高生産な並列プログラミング環境を実現するための 並列ランタイムシステムの設計

緑川 博子*[1]，鈴木 悠一郎*[2]

The Design of Parallel Runtime System for Highly Productive Parallel Programming Environment

Hiroko MIDORIKAWA [*1], Yuichiro SUZUKI [*2]

ABSTRACT： To realize highly productive parallel programming environments, newly designed parallel runtime system based on software distributed shared memory is proposed. It can provide a seamless parallel programming model from intra-node multithread programming to inter-node parallel programming. It overcomes the data inconsistent problem caused by multiple threads in a user program at remote page fetching. It provides node-shared data address space with a traditional relaxed memory consistency model as well as high-speed ad-hoc data copy interface between node-shared data and node-local data. User can easily extend their existing C, OpenMP and OpenACC programs to node parallel programs incrementally by directive-based programming API on this runtime system.

Keywords：Distributed Shared Memory, Parallel Programming Model, Parallel Processing, Cluster Computing, Global Address Space

（Received September 20, 2013）

１．はじめに

高性能コンピューティングでは，高速ネットワークで結ばれた非常に多数（数千規模の）の計算ノードによる並列処理システム（クラスタシステム）が用いられることが多く，さらに一つの計算ノード内プロセッサの複数化，CPU のメニーコア化・ヘテロ化が進んでいる．このようなクラスタシステム向けの並列プログラム作成，実行において，高プログラマビリティ（プログラムのしやすさ）と高性能という２つの（事実上）相反する要求を満たすのは難しい.従来のように,性能重視の観点から，計算ノード間の通信やデータ配置などの詳細をすべて指定するような，メッセージパッシング型の並列プログラミング（MPI など）だけで対応するのには，限界を迎えている.このため,柔軟性の高いプログラミングモデル，API，並列言語などの研究が，最近，特に盛んになってきている．

現在，高性能なプログラムを並列処理する場合，プログラムの並列性を，計算ノード内は OpenMP，計算ノード間は MPI で記述するハイブリッドプログラミングがデファクトスタンダードになっている．OpenMP は，逐次プログラムコードに pragma 指示文を挿入するだけで並列化を可能にするが，MPI は，実行時のタスクやデータのプロセッサ割当をユーザが記述しなくてはならない．また，MPI は，ノード間で共有するデータは，変数名がノード間で一意にならず，アドレス領域も変わるため，プログラムは複雑になり，開発コスト(コーディング，デバッグ，保守等)が高く，生産性が低いことが問題になっている．

現状のプログラムの生産性の難点を解決するために，多くの分散メモリ環境向け並列言語が開発されてきている．DARPA の HPCS プロジェクト[1]では，高生産性を指向した並列言語(Chapel[3], X10[4])の提案・実装がされおり，世界的にプログラマビリティがさらに高い並列言語が求められている[2].

*1：情報科学科 助教（midori@st.seikei.ac.jp）
*2：理工学研究科理工学専攻情報科学コース修士学生

本研究では，既存の逐次言語を使用するユーザに対して，ノード内並列と同様にノード間並列でも共有できるアドレス領域を仮想的に提供し，プログラマビリティとポータビリティの点で生産性が高いプログラミング環境を実現するためのランタイムシステムとして，MiMoSa(ミモザ)を提案する．MiMoSa は，近年開発されている並列言語(Chapel,X10)などと異なり，以下のようなユーザプログラミング環境・API を実現する基盤となる．

- ノード間で共有するデータ(共有するアドレス領域)へ制限なしにアクセスするプログラムが書ける
- 新たな言語で書き直す必要がないように，既存の C 言語から利用でき，ノード内の OpenMP や Pthread での並列性記述もそのまま用いることができる
- 限定的な共有データアクセス用途に，ノード間共有アドレス領域からノード内ローカルデータへコピーを可能にする高速アクセス API を提供する

