計算機アーキテクチャ 48－3
（1983. 3. 18）

分散処理用計算機のシステム・アーキテクチャ

甘田 早苗　土田 正士　佐藤 豊
後藤 勝己　馬場 公光（筑波大学）

1　まえがき

近年計算機システムの使用上の見地から分散処理方式が注目され，数多くの研究が行われると共に，かなりの数のシステムが実用に供されている．しかしその多くは既に存在する計算機システムをベースとして，これを分散処理の目的にかなうようにシステムとしてまとめ上げることに重点がおかれており，分散処理の基本的要請を分析し，これに即した新らしいシステムを開発しようという努力はあまり見受けられない．このような状況に留まっているのは，集中処理システムから逐次分散処理へ移行するケースが多い事，既存のプログラムとの互換性が要求される事などの理由によるのではあろうが，分散処理というものが一時的な流行ではなく，計算機システムの使用の拡大に対して本質的な要望をみたすものである以上，不満足なものだといわざるをえない．

われわれは分散処理の基本的な要請を改めて分析し，この要請を最新の技術水準でより多くみたすようなシステム・アーキテクチャはどうあるべきかを検討している．作業はまだ進行中であるが，以下に検討の経緯と概略の検討結果を述べる．

2　分散処理用計算機への要請

分散処理用計算機にどのような機能が必要であるかを巨視的に要約すると次の4項目になる．

(i) 高い性能価格比
(ii) 高いシステムの信頼性(Reliability/Security)
(iii) ソフトウエアの作成が容易なこと
(iv) 高いシステムの柔軟性

これらの項目を見る限りでは，他のどのような計算機に対する要請とも，なんら変るところはない．しかしその設置される環境を考察するとき，分散処理なるが故に特に慎重に検討しなければならない内容が含まれていることが明白になる．

分散処理システムの本来の姿が，システムを使用する企業・組織体の組織構造と密着した処理を企図するものであり，かつ処理装置が地域的にも分散配置されることを前提とする以上，[1] システムの設置，運用，保守上の環境は集中処理システムとは著るしく異なるものとなる．その環境条件とは，

a) 処理装置を運用し，保守するに際しては，計算機の専門技術者はまったく居ないか，居てもごく少人数であり，集中処理システムがいわゆるEDP要員によって運用，保守されるのとは著るしく異なる．
b) 応用プログラムは，専門の要員ではない各現場の職員によって作成される．プログラムの作成要綱等は準備されているとしても，現場の職員が日常業務の一環としてプログラムを開発するのが通常の姿となる．
c) システム製造業者の保守サービス拠点から，距離的にかなり離れている場合を想定しなければならない．従っていったん障害が起るとその修復には長い時間を要するし，プログラム作成上のアドバイス等も受けに

くい.

d) 各処理装置が現場に近接して設置される関係上，処理装置の停止が業務に与える影響は集中処理システムよりはるかに直接的である．処理内容そのものが現場の業務により密着しているためでもある．

e) 情報の処理量，処理内容，処理手順等は処理装置の設置される場所によって大幅に変化する．またいったん設置された装置も処理量，処理内容等が後になって大きく変動する可能性がある．処理装置はそのような多様性や変動に対応できなければならない．

f) しかし一方では，組織体の全システムを構成する各処理装置が，論理的にも相互に有機的に結合されることが必要である．そのためには各処理装置の基本的なアーキテクチャが共通であることが好ましい．

g) 各処理装置が有機的に結合されたシステムに対して，常に多数のユーザが種々なプログラムからアクセスすることになる．これに対応してプログラムやデータは強く保護されていなければならない．故意や過失でプログラムやデータが破壊される危険性は大きい．

表.1 要求機能と環境

	性能価格比	Reliability	Security	ソフトウエア作成	システムの柔軟性
運用・保守は現場の手で		○		○	
応用プログラム作成者は現場職員			○	○	
メーカの保守サービス拠点から遠い		○		○	
システムダウンの影響が直接的		○			○
処理に対する要求が多様					○
全システムの統合運用が重要					○
マルチユーザ・マルチプログラム環境			○		
上記条件をみたした上での経済性	○				

h) 以上述べてきた環境条件のもとでシステムは十分な経済性を保たねばならない．

上記の(i)~(iv)の項目とa)~h)で示した環境条件との関係を表.1にしめすが，表中○印はとくに関係が密なことをあらわす．われわれは，既存のシステムではとくにシステムの信頼性と柔軟性の面で問題が多いと判断し，これを念頭においてシステム設計を推進することとした．

