協調処理モデル Cellula の分散処理系†

吉田紀彦†† 楢崎修二†††

我々の提案している協調処理モデル Cellula について，これを具体化する処理系をワークステーション・ネットワーク上に作成した．これは Cellula の基本操作をシステム関数として持つ並列 Lisp のインタプリタであり，各プロセッサ上での多重処理およびネットワーク上での分散処理機能を持つ．本処理系にはセル（実行主体と通信媒体の一体化した存在）をネットワーク上に仮想的に拡散させる機構も仮想共有空間の概念を応用して実装した．これには“一時キャッシュ法”と名付けた手法を用い，従来のものに比べて簡便な機構ながら，状況によっては 1/2〜1/3 というネットワーク通信の低減を実現している．

1. はじめに

協調処理とは，複数の実行主体が互いに協力や調停を行いながら集団全体で1つの作業を行うような処理形態をいう．これは，全体的な組織形態や制御構造が事前に十分に規定できず，作業の実行過程が各主体の自律的判断に委ねられるような応用のためのものである．このような応用分野としては例えば次のものがある．

① シミュレーション：多数の要素から構成される系の挙動を，系全体の（「大数の法則」的な）性質ではなく個々の要素の性質とそれらの間の関係に注目して模倣する．

② 分散制御：多数のセンサとアクチュエータから構成されるシステムにおいて，中央機構なしに全体を正しく制御する．制御システムにおける中央機構の排除は，耐故障性向上の意味合いも持つ．

③ 協同作業支援：人間の協同作業を計算機によって支援する．人間の協同作業はそのものが協調処理である．

④ 知識処理・人工知能：単一のエキスパートでは解決の困難な問題を複数のエキスパートの協力によって解決する．または，協調処理に基づく並列化により問題解決の高速化を図る．

我々は，協調型問題解決を主な対象とする協調処理モデル Cellula を提案している[1)-3)]．これは黒板モデルないし場（環境）の概念を基礎とするものであり，その本質は次の2点にある．

① 集団行動の抽象化を支援する．これにより“集団の集団”という再帰的階層が記述可能になる．この点が TupleSpaceSmalltalk[4)] などの並行オブジェクト指向言語や Ether[5)]，Linda[6)] など黒板通信に基づく並行処理言語との最大の相違である．具体的には，実行主体としてのプロセスとカプセル・通信媒体としての場（環境）を一体化してセルと名付け，これを再帰的にネストさせる．

② 様々な形態の集団間／内の相互作用に必要な通信を支援する．具体的には，黒板通信を拡張したパターン照合を通信の基礎とし，通信の形式は通信情報の属性で規定する．

これにより例えば，1対1通信，あるグループへの放送，受信者を特定しない送信，通信情報の内容による選択的受信など，多様な通信形態が単純かつ統一的な枠組の中で実現される．また，一般の問題解決技法だけでなく，分割統治法のような再帰的技法への対応も可能になる．

これまでに，本協調処理モデルを具体化した処理系をワークステーション・ネットワーク上に作成し，その適用性と有効性を検証すべく，分散探索，協調型問題解決，再帰的問題分割など，様々な例題に適用してきた[1),3)]．

これらの実験の中から明らかになった最も大きな問題は，セルを各々1つのプロセッサに配置するため，通信媒体として機能するセルを持つプロセッサへのアクセス集中が生じる，というものである．これを解決すべく，仮想共有空間（virtual shared space）[7)-10)] の概念を応用し，ネットワーク全体に仮想的に拡散するセルの機構を導入して，これを処理系に実装した．これにより，特定プロセッサへのアクセス集中の回避と

† A Distributed Processing System for the Cooperation Model 'Cellula' by NORIHIKO YOSHIDA (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University) and SHUJI NARAZAKI (NTT Software Laboratories).

