計算機アーキテクチャ 79－12
（１９８９．１１．２１）

# 並列CADマシンCenjuのソフトウェア体系

松下 智　　中田登志之　　梶原信樹
田辺記生†　　浅野由裕‡　　小池誠彦

日本電気(株) C&Cシステム研究所,　　†日本電気(株) 超LSI CAD技術本部
‡日本電気技術情報システム開発(株)

Cenjuは回路シミュレーションに特化して設計された64台構成のMIMD型並列マシンである。Cenjuではシミュレーションアルゴリズムが粗粒度で通信が規則的である特性を利用し、ハードウェアを単純化した。一方、不足した機能を補うためシステムソフトウェアの機能は肥大化したが、ソフトウェアの階層化によって対応した。本報告ではCenjuのアーキテクチャ、システムソフトウェア系について、総括的に設計思想を示す。特に、ソフトウェア的に構築した通信機構について示す。次に、高級言語のライブラリとして提供したユーザ環境について簡単にふれ、最後に通信性能と回路シミュレーションの性能を示す。

# DESIGN OF SOFTWARE SYSTEM ON THE PARALLEL CAD MACHINE "CENJU"

Satoshi MATSUSHITA　Toshiyuki NAKATA　Nobuki KAJIHARA
Norio TANABE †　Yoshihiro ASANO ‡　Nobuhiko KOIKE

C&C Systems Research Laboratories, NEC Corporation
† VLSI CAD Engineering Division, NEC Corporation
† NEC Scientific Information System Development Ltd.

1-1 Miyazaki 4-chome, Miyamae-Ku, Kawasaki, Kanagawa 213, Japan
† 1753 Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211, Japan
‡ KSP R&D Building, 100 Sakato, Takatsu-Ku, Kawasaki, Kanagawa 213, Japan

Cenju is a dedicated parallel machine for circuit simulation, which employs 64 processors. The characteristic of the simulation algorithm, ie. coarse grain, low communication traffic and regularity of communication pattern, made it possible to reduce the hardware. But this simple hardware forced the system software to be complex. We realize many system functions by using layered system monitor. In this presentation, we discuss Cenju software system as a whole. Then we consider user programming view supplied as a library for high level language. Finally we evaluate Cenju's performance on some of its applications.

## 1 はじめに

我々は並列回路シミュレーションマシンとして、Cenju [1] [2] [3] を開発してきた。本マシンはバス結合されたクラスタ、およびクラスタ間ネットワークをもつ MIMD 型並列マシンであり、最大プロセッサ数は 150 台程度となる。現状では、64 台構成の Cenju および、8 台構成の mini-cenju が稼働している。Cenju の相互結合網の構築にあたっては並列回路シミュレーションが効率的に実行できる範囲内で必要最低限のハードウェア量となるよう配慮し、システム性能に影響を与えない部分は極力ソフトウェア的に実現するようにした。

ソフトウェアの設計に当たっては、回路シミュレーションに特化したアーキテクチャを極力隠蔽し、シミュレーション向け汎用並列マシンとして実用プログラムの開発、検証ができる環境をめざした。このため、幾つかの並列プリミティブを選択し、ライブラリとして高級言語に組み込んだ。また、これらライブラリの実行を支援する機能を Cenju モニタに組み込んだ。

本稿ではアプリケーションの特性、アーキテクチャの順に述べ、その後、Cenju のシステムソフトウェアを中心に述べる。

## 2 Cenju のアプリケーションの特性

Cenju の主たるアプリケーションはモジュール分割法によって並列化された、並列回路シミュレーションである。

Cenju ではエネルギー蓄積素子を含んだ非線形回路素子からなる回路シミュレーションを対象にしている。回路シミュレーションの並列化法は大別して、直接法、緩和法、モジュール分割法があるが、我々は粒度が細かい直接法、精度 / 収束性の保証がない緩和法を避け、モジュール分割法にもとづく並列アルゴリズムを提案した。[1]

