オペレーティング・システム　55－3
（1992．6．8）

共有メモリ型並列機のためのアクティビティ方式並列実行機構の研究

―――タスクの親子関係を利用する後処理実行機構の導入―――

中山　泰一　永松　礼夫　森下　巖
（東京大学工学部）

共有メモリ型並列計算機上で細粒度の並列処理を効率良くサポートするために、プロセス生成要求と実行を分け実行時に既存のプロセス資源を再利用する、アクティビティ方式が提唱されている。しかし、同期待ちの場合にはプロセス実体が新たに生成され効率が低下する。本発表ではタスクの親子関係を利用して、子タスク待ちにおける効率の低下を防ぐ方式について提案する。具体的には、子タスク待ちの後に行われる処理を「遺言」の形で宣言し、これを最後に終了する子タスクに実行させることによってプロセス実体が新たに生成されることを防ぐ方式である。提案方式により性能改善を図れることが、実験により確かめられた。

Activity Based Execution Mechanism for Fine Grain Parallel Processing on Shared Memory Machines

――A New Construct for Tasks to be Executed After the Completion of All the Children Tasks――

Yasuichi Nakayama　Leo Nagamatsu　Iwao Morishita
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113 Japan

The activity based execution mechanism was proposed for effective execution of a large number of fine grain tasks on shared memory machines. This mechanism deals with an execution request and its execution separately to reuse a light-weight process prepared for a previous task execution. In the proposed mechanism, when a process suspends for waiting the completion of all the children tasks, a new process must be created. In this paper, a new construct called "make will" is introduced to reduce the number of process creation. When a task is declared by the construct, it is executed after the completion of all the children tasks by utilizing the process used for the last child task. It is shown that both the execution time and memory consumption are reduced by the revised mechanism.

１．はじめに

汎用の高並列の計算機の開発が盛んに進められている。高並列機を利用しようとする場合、高並列の処理を効率よく管理、サポートするシステム・ソフトウェアの提供が必要である。

並列処理の管理方式としては、並行プロセスに基づく方式が広く用いられている。これは並列に実行すべき処理が発生するとプロセスを生成し、その実行が完了するとプロセスを消滅させる方式である。これは、たとえばUNIXではforkシステム・コールにより提供される。

しかしながら、高並列機において細粒度の並列処理を試みるとき、プロセスを用いる方式ではプロセスを実現するために費やされるプロセッサ時間、および、メモリ領域は小さくなく、これが大きな問題となる。最近では、MACHシステムにおけるスレッドの実現[1]をはじめとして、プロセスをできるだけ軽量化した軽量プロセスの実現方式に関する研究が盛んである[2]。軽量プロセスを用いる場合でも、その生成と消滅、およびその並列実行の管理にはかなりのプロセッサ時間を消費する。また、軽量プロセスでも少なくともある量のスタック領域は割り当てなければならないので、メモリ資源の浪費も依然大きな問題となる。

プロセスを実現するためのコストを軽減する方法として、田胡ら[3]はアクティビティ方式を提案した。これは並列実行可能な単位の各々にプロセスを割り当てるのではなく、実行単位の発生と、その単位を実行する機構の生成を別個のものとして扱う方式である。あらかじめプロセスを並列に動作可能な個数だけ用意して、これを繰り返し利用する。そのためにプロセスの生成と消滅に要するプロセッサ時間およびメモリ消費を節減できる。この方式により、論理的にはプロセスを直接利用するのと同等の環境を、より少ないオーバーヘッドで実現することが可能である。

しかしながら、従来のアクティビティ方式の設計では必ずしも効率よく実行できない種類の応用プログラムが多く存在する。具体的に、並列処理によりプログラミングする場合に、親のタスクが子のタスクを生成した後に子タスクの終了を待って後処理を実行するような、いわゆるfork-join型のアルゴリズムで応用プログラムを記述することがきわめて多い。このように親タスクが子タスクを待ち合わせる場合には、従来のアクティビティ方式の設計ではプロセスを新たに生成することになる。これが頻繁に起こる形式の応用プログラムでは実行効率がプロセスを直接用いる方法と等しくなってしまうという問題が起こる。

