システムソフトウェアと
オペレーティング・システム 67－7
(1994. 12. 8)

ゲーム木の並列探索のための分散的実行管理機構

中山 泰一, 赤澤 忠文, 野下 浩平
電気通信大学 情報工学科
〒182 調布市調布ヶ丘 1-5-1
yasu@cs.uec.ac.jp

あらまし　ネットワークにより結合された多数のUNIXワークステーション群を対象として、システム全体を仮想的な並列計算機とみなして並列処理を行う研究が現在盛んに行われている。本論文では、数十台のUNIXワークステーション群によりゲーム木の並列探索を効率的に行うための分散的実行管理機構を、UNIXの標準的なシステム機能のみを用いて設計・実現した。ゲーム木の例として詰将棋を解くプログラムについて計算実験を行ったところ、逐次プログラムにより長時間必要とした詰将棋問題のほとんどのものについて大幅に計算時間が節減でき実用的な時間で解ける結果を得た。問題によってはマシン台数に比例するよりはるかに大きい並列処理効果が確認された。

和文キーワード　ワークステーション群, 並列プログラミング, 探索, ゲーム, 詰将棋.

Distributed Execution Mechanism for Searching Game-Trees in Parallel on the Workstation Cluster

YASUICHI NAKAYAMA, TADAFUMI AKAZAWA
and KOHEI NOSHITA

Department of Computer Science,
The University of Electro-Communications
Chofu, Tokyo 182, Japan

Abstract　A workstation cluster, which consists of tens of UNIX workstations connected by networks, can be used for high-performance parallel computing. In this paper we describe an experimental study of solving game problems in parallel on the workstation cluster. We have designed a new distributed parallel execution mechanism, which is applicable to any type of UNIX workstations. We have implemented this parallel execution mechanism on sixty-four UNIX workstations, and have solved fifty hard Tsume-shogi problems (checkmating problems of Japanese chess). The experimental results show that the parallel program with our distributed parallel execution mechanism can solve most of the problems much more quickly than the best sequential program on a UNIX workstation. In particular the speed-up factor is more than the number of workstations for ten to twenty percent of those problems.

英文 key words　Workstation Cluster, Parallel Programming, Searching, Games, Tsume-Shogi.

1 はじめに

分散配置された多数のUNIXワークステーションをイーサ・ネットなどのネットワークにより結合して分散計算機環境を構築することは、広く見られるようになっている。このような分散計算機環境を対象として、システム全体を仮想的な並列計算機とみなして並列処理を行う研究が現在盛んに行われている [1, 2, 4, 10, 11]。

筆者らは上記の分散計算機環境を利用して、単一の計算機では時間のかかるようなゲーム木の探索を並列に実行させることをめざす。すなわち、ネットワークにより結合された多数のUNIXワークステーション群全体で1つのゲーム木を並列探索し、計算時間の大幅な短縮と解答率の向上を図るものである。ゲーム木探索は並列実行が本質的に有用であり、逐次的に実行させた場合と比べてマシン台数に比例するよりはるかに大きい性能向上が得られることも多くあると考えられる。

本論文では、ゲーム木の並列探索を効率的に行うための分散的実行管理機構をUNIXワークステーションに標準的に用意されているシステム機能のみを用いて設計し、ネットワークにより結合された数十台からなるUNIXワークステーション群において分散的実行管理機構を実現した。実現した実行管理機構はUNIXワークステーションであれば複数の機種が入り混じっている環境でも利用可能なものである。

本論文ではゲーム木の例として、詰将棋を解くプログラムについて計算実験を行った。単一の計算機では長時間かかる詰将棋問題50題を上記の分散的実行管理機構を用いて並列に解いたところ、多くのものについて大幅に計算時間が節減できる結果を得た。また、それらのうち10～20％のものについてはマシン台数以上の並列処理効果が得られた。

ゲーム木 (特にAND-OR木) では、ミニマックスの性質より、深さ優先探索の逐次プログラムに対して並列計算により台数以上の速度向上の可能なことが理屈として知られているが、それを本論文で具体例で実証したことになる。さらに、この実験によって、従来のどのプログラムによっても解けなかった問題をはじめて解き、並列計算により解答能力が真に増えることを示すこともできた。

