計算機アーキテクチャ 134－3
（1999．8．2）

RHiNETによる共有メモリプログラミングのサポート

建部修見[†] 工藤知宏[††] 山本淳二[††]
佐藤三久[††] 石川裕[††]

不規則アクセスを含む並列ループの効率実行のための実行手法を提案し，RHiNET, MPIを用いた実装，予備評価を行う．本手法は，multiple-writerプロトコルを用いたページベースのinspector-executor手法を基にしており，プリプロセスされたデータ量を削減している．予備評価では，大規模疎行列の行列ベクトル積を用い，問題サイズが大きい場合にはMPIによる実装でもスケーラブルな性能を示した．

Shared-memory programming support on RHiNET

Osamu Tatebe,[†] Tomohiro Kudoh,[††] Junji Yamamoto,[††]
Mitsuhisa Sato[††] and Yutaka Ishikawa[††]

We propose a new runtime solution for efficient execution of irregular applications and implement it using RHiNET or MPI. Our approach is based on a page-based inspector-executor model with multiple-writer protocol. Thanks to the page-base model, huge amount of preprocessed data can be reduced. Preliminary evaluations on sparse matrix-vector multiplication show scalable performance even with MPI if problem size is large enough.

1. はじめに

大規模疎行列ベクトル積などの不規則アクセスを含む並列ループの効率実行は，科学技術計算における行列解法，固有値解法などにおいて極めて重要である．この10年余りの間，不規則アプリケーションを分散メモリ型並列計算機において，効率的に並列実行する研究は盛んに行われている．不規則アクセスは，多くの場合入力データで与えられ，実行時にならないと参照関係を決定することができない．実行時にプリプロセスを行い最適実行する方式として，inspector-executor方式[4]が良く用いられている．Inspector-executor方式では，不規則アクセス並列ループをinspectorとexecutorに分割する．Inspectorでは分散した配列に対し通信が必要なデータ集合を求め，反復空間の分割を行う．executorではその集合に基づき通信し，計算を実行し，更に通信を行う．Kaliコンパイラ[4]ではinspectorで通信が必要なデータ集合を求めるだけではなく，通信が必要ない反復集合も求め，通信遅延の隠蔽も行っている．CHAOS/PARTI[3]はinspector-executor実行を支援するためのランタイムライブラリであり，様々なコンパイラによってもそのランタイムライブラリは用いられている．が，inspectorが重い操作であり，またinspectorの結果のデータが莫大であるため，複雑な関数間解析を行い，データ再利用の最適化[1]が必要となる．

また，ソフトウェア分散共有メモリ(DSM)技術を用いることにより，不規則アクセス並列ループのナイーブな実行は可能である．しかしながら，基本的にはソフトウェアDSMはオンデマンド方式であり，効率的な実行のためには，コンパイラによるプリフェッチ，メッセージ集合化などの最適化，あるいはinspector-executor方式を元にした実行時の最適化などが必要である．ソフトウェアDSMに，間接参照のvalidationを付加することにより，関数間の複雑な解析なしに，CHAOSランタイムライブラリを用いたものに比べ同等以上の性能を得たという報告もある[7]．

本研究では，multiple-writerプロトコルを用いたページベースのinspector-executor方式を利用した不規則アクセス並列ループの新しい実行方式を提案する．そして，RHiNET[5),9)]あるいはMPIを用いた実装についての考察を行い，疎行列ベクトル積において予備評価を行う．

2. RHiNET

RHiNET[5),9)]は新情報処理開発機構で開発中の，計算機間を接続して効率の良い並列処理を行うためのネットワークであり，ネットワークインタフェースはPCIバ

† 電子技術総合研究所
Electrotechnical Laboratory, AIST, MITI
E-mail: tatebe@etl.go.jp
†† 新情報処理開発機構
Real World Computing Partnership
E-mail: {kudoh,junji,msato,ishikawa}@rwcp.or.jp