田胡らは文献[3]において、なるべくタスクの待ち合わせをしない形式で応用プログラムを作成することが望ましい、と述べている。このことは利用者にとりプログラムの記述性を低下させるものであり、できるだけ利用者がこのことを気にせずプログラム作成できる方がより有用である。

そこで筆者らは、応用プログラムに構造的特徴、具体的には生成されるタスクの親子関係があることを積極的に利用し、親タスクが子タスクの終了後に後処理を実行することが必要となる場合においても効率よく処理できるように、アクティビティ方式の実行機構に設計変更を加えた方式を提案する。提案方式においては後処理が「遺言」の形式で子タスクに伝えられ、その「遺言」は最後に処理を終える子タスクによって最優先で実行される。また、提案方式により、後処理を含まない応用プログラムも効率よく実行することが可能である。

本論文では提案方式による設計と、その評価結果について報告する。理想共有メモリ型並列機のシミュレータにより実験を行った。その結果、提案方式により後処理が効率よく実行されることが確認された。さらに、後処理を行わないように応用プログラムを記述し従来方式により実行した場合と比較しても、よい効率が得られることが明らかになった。

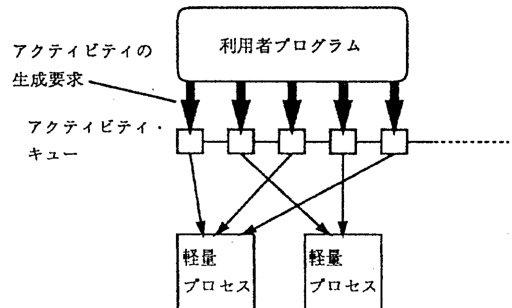


図１　アクティビティ方式による並列処理環境

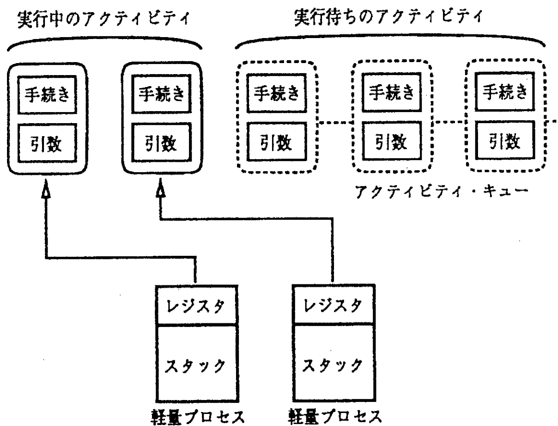


図２　アクティビティの実行機構

２．アクティビティ方式

２．１　アクティビティ方式の基本原理

本節ではアクティビティ方式の基本原理について述べる。アクティビティ方式では、並列に実行すべき処理単位の発生と、資源を割り当てて処理単位の実行を機構を用意することを別個に取扱う。並列実行可能な単位が生じると同時にその実行のためにプロセスを生成することはしない。

応用プログラムはアクティビティとよばれる単位で並列実行要求を生成する。応用プログラムがつぎつぎとアクティビティを生成すると、管理系はこれを図１に示すようにアクティビティ・キューに結合して保持する。

一方アクティビティの実行機構としてはプロセッサ数と同数の軽量プロセスをあらかじめ用意しておく。それ以上の軽量プロセスを用意しても、並列に実行することができないから無駄である。

軽量プロセスは、未実行のアクティビティをアクティビティ・キューから１個とってきて実行する。この軽量プロセスがそのアクティビティの実行を完了すると、また次のアクティビティをアクティビティ・キューから取ってくる。以下同様の動作を繰り返す。このように、あらかじめ用意された軽量プロセスを繰り返し利用するので、軽量プロセスの生成消滅に消費されるプロセッサ時間を節減できる。

具体的にアクティビティは図２に示すように、実行すべき手続きとその引数から構成されており、１個あたりのメモリ消費量は小さい。また管理系が実行しなければならない処理はアクティビティ・キューの結合と取り出しだけであり、処理量も小さくボトルネックを生じにくい。

軽量プロセスの内部状態はレジスタとスタックで表現される。アクティビティが終了するとこの内部状態はすべて不要になる。したがってプロセスはスタック・ポインタの初期化だけでそのまま新しいアクティビティの実行を開始できる。

