マルチメディア通信と分散処理 71−16
（1995．7．13）

ワークステーションクラスタにおけるSCIDDLEライブラリの評価

齊藤哲哉◇, 城和貴◇, Hans Peter Lüthi†, Peter Arbenz‡, 福田晃◇, 荒木啓二郎,◇

◇ 奈良先端科学技術大学院大学 情報科学研究科
† Interdisciplinary Project Center for Supercomputing, ETH
‡ Institute for Scientific Computing, ETH

SCIDDLEは、並列アプリケーションを並列コンピュータやワークステーションクラスタといった非均質なネットワーク環境に分散化するツールである。我々は、次世代のスーパーコンピューティング環境構築のための第一段階として、SCIDDLEの性能評価を行った。これにより、非同期呼び出しの制御を行う関数がシステムの性能に大きく影響することがわかった。また、実際にSCIDDLE上で動作する化学計算アプリケーションDISCOについても性能評価を行い、SCIDDLEがアプリケーションの性能にどのような影響を与えているかを調べた。これらの結果から、筆者らの提案する次世代のスーパーコンピューティング環境、すなわち、広域分散局所並列処理環境構築への指針を示す。

An Analysis of the SCIDDLE Library on a Workstation Cluster

Tetsuya Saito◇, Kazuki Joe◇, Hans Peter Lüthi†, Peter Arbenz‡, Akira Fukuda◇, Keijiro Araki◇

◇ Graduate School of Information Science
. Nara Institute of Science and Technology
† Interdisciplinary Project Center for Supercomputing, ETH
‡Institute for Scientific Computing, ETH

The SCIDDLE Compiler is a tool for distributing parallel applications on heterogeneous network environment of workstation cluster or on parallel computers. To construct the next generation of supercomputing environment, we evaluate the performance of the SCIDDLE system function in the first step. the results lead us to find the SCIDDLE library function of controling asynchronous calls has a large effect on the system performance. We also evaluate the DISCO, which is a quantum chemical application on SCIDDLE, to investigate how the SCIDDLE has effects on the application. the results show us the possible way of constructing the next generation of supercomputing environment, that is the globally distributed locally parallel processing environment we are going to propose.

1 はじめに

科学技術計算アプリケーションに必要とされる莫大な量の計算に対し、これまで計算機は主にベクトル処理を持ってその要求に答えてきた。ベクトル処理とは科学技術計算の時間的空間的局所性を利用した技術であり、プログラム中の完全なループや規則的なデータ・アクセスといった条件が整わなければ高速計算が困難であるという側面を持つが、近年に至る集積度とスイッチング速度の向上率が、これらソフトウェア的な制限を上回っていた。しかしながら、このようなハードウェア技術革新に物理的な限界が見え始めてきた今日、分散・並列処理が次世代の最も有力な大規模計算方式と言えるであろう。

並列処理技術は、時間的空間的局所性を利用したベクトル処理技術の拡張であると見なせる。すなわち、アプリケーションの持つ、時間的空間的規則性を利用して、当該プログラムを並列に高速実行させるのである。与えられた並列計算機に最適な並列プログラムを新たに開発することは、期待される速度向上率を考えると妥当なことではあるが、既存の逐次処理を前提としたプログラムを新たに並列プログラムとして再開発することは、コスト及び信頼性の問題から、困難な場合が多い。従って、ユー

ザがベクトル化コンパイラ同様、自動並列化コンパイラ[1]を必要とするのは当然のことであろう。ところが、既存の逐次型プログラムの自動並列化は、ベクトル処理に比べて極めて困難であり、現在研究段階にある。筆者らは既にループ並列化[2]、データ分割を考慮したタスク・グラフ[3]、データおよびタスク分割の統合アルゴリズム[4]といった基礎研究を行なっており、現在、自動並列化コンパイラの実装中である。

一方、分散処理技術はアプリケーションの持つ時間的空間的区分性、すなわちデータ分割とロードバランシングを利用した当該プログラムの分散高速実行方式である。逐次型プログラムの時間的空間的区分化は、プログラムの自動並列化同様、研究段階にある。筆者らは既に分散環境で並列タスクを実行するための分散化コンパイラ、SCIDDLE[5]を開発しており、化学アプリケーションに対する実際の適用例[6]についても報告している。