スに接続される．push(遠隔メモリ書き込み) /pull(遠隔メモリ参照)/ マルチキャストの片側通信，マッチングを含む一対一通信をプリミティブとして提供するだけではなく，アドレス変換，diff の作成，directory 機能など共有メモリ空間をサポートするためのプリミティブも提供されている．

片側通信，マルチキャストに関して，書き込みアドレス，読み込みアドレスは通信エリア ID + オフセットで与えられる．通信エリア ID は，MPI-2 の window に相当するものであり，それぞれのプロセスにおける共有データの先頭アドレスへの変換を行う．

diff の送信は，遠隔メモリ参照によりホストメモリ上にデータを読み込む際に，NIC 内の twin メモリに参照内容のコピーをおき，変更後書き戻す際に，ホストメモリの内容と NIC 上のメモリの内容を比較して変更されている部分のみを書き戻すことで実現している．

RHiNET-1 は 1.3Gbps の光コネクションもしくは GBIC(Gigabit Ethernet で用いられているインタコネクション) を用い構成され，8 Gbps クラスの光インタコネクションを用いた RHiNET-2 も開発中である．

3. 不規則アクセス並列ループ

本研究で扱うループは，多段の間接参照，インデックス計算に任意の式を含む (多重の) 並列ループである．また，並列ループに関しては OpenMP などの並列指示行により並列の指示がされているものとする．例えば，以下のようなプログラムである．

```
#pragma omp parallel for
for (i = 0; i < n; ++i) {
  for (j = 0; j < ii[i]; ++j)
    a[ia[j]][i] = b[ib[j] + 2] + c[i*j*j];
}
```

このプログラムでは，外側のループが並列実行可能なことが OpenMP の pragma で示されている．このプログラムでは ii, ia, ib がインデックス配列として使われ，配列 c は非線形なアクセスが行われてる．

4. 不規則アクセス並列ループの効率実行

不規則アクセスを含む並列ループを効率的に実行するためには inspector-executor 方式を用い，ランタイムライブラリ[3] あるいは同等の実行時処理を行うコンパイラを用いる．この場合，分散配列のアドレス変換テーブル，配列の分散情報により分割される反復空間の情報，送信データ情報，受信バッファ情報および大域アドレスから受信バッファアドレスへの変換テーブルなど，それぞれのループにつき大量のデータが必要になってしまう．

ソフトウェア DSM は基本的にページベースのオンデマンド方式のもの[2] と，load/store 命令に対し，一貫性維持コードを埋め込んでいるものとある．特に，並列ループに関してはオンデマンドにデータを持ってきたり，ループ中に一貫性維持コードがあったりすると，性能が低くなってしまう．そのため，効率的な実行のためには，プリフェッチ，メッセージ集合化などの最適化が必要とある．規則的な並列ループの場合は，正規区間解析，関数間解析などを用いることにより，コンパイラによる静的解析がある程度可能となっている．しかしながら，間接参照などを含む不規則的な並列ループの場合は，参照関係が実行時に定まるため，コンパイラによる静的解析を行うことができず，inspector-executor 方式などのような実行時最適化が必要となる．

そこで，我々は inspector-executor 実行方式に基づく，multiple-writer プロトコルを用いたページベースの実行時最適化機構についての提案を行う．

4.1 Inspector

Inspector フェーズでは，OpenMP の指示行などで決められた反復空間の分割に伴い，それぞれの分散配列についてホームと共有関係を決定し，また共有ページへの書き込み情報を調べる．

ホームと共有関係の決定に関しては，割り当てられた反復空間に対し，ページ単位のアクセス頻度を利用し，頻度が最多のプロセスをホームとする．間接参照の場合は，インデックス配列のホーム，共有関係を決定し，決定した共有関係に変更，更新し，再び間接参照されている配列のホームと共有関係を決定する．間接参照が多段にある場合は，内側の配列から同様の処理を順次行うことになる．

共有ページへの代入に関しては，データ更新のための書き込み情報を保持する必要がある．

ページの共有 (参照) 関係については，ディレクトリ (あるいはリスト) を作成することになる．現在はフルマップディレクトリを`MPI_Allreduce(MPI_BOR)` などを用いることにより作成している．

