汎用スケーラブル OS *SSS–CORE* のカーネル構成について

松　本　尚 †　　渦　原　茂 ‡
竹岡　尚三 ‡　　平　木　敬 †

SSS–CORE はユーザによる並列および分散プログラムの最適化の支援と、システムに共有メモリ空間としての統一的なビューを導入することを目的とした汎用スケーラブルオペレーティングシステムである。特殊な通信または共有メモリ用のハードウェアが存在しなくてもプロセッサのメモリ管理機構を最大限に活用することで共有メモリ空間としてのビューを低コストで提供する。本稿では SSS–CORE Ver.1.1 の設計方式およびカーネル内マルチタスキング、軽量システムコール、高機能メモリベース通信について述べる。そして、実機上の SSS–CORE Ver.1.1 に関して、基本機能の性能を報告する。

A general-purpose scalable OS *SSS–CORE* — Structure of the kernel —

Takashi MATSUMOTO† , Shigeru UZUHARA‡ ,
Shozo TAKEOKA‡ and Kei HIRAKI†

SSS–CORE is a general purpose scalable operating system for NUMA parallel distributed systems. It provides very efficient multi-tasking environment with timesharing and space partitioning. Furthermore, it tries to allow each parallel application to achieve maximum performance by cooperation of user and kernel level resource allocation and scheduling and by offering low-latency high-throughput memory based communication facilities. SSS–CORE also provides a low-overhead mechanism that allows information transfer between kernel and user level. In this paper we describe the structure of the SSS–CORE kernel, light-weight system-calls, memory-based FIFOs and memory-based signals. Finally we show the basic performance of the SSS–CORE Ver.1.1 system which has been developed from scratch.

1. はじめに

並列計算機も実用化時代に入り、多くの商用マシンが開発され実務に供されている。しかし、多くのマシンは複数の独立したジョブを高速に処理するサーバ機として使用されており、汎用の並列計算環境つまりマルチジョブ / マルチユーザ環境における高効率の並列処理はまだ実用レベルではない。一方、 LAN 用ネットワークの高速化に伴って、高速ネットワークで複数台のマシンを結合したワークステーションクラスタ (Network of Workstations: NOW) やサーバ機クラスタが注目を集めるようになってきている。現状ではまだ汎用の LAN 用ネットワークの性能が十分でない面があるが、データベース共有やファイル共有の分散処理程度であれば、並列計算機に取って代わりそうな勢いである。

汎用並列計算機や NOW における次なるチャレンジとしては、現在の分散処理環境と同等の汎用環境を維持しつつ、スケーラビリティと並列処理による高速性を安価に実現することである。汎用性と並列処理性能の高さから集中共有メモリやハードウェア分散共有メモリを持つ商用並列計算機の発売が相次いでいる。しかし、専用マシンは量産効果がなく非常に高価である。やはり、NOW のような量産効果を活かし安価に汎用高性能並列計算機を開発する方向を模索する必要がある。

このような背景の下で、我々は 1994 年より汎用スケーラブルオペレーティングシステム **SSS–CORE** [1] の開発を開始した。 SSS–CORE は並列アプリケーションと協調動作することにより、効率を極力落すことなくマルチユーザ / マルチジョブの汎用環境を実現する特殊なハードウェアを仮定しない分散メモリ型並列計算機および NOW を対象とした汎用オペレーティングシステム (汎用 OS) である。 1996 年には LAN 用の汎用ネットワークインタフェースを使ったユーザレベル高速通信同期機構であるソフトウェアメモリベース通信 (MBCF: Memory-Based Communication Facilities) [2] [3] を開発した。 MBCF は細粒度メモリベース通信用のハードウェア機構である Memory-Based Processor (MBP) [4] の機能を、通信の粒度を大きくすることにより汎用ハードウェアとシステムソフトウェアで効率良く実現したものである。さらに、分散共有メモリをユーザレベルのキャッシュエミュレーションコー

† 東京大学 大学院理学系研究科 情報科学専攻, Department of Information Science, University of Tokyo
†† 株式会社 アックス, AXE Inc.

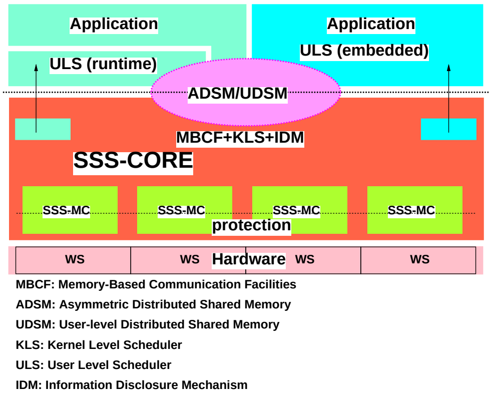

図1　SSS–CORE の機能構成

ドの挿入と通信回数ならびに通信量の削減を最適化コンパイラで実現するという新しいアプローチを考案した [5] [6]。これにより、分散共有メモリ実現に関わる通信同期トラフィックの粒度を大きくすることに成功し、MBCF のメモリベース通信機能とそれらのトラフィックの整合性の良さもあいまって、分散共有メモリ用ハードウェアを持たない NOW 上でも共有メモリベースのプログラムが効率良く並列実行可能となった [7] [8]。

