オペレーティング・システム 51－9
（１９９１．７．１８）

遠隔手続き派遣による分散透明なデータ共有

千葉 滋　加藤 和彦　益田 隆司

東京大学 理学部 情報科学科

分散環境上のアプリケーションの開発には、プロセスの間のデータ共有の機構が不可欠である。本稿では他のプロセスのアドレス空間上のデータを容易にアクセスできる機構を提供することで、プロセス間のデータ共有を実現する手法を述べる。このため、遠隔手続き呼び出しと遠隔ポインタの機構をアプリケーション記述言語に導入する。単純な実装では遠隔ポインタによる参照は実行効率が悪いので、さらに本稿は遠隔手続き派遣の機構を提案し、これをもちいて遠隔ポインタによる参照を改善する。遠隔手続き派遣とは、一連の遠隔データ・アクセスをひとつの手続きとして、データが存在するアドレス空間上で実行する機構である。

Remote Procedure Delegation for Distributed Data Sharing

Shigeru CHIBA　Kazuhiko KATO　Takashi MASUDA

Department of Information Science, The University Tokyo

Applications in distributed systems require facilities that enable data sharing between processes. To provide such facilities this paper suggests to allow a process to directly access data in an address space of another process. A language for describing applications is extended to support remote procedure calls and remote pointers. Because simple implementation of reference by remote pointers is inefficient, new linguistic facilities, called remote procedure delegation, are proposed. By this facilities, a procedure is executed in an address space where there exist data referenced by this procedure so that remote references can be efficiently carried out.

1 はじめに

ワークステーションの高性能化、およびネットワーク技術の発展により、CSCW (Computer Supported Cooperative Work) やグループウェアに代表される分散環境上のアプリケーションの需要が予想される [6, 9]。このような分散アプリケーションでは異なるサイト上の自律したプロセスが互いに密にデータを交換しあって全体の処理を実行する。したがって、分散アプリケーションを効率的に開発するためには、プロセスの間の情報の共有を容易におこなえるような機構が不可欠である。

プロセスの間でデータを共有するための方法としては、分散共有メモリ [7] やタプル・スペース [3] などのように、広い意味での共有アドレス空間を実現する方法がある。プロセスの間で共有されるデータは、分散環境で実現される仮想的な共有空間に配置される。実際にはデータは個々のプロセスがそのデータの複製を持つことで共有される。プロセスがデータを参照するときは自分の持つ複製を参照する。あるプロセスが共有しているデータを更新したときにはその更新を他のプロセスにも通知し複製を更新しなければならない。分散環境ではプロセスどうしの間の通信路が遅いのでデータの更新は時間がかかる。したがって、分散環境で共有アドレス空間を実現すると、実行上のオーバーヘッドが大きい。

本稿は、仮想的な共有アドレス空間を使わずに、個々のプロセスが他のプロセスのアドレス空間上のデータを容易にアクセスできる機構を提供することで、プロセス間の情報共有を実現する方法について述べる。共有するデータはあるプロセスにローカルなアドレス空間におき、他のプロセスはそのデータを直接参照する。この方法では、共有データを更新したときに、他のプロセスに更新を通知する必要がないために、共有アドレス空間を使う方法に比べて共有データの更新の効率がよい。しかしこの方法ではデータのアクセスのたびにプロセス間の通信が必要となる。共有アドレス空間を使った方法では、参照の場合は通信を必要としないので、参照はより遅くなる。そこで本稿が提案する方法では、アプリケーションの記述言語に、遠隔手続き派遣と呼ぶ新しい言語機構を導入して、データ参照の実行効率を改善する。遠隔手続き派遣は、一連のデータ参照・更新をひとつの手続きとして、データが存在するアドレス空間上でまとめて実行する機構である。個々の参照・更新をばらばらに実行するのに比べ、プロセスの間の通信回数を減らすことができる。