本アルゴリズムの概略を 図1 に示す。これは、大雑把に言って以下に示す 2 つのフェーズの繰り返し処理となる。

1. 非線形な部分回路の線形化 (子プロセスで並列処理可能)
2. 線形方程式の求解による境界条件の更新 (親プロセスで逐次処理)

ここで、第1の処理において部分回路 $S_1$ から $S_n$ を別々なプロセッサに配分すれば、全く通信を引き起こすことなく並列処理が可能である。[2]

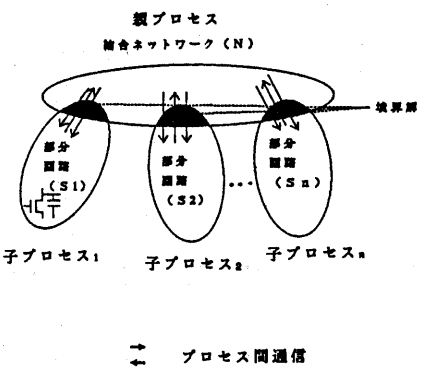


図 1: モジュール分割法

以下に、本アルゴリズムの特色を挙げる。

1. 処理の流れが統合 - 分割 (Divide-and-Conquer) 型である
2. 通信量は境界変数に関するものだけでよく、比較的小さく押えることが 可能である
3. 逐次部分は部分回路分割数に依存するが 1 から 4% 程度である
4. 負荷分散は部分回路の分割でほぼ静的に決まる
5. 通信は親プロセスと子プロセスの間でしか生じない
6. 通信の大半は、通信すべきデータ量と送り先領域を静的に 決定することができ、通信はデータ生成側からデータ参照側への一方的な書き込みとして 記述可能である (producer-consumer 型の通信)

## 3 Cenju アーキテクチャ

**アーキテクチャ決定の要因** (1) 前述の部分回路分割アルゴリズムの特性第6項から、通信はバッファへの転送を伴うメッセージとしてよりも、共有メモリへの write アクセスとして記述した方が効率がよいことが分かる。(2) また、第 2, 5項により、ネットワークの通信バスへの要求は高くない。(3) 一般に、逐次部分の比率を S とした場合、それ以外の部分を N 並列に処理することで、

$$\frac{1}{S + \frac{1-S}{N}}$$

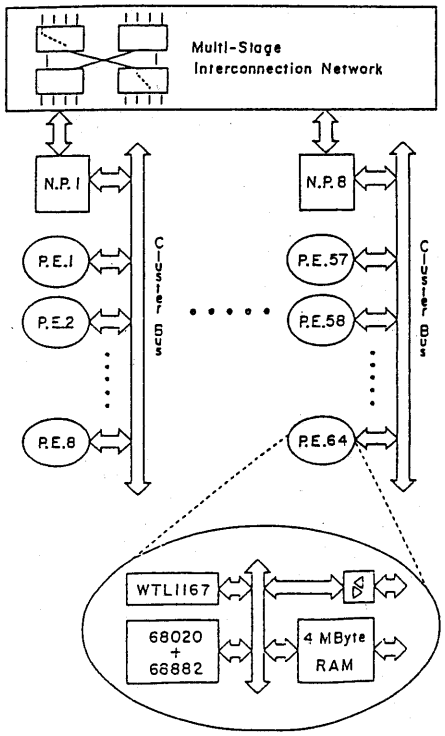


図 2: Cenju アーキテクチャ

の並列度が得られる。(Amdahl の法則)

したがって、通信オーバヘッドを低く抑えれば、第4、第3項により 100 プロセッサで 20 から 50 の並列度が得られる。よって、プロセッサ 100 台程度の構成をとれる必要がある。

上記 (1) から (3) を踏まえ、Cenju の基本アーキテクチャを以下の様に決定した。

1. バス・ネットワークからなるクラスタ構造
2. 分散共有メモリ参照型の通信
3. クラスタ間通信は write アクセスを強化する

図 2 に Cenju アーキテクチャを示す。[3]