2. ソフトウェア分散共有メモリ

MiMoSa は，ユーザレベルソフトウェアによる分散共有メモリ（SDSM）を基本とし，カーネルに変更を加えないことで高可搬性を実現する．本節では，MiMoSa の基礎的な部分である SDSM について述べる． SDSM とは，図 1 のようにネットワークで接続されたノード間のメモリをソフトウエアだけで共有メモリのように見せる技術である．ページベースの SDSM では，自ノードのメモリにないデータへユーザプログラムがアクセスすると，OS のメモリ保護機構を使用して，そのアドレスを検知し，OS ページサイズの倍数であるデータサイズで，アクセスされたデータを含むページを持つ遠隔ノードと通信して，必要なページを自ノードへ取得する．これにより，ユーザプログラムには透過に，遠隔データの取得を行い，仮想的に，複数ノードからアクセス可能な「共有メモリ」を実現する．

SDSM ではノード間で共有するデータをページ単位で管理し，各ノードが管理するページ(オーナページと呼ぶ)を決めており，管理外のノードは管理ノードからページをキャッシュ(キャッシュページと呼ぶ)することで，すべてのデータへアクセスが可能になる．図 2 に一般的な SDSM での保護属性を用いた，外部データアクセス時の取得方法を示す．

キャッシュページは，SDSM によって決められたメモリ一貫性モデルによってオーナへデータの変更部分の書込みを行う．多くの SDSM (TreadMarks[7], SMS[8], SCASH[9])では，このメモリ一貫性モデルに逐次一貫性モデルよりさらに緩和型なメモリ一貫性モデルを採用している．これは，分散メモリでのメモリ一貫性処理のコストが共有メモリに比べ非常に高く，一貫性を保証する部分を緩和することで性能を向上させるためである．

これまでの1990年代後半に開発されたSDSMは，スレッド技術が未成熟期に開発されており，Pthreadなどのスレッドユーザプログラムへの対応が不十分な点があり，実装自体もシングルスレッドであった．また，Ethernet上でのソケット通信で開発されており，近年の通信媒体の多様化によって使用できない環境が多くなりポータビリティ上の問題もあった．

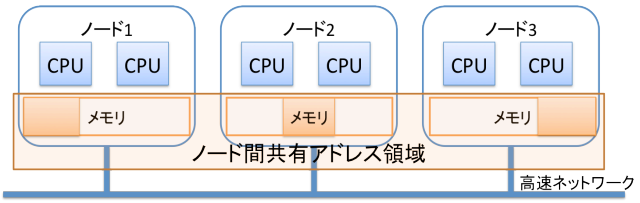

図 1　SDSM によるノード間共有アドレス領域

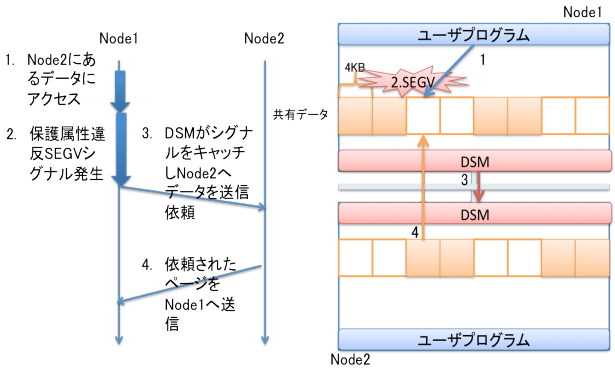

図 2　ページベース SDSM での遠隔データ取得手順

3. MiMoSa　ランタイムシステム

MiMoSa は，従来の SDSM には無かった，マルチスレッドユーザプログラムへの対応や，ポータビリティの高い通信ネットワークインターフェースを用いて実装する．また，Pthread や OpenMP などで書かれたノード内並列性の記述を変更せずに，透過的に遠隔データへのポインタアクセスを可能とし，データ整合性を保証することで，ノード間共有データへのアクセスにおいて高い生産性を実現する．MiMoSa では，ユーザは従来の C 言語で書かれた逐次プログラムコードをそのまま利用し，通信記述不要の SPMD スタイルのノード間並列へ拡張することが可能になる．また，ユーザへ提供する場合には，C 言語向け専用トランスレータ[12]を用意し，ユー