†† 九州大学工学部情報工学科
††† NTT ソフトウェア研究所

ネットワーク通信の低減を実現した．その実装には“一時キャッシュ法”と名付けた手法を用いている．これは従来の仮想共有空間機構に比べて簡便なものながら，状況によっては 1/2〜1/3 というネットワーク通信の低減を達成している．

本稿ではこの Cellula の分散処理系について，特に拡散セルの機構に焦点を当てて述べる．以下，第2章で協調処理モデル Cellula について本稿の理解に必要な内容を概観した後，第3章で分散処理系の基本部分について，第4章で拡散セルの機構について述べる．第5章はまとめである．

2. 協調処理モデル Cellula の概要

2.1 セル＝プロセス＋場

実行主体としてのプロセスとカプセル・通信媒体としての場（環境）を一体化した存在をセルと名付ける．すなわち，セルは実行主体として自律的に処理を進めるとともに，自分の内部に固有の通信情報プールを持ち，この中の任意の情報単位について書出し／読込みを行う．セルは Cellula に基づくシステムを構成する唯一の存在である．

セルは動的に生成・消滅し，新たに生れた子セルはその親セルの「内側」に置かれる．すなわち，生れたばかりの子は親の外側にいるセルの存在を知らず，一方，親の外側のセルは子の存在を知らない．これにより集団の抽象化が実現する．親は子に対して，抽象化のカプセル・間接通信の媒体・集団行動の監視者などとして作用する．

セルに対する基本操作として次の5つがある．

def-cell(Cell-Symbol, Cell-Code)
　セルのプログラム・コードを定義する
create(Cell-Symbol, {Range})
　子のセルを生成して，その子セルを返す
suicide()　自殺する
self()　そのセル自身を返す
env()　環境（親）のセルを返す

2.2 タプルとその属性

セル間で通信される情報単位をタプルと名付ける．通信はタプルとこれに対応するテンプレートの照合によってなされる．1つの通信には，タプルを書き出すセル，テンプレートを用意して対応するタプルを読み込もうとするセル，タプルおよびテンプレートの置かれる場を提供するセルの3つが関与する（ただし媒体セルは具体的な通信内容に関知しない）．

タプル（およびテンプレート）は名前と，0個以上のデータ要素の組から構成される．各要素（セルそのものも要素になり得る）は型を持ち，また，値を持つ actual かまだ持たない formal のいずれかである．タプルとテンプレートが対応する条件は，両者が同じ名前と型を持つ，および，同じ位置のデータ要素の値が等しいかいずれか一方が formal である，の2つである．照合に成功すると，テンプレートは対応するタプルで具体化される（すなわち，テンプレート内の formal 要素が対応するタプルの actual 要素で置換される）．

タプル（およびテンプレート）は，通信形式を指定するために幾つかの属性を持つ．

① 封鎖／非封鎖：封鎖通信か非封鎖通信かを指定する．

② 排他／非排他：1対1通信か1対多通信かを指定する．

③ 優先順位：競合するタプル／テンプレートの間で照合の優先順位を指定する．

タプルに対する基本操作として次の2つがある．なお，これらはいずれもアトミック・アクションとして動作し，またセルに対する相互排斥を保証する．

out(Cell, Tuple, {Attribute}, {Exception})
　タプルをセルに書き出す．
in(Cell, Template, {Attribute}, {Exception})
　テンプレートに対応するタプルをセルから読み込む．

3. Cellula の分散処理系

3.1 基本方針

ここでは，Sun-3 を Ethernet で結合したワークステーション・ネットワーク上に分散処理系の実装を行った．具体的には，前章で示した Cellula の基本操作（セル操作5つ，タプル操作2つ）をシステム関数として追加した形の並列 Lisp をまず設計し，これを Cellula/Lisp と名付けた．そして，その多重処理系を Common Lisp 上のインタプリタとして作成した．その基盤として最初は Kyoto Common Lisp[11]，次いで Hokkaido Common Lisp を用いている．その理由はいずれもソースが無償ソフトウェアとして公開・配布されていることにある．

処理系の実装は段階的に行った．すなわち，まず全セルを単一プロセッサ上に置く集中版，次いでプロセッサは分散させるが場はある1つのタプル・サーバ上に置く中央集権版，そして最終的に各セルを分散させ

た版を作成した．なお分散化にあたっては，TCP/IPのシステムコール・インタフェイスをCで記述して，ソケットを Common Lisp のストリームとして扱えるようにしている．以下，この最終版について述べる．

3.2 処理系の概要

各プロセッサ上の Cellula/Lisp 処理系の構成を図1に示す．各モジュールの機能の概要は以下のとおりである．

（1） CL-translater

Cellula/Lisp プログラムを仮想機械命令に変換する．なお，Common Lisp の関数を Cellula/Lisp から直接呼び出す機能，仮想機械命令を動的に定義・追加する機能も用意している．