### 3.1 プロセッサエレメント

プロセッサエレメント (以下 PE) は、MC68020 (20Hz)、MC68882 (20MHz)、および、四則浮動少数演算のアクセラレータとして WTL1167 (20MHz) から構成され、ピーク性能 1.5MFlops、Linpack 性能 0.5MFlops を有する。各 PE は 4MB のローカルメモリを持つ。分散共有メモリにするためローカルメモリを 2 ポート化し 1 方のポートをバスに接続した。

| Processor Number | Global Address |
|---|---|
| 0 | 0x80000000-0x803fffff |
| 1 | 0x80800000-0x80bfffff |
| 2 | 0x81000000-0x813fffff |
| ... | ... |
| 7 | 0x83800000-0x83bfffff |
| 8 | 0x88000000-0x883fffff |
| ... | ... |
| 56 | 0xb8000000-0xb83fffff |
| ... | ... |
| 63 | 0xbb800000-0xbb3fffff |

図 3: Cenju メモリマップ

Cenju では 4GB のアドレス空間をローカル空間・プロセッサ間割り込み空間・グローバルメモリ空間 (後 2 者を合わせてグローバル空間と呼ぶ) に分割する。グローバルメモリ空間はアドレス空間の上位 2GB を占有する。図 3 に示すように、メモリの 1 方のポートをローカル空間の 0 ～ 0x3fffff にマップし、他方をグローバル空間を 256 等分した 8MB ずつのウインドウの 1 つにマップする。

プロセッサ間割り込み空間には、PE ごとに異なるプロセッサ間割り込みアドレスが割り付けられ、このアドレスへのアクセスにより PE への割り込みが引き起こされる。

### 3.2 クラスタ内通信

クラスタはバスで結合される。このため、クラスタ内の通信手段はバス経由で行なわれるメモリアクセス (read, write, read modify write), プロセッサ間割り込みだけとなる。

### 3.3 クラスタ間通信

クラスタ間の通信手段として、クラスタ間のメモリ write をハードウェアでサポートする。これは、クラスタバスに付けられたハードウェア NADA (Network ADApter) によって行なわれる。

ネットワークは、多段スイッチを用いたパケット交換網である。NADA によってクラスタ間 write 要求パケットとバス上のクラスタ間 write アクセス要求の変換が行なわれる。クラスタ間 write アクセスを、NADA およびネットワーク内でパイプライン化することにより、スループットの向上をはかっている。

ただし、クラスタ間の write 機構だけでは、動的に変化するデータを扱えないうえ、排他制御を実現できないため、NADA 上にパケット書き込み用のポートを付加した、さらに、このポートを

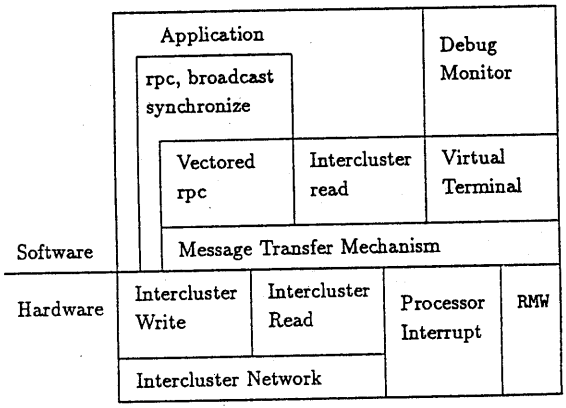


図 4: ソフトウェア階層

管理する専用 PE として NWP (NetWork Processor) をおいた。

クラスタ間のアドレスに対して read 要求、read modify write 要求があった場合、NADA は、そのアクセスに対しバスエラーを発行する。以降の処理は NWP と PE 上のソフトウェアを用いて実現することができる。

