研究課題別評価

1 研究課題名:

広域分散共有メモリ機構を支援する最適化コンパイラ

2 研究者氏名:

丹羽 純平

3 研究のねらい:

近年急増している新世代 E-Science アプリケーションは、大量の計算機パワーを必要としており、従来のスーパーコンピュータを多少増強した程度では扱いが困難であり、コストパフォーマンスの点からも非現実的であるという点があげられます。コストパフォーマンスの面では、研究機関内にある計算機資源をネットワークで接続した LAN クラスタが優れているものの、如何せん、計算機パワーの不足は否めません。一方、ネットワーク技術の進歩やスイッチング技術の向上により、SuperSINET を初めとする国内超高速バックボーンネットワークが急速に整備されてきました。そして、国際間の超高速バックボーンネットワークも整備されつつあります。その結果、研究機関の計算資源が超高速でつながれ、計算資源を広域的に利用した"次世代実験科学のための計算プラットフォーム" の構成(図1)がコストパフォーマンスの面からも現実味を帯びたものとなりつつあります。

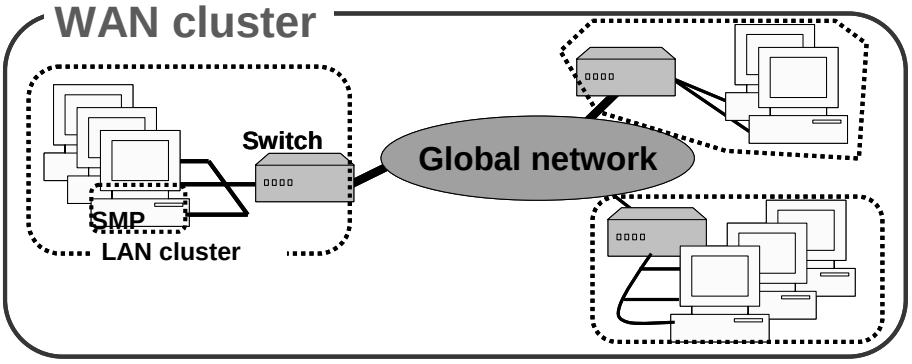


図1 WAN クラスタの構成

広域環境にある計算機資源をフルに活用するには、ユーザ自ら並列プログラムを記述することが求められます。共有メモリモデルは従来のスタンドアローンなプログラミングの自然な拡張であり、メッセージパッシングモデルに比べて、並列プログラムを記述しやすいという利点を持っています。もちろん、共有メモリモデルに従って記述された並列プログラム(例:OpenMP や PARMACS で拡張された C のプログラム)を、直接メッセージパッシングコード(分散計算機上のコード) に変換することは可能です。しかし、幅広いクラスのアプリケーションを効率良く扱うためには、アプリケーションが実行時に直接共有アドレスを扱えること(例:タスクキューを用いた動的負荷分散)が求められます。すなわち、実行時にシステム全体で仮想的に共有メモリを提供する機構:ソフトウェア分散共有メモリ機構(S-DSM)が必要になります。

そこで、効率の良いソフトウェア分散共有メモリ機構を構築できるかどうかが鍵になってきます。これまで、LAN 上の分散環境では、オペレーティングシステムの支援と最適化コンパイラの支援とランタイムの支援があれば、高性能な S-DSM を構築することが可能であることが示されてきました。

この結果から、超高速 WAN 上であっても、最適化コンパイラとランタイムの支援があれば、共有メモリモデルに従って記述された並列プログラムを効率良く実行できるのではないかと考察しました。もちろん、WAN は LAN と比較して様々な問題を抱えています。例えば、通信の遅延(レイテンシ)が大きいといった点が挙げられます。また、大量の共有メモリを必要とするアプリケーションが多いという点も挙げられます。更に、計算機の台数が多くなるために、一部の計算機が故障する可能性が増大するので、効率的な耐故障機能が必要となります。

