘史，吉永努，馬場敬信：“A-NET ローカル OS 第 3 版–メッセージ受理動作の高速化とその評価”，コンピュータシステム・シンポジウム論文集，pp.51–58(1993).

[8] 吉永努，馬場敬信：“トポロジカルなプログラミングが可能な並列オブジェクト指向言語 A-NETL”，信学論 (D-I)，**J77-D-I**, 8, pp.557–566(1994).

[9] 吉永努，馬場敬信：“並列オブジェクト指向トータルアーキテクチャA-NET のためのトポロジ独立なルータの構成”，情処学論, **34**, 4, pp.648–657(1993).