２．２　従来方式の問題点

アクティビティ方式では、実行途中で同期により実行が中断される場合には、その実行を再開するときのためにプロセスの内部状態を保持しておくため、新たにもう１つプロセスを生成することを行う。

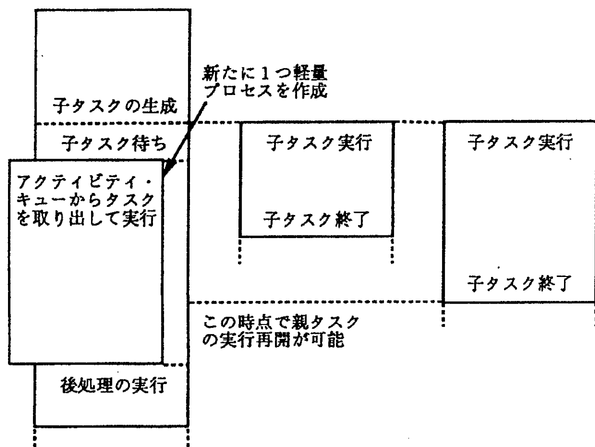
たとえば、並列処理によりプログラミングする場合には、親のタスクが子のタスクを生成した後に子タスクの終了を待って後処理を実行するような、いわゆるfork-join型のアルゴリズムで応用プログラムを記述することが一般的に考えられる。このように親タスクが子タスクを待ち合わせる場合には、図3（a）に示すように新たに軽量プロセスを作成し、その新しい軽量プロセスを用いてアクティビティ・キューから取り出してきたタスクを実行する。すべての子タスクが終了して親タスクの実行再開が可能となると、親タスクは元の軽量プロセスを用いて後処理を実行する。

以下、本章ではこのように後処理が実行される場合の問題点について述べる。

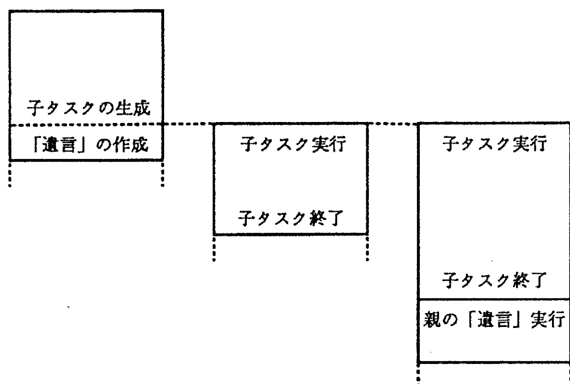
（１）プロセスを新たに生成することによるオーバーヘッドがかかること。

新たなプロセスを生成・消滅させることに要するプロセッサ時間は小さくない。またプロセス生成に要するメモリ消費、とくにスタック消費は膨大なものとなる。

具体的な例として７都市巡回セールスマン問題を並列処理を用いて解く場合を考えると、通常応用プログラムは図４に示すようにまず６個の子タスクを作り、そのそれぞれが５個の子タスクを作り…、を繰り返すようなアルゴリズムとなり、全体では1237個のタスクが作られることになる。実行途中で中断するタスクがない場合には、プロセスはプロセッサ台数分を用意す



（a）従来方式における子タスク待ちと後処理の実行



（b）提案方式における「遺言」の作成とその実行

図3　従来方式、提案方式それぞれにおける後処理の実行

るだけですむが、fork-join型のアルゴリズムを用いて子タスク待ちを行う場合、517個のタスクが実行途中で中断することとなる。結果として517回プロセスの生成が必要となる。プロセッサ台数が8台の場合を仮定すると、実行途中で中断するタスクがない場合と比べて約65倍の数のプロセスを生成することになる。スタックの消費もこれに比例して増大する。

田胡らはこのようなことをできるだけ防ぐように、できるだけ実行途中で中断しないように応用プログラムを作成するようにアドバイスしている[3]。新たなプロセスの生成が頻繁に起こる形式の応用プログラムでは実行効率がプロセスを直接用いる方法と等しくなってしまうことを述べている。

