プログラミング 8－13
（1996. 8. 28）

PPRAM ベース・システム向けプログラミング環境の検討

大澤 拓[†] 岩下茂信[†]
宮嶋浩志[†] 村上和彰[†,††]

本稿ではメモリ-マルチプロセッサ一体型 ASSP である *PPRAM* を構成要素とするコンピュータ・システム向けのプログラミング環境について検討している．*PPRAM* ベース・システムはスケーラブルなシステムであり，その対象は非常に広域であるため，要求されるプログラミング環境も多岐にわたる．本稿では *PPRAM* に適したプログラミング環境の開発方針，今後開発すべき環境について述べる．またその中で特に通信ライブラリを挙げ，*PPRAM* ベース・システムに適した通信ライブラリのインタフェース，アルゴリズムを検討する．*PPRAM* の特徴であるリモート・メモリへの転送命令，等を利用し，効率の良いアルゴリズムを提案している．

Programming Environment for *PPRAM*-Based Systems

Taku Ohsawa,[†] Shigenobu Iwasita,[†] Hiroshi Miyajima[†]
and Kazuaki Murakami[†,††]

PPRAM-based system is a scalable system, and therefore it intends for the broad range of applications. Accordingly, the programming environment is requested to be flexible. This paper discusses the programming environment for *PPRAM*-based systems. It takes up the communication library for *PPRAM* particularly and discusses the interfaces and algorithms. It proposes some efficient communication algorithms for *PPRAM* with using a remote-memory access command, and so on.

1. はじめに

本稿では，メモリ-マルチプロセッサ一体型 ASSP (Application-Specific Standard Product) である *PPRAM* (*Parallel Processing Random Access Memory*) を構成要素とするコンピュータ・システム向けのプログラミング環境について検討する．

PPRAM とは，DRAM-プロセッサ混載 LSI ならびに並列処理時代における新しいコンピュータ・システム構成法のための構成要素であり，

- 大容量メモリ：DRAM をはじめとして，SRAM，Flash EEPROM 等を単独あるいは組合せて構成
- マルチプロセッサ：汎用プロセッサ，特殊用途向けプロセッサ，FPGA 等の 0 個以上のプロセッサ/ロジックから構成されるホモジニアスまたはヘテロジニアスなマルチプロセッサ
- 通信：*PPRAM* チップ内のプロセッサ間，および *PPRAM* チップ間通信を標準の通信プロトコル "*PPRAM-Link*[4]" に準拠して制御

を 1 チップに集積した ASSP である．

この *PPRAM* チップは，1 個以上の *PPRAM* ノードで構成される．*PPRAM* ノードは 0 バイト以上のローカル・メモリ，0 個以上のプロセッサ/ロジック，1 個以上のネットワーク・インタフェースから構成される．この *PPRAM* チップを 1 個以上 *PPRAM-Link* で相互結合したスケーラブルなシステムを *PPRAM* ベース・システムと呼ぶ．*PPRAM* ベース・システムはホモジニアスあるいはヘテロジニアスなシステム構成が可能である．

一般に，並列処理コンピュータの実行性能を向上させ，マシンとユーザとの親和性を高めるには，並列処理のための一貫した言語処理系の開発が重要かつ最大の課題であると考えられる．そこで，本稿では *PPRAM* ベース・システム向けのプログラミング環境について検討する．

PPRAM ベース・システムがその対象とするのは，ハイエンドな科学技術計算用大規模コンピュータから，パーソナル・コンピュータ，組込み機器に至るまで，多岐にわたる．そのため，開発すべきソフトウェア開発環境も非常に多岐にわたる．しかしながら，対象とす

† 九州大学 大学院システム情報科学研究科 情報工学専攻
Department of Computer Science, Kyushu University
ppram@c.csce.kyushu-u.ac.jp
http://kasuga.csce.kyushu-u.ac.jp/~ppram
†† *PPRAM* コンソーシアム設立準備会
PPRAM Consortium: Launch Working Group

べきこれら様々な *PPRAM* ベース・システムも，

- "標準" 通信インタフェース/プロトコル
- メモリ-プロセッサ構成 (分散メモリ型マルチプロセッサ)

という点で共通である．これら共通部分から，可能な限り汎用のプログラミング環境の構築が必要である．

そこで，*PPRAM* ベース・システム向けのプログラミング環境の構築にあたって，

- プログラミング自由度の高い環境
- *PPRAM* ベース・システム間のソフトウェアの可搬性 (portability)
- 既存のプログラミング環境からのシームレスな移行
- スケーラブルなシステムへの対応

