サイエンティフィック・システム研究会　1999 年 8 月 20 日
科学技術計算分科会 第 1 回会合資料

計算物性物理におけるパラレルコンピューティング

東京大学物性研究所
物質設計評価施設
藤堂 眞治
wistaria@issp.u-tokyo.ac.jp

1. はじめに

東京大学物性研究所では平成 6 年度に VPP500 / 40 を導入し、平成 7 年度から全国共同利用プロジェクトを通じて、全国の物性物理研究者により利用されている。その目的は、通常の大型計算機センターでは実行できない先端的な大規模計算をサポートし、日本の物性科学研究における計算物理学の発展を図ることにある。

計算物性物理における計算手法は、第一原理電子状態計算、数値的厳密対角化法、モンテカルロ法、分子動力学法など非常に多岐にわたる。また、計算物性物理の特徴の一つとして、新しいアルゴリズムの開発そのものが新しい研究と密接に結びついている場合も非常に多い。すなわち、研究者一人一人が各自のプログラムを開発しつつ、そのプログラムを用いた大規模計算を実行することになる。したがって、物性研の VPP500 も、単なるアプリケーションサーバとして利用されるのではなく、新しいアルゴリズムを用いたプログラムの開発環境としても重要な役割を果たしている。

一般に、計算物性物理のシミュレーションにおいては、シミュレーションする系の物理的なパラメータ (温度等) に関してパラメータスキャンを行なうことが多い。また、ランダムネスをもつ系を扱う場合には、数十から数千のサンプルを独立にシミュレーションし、それらについての平均 (ランダム平均) を取ることになる。このような場合には、パラメータ (あるいはランダム平均) に関する自明な並列化 (いわゆる embarassingly parallel) が最適な並列化である。しかし、

(1) 1 プロセッサの主記憶容量をはるかに越える主記憶容量 (数十 GB〜数百 GB) が必要となる場合
(2) 1 プロセッサでの実行では、数日 (〜数年) を越える時間が必要となる場合

には、非自明な並列化が必要となる。ここでは、一番目の例として、古典スピン模型の自由エネルギーの計算に使われる「転送行列の厳密対角化法」、二番目の例として、量子スピン模型の物理量を確率的に求める「量子モンテカルロ法」を取り上げ、我々が実際に行なった並列化についてその手法を具体的に紹介し、VPP500 での性能評価、他機種との性能比較の結果を報告する。

2. 転送行列の厳密対角化法の並列化

2.1 転送行列の厳密対角化法

「二次元古典イジング (Ising) 模型」と呼ばれる模型を考える。この模型では、二次元の正方格子の各格子点に「1 (上向き)」、もしくは「-1 (下向き)」のどちらかの値を持つ「スピン」が存在し、それぞれの隣り合う二つのスピンがお互いに同じ向きを持つか、反対の向きを持つかによって、異なるエネルギーが与えられる。この模型は、現実の磁性体の相転移を記述する最も単純な模型の一つとして、非常に詳しく研究されてきた。この系の自由エネルギー (free energy) を (計算機の数値精度の範囲で) 厳密に求めようとすると、もっとも素直な方法では、位相空間の大きさと同じ、$O(2^N)$ の手間がかかる。(ここで N は系に含まれるスピンの個数である。$L \times L$ の系を考える場合、計算の手間は $O(2^{L^2})$ となる。) この計算の手間は、以下に述べる「転送行列 (transfer matrix) 法」を用いることにより、$O(L^2 2^L)$ にまで減らすことができることが知られている。

転送行列法では、次元が $2^L \times 2^L$ の行列を考える。この行列は、二次元格子を一方向にスライスした時の各スライスの自由エネルギーへの寄与を表わしている。この転送行列を L 回掛け合わせた後、トレースを取ることにより、系の分配関数 (partition function) が、さらにその対数を取ることにより、自由エネルギーが求められる。