（2）すべての子タスクが終了した直後に後処理が実行されるとは限らないこと。

すべての子タスクが終了して親タスクの後処理が行える状況になっても、親タスクを実行していたプロセッサはすでに新たなプロセスを生成しており、それが別のタスクを実行していることが考えられる。このタスクが終了するまで後処理の実行を待つことになると、実際には後処理の開始まで必要以上に待たされることになる。また、これら複数のタスクをコンテクスト切換えにより並行に処理するとしても、これに大きなコストが必要となる。複数のタスクを並行して処理しなければならないプロセッサの傍らで、子タスクを実行していたプロセッサはアイドル状態となっている、といった事態も起こりえる。

親タスクの後処理はすべての子タスクが終了した時点でできるだけ優先して実行されることが望ましい。それも、その後処理実行が子タスクを実行していたプロセッサ上で行えるようになっているとなおよい。その方がプロセッサ資源の利用効率も向上するものと考えられる。

3．新方式の提案

3．1　「遺言」の作成と最後の子による「遺言」の実行

前章に述べた問題は、実行途中で待ち合わせないように応用プログラムを作成すれば解決することである。しかし、この解決法では後処理を実行し

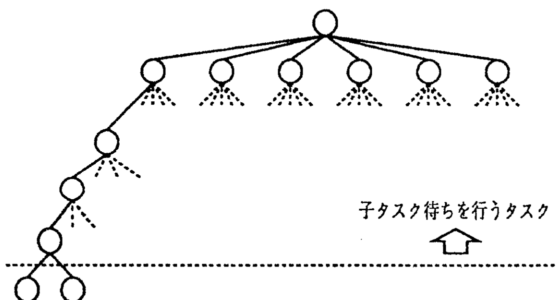


図4　7都市巡回セールスマン問題におけるタスク生成

表１　家系記述子(Family Tree Descriptor) の構造

（自分が実行する）手続き
（〃）　　引数
親のＦＴＤへのポインタ
「末っ子」のＦＴＤへのポインタ
現在生きている自分の子の数
「遺言」の手続き
〃　引数

ないように応用プログラムを作成することとなり、このことはプログラム記述に大きな制限が加わるというプログラマにとっての不便を起こす。

実行途中での待ち合わせのうちでも、子タスクの終了待ちは最も頻繁に起きるものである。子タスクの終了待ちにより親タスクが中断する場合に効率よく処理できる、すなわち後処理が効率よく処理できる実行機構にすることができれば、後処理を含むようなプログラム記述が許され、プログラマにとってかなり記述性のよいシステムにすることができる。

そこで、筆者らは、後処理が実行されるプログラムではプログラムの構造的特徴、具体的に、生成されるタスクの親子関係が後処理が完了するまで保持されることを積極的に利用することにした。まず、親子関係のツリーを構築するために、表１に示す家系記述子（Family Tree Descriptor、以下ＦＴＤとする）という構造体を新たに導入する。親タスクが子タスクを待って後処理を実行するしないにかかわらず、子タスクが作成される時点でＦＴＤを用意して、親のＦＴＤにポインタを用いてリンクする。これにより子タスクは自分の親が誰であるかを知ることが可能となる。また、親のＦＴＤには現在生きている自分の子の数を格納しておく。

次に、親タスクはもし後処理を実行したい場合には、後処理を手続きとその引数の形で宣言する。具体的には、図５に示すように、

make_will(後処理の手続き，引数)

というプリミティブを用いて宣言する。このプリミティブがよばれると、ＦＴＤの「遺言」の領域に後処理に関する情報が格納され、同時に親タスクは「遺言」を遺したまま強制的に終了される。

親タスクを実行していたプロセスはその内部状態を保持しておく必要がないので、そのまま次のタスクを実行することができる。これにより新たにプロセスを生成することをしないですむようになる。

最後に、子タスクは処理を終了すると、親のＦＴＤを参照して自分が子タスクのうちで最後に処理を終えたものであるかを判定する。自分が最後の子の場合は図3（ｂ）に示すように親のタスクの「遺言」を最優先で実行する。これにより親タスクの後処理は子タスクの終了直後に実行されることが保証される。

