システムソフトウェアと
オペレーティング・システム 83－7
（２０００．２．２５）

SR8000 上のスケジューラの実装と評価

宇都宮直樹*、円光豊*、高津弘幸**、助川直伸***
*日立製作所システム開発研究所
**日立製作所ソフトウェア開発本部
***日立製作所中央研究所

バリア同期機構を使用した SR8000 特有の並列アプリケーションプログラムをマルチプログラミング環境で効率的にサポートするギャングスケジューラを実装し、実アプリケーションプログラムによる性能測定を実施した。特に、空間分割と時分割を混合した新たなスケジューリング方法を提案・実装し、性能評価の結果、有効性を示すことができた。

Implementation and Evaluation of the SR8000 Task Scheduler

Naoki Utsunomiya*, Yutaka Enkou*, Hiroyuki Takatsu**, Naonobu Sukegawa***
*Systems Development Laboratory, Hitachi, Ltd
**Software Development Division, Hitachi, Ltd.
***Central Research Laboratory, Hitachi, Ltd.

The gang scheduler is implemented to support high-performance execution of SR8000 parallel application programs. The new scheduling policy with both benefits of space slicing and time slicing is presented and evaluated. It is shown that the new policy can amortize the additional performance penalty with the help of user-level hints and further improvement.

1.はじめに

近年著しく増大する計算パワーへの要求に答えるため、従来のベクトル型計算機から、分散メモリ型の並列計算機への移行が進んだ。通常分散メモリ型並列計算機は、単一、または、複数の RISC プロセサから成るノードを複数、高速ネットワークで接続したアーキテクチャとなっている。我々が開発した SR8000 も複数プロセッサ構成のノードをもつ分散メモリ型並列計算機で、以下の特徴を持つ。

(1) 従来のベクトル計算機を受け継ぎ、逐次実行プログラムとして記述されたユーザプログラムの DO-LOOP をコンパイラが自動並列化する。

(2) 自動並列化プログラムを逐次実行部分と並列実行部分の繰り返しと見なし、これを効率的に実行する協調型マイクロプロセッサ機構 (COMPAS: CO-operative Micro-Processors in single Address Space) としてハードウェアバリア同期機構を提供し、逐次実行部分と並列実行部分との移行オーベヘッドを削減する。

(3) キャッシュにのらないような大規模データを扱うプログラムの性能低下を押さえるため、疑似ベクトル機構 (PVP: Pseudo Vector Processing) を導入し、ソフトウェアパイプラインを有効利用したコードをコンパイラが生成する。

今日、ユーザの計算機利用形態も多様化し、センター運用の様に、アプリケーションの性能測定や、プログラムの開発、バッチジョブの投入など、多角的な利用方法に対応する必要がある。従って、オペレーティングシステムは、さまざまのユーザ要求をできるだけ満たしつつ、システム全体の資源の効率的使用をスケジュールする必要がある。

並列計算機においては、従来の単一プロセッサ計算機から受け継いだ時分割に加え、空間分割という新たな基準を加えて、設計・評価を行う必要がある。純粋な空間分割によるスケジューリングはある程度単純ではあるが、過去のユーザの慣習などに照らし合わせた柔軟性や性能などに問題があることが多く、時分割機構が必要なケースが多い [1]。従って、並列プログラムを時分割スケジューリングする方法として、ギャングスケジューリングが商用システムなどでは広く用いられているが、通常は、ジョブ毎にスレッド数が異なるといった空間分割も含めたスケジューリングを行うことが多い [2]。また、空間分割をダイナミックに行う 2 レベルス

ケジューラも、システムの負荷に基づき動的にスレッド数を変えたり、同期点でスケジューラにヒントを渡すことにより効率的なスケジューリング方法を提供している [3]。

SR8000 においては、ノード間をまたがるジョブに関しては、静的なパーティション分割に基づいたスケジューリングを行う。ノード内ジョブスケジューリング方法としては、COMPAS 機構を支援するハードウェアバリア機構を複数ジョブで共有するため、ギャングスケジューリング機構を提供する。本論文では、ノード内スケジューリングに焦点を定め、マルチプログラミング環境における性能評価を実施し、その結果を分析する。