以下、まずゲーム木の並列探索とそのために必要な機能について述べ、分散的実行管理機構の設計方式について述べる。最後に、ゲーム木探索の並列化による速度向上に関する計算実験について述べる。

2 ゲーム木の並列探索

2.1 対象とする問題

ゲーム木の探索アルゴリズムに関する研究はこれまでにも多く行われており [7] 、その深さが低い木については1台のワークステーション上での逐次プログラムにより十分に速く探索することが可能となっている。しかしながら、木の深さが増えるにつれ探索すべきノード数は爆発的に増加し、逐次プログラムでは実用的な時間で探索することが困難になる。

詰将棋を解くプログラム [3, 8] では、詰手数が17手以下の問題はこれまでの逐次プログラムで十分速く解ける。詰手数が19～25手の中篇問題は、深さ優先探索を基本とした逐次プログラムT2によればほぼすべてが解けるが、多くの問題について過度に時間がかかる。一方、最良優先探索を基本にした逐次プログラムItoによれば解ける問題はごく短い時間に解けるが、全部の問題の解を求めるのは難しい。この程度の手数の問題をすべて短時間に解くことが研究課題となっている。(詰将棋プログラムでは無駄合の取扱いなどのため、たとえば25手詰の問題でも深さ30を超える木を探索することがよく生じる。)

本論文では、逐次プログラムでは実用的な時間で探索することが困難なゲーム木探索問題を対象として、並列探索することにより計算時間の大幅な短縮と解答率の向上を図る。

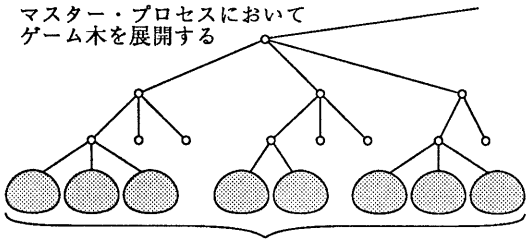

図 1: 並列探索のモデル

2.2 並列探索のモデル

ゲーム木を並列探索する手法として、システム全体で1つだけの管理プロセス(マスター・プロセスとよぶ)においてゲーム木を展開して多数の小問題に分け、それらの小問題を各マシン上にある下請けプロセス(スレーブ・プロセスとよぶ)に解かせるというモデルを採用する(図1)。

ネットワークにより結合されたUNIXワークステーション群を用いて並列処理を行う場合、マルチプロセッサ・システムなどと比べて通信のオーバーヘッドが大きい。ゆえに、個々のスレーブ・プロセスに解かせる小問題の粒度を大きくして、マシン間の通信回数を少なくすることが、並列探索アルゴリズムを設計する際に重要である[9]。また小問題の粒度を大きくすることは、マスター・プロセスがシステム全体で1つだけある本モデルにおいて、マスター・プロセスでの処理がボトルネックとなることを避ける効果もある。

2.3 実行管理機構に求められる機能

実行管理機構に求められる機能として、

(1) マスター・プロセスから遠隔マシン上にスレーブ・プロセスを自動的に起動して解くべき小問題が与えられること、
(2) スレーブ・プロセスが解いた小問題の結果をマスター・プロセスがすみやかに受けとれること、
(3) 実行中のスレーブ・プロセスに対してマスター・プロセスから中断・初期化できること、
(4) 初期化したデータやハッシュ表をなるべく共用できること、

が考えられる。(3)は、あるスレーブ・プロセスが返した結果により別のスレーブ・プロセスが解いている小問題が無駄なものとなった場合に用いられるものである。(4)は、ゲーム木探索では同一または近似局面の計算を何度も行わないようにするためハッシュ表を用いるのが一般的であり[3]、それを効率的に行うためのものである。

3 分散的実行管理機構の設計と実現

3.1 システムの概要

本論文で述べる分散的実行管理機構が対象とする計算機環境は、数十台程度のネットワーク接続されたUNIXワークステーション群である。UNIXワークステーションであれば、複数の機種が入り混じっていても構わない。

システムの移植性、拡張性を考慮し、ユーザレベルのライブラリとして実行管理機構を設計した。表1に、代表的なプリミティブとその機能を示す。このうち(A)〜(C)はマスター・プロセスにより呼ばれるプリミティブ、(D)はスレーブ・プロセスにより呼ばれるプリミティブである。