（2） VM-interpreter

仮想機械命令を実行する．インタプリタはスタック指向の命令セットを持ち，対話的なプログラムの定義・実行が可能である．仮想機械の概要を示すために，その命令セットを表1にまとめる．

（3） Scheduler

インタリーヴ方式のプロセス・スケジューリングを行う．なお，時分割処理は行わないため，プロセス切替えはタプル操作 (in/out) およびセルの生成 (create) の際にのみ生じる．スケジューリングはラウンドロビン方式ではなく，セル内のタプル数に応じて優先度を定める方式を採用している．これは，タプルを多く持つセルの処理を先行させてタプルの遅滞を低減するためである．

（4） Tuple-module

タプルの保持と管理を行う．タプルはセルとタプル名に基づく多段階ハッシュ構造として保持される．タプル管理についての詳細は後述する．

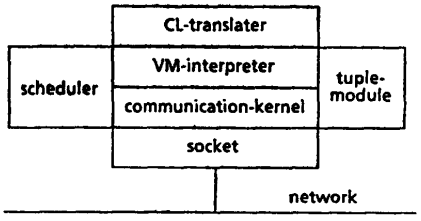

図1 Cellula/Lisp 処理系の構成
Fig. 1 Configuration of the Cellula/Lisp system.

（5） Communication-kernel

他プロセッサとの通信を行う．この中でセルの所在はネットワーク透明になる．プロセッサ間通信についても詳細は後述する．

（6） Socket

TCP/IP ソケット・ストリームを実現する．このモジュールのみCで記述されている．

3.3 タプル管理

タプルは各プロセッサごとに次のようなデータ構造で保持される．

セル名を鍵とするハッシュ表
↓
タプルを鍵とするハッシュ表
↓
タプル・テンプレート・統計データのスロット
↓
タプルの型を鍵とする連想リスト
↓
（タプル名に応じた）優先度順のキュー

表1 仮想機械命令セット
Table 1 Virtual machine instruction set.

スタック操作	push-stack pop-stack pop-stack-1 top-pop-push dup-stack apply-function-no-arg apply-stack-top apply-stack apply-stack-and-no-push eval-stack exchange-stack push-function set-roll-back-pointer pop-roll-back
実行制御	function-call macro-call function-return wait-terminal-input push-frame pop-stack-push-frame pop-stack-push-frame-variable-arg pop-frame jump pop-jump nil-branch nil-branch-or-pop pop-nil-branch non-nil-branch non-nil-branch-or-pop pop-non-nil-branch enter-critical-region exit-critical-region suicide-process
変数操作	push-var-value set-var-value pop-set-var-value define-variable define-variable-and-value push-to-var increment-variable increment-variable-1 increment-variable-by decrement-variable decrement-variable-1 decrement-variable-by setf-apply define-function define-cell
その他	set-process nop bootstrap exit-cellula

(ここで，下矢印は上のデータ構造が下のデータ構造を要素として含むことを示す)．そして，各セルは自分に対応するハッシュ表へのポインタを持つ．

データ要素として許される型を次に掲げる．

integer, real, complex, character, string, cell, list, array

これらのうちの後2者は構造データであり，処理系の簡便化と高速化のため，actual から formal への内容の複写のみを考え，actual どうしの値の照合は必ず成功するものとした．この制約は基盤とするモデルのセマンティクスにはないものであるが，これまでの様々なプログラミング実験において支障は生じていない．

タプル管理について本処理系の性能を示すと，変数参照に 0.4 msec を要するのに対し，自己のタプルの参照には 2 msec と約5倍の時間を要する．

3.4 プロセッサ間通信

ネットワーク上で，セルはプロセッサ・アドレスとプロセッサ内のセル識別子によって一意に識別される．この対をセル ID とする．

プロセッサ間で授受される情報には次のものがある．

テンプレートの送信
タプルの送信
セル割り付けプロセッサの獲得
セル割り付けプロセッサの決定
プロセッサのサービス終了
照合済みタプルの送信
照合済みテンプレートの送信
「対応するものがなかった」
「セルがなかった」
セルの生成要求
セル生成の返値
プロセッサのセル割り付け許可
プロセッサのセル割り付け一時禁止
セル定義の探索要求
セル定義の探索の返値