転送行列法のもう一つの利点は、半無限系 ($L \times \infty$) を扱うことができるという点にある。転送行列の無限回の積を考えると、自由エネルギーに寄与するのは、転送行列の最大固有値のみとなる。また、最大固有値と第二固有値の比から、系の相関長を求めることもできる。こうして、古典イジング模型の半無限系の自由エネルギーを求めるという問題は、「転送行列の固有値問題」へと帰着する。一般に、転送行列は $2^L \times 2^L$ の正定値の実対称行列である。全ての行列要素の保存には $O(2^{2L})$ の主記憶容量が必要であり、べき乗法、ランチョス法などの反復法を用いて最大固有値を求めるとしても、$O(2^{2L})$ の計算時間が必要となる。したがって、一台のプロセッサを用いた場合には、$L = 14$ が実質的な限界となる ($2^{2\times 14} \times 8(\mathrm{byte}) = 2(\mathrm{GB})$)。

しかし、系の持つ物理的特性 (すなわち、相互作用が近距離の 2 体相互作用の和からなる) を利用することによって、転送行列は L 個の「疎な」行列の積に書き直すことができる。それぞれの疎行列は、各行 (各列) で対角成分と一個の非対角成分を除いては全て零となっている。この疎行列分解を用いると、反復法に必要な、ベクトルと行列の積の計算にかかる手間は $O(L \times 2^L)$ にまで減少する。さらに、行列を記憶するために必要な主記憶容量も同程度にまで減少する。(零でない行列要素を「その場で」計算すれば、必要な主記憶容量を $O(2^L)$ にまで減らすことも可能である。) この方法により、1 プロセッサで最大 $L = 26 \sim 28$ 程度の大きさの系まで扱うことができる。

2.2 並列化

ここでは、疎行列の要素をあらかじめ主記憶上に記憶しておくのではなく、全てその場で計算する場合について考える。この場合、行列とベクトルの積の計算を行なうのに必要となる主記憶容量は、$N_{\mathrm{d}} = 2^L$ の長さのベクトル一本分のみである。(計算結果をもとのベクトルに記憶することにより一本で済む。) このベクトルを各プロセッサに均等に分割して配置する。ただし、プロセッサ数 (N_{p}) は 2 のべき乗であると仮定する。反復法を行なうのに必要な、ベクトルのノルム計算、ベクトルの内積計算、ベクトルの定数倍等の操作についての並列化は自明であるので、以下、行列とベクトルの積の計算についてのみ考え

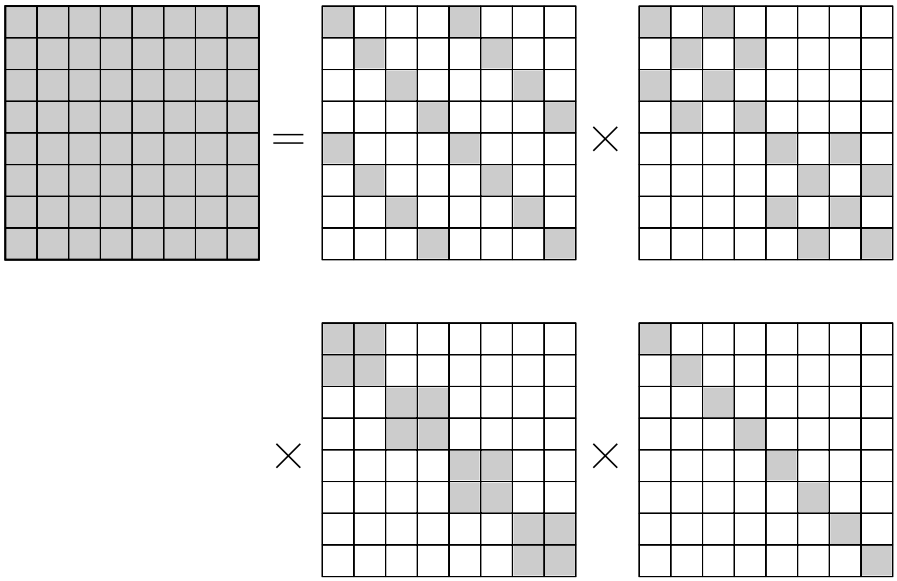

図 1: 転送行列の疎行列への分解 ($N_p = 8$ の場合)。小さな正方形は、それぞれ $(N_\mathrm{d}/N_\mathrm{p}) \times (N_\mathrm{d}/N_\mathrm{p})$ の行列を表わす。灰色で示した部分行列にのみ零でない行列要素が存在する。一方、白で示した部分行列の行列要素は全て零である。