3.2 スレーブ・プロセスの作成

ゲーム木を展開して生成する小問題は数が多く、マスター・プロセスは遠隔マシンのそれぞれに多数の実行要求を与えることになる。

しかしながら、実行要求が発生するごとに遠隔マシン上にスレーブ・プロセスを生成して小問題を解かせる方式では、プロセスの生成と消滅や通信回線の確立にかなりのプロセッサ時間を消費し、あまりにオーバーヘッドが大きい。たとえば、マシン台数が100台の場合、並列に実行することのできるスレーブ・プロセスの個数は100個であり、ゲーム木を展開することによ

表 1: 代表的なプリミティブとその機能

(A) スレーブ・プロセスの作成

`start_slave` 遠隔マシン上にスレーブ・プロセスを作成

(B) スレーブ・プロセスとのデータ送受信

`send_slave` スレーブ・プロセスに小問題を送信

`get_ready_slave` 結果を返すことができるスレーブ・プロセスの id を得る

`recv_slave` スレーブ・プロセスから計算結果を受信

(C) スレーブ・プロセスのリセットなど

`reset_slave` スレーブ・プロセスをリセット

`restart_slave` スレーブ・プロセスを強制終了・再起動

(D) スレーブ・プロセスにより呼ばれるプリミティブ

`reset_enable` リセット時にどの関数から再起動するかを設定

`reset_lock` リセットをサスペンドさせるためのロックをかける

`reset_unlock` 上記のロックを解放する

り小問題が10000個発生するからといって10000個のスレーブ・プロセスを生成するのは無駄である。

そこで本実行管理機構では、共有メモリ型マルチプロセッサ・システムのための実行管理機構であるアクティビティ方式並列実行機構 [6, 5] を応用し、あらかじめスレーブ・プロセスを各マシンに 1 つずつ作成し、それを繰り返し再利用する設計とした (図 2)。

スレーブ・プロセスを再利用することのもう 1 つのメリットとして、同一のスレーブ・プロセス内において初期化すべきデータやハッシュ表も再利用できることがある。

マスター・プロセスにおいて `start_slave` プリミティブを呼ぶことにより、遠隔マシン上にスレーブ・プロセスを自動起動できる。このとき、スレーブ・プロセスが実行するプログラムを引数で指定する。返り値としてスレーブ・プロセスの id が返される。遠隔マシン上でのプロセスの自動作成には、UNIX ワークステーション上で標準的に動作している rexecd を利用した。

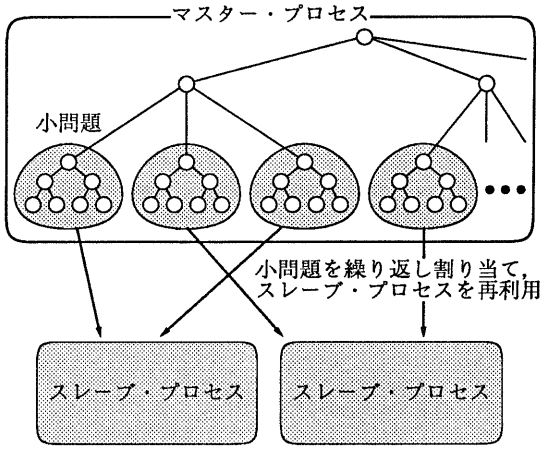

図 2: スレーブ・プロセスの再利用

上記のようにしてスレーブ・プロセスとして起動されたプログラムでは、必ず最初に `reset_enable` プリミティブを呼ぶ必要がある。これは、**3.5**で述べるマスター・プロセスからのリセットがかかった時にどの関数から再起動するのかを指定するためのものである。スレーブ・プロセスにおいて `reset_enable` プリミティブが呼ばれた段階で、スレーブ・プロセス作成の処理が完了する。

3.3 マスター・プロセス—スレーブ・プロセス間の通信回線

作成されたスレーブ・プロセスそれぞれにはマスター・プロセスとの間の通信回線として図 3に示す 2 種類の回線がつながれている。

「通常回線」はスレーブ・プロセスとのデータ送受信に用いられるもので、スレーブ・プロセス側では標準入出力に接続される。すなわち、スレーブ・プロセスは標準入力からのデータ読み込みによりマスター・プロセスから小問題を受け取り、標準出力へのデータ書き出しにより計算結果をマスター・プロセスに返すことができる。