SSS–CORE の通信同期機構として考案された MBCF は同一のハードウェア環境ではメッセージパッシング型の通信機構よりも本質的に性能が優れている [3]。また、保護および仮想化の能力も従来の通信プロトコルに優るとも劣らない。このため、 MBCF は分散アプリケーション用の高性能通信機構としても有望である。特に、多くの通信を同時に処理するサーバマシンにとっては、通信処理の負荷が減るメリットは大きい。 MBCF の優れた特性を活かすため、 SSS–CORE はターゲットアプリケーションを単なる「並列処理」から「分散並列処理」に広げ、今後はより一般的な汎用スケーラブルオペレーティングシステムを指向して開発を進めていく。

本稿では、これまでの論文では十分に説明されていなかった SSS–CORE の構成、機能、基本性能、高機能 MBCF の機能および性能 を報告する。

2.　SSS–CORE の設計思想

2.1　ユーザ最適化の支援

SSS–CORE の開発に先駆けて、我々は 1989 年よりすべての CPU から等距離でアクセス可能な集中共有メモリを持つ Uniform Memory Access（UMA）型並列計算機を対象にして、ユーザによる最適化を支援する汎用オペレーティングシステム核 SS–CORE [9] の研究開発を行った。 SS–CORE の研究においては議論を単純化するために、 UMA 型並列計算機に対象を限定し、実プロセッサの資源管理に的を絞った研究を行った。そして、 ユーザ最適化の支援のために、保護や仮想化やタスク間の公平性を OS が管理できる範囲で、可能な限り資源情報および資源管理の権限をユーザに解放するという基本方針を確立し、 SSS–CORE に受け継いでいる。 SSS–CORE がユーザタスクでの使用を前提とするカーネル / ユーザの資源割当状況や実行状況によるスケジューリングオプションつきのスピンウェイトである Snoopy Spin Wait (SS-wait☆) [10] も SSS–CORE の研究において開発された。

2.2　機 能 構 成

図 1に SSS–CORE の機能構成図を示す。各ノードに SSS–CORE の核となり、ノードの資源管理と資源保護を担当する SSS–MC（Micro Core）が常駐している。SSS–MC は単体で単一ノードの資源保護と資源管理を行うマルチタスクのオペレーティングシステムである。その SSS–MC の機能を使って、複数ノードに跨る MBCF やグローバルな資源スケジューラ（KLS）や資源管理情報開示機構（IDM）が提供されている。コンパイラ支援型分散共有メモリ機構（ADSM/UDSM） [5] [6] は、最適化コンパイラがユーザレベルのキャッシュエミュレーションコードと MBCF およびメモリ管理機構のユーザインタフェースを利用するコードを生成することにより実現され、図 1ではユーザ / カーネルの両境界に跨る機構として図示されている。ユーザアプリケーションは MBCF もしくは ADSM/UDSM を用いることにより NOW 全体に跨るグローバルな共有メモリ空間を低コストで使用することができる。

2.3　SSS–MC (Micro Core)

SSS–MC は、設計方針として「コンパクトなカーネル」を基本としているが、従来より提案されているマイクロカーネル [11] [12] [13] とは設計方針が異なる。つまり、性能を犠牲にしてまでカーネルのコンパクトさやカーネルインタフェースの統一を追求するようなことはしない。例えば、従来のマイクロカーネルでは、ポートやメッセージといったなんらかの抽象化を含んでおり、それらはソフトウェア的に高機能なサービスを提供しているが、その抽象化のためにオーバヘッドが増加する。そして、マイクロカーネルとのインターフェースが非常にプリミティブな操作に限定されているため、カーネル呼び出しの回数が非常に多い。これらのことが性能の問題になっている。それに対し、 SSS-CORE および SSS–MC ではハードウエアが提供する機能を、必要最低限の保護と仮想化を守った上で、できるだけ直接的にサービスとして見せることを基本としている。また、SSS–CORE のサービスとして重要であり、 SSS–MC と密に協調して実現されるべき高レベル機能は、別タスクとしてカーネル空間内に実装されている。

SSS–CORE は最適化された高速な OS サービスの提供☆☆と、コンパイラ / アプリケーションプログラマによる動的 / 静的最適化を積極的に支援することを目標としている。そして、 SSS–CORE が推奨し支援するモデル（MBCF や ADSM/UDSM）に従った並列アプリケー

☆ SS–CORE の名前の由来となる同期方式
☆☆ ただし、人間の操作が律速段階となるユーザインタラクション関連の OS サービス、および互換性のために用意された通信相手が律速段階となる従来型通信サービスは高速化の対象とはしていない。

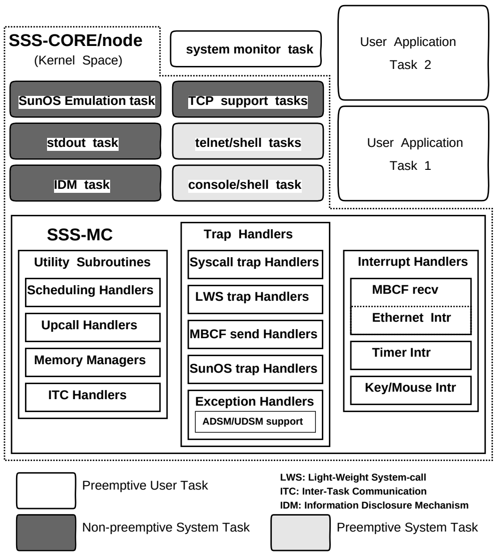

図2　ノード上の SSS–CORE/SSS–MC の構成

ションコード（基本的に最適化コンパイラが出力）に対して、ソースコードの書き直しを行うことなしにマシン環境の増強による性能向上をもたらすことのできるオペレーティングシステムを目指している。