以上の研究背景から自然に導かれるHPC環境は、巨大な数値計算プログラムに対し、広域に分散処理技術を適用し、局所的には並列処理技術を用いる、広域分散局所並列処理である。ここで、広域な分散処理とはインターネット等で接続されたノードに対して、プログラムとデータを適当に分割配置し、必要に応じてプログラムやデータの再配置を行なうことであり、局所的な並列処理とは、各ノードにおける並列処理である。各ノードは、ワークステーション・クラスタ、並列計算機、ベクトル計算機であり、与えられた分割済みのプログラムとデータに対してさらに並列処理をほどこす。

筆者らは現在この環境を実現すべく、SCIDDLEコンパイラの拡張を検討している。2章で説明するように、SCIDDLEは豊富なデータ通信とプロセス・コントロールを有するが、これを上記広域分散局所並列処理方式に拡張するためには、その詳細な分析が必要となる。

そこで本稿では、既存のワークステーション・クラスタにおいて、SCIDDLEの提供するライブラリ関数の性能を評価し、上記拡張への指針とする。さらに、同コンパイラの潜在的な性能を測定するために、既存のアプリケーションDISCOを同ワークステーション・クラスタにおいて実際に稼働させ、詳細な分析を行なう。本論文で得られる成果は、既存の並列処理と分散処理を一つの技術に融合するための第一歩であり、将来の科学技術計算の要求するであろう、膨大な計算量を低コストで実現するための極めて有望な研究につながるものと確信する。

以下本稿では、2章でSCIDDLEコンパイラの概要を、3章で、そのライブラリ関数のベンチマーク・テスト結果についてそれぞれ説明する。4章では、計算化学における分子軌道法（molecular orbital method）のハートリー・フォック方程式（Hartree-Fock equation）を直接自己無撞着場最適法（DirectSelfConsistent Filed Optimization method）で解くアプリケーション・プログラムDISCOについての概説を行ない、5章では当該プログラムにSCIDDLEコンパイラを適用して、ワークステーション・クラスタで実行したベンチマーク結果について報告する。6章では得られたベンチマーク結果を検討し、SCIDDLEコンパイラの拡張について考察する。

2 SCIDDLEとは

SCIDDLEは、移植性に優れ、並列アプリケーションをワークステーションクラスタや並列コンピュータといった非均質なネットワーク環境に分散化するために容易に利用できる通信環境である[5]。SCIDDLEはクライアント・サーバモデルに基づくプログラミングモデルであり、すべてのプロセス間通信はリモートプロシージャコール(RPC)によって行われる。利用者は、簡単な宣言言語を用いてクライアント・サーバ間の通信プロセスのインターフェースを定義する。また、SCIDDLEは、非同期RPCによる並列プログラミングもサポートしており、リモートプロセスや同期の管理といったSCIDDLEの機能は、ユーザライブラリのサブルーチンとして提供されている。プログラミング言語としては、C、C++、FORTRAN77をサポートしている。

RPCで動作するということは、他のアプローチ、とりわけメッセージパッシングと比較していくつか利点がある。まず第一に、プロセス間通信が基本的なプロシージャ・コールによって言語レベルで提供されていることである。プロシージャは、逐次プログラムを記述するときにも用いられている方法である。次に、スタブと呼ばれる通信を受け持つコードを適切な宣言から自動的に生成できることである。利用者は通信を意識することなくプログラミングすることが可能である。最後に、アプリケーションを分散化するのに僅かな変更を加えるだけでよいということである。前述の量子化学アプリケーションでは、クライアントルーチンに僅かな変更を加えるだけで分散化することができた。

この言語レベルでのアプローチとは対照的に、PVM[7]、MPI[8]といったメッセージパッシングシステムでは、メッセージの送受信に加えて、メッセージバッファの管理、データのマーシャリング、アンマーシャリング(基本的なデータ型やそれらのベクトル、構造型データは「手動で」基本的なデータ型に分けなければならない)といった低レベルな操作を利用者自身で行わなければならない。一般的なRPCシステム、特にSCIDDLEでは、これらの操作はシステムによって扱われる。従って、プログラマはプロセス間通信の詳細について知る必要はないのだが、このSCIDDLEパッケージの簡便さが、可能な通信トポロジにおける一般性の喪失とのトレードオフになっている。メッセージパッシングシステムでは、一般に通信パターンに対する制約はないが、SCIDDLEのプログラミングモデルでは、ツリー構造の通信トポロジしか許されていない。しかし、後者はやっかいな並列アプリケーションの通信パターンであるが、コンピュータネットワーク上に最も効率よく実装されているアプリケーションである。本稿の4章で議論する最適なエネルギーとそれに相当する電荷比重を計算するためのDISCOプログラムは、そのようなアプリケーションの典型例である。