をそのキーワードとする．

本稿は，我々が推奨する $PPRAM^{\mathcal{R}}$ の通信機能を中心とした概要 (2 章)，*PPRAM* の目指すプログラム環境の概要と今後開発すべき環境 (3 章)，および，*PPRAM* 向け通信ライブラリの設計 (4 章)，について述べている．

2. $PPRAM^{\mathcal{R}}$ の概要

PPRAM は通信を標準化した ASSP であり，LSI 開発者は対象とするアプリケーションに応じて，"標準" である通信以外の内部アーキテクチャを決めればよい☆．そこで，我々は現在の高性能マイクロプロセッサ相当の *PPRAM* の内部アーキテクチャとして，以下の特徴を有する $PPRAM^{\mathcal{R}}$ (*Reference PPRAM*) を提案している[1),4)]．

- 大容量 DRAM + シンプル・マルチプロセッサ：まずその時代で入手可能な最大容量の DRAM を搭載，次にそれが提供するメモリ・バンド巾に見合っただけの性能のプロセッサを搭載する．
- 分散メモリ型オンチップ・マルチプロセッサ：本質的に非常に高いオンチップ・メモリ・バンド巾を活用するために DRAM を各プロセッサにローカル・メモリとして分散配置し，さらにローカル・メモリに行バッファを複数個設けてこれをキャッシュとして活用する．
- 共有グローバル・レジスタ型オンチップ・マルチプロセッサ：本質的に低いチップ内通信レイテンシを活用するために，共有グローバル・レジスタ・ファイルを設けてチップ内プロセッサ間通信/同期を行う．

$PPRAM^{\mathcal{R}}$ チップの論理構成を図 1に示す．以下，プログラミング環境を左右するメモリおよび通信について概要を述べる．

☆ 例えば，プロセッサを備えない DRAM のみの *PPRAM* や，Flash EEPROM とサーチエンジンを組み合わせた *PPRAM*，あるいは通信のみを備えたブリッジ *PPRAM*，等も存在し得る．

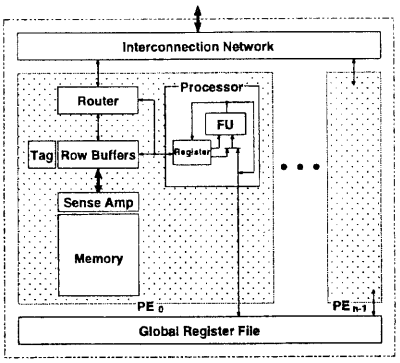

図 1 $PPRAM^{\mathcal{R}}$ の論理構成

2.1 メ モ リ

PPRAM ベース・システムは各 *PPRAM* ノード毎にメモリが分散配置されている分散メモリ型マルチプロセッサであるが，システム全体としては 64 ビット・アドレス長の単一かつグローバルな物理アドレス空間を提供する[5)]．64 ビット・アドレスの上位 32 ビットは各ノードの識別子 (nodeId)，下位 32 ビットはノード内オフセットである．これにより，*PPRAM* ベース・システムは最大で 4G 個のノードを持つことが可能であり，また，各ノードの最大メモリ容量は 4G バイトとなる．

2.2 リモート・メモリ・アクセス操作

ローカルメモリ同様，リモート・メモリに対するロード/ストア命令，および，不可分操作命令 (Fetch&Add, Compare&Swap, 等) を提供する．ただし，これらのリモート・メモリ・アクセス命令はノンコヒーレント (つまり，キャッシュ・コヒーレンスが保証されないメモリアクセス) である．

2.3 PUT/GET インタフェース

DMA 転送手段として，アドレス変換機構で保護された PUT/GET インタフェースをユーザ・プログラムに直接提供する．PUT/GET インタフェースとして，少なくとも以下の 2 命令を定義する．

PUT PUT 命令を実行するプロセッサのローカル・メモリから，PUT 先リモート・メモリに対して転送サイズ分のデータを送る．

GET GET 先ローカル・メモリから，GET 命令を実行するプロセッサのローカル・メモリに転送サイズ分のデータを取り込む．

これらはノンブロッキング命令であり，命令は転送の終了を待たずに完了することができる．転送先/転送元プロセッサは転送の終了をステータス・レジスタを検索することで知ることができる．あるいはオプションとして，割込みによる通知も可能である．他のオプションとして，