ザは OpenMP や OpenACC[6]で扱われている pragma 構文と同様なディレクティブの追加だけでノード間共有データのマッピング指示や，タスク配置，Work-sharing 構文への対応もすることができるようにする．MiMoSa は，C 言語向けライブラリとして実装される．

従来の SDSM にはない MiMoSa の新機構を以下に示す．

1. マルチスレッドユーザプログラムへの共有アドレス領域に対する透過的アクセス動作機構の導入
2. スレッドと MPI による通信と処理の効率化
3. ノード間で共有するデータのメモリ一貫性処理とは別ルートで，共有データのアドレス領域からノード内のローカルデータへコピーを可能にする高速アクセス API の提供

次節以降で，MiMoSa の実装について説明する．

3.1 共有アドレス空間とページ状態遷移

ノード間で共有するアドレス領域は，システム全体のノードで分割してマッピングし管理をする．ノードが担当したアドレス領域を所有領域(OWNER 領域)とし，その中のページをオーナページと呼び，オーナページを管理しているノードをオーナと呼ぶ．ノード数 3 で，60 要素の配列 a[60]をノード間共有アドレス領域へ分割マッピングした例を図 3 に示す．オーナ以外が，マッピングされたページへアクセスすると，図 2 で説明した手法でページを取得し，一時的なページ(以後キャッシュページと呼ぶ)として管理する．初期実装で採用した Weak メモリ一貫性モデルに基づき，データ同期時に適切にキャッシュページの変更部分がオーナページへマージされる．

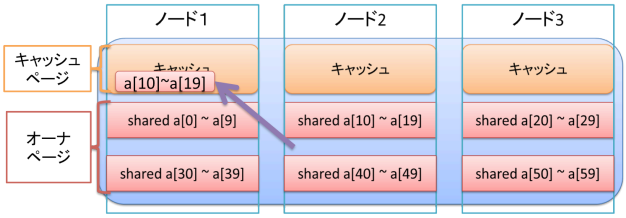

図 3　ノード間共有アドレス空間の実装

データ一貫性制御のためのページ状態遷移を図 4 に示す．初期実装として，単純な invalidate 方式のページ遷移を採用している．オーナページとキャッシュページでは遷移する状態が異なり，キャッシュページはページの保護属性，オーナはキャッシュへコピーされたか，によって主に遷移を行い，メモリコンシステンシの同期によって初期状態へ戻る．図 4 の状態とは別にオーナページが Shared 状態を取らない遷移も実装している．この 2 種類の状態遷移は，実行するアプリケーションによって実行時に選択ができ，アプリケーションのデータへのアクセスパターンによって最適な状態遷移手法を取ることを可能にしている．

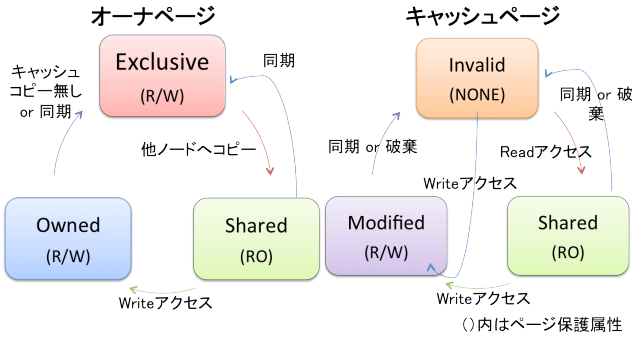

図 4　MiMoSa におけるページ状態遷移

3.2 遠隔ページ受信機構

3.2.1 マルチスレッドユーザプログラムへの対応方法

メモリ保護属性を使用するユーザレベル遠隔メモリページングをするシステムにおいて，ユーザプログラムがマルチスレッド実装の時，ページの受信時に不正なデータへアクセスが起きてしまう場合がある．ノードに無いページはアクセス不可のメモリ保護モードに設定されており，ユーザプログラムスレッドがこのページをアクセスした際に起動されたシグナルハンドラ内で，該当ページを持つ遠隔ノードからページを受信する．従来，ページを受信時に，該当ページのアドレス領域のメモリ保護属性を読み書き可能としこのアドレス空間に直接受信する．しかし，ユーザプログラムがマルチスレッドである場合，ページ受信中に，ハンドラ処理中で停止しているスレッド以外の他のユーザスレッドがそのアドレス領域へ読み書きができるため，データの不整合がおきる危険性がある．