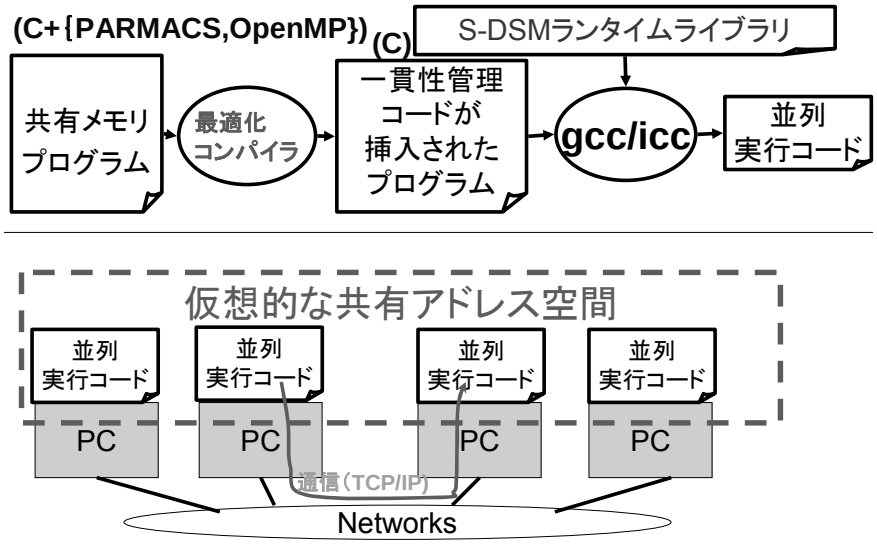


図2 本研究が提案する広域分散共有メモリ機構のイメージ

本研究では、こういった問題点を克服するコンパイル時最適化と実行時最適化を提案し、それらの最適化を可能とするインタフェイス(WDSM)を導入しました。実際にシステムを構築し、本手法の有効性を評価・検証してきました。図2は、本研究が提案するコンパイラが支援する広域分散共有メモリ機構のシステムを簡単に表しております。

4 研究成果:

4. 1 遠隔アクセスのレイテンシ削減

1)クラスタキャッシュ

共有メモリ機構の性能を向上させるためには、良く使用するデータは手元にキャッシュしておく必要があります。図1にあるように、WAN クラスタには3つの階層があり、PC 内部/LAN 内部/WAN 内部と分類されます。PC 内部では、CPU とメモリの速度のギャップを埋めるために、ハードウェアによる1次/2次キャッシュがあります。LAN 内部では、LAN アクセスとメモリアクセスの速度のギャップを埋めるために、ソフトウェアキャッシュがあります。本研究では、WAN 内部において、WAN のアクセス速度と LAN のアクセス速度のギャップを埋めるために、クラスタキャッシュを提案しました。

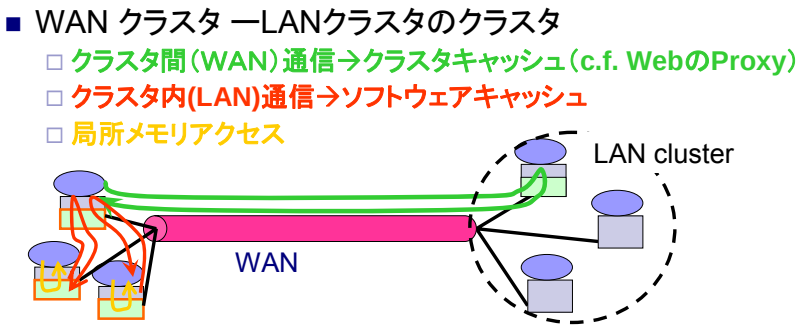


図3　クラスタキャッシュ

図3にあるように、LAN クラスタ（図左）は遠隔 LAN クラスタ（図右の点線で囲まれた部分）にあるデータを自分のクラスタにキャッシュすることで、できるだけ WAN 上の通信（図の緑色の部分）を起こさないようにします。以下に簡単に説明します。ある PC が遠隔クラスタにあるデータに初めてアクセスする場合、WAN 上の通信によりそれを入手します。しかし、それ以降、その PC と同じクラスタ内にある別の PC が同じデータにアクセスする場合には、遠隔クラスタから取ってくるのではなく、同一クラスタ内のクラスタキャッシュから取ってきます。全ての PC が同一データにアクセスするというのは並列計算では良く見受けられるので、この最適化は非常に効果的であるといえます。バリア同期に関しても、クラスタキャッシュと同様の最適化を適用しました。システムの階層性に着目して、LAN 内部でバリア同期を取ってから、WAN 内部でバリア同期を取るようにすることで、WAN 上の通信の回数を減少させました。