表1　グローバル・レジスタに対するアクセス操作

	Empty	Full
TEST&READ	失敗	読出し
TEST&READ&RESET	失敗	読出し&F/E←E
WRITE	書込み&F/E←F	
TEST&WRITE	書込み&F/E←F	失敗
TEST&WRITE&INTERRUPT	書込み&F/E←F&割込み	失敗

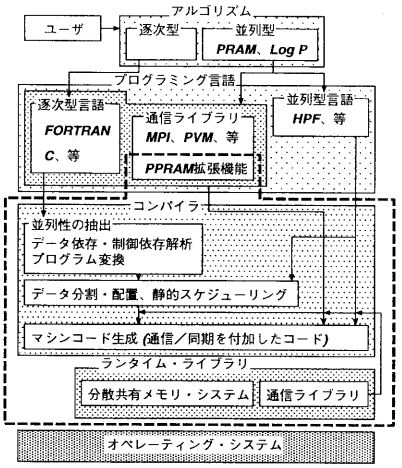

図2　プログラミング環境

- ストライド転送
- キャッシュ・バイパスの有無

を指定できる．

2.4 グローバル・レジスタ・ファイル

チップ内のすべてのプロセッサが論理的に 1 つのグローバル・レジスタ・ファイルを共有し，グローバル・レジスタを介して超低レイテンシ通信/同期が可能である．各グローバル・レジスタはデータ格納用のフィールド以外に，F/E (Full/Empty) ビットを備える．グローバル・レジスタに対するアクセス操作を表1に示す．グローバル・レジスタへのアクセス操作の成功/失敗は，ステータス・レジスタにより知ることができる．

3. プログラミング環境の概要

PPRAM ベース・システムのプログラミング環境の概念図を図2に示す．なお，図の破線で囲まれた部分は *PPRAM* ベース・システムで新たに開発予定の環境を表している．以下，それぞれについて述べる．

3.1 対ユーザ・インタフェース

ユーザに従来の逐次型言語，共有メモリ型並列言語，メッセージ交換型言語を提供する，また，*PPRAM* 向けにメッセージ交換ライブラリを開発する．

3.1.1 既存のプログラミング言語

以下の言語をサポートする．

- 逐次型：C，FORTRAN，等の一般的な手続き型言語
- 並列型
  - 共有メモリ型：HPF，等の並列型言語，および，スレッド・ライブラリ，等
  - メッセージ交換型：PVM，MPI，等のメッセージ交換ライブラリ

3.1.2 *PPRAM* 拡張メッセージ交換ライブラリ

PPRAM ベース・システムは，

- トポロジ
- チップ内のアーキテクチャ
- ノード内のアーキテクチャ

についてインプリメンテーション毎に異なるため，以下の事項について選択の幅がある．

- プロセスのハードウェアへの割り当て：
  - チップ内/間で通信性能/機能が異なる：例えば，ユーザは1チップ内のみで実行することを前提にプログラムを作成することが考えられる．
  - 各ノードはアプリケーション毎に様々なアーキテクチャが存在し得る：特定アーキテクチャを有するノードのみで実行することが考えられる．
- 通信手段：チップ内通信手段として PUT/GET あるいはグローバル・レジスタが選択可能である．
- 通信の条件：上記手段に連動して，通信の条件 (例えば，通信の開始・完了) が異なる場合がある．

上記選択を可能とする機能を持たせるため，既存のメッセージ交換ライブラリを *PPRAM* 向けに拡張する，あるいは新たに開発することを予定している．開発にあたり，プログラミング・インタフェースを構築する上で，システム構成および内部アーキテクチャを如何にユーザに意識させるか (あるいは隠蔽するか)，が問題となる．この問題に対しては，“ユーザに対しプログラミング自由度の高いインタフェースを提供する” ことを念頭に設計を行なう．

3.2 対言語インタフェース

逐次型言語に対しては，逐次型実行，自動並列化を提供し，既存の資産を効果的に利用する．しかしながら “自動並列化” についてはまだまだ実用レベルではなく，また作業量が膨大であるため，まずは逐次型の