「緊急回線」は、マスター・プロセスがスレー

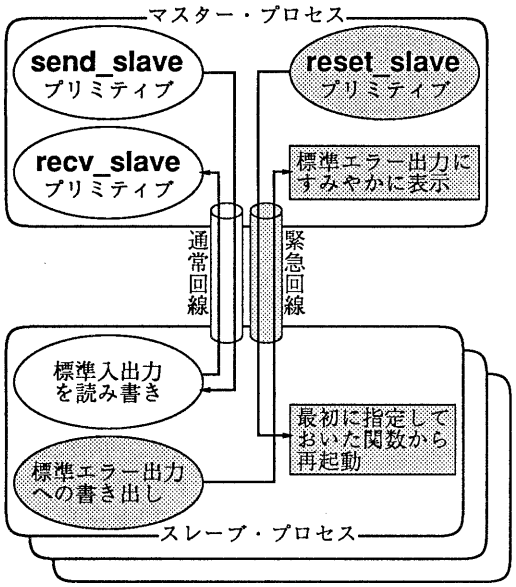

図 3: マスター・プロセス—スレーブ・プロセス間の通信回線

ブ・プロセスをリセットする時の緊急メッセージを送信するために用いられる。また、スレーブ・プロセス側で標準エラー出力に書き出したメッセージは「緊急回線」を通してマスター・プロセスに送られ、すみやかにマスター・プロセスの標準エラー出力に表示される。この機能により、デバッグ時のメッセージ出力などが可能である。

3.4 スレーブ・プロセスとのデータ送受信

マスター・プロセスからスレーブ・プロセスに小問題を与えるためには、そのスレーブ・プロセスの id を指定して send_slave プリミティブを呼ぶ。

マスター・プロセスがスレーブ・プロセスからの結果を受け取る場合には、まず get_ready_slave プリミティブを用いて結果を返すことができるスレーブ・プロセスの id を得たのち、recv_slave プリミティブでその id を指定して結果を受け取るという仕様とした。

マスター・プロセスでは、スレーブ・プロセスから結果を受け取ると、続いてそのスレーブ・プロセスを次節に述べる reset_slave プリミティブを用いてリセットし、再び send_slave プリミティブにより次の小問題を与えることを繰り返す。

3.5 スレーブ・プロセスのリセット

あるスレーブ・プロセスが返した結果により、別のスレーブ・プロセスが解いている小問題が無駄なものとなってしまうことがしばしば起こる。そこで、小問題を解いている途中のスレーブ・プロセスでもすみやかにリセットできる reset_slave プリミティブを用意した。

マスター・プロセス上で reset_slave プリミティブを呼ぶと「緊急回線」を通してスレーブ・プロセスに緊急メッセージが送られる。スレーブ・プロセスでは、「緊急回線」を通してメッセージが送られたことによる割り込みを受けて「通常回線」上のデータをクリアし、さらに、最初に reset_enable プリミティブで指定しておいた関数から再起動する。

なお、スレーブ・プロセス上でハッシュ表への挿入などの割り込み不可の処理を行う場合のために、reset_lock プリミティブと reset_unlock プリミティブが用意されている。また、不幸にして回線故障などでスレーブ・プロセスの制御ができなくなったときのために restart_slave プリミティブが用意されている。restart_slave プリミティブを用いると、現在動作中のスレーブ・プロセスを強制的に終了させ、同一のスレーブ・プロセスを再度作成しなおす。当然のことながら、プロセス生成、通信回線の確立などを再度行い、またハッシュ表などの再利用もできないものであるので、スレーブ・プロセスのリセットの目的には reset_slave プリミティブを用いる方がはるかに効率的である。

3.6 システムの実現

前節までに述べた設計により、分散的実行管理機構を実現した。プログラム記述には C 言語

表 2: 動作確認をしたマシン環境

機種名	OS (BSD or System-V)
Sun SPARC Station	SunOS 4.1.3 (BSD)
Sony NEWS	NEWS-OS 4.3 (BSD)
DEC system 5830	ULTRIX V4.3 (BSD)
IBM-PC 互換機	BSD/386 1.1 (BSD)
Silicon Graphics Indy	IRIX 5.2 (System-V)