この問題を解決するために，本機構ではページをバッファ領域に受信し，ユーザアドレス空間のメモリ保護機構はアクセス不可のままとする．ページを受信後，全ユーザスレッドを一時的に停止し、メモリ保護機構を書き込み可とし当該ページをユーザアドレス空間へコピーする手法をとる．本手法の流れを図 5 に示し，制御の流れを説明する．

1. 計算スレッドが遠隔にデータのある，ページ a にアクセスする．
2. オーナからページ a を受信し，バッファリングしておく．

3. ページを受信完了後，全ユーザスレッドを一時停止する．
4. バッファしていたページ a をユーザのアドレス領域のメモリ保護属性を Read/Write 可に変更しデータをコピーする．
5. コピー終了後，全ユーザスレッドを再開させる．

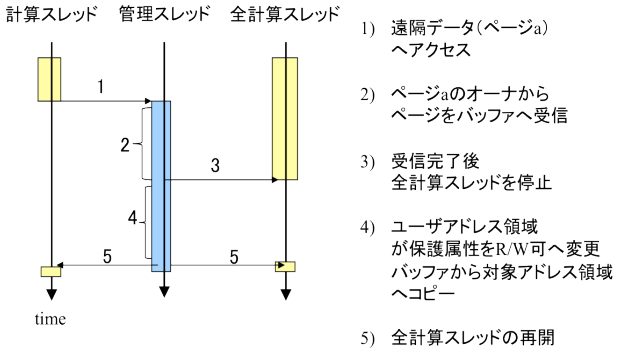

図 5　MiMoSa における遠隔メモリページ受信手順

3.2.2　ページ受信機構の実装方式

前節のように，遠隔ノードから取得したページを不可分に（アトミックに）ユーザアドレス空間にコピーするには，その時点のユーザプログラム中に存在するすべての計算スレッドに対し，一時的なサスペンドを行うシグナルを送る必要がある．しかし，現在，Linux 上で提供される NPTL スレッド環境では，個別のスレッドにシグナルを送信する機構は実装されているものの，1 プロセス内の自分以外の全てのスレッドにシグナルを一斉に送信する機構が存在しない．このため，全スレッドを停止するには全スレッドのスレッド ID を取得して個別にシグナル送信する必要があるが，全スレッドの ID を取得するインターフェースも実装されていない．一般的にスレッド ID を取得するには，各スレッドで pthread_self 関数により自スレッドの ID を取得するか，スレッドを生成した親スレッドがスレッド生成関数の返り値により ID を取得する方法しかない．しかし，どちらの手法も，ユーザプログラムの変更が必要で，ユーザ透過に行うことができない．また，ユーザが明示的にこのような処理をしようとしても，ユーザにスレッド生成が隠ぺいされている場合，たとえば，スレッド実装ライブラリを使った疑似逐次プログラムや，OpenMP で書かれたユーザプログラムの場合は，ユーザによる明示的指定も不可能となる．

MiMoSA では，この問題を解決するために，Linux のスレッド生成に用いられる pthread_create 関数を，我々が作成した独自の pthread_create 関数に置き換える手法を導入した（図 6）．共有関数ライブラリの動的ロードパス LD_PRELOAD を変更することにより，我々独自の pthread_create 関数がユーザプログラムから呼ばれるようになる．この関数では，もともとの pthread_create 関数を呼んでスレッドを生成した後に返値であるスレッド ID を，MiMoSA の内部データであるスレッド ID テーブルに登録する．これにより，ユーザプログラム中に動的にスレッドが生成，消滅した場合においても，アトミックなページコピーの際に，その時点で存在する全ユーザスレッド ID をこの表から取得し，一時中断のためのシグナルを送信することができる．この機構とシグナルハンドラとを組み合わせることによって，全ユーザスレッドを一時的に止める機能を実装した．