2.4　メモリベースの OS アーキテクチャ

ユーザアプリケーションの実行時間の最適化を最大目標としつつも、ユーザにノードを跨る資源やカーネル内の情報を統一的に見せるために、SSS–CORE は共有メモリビューを全面的に採用している。実行環境は通常の分散環境であるが、保護と仮想化がほとんどオーバヘッドなしに実現されているプロセッサのメモリ管理機構を活用して、付加的な保護や仮想化のオーバヘッドをほとんど導入せずに共有メモリ環境を実現する。SSS–CORE のメモリシステムの活用例として以下の項目が挙げられる。

- 中粒度メモリ操作としてシステム全領域の通信同期が定義されている（MBCF）
- カーネルの資源管理情報でユーザに有益な情報はノード内外を問わず、ユーザタスクからローカルに読み出し可能（IDM）
- 最適化コンパイラによって実現されるユーザレベル分散共有メモリ機構を支援（ADSM/UDSM）
- 自由にノード内に共有メモリを生成できる

3.　SSS–CORE Ver.1.1

SSS–CORE/NOW Ver.1.1 の各ノードの構成を図2に示す。各ノードにはノード OS 核である SSS–MC Ver.2.1 が常駐している。SSS–MC はソフトウェアトラップハンドラ群と割り込みハンドラ群からなり、SSS–MC とは独立にカーネル空間内に高機能のサービスを提供するためのカーネル内タスク群が存在する。この意味で、ノードのカーネル機構はマルチタスク構成となっている。

トラップハンドラはトラップの入口において排他性が保証されており、一旦トラップハンドラを実行開始したらハンドラを終了するまで実行が中断されることはない。排他制御コード☆の使用回数を抑え、リエントラント化するためのオーバヘッドを削減している。ただし、トラップハンドラ自体で実現される機能はノンブロッキングな操作のみに限定されており、トラップハンドラの実行は短時間で終了する。I/O デバイスの終了や通信相手からの反応を待つ必要のあるシステムコールは、トラップハンドラで要求を受け付けて、実際の処理は MBCF やタスク間通信（ITC）やカーネル内メモリを介してカーネル内の別タスクに委託して処理される。なお、カーネル内タスクからもシステムコールによってトラップハンドラのサービスを利用可能である。

割り込みハンドラはコンパクトに作られていて、基本的に I/O デバイスが資源を即時再利用可能なようにレジスタ内容や専用データバッファを主記憶上にバッファリングし、カーネルへ割り込み発生を主記憶上のフラグによって通知する方針を採っている。高レベルの処理はフラグを検知したカーネル内タスクまたは SSS–MC のトラップハンドラによって処理される。ただし、MBCF では、主記憶上にバッファリングする際に、通信同期対象タスクの対象アドレスを直接操作することにより高速化を達成している。

性能重視のため、機能的には複数のノードおよび SSS–MC を横断する MBCF の機能は SSS–MC 内部の割り込みハンドラとトラップハンドラで直接実現されている。この事実からも判るように、機能的な階層構造を直接実装上の階層構造とする旧来の手法は採らず、可能な限り高性能な実装を行うことを目指している。また、図 2内にはデバイスドライバが明示されていないのは、標準化された何種類かのデバイス形式（例えば UNIX のストリームデバイスとブロックデバイス）に合わせてハンドラや機能タスクが作られているわけではないからである。ハンドラや機能タスク毎に性能を落さない範囲でデバイス依存部分とデバイス非依存部のインタフェースを決めてプログラムが書かれている。このため、同一デバイスを複数のハンドラやタスクで使用する場合は、基本的には複数のデバイス依存コードとそのコード間の整合性を保証するコードを書く。

3.1　トラップハンドラ

SSS–MC は現在 4 種類のシステムコール用トラップハンドラと例外ハンドラ群を持っている。以下にその内容を簡単に説明する。

- システムコール用トラップハンドラ
  SSS–CORE の通常のシステムコールのためのハンドラであり、軽量システムコールまたは MBCF 要求システムコールとは異なり、呼び出したタスクのコンテクストを完全に退避してから処理を行う。このため、タスクスケジューリング（コンテクス

☆ 共有マルチプロセッサ構成のノードでは通常外部アクセスを伴うコストの大きな不可分操作命令を含む。

ト切替）を伴う可能性のある機能が実現できる。SSS–CORE Ver.1.1 がユーザにサポートしているシステムコールの種別と数は以下の通りである。なお、ノード内通信同期は SSS–MC が提供する同一ノード内のタスク間通信キュー、セマフォー、イベント機能である。これらの機能はブロッキングタイプとしてもノンブロッキング（ポーリング）タイプとしても使用できる。

表1　システムコールの数

システムコール種別	数
タスク制御	23
メモリ管理 / 操作	11
MBCF 管理	16
ノード内通信同期	15
ノード内オブジェクト管理	4
その他	12
計	81