4.2 Executor

Executor フェーズでは以下を行う．

(1) 分散配列の共有関係の変更，更新．
(2) ループ実行．
(3) データ更新．
(4) バリア．

ループの実行に関しては，割り当てられた反復空間に対し，実行前に参照するデータは全て揃っていることが保証されている．従って，共有関係のチェックなしで private メモリのみのアクセスによりループを実行することができる．

4.3 RHiNET による実装

RHiNET は twin メモリを利用した diff 作成のハードウェアサポートを備えているため，送信データリストは作成する必要がない．この場合，Inspector は以下のようになる．

(1) 割り当てられた反復空間に対し，ページ単位のアクセス頻度表，参照表を作成する．既に共有関係の決定している配列の共有ページへの代入に関する書き込みページリストを作成する．

(2) アクセス頻度が最多のプロセスをホームとし, (フルマップ) ディレクトリを作ることによりページの参照 (共有) 関係を決定する.

(3) 間接参照などにより, 共有関係の決定していない分散配列がある場合, もしくは代入される配列の書き込みページリストが作成されていない場合は, 必要な分散配列の共有関係の変更 (validation) を行い (1) に戻る.

ページの共有関係が変化した場合は, 新たに共有するプロセスはそのページが書き込みページであれば `pull_with_twin` を用いる. `pull_with_twin` はページを以前のホームから持ってくると同時に twin を作成する. 書き込みページでなければ単に `pull` を用いる. また, 以前から共有しているページであっても, 書き込みページとなった場合は`pull_with_twin` を用い twin を作成する必要がある.

Executor フェーズにおけるデータの更新では, 共有ページへ書き込みを行うページに対し `mcast_diff` (あるいは `push_diff`) を用い変更された部分のみを送ってデータを更新する. バリアはこれら片側通信の終了を確認するために必要である.

分散配列のデータについては, 分散配列当り, ページ数 (32 bits), 通信エリア ID(32 bits) が必要であり, またページエントリ当り, アクセスカウント兼ホーム (32 bits) と共有情報 (32 bits*) が必要である. 書き込みページを示すためにページ毎に 1 bit 必要であるが, この 1 bit に関してはアクセスカウント兼ホーム情報に含めてしまうか, あるいは書き込みページの配列を持つ.

4.4 MPI による実装

MPI でも TreadMarks などと同様に diff をソフトウェア的に作成することは可能であるが, inspector により送信データをあらかじめ定めることが可能であり, また更に inspector 時にデータの集合化などの最適化も可能であるため, 送信データのアドレスとサイズのリストを作成する. また, MPI-2 では片側通信を用いることができるが, MPI-1 にはないため, 受信データリストの作成が必要となる. Inspector は以下のようになる.

(1) RHiNET 同様, ページ単位に頻度表と参照表を作成する. 既に共有関係の決定している配列の共有ページへの代入は, 送信 (代入) データ (アドレスとサイズ) リストを作成し, 集合化 (aggregation) を行う.

(2) RHiNET 同様, ホーム, 共有関係を決定する.

(3) (MPI-1 のみ) 送信データリストを作成した場合, 受信データリストを作成する.

(4) RHiNET の (3) と同様.

MPI-1 の受信データリストの作成は, ページ毎に, 共有プロセス間において, 送信回数 (連続している送信データブロックの数) の `all_gather` および送信データリストの `all_gatherv` を行う. これらの gather は全プロセス間ではなく, ページを共有しているプロセス間でのみ行われる.

送信データリストの集合化は, データ数を N とした時, $O(N \log N)$ で整列を行い, その後, 線形にアドレスとサイズにより共通部分, 連続部分を結合することにより $O(N \log N)$ で集合化が完了する.

ページの共有関係が変化した場合は, 新たに共有するプロセスは以前のホームよりページを持ってくる.