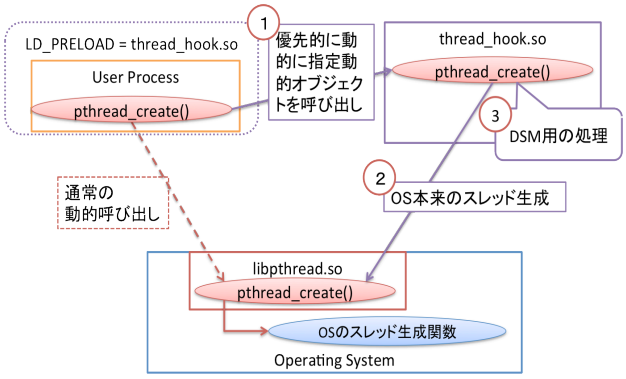

図 6　pthread_create 関数の置き換え

3.2.3　スレッド一時停止機構の性能への影響

このような，遠隔ページ受信時のユーザスレッドの一時停止の機構は，MiMoSa 構築前に，ユーザレベル遠隔メモリページングシステム DLM（分散大容量メモリ）に導入し，そのオーバーヘッドを調査した[10]．この結果，実用レベルで十分利用可能であることが明らかになった．図 7 は，この DLM を用いたとき，OpenMP で実装された FFTW ライブラリを利用した疑似逐次のユーザプログラムでの実行結果を示す．DLM では計算ノードは一つでマルチスレッドユーザプログラムが実行され，他の遠隔ノードでは計算をせずに計算ノードへのメモリサーバとして働く．このため計算ノードで，物理メモリサイズを超えるデータを扱うプログラムを実行することができる．遠隔ノードのメモリを使用する率が高くなるにつれ，性能向上率は落ちるものの，一定のパフォーマンスを示すことができている．

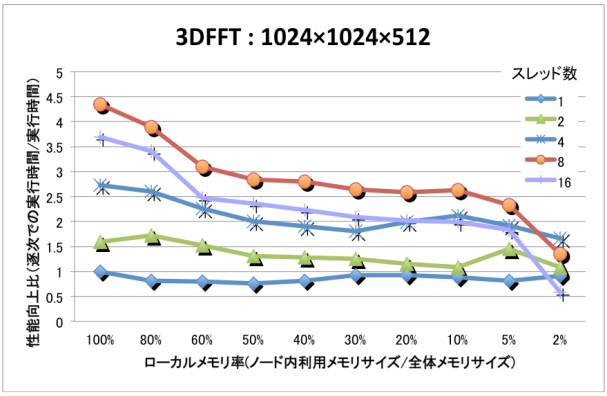

図 7　ユーザスレッド一時停止手法による遠隔ページ受信機構を利用したマルチスレッドプログラム性能

3.3　マルチシステムスレッドによる処理の効率化

MiMoSa の全体像を図 8 に挙げる．ノード間ネットワークは MPI で実装している．MPI は Ethernet, Myrinet, InfiniBand など，様々な高速通信媒体に対して各メーカーが最適な実装を施しており，従来の TCP/UDP を用いた SDSM に比べポータビリティの高い実装になっている．

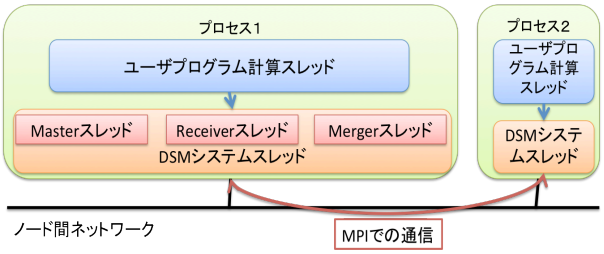

図 8　MiMoSa ランタイムシステムの構成

MiMoSa システムは 3 つのシステムスレッドによって構成され，ノード間のデータ送受信やノード内外からの処理要求に対し並列処理を行う．Master スレッドは，ノード内外の要求メッセージの処理を行う．要求メッセージは，ノード内とノード外をタグ番号により区別して別々に管理され，ノード外の要求を優先して FIFO で処理をする．Receiver スレッドは，メッセージとページを受信し処理する．Merger スレッドは，Receiver スレッドが他プロセスからページ変更内容を受け取った際に生成し，該当ページへ変更内容をマージするためのスレッドである．