### 3.4 ホストとの通信

Cenju には、sun, ews4800 などのワークステーションがホストとして接続される。

ホストとの通信は、クラスタ 0 のバスに接続された DMA コントローラを用いてなされる。

## 4 システムソフトウェア

Cenju のシステムソフトウェアの構成を図 4 に示す。Cenju ではシステムソフトウェアの核となる常駐モニタによって、汎用な通信イメージの提供、アプリケーションの実行支援機能、その他自己診断・ブート・デバッグ支援などの機能が提供される。

### 4.1 仮想化した通信イメージの提供

Cenju ではソフトウェアのサポートがない場合、クラスタ間 PE の通信手段はメモリ write しか提供されない。

そこで、Cenju では任意の PE 間で、

(1) メッセージの通信
(2) 分散共有メモリアクセス
(3) 遠隔手続き呼びだし (以降、RPC:Remote Procedure Call)

をソフトウェア的に提供する。ここでは、これらをソフトウェア的に構築した”仮想的な通信機構”と呼ぶことにする。

#### 4.1.1 仮想的な通信機構の役割

以下、仮想的な通信機構の役割を考え、その位置付けを明確にする。

(1) Software Transparency の提供

共有メモリイメージで並列化を行なった場合、PE 間のデータコピーのためのコードを挿入する必要がなく、逐次版に対するアルゴリズムの変更量は小さい。したがって、 read をソフトウェア的に提供し、疑似的な共有メモリ環境を提供しておくことで、クラスタ間 write のみを提供した場合に比べ並列化作業は簡単になる。こうしておいて、Cenju で実行しオーバヘッドの大きい部分だけ書き代えることで並列化に際する作業量の削減がはかれる。

一般にクラスタ間の read アクセスはソフトウェア的に提供されるので遅く、頻繁に生起した場合オーバヘッドになり得る。このため、アルゴリズムの変更によりクラスタ間の write アクセスに変換する必要がある。しかし、アクセス箇所が動的に変化するなどの理由で write に変換できない read が存在する。これは、RPC を用いることで、データを所持している側から read 要求側のデータ領域への write アクセスに変換できる。これを図 5に示す。read アクセスがバースト的に生じれば RPC のオーバヘッドは無視できる。

(2) システム管理機能の提供

プログラムの起動、アボートなど、制御を伝搬しシステムを管理する機構が必要である。プログラム起動時のコードの各 PE へのブロードキャストも単純な PE 間 write の繰り返しとして記述せず、各 PE に制御を移動させながらツリー状に行ないオーバヘッドを最小化する。これには、PE 間での制御の移動が素直に記述できる RPC が適している。

(3) 稼働率の向上

仮想的な通信機構の上にシステムソフトウェア、および、アプリケーションを構築することで、障害時の PE の切り離しや、さまざまなシステムコンフィギュレーションへの対応が容易になる。

(4) 排他制御機構の提供

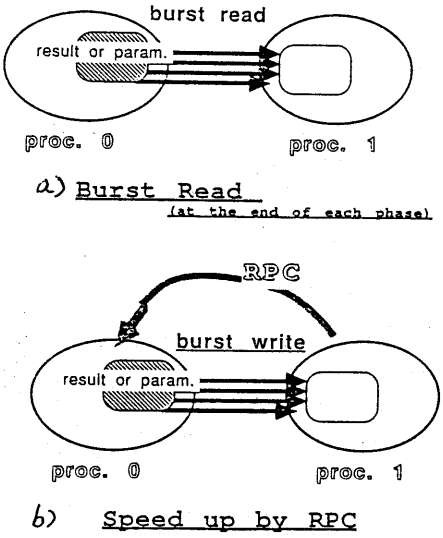


図 5: RPC によるクラスタ間 read の除去

クラスタ間 write は排他性が保証されない。クラスタをまたいだ場合 NADA のパケットポートを利用しない限り排他処理が記述できない。Cenju では、次節に述べる様に遠隔呼びだしとしてこれを実現する。

