サブタスク間の依存関係に基づくスケジューリング機構を備えた並列プログラミング環境の開発

関嶋政和[†] 高崎慎也[††] 中村周吾[†]
池口満徳[†] 清水謙多郎[†,††]

本論文では，大規模並列・分散システムにおいて，基盤となるハードウェア環境およびアプリケーションに柔軟に適応できる資源管理機能を備えた並列プログラミング環境 Parsley の設計と性能評価について述べる．Parsley は，並列処理可能なサブタスクを単位として，サブタスク間の依存関係に関する情報をもとに，実行時間を短縮するスケジューリングを行う．大規模並列計算機 HITACHI SR2201 上で，分子動力学シミュレーションを Parsley を用いて実行したところ，従来の分子動力学シミュレーションの並列化されたプログラムに対して，プロセッサ数 32 台以上で高い性能を示し，プロセッサ数 175 台で最大 3.80 倍の高速化を達成した．また，Parsley は実行して得られたサブタスクの実行時間などの情報を記録して以降の実行に適用し，スケジューリング方針の自動的な改善を行う．その効果を調べた予備的実験では，1.07~1.16 倍の性能向上が得られた．本論文ではさらに，依存関係に基づくスケジューリングで得られる性能向上について，個々の操作に要する時間を検討するなどして，詳細な性能の解析を行った．

Parsley: A Scalable Framework for Dependence-driven Subtask Scheduling in Distributed-memory Multiprocessor Systems

Masakazu Sekijima,[†] Shinya Takasaki,[††] Shugo Nakamura,[†]
Mitsunori Ikeguchi[†] and Kentaro Shimizu[†,††]

This paper describes the design and implementation of a new parallel programming environment called Parsley, which provides fine-grained scheduling services based on the applications' program structure. In Parsley, application programs are divided into subtasks which may run serially or in parallel. Parsley provides a programming interface that allows a user to define subtasks and to specify the precedence constraints among them. According to this specification, the Parsley system schedules subtasks and allocates processors. Thus, the subtasks are executed in a dependence-driven manner. We developed a parallel molecular dynamics simulation program based on the Parsley mechanism and executed it on scalable multiprocessor systems. We achieved good scalability and showed that our system is efficient for large-scale molecular dynamics simulation.

1. はじめに

MPI[1] と PVM[2] は，分散メモリに基づく標準的な並列プログラミングインタフェースとして，分子シミュレーション，量子計算，流体計算など，大規模並列計算を必要とする様々な科学技術計算の応用で利用されている．これらのシステムは，プロセス間通信（メッセージ受渡し）に関連した豊富な機能を提供し，特に MPI-2[3] の仕様では，動的なプロセス，一方向通信，並列入出力などの機能が追加されている．しかしながら，これらのシステムはそれ自身，資源管理機能を備えておらず，負荷分散や通信コストの低減などを考慮したプログラミングをプログラマの責任で行わなければならない．こうしたプログラミングは，しばしば基盤となるハードウェア環境に依存し，例えば，別なハードウェア環境にプログラムを移植した場合，その環境に合ったプログラミングを再度検討しなければならなくなる．しかも，現実の科学技術計算では，問題の性格上，静的な負荷分散が不可能であったり，その記述が困難であったりすることも少なくない．このため，基盤となる並列プログラミング環境のレベル

† 東京大学 大学院農学生命科学研究科応用生命工学専攻
Department of Biotechnology, The University of Tokyo
†† 東京大学 大学院理学系研究科情報科学専攻
Department of Information Science, The University of Tokyo

で資源管理を行い，負荷分散や通信コストの低減を，ハードウェア環境に適応した形で，動的に行うことが望まれる．

従来の，並列プログラミング環境における資源管理の研究の多くは，分散システムをターゲットとした，ジョブまたはプロセスを対象とする負荷分散に関するものがほとんどであった．このようなアプローチは，特定のプログラムの高速化という目的では，通信コストの大きさ，並列処理の粒度の点において，実際に有効となる応用は限定される．計算処理の高速化のためには，分散システムだけでなく，現実の大規模並列計算機上で特定のプログラムの並列処理を効率化することが求められ，そのためには，アプリケーションプログラムの構造に応じたきめの細かい資源の割当て方針が必要となる．

本研究では，アプリケーションプログラムを並列処理可能な部分問題に分割し，それらの依存関係をもとに，ハードウェア環境に適応した，きめの細かい資源割当てを行う，並列プログラミング環境 Parsley の設計とその性能評価について述べる．

2. 基本設計

Parsley では，アプリケーションプログラムは並列処理可能な部分問題（サブタスクと呼ぶ）に分割され，それらを単位としてプロセッサが割り当てられ実行される．サブタスク間には，実行の先行制約が依存関係として定義され，それとともにサブタスクグラフが形成される．システムはそのサブタスクの内容に従ってプロセッサの割り当て（スケジューリング）を行う．サブタスク T_i がサブタスク T_j に先行するという先行制約は，T_j の実行を開始するには T_i の実行が終了していなければならないことを意味する．あるサブタスクは，それに先行制約を持つすべてのサブタスクが実行を終了したとき，初めて実行可能となる．

図 1 に示すように，Parsley では，マスタ/スレーブ方式に基づき，マスタがサブタスクのプールを管理しスケジューリングを行い，スレーブがマスタからの依頼を受けてサブタスクを実行する．並列処理の形態としては，現実によく用いられている SPMD（single program multiple data）を想定している．SPMD 形態におけるスレーブプログラムの構造は，MPI を用いた記述では，**図 2** のようになる．

スレーブは，マスタから送られてくる実行要求を `MPI_Recv()` を用いて受け取り，要求された処理を実行した後，終了通知を `MPI_Send()` を用いてマスタに送り返す．その繰り返しがスレーブの基本的な動作である．サブタスクとは，スレーブがマスタから処理要求を受け取り，マスタに終了通知を返すまでの部分に相当する．オペレーティングシステムが実現するプロセス（タスク）を意味するものではない．図 2 のような MPI の記述を解析して，サブタスクを自動的に切り出すことも可能であるが，サブタスク間のデータの受渡しを通信によって明示的に行わなければならないなどの制約が生じることから，ユーザにある程度の負担をかけることは避けられない．我々は現在，MPI による記述をもとにしたサブタスクの切出しと，それらの間の依存関係の解析を自動化する作業を進めているが，本研究では，設計の 1 つのステップとして，ユーザがサブタスクを明示的に指定するようにした．その上で，サブタスク間の依存関係に基づくスケジューリングの機構やハードウェア環境に適応したスケジューリング方針の自動的な改善など，並列プログラミング環境の重要な機能を開発することを目指す．

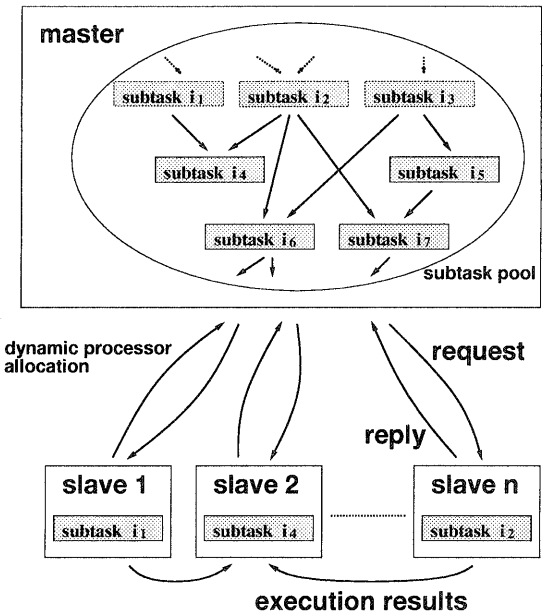


図 1 マスタ／スレーブモデルに基づくサブタスクの実行形態
Fig. 1 Subtask execution based on master-slave model.