- 軽量システムコール用トラップハンドラ
通常のシステムコールでは、その実行中にコンテクストを切替える可能性があるため、トラップの入口で全コンテクストをメモリに退避している。しかし、コンテクスト切替の可能性がない機能で頻度が高い物に関しては、この実装はオーバヘッドが大きい。高頻度でノンブロッキング機能専用のシステムコールとして用意されたのが軽量システムコール用トラップハンドラである。トラップの入口では必要最低限のコンテクストの退避しか行わない。SSS–CORE Ver.1.1 がサポートしている軽量システムコールは 11 個である。重要な物には、Memory-Based FIFO の構造体初期化と書き込まれた FIFO データの読み出し、Memory-Based Siganal の構造体初期化と連続して到着した Signal の受け取り、ユーザレベル分散共有メモリを支援するためのページ無効化と有効化等がある。このように通信同期で頻繁に使われるシステム機能に関しては、軽量システムコールによる実装が主体となる。
- MBCF 要求システムコール用トラップハンドラ
MBCF の発行は、専用のソフトウェアトラップ番号を割り当てて、オーバヘッドを極限まで削減している。MBCF 発行要求はノンブロッキングであることが保証されているため、軽量システムコール用トラップハンドラと同じオーバヘッド削減手法が適用されている。
SSS–CORE Ver.1.1 がサポートしている MBCF コマンドは 15 種類である。さらに、これらのコマンドはステータス返送の有無、エラー情報返送の有無、エラスティックメモリバリア使用の有無、後述する Memory-Based Signal に関する各種バリエーションによって機能が修飾されて、非常に多くのメモリベース通信機能が提供されている。
- SunOS システムコール用トラップハンドラ
現状の SSS–CORE は開発途上であり、ファイルシステムを持っていない。また、タスクの実行用ライブラリ (libc) も完備しているとは言い難い。このため、ファイルシステムを SunOS マシンに実行を肩代りしてもらう目的と、SunOS 用の libc を使ってタスクの実行オブジェクトを生成可能とする目的で SSS–CORE は SunOS のシステムコールをサポートしている。ただし、SunOS システムコールの仕様は大きいため、SSS–CORE Ver.1.1 では、メモリ獲得（sbrk）、プロセス制御の一部、ファイル入出力関係、IOCTL 関係のシステムコールのサポートに留まっている。メモリ獲得およびプロセス制御のシステムコールは本ハンドラで直接実現され、SunOS マシンとの通信が必要となるファイル入出力関係と IOCTL 関係のシステムコールは SunOS Emulation タスクに処理が委託される。
- 例外ハンドラ
ページフォルト、レジスタウィンドウのオーバフローおよびアンダーフロー、浮動小数点演算例外等の例外ハンドラが提供されている。SSS–CORE Ver.1.1 では、デマンドページングによる二次記憶は提供されていないが、ユーザレベル分散共有メモリ実現のためにユーザレベルページャをタスク単位で設定できる。この機能とメモリ管理用システムコールを組み合わせれば、ユーザレベルの二次記憶の実現が可能である。

3.2　タスクの種類

SSS–MC は以下の三種類のスケジューリング属性を持つタスクをスケジューリングする。

- プリエンプティブユーザタスク（PUsr）
タイムスライスでプリエンプトされ、プロセッサはユーザモードで走行する、通常のユーザアプリケーション用のタスク。
- ノンプリエンプティブシステムタスク（NSys）
タイムスライスでプリエンプトされない、プロセッサはスーパバイザモードで走行する、カーネルサービス記述用のタスク。スケジューリングは明示的なコンテクスト切替もしくはコンテクスト切替を伴うシステムコール時のみであるため、特別な処理をしなくてもクリティカルセクションが確保できる。ただし、長い時間走行する NSys タスクは弊害が多いため、短時間でコンテクスト切替の機会があるように OS 開発者によって記述される。
- プリエンプティブシステムタスク（PSys）
タイムスライスでプリエンプトされるが、プロセッサはスーパバイザモードで走行する、ユーザアプリケーションに近いカーネルサービス記述用のタスク。SSS–CORE Ver.1.1 では shell プログラムは PSys タスクとしてカーネル内に書かれている。shell は非常に多機能であるため、カーネル外で実現するためには非常に多種類のシステムコールを用意する必要がある。この手間を軽減するために、SSS–CORE ではカーネル内に shell 機能を取り込んでいる。ただし、shell 上のバッチコマンドの実行等により、特定のユーザが長時間プロセッサを占

有しないためにプリエンプティブなタスクとして実装されている。

3.3 カーネル空間内タスク

カーネル空間内タスクはカーネル内のデータ構造を低コストでアクセス可能にするため、スタック領域以外はアドレス空間を共有している。カーネル内タスクはスーパバイザ（カーネル）モードで実行され、プロセッサの特権命令を使用して I/O デバイスやメモリ管理機構を直接操作可能である。カーネル内タスクは他のタスクと同様に SSS–MC のスケジューラによってスケジューリングされる。 SSS–CORE Ver.1.1 はカーネル空間内タスクとして以下のタスクを実装している。