### 4.1.2 排他制御機構の設計

Cenju では、RPC を、システム機能の提供およびアプリケーション記述のための、排他制御の標準的な枠組として使用する。

すなわち、RPC の処理中は他の要求は実行されず実行待ちキューに入る。RPC には、abort 処理の記述のため、緊急と通常の 2 つの優先度を設けた。

以下、Cenju で RPC を排他制御機構として採用した理由を示す。

## 4.2 排他制御機構の比較

一般の排他制御機構としては以下の様なものが存在する。

**アトミックアクセス** クラスタ外へのアトミックアクセス要求を trap ハンドラでエミュレートすることにより、MC68020 の CAS(compare and swap) 命令, TAS(test and set) 命令 [4] を用いれば、マシン語レベルで排他制御の記述が可能になり、完全な共有メモリマシンをエミュレートできる。しかし、アトミックアクセスは低水準で排他制御の記述が複雑であるうえに、フラグのポーリングの度に packet がネットワークを流れることになり効率が極めて悪くソフトウェア的に実現するのは現実的でない。

**セマフォ** [5] アトミックアクセスに比べネットワークをポーリングのためのパケットが流れず通信効率は良くなる。しかし、P 操作と V 操作が分離しているため、依然記述性が低くデッドロックを招き易い問題がある。

また、第5節にも示すが Cenju 常駐モニタでは性能の向上と OS の単純化のために、システムのレベルでスケジューリングを行なわず、スケジューラの記述はアプリケーションプログラムに委ねることにした。このため、システムレベルでセマフォを実現してもプロセス管理による CPU 資源の有効利用というメリットはない。

**モニタ** [6] モニタは相互排除されるデータとデータ操作手続きをまとめて記述したものであり、前 2 者に比較して抽象度が高く記述性もよい。モニタ手続きの実行を排他的に行なうことで、排他制御を実現する。一般には、Concurrent Pascal のように同期のための条件変数を組み合わせている。プロセス管理もこの枠組内にあり、手続きの起動と条件同期の機構のための 2 本のプロセスキューを含んだインプリメントになる。

Cenju 常駐モニタではスケジューラの記述はアプリケーションプログラムに委ねることにしたため、条件変数による同期をモニタの枠内にいれることは不自然であり、同期機構は別に用意する。

**RPC: Remote Procedure Call** Sun ワークステーション [7], Mach の Matchmaker [8] などメッセージ指向分散環境で、メッセージ機構の上に構築され分散アプリケーション記述のため広く用いられている。

クライアントプロセスがサーバプロセスのサービスを呼び出し結果を受け取る枠組にあり、サーバプロセスの起動方式は割り込み起動に近く Cenju とのマッチングが良い。また、逐次処理に書きなれたユーザにとって、手続き呼びだしが排他的であるのは、極めて馴染み易い。このため、Cenju での基本排他機構として採用した。

**message** 送信点、受信点を考慮して書かなければならず、イベント駆動的に書ける遠隔手続き呼

び出しに比べ安全性は高いが記述は複雑になる。ただし、通信が一方方向であるため、遠隔 write とバッファを組み合わせパッケージ化することにより、排他制御機構としてではなく、同期手段およびデータ通信のための高水準なプリミティブとして採用する。

**まとめ** ゆえに、Cenju においては RPC を排他制御の基本的な枠組として位置付ける。

#### 4.2.1 全体同期のためのプリミティブ

Divide-and-Conquer 型のアルゴリズムではすべての PE の間で同期が必要となる。これに対して、Cenju では、barrier を導入する。barrier では、全てのプロセスが barrier を発行するまで block することにより同期を実現する。以下に、他の同期機構との比較を簡単に示す。

**非ブロック型 RPC** 全ての PE に非ブロック型の RPC で処理を依頼し、全ての処理を終了通知を待つ。Divide-and-Conquer アルゴリズムでは、永続プロセスの間の同期として考えた方が自然であるし、1 つの PE が全ての PE に RPC を発行するため、PE 数が増えた場合コストが増大する。