2 分散C言語

著者らの研究室では、C言語の手続き呼び出し機構を遠隔手続き呼び出しに拡張した分散C言語 [12, 10] を開発した。従来の遠隔手続き呼び出しと異なり、データを指すポインタや手続きを指すポインタを含む任意の型のデータを引数として渡せる。ポインタが引数として渡せるため、分散C言語ではプロセスにローカルなアドレス空間上のデータを指すポインタの他に分散した他のアドレス空間上のデータを指すポインタを別に用意している。前者を**局所ポインタ**、後者を**遠隔ポインタ**と呼ぶ。遠隔手続き呼び出しをおこなうときに、引数として渡される局所ポインタは遠隔ポインタに変換される。遠隔ポインタが指すデータを直接読みだすことや変更することを**遠隔参照** (remote reference) と呼ぶ。遠隔手続き呼び出しされた手続きの中で遠隔参照がおきると、ポインタの指すデータの値が転送され、呼ばれた側のローカルなアドレス空間に複製が作られる。実際のアクセスは複製にたいしてなされる。この複製にたいする更新は、呼び出された手続き終了時に元のデータに反映される。また手続きにたいする遠隔参照がおきると、制御をその手続きを保持しているプロセスに移して、手続きを実行する。

分散C言語のひとつの目標は通常のC言語の手続き呼び出しと全く同じ意味づけを遠隔手続き呼び出しに与えることである。遠隔ポインタは、遠隔手続きを呼び出した側のデータを呼ばれた側からアクセスするために導入された。したがって遠隔ポインタの使用には制限をもうけた。すなわち、遠隔ポインタは遠隔手続き呼び出しされる手続きの引数としてだけ現れ、その手続きの中だけで有効である。遠

隔ポインタの値を保存しておいて、別の手続きが呼び出されたときに使うことはできない。この制限により、あるプロセスAのアドレス空間にあるデータが遠隔参照されたとき、それを参照するプロセスBは、Aが遠隔手続き呼び出しで起動したプロセスであることが保証される。この制限がない場合は、遠隔ポインタがいくつかのプロセスに共有されるので、そのポインタが指すデータのアクセスに並行性(concurrency)が発生する。その場合、遠隔参照の際に複製をつくると、複製の間に非一貫性が生じる恐れがある。

分散C言語では遠隔ポインタの使用に制限をつけているので、遠隔ポインタを共有して、データの共有をおこなうことはできない。また遠隔ポインタは遠隔手続きの呼び出し側から呼ばれる側へしか渡せず、遠隔手続き呼び出しの戻り値として遠隔ポインタを渡すこともできない。このため遠隔手続き呼び出しされた側のプロセスが、呼んだ側のプロセスのアドレス空間を遠隔参照することしかできず、逆ができない。

本稿では異なるプロセスのアドレス空間を直接アクセスできるようにするため、分散C言語の遠隔ポインタの使用に関する制限をなくし、遠隔ポインタを、遠隔手続き呼び出しの引数としてだけではなく、一般的なデータとして扱えるようにすることを考える。制限をなくすと共有データのアクセスに並行性が生じるので、遠隔ポインタによるアクセスのたびに、実際に遠隔参照をおこなわなければならない。分散環境で遠隔ポインタによるアクセスのたびに遠隔参照を実行すると実行効率を著しく下げてしまう。そこで次章では遠隔手続き派遣をアプリケーション記述言語に導入して、遠隔ポインタによるアクセスの実行効率を改善する方法を提案する。

3 言語DCの遠隔参照

プロセスどうしの間で互いのアドレス空間を直接アクセスできるような機構を提供するために、アプリケーションの記述言語の中から遠隔ポインタを普通のデータとして扱えるようにする。遠隔ポインタが指している異なるアドレス空間上のデータをアクセスするたびに遠隔参照をおこなうことにすると、実行効率を大きく下げてしまう。そこで遠隔ポインタの意味づけを再検討し、さらに新たに遠隔手続き派遣という機構を考える。著者らは分散C言語をもとに、これらの機構を組み込み、個々のプロセスが他のプロセスのアドレス空間上のデータを直接にアクセスできるようにした分散言語 DC[1] を現在設計している。本稿では言語DCにおけるこれらの機構を解説する。