他のSCIDDLEのインターフェース定義言語のきわだった特徴としては、多次元配列操作の提供がある。特に、一定の部分配列を異なるサーバに容易に分散することができる[9]。逆に、伝統的なRPCシステムでは、メッセージパッシングシステムと同様に、通常一次元配列しか扱えない。これは、個々の部分配列の要素が連続したメモリ配置を確保しないような配列分散に対して、ユーザが部分配列要素を送信しするのに先だって明示的にそれらを別々の「バッファベクトル」にマーシャリングし、受信後に受信側でそれらをアンマーシャリングしなければならないことを意味している。これは、RPCシステムが扱うべき

である低レベルな仕事である。また、SCIDDLE ではミックスドランゲージも可能になっており、FORTRAN、C、C++でそれぞれ異なっている配列表現を考慮している。

3 SCIDDLE ライブラリ関数のベンチマーク

SCIDDLE はユーザが利用できる多くのライブラリ関数を提供している [5]。本章ではライブラリ関数のベンチマークの結果について述べる。

ライブラリ関数のベンチマークには、DEC 社の DEC 3000 が Ethernet で複数台接続されているワークステーションクラスタを用いてベンチマークを行った。しかし、SCIDDLE は DEC 3000 で構成されるワークステーションクラスタ上での実行を考慮していないため、ソースプログラムに対して若干の変更が必要であった。

今回用いたライブラリ関数のベンチマークプログラムは、SCIDDLE のランタイム・ライブラリに追加する形で C 言語を使用して作成した。各ユーザライブラリ関数の始めと終りに計測用の関数を呼びだし、その差をとって、標準エラー出力に関数名とその関数の処理にかかった CPU 時間を出力した。ここで CPU 時間とは、ユーザモードでの処理時間と、カーネルモードでの処理時間の和のことである。今回、Elapsed 時間の計測を行っていないのは、メッセージの送受信など、純粋な計算性能に直接関係のない要因を排除するためである。また、CPU 時間の計測には `getrusage()` 関数を使用した。

表 1は、後ほど紹介する DISCO というアプリケーションを実際に動作させたときに呼び出された SCIDDLE のライブラリ関数が消費した CPU 時間を示している。各時間の単位はマイクロ秒である。

Function	CPU Time(usec)
`sc_cgadd()`	30
`sc_cgcreate()`	0
`sc_cgextractready()_A`	51
`sc_cgextractready()_B`	582
`sc_initmaster()`	64741
`sc_initserver()`	23709
`sc_srvsendsig()`	2928
`sc_srvstart()`	6964
`sc_srvstartmulti()`	6708
`sc_srvterm()`	3456
`sc_svcattach()`	0
`sc_svcdetach()`	91744
`sc_svcdetachall()`	91744

表 1: SCIDDLE のライブラリ関数のベンチマーク結果

ここで `sc_cgextractready()` は、非同期呼び出しについて結果の求まった最初の非同期呼び出しを返す関数である。非同期呼び出しはクライアント側のキューで管理されており、すべての非同期呼び出しについて状態を保持している。`sc_cgextractready()` は、自分が呼び出された時に、キューにあるすべての非同期呼び出しについて状態をチェックするが、キューに結果の求まった非同期呼び出しがある場合とない場合でその動作を変えるため、`sc_cgextractready_A()`、`sc_cgextractready_B()` としてそれぞれ別々に計測することにした。

表から分かるように、処理時間を特に要しているのは、`sc_initmaster()` と `sc_initserver()` である。このことから、最上位のクライアント・サーバプロセスの初期化やサーバーのコンフィギュレーションを最小限度にとどめなければパフォーマンスに大きな影響を与えることがわかる。