2.SR8000 ハードウェア概要

本章では、ノード内スケジューリング機構の前提となるプログラム実行モデルと、SR8000 のノード内ハードウェアアーキテクチャについて説明する。

2.1 COMPAS プログラム実行モデル

COMPAS 機構で対象とするプログラムは、図 2 に示すように逐次実行部分と並列実行部分とから成り、各部分はコンパイラが自動生成する。OS はノードを構成するプロセッサ (IP: Instruction Processor と呼ぶ) 8 台中から、逐次部分実行用プロセッサ (SIP) と、並列実行用プロセッサ (VIP) を割りつける。プロセッサの構成はプログラム実行時に定まり、通常は SIP 1 台、VIP 7 台で、並列部分を両方合わせた 8 台で実行する。逐次実行部では、VIP は並列実行開始時のバリア同期を待つ。

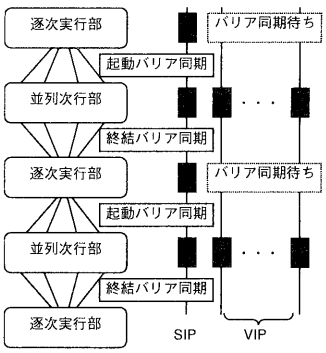

図 2 COMPAS プログラム実行モデル

2.2 ハードウェア概要

図 1 にノード内ハードウェアバリア同期機構を示す。各ノードには IP 9 台から成り、1 台は OS 専用 (SP: System Processor) で、残りはユーザプログラム用である。各 IP は Storage Controller (SC) を介して共有メモリへアクセスする。ノード内バリア同期機構は部分バリア同期を許すために、SC 上にマスクビットを設け、

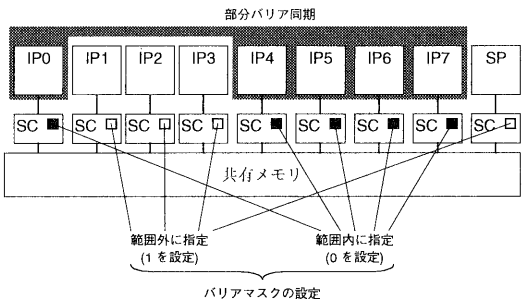

図 1 ハードウェアバリア同期機構

マスクビットが示すグループ内の IP が全てバリア同期点に到達すると、全 IP によるバリア同期待ちを解除する。マスクビットが示すグループ外の IP がバリア同期命令を発行すると例外が発生する。

スケジューリングの観点から見ると、本バリア同期機構は以下の特徴を有する。

(1) 逐次実行時、VIP 上のスレッドはバリア同期命令を発行し同期成立を待つ。

(2) バリア同期機構は 1 組存在し、IP 単位で使用する。

(3) バリア同期機構は同期状態を IP 単位で所有するため、IP とスレッドの対応は 1 対 1 で対応の変更は許さない。

3.ノード内タスクスケジューラ

多様な運用形態やユーザニーズに答えるためには、複数ユーザが同時に使用できるマルチプログラミング環境を提供する必要がある。COMPAS 機構ではノード内バリア同期機構を占有することによりユーザプログラムの実行性能低下を防いでいる。しかし、上記要求に答えるため、同ハードウェアを複数ユーザに共有させる必要がある。

そこで、ギャングスケジューリングによる時分割スケジューリングと、VIP 共有スケジュー

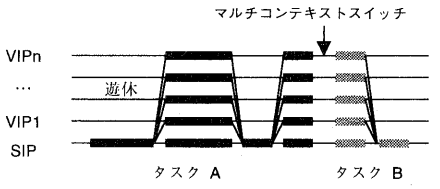

図 3 時分割スケジューリング

リングをスケジューリングポリシーとして提供する。

3.1 時分割スケジューリング

ギャングスケジューリングは、(1) 効率的なスレッド間細粒度同期の提供、(2) データ局所性の有効利用、(3) 対話的な応答時間の提供、(4) 空間分割できない資源の共有等の長所を備えている [1]。近年の並列マシンの使用形態は、センター運用などに代表されるように、複数のユーザが性能測定や、プログラムの開発を行うマルチプログラミング環境であり、資源を有効利用するためにギャングスケジューリングの導入が必要である。

時分割スケジューリングは、COMPAS プログラム実行モデルに基づく並列タスク[1]をギャングスケジューリング方式によりスケジュールする方法で、以下の特徴を有す。