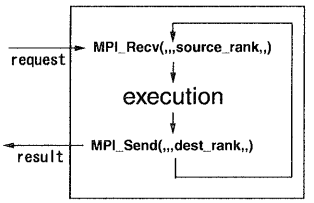


図 2 MPI によるスレーブの記述
Fig. 2 Slave-program structure in MPI.

```
        ⋮
i1 = Create_Subtask();
i2 = Create_Subtask();
i4 = Create_Subtask();
        ⋮
Set_Subtask_Dependency(i1,i4);
Set_Subtask_Dependency(i2,i4);
        ⋮
```

図 3 サブタスクの定義と依存関係の指定
Fig. 3 Subtasks creation and dependency specification.

図 3 は，サブタスクの定義とそれらの間の依存関係の指定例を示したものである．ユーザは，`Create_Subtask()` により，サブタスクを定義する．これにより，システムがサブタスクを一意に識別するための ID を割り当てる．この ID を `Set_Subtask_Dependency()` で指定することにより，サブタスクで処理する内容とサブタスク間の依存関係を指定する．各サブタスクは，自身の先行サブタスクのリスト PL（predecessors list）と後続サブタスクのリスト SL（successors list）を保持する．システムは，上記のユーザによる指定をもとに，各サブタスクの PL と SL に ID を追加していく．図 3 の例では，個々に依存関係が指定されているが，実際の応用では，依存関係に繰返し構造やその他の規則性があり，繰返し文などを用いて簡潔に定義できる場合が多いと考えている．

図 4 は，Parsley を用いてサブタスクを記述した例を示したものである．図 2 と比較して分かるようにサブタスクの記述そのものは，通常の MPI を用いた記述にほぼ対応している．ただし，実行要求の受信には `Parsley_Recv()`，送信には `Parsley_Send()` を用いており，これらの操作では，通信相手として，MPI のプロセス（ひいてはそのプロセスを実行しているプロセッサ）ではなく，Parsley のサブタスクの ID が指定される．サブタスクのプロセッサへの割当ては動的に行われるため，サブタスク ID から実際にサブタスクを実行する MPI プロセスのランクへの変換，およびこれらの操作のプロセス間通信操作への変換は Parsley システムにより，すべて実行時に行われる．

以上のように，Parsley では，ユーザが Parsley 独自のサブタスク間の通信プリミティブ（図 4 参照）を指定し，サブタスクとして実行するコード部分を明示するとともに，マスタではサブタスクの生成とそこで得られたサブタスク ID を用いたサブタスク間の依存関係を指定する必要がある．こうしたユーザが推定した情報をもとに，システムは，以下に述べるようなサブタスクのスケジューリングを行っている．

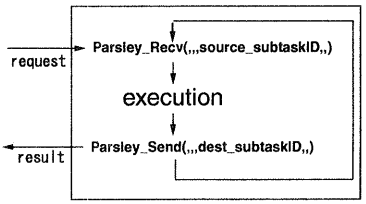


図 4 Parsley によるスレーブの記述
Fig. 4 Slave-program structure in Parsley.

3. サブタスクのスケジューリング

3.1 実行結果の受渡しの高速化

通常のマスタ/スレーブモデルでは，スレーブがサブタスクの実行結果を一度マスタに返し，その後マスタは後続のサブタスクにそれを配信するということになる．Parsley では，マスタの負荷を軽減するという観点から，実行結果の受渡しについては，マスタに返すのではなく，スレーブ間でサブタスクが直接受け渡すことにより，プロセッサ間の通信量の低減を図っている．この場合，スレーブはサブタスクの実行が終了すると，終了したことを伝えるメッセージだけをマスタに返し，実行結果を一時的に保存するとともに，別なサブタスクの実行を開始する．他のプロセッサで後続サブタスクの実行が開始されたことが通知されると，オンデマンドで後続サブタスクに直接実行結果を送信する．

後続サブタスクへの実行結果の送信要求の通知については，(1) マスタが後続サブタスクにプロセッサを割り当てるときに通知する方式と，(2) スレーブ自身が後続サブタスクを実行する前に通知する方式とが考えられるが，現状では，マスタへの負荷の集中を避けるため，(2) の方式を用いている．Parsley では，プロセッサをサブタスクに割り当てる際に，サブタスクの実行要求と一緒に，サブタスクの PL とそれらに割り当てられるプロセッサとの対応表を送っているため，それを受け取ったスレーブは，与えられたサブタスクの実行開始に先立ち，必要とする実行結果の転送を，先行するサブタスクを実行したプロセッサに直接要求することができる．スレーブは，すべての後続サブタスクに実行結果を送信するまで実行結果を保持する．

3.2 サブタスクの連結による通信の省略

Parsley では，後続のサブタスクが現在のサブタスクと同じプロセッサに割り当てられたとき，実行結果の受渡しをプロセッサ間の通信ではなく，同一プロセッ

サ内のメモリコピーによって行う機構を実現している．この機構を積極的に用いて通信量の軽減を図るアフィニティスケジューリングに関して検討を進めている．また現在，Parsleyでは，いくつかの連続するサブタスクをまとめて1つのサブタスクとし，プロセッサに割り当てる機構を実現している．サブタスクの粒度は，一般に性能に大きな影響を及ぼす重要な問題である．プログラミングの容易さの観点からすると，サブタスクはアプリケーションプログラムの構造に従って素直に定義できることが望ましい．一方，性能の観点からは，適切なサブタスクの粒度が望まれる．例えば，粒度が大きすぎると，十分な負荷分散が行われず，小さすぎると，プロセッサ割当てのオーバヘッドが増大する．そこで，現在のParsleyでは，サブタスクの粒度が小さい場合，実行時にそれらをまとめて1つのサブタスクとして実行する機構を実現している．現状では，空いたプロセッサに，実行可能なサブタスク群の中から，ユーザが指定した最大個数までのサブタスクをまとめてそのプロセッサに送って実行を要求するというアドホックな方式を用いている．今後，より一般的なサブタスク粒度の調整機構について検討していく予定である．

