ハイパフォーマンス
コンピューティング 78－8
（1999．10．7）

計算機クラスタ環境における並列タスクの同期処理の検討

三浦圭司, 三田勝史 朝倉宏一 渡邉豊英
名古屋大学大学院 工学研究科 情報工学専攻

{miura, sanda, asakura, watanabe}@watanabe.nuie.nagoya-u.ac.jp

概要

並列タスクを矛盾なく効率よく実行させるには、計算機クラスタ環境におけるタスク間の同期処理を検討する必要がある。我々はタスク間の同期処理を同期メッセージの送受信によって実現する。同期メッセージは、同期処理のためだけに送信されるものであり、メッセージが持つデータのサイズは非常に小さい。そこで、このような特徴を持ったメッセージを送信するには、汎用性の高い通信用のライブラリを用いても、無駄な処理が発生し、通信時間が遅延する。そこで本稿では、上記の点を考慮して、計算機クラスタ環境におけるタスク間の同期処理について検討し、同期メッセージを用いた同期処理の実現方法、及びその高速化について考察する。

Evaluation of Synchronization Processing of Parallel Tasks on the Computer Cluster Environment

Keiji Miura, Katsushi Sanda Koichi ASAKURA and Toyohide WATANABE
Department of Information Engineering,
Graduate School of Engineering, Nagoya University
{miura, sanda, asakura, watanabe}@watanabe.nuie.nagoya-u.ac.jp

Abstract

The synchronization processing is very important to execute parallel tasks on the computer-cluster environment correctly. In order to achieve efficient performance, we introduce the synchronization message for synchronization processing. This message is sent between tasks for communication of tasks. In our facility, we achieve the speed-up of synchronization processing by avoiding redundant procssing in normal task communication such as data packing.
In this paper, we investigate synchronization processing between tasks on Computer-Cluster environment, and introduce the speed-up of the synchronization facility with synchronization message.

1 はじめに

近年、高性能化したパーソナル・コンピュータやワークステーションをネットワークで接続して構成された、計算機クラスタ環境 [1] が並列処理プラットホームとして注目されている。我々は、計算機クラスタ環境上で効率のよい並列処理を達成するための並列処理手法について検討している [2][3]。

並列タスクは通常、別々の計算機上で並列に実行される。このとき、並列タスクを順序良く実行させ、正確な計算結果を得るには同期が必要となる。つまり、タスク間でデータを送受信したり、タスクの実行を同調させるためには、必ずタスク間の同期を取らなければならない。効率のよい並列処理には、タスク間での同期を取ることが必須である。

通常の並列計算機では、同期処理の高速化のためのハードウェア資源を有している。同期処理のための専用のハードウェアやネットワークが装備されており、効率よい同期処理が可能となっている。一方、我々が対象とする計算機クラスタ環境は、計算機とネットワークでのみ構成されており、並列処理のための特別なハードウェアは有していない。したがって、計算機クラスタ環境では、タスク間の同期処理はネットワークを介したタスク間通信により実現しなければならない。しかし、タスク間通信に使用されるMPI[4]などのライブラリを用いて同期処理を行なうのは効率的ではない。MPIなどでは任意のタスク間で任意の大きさのデータを送受信可能とするために、様々な付加処理が行なわれる。この付加処理によるオーバヘッドが大きく、同期処理が効率よく行なわれてない。

我々は、計算機クラスタ環境における並列タスク間で効率のよい同期処理を実現するために同期処理のみをつかさどるソフトウェア・パッケージHSSF（High-Speed Synchronization Facility）を提案する。このHSSFは、同期処理の際にタスク間で授受される同期メッセージの性質を考慮した設計となっており、同期メッセージに対しては不必要な付加処理を削減することで高速化を計っている。

本稿は以下のような構成になっている。2章において、タスク間の同期処理についてまとめ、同期メッセージの性質、それに対する高速化について述べる。3章ではHSSFの構成について述べる。4章ではHSSFの効率を評価するために行なった評価実験について述べる。最後に5章においてまとめと今後の課題について述べる。