(1) 並列タスク以外のプログラムの影響を排除するため、ノード内の IP を分割し、単一スレッドから成る通常タスクを、OS サービススレッドも含め SP 上でのみ動作させる。

(2) ハードウェア上の制約事項を考慮して、他の並列タスクのスレッドが同時に動作することを禁止する。

(3) 頻繁なコンテキストスイッチによる性能低下を防ぐため、クウォンタムを長めに設定する。

図 3 に時分割スケジューリング時の 2 ジョブの実行過程を示す。

3.2 VIP 共有スケジューリング

VIP 共有スケジューリングは、時分割スケジューリングにおける逐次処理時の遊休 (図 3 参照) によるプロセッサ利用率の低下を押さえるために、時分割機構に空間分割の利点を加味したスケジューリング方法である。

COMPAS プログラム実行モデルでは、プログラムの実行は逐次実行と並列実行の繰り返しとなる。この時、逐次実行部分と並列実行部分の実行時間比により台数効果が変わる。並列化率を ν、実行加速度を a とすると、台数効果は次式で与えられる。並列化率とは、プログラム実行の全命令数に対する並列実行可能命令数の割合である。実行加速度は、並列化可能命令に対して、並列化によりどれほど性能が向上するかを示す。

$$s = \frac{1}{\nu/a + (1-\nu)} \qquad (1)$$

台数効果を 8 IP の半分の 4 を目標とした場合、実行加速度が最高の 8 倍であったとしても、並列化率は $\nu = 6/7 = 0.857$ 以上必要である。台数効果を更に上げようとすると、並列化率も上げる必要がある。COMPAS プログラム実行モデルでは、逐次部分における VIP のバリア待ち時間を減らすことにより並列化率を上げることが可能となるが、これはコンパイラの最適化と、もともとのプログラムの性質に依存する。並列化率の高いプログラムは特定のベンチマークプログラムに限られるので、それ以外のプログラムに関しては、使用プロセッサ数増加による実行加速度上昇が台数効果に余り寄与しない。従って、逐次実行時の遊休時間にタスクを割りつけることによりプロセッサ利用率を向上することが可能となる。

VIP 共有スケジューリングは並列化率が中程度のプログラムを対象としたスケジューリングで、ノード内の IP を更に分割し、逐次実行専用の IP である SIP 4 台と 並列実行用 IP である VIP 4 台とに分ける。SIP は各タスクに占有させ、VIP は各タスク間で共有する、空間分割と時分割の混成構成を形成する。

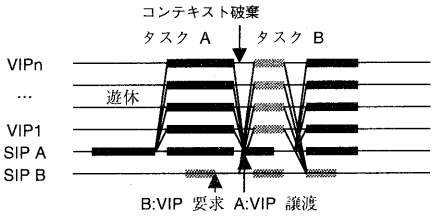

図 4 VIP 共有スケジューリング

本構成により、逐次実行同士、及び、逐次実行と単一タスクによる並列実行は同時実行可能である。あるタスク (図ではタスク A) が並列実行を行っている時他タスク (タスク B) が並列実行を行おうとすると、OS に対して割り込みが発生する。これはバリアマスク設定による例外発生が原因である。OS はこの割り込みを VIP 要求として保持し、VIP を要求したタスク (タスク B) をブロックする。並列実行中タスク (タスク A) がヒントとして発行する VIP 譲渡命令により OS は VIP の解放を検知し、VIP を

[1] タスクはノード内のプログラム実行に対応し、ジョブは複数タスクから構成されうる。但し、ここではノード内のスケジューラのみを扱っているため、タスクとジョブを特に区別しない。ジョブはユーザの観点から、タスクはスケジューラの観点から見たときに使用する。

要求したタスク (タスク B) に VIP を割りつけブロックを解除する。

このように、VIP 共有スケジューリングにおいては、並列部分の開始と終了を OS がスケジューリング契機として捕らえることにより、以前では遊休状態であった VIP を他タスクに割り当て、プロセッサ利用率の向上を計る。

4.スケジューラの実装

ギャングスケジューリングを実装するに当たり留意すべき点は以下の通りである [1]。

(1) マルチコンテキストスイッチの効率的実装

(2) ネットワーク装置などのハードウェアのコンテキスト対する考慮

(3) ハードウェア機構の利用

ギャングスケジューリングにおいては、タスクに含まれるスレッドを別のタスクのスレッドへと一斉に切り替えるマルチコンテキストスイッチを行う必要がある。本スケジューラの場合、バリア同期機構を考慮して、コンテキストスイッチを 2 フェーズに分けて行う。