3.3 スケジューリング方針の自動的な改善

Parsleyは，動的な資源管理に基づくプロセッサ割当ての機能に特徴があるが，サブタスクの実行時間および依存関係があらかじめ分かっている場合は，動的な割当てに伴う種々のオーバヘッドを低減するため，これらの情報を生かしたスケジューリング方針により，静的なプロセッサの割当てを行う．現状では，負荷分散と通信時間の最小化の両方を考慮したスケジューリング方針として，CP/MISF法[4]において，サブタスク実行時間に通信時間の見積りを加えてスケジューリングを行う手法を用いている．この場合，スレーブ間のデータ転送は，通信相手のプロセッサをマスタに問い合わせることなく，直接実行される．**図5**は，動的なプロセッサ割当てを行ったときと静的なプロセッサ割当てを行ったときの，サブタスクの実行および実行結果の受渡しのタイミングを比較したものである．例えば，この図で，スレーブが互いに実行結果を要求しても（破線矢印），すでに新しいサブタスクの実行が始まっているためにすぐに実行結果が得られず，後続サブタスクの実行が開始できないことが起こり得る（横線の部分）．静的なプロセッサ割当てを実施しておけば，実行結果を次に割り当てるプロセッサに送っておくことができるため，このような待ちが発生しない．

しかしながら，現実には，サブタスクの実行時間およびサブタスク間の実行結果の受渡しのための通信時間をあらかじめ予測することは容易ではない．そこで，現在のところ，一度実行して得られたサブタスクグラフを2回目以降の実行に適用することでスケジューリング方針の改善を試みている．最初は，サブタスクの実行時間は不明であるので，実行可能なサブタスクに順次，利用可能なプロセッサを割り当てて実行する．このとき，サブタスクの実行時間および通信時間を計測する．次の繰返しに入る際に，計測した時間をもとに，スケジューリングを行う．

将来的には，同じ操作を繰り返し実行する性質をもったプログラムにおいて，サブタスクグラフの繰返し構造を利用し，サブタスク実行時間および通信時間の計測結果を用いて，実行時にスケジューリング方針を自動的に改善をすることを考えている．ただし，サブタスクグラフの繰返し構造にそのまま対応させて行うと，オーバヘッドが増大するので，その周期は，再スケジューリングによる効率向上と，オーバヘッドの増大による効率低下のトレードオフで決定される．また，スケジューリングを行う際には，サブタスク間で同期をとることは避けなければならない．サブタスクグラフ上，繰返し構造に合わせて，スケジューリングのポイントを設け，進度の異なるサブタスクに対しては，そのポイントからの距離に応じて優先度を与えることにする．

4. 分子動力学シミュレーションの並列計算

分子動力学法（molecular dynamics, MD）とは，多原子系における原子の運動を，原子間の相互作用を計算しながら，個々の原子に対するNewtonの運動方程式を積分することにより求める方法である．このような積分は解析的に計算することが不可能であるため，有限差分法を用いて数値的に計算する．MDの基本手順は，有限差分法の各タイムステップにおいて，力の計算と原子座標の更新を繰り返すというものである．このような計算を生体分子のシミュレーションに適用するためには，大規模で複雑な計算が必要であり，その膨大な計算量から，並列処理による高速化が不可欠となってくる．現在，MDは大規模並列計算を必要とする代表的なアプリケーションの1つとなっている．

MDの並列計算アルゴリズムは，原子分割法（atom decomposition, AD）と空間分割法（spatial decomposition, SD）の2つに大別される．原子分割法は，プロセッサ数 P 原子数 N のとき，各プロセッサが N/P の原子についてそれぞれ力の計算と座標の更新を行い，すべてのプロセッサの間で，更新後の座標

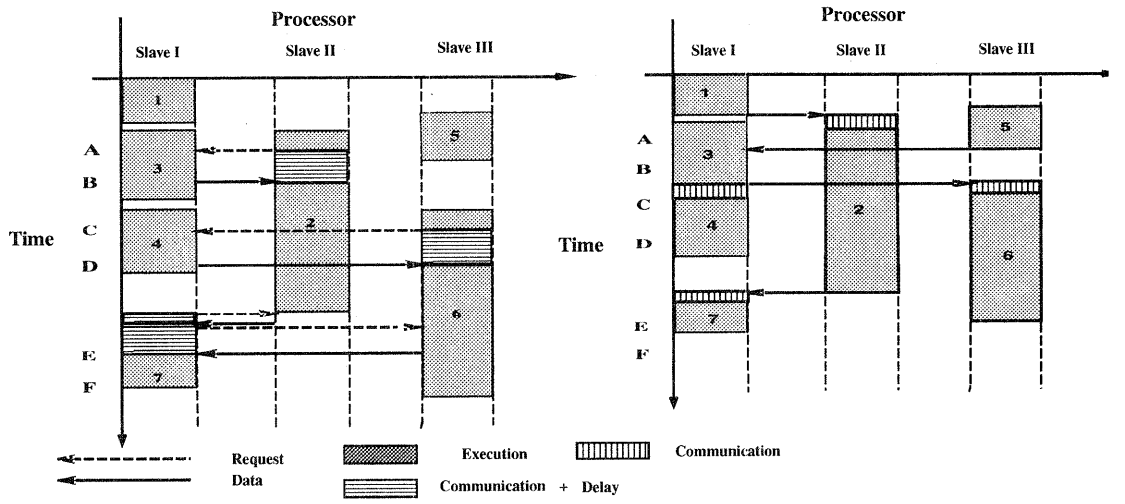


図 5 動的なスレーブ間通信（左）と静的なスレーブ間通信（右）
Fig. 5 Timing chart for processor allocation (left: Dynamic, right: Static).

データを交換するというアルゴリズムである[5]．一方，空間分割法は，計算対象となる系をセルに分割し，それらのセルの 1 つずつを各プロセッサに割り当てる方法である．プロセッサは担当セル内の原子について力の計算と座標の更新を行い，この計算結果を，それを必要とするセルを担当する限られたプロセッサに送る[6)~8)]．原子分割法が負荷分散を目的としているのに対し，空間分割法は主に通信時間の短縮を目的としている．空間分割法の方が原子分割法よりも通信コストがかからないため，一般に効率的であるが[9]，空間を分割したセル間で原子密度が異なることにより負荷にばらつきが生じるという問題がある（これは水と蛋白質の混在するような系で容易に起こり得る）．そこで我々は，空間分割法を基にして通信コストの低減を図りながら，Parsley を用いて負荷分散を効率的に行う MD の並列プログラムを開発した．Parsley を用いたプログラミングでは，従来のようにタイムステップごとに同期をとらず，各サブタスクの入力データ（座標，速度）が利用可能になると同時にその実行を開始する，依存関係に基づくデータフロー的な計算が実現される．このような依存関係に駆動された計算により，本来負荷にばらつきが生じる可能性のある空間分割法の計算を，プロセッサ時間を無駄にすることなく，効率的に行うことが可能となる．

図 6 は，空間分割法による MD のサブタスク間の依存関係の一部を示したものである．各タイムステップの最初の段のサブタスク群が力の計算（force），次の段のサブタスク群が座標の更新（update）を行う．この図の例では限られたサブタスクしか記載していないが，実際には，各セルの周辺のセル 27 個に実行結果を送信することになるので，例えば，各 force サブタスクの後続サブタスクの数は，系のエネルギーを計算するサブタスクと後続の update の合計 28 ということになる．