- TCP support tasks
  複数のタスク群で TCP のプロトコルを実現している。 NSys タイプのタスクを使用することでタスク間の排他制御の問題がクリアされている。
- console/shell task
  ノードのコンソール画面内の端末ウィンドウを介してユーザ端末を実現するためのタスク。カーネル内に組み込まれたシェル機能をもっており、コマンド行編集、コマンドロード、コマンドのリトリーブ、デバッグサポート、ユーザ環境変数、複数コマンドのバッチ実行等をサポートする。前述のように PSys タイプのタスクで実現されている。
- telnet/shell tasks
  一つのリモート端末に対して一つのカーネル空間内の telnet/shell タスクが対応して、 telnet によるリモート端末機能を提供する。機能は基本的に console/shell task と同じであり、 PSys タイプのタスクで実現されている。また、 telnet 通信に対するバッファリング機能も提供する。
- SunOS Emulation task
  SunOS トラップハンドラが検出した SunOS システムコールの内、ノード内で解決できない処理を担当するためのタスク。メモリ制御やプロセス（タスク）制御や標準入出力に関するシステムコールはトラップハンドラで実現される。他の SunOS 上の I/O に関するシステムコールは、本タスクが外部の SunOS マシン上のシステムコールサーバプロセスに TCP/IP 通信を介して処理を委託することで実現される。 SunOS マシンでの実行結果は TCP/IP 通信でこのタスクに返されてシステムコールを終了させる。処理委託期間中はシステムコールを発行したタスクは実行を中断した状態で待機する。本タスクは NSys タイプで実現されている。
- stdout task
  複数のタスクから同一端末への文字出力に対してバッファリングとある程度の可読性を確保するためのタスク。バッファリングのためにプログラムの意図しないところ（例えば行の途中）で、複数のストリームが混ざるようなことがないように制御する。 OS 開発用のカーネルからのメッセージとユーザタスクからのメッセージを制御して可読性を上げるために作られた。本タスクは NSys タイプで実現されている。
- IDM task
  SSS–CORE の情報開示機構（IDM: Information Disclosure Mechanism）を実現するためのタスク。同一ノード内の情報開示機構はメモリのマッピングのみで実現されるので、特別なタスクは必要とされない。 SSS–CORE では他ノードの資源割当状況や使用状況をユーザタスクの動的最適化の判断材料として提供する。このために IDM タスクはノード間で資源情報を MBCF により常時交換している（交換頻度はタイムスライスの 10 倍程度）。ただし、 SSS–CORE Ver.1.1 ではノードを跨る情報開示機能に関してはまだ完成していない。本タスクは NSys タイプで実現される予定。

4. 高機能メモリベース通信同期

本節では MBCF の中でもユーザに仮想化された高機能の同期手段を提供する Memory-Based FIFO [4] と Memory-Based Signal [4] の SSS–CORE Ver.1.1 における機能を説明する。

4.1 Memory-Based FIFO (MBCF_FIFO)

MBCF_FIFO はユーザレベルの FIFO キューにデータを登録する MBCF コマンドである。 MBCF ではユーザはメモリが許す限り任意個の独立した FIFO キューを任意アドレス（性能維持のためアラインメント制約のみ存在）に設定することができる。 MBCF_FIFO コマンドの発行タスクはデータを格納するアドレスを直接指定するのではなく、対象タスクにおいてキューを管理している構造体のアドレスを MBCF の操作対象として指定する。このキュー構造体の中にはデータ用のバッファ領域の先頭と末尾、現在のキューの書き込みポインタと読み出しポインタ、制御フラグが登録されている。この構造体はセキュリティ（MBCF 書き込み等で破壊されない）のために read-only 属性のメモリ作成可能であり、システムコールによって初期化される。 MBCF_FIFO コマンドパケットが到着すると FIFO 構造体が読み出されて指定されたメモリ領域にデータ格納後、構造体のポインタ等が更新される。これらの一連の処理は操作対象ノードでローカルかつ不可分に行われるため、安全かつ高速である。ポインタ操作の不可分性を保証するために、タスクが FIFO を読み出す場合には前出の軽量システムコールの MBCF_FIFO 読み出し手続きを使用する。

キューのバッファが full の場合はコマンドがキャンセルされる。ステータス返送のオプションが MBCF_FIFO コマンドに付加されていれば、コマンドキャンセル（もしくは登録成功）のステータスが発行元のタスクの発行時に指定されたアドレスに MBCF 書き込みで返送される。コマンドキャンセルの可能性があるためこのコマンドは同一タスクから同一キューへのデータ登録の登録順序を保証することができない。一回のデータ登録毎にステータス返送によって登録確認を

とることにより順序を保証できるが、これでは性能が向上できない。順序保証がなされた eager な FIFO 登録を可能にするためのコマンドとして MBCF_FIFOe が SSS–CORE Ver.1.1 には提供されている。

4.2 MBCF_FIFOe (MBCF_FIFO Eager)

MBCF_FIFOe は二個のコマンドから構成される。これらのコマンドは順序保証がなされ eager な FIFO 登録を実現するために使用される。 MBCF_FIFOe の二つのコマンドは MBCF_FIFO コマンド と同じ構造体を使用する。 MBCF_FIFOe と MBCF_FIFO の差はバッファ full の場合の動作のみである。