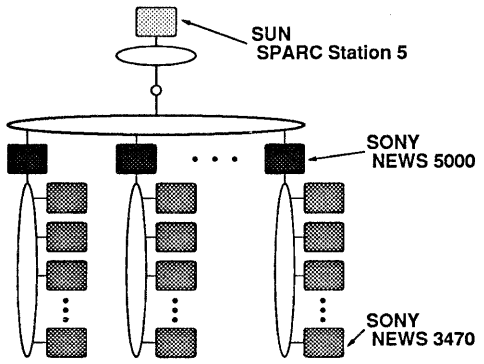

図 4: 計算実験に用いた分散計算機環境

を用い、全体で約1300行である。現在のところ、表2に示すマシンおよびそれらを組み合わた分散計算機環境で正常に動作することが確かめられている。

次章で述べる計算実験には、Sun SPARC Station 1 台と Sony NEWS 約 100 台がネットワークにより結合された分散計算機環境を使用した(図 4)。NEWS5000 相互間は FDDI、それ以外のマシン間はイーサ・ネットで接続されている。この分散計算機環境での基本的処理性能としては、スレーブ・プロセスの作成に 200 ミリ秒程度、マスター・プロセス—スレーブ・プロセス間のデータ送受信、および、スレーブ・プロセスのリセットに 10〜20 ミリ秒程度が必要である。

4 計算実験

4.1 実験対象

本論文では、ゲーム木の例として詰将棋を解くプログラムについて計算実験を行った。解くべき詰将棋問題を月刊詰将棋パラダイス (Vol.39, No.1-No.12) に掲載された、詰手数が 19〜25 手の「92 年度 (前・後期) 短大」の全 50 問題とした。**2.1**で述べたように、この程度の問題をすべて速く解くことが研究課題となっているが、ここで選んだものは最も難しい問題集とされているものである。逐次的プログラム T2 によると 49 問題解けるが 4 割以上は 3 時間以上かかる [3]。

詰将棋を解く並列プログラムとして、筆者らはこれまでに P12 と P32 の 2 種類のプログラムを作成した [9]。それぞれのプログラムの特徴は表3の通りである。ここで T2 と T3 はすでに作成ずみの逐次プログラムであり、現在最も進んでいるものの 2 つである。とくに、P32 は、本論文の実行管理機構を利用して既存の 2 つの逐次プログラムを結合して並列プログラムにまとめたものであり、ソフトウェア開発のためにも本論文の方法が有効であることがわかる。なお、この結合の仕方は詰将棋特有の事情があり自明ではない [9]。

表 3: 計算実験に用いた 2 種類の並列詰将棋プログラム

	マスター・プロセス	スレーブ・プロセス
P12	先読みにより局面の評価値を求め、その評価値により左下優先でゲーム木を展開し小問題を作成する。ただし、後手番の展開・小問題作成は逐次的に行う。	深さ優先探索にもとづくプログラム T2 により与えられた小問題を解く。与えられた小問題と手数に対して、その手数以下で詰むならば、手数と解手順、その手数以下で詰まないならば、そのことを報告する。
P32	最良優先探索にもとづくプログラム T3 を用いてゲーム木を展開し小問題を作成する。P12 と同様に後手番の展開・小問題作成は逐次的に行う。	

プログラム P12 を用いた計算実験では、1 台のマシンによる逐次プログラム T2(「1 台$_{(T2)}$」と記す) と、16 台、64 台のマシンによるプログラム P12 (それぞれ「16 台$_{(P12)}$」、「64 台$_{(P12)}$」と記す、以下同様) との計算時間を比較した。プログラム P32 を用いた計算実験では、1 台$_{(P32)}$、16 台$_{(P32)}$、64 台$_{(P32)}$の計算時間を比較した。