子タスクが「遺言」の実行を終えた結果、もし親が「親の親」から見て最後の子である場合には、さらに「親の親」の「遺言」を実行する、ということを繰り返す。実行すべき「遺言」すべての実行を終えて初めてアクティビティ・キューからの取り出しを行う。

３．２　「子」の１人への実行環境の譲り渡し

前節に述べた機構により、親は後処理を「遺言」の形で残し、最後に処理を終えた子により最優先で実行してもらうことができる。また、親は後処理を宣言した時点で実行環境を保持しておく必要がなくなり、確実にこれを手放すことが可能である。

手放されて開きとなった実行環境において新たなタスクを実行する時に、これをアクティビティ・キューから取ってきてもよいが、親タスクには必ず自分が生成した子タスクが存在するのであるから、自分の「子」のうちの誰か１人に実行環境を譲り渡してやればよい。

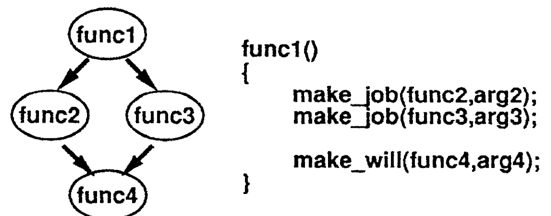


図５　子タスク生成、「遺言」作成のプリミティブ

実際の設計においては「末っ子」に実行環境を譲り渡すようにした。

実行環境を自分の「末っ子」に譲り渡す直接的なメリットとしては、実行すべき新たなタスクをアクティビティ・キューから取り出してくるコストを削減することができることである。ＦＴＤの導入に伴うシステム処理時間が増加することが予想されるが、逆に、アクティビティ・キュー操作の処理時間が減ることで、結果として全体のシステム処理性能が向上することも可能であると考えられる。

自分の子の１人に実行環境を譲り渡すことによる別の効果としては、自分の子孫の処理が他の処理よりも優先して処理されることがある。なぜならば自分の子孫の枝のうち常に１本は優遇され、待たされることなく処理が実行され、葉に至るまでこれが続くからである。これにより応用プログラムによっては処理時間が大きく短縮される可能性がある。少なくとも性能が悪くなることはないと考えられる。