2 同期メッセージを用いた同期処理

本章では同期処理の際にタスク間で送受信するメッセージとして、同期メッセージを定義し、同期メッセージの送信方法および、MPI[5]を用いた際の同期メッセージの実装における問題点について述べる。

2.1 同期メッセージ

同期処理とは、タスク間の同期をとるための処理であり、これによりデータを受け渡すタイミングや、各タスクの実行開始の時期をタスク自身が把握できる。

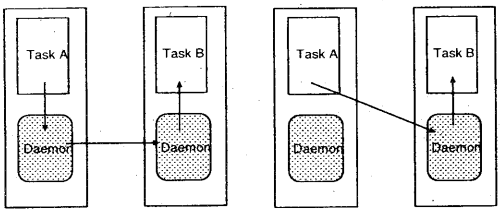
同期メッセージは同期処理のためにタスク間で送受信されるメッセージである。同期メッセージとして送信するデータは、できるだけ小さいことが望ましい。一般に、同期通信は通常の通信処理とは区別して考えられ、同期処理を実行し、タスク間の同期をとった後で、データ通信処理などが行なわれる。ゆえに、同期処理に要する時間はできるだけ短くする必要がある。

同期処理に必要な時間を短縮するには、同期メッセージの送受信にかかるコストをできるだけ小さくする必要がある。同期メッセージは、タスク間の同期処理のためのメッセージであり、メッセージの内容はどのタスクに対するどの変数のための同期メッセージであるか、を表している。したがって、同期メッセージとして送受信されるデータはタスクと変数の識別子のみであり、データの量は非常に小さく、固定長である。我々は、このような同期メッセージの特徴を考慮して、専用の送信機構を設計し、効率のよい同期処理を可能とする。

次節では、これらの前提条件のもとでの同期メッセージの具体的な送信方法について検討する。

2.2 同期メッセージを用いた同期処理の実現

同期メッセージを送信する際に考慮すべき点は、同期メッセージの特徴と、同期メッセージの送信にかかる時間をいかに短縮するかの2点である。まず、前節でも述べたように、通常のタスク間通信と比較すると、同期メッセージは送受信するデータのサイズが小さいことが挙げられる。この送信すべきデータの量が少ないという特徴に注目し、メッセージの送信にかかる時間を短縮する。送信側タスクAと受信側タスクBが図1の(a)のように存在するときに、タスクAからタスクBに同期メッセージを送信する場合を考える。タスクBがタスクAからのメッセージの到着を待っている状態である時、同期メッセージはタスクAからタスクBへのタスク間通信によって送信される。しかし、タスクBが別の



(a) Data Flow of MPI (b) Data Flow of HSSF

図 1: HSSF におけるメッセージの流れ

処理を行なっている状態では、タスク A から送信されたメッセージの内容を確認することができない。そこで受信側のタスク B に受信用のバッファを用意し、送信されたメッセージはデーモンによって受理され、このバッファに蓄える。タスク B は必要に応じてこのバッファを参照する。

受信側タスクは同期メッセージが到着した時点で、次に必要としているデータの生産が終了していることを知ることができるため、メッセージからデータの格納されている仮想共有メモリ空間のアドレスをと取り出し、データを参照することが可能となる。仮に複数のタスクからのデータの到着を待っているような状態であれば、どのタスクから届いたメッセージがどこに格納されているのかをあらかじめ決めておくことで、必要なデータを分散共有メモリから取り出すことができる。

2.3 MPI を用いた同期メッセージ実装の問題点

MPI は、いかなる種類のメッセージにも柔軟に対応ができるようなインターフェースである。ゆえに、データサイズが大きなメッセージや、複数のデータタイプが混在するメッセージの送信にも何ら問題が生じない。一般に MPI では、以下のような処理を送信側タスクが存在する計算機のデーモンで行なっている。送信側タスクは、送信すべきメッセージの内容となるデータをデーモンに渡す。データを受けとったデーモンはそれらのメッセージを通信バッファのサイズに分割し、複数の種類のデータをまとめて送信できるようにパッケージ化する。これらの前処理がなされた後、デーモンは受信側タスクが存在する計算機のデーモンにメッセージを送信する。メッセージを受けとった受信側のデーモンは、前処理によって変換されたメッセージを、変換される前のもとの形式に戻す。