ただし、非ブロック型 RPC は非同期的な処理の起動で有効なため、Cenju でサポートする。

**signal-wait** 一般に、1 対 1 の待ち合わせを示すが、ある範囲のプロセスを待ち合わせるように、拡張することができる。

wait を行なうプロセスと signal を発行するその他のプロセスとの非対称性が生じるうえ、同期の成立を signal をかけたプロセスに逆に伝達する必要があり、記述量も多く、同期点が不明確になりやすい。

**Barrier** 対称性が良く記述力もよく、内部的には、signal-wait の処理に伴うフラグハンドリングを行なうだけでよいため、インプリメントも比較的容易である。また、基本的に write アクセスのみを用いインプリメントできることから、基本的な同期機構として採用した。

**まとめ** このように、Cenju では全体同期のための機構として Barrier を採用した。また、ブロック型・非ブロック型 RPC でも同期を記述することができ、ユーザに対し 2 つの独立した枠組を提供していることになる。

### 4.3 ユーザプログラミング環境の提供

システムソフトウェアでは、これまでに述べたプロセッサ間通信機能の上に、ホストを経由したユーザとの対話環境、ファイルの入出力機能、デバッグのためのモニタ機能を提供する。

#### 4.3.1 その他の機能

アプリケーションのロード、実行管理の機構、自己診断、システムコンフィギュレーションの自動判定機能、障害の通知機能、デバッグ用のコマンドを実行するデバッグモニタとしての機能が挙げられる。

## 5 常駐モニタのインプリメンテーション

Cenju は専用アクセラレータとしてシングルタスクマシンとして設計され、MMU を搭載していない。このため、資源管理や、コンテクストの管理機構はアプリケーション側で記述することにし、これらを高級言語レベルで移植性よく記述するための機能をサポートするに留めた。

常駐モニタによるシステム機能のサービスは、アプリケーションから呼ばれるサブルーチン、もしくは、アプリケーションルーチンを呼び出す割り込みハンドラとして実現される。

これまで示した様に、システムソフトウェアに要求される機能は、比較的高機能で数も多く、信頼性への要求が強いが、アプリケーション性能への影響は少ない。そこで、階層化を行なった。概念図を 図4 に示してある。

**第 1 層 メッセージ** ソフトウェア最下層である。

**第 2 層** 第 2 層では、メッセージを用い以下の 3 種類の基本機能を実現する。

- **文字入出力** アプリケーションの文字入出力、デバッグ用モニタのコマンド解釈ルーチンへの文字入力、PE でのエラー情報の出力などの文字ストリームを扱う。
- **クラスタ外 read** クラスタ外 memory read で生起するバスエラーを受けて、アクセスアドレスを含む memory read 要求メッセージを生成送信する。memory read 要求メッセー

ジに対しては該当クラスタのNWPが処理を代行する。

- **VRPC: Vectored Remote Procedure Call**
  システムレベルで用いる排他性のあるRPCである。引数をとりリターン値を返す。2つの優先度があり、高い優先度のものによりabortを記述する。手続きの指定はベクタ番号で行ない番号の範囲で優先度、引数の形態を分類している。

**第3層** 常駐モニタとしてのサービスを記述するサブルーチン群である。

## 6 ユーザプログラミング環境

Cenjuでは高級言語でのプログラミングを支援するため、通信、同期、排他制御を実現する手続き群(ライブラリルーチン)を作成した。ライブラリルーチンには、並列記述プリミティブとホストとの間でのファイル/文字ストリームの入出力、タイマ管理などがある。

これによって、現在Cenjuでは、C言語、Fortranを用いてプログラミングすることが可能である。

Cenjuでは仮想的に共有メモリ環境が構築されているため、グローバルアドレスをアプリケーションから直接アクセスする通信を行なうことも可能であるが、プリミティブを用いることで以下に示す利点が得られるため、アプリケーションにおいて同期、排他制御は、ほとんどプリミティブによって記述されるようになってきた。