コンパイラを移植する．これによって少なくとも，

- 既存の逐次型言語
- 既存の言語+メッセージ交換ライブラリ

のコンパイルを可能とする．

3.3 対コンパイラ・インタフェース

システムの実行モデルとして以下のモデルが挙げられる．

- 分散メモリ型モデル：プログラムは分散型のメモリ空間を前提として実行される．通信ライブラリを用い，各ノード間で通信/同期を行なうことができる．これにより，ノード間でメッセージ交換型並列実行が可能である．
- 共有メモリ型モデル：プログラミング・インタフェースとして，システム全体で単一な仮想アドレス空間を提供する．プロセス間の通信/同期は共有メモリへのロード/ストアで行なうことができる．

3.3.1 ランタイム・ライブラリ

以下のようなランタイム・ライブラリを開発し，実行モデルを実現する．

- 分散共有メモリ・システム[6]：各ノード毎に分散配置した物理メモリをグローバルな仮想アドレス空間にマッピングし，全てのプロセスに単一な共有仮想アドレス空間を提供する．
- 通信ライブラリ：これについては4章で述べる．

3.3.2 オペレーティング・システム (OS)

一般的な UNIX 系 OS の存在を前提にする．

4. 通信ライブラリ

3.3.1節で述べた通信ライブラリについて，インタフェース，機能，およびアルゴリズムについて検討する．ここでは，メッセージ交換型並列プログラムにおいて重要と考えられる Send/Receive，同期について主に検討する．また，プログラミング自由度の高い環境を構築するため，このレベルではなるべく簡素にし，意思の決定はより上位の言語に任せるよう設計する．

なお，プロセスは固有のプロセス番号 (PID：Process ID) を持ち，プロセス毎に独立した仮想アドレス空間を持つこととする．

また，文中の引数は以下の通りとする．

```
int  S_PID          Source PID
int  D_PID          Destination PID
void *S_VA          Source Virtual Address
void *D_VA          Destination Virtual Address
int size            データのサイズ
void *buf           送信バッファの開始アドレス
```

4.1 PUT/GET を用いた通信

PUT/GET インタフェース (2.3節参照) を用いて Send/Receive によりノード間通信を行なう．PUT/GET はメモリ領域の転送命令であるため，Send/Receive もメモリからメモリへの連続した領域の転送命令であると定義する．

以下，用意する命令についてそれぞれ解説し，4.1.2,4.1.3節で実際の通信の流れについて概略し，考察を加えている．

4.1.1 準　　備

送信側はローカル・メモリに送信バッファへのポインタ，バッファの状態を持つメッセージ・ボックス (box) を用意する．

```
struct box{
        int status;
        void *buf_ptr;
}
struct box *box_ptr
```

status は以下の値を持つ．

値	意味
Empty	通信が行なわれていないことを表す．
Ready (D_PID)	Destination に対して送信の用意ができたことを表す．
Complete	Destination に対するデータの転送が終了したことを表す．

4.1.1.1 ノンブロッキング送信

形式

```
int PG_nsend(S_VA, size, D_PID, box_ptr, buf)
```

機能

送信バッファ (*buf) を利用し，ノンブロッキング送信を行なう．S_VA から size 分のデータをプロセス D_PID に対して送信する．

アルゴリズム

(1) status を参照し，Empty でなければエラー値を返す．
(2) buf_ptr を buf にし，*buf にsize 分の送信データをコピーする．
(3) status を Ready にする．

4.1.1.2 受信完了確認

形式

```
int PG_wait(D_PID, box_ptr)
```

機能

ノンブロッキング通信の完了を確認する．

アルゴリズム

(1) status を参照し，Complete なら Empty にして操作を終了する．そうでなければエラー値を返す．

4.1.1.3 ブロッキング送信

形式

```
int PG_send(S_VA, size, D_PID, box_ptr)
```

機能

ブロッキング送信を行なう．S_VA から size 分のデータをプロセスD_PID に対して送信する．

アルゴリズム

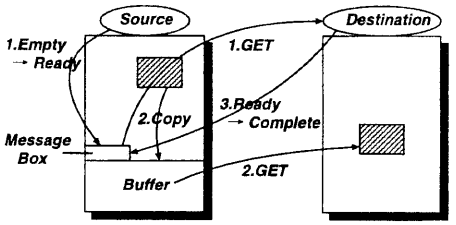

図3　ノンブロッキング送信 – ブロッキング受信

(1) **status** を参照し，Empty でなければエラー値を返す．
(2) **buf_ptr** を **S_VA** にし，**status** を Ready にする．
(3) **status** が Complete になるまでスピンする．
(4) **status** を Empty にし，操作を終了する．