3 基本設計

上記の検討をもとにシステムとしての基本設計を行う．その主な項目を以下に示すが，また各項目と上記(i)~(iv)の関係を表.2に掲げる．さらに各項目間にも相互関係があるので，これを表.3にしめす．

3.1 マルチプロセッサ方式の採用

然るべき設計をしたマルチプロセッサ方式であれば，プロセッサをシステムに組込むか，取り除くかによって，システム性能をかなり大幅に変更することができる．従って通常ハードウエアの性能向上のために行われるシステム入替は，新たに必要なプロセシング・ユニットをシステムに追加するだけで済まされる．逆にシステムに対する要求性能が縮小した場合にもプロセシング・ユニットを取り外すだけで経済的な運用が可能である．

また結合するプロセッサとして特定業務にチューニングしたものを使えば，多彩なアプリケーションに対して高い効率をもったシステムを用意することができる．極めて特殊な処理の要求があれば，その目的に適合した専用プロセッサを接続するのも一つの方策である．

オペレーティング・システムが適切

ならばフェイル・ソフトなシステムにすることができ，Reliability の向上に役立つ．

ハードウエア・インタフェース部の設計が適切であれば，プロセッサの着脱に要する時間はかなり短かくすることが可能である．これは現場に近く設置されるためにシステムの停止時間を極力小さくしなければならないという要望に良く適合するものである．

現在の半導体技術をもってすれば，マルチプロセッサ方式が性能価格比を劣化させる要因とはなりえない．むしろ並列処理によって性能を向上しうる可能性が高い．

これらの条項を勘案して，マルチプロセッサ方式を採用することに決定した．

3.2 プロセッサ間通信方式の開発

マルチプロセッサ方式の最大の欠点は，プロセッサ間通信の疎通能力によりシステム性能が制約されることである．これの対策はプロセッサ間通信の頻度をへらすこと，通信路の転送性能を上げること，通信に伴うオーバヘッドを小さくすることにつきる．

通信頻度をへらす対策としては，1つのタスクを無理に分割して複数のプロセッサに割り当てるのをやめることが考えられる．このシステムが通常マルチユーザ，マルチプログラミングの環境下で動作することを考えるならば，このような方針はマルチプロセッサによる並列処理に期待した性能の向上を損うものではない．実現はオペレーティング・システムの機能の決め方によって可能である．

プロセッサを容易にシステムに結合し，また切放せることと，通信路の情報転送能力を極力高くするという観点から，通信路にはバス構造を主体としたネットワークを採用する．もちろんバスとプロセシング・ユニットとのインタフェース部分の設計には慎重な配慮が必要である．とくにこの部分の素子の障害によってバスが機能を失い，システムダウンが惹き起されることがないよう，注意しなくてはならない．バス使用権管理の容易さや，このバスが内部バスの位置づけにあることなどを勘案して，バスの制御手順にはデータ通信におけるベーシック制御手順に近い方式を採用することとした．

バス上を流れる情報については，メッセージ・レベルの短かい制御情報と，

表.2 要求機能とシステム・アーキテクチャ

	性能価格比	Reliability	Security	ソフトウエア作成	システムの柔軟性
マルチプロセッサ方式の採用		○			○
プロセッサ間通信方式の開発	○				○
Capability-Based Address			○	○	
高水準言語の採用		○		○	
ファームウエアの重視	○				○
オペレーティング・システムの見直し	○	○	○	○	○

表.3 システム・アーキテクチャ上の項目の相互関係

△印は相互関係が幾分弱いことをしめす．

動機 ＼ 対策	マルチプロセッサ方式の採用	プロセッサ間通信方式の開発	Capability-Based Addr.	高水準言語の採用	ファームウエアの重視	オペレーティング・システムの見直し
マルチプロセッサ方式の採用		○	○			○
プロセッサ間通信方式の開発			△		△	
Capability-Based Addr.				△	△	○
高水準言語の採用					○	
ファームウエアの重視						
オペレーティング・システムの見直し				△	△	