**Cenju 並列プリミティブの利点**

- 高水準化
  - クリティカルセクションの見落としを減らし、非同期事象 生起によるバグの混入を防ぐ
  - アーキテクチャへの依存性を減らし、後継機に対するソースの互換性を上げる
- 高速化
  - 並列プリミティブではハードウェア構造を意識した内部記述を行う。このため、ユーザがハードウェアを意識せず記述するより高い性能が期待できる

しかし、データ転送においては転送のパターンが応用毎に異なるため統一的な枠組を設定するのは難しく、現在一部のアプリケーションの記述に用いられているにすぎない。現在、アプリケーションからのフィードバックを受けて、ライブラリの整備を継続中である。

表1: Cenju 並列記述プリミティブの一部

| |
|---|
| **CJForki** 並列実行の開始、Mach,Dynixのforkと同様に、コンテクストをCenju内のプロセッサにブロードキャストし、並列実行を開始する |
| **CJSysBarrier** 全てのプロセッサ間で同期をとる |
| **CJBarrier** 設定したプロセッサグループで同期をとる |
| **CJRpc** 他PEの手続きを呼び出す、呼びだされたルーチンの終了まで呼びだし側はブロックする |
| **CJNbrpc** 他PEの手続きをポインタで指定し呼び出す。呼びだし側はブロックしない |
| **CJSend, CJRecv** 領域(配列)の転送。転送が終了するまで双方はブロックする |
| **CJASend, CJARecv** ブロックしない転送 |

```
      DO 121 JL=1,LMCAL
         DO 120 JR=NRL1,NRU1
            VTX2(JR,JL)=VTX1(JR,JL)+WW(JR,JL)*ZDT
120      CONTINUE
121   CONTINUE
      ZDT=0.5D0*DT
C 最左プロセッサのみでの処理
      IF (ILPID.LT.0) THEN
C 状況のレポート
         call cjfprintf(3,"%P leftmost\n")
         DO 130 JL=0,LMCAL
            IF(MM(JL).NE.0)                          GO TO 130
            VTX2(0 ,JL)=
                -BBC/AAC*VTX2(1 ,JL) - CCC/AAC*VTX2(2,JL)
130      CONTINUE
      ENDIF
C
C VTXを近隣のプロセッサに配布 (各JLについて配布する必要あり。)
C CJASendDbl(送り先、配列の先頭、要素の個数、要素間の飛び幅)
      CALL CJASendDbl(ILPID, VTX2(NRL,0), LD+1, ND+1)
      CALL CJASendDbl(IUPID, VTX2(NRU,0), LD+1, ND+1)

C CJSysBarrier 全プロセッサがこのディレクティブを発行するまで停止する。
      CALL CJSysBarrier

C CJARecvDbl(送り元、受け配列の先頭、要素の個数、要素間の飛び幅)
      CALL CJARecvDbl(ILPID, VTX2(NRL-1,0), LD+1, ND+1)
      CALL CJARecvDbl(IUPID, VTX2(NRU+1,0), LD+1, ND+1)
```

図6: 並列プリミティブを用いた記述例

表1に並列記述プリミティブの一部を示す。

さらにFortranによる偏微分方程式求解の一部を記述例として図6に示す。

## 7 実装および評価

### 7.1 システムソフトウェアの評価

モニタプログラムのコードサイズを表2に示す。エラー回復機能、デバッグのためのモニタ機能を

表 2: ソース量

| 分類 | 記述言語 | ライン数 |
|---|---|---|
| カーネル本体 | C, アセンブラ | 12000 |
| ホストコマンド | C | 20000 |
| ライブラリルーチン | C | 8000 |