2）プリフェッチ機構

クラスタキャッシュ最適化は無駄な WAN 上の通信を削減する手法とみなすことができます。どうしても必要な一回目の WAN 上の通信（図3の緑色の通信）の遅延を削減するために、本研究ではアグレッシブなプリフェッチ機構を提案しました。プリフェッチは本来 CPU とメモリのスピード差を埋めるために開発された手法で、データの使用前にノンブロッキングなメモリアクセス要求を発行します。

WAN の通信遅延を考えると、少し先読みするだけでは不十分であるといえます。そこで、プログラムの意味を変えることなくできるだけ早くかつ無駄なくデータの要求を行う手法を提案しました。

1つ目は、部分冗長性削除のフレームワークを活用して、if 文や case 文のような条件節があってもプリフェッチをさかのぼって発行できるようにしました。図4はコンパイラによるコード生成例です。配列 a への読み出しは、従来の手法だと、if 節の先頭に挿入されていましたが、今回提案した手法では、更に遡ってバリア同期の直後に挿入されます。無駄な通信を誘発しないように、条件節つきのプリフェッチとなります。

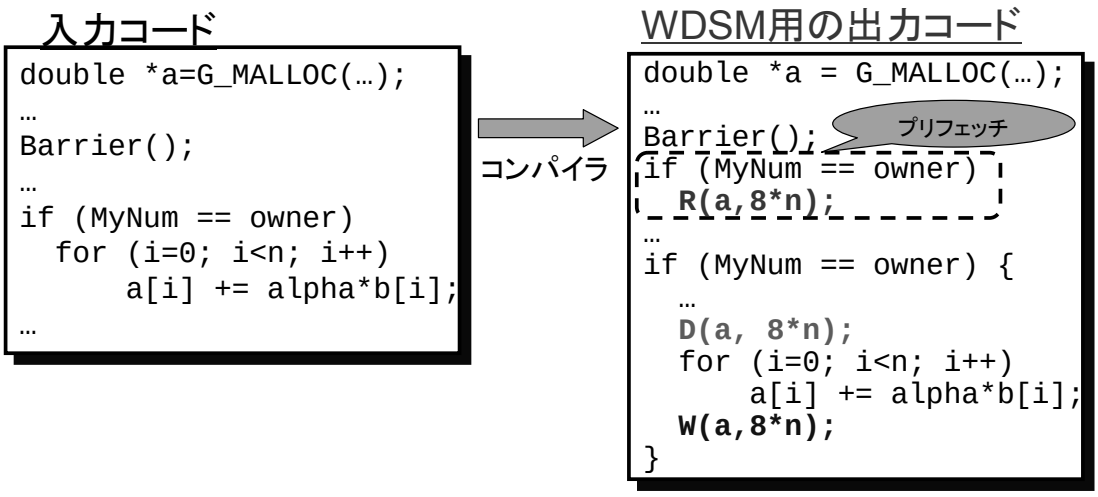
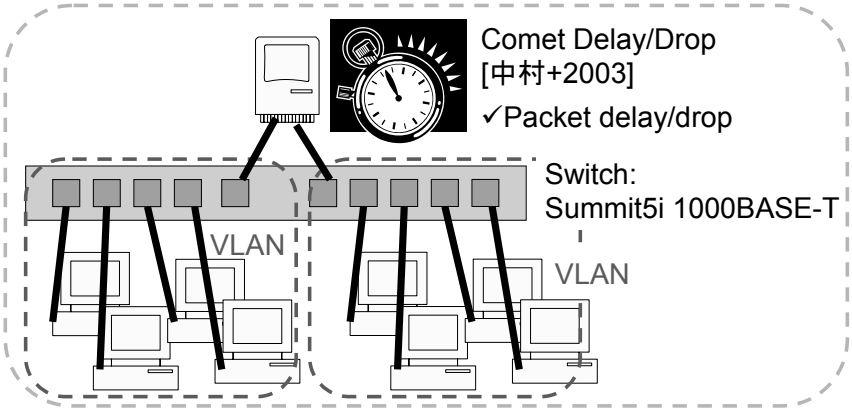