しかし、これらの処理は同期メッセージの送信には冗長である。同期メッセージは固定長でサイズも非常に小さいという特徴がある。ゆえに、上記のような前処理は不要である。すなわち、同期メッセージの特徴を考えると、同期メッセージはデータ・サイズが小さいので通信バッファのサイズに分割する必要はなく、またパッケージ化も必要ない。したがって、同期メッセージの送信には、上記の前処理を省略することで、効率のよい送信処理が実現できる。

我々は、同期メッセージの特徴を考慮した、同期メッセージ専用の通信機構 HSSF(High-Speed Synchronization Facility) を提案する。アプローチとしては、データのパッケージ化や、サイズの大きなデータのバッファサイズへの分割など、同期メッセージには不要である前処理の削除と、通信経路の単純化の 2 点をとりあげる。前処理の簡略化については先に述べたように、同期メッセージの送信にはデータの分割、パッケージ化などの前処理を省略することで効率のよい送信処理を実現する。

また、前処理を削除することで送信側のデーモンで行なわれる処理が減少するので、送信側デーモンの負担が軽くなる。そこで、送信側デーモンが今まで行なっていた処理を送信側タスクによって実行し、受信側デーモンへ送信側タスクを介することなく、直接送信する手法をとる。以上をまとめると、HSSF でのメッセージの流れは、図 1の (b) のようになる。これにより、同期メッセージ通信を高速化できる。

3 HSSF の実装方法

本章では、HSSF を実装する際の諸問題について述べる。

3.1 共有メモリバッファによるデーモン・タスク間通信

前章で述べたように、同期メッセージは送信側タスクからデーモンに送信される。したがって、デー

モンで受信された同期メッセージは受信側タスクに送信されなければならない。同計算機内のデーモンとタスク間の同期メッセージ通信の高速化のため、我々は、OSで提供されている共有メモリ機構を活用する。共有メモリ機構により、デーモンとタスクで同一のメモリ空間を共有する。デーモンは受信した同期メッセージを共有メモリ空間に保持することにより、タスクが同期メッセージを参照可能となっている。したがって、デーモンとタスク間で通信処理を陽に行なうことなく、タスクが同期メッセージを受信することができ、計算機内出野同期メッセージ授受の高速化を計ることができる。

従来であればデーモン・タスク間においてもう一度通信が発生するが、この通信処理を共有メモリ空間へのread, writeに変換することで、同期メッセージ通信が高速化される。具体的には、デーモンがメッセージを受信すると共有メモリバッファへの書込みが発生する。バッファへの書込みが発生すると、バッファを参照している、または参照しようとしているタスクの実行を制御する。そこで、バッファへの更新・参照を制御するために、セマフォを用いたメモリ空間の排他制御が必要となる。

これらの点を踏まえて、我々は同期メッセージ専用の送信機構を提案する。

3.2 共有メモリバッファの分割・制御

共有メモリバッファを用いたデーモン・タスク間通信には、セマフォを用いて排他制御する必要がある。現段階では、共有メモリ全体を一つのセマフォによって制御している。しかし、デーモンからの書込み要求が発生する度に、全てのタスクの参照要求を制御していては並列実行性が失われるとともに、タスクの実行効率が低下する。我々は、共有メモリバッファをいくつかの大きさに分割し、セマフォの制御を各々の共有メモリバッファ単位で行なう。前提条件として、共有メモリバッファのどこにデータを書込むかは既に決められているとする。ゆえに、タスクが参照している共有メモリバッファへの書込みが発生したときには、その共有メモリバッファを参照しているタスクに対してのみセマフォによる実行制御を行なう。図2に例を挙げて、共有メモリバッファの分割・制御の概要を説明する。図2(a)では共有メモリバッファが一つ存在しており、全てのタスクはこの共有メモリバッファを参照する。デーモンに同期メッセージが到着すると、共有メモリバッファで受信したメッセージの書込みが発生する。このとき、共有メモリを参照しようとしているタスクは全て、実行中断状態になる。しかし、図2(b)のように共有メモリバッファが複数に分割されていれば、例えばメッセージの到着によって共有メモリバッファへの書込みが発生したとしても、書込みが発生したバッファに参照するタスクの実行は中断されるが、それ以外のバッファには参照可能である。図2(b)では、shared memory buffer 4への書込みが発生しているとき、Task 1、Task 2の処理が中断されないことを示している。