多量のデータを転送する際の長い情報と短い情報が混在することは明白であり，このような場合短かい情報の疎通にまず問題が生じることが知られている.[2] システム制御情報の転送が遅れることは大きな問題なので，バスを2本設け，短かい情報と長い情報とでバスを使い分けることとする．またバッファ管理などの観点から，双方共最大ブロック長を設定する．

通信のオーバヘッドの縮小は，転送時のプロトコルを簡略化することと，プロトコルの生成，解読の速度を上げることで達成される．後者に対してはファームウエアの活用によって処理速度を上げ，前者に対しては，とくに短かい情報を転送するバスに関してハードウエアのバス・アービタと，これから星形に各プロセシング・ユニットにつながれる信号線とを設け，これによって使用権を与える方式を案出して採用することとした．この方式ではバス使用権の要求と割当は単に上記信号線の論理レベルを変更するだけでよいので，処理は短時間で済む．またバスの使用権はアービタが一元的に割当てるのでバス上でデータが衝突することはない．さらにアービタに若干のインテリジェンスを与えるなら，バス使用権を有機的に管理することも可能である．

3.3 Capability-Based Address方式の採用

本システムに於ては記憶保護は極めて重要な課題である．また命令セットとしてgenericな命令が使えるならば，プログラムの作成が容易になる．これらを実現する手段としてcapability-based address方式を採用することとした.[3] いうまでもなく，この方式はリスト参照回数が増加し，トータルスループットが低下するおそれがあるが,[4] これはハードウエア設計上の工夫によって解決することとした．なおこの方式では所要の記憶領域が増大するのではないかとの疑念も持たれるが，実質的には問題がないという議論がすでに行われている.[5]

またこの方式の採用により，仮想記憶方式を同時に実現できることも1つの魅力であるし，マルチプロセッサ構造を意識することなくプロセス間通信やオブジェクトの共有ができることも好ましい．

3.4 高水準言語の全面的採用

ハードウエアをいわゆる中間言語マシンとし，アプリケーション・プログラムのみならずシステム記述言語としても高水準言語を使用し，プログラム開発効率を向上させる．後述するように本システムのプロセッサはファームウエア・マシンとするので，アプリケーション用言語の種類にはあまりこだわらないが，当面Pascal系言語を対象として考える．これに伴ってシステム記述言語としてはConcurrent Pascalを拡張して使う予定である．この言語によりシングルユーザのもとでの並行動作プログラムを記述した例があり，また仕様の拡張によってマルチユーザのシステム記述に十分耐えることが知られているからである.[6]

3.5 ファームウエアの重視

すでに述べてきた事柄からも推測され，また一般的な技術動向からもそうであるように，本システムのプロセッサはファームウエアを重視した構造とする．すなわち，中間言語の処理，基本的なアーキテクチャは統一しながら実行上のアーキテクチャは処理内容にチューニングすること（いわゆるWCSを含む），従来ソフトウエアに委されていた処理の一部をファームウエア化して処理効率を向上することなどがそ

のねらいであり，これに代る方法は見当らない．

ファームウエアの設計を慎重に行なえば，同一のハードコア（将来LSI化を考える）によってかなり多様なプロセッサをインプリメントすることが可能なはずであり，それが実現すればシステムへの柔軟性の付与と経済化にもつながる．

3.6 オペレーティング・システムの見なおし

以上述べてきたシステムを管理制御するオペレーティング・システム（以下OSと書く）は，当然従来の集中処理システム用のそれとはかなり趣を異にするものになるはずである．ここでさらに積極的にシステム効率の向上をはかると共に，保守や機能の変更が容易なものとしたい．基本的には階層化・モジュール化を推進し，処理の単位をプロセスとして取扱い，全体的な動作はプロセス間通信として実行することとする．さらに従来むしろ慣習的に行われてきた複雑な手続きなどを整理し定式化するといった改良をつけ加えてゆくこととしたい．

4 基本設計仕様

以下に設計仕様の概要を述べる．

4.1 システム構成

基本的なシステム構成は図.1にしめすように先にわれわれが発表したとおりである.[7] 各種類のプロセッサについてそれぞれ複数台をシステムに組込むことを許すことにしている．先にあげた短かい情報用と長い情報用のバスに，それぞれメッセージ・バスとメモリ・バスという名称を与えている．