図4 コード生成例

2つ目は履歴の活用です。WDSM ではキャッシュミスを起こしたページ番号の履歴を残すことが可能です。そこで、バリア同期を使用して毎回同じ領域にアクセスするような反復計算(いわゆるループ)に関しては、履歴を活用することで更に通信の遅延を削減することができます。すなわち、最初の一回目のイテレーションでキャッシュミスを発行したページ番号を記録しておき、二回目以降のイテレーションでは、その情報を使用して、バリア同期の際にキャッシュブロックリクエストをまとめて発行することで通信の遅延を削減します。二回目以降のイテレーションでは、キャッシュのミス／ヒット判定も不要となるので，そのオーバヘッドも削減できます。

実 験

本方式の有効性を確認するために、以下の擬似広域環境で実験を行いました。図5に示してあるように、Summit5i を利用して VLAN を作成し、VLAN 間同士で通信する場合には、Comet Delay./Drop を経由するように設定しました。この Comet Delay/Drop が、ユーザが指定した間隔でパケットを遅延させたり、落としたり、バンド幅を制限したりします。WAN クラスタ(8台)は、2つの LAN クラスタ(4台)から構成されます。実行されるアプリケーションのサイズを考慮して、メトロポリタン地域で想定される通信遅延(0～10ミリ秒)を入れて実験を行いました。



Node:Dell PowerEdge1650, 1.26GHz Pentium III (512KB cache)
2GB Memory, 1000BASE-T

図5　仮想広域環境

使用したアプリケーションは SPLASH2 の密行列の LU 分解です（行列のサイズは 4096×4096）です。図6が実験結果を示しています。WDSM というのが今回提案した方式で、ADSM というのが従来の LAN で使用されてきた方式です。

