計算機アーキテクチャ 139－23
（2000．8．4）

MT版Paratoolによるマルチスレッド実行方式の評価

河場基行† 安里彰† 斎藤淳††
加納賢†† 深谷俊晴††

マルチスレッドプロセッサのマイクロアーキテクチャの検討を行なうため，**TaMTaM** シミュレータを作成した．このシミュレータはトレースドリブン方式かつ非サイクルベース方式の高速なシミュレーションを特長とする．**TaMTaM** シミュレータを用いて，命令フェッチ時のスレッド選択方式(icount 方式，ラウンドロビン方式，キャッシュミス方式)，命令フェッチ幅分割方式に関して，FFT プログラムを対象に比較検討を行なった．icount 方式とラウンドロビン方式の性能が良いが，これら 2 方式の性能差が 3.6%と僅かであること，また命令フェッチ幅分割は，2 つのスレッドより同数フェッチする方式で十分性能が得られることがわかった．

TaMTaM - Trace Analyzer for Multi-Threading Architecture Model

MOTOYUKI KAWABA,† AKIRA ASATO,† ATSUSHI SAITO,††
MASARU KANOU†† and TOSHIHARU FUKAYA††

We have developed a trace-driven simulator for multithreaded processors, called **TaMTaM**, in order to investigate the behavior of multi-threading applications. Because the **TaMTaM** is not a kind of cycle-based simulator, its feature allows us to evaluate faster than cycle-based simulators.

Through the **TaMTaM** simulation we studied the strategy of fetching instructions and partitioning fetch bandwidth. As for the FFT program derived from the SPASH2 suite, the icount method can improve the CPU throughput(IPC) the best among several strategies, while the round-robin method is comparable with the icount method. Also we have found the 50%-50% fetch bandwidth partitioning, which fetches the same number of instructions from 2 threads, achieves enough throughput gain.

1. はじめに

近年の著しい CPU の高速化と比較してメモリアクセス速度の向上率は低く，CPU とメモリとの速度格差が大きくなっている．このため，単一命令流のスーパスカラプロセッサでは，その実行時間の大部分がデータ待ちとなるケースが多く見られるようになってきた．このデータ待ち期間に空いている機能ユニットを十分利用できるだけの並列度は単一命令流にはなく，CPU の高速化によるアプリケーションの高速化が頭打ちになっている．

この状況を打開するためのアーキテクチャとして，複数のスレッドを 1CPU 上で同時に動かし高速な CPU を有効利用するプロセッサ (マルチスレッドプロセッサ) が脚光を浴びている．このプロセッサは，複数スレッドを 1CPU に投入することで命令のスループットを向上させる効果がある．このためメモリ系に対する負荷が大きい商用アプリケーションや科学技術計算への適用が期待される．

Tullsen らが Simultaneous Multithreading(SMT)[3] を発表して以来，マルチスレッドプロセッサの性能評価は実行ドリブン方式の性能評価シミュレータによるものが主であった．これは全くコンテキストが異なるスレッドを 1CPU コアで動作するために，実行トレースが採取が困難であることや，スレッド間の同期のためのメモリアクセスも含めて動的に実行命令列が変わることによって，トレースドリブンシミュレーション方式では誤差が大きくなりすぎてしまう可能性があるといった理由に因る．一方，実行ドリブン方式では，アプリケーションが実際に動作するためのリソース（特にメモリ）が必要であるため，アプリケーションの規模が大きいと評価ができないという問題がある．

われわれはこの問題に対処するために，バリア同期をサポートしたトレースドリブンシミュレーション方式のマルチスレッドプロセッサシミュレー

† (株) 富士通研究所
FUJITSU LABORATORIES LTD.
†† (株) 富士通ソーシアルサイエンスラボラトリ
FUJITSU SOCIAL SCIENCE LABORATORY LTD.