ることにする。

転送行列全体としては密であるので、最も単純に考えると、各プロセッサがそれぞれに割り当てられたベクトルについての計算を行なうためには、$N_\mathrm{d}(N_\mathrm{p}-1)/N_\mathrm{p}$ の量のデータをネットワーク経由で他のプロセッサから取り寄せなければならないように思える。しかし、すでに述べた疎行列分解の方法をつかうと、以下に示す通り、データ転送量は $2N_\mathrm{d}(\log_2 N_\mathrm{p})/N_\mathrm{p}$ で済むことがわかる。

図 1 に、$N_\mathrm{p} = 8$ の場合の転送行列の疎行列分解を示す。もともとの転送行列は密であるが、四つの疎行列に分解した後では、それぞれ、対角部分行列とそれ以外にただ一つの零でない要素を持つ部分行列からなっていることがわかる。したがって、各疎行列の積のステップでは、各プロセッサは、それぞれペアを作って、ベクトルのトランスポーズ転送を行ない、各部分積を計算した後、逆トランスポーズ転送を行なうことになる。転送パターンは、FFT などでおなじみのバタフライ演算と同じパターンであり、一プロセッサあたりの転送量は、合計 $2N_\mathrm{d}(\log_2 N_\mathrm{p})/N_\mathrm{p}$ となる。

さらに、「動的領域分割」の手法を用いることにより、転送量を (漸近的に) 半分にすることができる。上記で述べた並列化では、各プロセッサの受け持つ部分ベクトルの割り当ては静的であった。計算の終わった部分ベクトルをもとのプロセッサに送り返す代わりに、各プロセッサへの割り当てそのものを動的に変化させることにより、二度目のトランスポーズ転送は不要になる。この方法では、最終的な辻褄を合わせるために一回の転送が余分な必要となるので、一プロセッサあたりの転送量は、合計で $N_\mathrm{d}(\log_2 N_\mathrm{p}+1)/N_\mathrm{p}$ となる。この場合の一プロセッサあたりの演算量と転送量との比は、

$$\frac{N_\mathrm{d}(\log_2 N_\mathrm{p}+1)/N_p}{(log_2 N_\mathrm{d})N_\mathrm{d}/N_\mathrm{p}} \sim \frac{\log_2 N_p}{\log_2 N_\mathrm{d}}$$

である。すなわち、全体問題規模 (N_d) が一定のまま、プロセッサ数を増やすと、非常にゆっくりと転送の占める割合が増加していく。逆に、一プロセッサあたりの問題のサイズ

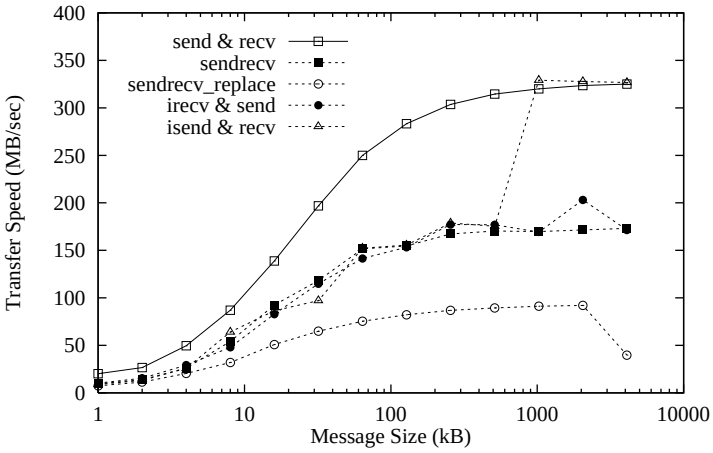

図 2: VPP500 の MPI の実効転送速度。横軸はメッセージ長、縦軸は実効転送速度を表わす。実行時環境変数 `VPP_MBX_SIZE` は、`33554432` (32MB) に設定した。

($N_{\rm d}/N_{\rm p}$) を固定したまま、プロセッサ数を増やす場合には、この割合は一定の値に留まることがわかる。

2.3 VPP500 の MPI 実効性能評価

厳密対角化法の並列化においては、行列のトランスポーズ転送が並列時間においてかなりの部分を占めている。この転送は、回数は少ないが一回の転送量は数十 MB〜数 GB である。そのため、並列実効性能には、ネットワークの立ち上がり性能よりむしろ、スループット性能が重要となる。そこで、我々はまず、VPP500 におけるネットワークスループット実効性能を簡単なベンチマークプログラムにより評価した。MPI を用いた場合、プロセッサ間でのベクトル (配列) の交換を行なう方法としては、