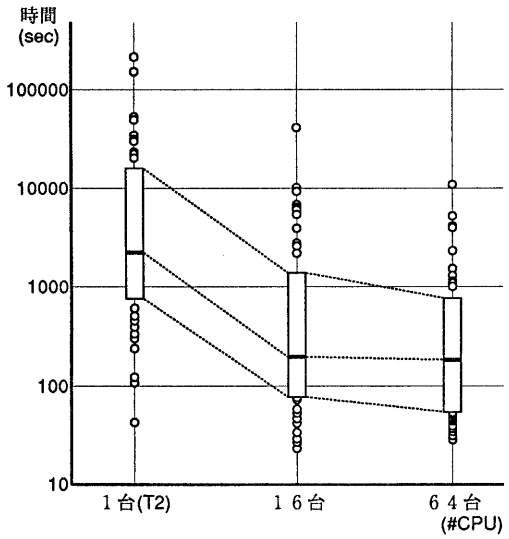

図 5: プログラム P12 の台数効果,
▬: 中央値 (median), □: ±25 %,
○ : 実際の値 (actual value).

表 4: P12 で顕著な性能向上を示した例
(単位:秒, 括弧内は性能向上比)

問題	1 台$_{(T2)}$	16 台	64 台
前期 2	55000(1)	72(763)	66(833)
23	243000(1)	6496 (37)	2413(100)
後期 3	16200(1)	229 (70)	155(104)
11	10200(1)	59(172)	33(309)
24	23100(1)	90(256)	167(138)

なお、マスター・プロセスが配置されているマシン上にもスレーブ・プロセスが配置されている。とくに、1 台$_{(P32)}$は 1 台のマシン上だけにマスター・プロセスとスレーブ・プロセスとが配置されている。

4.2 実験結果と考察

まず、プログラム P12 を用いた計算実験の結果を示す。64 台$_{(P12)}$では、

50 問題中 44 問題が 3 時間以内で解けた。
43 問題が 1 時間以内で解けた。
31 問題が 10 分以内で解けた。

64 台$_{(P12)}$で 3 時間以内で解けた 44 問題に対して、計算時間を比較したのが図 5である。中央値とその上下 25 %のデータに着目すると確かに並列処理による速度向上が現れている。顕著に並列処理効果が現れたものの例を表 4に示す。

次に、プログラム P32 を用いた計算実験の結果を示す。64 台$_{(P32)}$では、

50 問題中 48 問題が 3 時間以内で解けた。
46 問題が 1 時間以内で解けた。
39 問題が 10 分以内で解けた。

1 台$_{(P32)}$で 10 時間以内で解けた 45 問題に対して計算時間を比較したのが図 6である。中央値より上 25 %のデータでは台数が増えるごとに速度向上が現れている。しかしながら、中央値より下 25 %のデータでは 64 台$_{(P32)}$は 16 台$_{(P32)}$ほど性能向上しなかった。これは、初めに 64 台のスレーブ・プロセスを生成するオーバーヘッドがかかり短時間で解ける問題ではそのオーバーヘッドの割合が大きくなるためである。

顕著に並列処理効果が現れたものの例を表 5に示す。1 台$_{(P32)}$で 10 時間以内で解けなかった 2 問題が、64 台$_{(P32)}$で 1 時間以内で解けるようになり、全体として 95 %以上が実用的な時間で解けるようになった。

ここで、後期 22 番の問題は、従来どのプログラム (T2 も含む) でも解けていないものである。すなわち、並列計算により解答能力が真に向上することを実証したとともに、この問題が詰むという意味で正しいことをコンピュータで初めて検証したことを意味する。

5 おわりに

本論文では、ネットワークにより結合された多数の UNIX ワークステーション群によりゲーム木の並列探索を効率的に行うための分散的実行管理機構の設計方式について述べた。