5. 性能評価

5.1 シミュレーションの条件

我々は，分子動力学シミュレーションの対象として，Protein Data Bank（PDB)☆ に登録されている 2 種類のタンパク質 BPTI と Lysozyme の結晶構造データファイル 4PTI と 1HEL をそれぞれ用いた．生体環境でのシミュレーションを行うため，本測定では，BPTI の周囲 6 Å，Lysozyme の周囲 5Å を TIP3P モデルの水で覆い，BPTI + 水系（原子数 16735) と Lysozyme + 水系（原子数 29754) として実験を行った．また，遠距離力の cut-off の長さを 9 Å として，BPTI + 水系の空間を 5 × 5 × 5 分割，Lysozyme + 水系を 5 × 7 × 5 分割した．

5.2 Parsley 上の MD と空間分割法の比較

HITACHI SR2201 上で行った BPTI + 水系における Parsley 上の MD（Parsley MD) と従来の空間分割法（SD MD) の 10 ステップのシミュレーション実行時間を**図 7** に示す．SD MD において，プロセッサ数が 1 台のときの実行時間は 68.4 秒であり，プロセッサ数 P のとき，その実行時間の $1/P$ を示したのが，図中の Ideal である．それぞれの手法で最も短時間で

☆ http://www.rcsb.org/pdb/

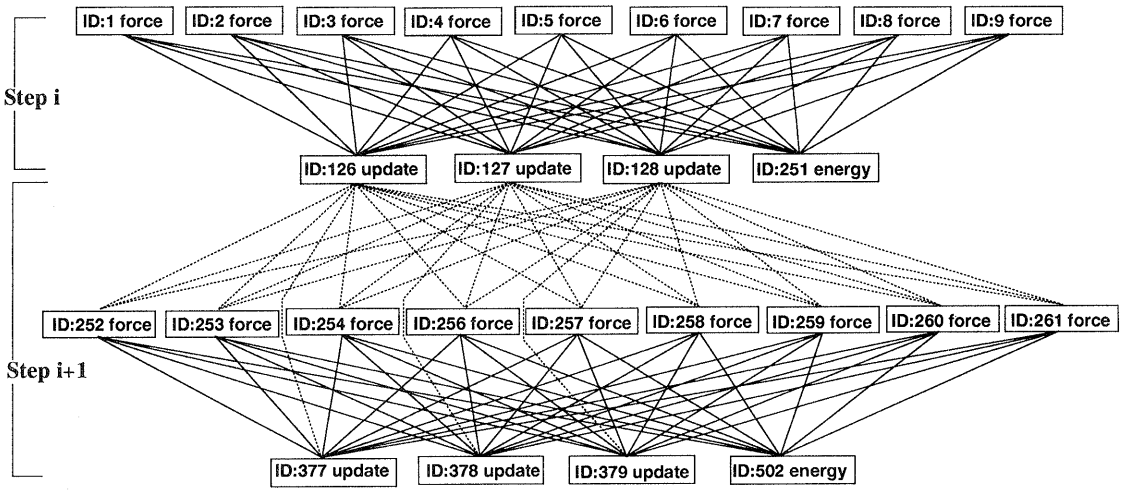


図 6 分子動力学におけるサブタスクの依存関係
Fig. 6 Subtask dependencies in molecular dynamics simulation.

実行を終了したときのプロセッサ数とそのときの計算時間は，それぞれ 125 台で Parsley MD が 2.28 秒，SD MD が 7.97 秒であり，プロセッサ数が 1 台のときの SD MD による実行時間のそれぞれ 30.0 倍，8.58 倍となった．プロセッサ数が 125 台のとき，Parsley MD が SD MD よりも 3.49 倍，実行時間を短縮している．

同様に Lysozyme + 水系に対して行った Parsley MD と SD MD の 10 ステップのシミュレーション実行時間を**図 8** に示す．SD MD において，プロセッサ数が 1 台のときの実行時間は 136 秒であり，プロセッサ数 P のとき，その実行時間の $1/P$ を示したのが，図中の Ideal である．それぞれの手法で最も短時間で実行を終了したときのプロセッサ数とそのときの計算時間は，それぞれ 175 台で Parsley MD が 3.36 秒，SD MD が 12.8 秒であり，プロセッサ数が 1 台のときの SD MD による実行時間のそれぞれ 40.5 倍，10.7 倍となる．プロセッサ数が 175 台のとき，Parsley MD が SD MD よりも 3.80 倍，実行時間を短縮している．

図 7，図 8 から分かるように，Parsley MD が空間分割法に対して有利な結果を得るようになるのは，両系ともにプロセッサ数が 32 以上の場合である．両系とも，プロセッサ数が少ない場合においては，例えばプロセッサ数 2 台のとき，Parsley MD の実行時間が BPTI + 水系で Parsley MD：171 秒，SD MD：34.8 秒，Lysozyme + 水系で Parsley MD：530 秒，SD MD：68.4 秒というように，Parsley MD が SD MD と比べて大幅に実行時間を要している．両系において同様の傾向があるので，ここでは BPTI + 水系のデータのみ詳細に解析を行う．

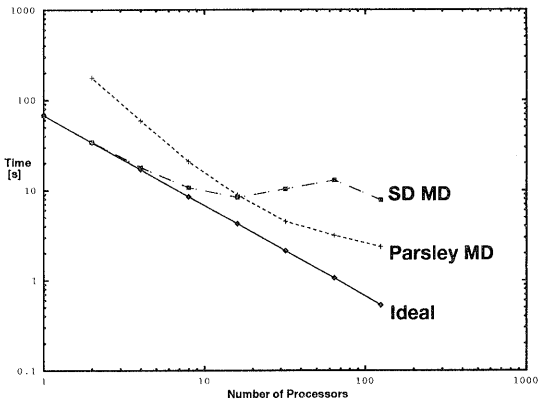


図 7 BPTI+水系における Parsley MD と空間分割法の並列化効率の比較
Fig. 7 Performance of Parsley MD and conventional parallel based on spatial decomposition (BPTI + water).