タ**TaMTaM**(Trace Analyzer for Multi-Threading Architecutre Model) を作成した．**TaMTaM** シミュレータは 1993 年に我々が開発した命令レベルトレースドリブンシミュレータ Paratool[1] をベースにしている．

本稿では**TaMTaM** シミュレータの仕組みとこれを用いたマルチスレッド実行方式の評価について述べるものである．

2. 対象アーキテクチャ(マルチスレッドプロセッサ) の概要

TaMTaM シミュレータは，SPARC 命令セットを持ったマルチスレッドプロセッサをシミュレートする．図 1 に対象アーキテクチャの構造を示す．レジスタファイルを複数セット持った SMT[3] と同様なアーキテクチャとなっている．(命令バッファ，リザベーションステーション，機能ユニット，キャッシュ等はスレッド間で共有される．)

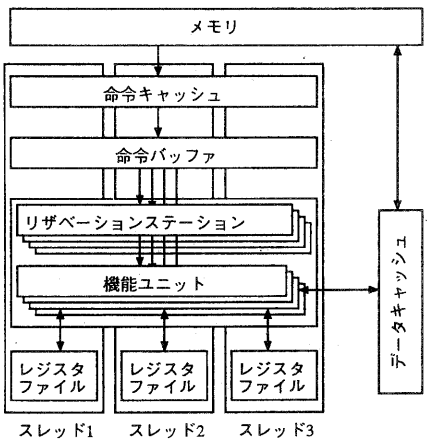

図 1　マルチスレッドプロセッサの構造

3. TaMTaM シミュレータの概要

3.1 特　　長

以下に特長を示す．

(1)　トレースドリブン方式

マルチスレッドプログラムの実行履歴 (トレース) に基づいたシミュレーションを行なう．

(2)　非サイクルベース方式

非サイクルベースとは，1 命令毎に処理タイミング (フェッチ，デコード，ディスパッチ，キャッシュアクセス，コミット) を見積もりながらシミュレーションを行なう方式である．この方式は CPU の各モジュールの状態を管理する必要がないためシミュレーション速度が高速であることが特長である．ところが非サイクルベースシミュレータでは，フェッチされる命令の順序が不変の場合は比較的精度良くシミュレーションできるが[2]，マルチスレッドプロセッサのようにフェッチされる命令順序が変ってしまう場合には精度が大きく落ちてしまう．このため**TaMTaM** シミュレータでは，後述するように命令キャッシュアクセスと命令バッファシミュレーション部に工夫を施している．

(3)　バリア同期シミュレーションのサポート

トレースデータにシミュレータ用同期命令を挿入することで，fork-join 型のスレッド間の同期をとることが可能である．

3.2 シミュレータの構成

図 2 に**TaMTaM** シミュレータの構成図を示す．

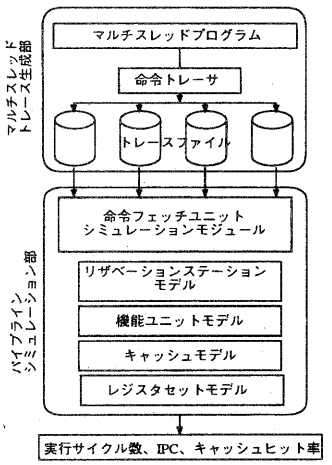

図 2　TaMTaM シミュレータの構成

TaMTaM シミュレータは以下の 2 つの処理部から構成されている．

- マルチスレッドトレース生成部
- パイプラインシミュレーション部

以下それぞれの処理部の説明を行なう．

3.2.1 マルチスレッドトレース生成部

SUN Microsystems によって開発された命令トレーサ**shade**[6] を用いて，マルチスレッドプログラムの実行履歴 (トレース) を採取しスレッド毎にトレースファイルを生成する．図 3 にトレースファイル作成手順を示す．

トレースファイルを生成するにあたり，ソースコードを複数タスク (スレッドに相当) に分割した後，以下に示すタスクの情報を付加しコンパイルする．(タスク分割は人手によって行なう．)

- タスクを実行するスレッド番号
- タスク開始命令
- タスク終了命令

タスク開始命令とタスク終了命令は，通常使われない

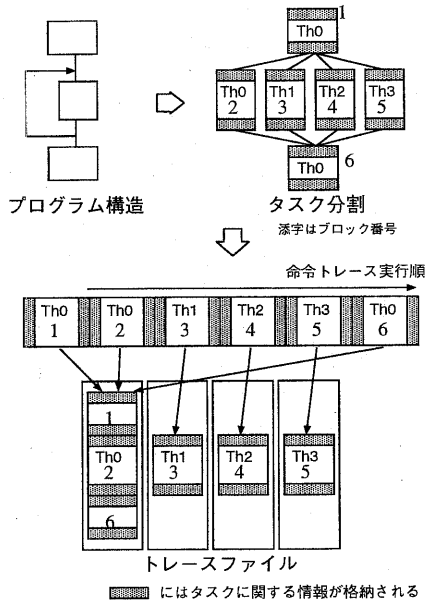

図 3 トレースファイル作成の手順

SPARC 命令を使用した.