以下では個々のプロセスは遠隔手続き呼び出し(remote procedure call)で通信しあうとする。さらに各プロセスの内部には並列性はないものと考える。

3.1 遠隔ポインタによるデータの参照

遠隔手続き呼び出しの引数として渡された遠隔ポインタによってデータをアクセスするときはアクセスのたびに遠隔参照するのが直観的である。しかしアクセスのたびに遠隔参照するのでは実行効率が悪いので、遠隔ポインタによるアクセスの意味づけをより効率よく実行できるものに変える。

言語 DC では一般に遠隔ポインタによるアクセスがおきたときは、最初のアクセスのときに遠隔参照によってローカルなアドレス空間にコピーを作り、以後のアクセスはコピーにたいしておこなう。遠隔参照はデータがはじめてアクセスされるときに一回しかおきないので、実行効率がよい。このコピーにたいする更新は元のデータに反映されることはないので、言語 DC では遠隔ポインタによってオリジナルのデータを更新することはできない。また途中でオリジナルのデータが更新されても、それはコピーには反映されない。オリジナルのデータを更新するとき、あるいはオリジナルのデータの最新の値を参照するときは、次節に述べる遠隔手続き派遣の機構を使わなければならない。

言語 DC での遠隔ポインタによる参照は、実行効率がよいぶん、従来の意味づけでのポインタによ

[1] Distributed C の意味

る参照に比べ意味的に弱い。一方、遠隔手続き派遣はオリジナルのデータの更新などが可能で意味的に強力だが、実行効率は劣る。プログラマは、オリジナルのデータを更新する場合など、遠隔ポインタによる参照では目的を果たせないときにのみ、より実行効率が悪い遠隔手続き派遣を使う。遠隔ポインタによる参照と遠隔手続き派遣を使いわけることで、遠隔ポインタによるアクセスのたびに必ず遠隔参照する場合に比べ、実行効率を改善することができる。

遠隔ポインタを以下に示すような意味で使う場合は、遠隔手続き派遣でなく、遠隔ポインタによる参照だけですむ。遠隔手続き呼び出しのとき、グラフ構造や木構造のように内部にポインタを含む構造のデータを引数として手続きを値呼び (call by value) するのは、実行効率が悪い。ポインタはそのまま渡すことができないので、いちどポインタを含まない外部表現にデータを変換し、データを渡された手続きの側で再びポインタを含む表現に変換しなければならないからである [4]。このような場合はデータではなくデータを指すポインタを引数として渡すのがよい。遠隔ポインタがこのような意味で渡されるなら、遠隔ポインタによるアクセスが言語 DC のような意味づけでも問題ない。データそのものをを引数として値呼びするかわりにポインタを渡すので、オリジナルのデータを更新できなくてもよいといえる。また引数として渡されるポインタがオリジナルのデータのコピーを指す、という意味づけは、データそのものを引数として値呼びすることに意味的に近い。

言語 DC での遠隔ポインタによるアクセスは、いくつか注意しなければならない性質がある。遠隔ポインタによるアクセスはローカルなアドレス空間に作ったコピーにたいしてなされる。このコピーはデータが初めてアクセスされたときに遠隔参照によってオリジナルのデータから作られる。グラフ構造のデータのように、いくつかのデータがポインタによってつながってひとつのデータを構成しているような場合、個々のデータのコピーは参照されるまで作られない。したがって個々のデータのコピーが作られる時間はそれぞれ異なる。オリジナルのデータは随時更新される可能性があるので、データのコピーは全体として一定時点のオリジナルのデータの状態を反映しない。またデータそのものを引数として値呼びする場合と、データを指すポインタを引数として渡す場合とでは、意味的に同一ではない。手続き呼び出し開始からデータがアクセスされてコピーが作られるまでの間にデータが更新される可能性があるからである。