MBCF_FIFOe_NORMAL は MBCF_FIFOe において通常のデータ登録に用いられるコマンドである。バッファ溢れにより一度 MBCF_FIFOe_NORMAL コマンドのキャンセルが起こると、後続する同一 FIFO への MBCF_FIFOe_NORMAL コマンドはすべて、たとえバッファに登録余地が生まれていても、キャンセルされる。 **MBCF_FIFOe_RETRY** コマンドは、停止している FIFO キューに新たなデータ受け付け余地があれば、その FIFO キューを再開させる。

本 MBCF_FIFOe 方式はコマンド発行元が発送したパケットの管理とデータのコピーを手元に保存していることが前提となっている。対象がバッファ溢れを起こすと、 FIFO が停止状態になったというフラグを立てる。そして通信元に原因となったパケットの ID と、 FIFO が停止状態に遷移したことを示すレポートを返す。後続の MBCF_FIFOe_NORMAL で eager に送られたパケットも必ずキャンセルされる。状態遷移のレポートを受けた発送元はこれらのキャンセルされたパケットを手元のコピーを使って正しい順序で再発送する。その際、再発送する先頭のパケットは MBCF_FIFOe_RETRY で送る。これが受理されると、 FIFO の停止状態を示すフラグを降ろす。後続のパケットは、以前 eager に送っていても受理されていないことが保証されているので再発送する (複雑なチェックを行なわなくても 2 重に受理されることはない)。

もし、通信システムが発行元にパケットコピーを必要としない方式であれば、この MBCF_FIFOe 方式はキャンセルを示すレポートに元のパケットを添付して返送する方式で実装される。このことは本フロー制御方式が “return-to-sender” 方式 [14] の拡張になっていることを示している。従来の “return-to-sender” 方式はパケットの到着を保証するのみで、同一発送元からのパケット到着順序まで保証することができなかった。

4.3 Memory-Based Signal (MBCF_SIGNAL)

MBCF_SIGNAL コマンドは、対象タスクの実行権限で対象タスク内で定義されているプログラムを対象ノード内で起動する MBCF コマンドである。プログラム起動の機構は UNIX シグナルと類似であり、起動されるプログラムは対象タスクのスケジュール期間に現在実行中のタスクコンテクストに優先して走行する (非同期割り込みによる実行)。そして、起動プログラムからリターンすると中断されたタスクコンテストが実行を再開する。

保護と仮想化のために MBCF_SIGNAL コマンドは対象タスクのメモリ内の MBCF_SIGNAL 構造体を通信対象として発行される。この構造体は FIFO 構造体と同様に任意の場所に任意個数設定することができる。MBCF_SIGNAL 構造体は以下の内容を含んでいる。

- 三種類の副コマンド
  （起動タイプ、パラメータ格納タイプ、起動条件）
- 起動プログラムのエントリポイント
- データバッファ / キューへのポインタ群

MBCF_SIGNAL 構造体のこれらのパラメータが機能に大きなバリエーションを与えている。

保護とセキュリティのために、対象タスク（コマンド受信）側が MBCF_SIGNAL 構造体を設定し、MBCF_SIGNAL コマンドの機能を制約することができる。 MBCF_SIGNAL 構造体の中の情報は重要かつ危険なので、構造体のメモリ領域は read-only に設定されるべきである。

起動タイプ副コマンドは起動するプログラムを以下の三種類から選択する。

(1) User-Level Scheduler (ULS)
MBCF_SIGNAL 構造体内に設定されているユーザレベルスケジューリングを実現するプログラム。ただし、必ずしもスケジューリングプログラムには限定されない。 DSP の場合とデータ引き渡し方法が若干異なり、パケットで運ばれた引数データ以外に起動すべきプログラムのエントリポイントがデータとして ULS に引き渡される。

(2) Destination-Specified Program (DSP)
MBCF_SIGNAL 構造体内に設定されている対象側指定のプログラム。 ULS の場合とデータ引き渡し方法が若干異なり、プログラムのエントリポイントの引き渡しは行われない。

(3) Source-Specified Program (SSP)
MBCF_SIGNAL コマンドパケット内に指定されているコマンド発行側指定のプログラム。セキュリティ上は非常に危険であり、並列処理を行うタスク以外では SSP 起動は通常指定されない。本副コマンドの指定により、 Active Message [15] [16] が簡単に実現できる。

MBCF_SIGNAL コマンドはプログラム起動を引き起こすだけではなく、そのプログラムに引き渡すためのデータ転送を遠隔メモリ書き込みもしくは遠隔 FIFO 登録（MBCF_FIFO と同等）によってプログラム起動と同時に実現する。このデータ転送方式を指定するのがパラメータ格納タイプを示す副コマンドである。データ格納領域へのポインタは MBCF_SIGNAL 構造体内に格納され管理されている。オプションによって、コマンド発行元から渡されたデータ以外に受信した MBCF_SIGNAL コマンドの発行元に関する情報をこの領域に付加できる。この機能により起動プログラムが機能実行前に要求元を確認することが可能である。

最後の副コマンドはプログラムの起動条件を示す物である。 MBCF_SIGNAL コマンドを受け取った時に対

象となる構造体に対応したプログラムが起動されていなければ、他の二つの副コマンドの指示に従ってデータ転送をして、指定されたプログラムを起動させる準備をする☆。しかし、コマンド受信時に先行するコマンドでプログラムを起動させており、そのプログラムの実行がまだ終了していない場合には、起動条件副コマンドで以下の動作の内一つを選択する。