これらは次の共通フォーマットでネットワーク上を転送される．

```
(message-type-and-attributes
   ( receiver-cell-id
     sender-cell-id .
     medium-cell-id)
   tuple
   ( type mode. template-id) .
   priority)
```

4. 拡散セル

4.1 基本概念

当初の処理系では，セルの持つ場（環境）はそのセルのプロセスがいるプロセッサ（以下，これをホーム・プロセッサと呼ぶ）に配置していた．すなわち，通信媒体として機能するセルへのタプルとテンプレートの書込みはすべてそのホーム・プロセッサに集中することになり，このアクセスの集中が処理上のボトルネックとなる．これをある程度回避するようにアプリケーション・プログラムを記述する技法も存在する[2]．しかし，このようなアクセス集中を意識しないプログラムについても，処理系が自動的にこれを回避するのが望ましい．そこで，仮想共有空間の概念を用いてネットワーク全体に場（環境）が仮想的に拡散しているセルを導入することにより，アクセス集中の回避，すなわちプロセッサ間通信回数の低減を図った．この拡散セルの機構を図式的に図2に示す．

ワークステーション・ネットワークではプロセッサ間通信時間がプロセッサ内計算時間よりも2桁程度大きいため，通信回数の低減が全体の処理時間の改善にも寄与することが期待される．

仮想共有空間とは，物理的な共有メモリを持たないアーキテクチャの上で論理的な共有メモリを仮想的に実現してプロセッサ間通信を低減する機構である．その手法は幾つか存在するが[7]，次の2つが代表的である．

① キャッシュの手法を応用する[8]．共有空間をページ単位に分割して各プロセッサに割り当てておく．プロセッサは自分の持つページ以外にアクセスする場合，その持主からコピーを受け取ってキャッシュに納める．書込みが生じるとそのコピーが持主に返されるとともに，そのページの無効化メッセージが他のプロセッサに放送ないしマルチキャストさ

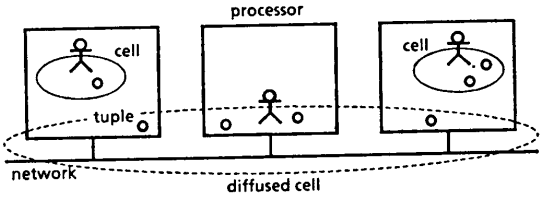

図 2 拡散セルの概念
Fig. 2 A schematic picture of a diffused cell.

れる.

② すべてのプロセッサに共有空間の同一コピーを持たせる[9]. すべての書込みはネットワーク全体に放送されるが, 読込みは各プロセッサ内でローカルになされる.

特に前者は複雑な手順を要し, またいずれも放送（またはマルチキャスト）機構を前提としている. しかし, 放送機構は着信保証が難しい, 通信コストがネットワークの大きさや形状に依存しがちである, などの難点を持つ. したがって, 放送機構を用いない手法が望まれる.

4.2 一時キャッシュ法

我々は, Unix の放送機構である UDP は用いず, 着信保証のある TCP/IP 上の通信のみを用いて, 仮想共有空間上の拡散セルを実現する簡便な手法を考案した. この手法は2.2節で述べた排他型（1対1）通信を対象とするものであり, その本質は次の2点にある.

① 拡散セルのホーム・プロセッサへタプル／テンプレートを送り出すのをある時間だけ遅らせ, その間に対応するテンプレート／タプルが自プロセッサ内で現れたならば, プロセッサ間通信なしに照合を行う.

② 拡散セルのホーム・プロセッサへテンプレートを送り出すに際して, そのコピー（以下, ゴーストと呼ぶ）を手元に残しておく. そして, 対応するタプルがホーム・プロセッサ内よりも自プロセッサ内で先に現れたならば, 真のテンプレートとの照合をゴーストとの照合で代用する. 真のテンプレートとゴーストとの一貫性は後に述べる方法で保つ.

送出すべきデータを自プロセッサに留めおき, さらにコピーを手元に残すところから, この手法を“一時キャッシュ法”と名付ける. これの適用がモデルのセマンティクスを変えてしまうことはない. 具体的な手続きは次のとおりである.