3.2 遠隔手続き派遣

前節で述べたように、遠隔手続き呼び出しで引数として渡された遠隔ポインタを使ったアクセスでは、オリジナルなデータへの遠隔参照はされない。オリジナルなデータを直接参照・更新するときには本節で述べる**遠隔手続き派遣** (remote procedure delegation) を使う。

遠隔手続き派遣とは、ローカルなアドレス空間上の手続きを、異なるアドレス空間上で実行する機構である。いわゆるプロセス・マイグレーションと異なるのはプロセス全体でなく手続きだけがアドレス空間を移動することである。ある手続きが別のアドレス空間に遠隔手続き派遣されると、派遣先のアドレス空間上のデータを指す遠隔ポインタによるアクセスはオリジナルのデータへの直接参照および更新となる。手続き内の局所ポインタは遠隔ポインタとして扱われ、それによるアクセスは常に遠隔参照によってオリジナルのデータへの直接参照・更新となる。普通の遠隔ポインタによる参照では、オリジナルのデータへの遠隔参照とはならないので、この点が特別である。さらに元のアドレス空間上の手続きを派遣された手続きの中から呼び出すと、その手続きが遠隔手続き呼び出しされる。

通常の遠隔手続き呼び出しと遠隔手続き派遣との違いは名前空間の取り扱いの違いである。遠隔手続き呼び出しの場合は、呼ばれた側の名前空間が使われるので、呼ばれた側のアドレス空間の大域変数や手続きを名前で参照できる。一方、遠隔手続き派遣の場合には、手続きを派遣した側の名前空間が使

われるため、派遣元のアドレス空間の大域変数や手続きを名前で参照できる。逆に派遣先のアドレス空間の大域変数や手続きを参照するためには、それを指すポインタをあらかじめ獲得していなければならない。名前空間の違いを除けば、遠隔手続き呼び出しと、遠隔手続き派遣はほぼ同じ機能をもつ。同じ処理をする手続きを派遣先に用意しておいて、遠隔手続き派遣をおこなうかわりに、その手続きを遠隔手続き呼び出ししても同等の結果が得られる。しかし遠隔手続き派遣の場合、派遣された手続きの中から派遣元のアドレス空間のデータを名前で参照でき、また派遣元のアドレス空間を指す遠隔ポインタによる参照はすべてコピーではなくオリジナルのデータへの遠隔参照になる。遠隔手続き呼び出しされた手続きの中から呼び出し側の手続きを呼ぶときは、遠隔手続き呼び出しの引数として、その手続きを指すポインタも渡さなければならない。派遣される手続きは、派遣元のアドレス空間上の大域変数などのデータを参照し、あるいは手続きを呼び出しながら、派遣先のアドレス空間上のデータを参照すると考えられるので、派遣元の名前空間が使われる遠隔手続き派遣を導入した方がプログラムの記述が容易になると考えられる。遠隔手続き派遣だけでプログラムを記述すると、プログラマは異なる名前空間に属するデータを参照したり手続きを呼ぶために余計な記述を強いられる。

言語 DC では遠隔参照はすべてこの遠隔手続き派遣を使っておこなう。プログラマはこれによってポインタによるアクセスの効率があがるように、手続きを実行するアドレス空間を明示することができる。ポインタによるアクセスごとに遠隔参照するのと異なり、一連の参照を手続きの中にまとめて派遣するので、アクセス効率が向上する。逆にただ一つの整数変数の値を更新するときも、それだけを実行する手続きを記述して遠隔手続き派遣を実行しなければならなず、記述量が増えてしまう。