`sc_cgextractready()` は非同期呼び出しの制御に使われている。`sc_cgextractready_A()` と `sc_cgextractready_B()` では処理時間にかなり差があるので、このモジュールがアプリケーションの性能に大きな影響を与えているといえる。また、`sc_srvstart()`、`sc_srvterm()` の結果は、同一プログラム中で動的にサーバのコンフィグレーションを換えるようなことを行わない方が良いことを示している。サービスの接続や終了を担う関数 `sc_svcattach()`、`sc_svcdach()`、`sc_svcdetachall()` は、際だって悪い値を示していないが、サービスを動的に換えるようなアプリケーションには相性が悪いと考えられる。

一方、非同期呼び出しを制御する関数は `sc_cgextractready()` 以外、いずれも効率が良く、これを使ったロードバランシングやスケジューリングへの可能性を示している。

4 DISCO とは

Schrödinger 方程式を解く方法の中では、ハートリー・ホック法が最も重要である。ハートリー・ホック方程式は一般的な固有値問題として定式化できる。

$$F\psi = ES\psi, \tag{1}$$

ただし、ホック演算子 Fはハミルトン演算子に対応し、(非直交) ガウス型関数の原理で表される。この方程式は、最適エネルギー Eと対応するホック方程式 (1) の固有値 (電荷比重) を得るために変動的に解かれる。

ハートリー・ホック方程式は、近似値理論に基づいているにもかかわらず、かなり正確な結果を出し、計算化学では広く用いられている。例えば、分子構造は、たいていの場合、2%より少ないの誤差で計算される。より重要なこととして、ハートリー・ホック法が理論手法のより高いレベルへの開始点として役に立っているということがある。

この手法は、n 個の基底関数 (システムが考慮している電子数に比例する) において、$O(n^3)$ になっている。大規模分子 (50 から 100 原子) の構造最適化をするのはスーパーコンピュータで数時間程度で可能である。

ハートリー・フォック方程式を解く直接自己無撞着場最適法 (DIrect Self Consistent Field Optimizatin method; direct SCF method) は、Almlöf らによって DISCO プログラムとして始めて実装された [10]「直接」アルゴリズムでは、必要なすべてのデータを中間記憶装置から取り出すというよりも、secular 方程式を必要に応じて設定して計算することで伝統的なハートリー・フォックモデル実装の入出力ボトルネックを回避している。この CPU 制約アルゴリズムによって、伝統的な手法で解けた問題よりもかなり大規模な問題を解くことが可能になった。

ハートリー・フォック計算での計算負荷は、固有値問題の解法での負荷というよりもむしろフォック行列の評価での負荷である (1)。しかし、フォック行列の計算は、独立にかつ任意の順序で実行可能なタスクに分割することができる。

ハートリー・フォック計算での通信比率に対する計算は、アプリケーションの型 (分子エネルギーの計算、分子

構造に関するエネルギー誘導体の計算) に依存して n^3/n^2 もしくは n^3/n であるので、この型の粗粒度並列方式は、(マルチプロセッサ) マシンのネットワークに容易に拡張できる [11]。SCIDDLE-DISCO を用いることで、専用モードでの 2 台のクレイ C90 コンピュータのネットワーク上で 16 ギガフロップ以上の性能を得ることができた。その 16 プロセッサを搭載する 2 台のマシンは、それぞれピッツバーグスーパーコンピュータセンター (PSC) とミネソタ州にあるクレイリサーチ社共同計算センター (CCN) に位置しており、インターネットで接続されていた。実験の日 (1993 年 8 月 14 日 (土))、平均持続通信割合はだいたい 50KBytes/s であった。通信を含む全アプリケーションに対して得られた並列度は 99.3%であり、これは 29.5 (32 台中) 台のクレイ C90 のそのままの性能、もしくは、75 台の Cray Y-MP プロセッサ以上の性能を示している。

最近では、SCIDDLE 3.0 を用いて、ベクトル共有メモリ型並列コンピュータや Cray C90 や IBM SP2 といった分散メモリ型並列コンピュータのようなヘテロな型の計算資源を接続しはじめた。これによって、異なる型のタスクを最も適した資源へスケジューリングできる。この場合、SCIDDLE は、異なる (マルチプロセッサ) サーバを接続するのに用いられる。サーバ内の計算は、最も速いプロセス間通信を選択するために、別のプログラミングモデル (HPF、MPI、自動タスキング) を用いて実行される。

5 ワークステーションクラスタにおける DISCO のベンチマーク

本章では、ワークステーションクラスタ上で実行した DISCO のベンチマーク結果について述べる。ベンチマーク環境は SCIDDLE のライブラリ関数のベンチマーク環境と同じである。