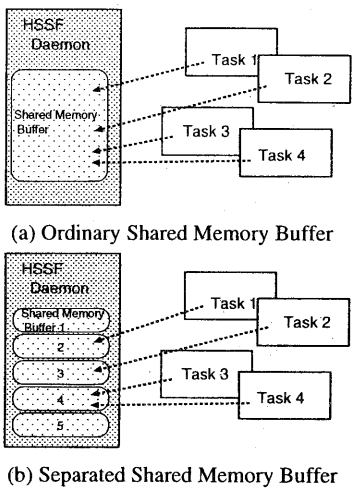


図 2: 共有メモリバッファの分割・制御方法

このような手法により、書込みが発生した共有メモリバッファ以外に参照するタスクは、実行を中断されることなく、処理を継続することができる。

4 評価実験

我々は、これまでにHSSFの概要と特徴について述べた。HSSFの性能を評価するために、同期メッセージの送信時間を一般的な通信機構の1つであるPVM(Parellel Virtual Machine)[5]と比較し、次の2種類の実験を行なった。

4.1 実験の内容

今回の実験では、同期メッセージの送信時間をHSSFとPVMで比較した。PVMは計算機クラスタ環境で実際のデータ送受信に用いられている機構である。

まず、実験1では受信側タスクが同期メッセージの到着を待っている状態で、同期メッセージの送信時間を測定した。図3(a)のように送信側タスクによって送られたメッセージを受信側デーモンが受け取った後、デーモンがすぐに受信タスクにメッセージを送るまでの時間を測定した。この状態では受信側タスクは何ら処理を行なわず、メッセージの到着を待ち続けている。

実験2では、受信側タスクによって既に受け取られているメッセージをタスクが参照し、メッセージの内容を確認するまでの時間を測定した。このとき、図3(b)のように、受信側タスクは別の処理を行なっている状態で、HSSFであればその処理が終り次第すぐに共有メモリバッファにアクセスする。また、PVMでは受信デーモンから受信タスクへのメッセージ送信時間を測定した。

4.2 結果及び考察

実験1及び実験2の結果は次のようになった。

表 1: 実験1及び実験2の結果

	実験 1	実験 2
PVM	1356.5μs	127.8μs
HSSF	743.4μs	1.6μs

実験にはSunのUltra60及びUltra30を使用し、通信速度は100Mイーサネットである。

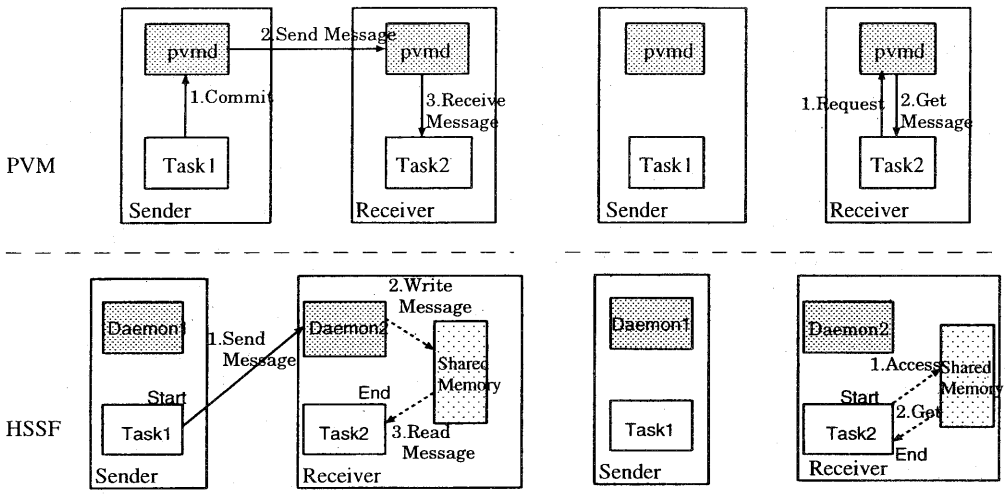
また、これとは別に同期メッセージとして実際に送信されるであろうデータを想定して、共有メモリバッファへの書込み時間を測定した。上記と同様の環境において実験したところ、共有メモリバッファへの同期メッセージの書込みに要する時間は10.5μsであった。