各プロセッサはめいめい主記憶をもち，原則として共有記憶はない．共有記憶を持つことに長所も短所もあることは周知の通りであるが，この選択はプログラムの局所性を重視し，共有記憶での競合を避け，かつ共有によって生じるOSの複雑さを回避することに重点を置いたことによる．しかし疎結合マルチプロセッサ方式で高いパフォーマンスを得るためには，主としてOSに新たな工夫が必要なこともまた明白である．

われわれはこのシステムをADMS (Advanced Distributed Multiprocessor System)と名付けた．

4.2 プロセシング・ユニットの構成

図.1の諸プロセッサに該当する各プロセシング・ユニットは，図.2にしめすように2個のプロセッサ・エレメント，主記憶，およびBIUを含むインタフェースから構成される．

第1のプロセッサ・エレメントをデータ・プロセッサと呼ぶこととし，内部には1つのプロセッサ，キャッシュ・メモリ，若干の補助回路，制御記憶，およびインタフェースをもつ．このプロセッサの1つの役割は，高水準言語で書かれたソースプログラムをコンパイルして得られた中間コード列を，ファームウエアと若干のソフトウエアとに

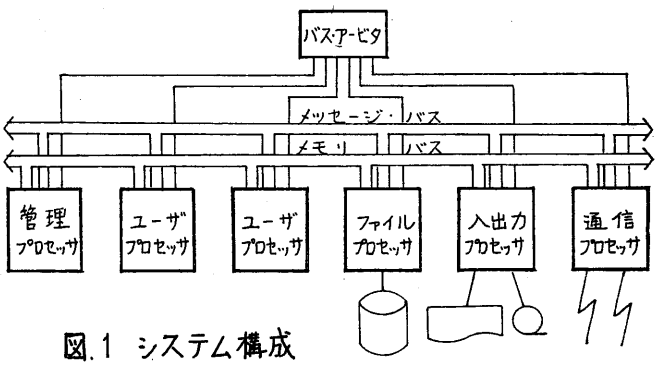
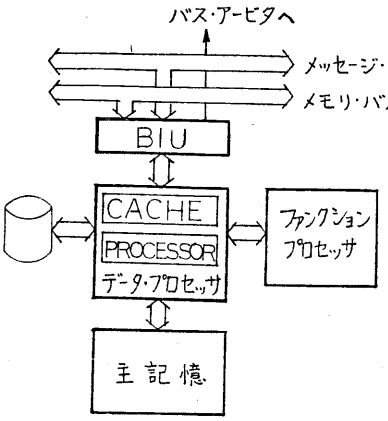


図.1 システム構成

図.2　プロセシング・ユニットの構成



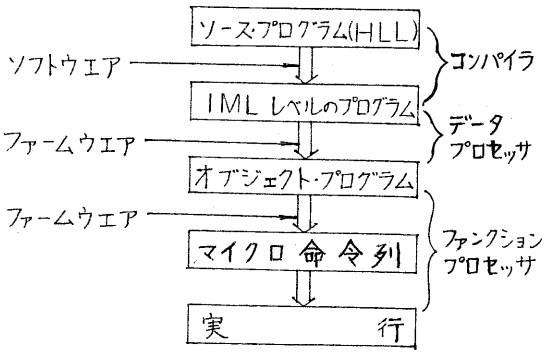
よってオブジェクトプログラムに変換することである．この変換過程において，仮想アドレスを物理アドレスに変換するとともに，データの型とアクセス権のチェックを行い，その結果は，キャッシュに貯えられる．今1つの役割はキャッシュの内容を第2のプロセッサ・エレメントであるファンクション・プロセッサ，主記憶，バスとのインタフェースを司るBIU(Bus Interface Unit)，およびこのプロセッサに接続される周辺装置等へ転送し，またそれらから情報を受けとることである．

ファンクション・プロセッサはデータ・プロセッサ内のキャッシュ上の情報を受け取り，処理を実行するものであるが，受け取ったオブジェクト・プログラムはファームウエアによってマイクロ命令レベルに変換した後，ハードコアで処理する構成である．ファームウエアを処理内容に応じてチューニングすれば処理効率は向上する．

かようにプロセッサ・エレメントを2段階設けるのは，同一仕様で作られたデータ・プロセッサ相互間で情報の転送を行うことによって，プロセシング・ユニット間の結合お容易に緊密に行なえること，処理目的別にチューニングされたファンクション・プロセッサによって処理効率を高められること，かなりの深さをもつファームウエア処理を階層化してファームウエア・レベルの効率を高めること，および2つのプロセッサ・エレメントが独立に並列動作することによるスループットの向上をねらったものである．上記のプログラム実行過程を図.3に示す．