UNIX に標準的に用意されているシステム機能のみを用いて分散的実行管理機構の設計・実現し、ゲーム木の例として詰将棋を解くプログラムについて計算実験を行った。

単一の計算機上で逐次プログラムにより長時

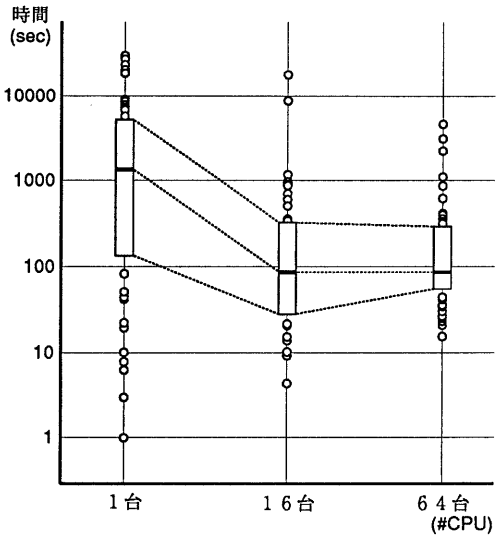

図 6: プログラム P32 の台数効果,
▬: 中央値 (median), □: ±25 %,
○: 実際の値 (actual value).

表 5: P32 で顕著な性能向上を示した例
(単位:秒,#は 24 時間以内で解けない,
括弧内は性能向上比)

問題	1 台	16 台	64 台
前期 7	17508 (1)	757(23)	283 (61)
13	4304 (1)	30(87)	49(143)
後期 4	3119 (1)	58(53)	26(119)
13	# (—)	9883(—)	2208 (—)
22	# (—)	6114(—)	2025 (—)

間必要とした詰将棋問題を対象として並列に解いたところ、ほとんどのものについて大幅に計算時間が節減でき実用的な時間で解ける結果を得た。問題によってはマシン台数に比例するよりはるかに大きい並列処理効果が確認された。

本論文で述べた分散的実行管理機構では、スレーブ・プロセスを繰り返し再利用する設計としたため同一のスレーブ・プロセスでのハッシュ表の再利用が可能である。計算実験に用いた並列プログラムにおいてもマスター・プロセス内、および同一のスレーブ・プロセス内でのハッシュ表利用による計算の効率化を行っている。

しかしながら、複数のスレーブ・プロセスにまたがってハッシュ表を利用することはできない。複数のスレーブ・プロセス間でもハッシュ表が共用できるように分散的実行管理機構に共有ハッシュ機構を用意する改良が考えられ、さらに効率化を図れる可能性がある。共有ハッシュ機構をいかに小さなオーバーヘッドで設計・実現するかについては、今後の課題とする。

参考文献

[1] Barcellos, A.M.P et al.: The HetNOS Network Operating System: a Tool for Writing Distributed Applications, *ACM Operating Systems Review*, Vol.28, No.4, pp.34-47 (1994).

[2] Douglas, C. C. et al.: Parallel Programming Systems for Workstation Clusters, *Yale University Department of Computer Science Research Report*, YALEU/DCS/TR-975 (1993).

[3] 伊藤 琢巳, 野下 浩平: 詰将棋を速く解く 2 つのプログラムとその評価, 情報処理学会論文誌, Vol.35, No.8, pp.1531-1539 (1994).

[4] 前富 博, 伊藤 聡, 島崎 眞昭: 分散処理環境上の仮想並列計算機に対するスケジューリングアルゴリズムの設計とその検討, 第 49 回情報処理学会全国大会, 3T-9 (1994).

[5] 本橋 健, 中畑 昌也, 中山 泰一, 永松 礼夫, 出口 光一郎, 森下 巖: 分散要求管理を用いるアクティビティ方式並列実行機構, 情報処理学会論文誌, Vol.35, No.10 (1994).

[6] 中山 泰一, 永松 礼夫, 出口 光一郎, 森下 巖: 共有メモリ型並列機のための新しいアクティビティ方式並列実行機構, 情報処理学会論文誌, Vol.34, No.5, pp.985-993 (1993).

[7] Nilsson, N.J.: *Problem-Solving Methods of Artificial Intelligence*, McGraw-Hill, New York (1971).

[8] 野下 浩平: 詰将棋, 数学セミナー, Vol.33, No.10, pp.26-27(1994).

[9] 野下 浩平, 中山 泰一, 松本 真一, 赤澤 忠文: 分散的並列計算による詰将棋の解法, ゲーム・プログラミングワークショップ'94 論文集, pp.22-31 (1994).

[10] Rinaldo, F. J. and M.R. Fausey: Event Reconstruction in High-Energy Physics, *Computer*, Vol.26, No.6, pp.68-77 (1993).

[11] Sunderam, V.: PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience*, Vol.2, No.4, pp.315-339(1990).