表 3: 平均メモリアクセスタイム

| Access Kind | | Access Time |
|---|---|---|
| Local Access | | 250 nsec |
| Intra-Cluster Global Access | | 750nsec |
| Inter-Cluster | (Access) | 5.2 $\mu$sec |
| Global Write Access | (Throughput) | 800 nsec |
| Inter-Cluster Read Access | | 240 $\mu$sec |

含むためプログラムは大きなものとなった。

メモリアクセス性能を 表 3 に示す。このように、クラスタ間 read の性能は他に比べ大幅に悪い。これは、バスエラーハンドラの遅延、ハングアップ防止のタイムアウト機構、NWP を経由するオーバヘッドなどが原因である。

RPC の latency を実測すると、1 回の呼びだし結果を得るまでに、$640\mu S$ となる。これから、N 回個のクラスタ間 read を 1 回の RPC と N 回のクラスタ間 write に変更したとすれば、

$$\frac{85.7}{1+228.6/N} \approx \frac{N}{2.68}$$

倍の速度向上が得られ、N=3 以上ではこの書き換えが有効である。

## 8 アプリケーションを実行した場合の性能

現在、並列回路シミュレータをはじめ、故障シミュレータ、機能レベルシミュレータ、など CAD 分野をはじめ、科学技術計算分野、ニューラルネットワークシミュレーションなど実用規模の並列アプリケーションの実装、評価が進められている。

並列回路シミュレーション　部分回路分割による並列回路シミュレーションの実行性能を 表 4 に示す。速度向上率低下は逐次部分によるもので、これはデータの特性に依存する。

1 台の時に比較し現段階で 16 倍程度の速度向上を得ている。RPC による、クラスタ間 read の書き換えを行ない、通信オーバヘッドについてはほ

表 4: 回路シミュレーションの実行時間と相対速度

| Numberof PEs | Circuit-1 | Circuit-2 |
|---|---|---|
| 1 | 13 hr. 44min (1.0) | ... |
| 2 | 7 hr. 36min (1.81) | ... |
| 4 | 4 hr. 07min (3.33) | ... |
| 8 | 2 hr. 19min (5.93) | 10hr 35min (6.25) |
| 16 | 1 hr. 26min (9.58) | 6hr 35min (9.94) |
| 32 | 1 hr. 26min (10.56) | 4hr 47min (13.82) |
| 64 | 0 hr. 56min (14.7) | 4hr 47min (15.81) |

とんど解消しており、オーバヘッドの大部分は、逐次処理にある。今後は逐次部分の高速化などによるチューニングを行なっていく予定である。

## 9 まとめ

Cenju 上では、 並列プログラミング環境が一応整備され、現在各種アプリケーションの並列化を検討中であり、比較的広いアプリケーションに対して Cenju が適用可能であることが実証されつつある。

今後、Cenju 上にスケジューリング機能をもち、より記述性の高い並列言語を構築していきたい。

## 参考文献


[1] 小池, 中田, 田辺, 小野塚, 黒部 : 並列回路シミュレーションマシンのプロトタイプ, 信学技報, CPSY-87-19, (1987)

[2] Nakata, Tanabe, Onozuka, Kurobe, Koike: *A Multiprocessor System for Modular Circuit Simulation,* Proc. ICCAD '87, pp.364-367, (1987)

[3] 松下, 中田, 梶原, 浅野, 小池 : 並列シミュレーションマシン, 情処 37 回全国大会予稿集, (1988)

[4] *MC68020 32-Bit Microprocessor User's Manual* 2nd Edition, Prentice Hall

[5] Dijkstra E.W.: *Co-operating sequential process, in Programming Language,*

[6] Hoare C.A.R.: *Monitor; an operating system structuring concept,* Comm.ACM, Vol.17,No.10,(1974)

[7] *Sun OS 4.0 Programmer's Guids; Network Programmings,* Sun Microsystems

[8] Michael B.Jones, Richard F.Rashid: *Mach and Machmaker;Object-Oriented Distributed Systems,* CMU Techinical Report, CMU-CS-87-150, (1987)