(1) `MPI_SENDRECV`
(2) `MPI_SENDRECV_REPLACE`
(3) `MPI_ISEND` (非ブロッキング送信) と `MPI_RECV` の組合せ
(4) `MPI_IRECV` (非ブロッキング受信) と `MPI_SEND` の組合せ

等が考えられる。VPP500 における、それぞれの場合の実効転送速度 (スループット) の測定値を図 2 に示す。図中には、`MPI_SEND` と `MPI_RECV` を用いたピンポン転送による片方向の実効転送速度の実測値も示している。片方向の転送では、メッセージ長が十分長い場合には、理論ピーク転送速度の 8 割程度が得られている。

まず気が付くのは、`MPI_SENDRECV` の転送速度が、片方向の転送の場合の半分程度であるという点である。すなわち、`MPI_SENDRECV` では、ハードウェアの持つ全二重性はほとんど生かされておらず、片方向づつ順番に転送を行なうのと同程度の性能しか出ない。さらに、`MPI_SENDRECV_REPLACE` では、ライブラリルーチン内で余分なメモリコピーが生じることによるオーバーヘッドを差し引いたとしても、非常に低い性能に留まっている。上記の 4 種類の転送方法の中で最も高速なのは、「(3) `MPI_ISEND` と `MPI_RECV` の組合せ」である (ただし、メッセージ長があまり長くない場合には、`MPI_SENDRECV` とほぼ同じ性能に留まる)。以下では、我々は、「(1) `MPI_SENDRECV`」と「(3) `MPI_ISEND` と `MPI_RECV`

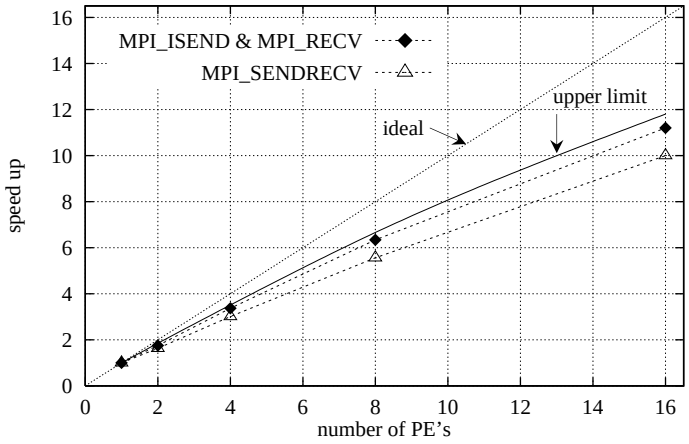

図 3: VPP500 における転送行列法の並列化効率。横軸は PE 数、縦軸は速度の向上の度合を示す。また、実線はピーク転送速度を用いて計算した理論上の上限。

の組合せ」の二通りの転送方法を用いてプログラムを作成し、性能評価を行なった。

2.4 並列性能評価

図 3 に、VPP500 における 16PE までの並列化効率の実測値を示す。「`MPI_ISEND` と `MPI_RECV` の組合せ」を用いたプログラムでは、ピーク転送性能から見積もった並列化効率と比較しても、十分な並列化効率が得られている。一方、`MPI_SENDRECV` を用いたプログラムでは、前者と比較して 1 割以上の性能低下が見られる。

3. 量子モンテカルロ法の並列化

3.1 量子モンテカルロ法

前節で述べた厳密対角化法では、必要となる計算時間、および主記憶容量が系のサイズに対して指数的に増加するため、扱うことのできる系のサイズは非常に限られる。前述の並列化手法を用いたとしても、$L = 34$ 程度が、現在のスーパーコンピュータでの限界である。さらに大きな系を扱う方法として、広く用いられている方法の一つに「モンテカルロ法」がある。この方法では、広大は位相空間の中から、代表的な少数の点をある「重み」に従って確率的にサンプリングすることにより、系の物理量を求める。