遠隔参照するデータを他のプロセスが並行して更新しないことが明らかな場合には、分散C言語がしているように、ローカルな複製を作る方法の方が遠隔手続き派遣よりもアクセス効率の改善に有効である。しかし複製を作る方法は、複数のプロセスが同一のデータをアクセスする場合に効率が悪い。ひとつのデータにたいしていくつかの複製が存在する場合、その複製の間の一貫性をたもたなければならないからである [1, 8]。ある複製が更新されたときには、即座に他の複製にもその更新を反映させなければならない。二種類の更新がほぼ同時に起きたときには、それらの更新の間の順序関係がすべての複製で同一になるようにしなければならない。例えば更新1が X:=1 で、更新2が X:=2 のとき、ある複製では更新1が先、べつの複製では更新2が先ということになると、その結果決まる X の値が複製によって異なってしまう。これを避けるための複製の一貫性制御は更新のたびに複製をもつプロセスの間の通信を必要とするので、実行効率を下げてしまう。遠隔手続き派遣では、すべての更新を集中制御し、オリジナルのデータに直接反映させるので、このようなことは起こらない。いくつかの手続きが派遣されたときに派遣先のサイトがそれらの手続きを並列処理できない場合には、複製を作った方が並列処理ができる点で有利である。だがイーサネットを通したプロセス間の通信は、UNIX socket による通信を実測した結果、数ミリ秒必要なので、並列処理を考慮しても遠隔手続き派遣の方が全体として効率がよい。複製を使った方法では更新のたびに通信が必要だが、遠隔手続き派遣は、複数の更新をおこなうときも、手続き実行前と後の2回の通信しか必要としない。

4 言語 DC の処理系の実現方法

言語 DC におけるプロセス間の通信は、

- 遠隔手続き呼び出し
- 遠隔手続き呼び出しされた手続きの中から 遠隔ポインタでデータを参照したとき
- 同様に遠隔ポインタで手続きを呼び出したとき
- 遠隔手続き派遣

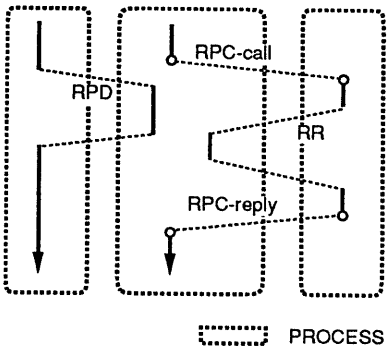

図 1: 遠隔ポインタによるアクセスと制御の流れ (RPC: 遠隔手続き呼び出し、RR: 遠隔参照、RPD: 遠隔手続き派遣)

- 遠隔手続き派遣された手続きの中から遠隔参照が起きたとき

におきる。言語 DC は分散C言語の遠隔ポインタの使用に関する制限をなくしたので、遠隔ポインタによるデータのアクセスは並行性が生じる。また遠隔手続き派遣は随時おこなえる。したがって個々のプロセスは通常の処理と並行して、遠隔手続き派遣や遠隔参照、さらに遠隔ポインタによる手続き呼び出しを受け付けなければならない (図 1)。このため個々のプロセスごとにそれらの処理をおこなう実行時システムを用意する。

4.1 暗黙の型変換

言語 DC では、局所ポインタ型の他に遠隔ポインタ型を用意する。プログラマは明示的に局所ポインタ型と遠隔ポインタ型を区別して使うことができる。遠隔ポインタ型はアドレス・ポインタの他にアドレス空間の識別子を含んでいるので、局所ポインタ型とワード長が異なる。局所ポインタ型の変数に遠隔ポインタ型の値を代入することはできない。(逆は可能である。) また局所ポインタ型による参照ではポインタの値をアドレスとして参照するのに対し、遠隔ポインタ型による参照では、ポインタの値からローカルなアドレス空間上においたコピーのアドレスを求め、そのアドレスにたいして参照がおこなわれる。