① タプルおよびテンプレートをΔ時間だけ自プロセッサに留めおく. Δの与え方については後に触れる.

②-a Δ時間内に対応する相手（タプルの場合にはテンプレート, テンプレートの場合にはタプル）が現れた場合には, 自プロセッサ内で照合処理を行う.

②-b Δ時間内に対応する相手が現れない場合には, 拡散セルのホーム・プロセッサに送り出す. この際に, 排他テンプレートならば自プロセッサ内にゴーストを残す. そして:

③ ホーム・プロセッサは, 送られてきたタプルとテンプレートについて従来どおりの照合処理を行う. ただし, テンプレートをすでに送ってきているセルがさらにテンプレートまたはタプルを送ってきた場合は, 最初のテンプレートはその自プロセッサ内ですでに照合が起きているので削除する.

④-a ゴーストを持つ自プロセッサ内での照合が先に起きた場合には:

⑤ （照合に伴い, ゴーストはテンプレートとして消去されている.）

⑥ その後ホーム・プロセッサ内でも照合が起きてテンプレートに対応するタプルが自プロセッサに送られてくる可能性がある. しかし, その照合はすでに無効なので, 送られてきたタプルを未照合のものとして改めてΔ時間だけ自プロセッサに留めおく.

④-b ホーム・プロセッサ内での照合が先に起きて, テンプレートに対応するタプルが自プロセッサに送られてきた場合には:

⑤ ゴーストを消去する.

⑥ 送られてきたタプルが封鎖型ならば, それを書き出したセルにシグナルを送って封鎖を解除する.

ここで, ③および⑥が真のテンプレートとゴーストとの一貫性を保つための手続きである.

なお, 現在の処理系では, 拡散セルとするか否かはタプル名ごとに指定できるようになっている.

4.3 評　価

拡散セルにおけるプロセッサ間通信回数について見積りを行う. なお, これに先立って, 通常のセルを媒体とした場合の通信回数を示すと, 読み書きするセルとその媒体セルが別プロセッサ上にある場合には,

テンプレートの送出
タプルの送出
照合結果のタプルの返送

となり, 最大3回である.

ここでは, 対応するタプルとテンプレートが同じプロセッサ内で書き出される確率をp, そのうちでこれがΔ時間内に起きる確率をqとする. pはプロセッサ台数を増やすと減少するものと考えられ, また, qはΔを大きくすると, あるいはプロセッサの計算速度が向

上すると増大する.

以下に通信回数の期待値を，各々の場合の生じる確率に基づいて計算する.

① Δ 時間内に自プロセッサ内で照合が起きる確率は pq. この場合の通信回数は 0.

② Δ 時間以降に自プロセッサ内で先に照合が起きる確率は $p(1-q)$. 通信回数は，テンプレートが先に書き出されていたならば 1 (ホーム・プロセッサへのテンプレートの送出)，タプルが先に書き出されていたならば 3 (タプルの送出，テンプレートの送出，タプルの返送). 1対1通信のため両者は等確率と考えられ，平均 2.

③ 自プロセッサ内で照合が起きない確率は $1-p$. 通信回数は 3.

したがって，通信回数の期待値は，

$$pq\cdot 0+p(1-q)\cdot 2+(1-p)\cdot 3$$
$$=3-p(1+2q)$$

となり，最悪でも従来の 3 を越えない.

なお，同様にして全通信時間の期待値を計算すると，プロセッサ間通信時間を単位として，

$$pq\cdot 0+p(1-q)\cdot\{0.5(1+\Delta)+0.5(3+\Delta)\}$$
$$+(1-p)\cdot(3+\Delta)$$
$$=3+\Delta-p\{1+q(2+\Delta)\}$$

となる. 本手法の適用による処理の遅れ時間は最悪でも Δ を越えない.

Δ の与え方についての理論的根拠はない. しかし，その指針に関して次のような考察が可能である. すなわち，本手法がデータの送出を Δ だけ遅らせることで，これを適用しない場合に 3 (×プロセッサ間通信時間) だけ要していた自プロセッサ内の通信を 0 にするものであることを考えると，Δ はプロセッサ間通信時間の 3 倍以下とするのが適当であろう.