ここでは、量子反強磁性ハイゼンベルグ模型と呼ばれる量子スピン系を考える。前節で扱ったイジング模型はスピンのもつ量子性を無視した模型であり、スピンは上向き、下向きの 2 つの自由度しか取り得なかったが、量子スピン系では、不確定性原理に起因する量子ゆらぎ (あるいは演算子の非可換性と呼んでもよい) のため、スピンを「単なる方向をもった矢印」として扱うことはできない。現実の系においても、低温になればなるほどこの量子ゆらぎの効果が強くなり、シミュレーションにおいてもこの効果を正しく取り入れた模型を考える必要がある。

この量子的にゆらいだスピン状態を計算機の中で表現するために、一般には「経路積分 (path integral)」と呼ばれる手法が用いられる。すなわち、d 次元の量子系を $(d+1)$ の次

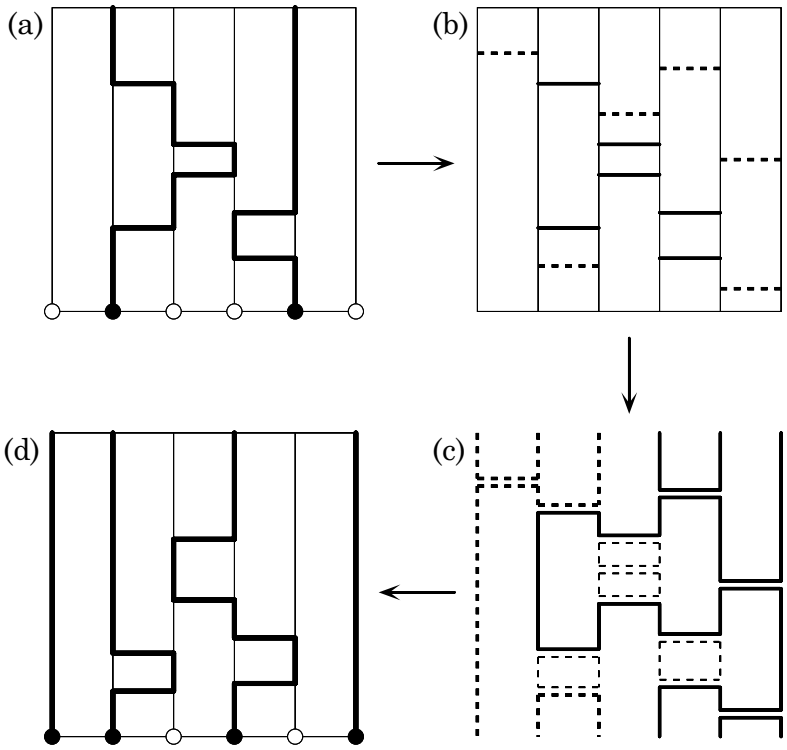

図 4: 量子モンテカルロ法 (ループアルゴリズム) における 1 モンテカルロステップ。水平方向が実空間方向、垂直方向が虚時間方向を表す。虚時間方向には、周期的境界条件が課されている。

元をもつ古典系へとマップする手法である。スピン状態はこの余分な 1 次元方向 (虚時間方向と呼ぶ) にゆらいだ「世界線 (world line)」として表わされる (図 4a)。この世界線の配置は、計算機の中では、ダブルリンクリストを用いて、時刻 0 での状態の情報に加え、世界線がジャンプした時刻と場所を覚えておくことにより表現される。

量子モンテカルロ法では、この世界線の配置を「重み」に従って確率的に変化させていくことにより位相空間のサンプリングを行なう。以前は、世界線を局所的に変形させて量子モンテカルロを行なっていたが、この方法では、非常に低温、もしくは相転移の臨界点近傍になると、状態の変化が非常に遅くなり、効率的なサンプリングが不可能になる。この問題は「臨界減速 (critical slowing down)」と呼ばれる。最近、この問題を回避する方法として「ループアルゴリズム」が提案された。この方法では、世界線の配置の更新は、局所的に行なわれるのではなく、グローバルなオブジェクト (ループと呼ばれる) を反転させることにより行なわれる。1 モンテカルロステップは、具体的には以下のような手順となる。

(1) ラベル付け (labeling)
世界線の配置 (図 4a) の中から、世界線が通っている部分と通っていない部分が隣りあっている領域を探し出し、そこにある「密度」で確率的に「リンク」を配置する。また、世界線がジャンプしている箇所には確率 1 で「リンク」を配置する (図 4b)。

(2) ループ認識 (loop identification)
上記のラベル付け操作により作成されたグラフの上をある点から辿っていく。途中で「リンク」に出会ったら、隣りにジャンプし、進む方向を反転する。この操作をルー