データ更新は, 共有ページへの書き込みに対し MPI-2 の場合は, 送信データリストを元に `MPI_Put` を用い, MPI-1 は, 送受信データリストを元に一対一通信で行う. バリアは MPI-2 の場合 `MPI_Put` などの片側通信の終了確認のために必要であるが, MPI-1 では一対一通信でデータ更新を行うため必要ない.

分散配列のデータについては, RHiNET の時の情報に加え, ページエントリ当りの送信データリストが必要となる. 更に MPI-1 ではページエントリ当りの受信データリストが必要となる. なお, RHiNET の通信エリア ID の代わりに, MPI-2 では window, MPI-1 ではベースアドレスが必要となる.

5. Inspector フェーズの実行時最適化

Inspector によりプリプロセスされたループ情報 (分散配列の共有情報など) は, 二度目以降の executor で利用することができ, 二度目以降に関しては最適化された効率的な実行ができることになる. しかしながら, ここで再利用可能かどうかは, インデックス配列の内容, ループの範囲が変化していない場合である.

インデックス配列の内容が変化したかどうかは, 関数間解析などにより静的に調べる[1] か, インデックス配列を書き込み禁止にしてページトラップにより動的に調べる[7] か, あるいはプログラマの責任として新たに pragma を追加する方法が考えられる. pragma は, インデックス配列の内容の変化はないという宣言を意味し, この宣言により, ループ情報の再利用はインデックス配列のアドレスのチェックで行うことができる.

また, 全ての分散配列の共有関係が, 現在の共有関係と同じか, あるいは部分集合である場合, executor による共有関係の変更, 更新は必要ない. 共有情報のポインタが等しい場合のチェックは簡単だが, 等しくない場合は, 一度チェックを行い, 等しいあるいは部分集合である場合は, それらのアドレスをキャッシュしておくことにより, 二度目からのチェックを $O(1)$ で行うことができる.

送信 (書き込み) データリスト (と受信データリスト) はループ毎に保存することになるが, 共通な場合は共通にしてしまい保持するデータを削減することができる.

RHiNET で実装する場合は, diff の作成にハードウェ

---

* 現在はフルマップディレクトリを用いているため 32 bits では 32 プロセッサしかサポートできない. ディレクトリについてはこれから検討の余地がある.

```
void
vec_mult(n, idx, ib, a, b, c)
    int n, *idx, *ib;
    double *a, *b, *c;
{
    int i, j;

    #pragma omp for private (j)
    for (i = 0; i < n; ++i) {
        double t = 0;
        for (j = ib[i]; j < ib[i + 1]; ++j)
            t += a[j] * b[idx[j]];
        c[i] += t;
    }
}
```

図 1 疎行列ベクトル積のプログラム

アサポートがありしかも動的に生成するため, 共有関係が等しい, あるいはほぼ等しい場合は, 共有関係のマージを行うことにより保持するデータ量を削減することができる. 共有関係が大きく違うものとマージしてしまうと, ページを共有するプロセスが増えてしまうが, どれくらいの共有プロセスまでマージしてもシステムとしての性能が落ちないかは, システム依存であり解析の必要がある.

Executor におけるバリアは, 片側通信の終了を確認するためのものであるが, バリアを split-phase にし, 静的解析によりその完了を遅らせることもできる.

5.1 コード変換例

この節では行圧縮形式 (Compressed sparse raw format) で格納された行列とベクトルの積 (図 1) を元に, 提案手法による inspector-executor ループへの変換, および実行時の inspector 結果の再利用のためのコード変換について考える. 行圧縮形式は, 行毎に圧縮して非ゼロ要素を並べた配列 `a`, それぞれの要素の行数を示す配列 `idx`, 第 i 列の先頭アドレスを示す配列`ib`, 第 i 列の最終アドレスを示す配列 `ie` で構成される. `a` が行毎に連続的にならんでいる場合は `ie` は省略でき, 図 1 でも省略している.

このプログラムを図 2 の様に変換する. OpenMP の場合, ループ範囲としてブロック分割, サイクリック分割などの静的な分割の他に, ガイド分割などダイナミックな分割も指定できるようになっているが, 反復区間の分割が動的に変わると, 変わるたびに inspector を実行する必要があり, 効率が悪くなる.