タスク開始命令 or %g4,%g4,%g0

タスク終了命令 or %g5,%g5,%g0

命令トレーサはマルチスレッドプログラムをシーケンシャルに実行する. 図 3 のようにマルチスレッド実行をエミュレートしながら, トレースを採取し, ソースコードに付加されたタスク情報に基づいて, スレッド毎にトレースファイルを生成する. マルチスレッド用トレースファイルには以下の情報が格納される.

- 命令語, プログラムカウンタ
- メモリアクセスアドレス
- タスク開始命令, 終了命令

3.2.2 パイプラインシミュレート部

トレースファイル内のデータを解析し, 実行時間や IPC(Instruction Per Cycle) 等を報告する. マルチスレッドプロセッサシミュレータでは, とくにスレッド間のメモリアクセスタイミングと命令バッファへの命令投入のタイミングが問題になる. そこで以下の手順でトレースデータの読み出し及び処理を行ない, マルチスレッドの順序を正しくシミュレートする.

(1) 各スレッドのトレースファイルから 1 つずつ命令を読み出し, フェッチ開始可能なサイクルを見積もる.

(2) 最も早いサイクルでフェッチ可能なスレッドを選択する. このとき複数スレッド存在していた場合, 3.3 節で示す方式を用いて 1 スレッドを選択し, 命令キャッシュアクセスシミュレーションを行なう.

(3) 上記の処理で命令キャッシュミスを起こした命令は, フェッチ可能サイクルにキャッシュミスレイテンシを加算し (2) を実行する.

(4) 命令キャッシュヒットした命令は, 命令バッファ投入可能サイクルを見積もる. 命令フェッチした時点で命令バッファに空きが無かった場合, フェッチサイクルを命令バッファが空くサイクルにセットし (2) を実行する.

(5) フェッチ時に命令バッファに空きがあった場合, 以降の実行 (デコード, 実行, コミット) のタイミングを見積もる. タイミング見積を終えた後, 同一スレッドのトレースファイルより次命令を読み出し (2) を実行する.

処理の様子を図 4 に示す.

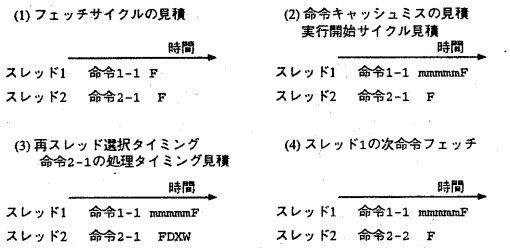

図 4 TaMTaM の処理方式

命令バッファはリングバッファとしてモデル化しており各バッファエントリが空くサイクルを保持している.

TaMTaM では命令バッファへ命令が投入されるタイミングと命令キャッシュアクセスを行なうタイミング以外は, 命令毎に処理タイミングを完全に見積もる. このためサイクルベースのシミュレータのように CPU の各モジュール状態を 1 サイクルずつ更新する作業がなく, 高速にマルチスレッドプロセッサのシミュレートを行なうことができる.

3.2.3 スレッド間の同期

各スレッドはトレースファイル中に埋め込まれたタスク開始命令とタスク終了命令をきっかけとして同期がとられる. これによりバリア同期をシミュレートする.

3.3 TaMTaM シミュレータのアーキテクチャパラメータ

機能ユニット数, レイテンシ, キャッシュサイズ, 各種バッファサイズの他に以下のようなマルチスレッドプロセッサアーキテクチャ設定が可能である.

- スレッド数 (1〜)
- スレッド選択方式 (icount 方式[5], キャッシュミス方式, ラウンドロビン方式)
- フェッチ幅分割方式 (1 方式, 1-1 方式, 2-1-1 方式, 1-1-1-1 方式)

スレッド選択方式は, 今までに提唱されている手法のうち以下の 3 つを選択することが可能である.