4.1 ソフトウェアアーキテクチャ

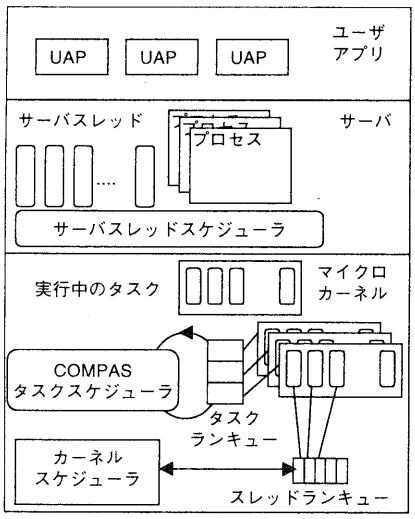

図 5 OS ソフトウェアアーキテクチャ

図 5 に OS のソフトウェアアーキテクチャを示す。本 OS は Mach をベースとしており、基本機能を提供するマイクロカーネル上にプロセス機能等を提供するサーバが存在する。カーネル中にはスレッド単位でプロセッサをスケジュールするカーネルスケジューラと、複数スレッドから成るタスクを単位としてスケジュール(ギャングスケジュール) する COMPAS タスクスケジューラを有する。COMPAS タスクスケジューラはカーネルスレッドとして SP 上で動作する。

更に、サーバ自体もカーネルスレッドと一対一に対応するサーバスレッド機能を提供し、独自のスケジューラとしてサーバスレッドスケジューラを有する。

COMPAS スケジューラをカーネル中に実装した理由は以下のカーネルコールオーバヘッドを抑制するためである。

(1) スケジューリング制御のために必要なカーネルコール

(2) スレッドの停止を保証するために必要なカーネルコール

(3) ハードウェアバリア同期機構の状態を退避・回復するためのカーネルコール

主に、カーネルコールオーバヘッドを考慮してカーネル中に COMPAS スケジューラを実装したが、適切なインタフェースを新設することにより、Unix サーバ内にスケジューラを実装することも可能である。

4.2 2 フェーズコンテキストスイッチ

ギャングスケジューリング機構を実現するためにキーとなるのは、マルチコンテキストスイッチである。本システムの場合、レジスタや仮想メモリ情報に代表される一般のコンテキスト以外に、時分割利用するハードウェアバリア同期機構をもコンテキストとして退避・回復する必要がある。ところが、ハードウェアバリア同期機構はバリア同期を構成するスレッド (IP) の命令により任意の時点で状態が変化する可能性がある。従って、ハードウェアバリア機構の状態を固定するためには、関連するスレッドを停止し、状態変化が起こらないことを保証する必要がある。そこで、コンテキストスイッチを以下のように 2 フェーズで行う。

フェーズ	タスクの停止	タスクの再実行
1	・各スレッドの停止	・ハードウェアバリア状態の回復
2	・ハードウェアバリア状態の退避	・各スレッドの再実行

ハードウェアバリア同期機構導入によりユーザプログラムの実行性能は向上するが、フェーズ 2 が必要であるため、通常のマルチコンテキストスイッチ以上に時間がかる。更に、ハードウェアのバリア状態の退避・回復は各 IP からのみ行うことが可能で、そのための各 IP に対するプロセッサ間割り込みオーバヘッドが、コ

ンテキストスイッチ開始要求として新たに発生する。

5.性能測定

本章では NAS Parallel Benchmark (NPB) プロ

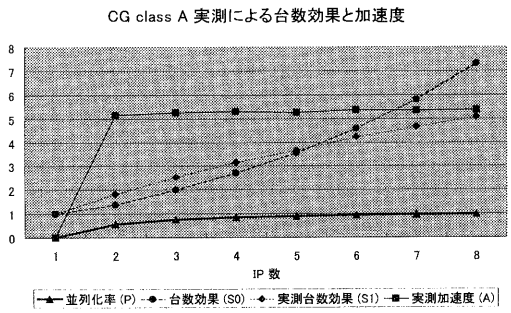

図 8 台数効果測定結果

グラムの一つである CG class A を用いて、スケジューラの性能測定を実施した結果について示す。CG は共役勾配法を用いて正値対称大規模疎行列の最小固有値の近似値を求めるプログラムである [4]。NPB には問題規模の大きさにより class A と class B とがあり、class A は問題の規模が小さい。