なおプロセッサ自体の仕様は，語長32ビット，仮想アドレス空間4Gバイトを想定している．

図.3　プログラムの実行過程



4.3　バスおよびBIU仕様

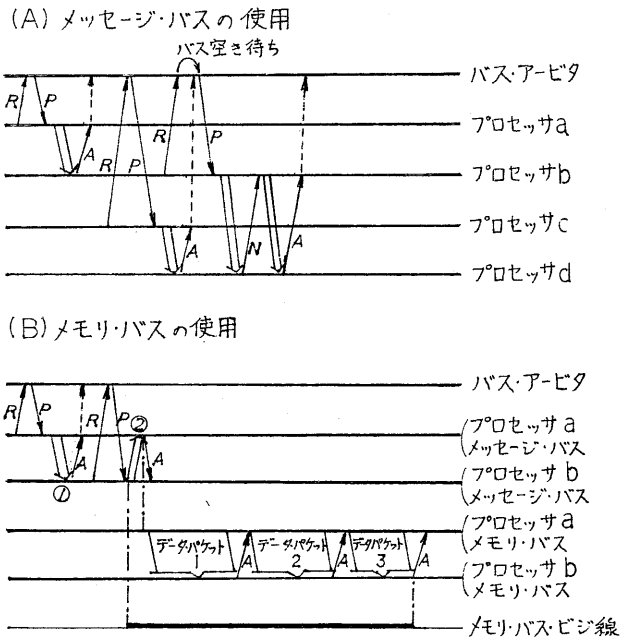
本システムで使用する2本のバスは共にビット・パラレル，バイト・シリアルな情報転送を行い，転送速度は10Mバイト/Sを目途とする．転送速度は速いにこした事はないが，TTL素子を使用した場合この速度はほぼ上限であり，また32ビット／語の主記憶のサイクルタイムを400nS程度に設定した場合，大凡それに見合う速度である．バスの多重化使用は行わない．

転送速度を確保する立場から，バス長は最大10m，バスに接続するプロセッサ数は最大16台を想定している．またメッセージ・バスは通常16バイト以下，メモリ・バスでは最大256バイト(拡張可)で1パケットを形成する．

BIUはこのようなバス，およびバス・

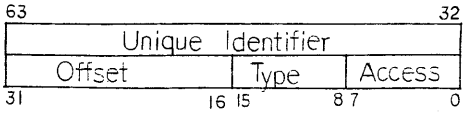
アービタに接続する星形の信号線と，データ・プロセッサとの間のインタ・フェースを形成するものである．これはバスを経由して通信を行う際，データ・プロセッサにかかる負担をなるべく小さくすることを設計上の基本方針としている．そのため送受信に際して使用する若干のバッファを持ち，また伝送制御手順がほとんどBIU相互の間で完結するよう配慮している．伝送制御手順はベーシックモード手順をやや簡略化したものを使用し，DDX網でいうデリミタクラスに相当する．誤り検出は水平パリティ・チェックのみであるが，これはバス長が短かく，かつ装置内に設けられ，線材も同軸線を使うので，伝送誤りは十分小さいという予想に基ずく．バス使用のシーケンスの1例を図.4にしめす．

図.4 バス使用のシーケンス



略号　R メッセージ・バス要求　①メモリ・バス転送通知
P メッセージ・バス使用許可　②受信レディ
A 肯定応答
N 否定応答

図.5 Capability Format



4.4 ケーパビリティのフォーマット

フォーマットは図.5にしめす構造をもち，各ブロックの意味づけは次のとおりである．

- Unique Identifier：論理アドレス空間を指定するもので，システム全体でユニークに設定される．
- Offset：実行時に物理アドレスに変換されたUnique Identifierからの変位をしめす．
- Type：システムタイプ12種，データタイプ10種の指定．
- Access：アクセス权4種，アクセスのステータス4種をしめす．