プが閉じる (出発点に戻る) まで続ける。次に、まだループの通っていない点から出発し、同じ操作を繰り返す。全ての点が必ずどれか一つのループに属するまで繰り返す (図 4c)。

(3) ループ反転 (loop flip)
得られたそれぞれのループ毎に、そのループ上のスピン状態をある確率 (一般的には 1/2) で反転する。これにより次のステップの世界線の配置が得られる (図 4d)。

この操作を何万回、もしくはそれ以上の回数繰すことにより物理量の測定を行なう。

3.2 ベクトル化

ループアルゴリズムでは、世界線の配置はダブルリンクリストを用いて表現され、このリストに対してリンクの挿入・削除や、リンクリストを辿るといった操作が主体となっており、残念ながらベクトル化は全く不可能である。実際、VPP500 では、Alpha チップなどの RISC CPU と比較すると数十分の一から数百分の一程度の性能しか出ない。ループアルゴリズムを用いた実際のシミュレーションには、Compaq Alpha ワークステーション、SGI Origin 2000、および Hitachi SR-2201 等のスカラ (並列) 機を使用している。

3.3 並列化

一般の局所的な状態更新に基づくモンテカルロ法は、$(d+1)$ 次元空間を各プロセッサに分割し、各モンテカルロステップで境界上のスピン状態を交換することにより、比較的自明に並列化可能である。一方、ループアルゴリズムでは、グローバルな操作が必要なので、並列化は非自明となる。我々は以下に挙げる方針に基づき、並列化を行なった。

- $(d+1)$ 次元空間を虚時間軸に関して各プロセッサに均等に分割し、世界線の情報を保持する。
- この場合、各プロセッサは「(1) ラベル付け」に必要な情報は全てローカルに保持しているので、このステップに関しては、各プロセッサが完全に独立に実行可能である。
- 「(2) ループ認識」については、以下の 2 つのサブステップにわけて実行する。
  ステップ (2-1) : まず、各プロセッサが自分の受け持ちの部分に関してループ認識を行なう。他のプロセッサとの境界にまたがるループ (閉じないループ) と各プロセッサの中で完全に閉じたループが構成される。
  ステップ (2-2) : 前サブステップで構成されたループのうち、「閉じていないループ」に関して、二分木を用いて徐々に大きなループに繋ぎ合わせる。
- 前ステップで得られた「グローバルなループ」に関する情報を、再び二分木で各プロセッサに配布した後、「(3) ループ反転」を各プロセッサ毎に独立に実行する。

$(d+1)$ 次元空間のうち、(d 次元の) 実空間方向は離散的な格子であるが、虚時間方向に関しては、連続変数となっている。したがって、プロセッサ数によらず均等な分割が可能となる。また、アルゴリズムが空間次元によらない、というメリットもある。

3.4 並列性能評価

上記の並列化アルゴリズムの理論的な並列化効率は以下のように見積もることができる。

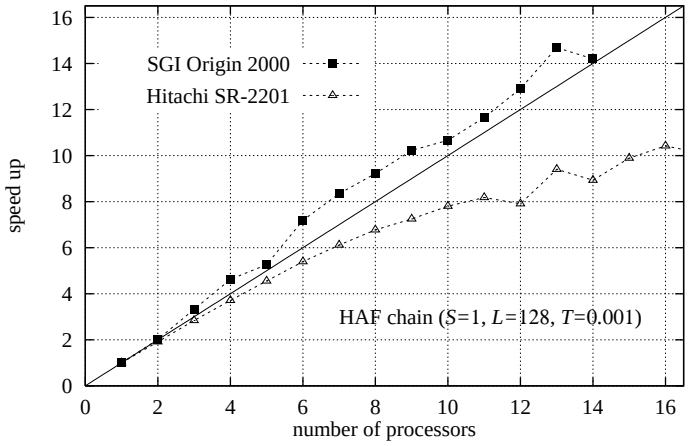

図 5: 量子モンテカルロ法 (ループアルゴリズム) の並列化効率。横軸はプロセッサ数、縦軸は速度の向上の度合を示す。

実空間方向の系のサイズを L、虚時間方向のサイズ (これは系の温度の逆数に比例する) を β とすると、逐次実行の場合の実行時間は、ステップ (1)、(2)、(3) 共に βL^d に比例する。$N_{\rm p}$ プロセッサで並列実行した場合、ステップ (1)、(2-1)、および (3) の所要時間は単純に $1/N_{\rm p}$ 倍となる。一方、ステップ (2-2) については、$O(L^d \log N_{\rm p})$ の時間が必要であるので、並列化によるスピードアップは、c をある定数とすると、