遠隔手続き呼び出しの引数としてポインタ型を指定するときはその引数を遠隔ポインタとして宣言する。遠隔ポインタと宣言された引数に局所ポインタ型の値を渡すと、暗黙のうちに遠隔ポインタ型に変換される。また遠隔手続き派遣された手続きの中では、局所ポインタ型として宣言された変数は暗黙のうちに遠隔ポインタ型に変換される。さらに構造体のメンバがポインタ型であるときも型変換が必要となることがある。局所ポインタ型のメンバをもつ構造体を考える。この構造体が遠隔ポインタで指されている場合、局所ポインタ型のメンバの値はすべて遠隔ポインタ型に型変換されて扱われる。構造体のメンバがさらに別の構造体を指すポインタ型になっていて、そのポインタをつかって構造体をたどるようなときは、ここで述べたような型変換を構造体をひとつたどるたびにおこなう必要がある。

遠隔ポインタと局所ポインタの間の型変換は分散環境に拡張した型システムによる型推論を使ってコンパイラが静的に実行できる。型システムに関する詳細は参考文献 [10] にゆずり、ここでは、型システムから導かれる型付け規則の例を示すにとどめる。ポインタ参照された構造体の局所ポインタ型のメンバを遠隔ポインタ型に型変換するかどうかは次のような推論規則から決定できる。構造体の型を τ_1、ポインタ型のメンバの型を τ_2 とする。型 τ を指す局所ポインタ型を $ref(\tau)$、遠隔ポインタ型を $rref(\tau)$ で表す。さらに $remote(\tau)$ で型 τ が定義されているアドレス空間はローカルなアドレス空間ではないことを表す。すると、図2のような型付け規則がいえる。M は型 τ_1 の構造体を指すポインタを使って型 τ_2 へのポインタ型のメンバの値を取り出す演算子を表す。C言語の -> 演算子に相当する。ポインタ x が局所ポインタ型のときは取り出される値 $M(x)$ の型も局所ポインタ型となり、遠隔ポインタ型のときは遠隔ポインタ型となる。

$$\frac{A \triangleright M : ref(\tau_1) \rightarrow ref(\tau_2) \quad A \triangleright x : ref(\tau_1)}{A \triangleright M(x) : ref(\tau_2)}$$

$$\frac{A \triangleright M : remote(ref(\tau_1) \rightarrow ref(\tau_2)) \quad A \triangleright x : rref(\tau_1)}{A \triangleright M(x) : rref(\tau_2)}$$

図 2: 型付け規則

4.2 遠隔手続き派遣の実現方法

遠隔手続き派遣の実現方法は、

(1) コンパイラで静的に派遣先のプロセスに派遣する手続きを組み込んでおく

(2) インタプリタで処理する

(3) 分散 OS XERO のロード／アンロードの機能を使う

が考えられる。言語 DC の最初の実装では (1) の方法を使うことを予定している。この方法では遠隔手続き派遣の際に、コードの転送が不要になるので実行効率がもっともよい。また型推論をコンパイル時におこなえる利点もある。欠点としては遠隔手続き派遣をする手続きやその派遣先を動的に決められないことがあげられる。(2) は派遣する手続き自体を外部表現に変換して派遣先のプロセスで解釈実行する方法である。最後の (3) は筆者らの研究室で開発中の分散 OS XERO 独自の機能を使う方法である [11]。分散 OS XERO ではマルチコンテキストの機能を提供している。コンテキストとはプログラムの実行単位である。ロード／アンロードの機能を使うと任意のコンテキストを任意の時点で任意のアドレス空間にロード／アンロードすることができる。派遣したい手続きをひとつのコンテキストにするとロード／アンロードで遠隔手続き派遣を実現できる。

5 関連研究

Shared Virtual Memory[7] や Tuple Space[3] は全てのプロセスにひとつの共有されたデータ空間を提供する。任意のデータは共有空間におくことで、すべてのプロセスから共有データとして扱えるようになる。これらの実現には複製を作る方法が有効で、共有されるデータの更新の頻度が小さいときに効率がよくなる。しかし更新が頻繁におこるときには効率が悪くなる。3 章で述べたような複製間の一貫性制御が必要になるからである。