図 7 においてプロセッサ台数が少ないとき，Parsley MD が SD MD に比べて大幅に実行時間が大きい原因を探るため，Parsley MD，SD MD ともに全実行時間から通信時間などを差し引き，力の計算や座標の更新の計算時間のみを抽出（これを以後計算時間と呼ぶ）すると，プロセッサ数 2 台のとき，Parsley MD：70.0 秒となる．SD MD では並列化効果により計算時間がプロセッサ数が 2 台で 1 台の約 1/2 の高速化を達成しているのに対し，Parsley MD では使用プロセッサ数が 2 台の場合，スレーブが 1 台ということになり並列化の効果が得られず，これにより生じるオーバヘッドのみがそこに加わることになる．実際，このときの

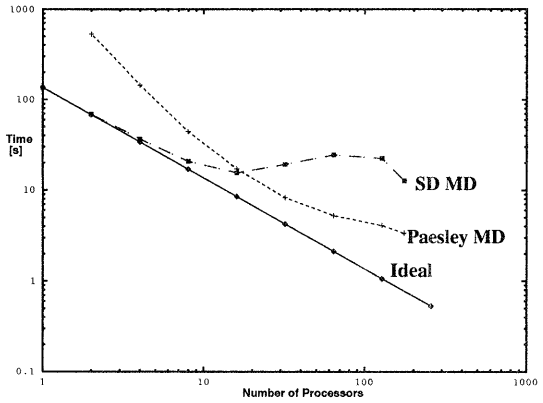


図 8 Lysozyme+水系における Parsley MD と空間分割法の並列化効率の比較
Fig. 8 Performance of Parsley MD and conventional parallel based on spatial decomposition (Lysozyme + water).

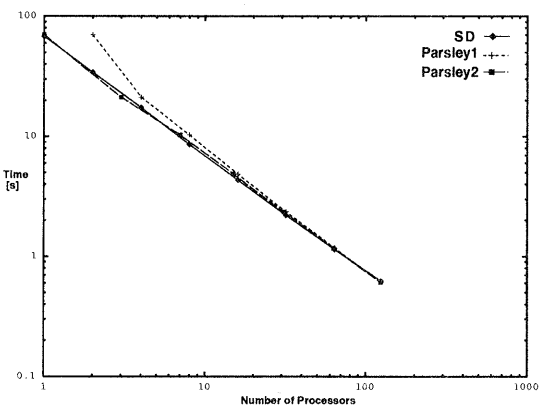


図 9 BPTI+水系における Parsley MD と空間分割法の計算時間の比較
Fig. 9 Computation times of Parsley MD and SD MD.

計算時間は SD MD におけるプロセッサ数 2 台のときの値よりも 1 プロセッサのときの値に近いものになっている．

次に，どれほど多くのプロセッサを用いればマスタ 1 台分のプロセッサを使用していることの影響を無視し得るのか，もしくは無視し得ないのかを調べる．**図 9** において SD は SD MD における計算時間を全プロセッサ数に対してプロットしたものであり，Parsley1 は Parsley MD における計算時間を全プロセッサ数に対してプロットしたものであり，Parsley2 は Parsley MD の計算時間をスレーブのプロセッサ数に対してプロットしたものである．Parsley1 と Parsley2 の線が重なるほど，マスタのためのプロセッサを余分に確保していることの影響から免れることになるが，図 9 の

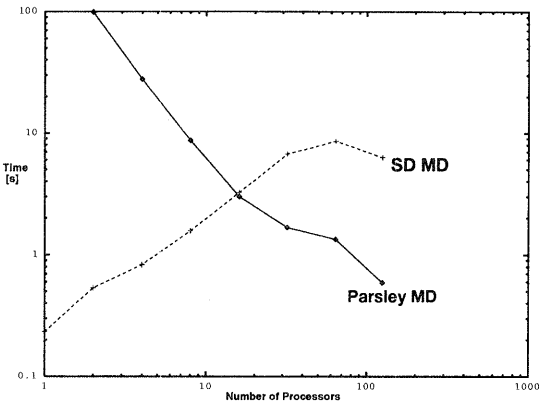


図 10 BPTI+水系における Parsley MD と空間分割法の通信時間の比較
Fig. 10 Communication times of Parsley MD and SD MD.

結果から，プロセッサ数 32 台以上のとき，Parsley1 と Parsley2 はほぼ重なっていることが分かる．つまり，計算時間の観点からするとプロセッサ数 32 台以上のとき，Parsley MD は SD MD に対してマスタ／スレーブモデルを採ったことによる不利さを無視し得るようになる．

図 10 は BPTI + 水系における Parsley MD と空間分割法における通信時間を表している．図 10 から，プロセッサ数の増加に従い Parsley MD では通信時間が減少し，SD MD では逆に通信時間が増加していることが分かる．図 9 より計算時間は，Parsley MD と SD MD であまり変わらないので，通信時間が全体の実行時間を左右することが分かる．Parsley MD ではプロセッサ数が 2，4 台の場合，通信時間が計算時間よりも大きい値を示す．空間分割法ではプロセッサ数が 32，64，125 台の場合，通信時間が計算時間よりも大きい値を示す．

SD MD ではある格子を担当したプロセッサは周辺 26 の格子とデータのやりとりをする必要があるが，プロセッサ数が少ないときは周辺格子も自分自身が担当している場合が多くなるため，データの受渡しも実際の通信を行わずメモリコピーで済ませられることが多くなる．プロセッサ数が多くなると，データ受渡しを他のプロセッサと行う回数が増えるため，結果としての通信回数が増大する．ただし，これは，SD MD は `MPI_Bsend()` をすべて行ってから `MPI_Recv()` を行うという現在の実装に起因している可能性もあるので，現在検討をすすめている．Parsley MD では，1 回の送信（`Parsley_Send`)/受信（`Parsley_Recv`) を実行するのにオーバヘッドとして時間が 0.00169 秒かかる．

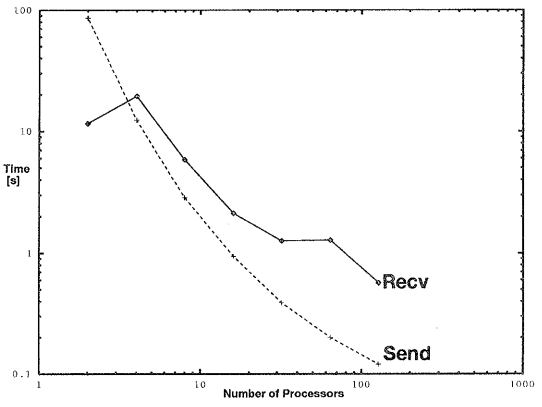


図 11 BPTI+水系における Parsley MD の通信時間の内訳
Fig. 11 Details of communication times in Parsley MD.

各プロセッサが送受信を実行する回数は，**図 11** のようにスレーブ P 台で $1/P$ となるのでプロセッサ数が増加すると極めて減少することになる（1 スレーブ当たりの通信回数はスレーブ 1 台のとき 58699 回，スレーブ 124 台のとき最大 700 回最小 268 回で平均 473 回）．Recv の方が Send と比較して時間の減少幅が小さいのは実際にデータ通信を行っているからである．この 1 回の送受信にかかる Parsley を使用することに起因するオーバヘッドは，スケジューリング等による実行時間縮小の方針とはまた別に実行時間を縮小するとしてこれからの改善すべき課題の 1 つであると考えている．

図 12，**図 13** はそれぞれ BPTI + 水，Lysozyme + 水の両系における 32 台と 125（Lysozyme では 175）台使用時の実行時間の内訳を示している．ここまで議論してきたように，プロセッサ数が多いときの Parsley MD と空間分割法による実行時間の違いが主に通信時間によるものであることが分かる．