DISCO はもともと RS/6000 上で FORTRAN77 を用いて実装されていたため、DEC FORTRAN でコンパイルするのに多少の変更を必要とした。また、筆者らの環境では、数値演算ライブラリ BLAS なかったため、別途用意した。その他、DEC FORTRAN でサポートされていない他の関数は、筆者らで用意した。

DISCO の性能評価は、サーバ数を 1 から 50 まで変化させて行った。この評価の主な目的は、DISCO から呼び出される SCIDDLE のライブラリ関数の処理時間と DISCO 自体の処理時間との間にある関係を調べることであった。

図 1は、DISCO を実行した際に、DISCO と SCIDDLE のライブラリ関数がそれぞれ消費する CPU 時間を示している。この図では、サーバ数が増加するにつれ、CPU 時間が増加しているが、サーバ数が 20 を過ぎたあたりから増加の度合が多少であるが小さくなっているところが興味深いところである。この現象を調べるために、別の実験を行って結果を詳細に分析することにした。

図 2は、DISCO が消費する CPU 時間に対して SCIDDLE ライブラリ関数が消費する CPU 時間の占める割合を示している。この図から、SCIDDLE ライブラリ関数自体の処理時間がアプリケーションの処理時間に与える影響は比較的少ないことがわかる。

図 3は、10 台のサーバによる DISCO の実行過程を詳細に示している。クライアントの何らかの活動 (例えば逐次動作) によって割り込まれた 8 つのイタレーション

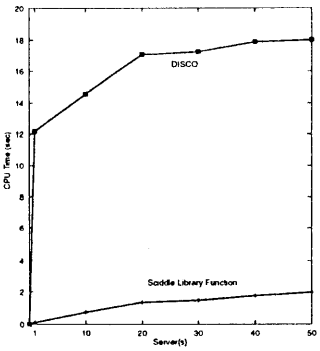

図 1: DISCO と SCIDDLE のライブラリ関数がそれぞれ消費した CPU 時間

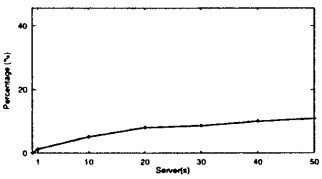

図 2: SCIDDLE ライブラリ関数が占める CPU 時間の割合

がある。各縦線はサーバが処理するためにクライアントが送信したタスクパッケージの開始を示している。この例では、全体で 274 CPU 時間しかないが、476 タスクが生成されるので、負荷分散が困難である (25 秒から 30 秒の間にある 2 番目のイタレーションを参照)。しかし、ここでは、めざましい速度向上を得るというよりもむしろ SCIDDLE のライブラリコールの効果を測定することに関心があった。

図 4は、SCIDDLE のライブラリ関数によって消費される全 CPU 時間に対して、`sc_cgextractready()` の消費する時間が占める割合を、`sc_cgextractready_A()`、`sc_cgextractready_B()` について調べたものである。

また図 5は、SCIDDLE のライブラリ関数 `sc_cgextractready()` について、`sc_cgextractready_A()`、`sc_cgextractready_B()` のそれぞれの呼び出し回数について調べたものである。この図から、サーバ数 20 台を境に、`sc_cgextractready_A()`、`sc_cgextractready_B()` の呼び出し回数に大きな変化が生じていることがわかる。これは、サーバ数の増加によって、結果の求まった非同期呼び出しがキューに入っている状態が多くなるためである。

これら 2 つの図から、`sc_cgextractready_A()` の呼び出しが増えているにもかかわらず、`sc_cgextractready()` が全体に占める割合は低くとどまっていることがわかる。これは、表 1からも明らかなように、`sc_cgextractready_A()` の消費する CPU 時間が `sc_cgextractready_B()` の消費する CPU 時間よりかなり少ないためである。図 4を見ると、`sc_cgextractready_B()` の割合が急激に減少している。これは `sc_cgextractready_B()` が