6. 予備評価

6.1 疎行列ベクトル積

高温超伝導体など強相関電子系における, 量子多体効果を取り扱うためのモデルの一つにハバードモデル[10] がある. このハバードモデルのハミルトニアンの固有値, 固有ベクトルを求めることにより, 基底状態, 励起状態のエネルギーやさまざまな物理量を計算することができる[6),8)]. また, それらの計算結果のサイズ依存性を調べ, 無限系へ外挿する事により超伝導, 磁性などといった固体電子の示す巨視的な現象を理解することができる. ハミルトニアン行列は大規模疎行列であるため, ランチョス法などの反復解法を用い固有値を求めることとなるが, 反復解法においては行列ベクトル積が特に重要となる. 図 3 に 8 サイト 6 電子の場合のハミルトニアン行列の非ゼロ要素パターンを示す. この時, 行列サイズは 3136, 非ゼロ要素数は 29456 である.

この行列に対し, 図 2 の行列積のプログラムを SGI Origin2000 で実行させた時の executor の時間 (二度目以降の行列積の時間) および計算だけに要した時間を図 4 に示す. ランタイムライブラリで用いたページのサイズは 4K バイトである.

1 プロセスでは 61 MFlops である. この問題の場合はページのサイズに比べ問題が小さ過ぎるため, データ更新の割合が 2 プロセスで 15%, 4 プロセスで 34% にもなっている.

次に, もう少し問題サイズを大きくし, 12 サイト 10 電子の問題を取り上げる. この場合, 行列サイズは 627264, 非ゼロ要素数は 8593992 である. 図 5 に同様に executor の時間と計算だけに要した時間を示す. 1 プロセスでは 32 MFlops であった. この問題の場合は, executor の実行時間と計算だけに要した時間に殆んど差が見られず, 1% から 2% 程度の差であった. データの更新により転送されるデータは配列 `c` のみであり, 外側のループをブロック分割することにより `c` もブロック分割される. 配列 `c` の分割がページ境界であれば通信は起こらず, そうでなければ高々ページサイズ 4K バイトの通信が隣のプロセスと起きるだけであるからである.

2 プロセス以上に対し, 計算だけに要した時間が理想的な時間より 1.18 倍から 1.25 倍かかっている理由は良く分かっていない.

6.2 RHiNET における実行の予測

RHiNET では, NIC 上の twin メモリにデータがヒットしている場合 diff を作りながらの送信と通常の送信に処理時間の差はない. 現在開発中の RHiNET では, n ワードの遠隔メモリ書き込みにおよそ $5 + 0.03n$ μsec. であることが予測されている.

6.1節のプログラムの場合, executor における通信が必要な配列は `c` だけであり, 共有ページもブロックの境界のページ (最悪でも 4K Byte) だけである. 従って, データの更新に関するデータ転送時間は高々 35 μsec. ということになる. 6.1節の 8 サイトの問題で SGI Origin2000 の MPI を用いたものはデータ更新に 130 - 150 μsec. もかかっており, 仮に, RHiNET のデータ更新に 40 μsec. かかったとしても, 4 プロセスの場合 Origin2000 では通信他のオーバヘッドが 34% なのに

```
void
vec_mult(n, idx, ib, a, b, c)
  int n;
  DSM_Int_array idx, ib;
  DSM_Double_array a, b, c;
{
  ループ範囲の決定.
  if (該当のループの配列情報がないあるいは
      index 配列のアドレス変更があるかあるいは
      ループ範囲の変更があるか) {
    /**  Phase I: Inspector  **/
    必要数の新しい分散配列情報の生成.
    配列 ib の参照解析 (1) (例)
    for (i = l_s; i < l_e; i += l_st) {
      access(&ib->e[i], ib_info);
      access(&ib->e[i + 1], ib_info);
    }
    分散配列 ib のホームと共有情報の決定.
    分散配列 ib の共有情報の変更, 更新.
    配列 a, idx, c の参照解析 (2)
    分散配列 a,idx,c のホームと共有情報の決定.
    分散配列 a,idx,c の共有関係の変更, 更新.
    配列 b の参照解析と配列 c の代入解析 (3) (例)
    for (i = l_s; i < l_e; i += l_st) {
      for (j=ib->e[i]; j<ib->e[i+1]; ++j)
        access(&b->e[idx->e[j]], b_info);
      add_send(&c->e[i], double, c_info);
    }
    分散配列 b のホームと共有情報の決定.
    配列 c の送信リストの集合化.
    配列 c の受信リストの作成.
  }
  /**  Phase II: Executor  **/
  分散配列 idx,ib,a,b,c の共有情報の変更, 更新.
  計算の実行 (例)
  for (i = l_s; i < l_e; i += l_st) {
    double t = 0;
    for (j=ib->e[i]; j<ib->e[i+1]; ++j)
      t += a->e[j] * b->e[idx->e[j]];
    c->e[i] += t;
  }
  データの更新.
  バリア.
}
```