5.1 台数効果

図 8 は実測に基づく台数効果 (S1)、及び、実行加速度 (A) と、理想的な実行加速度による台数効果 (S0) を示す。実測による台数効果 S1 は、プログラムの正味の実行時間で比較している。ここで、正味の実行時間にはスレッド生成などの初期化処理は含まれていない。台数効果 S0 は実行加速度として、IP 数そのものを使用している。即ち、並列化可能部分は IP 数に比例して性能が向上するという仮定をおいている。

実際には、グラフ A が示すように実行加速度の値が 5 強と IP 数に依存しない一定の値になっているため、理想的な実行加速度を使用した台数効果 S0 と異なり IP 数 6 以上では台数効果が十分に出ていない。別な観点では、実行加速度はあるプログラムの並列化可能部分に対して、並列性をどの程度引き出せるかの尺度を示しており、CG の場合では、5 IP で十分性能が引き出せることを示している。

5.2 ジョブ多重度による実行時間の変化

図 6 は、ジョブ多重度を変化させた場合の、プログラム実行時間の変化を示す。測定環境は以下の三つの環境である。

(A) 時分割スケジューリング

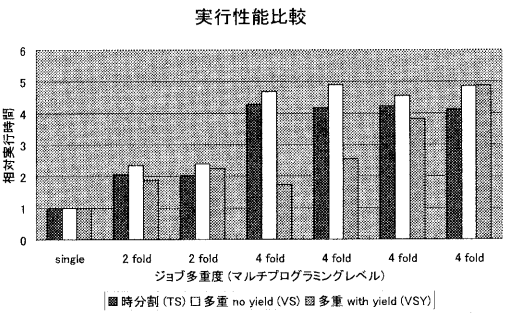

図 6 ジョブ多重度による実行時間の変化

(B) VIP 共有スケジューリング (譲渡命令なし)

(C) VIP 共有スケジューリング (譲渡命令あり)

8 IP 使用の時分割スケジューリングでジョブ多重度が 2 の場合 (グラフ TS)、理想的ケース (1 ジョブのケースを単純にスケールアップする) と比較して、実行時間の 0.9 ～ 2.8 % 増加がスケジューリングオーバヘッドとして見えている。4 多重実行の場合では、これが 3.0 ～ 7.0 % となる。

VIP 共有スケジューリングの場合は、測定プログラムが VIP 譲渡命令を発行する場合としない場合について測定した。VIP 譲渡命令を発行しない場合、実行時間の増分は理想的なケースと比較し、13.8 ～ 22.3 % となる。これに対し、VIP 譲渡命令を発行するオブジェクトを使用すると、逐次実行中の VIP を有効利用することができ、その結果ジョブ多重度が 2 の場合は、実行時間の 6.3 % 削減か、12.3 % 増加とばらつき、ジョブ多重度 4 の場合は、56.7 % 削減から 21.7 % 増加とばらつく。更にグラフより、VIP 譲渡命令ありの場合、ジョブ実行時間がジョブに対して単調増加するバッチスケジューリングに近くなっていることが分かる。

スケジューリングの違いによるプログラム実

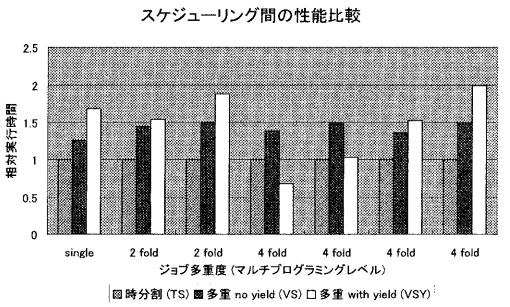

図 7 スケジューリングによる実行性能の差違

行性能を比較した結果は図 7 である。VIP 譲渡命令を挿入することにより、単独ジョブの実行時間が約 1.7 倍になってしまう。ばらつきが大きいため、多重スケジューリングと比較し、良いケースでは実行時間が 30 % 程削減できているが、悪い場合は実行時間が約 2 倍となってしまう。使用 IP 数が等しい、VIP 譲渡命令なしのケースと比較すると、4 多重時良い場合は実行時間が半分程になるが、悪い場合は 3 割強増加する。