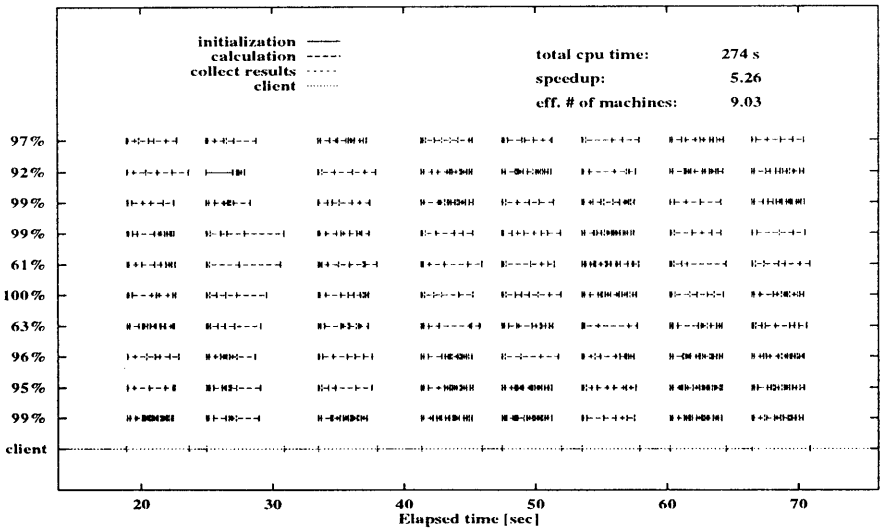

図 3: 10 サーバによる DISCO の実行結果

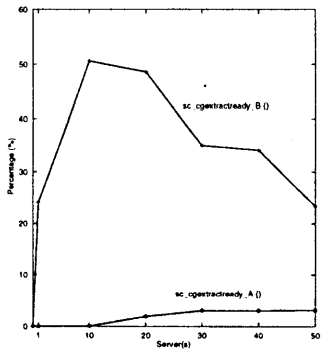

図 4: sc_gextractready() の消費する CPU 時間が占める割合

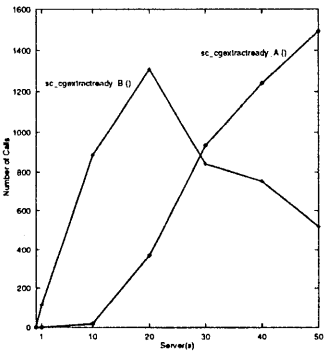

図 5: sc_gextractready() の呼び出し回数

sc_cgextractready_A() に置きかわった効果が現われた結果である。

図 6は、クライアントとサーバ (一台あたり) がそれぞれ消費する CPU 時間を示したものである。クライアントの CPU 時間に注目してみると、やはりサーバ数が 20 台を越えたあたりから若干ではあるが変化の割合が小さくなっている。これは前述のとおり、sc_cgextractready_A() の呼び出し回数が sc_cgextractready_B() の呼び出し回数を上回ったためであるといえる。今回のベンチマークでは、サーバの処理が比較的小さいためその効果があまり出ていないが、サーバの処理が大きければ、より効果が出ていたと考えられる。

6 考察

広域分散局所並列処理環境の構築という目的達成のために、広域でも局所でも矛盾のないのはもちろん、フレキシブルなシステムを設計しなければならない。

実験結果から、非同期呼び出しを管理する関数 sc_cgextractready() がシステムの性能に大きく影響していることがわかった。SCIDDLE システム全体の性能は悪くないが、さらなる性能向上のために、アクティブ・メッセージ [12] といった別の技術を導入することも考えられるだろう。SCIDDLE は非同期呼び出しシーケンスをサポートしているので、これらを使用することが可能である。

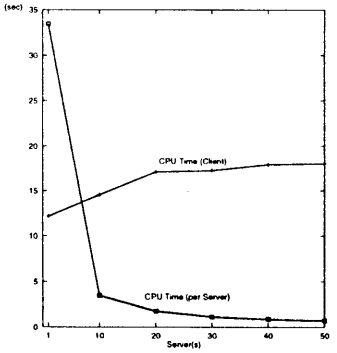

図 6: クライアントとサーバで消費される CPU 時間

複数サーバに関しては、data privatisation や冗長なデータアクセスの除去を考慮すべきである。複数サーバでは、どのサーバが提供されたデータに責任を持つか決定するのに owner computing rules[13] が適用できる。これもまた冗長なデータアクセスを引き起こす。そのような場合、その除去に関していくつかの技術 [14] が必要となると考えられる。

今後、SCIDDLE に改良を加え、局所並列処理を行うことを考えている。今回の実験によって SCIDDLE の分散処理の性能を確かめることができた。従って、次の重要なプロジェクトである局所並列処理に移行することができるであろう。