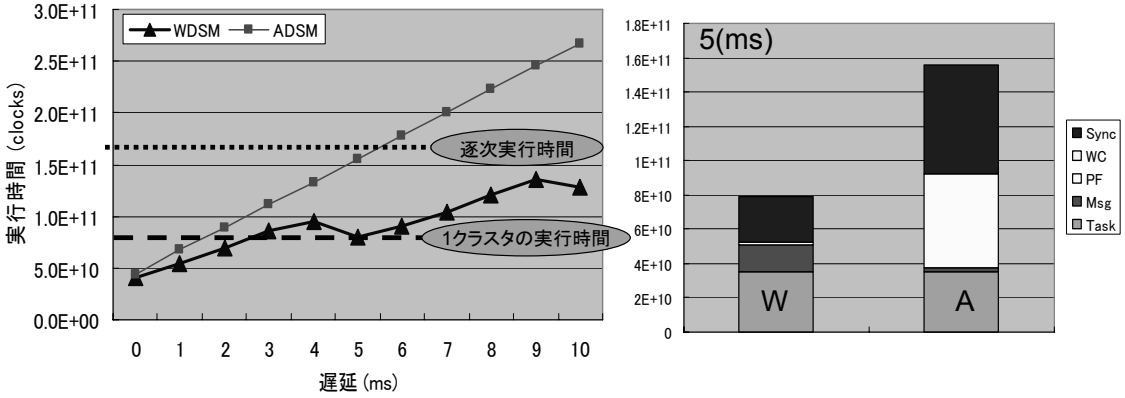


図6　遅延と実行時間の関係（左）と 遅延が 5（ms）の時の実行時間の内訳（右）

図6左から、WDSM は ADSM よりも遅延に対してロバストであることが分かります。図6右の棒グラフの各成分について簡単に説明します。Msg はメッセージの処理時間、PF はキャッシュミスの待ち時間、WC は書き込み後のコヒーレンス管理コードの実行時間、Sync はバリア同期の実行時間を表しています。残りの時間が Task になります。すなわち本来の計算時間と、プリフェッチの実行時間とを合わせたものです。左側の棒グラフが WDSM の結果で、右側のそれが ADSM の結果となります。WDSM は ADSM と比べて PF（キャッシュミスの待ち時間）が大幅に削減されていることが分かります。以上の結果から本研究で提案した最適化方式は効果的であるということができます。

4．2　スケーラブルな共有メモリ

32 ビットアーキテクチャの場合には、論理アドレスが 32 ビットしかないので、アドレス空間として 4 GB しか使用できません。すなわち、計算機を何台接続させたとしても、共有アドレスは最大 4 GB まで

しか確保できません。広域環境上で動作させたいアプリケーションの性質を考えると、4 GB では十分であるとはいえません。近年、64 ビットアーキテクチャ（例：AMD64, EMT64, Potwer5）の低価格化と高性能化が進んだことを受けて、本研究では、64 ビットアーキテクチャを計算機として仮定しました。64 ビットアーキテクチャの場合、仮想メモリサイズはほぼ無限大なので、計算機の台数に比例してスケーラブルに共有メモリを確保することができます。しかしながら、共有メモリサイズが物理メモリサイズを超える場合には、適宜（ソフトウェア／クラスタ）キャッシュをリプレイスする必要が生じます。

本研究では、キャッシュをロバストにリプレイスする手法を提案し、実装しました。ホーム（オリジナル）はリプレイスの対象にはなりません。無効なキャッシュは同期ポイントですでにアンマップされているので、あらためてリプレイスする必要はありません。有効なキャッシュがリプレイスの対象となります。その際、読み出しのみのキャッシュは、ただアンマップするだけで良いのですが、書き込まれた（る）キャッシュは、ただアンマップするだけではプログラムの正確性が失われる危険があります。すなわち、書き込んだ(書き込む)データだけ、ホームに書き戻してから、アンマップするようにします。

そのために、本研究では、書き込み前のコヒーレンス管理コードを導入しました。図4のコード生成例に示してあるように、配列 a に実際に書き込む前に、これから書くというコード（D(a, 8*n)）を挿入します。

書き込み前のコヒーレンス管理コード D は書き込まれるブロックのアドレスとサイズ（図4の例では a と 8*n）をリストに登録します。書き込み後のコヒーレンス管理コード W は書き込まれたブロックをホームに転送します。W はその際に書き込まれたブロックのアドレスとサイズの組をリストから削除します。キャッシュリプレイスが発生した場合には、リストに残っている組に関して、対応するブロックをホームに書き戻すことで、プログラムの正しさが保障されることになります。

実 験

本方式の有効性を確認するために以下の環境で実験を行いました。

1）PC：AMD Opteron242 (物理メモリ4GB) 8 台

2）ネットワーク：ギガビットネットワーク（NetGear GS524T）

3）OS：SUSE Linux9.0

各 PC ですが、ホームとして 2 GB 使用し、キャッシュとして 2 GB 使用するように設定しました。その結果、システム全体として、16 GB の共有メモリを扱うことができました。その上で、N 体問題のシミュレーションを行うツリーコードを動作させました。粒子数は6億 4 千万で、ステップ数は 32 です。必要とされる共有メモリは 10 GB で、一台の PC では実行不可能な問題サイズであるといえます。その実行結果が表1に示されています。Msg はメッセージの処理時間、PF はキャッシュミスの待ち時間、WC は書き込み前／後のコヒーレンス管理コードの実行時間、Sync はバリア同期の実行時間を表しています。残りの時間が Task になります。すなわち本来の計算時間に、プリフェッチの実行時間を合わせたものです。