図 14 は Parsley と SD MD それぞれの実際の実行の様子を示している．Parsley ではサブタスクの実行が非同期に行われている様子がうかがえる．エネルギー計算のサブタスク（energy）は，図 6 に示しているように後続サブタスクがない．ここではこのサブタスクに先行するサブタスクは直前の force だけである．このように，Parsley はすでに先行制約が満たされて実行可能なサブタスクをスレーブに割り当てる際に，現在担当しているサブタスクを最も早く終了したスレーブに実行させることが可能である．SD MD のサブタスクは，Parsley MD のサブタスクより長くなっているが，これは，64 プロセッサの段階で，SD MD ではすでに計算時間よりも通信時間の方が実行時間の中で大きな割合を占めており，ここに表れている実行時間の半分以上が通信時間となっているためである．

本実験は，マルチプログラミングではない環境で行ったため，各プロセッサが計算のみを行っている時間は Parsley MD，SD MD とも同じである．負荷分散の効果は同期待ち時間の割合を比較することに求めることができる．プロセッサ数 64 台の時，実行時間は，Parsley MD 3.08 秒，SD MD 12.6 秒である．通信時間は，それぞれ，1.34 秒，8.59 秒であり，同期待ち時間は，それぞれ，0.426 秒，2.87 秒である．つまり，今回の実験においては Parsley MD は SD MD に対して通信時間を 15.6%にし，同期待ち時間を 14.8%にしていることから通信時間の減少と負荷分散の効果が同程度働いて実行時間を短縮したと言える．

5.3 スケジューリング方針の自動改善の効果

スケジューリング方針の自動改善については，プログラム実行中にインクリメンタルにスケジューリング方針を改善するまでに至っておらず，現在のところ，アプリケーションプログラムを 1 回実行して，サブタスクの実行時間，通信時間を計測し，その結果得られたサブタスクグラフを保存して，次回以降の実行で，それを用いてスケジューリングを行う機能を実現している．これにより，2 回目以降の実行では，ハードウェア環境に応じた性能向上が達成されると期待される．

表 1 は，MD に上記の Parsley におけるスケジュール方針の改善方法を適用した結果を示している．

サブタスク実行時間の予測値として，すべて同一と仮定した場合（initial policy）と実際に計測した値を用いた場合（improved policy，サブタスクグラフの保存・回復機能を用いた場合）とを比較した．後者の方が良い性能を示しており，スケジューリング方針の改善に効果のあることが分かる．ただし，両者の性能の差はそれほど大きくない．これは，基本となるスケジューリング方針が，通信コストを十分に考慮できていないため，通信時間の占める割合が大きく，スケジューリング方針の改善の効果が十分に得られないためである．

6. 関連研究

分散システムにおける大域的スケジューリングの研究は，80 年代の分散オペレーティングシステムの研究で盛んに行われた．その後，スケジューリングの機能だけでなく，異種分散環境に対応した資源管理，柔軟なスケジューリング方針，MPI，PVM，HPF などとの相互運用など，現実的な機能を備えたジョブ管理システム [10)~12)] や資源管理システム [13)~15)] が多数現われた．また，ジョブレベルの管理ではなく，特定

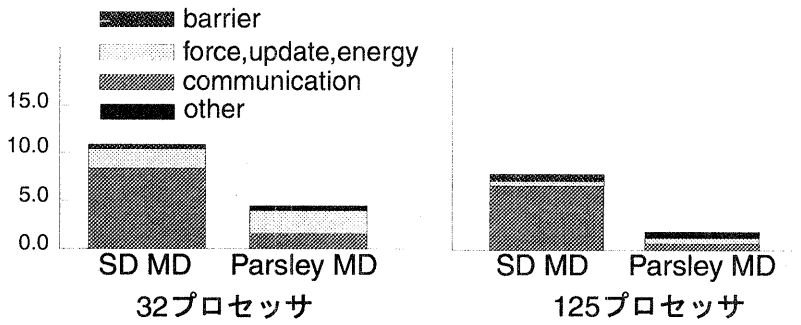


図 12 BPTI+水系における Parsley MD と空間分割法の比較：(左) 32 プロセッサ, (右) 125 プロセッサ
Fig. 12 Details of execution times (left:32 processors, right:125 processors) (BPTI + water).

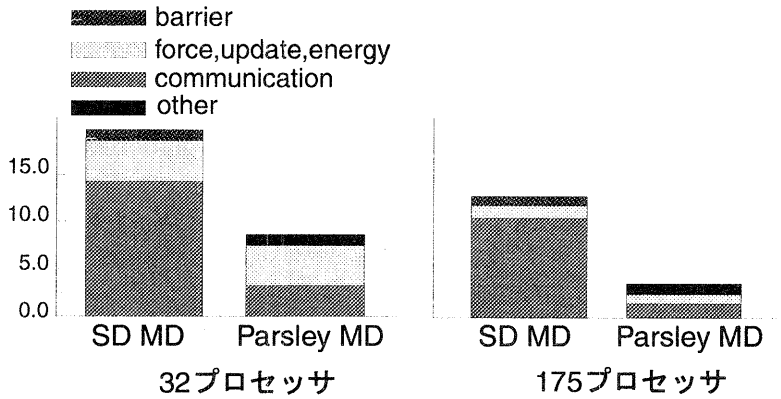


図 13 Lysozyme+水系における Parsley MD と空間分割法の比較：(左) 32 プロセッサ, (右) 175 プロセッサ
Fig. 13 Details of execution times (left:32 processors, right:175 processors) (Lysozyme + water).

表 1 スケジューリング方針の改善の効果
Table 1 Execution times (seconds) for various scheduling policies.

# processors	dynamic allocation	initial policy	improved policy
16	6.38	6.17	5.77
32	2.73	2.99	2.58

のプログラムの高速化を目的としたスケジューリング機能をもつシステムの研究も盛んに行われた[16),17)]. 例えば, DynamicPVM[16)] は, PVM 上にプロセス移送を実現したシステムで, Condor[15)] によるワークステーションクラスタのためのスケジューリング機構を用いている. MPVM[17)] も同様に PVM にプロセス移送の機能を実現したシステムで, 各ノードに分散したデーモンプロセスが収集したノードの負荷情報をもとに負荷分散を行う. これらのシステムは互いに独立なプロセスの生成・移送により, スケジューリングを実現したものである.

スケジューリングの効果は, アプリケーションの構造を考慮することによりさらに向上すると期待される. DOME[18)] は, 負荷分散とフォールトトレラントの機能を有する分散オブジェクトのライブラリで, SPMD 形態の C++言語で記述された PVM のプログラムに適用される. DOME では, 単純な負荷分散の方式に加えて, オブジェクト間の通信のパターンに適したプロセッサ割当てを試みるという手段も提供している. 負荷分散の方針は, プログラムをユーザが指定したステップ数だけ実行し, そこで計測されたプロセッサ負荷をもとに, 移送先を決定するという方式を用いている.