今回の実験では同期メッセージの送信時間を測定して、HSSFとPVMを比較した。実験1では受信側タスクが待ち状態であり、送信されてからすぐに受信されるまでの時間を計測した。この結果、PVMに比べて、HSSFの方が約2倍高速であった。実験2では、デーモンによって既に受理されているメッセージへアクセスするまでの時間を計測した。これは、共有メモリバッファへのアクセスと、タスク間通信との比較であり、期待通りの結果が得られた。しかし、実験1のような状態が頻繁に発生する状況では、HSSFを用いることによる並列処理全体の実行時間の短縮はそれほど望めない。実験2程の差が出れば、実行時間への影響が現れると考えられる。ゆえに、今回の実験結果から考慮すると、スケジューリングの段階において、実験1の状態のようにタスクが待たされることが極めて少ない状況をつくり出すことによって実行時間を短縮できる。

5 おわりに

本稿では、計算機クラスタを用いた並列処理環境における同期処理方法について述べた。計算機クラスタ環境は、通常の並列計算機と比較して計算機間通信に要する時間が長いので、それを改善するために、同期メッセージという概念を導入し、同期メッセージ通信によって同期処理を実現する方法を報告した。さらに、同期メッセージの特徴を考慮して、同期メッセージをより高速に通信する機構として、HSSFを提案した。また、HSSFにおける共有メモリバッファの制御によるタスクの待ち時間を軽減するために、共有メモリバッファを分割して管理する手法を提案した。最後にHSSFの性能を評価するために、PVMとHSSFで同期メッセージの送信時間を比較し、HSSFの優位性を示した。

今後の課題としては、共有メモリバッファの分割制御の具体的な設計を検討し、その有効性を評価する必要がある。また、今回の評価実験では1対1のタスク間通信に限定した実験であったが、複数のタスクが同一計算機内に存在するという状況も考えられるので、1対多通信でのHSSFの性能を評価しなければならない。



(a) Experiment 1　　(b) Experiment 2

図 3: 実験方法

謝辞

日頃からご鞭撻、ご教授いただく中京大学情報科学研究科・福村晃夫教授、本学工学研究科・稲垣康善教授、鳥脇純一郎教授に深謝するとともに、日々討論いただく研究室の皆様に感謝します。

参考文献

[1] 平木敬, 丹羽純平, 松本尚: "分散共有メモリに基づく計算機クラスタ", 情報処理, Vol. 39, No. 11, pp. 1078–1083 (1996).

[2] 朝倉宏一, 渡邉豊英: "ワークステーション・クラスタ環境における並列化コンパイラの構成", 電気学会論文誌 C, Vol. 118-C, No. 4, pp. 558–568 (1998).

[3] 三田勝史, 朝倉宏一, 渡邉豊英: "配列データ分割による漸進的並列処理手法の適用", 並列処理シンポジウム JSPP '98, p. 147 (1998).

[4] S. Lumetta and D. Culler: "Managing Concurrent Access for Shared Memory Active Messages", *Proceedings of the International Parallel Processing Symposium*, pp. 272–278 (1998).

[5] A.Geist, A.Beguelin, H.Dongarra, W.Jiang, R.Manchek and V.Aunderam: "PVM:Paralle Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing", *MIT Press.* (1994).