図 2　変換後のプログラム

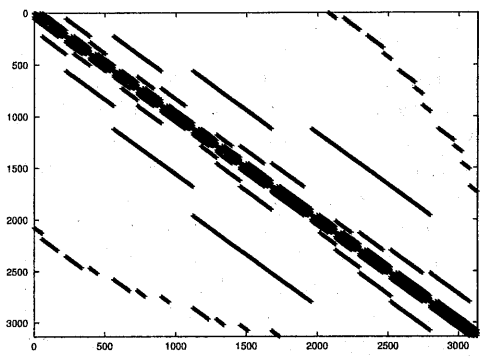

図 3　ハミルトニアン行列の非ゼロ要素パターン

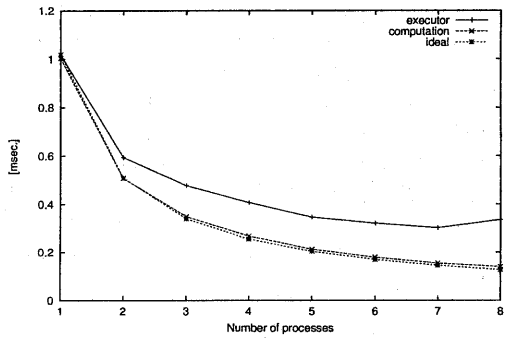

図 4　ハミルトニアン行列積の executor 実行時間 (1)

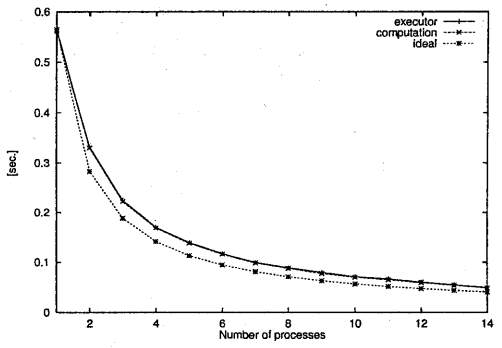

図 5　ハミルトニアン行列積の executor 実行時間 (2)

対し, RHiNET の場合は 13% となる.

7. 関連研究

CHAOS や Kali コンパイラで用いられている inspector-executor 方式では, まず何らかの方法でデータを分割し, owner-computes ルールあるいは almost-owner-computes ルールにより反復空間の分割を行う. almost-owner-computes ルールとは, 多数アクセスする要素もつプロセッサが計算を行うルールである. その後, inspector により通信するデータを調べている.

我々の方法は, 反復空間をまず分割し, inspector によりページの参照関係を調べ, 共有ページに対する書き込みデータを調べる. Executor により, その参照関係の変更, 更新を行い, ループを実行し, 書き込みデータの更新を行う. 参照関係をページ単位としたことにより, CHAOS のアプローチに比べ分散配列のアドレス変換表が必要なく, また通信データの集合 (通信スケジュール) も小さくなっている. また更に反復空間の分割情報も保持する必要がない.