メタコンピューティング環境の AppLeS[19)] では,

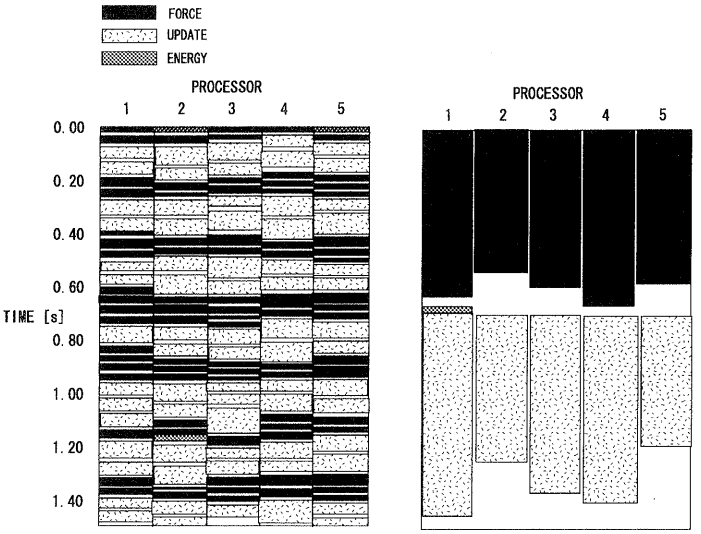


図 14 64 プロセッサにおける Parsley によるサブタスクによる並列化（左）と SD MD による並列化（右）の実行の様子
Fig. 14 Processor utilization of Parsley MD(left) and conventional Parallel MD(right).

通信し合うタスクに対して，プログラマが柔軟に負荷分散の方針を指定できる枠組みを提供している．Prophet[20] は Mentat システム上の SPMD 形態のパイプライン方式で並列実行するタスク群に対して，Ethernet，ATM の LAN の上で移動する独自の資源管理システムが収集した情報をもとに，実行時間を最小化する方針のスケジューリングを行っている．アプリケーションの構造に則したスケジューリングを，プロセス間の依存関係を用いることにより実現しようとするアプローチはごく自然なことであり，このようなアプローチを用いている並列プログラミング環境はいくつか存在する．VDCE[21] では，アプリケーションを構成する複数のタスクとそれらの依存関係をユーザが指定し，それに従って，システムがリストスケジューリングによってプロセッサを割り当てる．メタコンピューティング環境として，広域ネットワーク上の異種分散システムに対応しており，タスクの属性としてタスク間の通信のプロトコル，MPI，PVM などの通信のタイプ，マシンタイプなどを指定することできる．タスク間の依存関係は，Web ベースで，専用のエディタで指定できるようになっている．これらのシステムは，依存関係に基づいたスケジューリングを実現しているという点で本研究と共通しているが，タスク（プロセス）がスケジューリングの単位であり，また，スケジューリング方針そのものが単純であるという点で本研究と異なっている．

1 つのプログラムの実行に対して，より粒度の細かいスケジューリングを実現するため，特定のプログラムを並列処理コンパイラによるコードの解析やユーザの指定をもとに，ランタイムシステムが，システムの動的な状態の変化に応じて，スケジューリングを行うアプローチもある． 先駆的なシステムである Chores（共有メモリ型マルチプロセッサをターゲットとする）[22] や DUDE（Definition-Use Description Environment）[23] が例として挙げられる．DUDE は，`doall`，`doacross` などの明示的な並列処理の指定をもとに，コンパイラがデータおよび制御の依存関係を解析し，ランタイムシステムがその依存関係に基づいてスケジューリングを行う．入れ子のループ構造など，特定のプログラム構造に対してのみ有効である．

MARS（Metacomputer Adaptive Run-time System）[24] はタスク（プロセス）の移送機構をもとに負荷分散を行うが，我々のシステムと同じくプログラムの依存関係を用いて，よりきめの細かいスケジューリングを行っている．MARS では，システムがプログラムを解析して，send/receive ごとに基本ブロックを定義する．基本ブロックの単位でどのノードにタスクを移送するかは，各ノードのデーモンプロセスが収集した負荷情報をもとに判定する．システムがプログラムを解析して依存関係を求める部分は Parsley では未実装であるが，基本ブロックを単位にタスクを移送するには，基本ブロックをまたいだデータの受渡しはすべてプロセス間通信によらなければならないなどの制約が生じる点は，現在の Parsley と同じである．

MARSは分散システムのみをターゲットとしているため，SPMD形態を前提としたプロセス間通信によるサブタスクの効率的な移送は考えていない．また，スケジューリング方針についても，実行した結果に基づくスケジューリングを前提としている反面，静的スケジューリングおよびスケジューリング方針の自動改善の機能は実装されていない．さらに，MARSでは，本研究のように，依存関係に基づくスケジューリングが実際のアプリケーションでどのような効果があるか，詳細な解析は一切行っていない．なお，静的スケジューリング方針の研究としては，従来の共有メモリ型マルチプロセッサで用いられてきた静的スケジューリング方針を拡張し，分散システムをターゲットとした，通信コストを加味した依存関係のあるタスクのスケジューリング方針が数多く提案されている[25),26)]が，方針そのものの研究は，今後の研究課題である．

7. おわりに

並列プログラムの性能向上には，アプリケーションプログラムの構造に基づくスケジューリングが必要である．本論文では，並列処理可能なサブタスクを単位として，サブタスク間の依存関係に関する情報をもとに，スケジューリングを行う，新しい並列プログラミング環境Parsleyの設計と性能評価について述べた．Parsleyは現状ではユーザがサブタスクおよびそれらの依存関係を指定する方式を採っているが，その指定にあたっては，プロセッサ台数，プロセッサ処理速度，通信速度などのハードウェア環境，およびサブタスク間の負荷のバランスは一切考慮する必要がなく，システムがハードウェア環境および動的な資源の利用状況に応じて，実行時間を短縮するスケジューリングを行う．従って，開発したアプリケーションプログラムを他のハードウェア基盤にそのまま移植しても高い性能を得ることができる．特に，静的な負荷分散が容易でない計算処理において（例えば，分子動力学シミュレーションの例では，電荷の計算，構造のサンプリングなどの計算など，異なる種類の計算を一緒に行うなど），大きな威力を発揮するものと期待される．

Parsleyは，現在，HITACHI SR2201，Sun Ultra-Enterprise 10000，NEC Cenju-3，LANで結合したUNIXワークステーション上で稼動している．システム自身はCで記述され，MPIを用いて実装されているので，移植性は高い．今後は，利用可能な他の並列計算機に移植して性能評価を行い，スケジューリング方針の自動改善などの効果を解析したいと考えている．基本となるスケジューリング方針を改善することにより，その効果はさらに増大するものと期待される．そのほか今後の課題としては，MPIで記述されたアプリケーションプログラムからサブタスクの切り出しと依存関係の抽出を自動的に行えるようにする，サブタスクの一括した割当てを自動的に行う機構を実現することなどが挙げられる．また，現状ではスレーブの数は，HITACHI SR2201で174台以下であり，マスタへの負荷の集中は回避されているものの，今後は，スケーラビリティを考慮したマスタの多重化，それに伴うサブタスク群の分割割当て（依存関係を記述したサブタスクグラフのクラスタ分割）なども検討していきたい．