7 結論

広域分散局所並列処理環境構築に向けての第一段階として、SCIDDLE のライブラリ関数と実際のアプリケーションである DISCO のベンチマークをワークステーションクラスタ上で行った。SCIDDLE のライブラリ関数の評価結果は、非同期呼び出しの管理関数がシステムの性能に与えうる影響を示唆していた。また、SCIDDLE-DISCO のベンチマーク結果においていくつかの注目すべき現象があり、それらを詳細に解析することによって、非同期呼び出しの管理関数がどのようにアプリケーションの性能に影響しているかが確かめられた。また、これらの結果を考慮することで、SCIDDLE コンパイラを広域分散局所並列環境の一部として拡張するのに考えられる方法を示した。

次のステップとして考えられるのは、局所並列処理との融合とともに、実際に SCIDDLE コンパイラを拡張することである。

参考文献

[1] Polychronopoulos, C., Girkar, M. B., Haghighat, M. R., Lee, C. L., Leung, B. P. and Schouten, D. A.: Parafrase-2: An Environment for Compiling, Partitioning,Synchronizing, and Scheduling Parallel Programs, *International Journal of High Speed Computing*, Vol. 1, No. 1 (1989).

[2] Kitasuka, T., Joe, K., Schouten, D., Fukuda, A. and Araki, K.: A Loop Parallelization Technique for Linear Dependence Vector, in *Proceedings of International Conference on Parallel Architectures and Compilation Techniques* (1995), will appear.

[3] Nakanishi, T., Joe, K., Polychronopoulos, C., Fukuda, A. and Araki, K.: The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning, in *Proceedings of 7th Int' Wokshop on Languages and Compilers for Parallel Computing* (1994).

[4] Nakanishi, T., Joe, K., Saito, H., Polychronopoulos, C., Fukuda, A. and K.Araki, : CDP^2 Algorithm – a Combined Data and Program Partitioning Algorithm on DPG –.

[5] Arbenz, P., Lüthi, H. P., Sprenger, C. and Vogel, S.: SCIDDLE: A Tool for Large Scale Distributed Computing, Research Report 213, ETH Zürich, Computer Science Department (1994), To appear in Concurrency: Practice and Experience 1995.

[6] Pittsburgh Supercomputer Center, : *Grand Challenge Computing: Stalking the Mighty Teraflop – Projects in Supercomputing 1994* –, Pittsburgh, PA (1994).

[7] Sunderam, V., Geist, G. A., Dongarra, J. and Manchek, R.: The PVM Concurrent Computing System: Evolution, Experiences, and Trends, *Parallel Computing*, Vol. 20, pp. 531–545 (1994).

[8] Walker, D. W.: The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers, *Parallel Computing*, Vol. 20, pp. 657–673 (1994).

[9] Arbenz, P., Gates, K. and Sprenger, C.: A Parallel Implementation of the Symmetric Tridiagonal QR Algorithm, in Siegel, H. J. ed., *The Fourth Symposium on the Frontiers of Massively Parallel Computation*, pp. 382–388, Los Alamitos, CA (1992), IEEE Computer Society Press.

[10] Almlöf, J., Faegri, Jr., K. and Korsell, K.: Principles for a Direct SCF Approach to LCAO-MO Ab-Initio Calculations, *Journal of Computational Chemistry*, Vol. 3, pp. 385–399 (1982).

[11] Lüthi, H. P. and Almlöf, J.: Network Supercomputing: A Distributed-Concurrent Direct SCF Scheme, *Theoretica Chimica Acta*, Vol. 84, pp. 443–455 (1993).

[12] T. Eicken, S. G., D. Culler and Schauser, K.: Active Messages: A Mechanism for Integrated Communication and Computation, in *Proceedings of The 19th Annual International Symposium on Computer Architecture*, pp. 256–266 (1992).

[13] S. Hiranandani, K. K. and Tseng, C.: Compiler Optimizations for Fortran D on MIMD Distributed Memory Machines, in *Proceeding of Supercomputing'91*, pp. 86–100, Los Alamitos, California (1991), IEEE Computer Society Press.

[14] Granston, E. and Veidenbaum, A.: Detecting Redundant Accesses to Array Data, in *Proceedings of International Computer Symposium*, pp. 854–869 (1991).