4.4 実　　験

一時キャッシュ法に基づく拡散セル機構がプロセッサ間通信回数に及ぼす効果を実際に検証するために，通常のセルと拡散セルを通信媒体にした場合について，比較実験を行った. 具体的には，次の 2 つの例題プログラム[3]を用いた.

- 分散分枝限定法による巡回セールスマン問題
- 協調型問題解決による覆面算

これらはいずれも原理的には，ある環境の中に解の候補をためておいてこれを絞り込む，というプログラムであり，数値計算や問題解決の協調分散化において広い応用性を持つ. ここでの環境セルが実験の対象となる.

図 3 に，各々の例題について環境を通常セル／拡散セルとした場合の全通信回数の，プロセッサ台数に応じた変化を示す. ここでは $\Delta=3$ (×プロセッサ間通信時間) としている. 図からわかるように，いずれの例題についても，プロセッサ 3 および 4 台の場合で概ね 1/2〜1/3 の通信回数の低減を実現している. ただし，分散分枝限定法でプロセッサ 2 台の場合には，拡散セルで通信回数がかえって増加している. これは，一時キャッシュ法による通信の遅延に伴って探索木の枝刈順が変化し，解の発見が遅れたためと考えられる.

特にプロセッサ 3 および 4 台の場合の実験結果について，4.3 節で得られた見積りをもとに検討を加える. ここで，環境が通常セルと拡散セルのいずれの場合もセル間通信回数 (プロセッサ内／間を問わず全体の) は等しいものと仮定する. プロセッサ間通信回数を通常セルの場合を Co, 拡散セルの場合を Cd とすると，次の関係が成り立つ.

$$\{3-p(1+2q)\}/3=Cd/Co$$

分散分枝限定法でプロセッサ 4 台の場合について注目すると，$p=0$〜1 という制限から $q=0.52$〜1 という結果が得られる. 仮に $q=0.75$ とおいてみると p の値として次が得られる.

- 分散分枝限定法… 3 台：0.66, 4 台：0.82

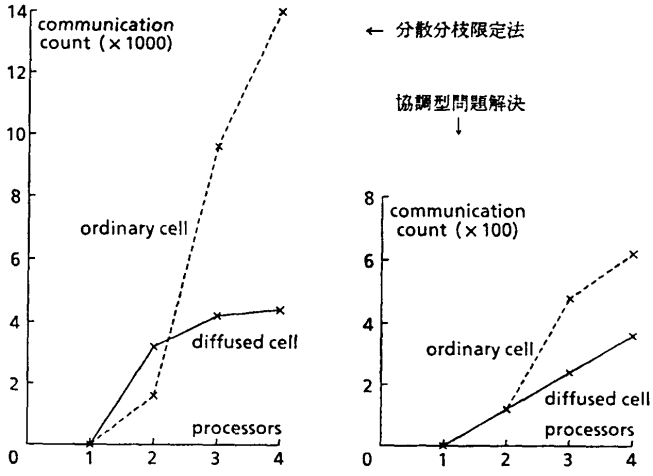

図 3 通常セルと拡散セルの通信回数の測定結果
Fig. 3 Communication counting on an ordinary cell vs a diffused cell.

● 協調型問題解決… 3台：0.61, 4台：0.50

分散分枝限定法で，p が台数の増加に従って増大しているが，これも上と同様に，探索木の枝刈順が変化したためではないかと考えられる．

なお，プロセッサ通信回数の低減に伴い処理時間の改善も期待される．しかし，この測定については本論文では割愛する．なぜなら，ワークステーション・ネットワーク上の処理系であって他のシステム・プロセッサなども稼働していること，キャッシュ・ヒット率の影響を受けること，ガベージ・コレクションが随時発生することなどから，正確な測定が難しいためである．

4.5 他の手法との比較

ここで提案した一時キャッシュ法は一般の仮想共有空間の実現手法に比較して，次のような限定された状況を想定している．

① アドレスを指定する共有メモリ上の通信ではなく，パターン照合による通信を対象とする．

② 1対1通信のみを対象とし，書込み1回と読出し1回とが必ず対応する．

③ 1対1という通信形式が，授受される情報の属性として明示的に与えられる．

この①と②により，放送機構を用いない簡便な実現が可能になったものである．また①により，一貫性の管理も容易になっている．しかし逆に言えば，パターン照合による1対1通信しか適用できない．