4.1.1.4 ブロッキング受信

形式

```
int PG_recv(D_VA, size, S_PID, box_ptr)
```

機能

ブロッキング受信を行なう．プロセス **S_PID** から送信された **size** 分のデータを **D_VA** に格納する．

アルゴリズム

(1) リモート・ロード命令により送信側の **box** をコピー，**status** を参照し，Ready でなければエラー値を返す．
(2) GET 命令により，***buf_ptr** から **D_VA** に **size** 分のデータを取り込む．
(3) PUT 命令により送信側の **status** を Complete にする．

4.1.2 ノンブロッキング送信 – ブロッキング受信

PG_nsend , **PG_recv** , **PG_wait** を用いてノンブロッキング送信 – ブロッキング受信を行なう (図3参照)．バッファは送信側のメモリにアドレス，サイズ固定の送信専用バッファとして用意される．これは，送信毎に確保/解放手続きを行なう必要はない．このバッファとメッセージ・ボックスを用いて，以下のようなアルゴリズムにより通信を行なう．また，メッセージ・ボックスとバッファの組を複数個設け，複数プロセスへの送信を重複させることも可能である．

(1) Source は**PG_nsend** によりバッファに送信データをコピーする．
(2) メッセージ・ボックスを Ready にすることでバッファを Destination のみに解放し，操作から戻る．
(3) Destination は**PG_recv** によりメッセージ・ボックスを監視し，送信の用意ができたことを知ると，GET 命令によりバッファからデータを取り込む．
(4) PUT 命令によりメッセージ・ボックスを Complete にし，操作から戻る．
(5) Source は **PG_wait** によりメッセージ・ボックスを監視し，送信の終了を確認する．

4.1.3 ブロッキング送信 – ブロッキング受信

GET 命令はリモート・メモリからローカル・メモリへ直接データを取り込むことができるため，バッファを用いない通信も可能である．ただし，送信データの内容を保証するため，送信側もブロックする必要がある．ここでは**buf** は送信データの開始アドレス (**S_VA**) を表すことになる．

4.2 PUT/GET を用いた同期

Butterfly 型にメッセージ交換を行なうことにより，複数プロセス間のバリア同期を実現する．メッセージは同期を行なう全てのプロセスの特定アドレスに確保された固定長の同期カウンタに対し，PUT/GET 操作を行なうことによって実現する．同期カウンタは同期を行なうプロセスのグループ毎に全てのプロセスのローカル・メモリに用意する．すなわち同期カウンタは，あるグループ専用であり，そのグループ以外からのアクセスは起こり得ない．そのため，ある値の同期カウンタへのアクセスは，その値に対応したある1つのプロセスからしか起こり得ず，排他制御を必要としない．

同期カウンタは以下の値を持つ．

値	意味
Empty	同期操作は行なわれていない．
Active (Count)	メッセージが到着した回数を表す．

形式

```
int PG_barrier(comm)
Comm_group comm
```

機能

プロセスのグループ (**comm**) 間でバリア同期を実現する．グループ中の全てのプロセスが **PG_barrier** を呼び出すまで戻らない．

アルゴリズム

(1) $x = 0$ とする．
(2) リモート・ロード命令により，プロセス $P_x^{\dagger}$ の同期カウンタを参照し，n になるまでスピンする．
(3) PUT 命令により，プロセス P_x の同期カウンタに $x+1$ を書き込む．
(4) プロセス A 中にある同期カウンタを参照し，値が $x+1$ になるまでスピンする．
(5) x を1プラスし，$\lfloor \log n \rfloor$ になるまで (2)～(4) を繰り返す．

†：Butterfly 型の通信で x 番目にメッセージを送信する対象

4.3 グローバル・レジスタを用いた通信/同期

PPRAM$^{\mathcal{R}}$ 独自の機能として，グローバル・レジスタ・ファイル (2.4節参照) の存在がある．*PPRAM*$^{\mathcal{R}}$

ではチップ内において，これを以下のような用途に用いることができる．

- 少量のデータの通信路
- 通信用のチャネル
- 共有変数 (セマフォ，バリア変数，共有変数，等)

集団通信を行なう場合，通信/同期を行なうべきノードがチップ内/外に跨っている場合が考えられるが，これについてはより高いレベルのライブラリで処理することにする．

4.3.1 少量のデータの通信

形式

```
int GR_send(S_VA, size, D_PID)
int GR_recv(D_VA, size, S_PID)
```

機能，アルゴリズム

グローバル・レジスタを共有メモリと見做し，Read/Write によって通信を行なう．このとき，F/E ビットによって排他制御，生産者 – 消費者同期を保証する．