表1　ツリーコードの実行時間とそのブレイクダウン

Total	Task	Msg	PF	WC	Sync
865(分)	769(分)	1(分)	52(分)	5(分)	38(分)

このアプリケーションでは、参照の局所性が高いためにキャッシュのリプレイスは発生しませんでした。この表から、WC の実行時間は非常に小さいということがわかります。従来のシステム(ADSM)で問題の規模が 8 分の1(粒子数 8 千万)の問題を解いた所、実行時間は 93.4 分かかりました。ツリー法の計算量は O(*nlogn*)であることが知られているので、計算量の比と実際の実行時間の比を比べると、今回導入した書き込み前のコヒーレンス管理コードのオーバヘッドの大小の判別の指標となります。結果は両者ともほぼ9となり、書き込み前のコヒーレンス管理コードのオーバヘッドは十分小さいものであるということがわかります。

4．3　効率的な耐故障機能

全体の PC の台数が増加すると、それだけ故障する台数も増加します。もし、何も対策を講じなければ、計算の途中で一台の計算機に不具合が発生した場合には、全ての計算機で、最初から再計算することとなります。それを防ぐために、今まで計算しておいた結果を保存する必要が発生します。すなわち、ある時点でのプログラムの実行状態を保存しておいて(checkpointing)、後になって、そこから、プログラムを再実行可能なようにしておく必要があります。Checkpointing に関する研究は長年に渡って行われていますが、本研究では、できるだけ効率良く実装するということに力点を置きました。

本研究では、既存の逐次計算に対する耐故障ライブラリ:libckpt を改良しました。まず、Dynamic library に対応できるようにしました。また、ソケットやファイルも保存することができるようにしました。状態管理の容易性から、Coordinated checkpointing を採用しました。すなわち、バリア同期の時に、全ノードが同時に checkpointing を行うようにしました。ただし、バリア同期の都度、checkpointing していてはオーバヘッドが増大する危険があります。そこで、タイマー機能を付加することにより、一定の時間間隔があかないと checkpointing は行わないように制御しています。

実 験

本方式の有効性を確認するために以下の環境で実験を行いました。

1)PC:Dell PoweEdge1650　(P-III 1.26GHz, 2GB メモリ)　4 台

2)ネットワーク:ギガビットネットワーク(Summit 5i)

3)OS:FreeBSD5 .1

4)アプリケーション:SPLASH2 の LU 分解(行列のサイズは 2048×2048)

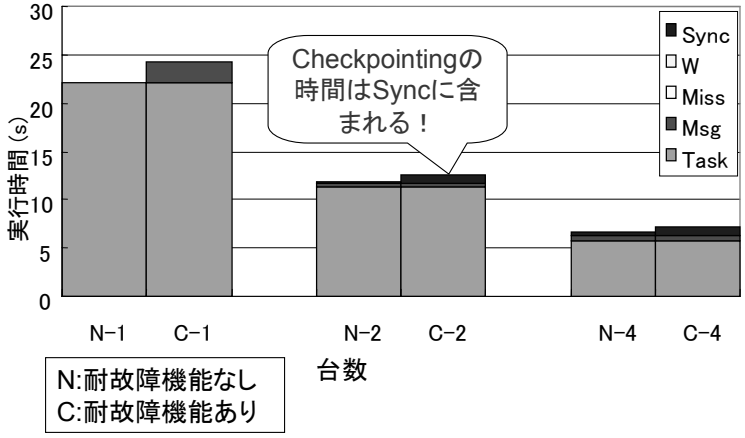


図7　Checkpointing のオーバヘッド測定

結果が図7です。横軸は台数、縦軸は実行時間を表します。2つの隣接している棒グラフの左(N-＊)が耐故障機能のない場合の結果で、右(C-＊)が耐故障機能のある場合の結果です。Msg はメッセージの処理時間、PF はキャッシュミスの待ち時間、WC は書き込み前／後のコヒーレンス管理コードの実行時間、Sync はバリア同期の実行時間を表しています。耐故障機能がついている方(右側の棒グラフ)では、checkpointing の時間は Sync に含まれています。残りの時間が Task になります。台数によらず、checkpointing のオーバヘッドは全実行時間の10%程度になっていることが分かります。