プロセッサ間通信を遅らせることで通信回数を低減するアプローチとして，Munin[10]では delayed update という手法が提案されている．しかし，一時キャッシュ法では遅らせる目的がその間に生じた自プロセッサ内通信を吸収するところにあるのに対して，この delayed update では数回分の通信をためておいて1つにまとめるために遅らせる点が異なる．

5. おわりに

我々の提案している協調処理モデル Cellula について，これを具体化すべくワークステーション・ネットワーク上に作成した分散処理系について述べた．これは多重処理および分散処理機能を持ち，さらにネットワーク通信軽減のためにセルをネットワーク上に仮想的に拡散させる機構も有している．

現在の分散処理系はインタプリタ方式であり，処理効率の面から見て理想的な実現とは言い難い．並列 Common Lisp の上で処理系を再構築することを検討中である．

一時キャッシュ法に基づく拡散セル機構については，従来の仮想共有空間機構に比べて簡便なものながら状況によっては効果を上げている．ただし，パターン照合による情報授受という Cellula の通信形態に依存した機構であり，また1対1通信のみを対象としている．Cellula の非排他型（1対多）通信を対象とする同様な拡散セル機構の開発が今後の課題である．

謝辞 日頃からご支援を頂く九州大学牛島和夫教授に深謝する．また，多くの有益なご助言を頂いた査読者の方々に感謝する．

参考文献

1) 吉田紀彦，楢崎修二：場と一体化したプロセスの概念に基づく並列協調処理モデル Cellula，情報処理学会論文誌，Vol. 31, No. 7, pp. 1071-1079 (1990).
2) 楢崎修二：場と一体化したプロセスの概念に基づく協調処理モデル Cellula，九州大学修士論文 (1990).
3) Yoshida, N. and Narazaki, S.: A Cooperation and Communication Framework for Distributed Problem Solving, *Proc. Second IEEE Int. Conf. Tools for Artif. Intell.*, Washington D.C., pp. 530-536 (1990).
4) Matsuoka, S. and Kawai, S.: Using Tuple Space Communication in Distributed Object-Oriented Languages, *Conf. Proc. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA '88)*, San Diego, pp. 276-284 (1988).
5) Kornfeld, W.A. and Hewitt, C.E.: The Scientific Community Metaphor, *IEEE Trans. Sys. Man and Cyber.*, Vol. SMC-11, No. 1, pp. 24-33 (1981).
6) Carriero, N. and Gelernter, D.: How to Write Parallel Programs: a Guide to the Perplexed, *ACM Comput. Surv.*, Vol. 21, No. 3, pp. 323-358 (1989).
7) Stumm, M. and Zhou, S.: Algorithms Implementing Distributed Shared Memory, *IEEE Comput.*, Vol. 23, No. 5, pp. 54-64 (1990).
8) Li, K. and Hudak, P.: Memory Coherence in Shared Virtual Memory Systems, *ACM Trans. Comput. Syst.*, Vol. 7, No. 4, pp. 321-359 (1989).
9) Carriero, N. and Gelernter, D.: The S/Net's Linda Kernel, *ACM Trans. Comput. Syst.*, Vol. 4, No. 2, pp. 110-129 (1986).
10) Bennett, J.K., Carter, J.B. and Zwaenepoel, W.: Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence, *Proc.*

Second ACM SIGPLAN Symp. on Principles and Practice of Parallel Programing (PPoPP), Seattle, pp. 168-176 (1990).
11) 湯浅太一，萩谷昌己：Common Lisp 入門，岩波書店 (1986).

(平成2年8月27日受付)
(平成3年2月12日採録)

吉田 紀彦 (正会員)

1957年生．1979年東京大学工学部計数工学科卒業．1981年同大学大学院修士課程修了．(株)三菱総合研究所，東京大学工学部を経て，現在，九州大学工学部情報工学科助教授．工学博士．プログラミング方法論，並列・分散・協調処理などに興味を持つ．計測自動制御学会，ソフトウェア科学会，ACM, IEEE, AAAI 各会員．

楢崎 修二 (正会員)

1965年生．1988年九州大学工学部情報工学科卒業．1990年同大学大学院修士課程修了．現在，NTT ソフトウェア研究所勤務．並列／分散／協調処理，記号処理などに興味を持つ．ソフトウェア科学会会員．