以上、本章で述べた方式により親タスクの後処理が効率よく実行できる機構が実現できる。これをフローチャートで表すと図６に示すようになる。

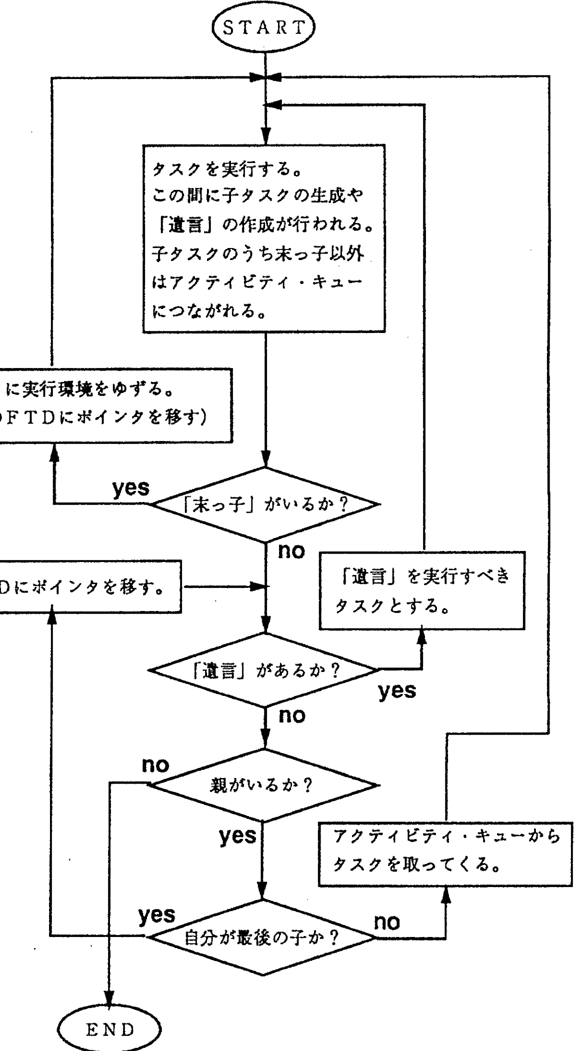


図６　提案方式における並列実行機構のフローチャート

４．実験

４．１　実験環境

提案方式による並列実行機構を設計し評価するために、理想共有メモリ型並列機のシミュレータにおける実験を行った。

実験に用いたシミュレータは、筆者らの研究室において現在設計／実現中である、PSMアーキテクチャによる共有メモリ型マルチプロセッサ・システム[4]の性能評価のために作られたものである。本来は多重命令流プロセッサに多段結合型ネットワークにより結合された共有メモリが付加されたシステム構成の性能評価実験を行うためのものであるが、比較用に理想共有メモリのシステム構成における実験も可能なように作られている[5]。

今回実験するにあたって、より一般的な並列機環境を作り出すため、

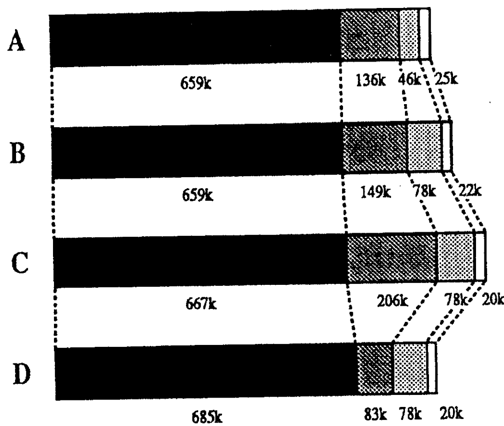
○理想共有メモリ

○単一命令流プロセッサ

の条件でシミュレータを用いた。プロセッサ台数は８台とした。

応用プログラムとしては、７都市巡回セールスマン問題について、枝刈りをしない場合、する場合の両方を実現し動作させた。

7都市巡回セールスマン問題（枝刈りをしない場合）



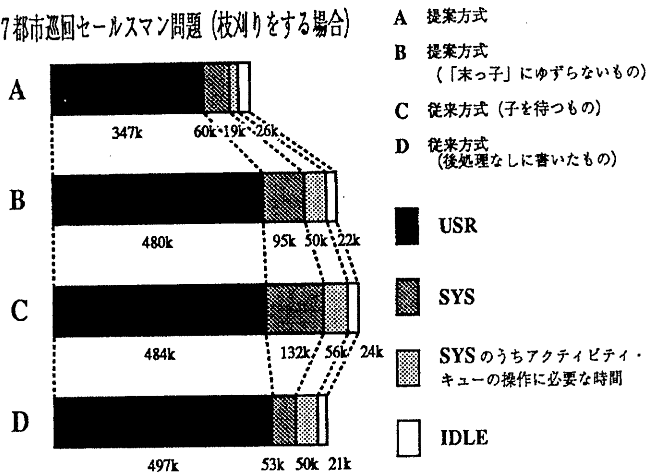


図7　シミュレーション実験の結果

並列実行機構としては、

（A構成）提案方式、

（B構成）提案方式のうち「末っ子」への実行環境の譲り渡しを行わないもの、

（C構成）従来方式で後処理のために子タスクを待つもの、

を用意した。さらにこれに加え、

（D構成）後処理を含まないように応用プログラムを書き換え従来方式の並列実行機構で動かしたもの、

を用意し、計4つのシステム構成で実験を行って比較した。

4．2　実験結果と考察

実行した処理時間の結果を図7に示す。数字はシミュレータのクロック時間を示す。

まず、枝刈りをしない場合ではA構成、B構成、C構成の順によい性能が得られた。従来方式によるC構成に比べB構成はアクティビティ・キュー操作の時間には変化がなかったものの、その他のシステム処理時間が大きく減少した。これは子タスク待ちにおけるプロセス生成・消滅の時間が削減できたためと考えられる。