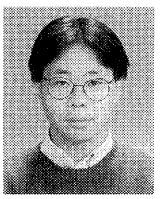
謝辞 本研究の一部は，文部省科学研究費補助金（特定領域研究 課題番号 09245105，基盤研究(c) 課題番号 10680336）および並列処理研究推進機構（PDC）の助成による．

参考文献

1) Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard*, 1.1 edition (1995).
2) Geist, A., Beguelm, A., Dongara, J., Jiang, W., Manchek, R. and Sunderam, V.(eds.): *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press (1994).
3) Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface* (1997).
4) Shirazi, B., Wang, M. and Pathak, G.: Analysis and Evaluation of Heuristic Methods for Static Task Scheduling, *J. Parallel and Distributed Comput.*, Vol. 10, pp. 222–223 (1990).
5) Smith, W.: Molecular dynamics on hypercube parallel computers, *Comp. Phys. Comm.*, Vol. 62, pp. 229–248 (1990).
6) Fincham, D.: Parallel computers and molecular simulation, *Molecular Simulation*, Vol. 1, pp. 1–45 (1987).
7) Heller, H., Grubmüller, H. and Schulten, K.: Molecular dynamics simulation on a parallel computer, *Molecular Simulation*, Vol. 5, pp. 133–165 (1990).
8) Ballestrero, P., Bagliletto, P. and Ruggiero, C.: Molecular Dynamics for Proteins: Performance Evaluation on Massively Parallel Computers Based on Mesh Networks Using a Space Decomposition Approach, *J. Comp. Chem*, Vol. 17, pp. 469–475 (1996).
9) Plimpton, S. and Hendrickson, B.: A New Parallel Method for Molecular Dynamics Simu-

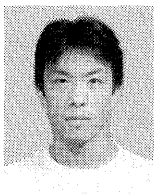
lation of Macromolecular Systems, *J. Comp. Chem*, Vol. 17, pp. 326–337 (1996).
10) *LSF User's Guide* (1996).
11) Supercomputer Computations Research Institute: *DQS 3.1.3 User's Guide*, Florida State University (1996).
12) Craysoft: *NQE User's Guide*, Cray Research Inc. (1997).
13) Neuman, B. C. and Rao, S.: The Prospero Resource Manager: A Scalable Framework for Processor Allocation in Distributed Systems, *Concurrency: Practice and Experience*, Vol. 6, pp. 339–355 (1994).
14) Zhou, S., Zheng, X., Wand, J. and Delisle, P.: Utopia: A load sharing facility for large, heterogeneous distributed computer systems, *Software - Practice and Experiences*, Vol. 23, pp. 1305–1336 (1993).
15) Litzkow, M., Livny, M. and Mutka, M.: Condor - a hunter of idle workstations, *Proc. 8th International Conference on Distributed Computing System*, pp. 104–111 (1988).
16) Dikken, L., van der Linden, F., Vesseur, J. and Sloot, P.: DynamicPVM – Dynamic load balancing on parallel systems, *Proceedings of HPCN-94*, pp. 273–277 (1994).
17) Casas, J., Konuru, R., Otto, S. W., Prouty, R. and Walpole, J.: Adaptive load migration systems in PVM, *Proceedings of Supercomputing '94*, pp. 390–399 (1994).
18) Arbenz, P., Beguelin, A., Lowekamp, B., Seligman, E., Starkey, M. and Stephan, P.: Dome: Parallel programming in heterogeneous multi-user environment, Technical Report CMU-CS-95-137, Carnegie Mellon University (1995).
19) Berman, F. and Wolski, R.: The AppLeS project: A status report, *Proc. NEC Symp. on Metacomputing* (1997).
20) Weissman, J. and Zhao, X.: Runtime support for scheduling parallel applications in heterogeneous NOWS, *Proc. Sixth IEEE Symp. on High Performance Distributed Computing*, pp. 345–355 (1997).
21) Topcuglu, H., Hariri, S., Furmanski, W., Valente, J., Ra, I., Kim, D., Kim, Y., Bing, X. and Ye, B.: The Software architecture of a virtual distributed computing environment, *Proc. High-Performance Distributed Computing Conf.*, pp. 40–49 (1997).
22) Eager, D. L. and Zahorjan, J.: Chores: Enhanced Run-Time Support for Shared-Memory Parallel Computing, *ACM Transactions on Computer Systems*, Vol. 11, No. 1, pp. 1–32 (1993).
23) Grunwald, D. and Vajracharya, S.: The DUDE run-time system: An object-oriented macro-dataflow approach to integrated task and object parallelism. `http://www.cs.colorado.edu/~suvas/paper/paper.html`.
24) Gehring, J. and Reinefeld, A.: MARS – a framework for minimizing the job execution time in metacomputing environment, *Future Generation Computer Systems*, Vol. 12, No. 1, pp. 87–99 (1996).
25) Becker, W.: Dynamic balancing complex workload in workstation networks – challenge, concepts and experience, *Proceedings of HPCN-95*, pp. 407–412 (1995).
26) El-Rewini, H. and Lewis, T.: *Distributed and Parallel Computing*, Prentice-Hall (1998).

(平成 11 年 7 月 16 日受付)
(平成 11 年 12 月 29 日採録)



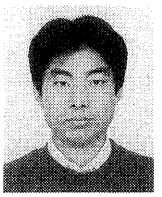
関嶋 政和（学生会員）

1973 年生．1999 年東京大学大学院農学生命科学研究科応用生命工学専攻修士課程修了．現在，同大学大学院農学生命科学研究科応用生命工学専攻博士課程在学中．科学計算，並列・分散処理に関する研究に従事．



高崎 慎也

1977 年生．1999 年東京大学理学部情報科学科卒業．現在，同大学大学院理学系研究科情報科学専攻修士課程在学中．並列プログラミング環境の開発に関する研究に従事．



中村 周吾

1968 年生．1993 年東京大学大学院農学系研究科応用生命工学専攻修士課程修了．1995 年東京大学大学院農学生命科学研究科応用生命工学専攻博士課程中退．同年より東京大学大学院農学生命科学研究科助手．核酸・タンパク質などの生体高分子の構造・機能の計算機による解析と，生体高分子の計算機シミュレーションに必要な並列処理技術に関する研究に従事．生物物理学会会員

池口 満徳

1967年生．1994年東京大学大学院農学系研究科応用生命工学専攻博士課程修了．同年，東京大学農学部助手．1996年より東京大学大学院農学生命科学研究科助手．博士（農学）．生体分子，液体統計力学，分子シミュレーション解析の研究に従事．生物物理学会会員

清水謙多郎（正会員）

1957年生．1985年東京大学大学院理学系研究科情報科学専攻博士課程修了．理学博士．1998年より東京大学大学院農学生命科学研究科教授．オペレーティングシステム，並列・分散処理，生命情報科学の研究に従事．ACM，IEEE Computer Society，電子情報通信学会，日本ソフトウェア科学会，日本シミュレーション学会，生物物理学会，農芸化学会各会員．