(1) コマンドの受信を拒否(キャンセル)する
(2) データのみ指定されたバッファ/キューに格納するが、プログラムは起動しない。
(3) データを指定されたバッファ/キューに格納し、先行プログラムの実行状態にかかわらず新しいプログラムの起動を行う。

5. SSS–CORE の基本性能

5.1 測定マシン環境

SSS–CORE Ver.1.1 は Sun Microsystems 社製のワークステーション SPARCstation 20 もしくはその互換機上で動作する(SPARCstation 10 も可)。今回の測定に使用したマシンは 85MHz の SuperSPARC プロセッサを各マシン(ノード)に 1 台搭載している。また、各マシンには Sun Microsystems 社製の Fast Ethernet Adapter II が追加されており、メモリベース通信には Fast Ethernet (100BASE-TX) を使用する。

5.2 システムコールの性能

表 2に非常に単純なシステムコールとタスクスケジューリングを引き起こすシステムコールの実行時間を示す。 get_taskid() は自分のタスク ID を取り出すシステムコールであり、 UNIX の get_pid() に相当するものである。 sleep(0) は明示的なコンテクスト切替えを実行するシステムコールであり(0 以外の引数の場合は sleep 状態になる)、切り替わるべき ready 状態のタスクがないときは sleep(0) を発行したタスクが再びスケジューリングされる。 sleep(0) の項目は他に走行中のタスクがない状態での sleep(0) の実行時間、 sleep(0)x2 は二つの sleep(0) を繰り返すタスクを同時に走行させた場合の片側のタスクでの sleep(0) の実行時間(システムコール開始から終了までの実時間)を示す。 ready タスクの数が増えると若干オーバヘッドが増える可能性はあるが、コンテクスト切替えは 20μsec 台である。

表2 システムコールのコスト

システムコール	時間(μsec)
get_taskid()	7.20
sleep(0)	22.05
sleep(0)x2	47.50

表 3には、 SSS–CORE に設計思想で大きなウェイトを占める、属性を指定して実メモリページを獲得する(物理メモリの獲得と論理アドレスとのマッピングを行う)システムコールとそのメモリを返却するシステムコールの実行時間をメモリサイズ毎に示す。 SSS–CORE Ver.1.1 は 4K, 256K, 16M, 4G の各ページサイズを扱うことが可能であるが、本測定ではソフトウェア分散共有メモリの便宜を考えて、獲得領域のサイズによらず 4K ページサイズを指定した。

☆ 実際に起動されるのは対象タスクに実行権が回ってきた時である。

表3 メモリ獲得/返却システムコールのコスト

サイズ (byte)	4K	16K	64K	256K	1M
alloc (μsec)	23.91	28.91	48.77	123.2	431.2
free (μsec)	19.49	20.36	23.91	36.23	99.06

5.3 軽量システムコールの性能

表 4に軽量システムコールの実行時間を示す。

表4 軽量システムコールのコスト

軽量システムコール	時間(μsec)
get_taskid_light()	1.12
page_invalidate (4K)	6.10
page_invalidate (64K)	25.91
page_validate (4K)	6.14
page_validate (64K)	26.74

get_taskid_light() はシステムコールとして実装されている get_taskid() を比較のために軽量システムコールとして実装した物である。ただし、他の軽量システムコールと同じ扱いをして、 get_taskid_light() だけが高速に処理されるような実装はしていない。 get_taskid_light() の実行時間が get_taskid() の 1/6 以下である。 page_invalidate はソフトウェア分散共有メモリ実現のために、メモリページを無効化する軽量システムコールである。 (4K),(64K) は無効化したメモリの大きさを示し、それぞれ 1 ページと 16 ページに相当する。逆に page_validate は無効化したメモリページを有効に戻す操作である。 SuperSPARC にはハードウェア walk を行う MMU を活用した probe 命令(ページ構造体をハードウェアトラバースする命令)が存在するが、これらの実装では使用しておらず、ソフトウェアでページ構造体をトラバースした場合の実行時間である。

5.4 ユーザレベルページャの性能

SSS–CORE ではユーザレベルのソフトウェア分散共有メモリを支援するために、ユーザレベルプログラムをページフォルト時に起動することができる。ユーザレベルの起動プログラムとして中味が空(return のみ)の物でユーザレベルページャ起動オーバヘッドの測定を行った。ページフォルトを起こすメモリ書き込み直前から、フォルトが起こりページが有効化され、ユーザレベルページャが起動し、リターン後カーネル経由で元のユーザコンテクストの書き込みが実行されるまでの時間は、**20.76μsec** と高速である。この値はほぼユーザタスクへのアップコールのオーバヘッドとページフォルト処理のためのオーバヘッドを足し合わせたものである。

5.5 MBCF_FIFO, MBCF_SIGNAL の基本性能

表 5に 100BASE-TX(Fast Ethernet)を使った SSS–CORE Ver.1.1 上の MBCF による MBCF_FIFO と MBCF_SIGNAL のレイテンシ(片道)を示す。測定は 100BASE-TX 用のハブで接続された異なるノード上の二つのユーザタスク間で行われた。