TreadMarks のランタイムに間接参照の validate 機能をつけ, ループの前にそれらの validation を行うアプローチ[7] は我々の方法に近いが, 我々はソフトウェア DSM を用いず, MPI を用いたランタイムライブラリあるいは RHiNET のハードウェアの diff 作成機能を用いているところが異なっている. が, これらの方法との比較は必要である.

8. まとめと今後の課題

不規則アクセスを行う並列ループの効率的な実行方式の提案, 実装, 予備評価を行った. 我々の方法は, 反復空間を規則的に分割するため, 反復空間の分割に関する情報が必要ない. また, ページベースのため, 分散配列に対し CHAOS のようなアドレス変換表が必要ない. MPI を用いた Origin2000 による予備評価では, 大規模な問題に対しスケーラブルな性能を示した.

これから, RHiNET のプリミティブを用い実機における評価, MPI-2 ベースのランタイムライブラリの作成をしていくつもりである. また更に, OpenMP コンパイラにこれらのランタイムを組み込んでいきたい.

謝　辞

RHiNET に関する議論を通じて貴重な御意見をいただいたシナジェテックの清水敏行氏に感謝致します. ハバードモデルに関するプログラムの提供および理論について御教授いただいた電子技術総合研究所電子基礎部の浅井美博氏に感謝致します. また, 本研究を遂行するにあたり貴重なご助言, ご討論いただいた電子技術総合研究所大蒔和仁部長, HPC グループ, 新情報処理開発機構, 筑波大学, 電子技術総合研究所 TEA グループ, 電子技術総合研究所並列アーキテクチャラボのメンバ諸氏に感謝致します.

本研究は電子技術総合研究所と新情報処理開発機構の「リアルワールドコンピューティングの基礎研究」に関する共同研究契約に基づいている.

参考文献

1) Agrawal, G. and Saltz, J.: Interprocedural Communication Optimizations for Distributed Memory Compilation, *Proc. 7th Workshop on Languages and Compilers for Parallel Computing*, pp. 283–299 (1994).

2) Amza, C., Cox, A. L., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaenepoel, W.: TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, No. 2, pp. 18–28 (1996).

3) Das, R., Uysal, M., Saltz, J. and Hwang, Y.-S.: Communication Optimizations for Irregular Scientific Computations on Distributed Memory Architectures, *Journal of Parallel and Distributed Computing*, Vol. 22, No. 3, pp. 462–479 (1994).

4) Koelbel, C. and Mehrotra, P.: Compiling Global Name-Space Parallel Loops for Distributed Execution, *IEEE trans. of Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 440–451 (1991).

5) Kudoh, T., Yamamoto, J., Ishikawa, Y., Sato, M., Sudoh, F. and Amano, H.: Memory based light weight communication architecture for local area distributed computing, *Proc. IWIA'97* (1997).

6) Leung, P. W., Liu, Z., Manousakis, E., Novotny, M. A. and Oppenheimer, P. E.: Density of states of the two-dimensional Hubbard model on a 4 × 4 lattice, *Phys. Rev. B*, Vol. 46, No. 18, pp. 11779–11786 (1992).

7) Lu, H., Cox, A. L., Dwarkadas, S., Rajamony, R. and Zwaenepoel, W.: Compiler and Software Distributed Shared Memory Support for Irregular Applications, *Proc. PPoPP'97*, Vol. 32, No. 7, pp. 48–56 (1997).

8) Shiba, H. and Ogata, M.: Properties of one-Dimensional Strongly Correlated Electrons, *Progr. Theor. Phys. Sup.*, Vol. 108, pp. 265–286 (1992).

9) 工藤 知宏, 山本 淳二, 建部 修見, 佐藤 三久, 西 宏章, 天野 英晴, 石川 裕: PC 間ネットワークによる共有アドレス空間を持つ並列処理システム, 情報処理学会研究報告 99-ARC-132, pp. 121–126 (1999).

10) 山田耕作: 岩波講座 現代の物理学 16 電子相関, 岩波書店 (1993).