以下に，この構成における通信と処理の効率化について述べる．

3.3.1　ページ送受信の並列化

ページ送受信は，Master スレッドがページの送信，Receiver スレッドがページの受信を担う．Master がノード内のメッセージでページ取得処理に入る場合，該当ページのオーナノードへページ送信依頼を送信し，該当ページの受信関数（MPI_Irecv）を呼び出しノンブロッキング通信での受信を開始しておく．Receiver は MPI_Wait で受信状況を逐一確認し，受信が完了したページをユーザのアドレス領域へ不可分にコピーをし，該当ページを要求していたユーザプログラムスレッドを再開させる．この実行の流れの例を図 9 に示す．ページ送受信を並列で行えるようにし，ページ受信をノンブロッキングで行っていることで，ページ送受信にかかるシステムの処理の効率化を行っている．

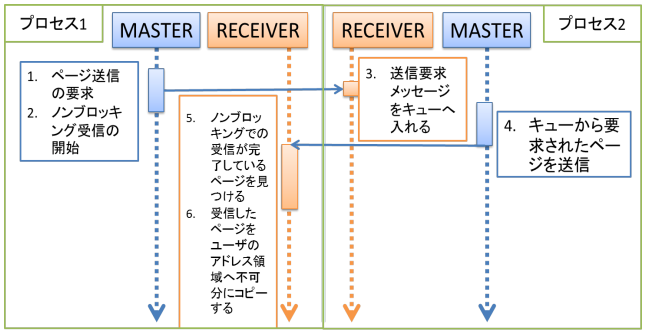

図 9　2 スレッドによるページ送受信の効率化

3.3.2　ページ変更マージ処理の効率化

従来の SDSM ではページのマージ処理を行う際に，作業するスレッドが一つであったため，ユーザプログラムによりデータ同期関数が呼ばれてから処理を始めていた．MiMoSa では，マージ処理が必要な場合，その他の通信や処理を阻害しないように，マージ処理専用の Merger スレッドを用いて，同期以前であってもマージ処理を並列で実行可能とした．ただし，各ノードからの変更部分のマージは行っても，オーナーページへの反映はすべての計算ノードがデータ同期関数をコールした後に行う．手順を図 10 に示す．このことにより，他のシステムスレッドの動作への影響を最小限に，変更部分の受け取りからマージまでを並列に実行することを可能にした．

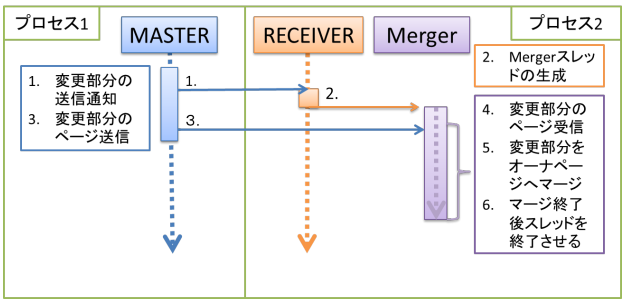

図 10　Merger スレッドでの変更部分マージの効率化

3.4　共有データへのアドホックアクセス機構

SDSM でのプログラムでは，ノード間共有アドレス領域へのアクセスをユーザ透過で行えるが，小規模なデータへのアクセスもページ単位で通信をしてしまうだけでなく，共有データの一貫性管理処理がされることになる．そこで，ユーザが共有アドレス領域へ明示的にアクセスする（PUT，GET のような）インターフェースを提供することで，ユーザによる共有データのローカルデータへの代入と，ローカルデータを必要な部分だけ共有データへ反映でき，ページ単位ではなくオブジェクト単位によるデータコピー機能をユーザ側で可能にする．このアドホックなアクセスによるデータのコピーは，共有データのデータ一貫性処理を伴わずに行われることが前提であるため，オーバーヘッドが非常に少ない．実装上はほとんど MPI の送受信関数と同等の機構になる．