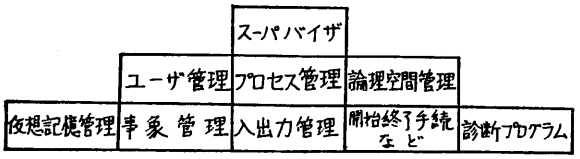
4.5 オペレーティング・システム

OSの基本的設計方針は一貫性，単純性，安全性，柔軟性の追及である．またその底流にあるのは，従来行われてきたような管理・制御機能のすべてを，何から何までOSにおしつける立場を見直し，真にOSが担当すべき分野が何であるかを見極め，ハードウエアと協調分担して全体としてのシステム効率を高めることにある．

さらに具体的にいえばわれわれのシステムが疎結合マルチプロセッサ方式，抽象データ型とcapability-based addressingを採用し，通信機能を重視する必要があり，システムの柔軟性を強く要求されるといった事情を反映する必要がある．

現実の構造は図.6にしめすように階層化，モジュール化を基

図.6　オペレーティング・システムの構造



本としているが，これは並列性による高速化，柔軟性，および開発の容易性を企図したものである．オブジェクト単位での抽象データ表現・操作を基本とし，ＯＳの各モジュールもプロセスオブジェクトとして実現する．

通信負担の減少，各プロセッサの負荷の均等化，システムのフェイル・ソフト性などの要件をみたす手段としてプロセスの動的アロケーションを行い，外部オブジェクトへのアクセスはポートを介して行う[8]．名前管理の手段としてはユーザごとにスコープとスコープルールを与え，スコープは木構造の名前で実現する．このことを利用して，システム内の全対象をすべてオブジェクトとしてとらえ，一様なアドレス方式とアクセス手段を適用する．

以上のような構想のもとに仕様を作成中である．

4.6　その他

ローカルエリア・ネットワークや通信回線との結合については，世界的なレベルで標準化が進みつつあるので，これに準拠する．

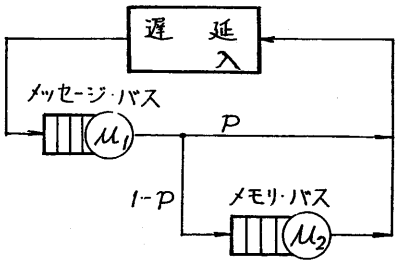
入出力装置に対しては従来の仕様をほぼそのまま適用する．

ファイルの構成と管理の問題が分散処理において特に重要なことは明白であるが本稿ではこの問題にはふれない．

5　数値的な解析

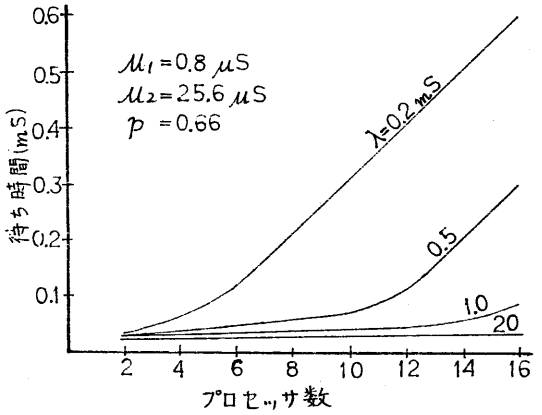
5.1　システム・シミュレーション

図.7　バスのトラフィック解析モデル



バスの情報疎通能力がシステム性能に大きな影響を及ぼすはずであるので，図.7にしめすような解析モデルを作成してシミュレーションを行った．結果の一例を図.8にしめすが，現在の仕様において計算機の諸元が現実的な値の範囲内であるならば，１６台のプロセッサを接続してもバス・ネックによってシステムの性能が劣化することはないという結果が得られている．

図.8　メモリ・バス待ち時間



5.2　中間コードの出現頻度

つぎにハードウエアの具体的設計やシステム記述言語の仕様作成に資するために，中間コードの振舞を追跡した．対象として，基本的にはわれわれのシステムに親和性の良いSolo OS[9]を選び，そのＯＳのもとでコンパイルを実行中に出現する中間コードの頻度を集