5.3 考察

実測結果より、VIP 共有スケジューリングの性能に関して以下のことが分かる。

(1) 単体実行での性能が VIP 譲渡命令挿入により 30 % 強遅くなる。

(2) VIP 譲渡命令を用いたプログラムの多重実行時は、あるジョブの逐次実行中の VIP を有効利用することにより実行時間短縮が達成できる。

(3) 上記 (2) による効果はばらつきが大きく、フェアなスケジューリングができていない。

VIP 共有スケジューリングによる性能向上の要点は、並列実行終了時点でユーザプログラムがスケジューラに対して VIP の譲渡をヒントとして与えることによる。このヒント情報を与えるオーバヘッドは、カーネルコールのオーバヘッドとなるため、次に続く逐次実行部の処理時間が VIP 譲渡要求よりも長いという条件を満たす必要がある。従って、VIP 譲渡命令をいつ発行するかが問題となる。本来ならば、コンパイラの解析結果や、動的な実行状態の把握により、VIP 譲渡命令挿入位置を適切に定める必要があるが、現状のテストプログラムでは全ての並列実行後に VIP 譲渡命令を発行しているため、OS 側で VIP 譲渡頻度を緩和している。

ばらつきが大きいのは、VIP 譲渡頻度緩和により 実行形態が時分割よりもバッチ処理に近くなってしまったためである。従って、適切な VIP 譲渡命令発行をアプリケーションプログラムレベルで行えば、VIP 共有スケジューリングはプロセッサ利用率向上に対して十分に効果があることが期待できる。

6.結論

本稿では SR8000 におけるノード内スケジューリング方法として、ギャングスケジューリングをベースとした時分割スケジューリングと、VIP 共有スケジューリングについて概説し、各方法における実アプリケーションの性能測定結果を提示した。そこから、VIP 共有スケジューリングは潜在的にプロセッサ利用率を向上可能であることを示したが、そのためには適切な VIP 譲渡命令の発行が不可欠であることが分かった。

ギャングスケジューリング以外にも、プロセスコントロールやセルフスケジューリングのように、アプリケーションプログラムのプロセッサ負荷に応じて使用するプロセッサ数を変動し、プロセッサ利用率を向上するアプローチがある [3], [5]。これらの方式を適用するためには、プログラム自体がその方式に適応したコードを生成する必要があり、かつ、ユーザレベルのスレッドライブラリを使用する必要がある。これに対し COMPAS 機構では、ユーザプログラム性能向上のため必須なバリア同期機構をサポートする必要上、ギャングスケジューリングで時分割実行を行う必要がある。

今後の課題として、(1) VIP 共有スケジューリングによる性能向上のため、VIP 譲渡や VIP 要求のオーバヘッドの削減とより広範囲なアプリケーションプログラムによる詳細な性能解析の実施、(2) VIP 譲渡による性能低下を防ぐため、コンパイラの助けを借りた、VIP 譲渡命令の頻度削減、更に、(3) ノード間スケジューリングに関しては、ノード間ギャングスケジューリングなどによる時分割機構の導入、等を行う必要がある。

7.参考文献

[1] Feitelson, Dror G, "A Survey of Scheduling in Multiprogrammed Parallel Systems," Research Report RC 19790, IBM T.J. Watson Reasearch Center, Oct. 1994.

[2] Ousterhaut, J.K., "Scheduling Techniques for Concurrent Systems," In 3rd Intl. Conf. Distributed Comput. Syst., pp.22-30, Oct 1982.

[3] Shen, Kai, Tang, Hong, and Yang Tao, "Adaptive Two-level Thread Management for Fast MPI Execution on Shared Memory Machines," In Proceedings of Supercomputing'99, Nov. 1999.

[4] Bailey, D., Barszcs, E., and others, The NAS Parallel Benchmarks, http://science.nas.nasa.gov/Software/NPB/.

[5] Gupta, Anoop, Tucker, Andrew, and Stevens, Luis, "Making Effective Use of Shared-Memory Multiprocessors: The Process Control Approach," Stanford University Technical Report, Jul. 1991.