多くの数値計算処理などで頻出するデータ領域分割による並列処理の場合，隣接境界データのリードだけのために，ノード間共有データからノードローカルデータへのコピーのこのようなアドホックな機能が使えると，性能上有利である．これまで SDSM では，このような限定的なデータ領域のリファレンスであっても，通常のページベースの通信とそれに伴うデータ一貫性管理維持処理を伴い，性能低下の原因になっていた．このようなアドホックなアクセス機構を準備することにより，共有アドレス名前空間を提供しながら，ほとんど MPI にくらべて性能低下のないデータ交換ができることになる．

4. MiMoSa におけるプログラムインターフェース

MiMoSA の提供する C 関数ライブラリによって，ノード内並列には OpenMP などスレッド並列，ノード間並列には本システムライブラリを使用した SPMD モデルでの並列コードを図 11 のように記述することが可能となっている．同じプログラムを従来の MPI と OpenMP で記述した場合には，図 12 のようになり，共有データへの代入文ではなく，明示的なノード間の通信の記述が必要となる．

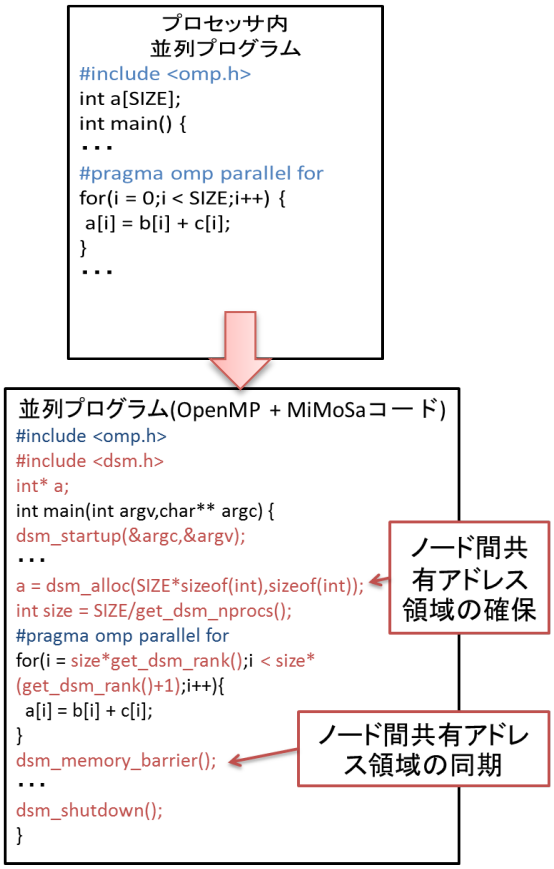

図 11　MiMoSa ライブラリを利用したプログラム例

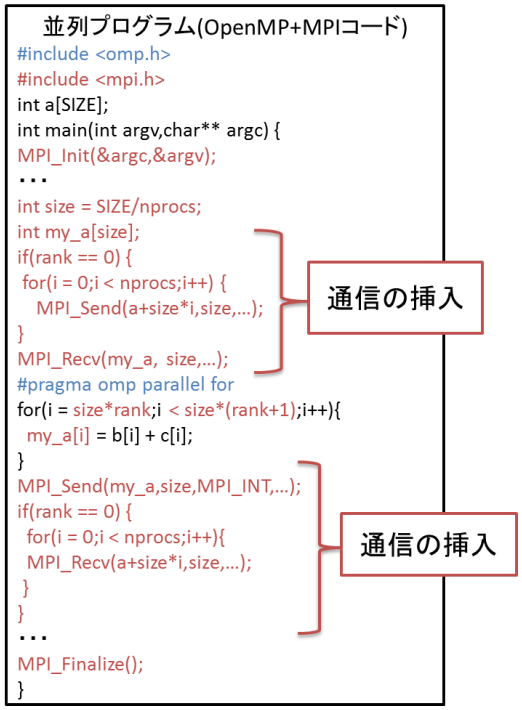

図 12　MPI と OpenMP による従来プログラム例

さらに，現在，SPMD モデルや，ユーザによるタスクの配置などのノード間並列での並列性記述部分のプログラマビリティやポータビリティの低さを解決するトランスレータ[12]の実装を進めており，図 13 のようなディレク