(1) icount 方式
スレッド毎にリザベーションステーション内の命令数をカウントし，この数が少ないスレッドから優先的にフェッチを行なう．機能ユニットを有効利用するためにストールしていないスレッドから命令をフェッチするという戦略に基づいている．

(2) キャッシュミス方式
スレッド毎にキャッシュミスをカウントし，この数が少ないスレッドから優先的にフェッチを行なう．キャッシュミスによるパイプラインストールを減らす事を主目的とする．

(3) ラウンドロビン方式
毎サイクル 1 スレッドのみをラウンドロビン方式で選択し命令をフェッチする．ただし命令キャッシュミスを起こしていないスレッドに関してのみ選択の対象とする．

フェッチ幅分割方式は，1 度の命令フェッチでフェッチする各スレッド毎のフェッチ幅の割合を指定する．「1 方式」は 1 スレッドのみから命令フェッチを行なう．「1-1 方式」は 2 スレッドから同じ数の命令をフェッチする．「2-1-1 方式」は 3 つのスレッドより，2:1:1 の割合で命令をフェッチする．また「1-1-1-1 方式」は 4 つのスレッドより同じ数だけ命令をフェッチする．

3.4 TaMTaM シミュレータの出力

TaMTaM シミュレータはスレッド毎の実行サイクル数，実行命令数，キャッシュミス率，分岐予測正解率等を報告する．図 4 にあるようなパイプライン図表示機能を持つ．

4. TaMTaM シミュレータによるマイクロアーキテクチャの比較検討

TaMTaM シミュレータを用いてマルチスレッドプロセッサのマイクロアーキテクチャの比較検討を行なった．ターゲットプログラムは SPLASH2[7] ベンチマークの中から FFT を使用した．とくにここでは以下の点について検討を行なった．

(1) 命令フェッチ方式
(2) 命令フェッチ分割方式

シミュレーションに使用したアーキテクチャパラメータを表 1 のように設定した．

4.1 命令フェッチ方式とととスループット向上率

icount 方式，ラウンドロビン方式，キャッシュミス方式の 3 つの命令フェッチ方式のスループット向上率を測定した．スループットの指標として IPC(Instruction per cycle) を用いた．命令フェッチ幅分割方式は 1-1 方式を用いた．

図 5 に結果を示す．また図 6 に，1 サイクル中に機能ユニットに投入された命令数の頻度グラフを示す．ここで使用したアーキテクチャモデルの機能ユニット数が 16 であるため，図 6 の横軸 (1 サイクル中に機能ユニットに投入された命令数) の最大値は 16 となっている．

表 1 アーキテクチャパラメータ

機能ユニット	個数	レイテンシ
整数 ALU	4	1
ロード/ストア	4	2
浮動小数点加算	3	3
浮動小数点乗算	3	4
浮動小数点除算		21
分岐ユニット	2	2
命令フェッチ幅		8 命令
命令デコード幅		8 命令
L1 命令キャッシュ 32K バイト 2Way		
L1 データキャッシュ 32K バイト 2Way		
L2 16MB バイト Direct Map		
L1 ミスレイテンシ		6cycle
L2 ミスレイテンシ		150cycle

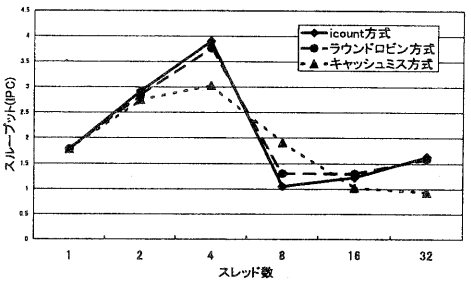

図 5 スレッド選択方式とスレッド数による速度差

図 5 よりスレッド数 4 までは，icount 方式が最も高いスループットが得られている．しかしながらラウンドロビン方式とのスループット差は小さく最大 3.6% 程度である．

図 6 にある機能ユニットに投入された命令数が 0 の値は，垂直方向の無駄 (vertical waste[3]) に相当する．これを比較すると，キャッシュミス方式により，他の方式の方が垂直方向の無駄を省いていることがわかる．キャッシュミス方式ではキャッシュミスによる垂直方向の無駄を省くことに効果があるが，演算レイテンシによって発生したものに関してはあまり効果がない．今回シミュレーションを行なった FFT プログラムのようにレイテンシが複数サイクルに渡る浮動小数点演算が多いものに関しては，icount 方式やラウンドロビン方式の方が効果があると考えられる．