$$\frac{\beta L^d}{\beta L^d/N_{\rm p} + cL^d \log N_{\rm p}} \sim N_{\rm p}\left\{1 - c\frac{N_{\rm p}\log N_{\rm p}}{\beta}\right\}$$

すなわち、$N_{\rm p}\log N_{\rm p}$ に比べて β が十分に大きければ (温度が十分に低ければ)、高い並列化効率が期待される。

図 5 に、SGI Origin 2000 および Hitachi SR-2201 での実測例を示す。特に SGI Origin 2000 では、非常に高い並列化効率が得られている。実際のシミュレーションでは、この並列化したアルゴリズムを用いて、256 プロセッサまでの並列計算を行なっている。

4. おわりに (MPI、VPP Fortran、OpenMP $\cdots$)

今回は、「転送行列の厳密対角化」と「量子モンテカルロ法 (ループアルゴリズム)」の二種類のプログラムについての並列化手法、および並列性能評価結果を報告した。いずれのプログラムについても、十分な並列化性能が得られている。

	MPI	VPP Fortran	OpenMP
DATA の分割	explicit	explicit	implicit
段階的並列化	×	○	◎
ポータビリティ	◎	×	○
参入障壁	大	中	小
性能	◎	◎	?

表 1: 並列プログラミング環境の比較

今回は、いずれのプログラムも MPI を用いて並列化を行なった。他の並列プログラミング環境としては、最近広く使われるようになってきた OpenMP、また、VPP 独自の VPP Fotran などがあるが、筆者の独断と偏見に基づくそれぞれの並列プログラミング環境についての利点・欠点を表 1 に示す。共有メモリ機でしか使えないという欠点を除けば、これらの 3 つの中では、OpenMP が最も「取りくみやすい」並列プログラミング環境であると言える。

しかし、残念ながら、OpenMP を使って高い並列性能を引き出すには、かなりの熟練を要する。SGI Origin 2000 に代表される「cc-NUMA」のような共有分散アーキテクチャに基づく並列機では、データが物理的にメモリ上のどこに配置されているかによって、アクセス速度にかなり違いが生じる。また、フォルスシェアリング等のオーバーヘッドにより極端に性能が低下する場合もある。結局のところ、高い並列性能を引き出すためには、ユーザが物理的なメモリ配置を意識したプログラミングをする必要が生じる。最も大きな問題は、データ分割が言語上は implicit であるため、ユーザから非常に見えにくいという点にある。

一方、MPI は、共有/分散メモリを問わず、非常に高いポータビリティを持っている。(実際、LINUX が動作するシングル CPU のノートブック上で、MPI の並列プログラムを開発・デバッグし、その後、超並列計算機上でソース変更なしに実行することも可能である。) しかも、データ分割の設計が最初にしっかりとなされるので、完成したプログラムはかなり高い並列性能を示す場合が多い。我々の経験では、一般に、OpenMP で高い並列性能を示すプログラムは、MPI を用いて並列化を行なった場合にも、同じく高い並列性能を示す。しかし、逆はかならずしも成りたたない。(今回報告した「転送行列法」は MPI、OpenMP、いずれの並列化でも同程度の並列性能を示す。しかし、「量子モンテカルロ法」については、OpenMP による並列化は、ベクトル化と同じくほとんど不可能であり、MPI が唯一の選択肢である。) しかしながら、MPI では、OpenMP を用いた場合のような「段階的並列化」はほとんど不可能なので、初心者にとっては非常に障壁が高い。

その意味で、今必要とされているのは、「共有メモリ機でも動作する VPP Fortran」ではないだろうか？ すなわち、並列プログラムの作成において、手続きをまず分割しておいて、より高い性能を求める場合には、あとから「明示的に」データの分割を行なう (もちろん逆の順番でも可) ことができるような言語が開発されれば、初心者にとっても、エキスパートにとっても、非常に有効な並列プログラム開発環境となると期待される。