ティブ挿入により，ノード並列，ノード内コア並列，両方のハイブリッドなど，逐次プログラムから，インクリメンタルに並列性を容易に付加できるような並列プログラミングモデルを実現できるようにする．

```
並列プログラム(トランスレータ使用)
#include <omp.h>
#pragma smint share [nprocs][]
int a[SIZE];
int main() {
・・・
#pragma smint papallel for
#pragma omp parallel for
for(i = 0;i < SIZE;i++) {
  a[i] = b[i] + c[i];
}
・・・
}
```

図 13　MiMoSa 向け smint ディレクティブを利用したプログラム例

5. おわりに

高生産な並列プログラミングを可能にする基盤システムとして，MiMoSa を設計し初期実装を行った．MiMoSa は SDSM を基盤システムとし，従来に無い機構を導入することによって，共有データへのポインタや，ノード間で共有するアドレス領域へ制限なしにアクセスを可能にする一方で，既存の逐次言語(C 言語)や OpenMP などのノード内並列言語もそのままで利用できる．これにより，プログラマビリティとポータビリティの高い環境を提供することが可能になった．

高プログラマビリティと高性能の２つの要求を満たすために，柔軟性の高いプログラミングモデル及び API と，ユーザが「データアクセス柔軟性重視」か「性能重視」を選択できる機能選択的な並列ランタイムシステムを備えることで，実用的・効率的な並列プログラム実行・開発環境を提供する．

現在，試作システムを構築し，複数ノードでの動作実験を行って，最適化を施している．また MiMoSa を基盤システムとして，既存プログラムからの並列化が容易にできるディレクィブベースの並列プログラミング API とトランスレータを構築し，最終的に高生産な並列プログラミング環境を実現する．

参考文献

[1] http://www.darpa.mil/Our_Work/MTO/Programs/High_Productivity_Computing_Systems_(HPCS).aspx [ONLINE] (2013,9.20)

[2] http://www.darpa.mil/Our_Work/MTO/Programs/Ubiquitous_High_Performance_Computing_(UHPC).aspx [ONLINE] (2013,9.20)

[3] http://chapel.cray.com/ [ONLINE] (2013,9.20)

[4] http://x10-lang.org/ [ONLINE] (2013,9.20)

[5] http://upc.gwu.edu/ [ONLINE] (2013,9.20)

[6] http://openacc.org/ [ONLINE] (2013,9.20)

[7] P. Keleher, A.L. Cox, S.Dwarkadas, and W. Zwaenepoel, : ” TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems”, Proc. of the Winter USENIX Conference, pp.115-132 (1994,1)

[8] 緑川，飯塚："ユーザーレベル・ソフトウエア分散共有メモリ　SMS の設計と実装"，情処論文誌 HPC,Vol.42, No.SIG9(HPS 3), pp.170-190 (2001)

[9] H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto, and T. Takahashi : “Dynamic Home Node Reallocation on Software Distributed Shared Memory”,In Proceedings of HPC Asia 2000, Beijing, China, pages 158–163 (2000.5)

[10] 鈴木，鷹見，緑川："マルチスレッドプログラムのための遠隔メモリ利用による仮想大容量メモリシステムの設計と初期評価"，情報処理学会,HOKKE-19,Vol.2011-HPC-132, No.13, pp.1-6 (2011,11)

[11] Kentaro Hara, Kenjiro Taura: ”Parallel Computational Reconfiguration Based on a PGAS Model” IPSJ Journal of Information Processing. Vol.20, No.1. (2012,1)

[12] 直木,緑川,甲斐 : “マルチコアプログラムにノード並列機能を加える API の提案”,FIT2012(2012,9)