表5 One-way latency of the MBCF/100BASE-TX

data size (byte) command	4	16	64	256	1024
MBCF_FIFO (μs)	32	32	40.5	73	210.5
MBCF_SIGNAL (μs)	49	52.5	60.5	93	227.5

MBCF_FIFO のレイテンシは送信側が MBCF 要求システムコールを発行してから、受信側が FIFO 構造体のポーリングによって到着を検知し、システムコールによって受信データを別メモリ領域に読み出し終るまでの時間である。実際の測定は、受信側が再度 MBCF_FIFO コマンドで発信元に同一データを転送して、転送元が返送されたデータを読み出した時刻から最初の要求発行時刻を引いてラウンドトリップタイムを求めて半分にした。この MBCF_FIFO のレイテンシは、二回の軽量システムコールと 1 回の FIFO 読み出しに伴うデータコピーが含まれた値である。なお、厳密に言えば、 Ethernet 上のパケット転送保証のために、 MBCF 要求システムコールの中でもパケットのコピーを 1 回行っている。

MBCF_SIGNAL のレイテンシは送信側が MBCF 要求システムコールを発行してから、受信側において設定されたプログラム（DSP）が起動されて、その起動プログラム内で MBCF_SIGNAL コマンドが運んだデータを FIFO キューから読み出すシステムコールを終了するまでの時間である。実際の測定は、 MBCF_FIFO コマンドの場合と同様に、ラウンドトリップタイムを求め半分にした。この MBCF_SIGNAL のレイテンシは、二回の軽量システムコールと 1 回のユーザ権限でのプログラム起動（アップコール）と 1 回の FIFO 読み出しに伴うデータコピーが含まれた値である。

6. おわりに

スクラッチから開発した汎用スケーラブル OS SSS–CORE Ver.1.1 の設計方式およびカーネル構造について説明し、基本的なサービスに関して実測値を示した。また、高機能メモリベース通信の詳細と性能の高さを示した。 SSS–CORE Ver.1.1 は、メモリ操作を中心にした最適化と最適化重視の設計方式により、非常に高速なオペレーティングシステムとなっている。一世代以上古いプロセッサの上に実現されているにもかかわらず、ユーザレベルのプログラムからページの無効化が 6μsec、ユーザレベルページャ起動のオーバヘッドが 20μsec 強である。現在、最新鋭プロセッサ（UltraSPARC）を搭載したワークステーションに移植中である。

謝　　辞

本研究の一部は情報処理振興事業協会（IPA）が実施している独創的先進的情報技術に係わる研究開発の一環として行われた。

参 考 文 献

1) 松本 尚, 平木 敬: 汎用並列オペレーティングシステム SSS–CORE の資源管理方式. 日本ソフトウェア科学会第 11 回大会論文集, pp.13–16 (October 1994).

2) 松本 尚, 平木 敬: 汎用超並列オペレーティングシステム SSS–CORE のメモリベース通信機能. 第 53 回情報処理学会全国大会講演論文集 (1), pp.37–38 (September 1996).

3) T. Matsumoto and K. Hiraki. MBCF:A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility. In *Proc. of the 1998 ICS*, (July 1998).

4) 松本 尚, 平木 敬: 超並列計算機上の共有メモリアーキテクチャ. 信技報, CPSY 92-26, pp.47-55 (August 1992).

5) 松本 他: メモリベース通信による非対称分散共有メモリ. 情処コンピュータシステムシンポジウム論文集, pp.37–44 (November 1996).

6) T. Matsumoto, et al.: Compiler-Assisted Distributed Shared Memory Schemes Using Memory-Based Communication Facilities. In *Proc. of the 1998 PDPTA*, (July 1998).

7) 丹羽, 稲垣, 松本, 平木: 非対称分散共有メモリ上におけるコンパイル技法. 情報処理学会研究報告 97–HPC–67, 情報処理学会, Vol.97, No.75, pp.121–126 (August 1997).

8) T. Inagaki, J, Niwa et al.: Supporting Software Distributed Shared Memory with Optimizing Compiler. In *Proc. of the 1998 ICPP*, (August 1998).

9) 松本 他: 粒度に基づいた並列計算の分類法とマルチプロセッサの資源管理法について. 日本ソフトウェア科学会第 7 回大会論文集, pp.133-136 (October 1990).

10) 松本 尚: マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察. 計算機アーキテクチャ研究会報告 No.79-1, 情報処理学会, pp.1-8 (November 1989).

11) M. Accetta, et al.: Mach: A New Kernel Foundation for UNIX Development. *Proc. Summer 1986 UNIX Conf.*, USENIX, pp.93–112(1986).

12) S. J. Mullender, et al.: Amoeba: A Distributed Operating System for the 1990s. *IEEE Computer*, vol.23, pp.44–53 (May 1990).

13) M. Berver, et al.: Distributed Systems, OSF DCE and Beyond. in *DCE—The OSF Distributed Computing Environment*, Berlin:Springer-Verlag, pp.1–20 (1993). M

14) R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith: The Tera computer system. *Proc. 1990 Int. Conf. on Supercomputing* (June 1990).

15) T. von Eicken, D. E. Culler et al.: Active Messages: A Mechanism for Integrated Communication and Computation. *Proc. 19th Int. Symp. on Computer Architecture*, pp.256–266 (May 1992).

16) T. von Eicken, A. Basu, and V. Buch: Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, pp.46–53 (February 1995).