5　自己評価:

遠隔アクセスのレイテンシ削減に関しては、クラスタキャッシュとプリフェッチ機構の導入により、実際のアプリケーションに対して当初の予想以上の成果を収めることができました。プリフェッチに関しては、厳密にループ不変でなくても再帰的なアクセスをしている場合（例:N 体問題におけるツリー法でツリーをたどる場合）であっても、履歴の活用が有効であるということが分かりました。

今後は Grid 技術と融合して実際の広域環境上で実験を進めていきたいと思います。

スケーラブルな共有メモリの適用に関しては、当初は、低価格高性能な 64 ビットアーキテクチャが普及していなかったので、ソフトウェアによる64ビット共有アドレスから32ビット論理アドレスへの変換を想定していました。それに比べると64ビットアーキテクチャを使用することで、自然にオーバヘッドを削減できたと思います。どれだけ、キャッシュを確保するのがよいか？という点に関しては、論理的に最適な値を発見できませんでした。環境依存なので、実験によって求めていくしかないと思われます。

効率的な耐故障機能の提供に関しては、当初の予定通り、システムレベルのチェックポインティング（コアダンプ方式）の実装を行いました。どんなアプリケーションであれ、中断、再開することが可能になりました。実験を行うことで、実際のアプリケーションで鍵となるデータ構造の数はそんなに多くないということが分かりました。アプリケーションレベルのチェックポインティングの方が低オーバヘッドで実現できるのではないかと思われます。ただし、自動的に鍵となるデータ構造を発見するのは困難であるという問題点があります。そこで、ユーザからの指示を与えるという形を取ることで問題点を克服できるのではないか？と思います。 今後はシステムレベルとユーザレベルを融合したチェックポインティングを目指したいと思います。

6 研究総括の見解:

近年，大規模科学計算のために大量の計算機パワーが必要とされ，グリッド計算システムが注目されている．通常は高速LANで結合された計算機群が利用されるが，超高速広域ネットワークで結ばれた研究機関の計算資源を利用する次世代実験科学のための計算プラットフォームが今後重要になると考えられる．丹羽氏の研究は，広域環境中の共有メモリ並列プログラムを高速に動作させるための，最適化コンパイラやランタイムシステムの構築法に関する研究である．クラスタキャッシュやプリフェッチ機構，スケーラブル共有メモリ機構，効率的耐故障機能などの方式を考案，実装と評価を行い，それらの有効性を実証した研究である．このような研究は今後のさきがけとなるものであり，丹羽氏の研究は高く評価できるものである．

7 主な論文等:

論文

1. 丹羽純平，広域ソフトウェア分散共有メモリ機構を支援する最適化手法，
情報処理学会論文誌:コンピューティングシステム Vol.45 No. SIG 11(ACS 7) 36—49, 2004.年.
2. J. Niwa, Prefetch Mechanism in Compiler-Assisted S-DSM System,
CRTPC-04/ICPP-04, Montreal, Quebec, Canada, pp. 520—529, 2004/08/15--18.
3. J. Niwa, Compiler-Assisted Software DSM on a WAN Cluster,
Parallel and Distributed Computing: Applications and Technologies, LNCS vol.3320, pp. 815—828, (PDCAT 2004, Singapore, 2004).
4. 丹羽純平，コンパイラが支援するソフトウェア DSM におけるレイテンシ削減技法，
情報処理学会 ハイパフォーマンスと計算科学シンポジウム pp 81—88, 2005 年.
5. 丹羽純平，コンパイラが支援するソフトウェア DSM におけるレイテンシ削減技法，
情報処理学会論文誌:コンピューティングシステム Vol.46 No. SIG7 (ACS 10) 74—84, 2005 年