4.3.2 通信用のチャネル

形式

```
int GR_csend(S_VA, size, D_PID, option)
int GR_crecv(D_VA, size, S_PID)
```

機能，アルゴリズム

グローバル・レジスタを通信用のチャネルとして用いることにより，ノード間通信を行なう．また，option によってはアクティブ・メッセージの送信も可能である．

4.3.3 共有変数としての利用

形式

```
int GR_renew(GR_number, value)
int GR_read(GR_number, value)
void value
```

機能

- GR_number で表されるグローバル・レジスタを value 値に更新する．
- GR_number で表されるグローバル・レジスタの値を value に読み込む．

5. おわりに

PPRAM 共通のプログラミング環境について検討し，今後開発する環境として通信ライブラリの設計について述べた．

今後の予定として，現在開発中の $PPRAM^{\mathcal{R}}_{mf}$ [☆2)] について，以下を行なう．

- $PPRAM^{\mathcal{R}}_{mf}$ 用通信ライブラリの開発
- $PPRAM^{\mathcal{R}}_{mf}$ 用コンパイラの開発

  1ノードの $PPRAM^{\mathcal{R}}_{mf}$ 用コンパイラ．現在，GCC (GNU C Compiler) のマシン依存部分を書き換え，$PPRAM^{\mathcal{R}}_{mf}$ に移植中である．
- $PPRAM^{\mathcal{R}}_{mf}$ シミュレータによる性能評価
  - アーキテクチャおよびハードウェア構成の違いによる性能差を測定し，$PPRAM^{\mathcal{R}}_{mf}$ の設計にフィードバックする．
  - ベンチマークプログラムによる $PPRAM^{\mathcal{R}}_{mf}$ の絶対性能の評価．
- $PPRAM^{\mathcal{R}}_{mf}$ 用に拡張したメッセージ交換ライブラリの開発

謝辞 日頃から御討論頂く，九州大学 大学院システム情報科学研究科 安浦寛人教授，岩井原瑞穂 助教授，*PPRAM* プロジェクト・メンバ，安浦・村上・岩井原研究室の諸氏，ならびに，*PPRAM* コンソーシアム設立準備会の会員諸氏に感謝致します．

参考文献

1) 岩下茂信，宮嶋浩志，村上和彰，“次々世代汎用マイクロプロセッサ・アーキテクチャ*PPRAM* の概要，” 情処研報，ARC-113-1, 1995 年 8 月．
2) 岩下茂信，宮嶋浩志，村上和彰，“リファレンス *PPRAM*「$PPRAM^{\mathcal{R}}$」に基づく『$PPRAM^{\mathcal{R}}_{mf}$』アーキテクチャの概要，” 情処研報，ARC-119-28, 1996 年 8 月．
3) 笠原博徳，並列処理技術，コロナ社．
4) 村上和彰，岩下茂信，宮嶋浩志，白川 暁，吉井 卓，“メモリ-マルチプロセッサ一体型 ASSP (Application-Specific Standard Product) アーキテクチャ:*PPRAM* ，” 信学技報，ICD96-13, CPSY96-13, FTS96-13, 1996 年 4 月．
5) 村上和彰，岩下茂信，宮嶋浩志，“メモリ-マルチプロセッサ一体型 ASSP「*PPRAM* 」用標準通信インタフェース『*PPRAM-Link* Standard』Draft 0.0 の概要，” 情処研報，ARC-119-27, 1996 年 8 月．
6) 村上和彰，吉井 卓，岩下茂信，宮嶋浩志，“*PPRAM* ベース・システム向け分散共有メモリ・システムの提案，” 情処研報，OS-73-2, 1996 年 8 月．
7) Dinning, A., “A Survey of Synchronization Methods for Parallel Computers，” *Computer*, Vol. 22, No 7, pp.66–77, Jul. 1989.

☆ $PPRAM^{\mathcal{R}}$ として備えるべき最低限の機能 (mf: minimal functionality) のみを備えたアーキテクチャ．$PPRAM^{\mathcal{R}}$ のアーキテクチャ上の枠組に基づいた，プロトタイプ・アーキテクチャおよび評価ベースラインとして開発中．