表4　中間コードの使用頻度

中間コード名	出現回数	%	% (累計)
PUSHGLOB	29964	12.83	12.83
PUSHCONST	25337	10.85	23.68
GLOBADDR	24197	10.36	34.04
PUSHLOCAL	23861	10.21	44.26
FALSEJUMP	14695	6.293	50.55
COPYWORD	13454	5.761	56.32
EQWORD	11198	4.795	61.11
INDEX	9617	4.118	65.23
CALL	6423	2.750	67.98
ADDWORD	6047	2.589	70.57
FIELD	5427	2.324	72.89
EXITCLASS	4412	1.889	74.78
ENTERCLASS	4412	1.889	76.67
ENTERPROC	4259	1.823	78.50
CALLSYS	4259	1.823	80.32
EXITPROC	4257	1.823	82.14
JUMP	3562	1.525	83.67
COPYBYTE	3389	1.451	85.12
LOCALADDR	2971	1.272	86.39
PUSHBYTE	2863	1.226	87.62
PUSHIND	2782	1.191	88.81
MODWORD	2640	1.130	89.94
ORWORD	2577	1.103	91.05

計して分析を行った．結果の一部を表．4にしめすが，コンパイルの過程で現れた23万個あまりの中間コードの集計結果である．このデータを分析した結果次のようなことが明らかになった．

- 中間コードの使用頻度にかたよりがあるのは当然であるが，全部で112種の中間コードのうちの1割，11種で全使用頻度の73%をしめる．ハードウエア，ファームウエアでこれらのコードの処理時間を短縮することが必要である．
- スタック操作関係の中間コードが約45%，演算関係が約14%，プロシジャー・コール関係が約9%，そしてアドレス計算関係が約12%の頻度で現れる．従ってプロシジャー・コールは平均11命令毎に発生する．
- 入出力命令は約3,100命令に1回の割合で発生し，その都度平均512バイトのデータが転送される．1命令あたり平均1.3ビットの入出力が行われることになるが，これは従来の通念にほぼ適合するものである．
- これはまた，1,550命令が実行されるたびに256バイトの入出力が行われることを意味する．プロセッサの性能を1MIPSと想定した場合，これは1.55mSごとに256バイトのデータ転送が起るということであり，図.8において$\lambda=1.55$を求めるなわ，本システムの妥当性が証明されたことになる．
- プロセス間通信は約4,000命令に一度行われる．

6　あとがき

分散処理システムが集中処理システムと要請上異なる点を明らかにし，それに基いて分散処理用計算機のシステム設計を行い，基本設計仕様をまとめた．現在その内容の妥当性の検討を行うとともに，細部仕様の設定と論理設計の作業をすすめている．

当然作業結果のフィードバックによって仕様に変更が生じる可能性はあるが，大筋は固定されたものと考えて，ここに報告する．

なお上に述べた項目のうち若干のものについては，別途詳しく報告をしたので，(10)(11)(12)(13) 参照していただきたい．

本研究は筑波大学の研究・教育設備，特別設備費による設備，および富士通(株)の奨学寄附金によって行っているものである．

参考文献

(1) R.W.Watson; Distributed System Architecture Model, Lecture Notes in Computer 105, 1981, pp.10-43, (Springer-Verlag)
(2) 高平 叡; 分散処理用オペレーティングシステムの課題,

情報処理, Vol.20, No.4, 1979, pp.284-288.
(3) R.S.Fabry; Capability-Based Addressing, Comm. ACM, July 1974, pp.403-412.
(4) T.A.Linden; Operating System Structure to Support Security and Reliable Software, Computing Surveys, December 1976, pp.409-445.
(5) G.J.Myers; Advances in Computer Architecture, 1978, (John Wiley and Sons, Inc.)
(6) R.J.Price; A Language for Distributed Processing, National Computer Conference, 1979, pp.957-967.
(7) 土田, 佐藤, 甘田; マルチプロセッサ向きシステム・アーキテクチャの考察, 情報処理学会第24回全国大会, 1D-5, 1982.
(8) B.Liskov; Primitives for Distributed Computing, 7th Symp. on Operating Sys. Principles, 1979, pp.33-42.
(9) P.B.Hansen; The Architecture of Concurrent Programs, 1977, (Prentice-Hall Inc.)
(10) 土田, 甘田; マルチプロセッサシステムのシステム設計, 情報処理学会第26回全国大会, 1N-1, 1983.
(11) 佐藤, 土田, 甘田; マルチプロセッサ向きOSの構造に関する一考察, 同上, 1N-2, 1983.
(12) 馬場, 甘田, 土田; マルチプロセッサ向きアーキテクチャ支援言語の設計, 同上, 1N-3, 1983.
(13) 後藤, 甘田, 土田; マルチプロセッサにおける内部バスの構成, 同上, 1N-4, 1983.