さらに、実行環境を「末っ子」に譲り渡すA構成ではこれに加えてアクティビティ・キュー操作の時間も削減されることが確かめられた。

また、応用プログラムを後処理を含まないように書き換えて従来方式の並列実行機構を用いて動作させたD構成との比較においてもわずかながらA構成は性能の向上が見受けられた。ＦＴＤの導入に伴うコストの増加の分はアクティビティ・キュー操作のコストの減少で十分に補うことができることが明らかになった。総じてプログラムの記述性がよい分A構成が優れていると判断できる。

次に枝刈りをする場合においても、提案方式が効率よく処理できることが確認された。とくにA構成ではユーザ処理時間が著しく減少した。これは、「末っ子」に実行環境を譲り渡す結果、自分の子孫を優遇して実行する、いわばDepth Firstなスケジューリングが行われる結果、巡回セールスマン問題において枝刈りの判定のよりよい基準が早く求まるためと考えられる。

最後に、メモリ消費の観点から提案方式と従来方式との比較を行った。アクティビティ・キューやＦＴＤなど、並列実行機構の管理のため用いられるメモリ量は提案方式の方がわずかに多いという程度で、大きく増加することはなかった。逆にスタック消費は表2に示す通り、提案方式のA構成は従来方式のC構成の約1/63のスタック消費ですみ、大きくスタック消費を節減できた。

表2　スタック消費の比較（単位は bytes）

A構成（提案方式）	1664
C構成（従来方式）	105240

5．おわりに

従来のアクティビティ方式による並列実行機構では効率よく実行することが困難であった後処理を含む応用プログラムにおいても、効率よく実行することが可能となるような実行機構を提案した。また、シミュレーション実験よりプロセス生成・消滅のためのプロセッサ時間およびメモリ消費が実際に大きく節減されることを確認した。

本提案方式では「遺言」の形式で後処理を子タスクに伝え、これを最後に処理を終える子タスクにより再優先に実行させることが可能である。この「遺言」の問題点としては、スタックが切り替わるためにローカル変数などの情報を渡すことができないといったことがある。これを解決する方法としては、親タスクのスタックなどの環境も後処理を実行する子タスクに移動させる方法がある。この、環境切替により後処理実行を実現する方式についても現在小林ら[6]が設計、実現を行っており、システムが動作している。

今回の実験結果において、提案方式はとくに応用プログラムにおいて枝刈りをする場合に効果が現れることが判明した。これはスケジューリング順序がどのようになっているのが適切かという問題である。この考察については今後の課題である。

並列機の共有メモリのアクセスに局所性がある場合には、並列実行要求のウェイティング・キューを各プロセッサごとに分けるとよいことが知られている[7]。アクティビティ方式においても共有メモリ・アクセスに局所性がある場合にはアクティビティ・キューを各プロセッサごと分ける設計にすればよい。本方式は共有メモリ型並列機に広く適用でき、またその設計・実現も比較的容易に行うことができるものである。

参考文献

[1]Rashid, R.F.: Threads of a New System, UNIX Review, vol.4, No.8, pp.37-49 (1986)

[2]新城靖，清木康：並列プログラムを対象とした軽量プロセスの実現方式，情報処理学会論文誌，Vol.33, No.1, pp.64-73 (1992).

[3]田胡和哉，桧垣博章，森下巖：共有メモリ型並列計算機のためのアクティビティ方式を用いる並列実行環境，情報処理学会論文誌，Vol.32, No.2, pp.229-236 (1991).

[4]森下巖：多段結合ネットワークを用いる超並列マシンのためのパイプライン化MIMDプロセッサ，情報処理学会論文誌，Vol.31, No.4, pp.523-531 (1990).

[5]永松礼夫，数藤義明，森下巖：多重命令流プロセッサを用いる多段ネットワーク結合共有メモリ型並列機のシミュレーションによる性能評価　――グラフ探索問題の処理時間――，電子情報通信学会技術研究報告，CPSY91-29(1991).

[6]小林健一，中山泰一，永松礼夫，森下巖：共有メモリ型並列機のためのアクティビティ方式並列実行機構(2)　――環境切替により後処理実行を行なう方式の提案――，第45回情報処理学会全国大会（予定）.

[7]Mohr, E., Kranz, D.A. and Halstead, R.H.: Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, IEEE trans. Parallel and Distributed Systems, Vol.2, No.3, pp.264-280 (1991).