一貫性制御の効率を改善するための方法として、弱い一貫性 (weak consistency) を導入する方法が最近研究されている [5]。弱い一貫性を導入すると共有データがプログラマにとって非常に扱いにくくなる。しかし共有データのアクセスの形態によっては弱い一貫性を導入してもしなくても共有データがプログラマにとって同様の振舞いをする場合がある。そこで筆者らはそのような共有データにたいしては弱い一貫性を使い、そうでないデータにたいしては普通の一貫性を使うことで、プログラマからみてすべての共有データが普通の一貫性で制御されているようにみせる方法を研究している [2]。複製を使って共有データ空間を提供するアプローチは本稿で述べたような異なるアドレス空間を自由に参照できるようにするアプローチと相対するものではなく、分散アプリケーション記述を容易にするために共に必要なアプローチと考えられる。

6 おわりに

本稿では、分散環境で個々のプロセスが互いのアドレス空間を容易にかつ効率的にアクセスできるようにするための方法を述べた。手続き呼び出しの機構とポインタによるデータの参照機構を持つ言語に、遠隔ポインタと遠隔手続き呼び出し、遠隔手続き派遣の機構をいれることで、異なるアドレス空間

を容易にかつ効率的にアクセスできるようになる。

著者らは、C言語に本稿で述べた機構を導入した言語DCを現在設計している。言語DCは、分散環境上のアプリケーションを容易に記述できるようにすることを目標としている。本稿で述べた機構に加えて、さらにいくつかの機構の導入を現在検討中である。ひとつはガベージ・コレクション(GC)機構である。言語DCでは遠隔ポインタによるアクセスのたびにデータのコピーができる。不要になったコピーを削除するためのGC機構を検討している。また並列プログラミングを容易にするために、トランザクション機構を導入することも検討している。

参考文献

[1] Bal, H. E. and A. S. Tanenbaum, "Distributed Programming with Shared Data," in *Proc. of the International Conference of Computer Languages*, pp. 82–91, 1988.

[2] Chiba, S., K. Kato, and T. Masuda, "Optimization of Distributed Communication in Multiprotocol Tuple Space," *submitted for publication.*

[3] Gelernter, D., "Generative Communication in Linda," *ACM Trans. Prog. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.

[4] Herlihy, M. and B. Liskov, "A Value Transmission Method for Abstract Data Types," *ACM Trans. Prog. Lang. Syst.*, vol. 4, no. 4, pp. 527–551, 1982.

[5] Hutto, P. and M. Ahamad, "Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories," in *Proc. of the 10th International Conference on Distributed Computing Systems*, pp. 302–309, 1990.

[6] Ishii, H., "TeamWorkStation: Towards a seamless shared workspace," in *Proc. of the Conference on Computer-Supported Cooperative Work*, pp. 13–26, 1990.

[7] Li, K., *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Dept. of Computer Science, Yale Univ., 1986.

[8] Stumm, M. and S. Zhou, "Algorithms Implementing Distributed Shared Memory," *IEEE Computer*, vol. 23, no. 5, pp. 54–64, 1990.

[9] Watabe, K., S. Sakata, K. Maeno, H. Fukuoka, and T. Ohmori, "Distributed Multiparty Desktop Conferencing System: MERMAID," in *Proc. of the Conference on Computer-Supported Cooperative Work*, pp. 27–38, 1990.

[10] 加藤, 大堀, 村上, 益田, "高階遠隔手続き呼出しに基づいた分散C言語について," *submitted for publication.*

[11] 成田, 加藤, 猪原, 益田, "分散OS XEROにおけるマルチコンテキストの実現方式について," 並列/分散/協調処理に関する『大沼』サマー・ワークショップ, 1991.

[12] 村上, 加藤, 益田, "遠隔手続き呼びだしに基づいた分散C言語について," 第 *42* 回情報処理学会全国大会論文集, 1990.