全体的にスレッド数 4 までは複数スレッドを投入することによる IPC 向上が認められるものの，それより大きくなると一旦 IPC が劣化する．ところがさら

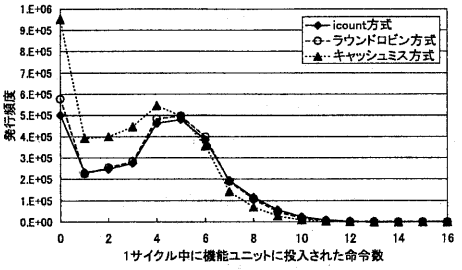

図 6　機能ユニットに投入された命令数 (4 スレッド実行)

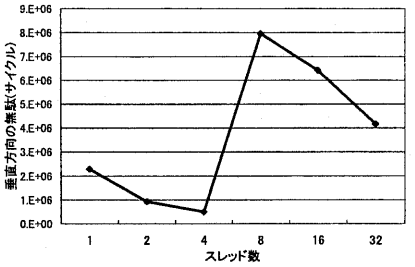

図 8　スレッド数増加による垂直方向の無駄の変化 (icount 方式)

にスレッド数を増やすと IPC が向上し始める．これは 2 次キャッシュミスと関連がある．図 7 に 2 次キャッシュミス率を示す．

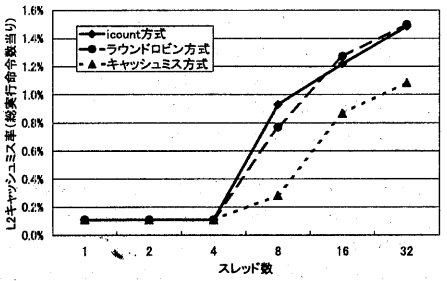

図 7　スレッド数増加による 2 次キャッシュミス増加

同時に実行されるスレッド数が多くなると，複数スレッドによって同時にアクセスされるメモリ領域が増え，キャッシュミスが多くなり IPC が低下する．これにより垂直方向の無駄が増える．ところがさらに多くのスレッド数を投入することで，機能ユニットの使用率が向上し，IPC が向上するものと考えられる．図 8 にスレッド数増加による垂直方向の無駄の変化を示す．

4 スレッドから 8 スレッドにかけて垂直方向の無駄が激増し，以降 32 スレッドにかけて徐々に現象していることがわかる．

一方キャッシュミス方式では，他の 2 つの方式と比較してキャッシュミス率の増加が緩やかである．これはキャッシュミス方式では，他の方式に比べて スレッドを切り替える機会が少なく，同時にアクセスされるメモリ領域の増加が少ないためと考えられる．このためキャッシュミス方式では，多くのスレッドを投入したことによるスループット低下が小さくなると推測される．

4.2　命令フェッチ幅分割方式による性能影響

命令フェッチ幅分割方式による性能の変化を調べた．命令フェッチ幅分割のバリエーションは 3.3 で示した 4 つ (1 方式,1-1 方式,2-1-1 方式,1-1-1 方式) である．ここでは 4 スレッド /8 スレッド時の性能についてシミュレーションを行なった．結果を図 9 に示す．　4 スレッド時には全体的に多くのスレッドより命令をフェッチした方がスループットが高くなる傾向が見られる．とくに 1 方式 (ラウンドロビン方式) から 1-1 方式にしたときに性能向上が大きい (7.9% スループットが向上)．図 10 は分割方式による機能ユニットに投入される命令数の影響を示したものである．1 スレッドのみからフェッチする 1 方式に比べて，他の方式は垂直方向の無駄がやや増えているものの，グラフが全体的に右へシフトしており，1 サイクル中により多くの命令が機能ユニットに投入されていることがわかる．

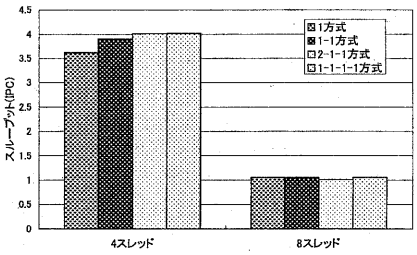

図 9　命令フェッチ分割方式によるスループット差

これは独立な命令流であるスレッドから命令をフェッチすることで並列実行できる命令数が増加するためであると考えられる．また今回使用した FFT プログラムは SIMD 型のマルチスレッドプログラムであるた

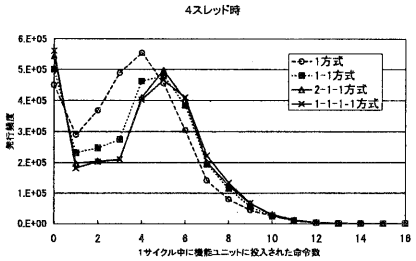

図 10 命令フェッチ分割方式による命令発行数の分布差 (4 スレッド)

め，スレッド数を増やしても実行されるコードサイズが大きくならず，命令キャッシュミスに変化がないことも要因の一つである．

一方 L2 キャッシュに対して過負荷な状態になっている 8 スレッド実行時には，命令フェッチ幅分割方式による差はほとんどない．これは図 8 に見られるように垂直方向の無駄がもともと多いため，スループットに対する影響が小さいものと考えられる．

5. 考　　察

5.1 TaMTaM シミュレータについて

マルチスレッドプロセッサのシミュレーション方式として，トレースドリブン方式を採用した．また非サイクルベースシミュレーション方式を用いることで，高々スレッド数個分の命令を管理すればよく，高速に処理が可能である．これによって UltraSPARCII 300MHz のマシンで 100K 命令/秒 程度のシミュレーションが可能になった．またバリア同期シミュレーションもサポートしており，とくに大規模な数値計算プログラムを評価するツールとして有効であると言える．

この事は実行ドリブン方式によるマルチスレッドプロセッサ評価を否定するものではない．シミュレーションをする対象アプリケーションによって適用する方式を変更するべきであり，実行ドリブン方式とトレースドリブン方式の境界領域で，シミュレーション精度の検証を行なうことが重要であると考える．

5.2 FFT プログラム評価について

今回は命令フェッチ方式，命令フェッチ幅分割方式についてシミュレーションを行なった．結果 icount 方式が最も高いスループットが得られたが，ラウンドロビン方式とほとんど差がなかった．また命令フェッチ幅分割方式では小さく分割しフェッチした方がスループットが高かった．いずれも FFT プログラムが持つ特徴 (数値計算かつ SIMD 型) に大きく依存する．マルチスレッドプロセッサの総合的な評価を行なうために，処理が非対称なアプリケーションもターゲット含め評価を進める必要がある．

6. ま と め

マルチスレッドプロセッサアーキテクチャの性能評価を行なうため，**TaMTaM** シミュレータを作成した．このシミュレータにより，単一命令流のスーパスカラプロセッサの挙動とは異なるマルチスレッドプロセッサの挙動を測定した．今後本シミュレータのシミュレーション精度検証が重要な課題として挙げられる．またマルチスレッドプロマルチスレッドプロセッサをプロセッサエレメントに持つマルチプロセッサ評価システムへの拡張も予定している．

参 考 文 献

1) 志村 浩也 他.「スーパスカラプロセッサの性能評価 - Paratool - 」, 情報処理学会研究会報告 93-ARC-102-1, Oct. 1993
2) 河場, 木村.「UltraSPARC 命令レベルシミュレータ UltraSim」, 第 55 回情報処理学会全国大会 (平成 9 年後期), 2F-2, Sept. 1993
3) Dean M.Tullsen, Susan J.Eggers and Henry M.Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", Proc. of th e22nd Annual Intl. Symp. on Computer Architecture, June 1995.
4) Jack L.Lo, Luiz Andre Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh, "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors", Proc of the 25 th annual Intl. Symp. on Computer Architecture, June 1998.
5) D.M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J.L.Lo, and R.L.Stamm, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In Proc. of 23rd Annual Int. Symp. o Computer Architecture, May 1996.
6) "Shade and Spixtool", Available at `http://www. sun.com/microelectronics/shade/`
7) "Stanford Parallel Application for Shared Memory", Available at `http://www-flash.stanford.edu/apps/SPLASH`