3. 実例 1

Niagara: 32 ウェイマルチスレッド SPARC プロセッサ

NIAGARA: A 32-Way Multithreaded SPARC Processor

Poonacha Kongetira	Sun Microsystems poonacha.kongetira@sun.com
Kathirgamar Aingaran	Sun Microsystems
Kunle Olukotun	Stanford University
翻訳：坂井 修一	東京大学大学院情報理工学系研究科 sakai@mtl.t.u-tokyo.ac.jp
五島 正裕	東京大学大学院情報理工学系研究科 goshima@mtl.t.u-tokyo.ac.jp

[原文] Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun: NIAGARA: A 32-Way Multithreaded SPARC Processor, IEEE Micro, Vol.25, No.2, pp.21-29 (Mar. -Apr. 2005) より許可を得て翻訳.

Niagara プロセッサはスレッドリッチなアーキテクチャを採用し，商用サーバ向けアプリケーションのための高性能ソリューションを提供する．このハードウェアでは，チップ内クロスバー，2 次キャッシュ，メモリ制御装置が高度に統合された構成で 32 個のスレッドを実行することができ，サーバアプリケーションの持つスレッドレベル並列性を活用してスループットを高め，同時に消費電力を低減する．

はじめに

過去 20 年以上にわたり，マイクロプロセッサの設計者は，周波数の向上や命令レベル並列性（ILP）の活用によって，デスクトップ処理環境における単一スレッド性能の改善に注力してきた．このうち，後者の具体的な技術が複数命令同時発行，アウトオブオーダの発行，投機的な分岐予測などである．これまで単一スレッドの性能ばかりを追求してきた結果，主記憶に対するレイテンシの限界やアプリケーションに内在する ILP の低さのため，性能が頭打ちとなっている．また，単一スレッドの高性能化によって，マイクロプロセッサの設計は爆発的に複雑になり，さらに消費電力が大きな問題となってきた．

こうした背景のもと，サンマイクロシステムズの Niagara プロセッサは，これまでとは異なる斬新なアプローチをマイクロプロセッサのデザインに採用することとなった．すなわち，単一スレッドやたかだか 2 つのスレッドの性能だけを追求するのではなく，商用サーバ環境におけるマルチスレッド性能を考慮してアーキテクチャ最適化を図っている．このアプローチは，全体スループットの向上，すなわち多くのスレッドにまたがる処理スループットを増やすことによって，性能の向上を図るものである．これは，データベース [1] や Web サービス [2] など，大きなスレッドレベル並列性（TLP, Thread Level Parallelism）のある商用サーバのアプリケーションで，特に効果的となる．

本稿では，Niagara プロセッサのアーキテクチャについて解説する．これは，SPARC V9 抽象アーキテクチャの最新の実装である．SPARC V9 抽象アーキテクチャは，高スループットの実現のために，大規模なオンチップ並列処理を行う．Niagara は，チップマルチプロセッサ [3] と細粒度マルチスレッディング [4] を組み合わせることにより，32 のハードウェアスレッドを同時動作させることができる．別の研究 [5] は，マルチスレッドのワークロードに対して本方式を採用することにより，性能が飛躍的に改善すると指摘している．多数のスレッドの並列実行により，メモリレイテンシも効果的に隠蔽される．一方で，32 のスレッドを利用することは，メモリシステムに対して高いバンド幅を要求することになる．この

ベンチマーク	アプリケーションの種類	命令レベル並列度	スレッドレベル並列度	ワーキングセット	ワーキングセット
Web99	Web サーバ Java アプリケーション	低	高	大	低
JBB	サーバ	低	高	大	中
TPC-C	トランザクション処理	低	高	中	中
SAP-2T	ERP	中	高	中	中
SAP-3T	ERP	低	高	大	高
TPC-H	意思決定システム	高	高	大	中

表-1　商用サーバアプリケーション

バンド幅を得るため，メモリ参照は全スレッドが共有するバンク構造のオンチップ 2 次キャッシュをクロスバー接続することで行うことにした．また 4 つの独立したオンチップメモリコントローラによって，メモリは 20 GB/s を超えるバンド幅を確保している．

TLP を利用することにより，CPU のクロック周波数の限界に挑むことなく，性能を大幅に改善することができる．TLP に加え，複数のスレッド間で CPU パイプラインを共有することによって，面積の効率的な利用や電力節約の面で優れた設計が可能になる．Niagara の消費電力は約 60 W 程度と見積もられているが，この数字は，計算集約的な環境では大変魅力的である．たとえば，データセンタでは電力や空調のコストが重大問題となっている．そこでは，電力供給の限界を超えることが原因で，サーバ群をラックいっぱいに詰め込むことができなくなることがよく起こる．

我々は，Niagara を Solaris オペレーティングシステムが動作するように設計した．そのため，既存の Solaris アプリケーションは，修正することなく Niagara システム上で処理することができる．アプリケーションソフトウェアに対しては，Niagara プロセッサは，32 個の単独のプロセッサに見える．すなわち，OS のレイヤでは，スレッドがハードウェアを共有していることは隠されてしまう．現在，対称型マッププロセッサ（SMP）システムで実行されている多くのマルチスレッドアプリケーションに対して，Niagara はさらに高い性能を出すことが望まれる． Sun で以前開発されたマルチスレッドプロセッサ [6), 7)] や，現在，研究所で導入テスト段階にある Niagara チップ／システムの事例は，この高い性能を保証している．

近年，多くの小売業やビジネスが Web に参入したため，商用サーバ向けアプリケーションが増大する傾向が見られる（**表 -1**）．こうしたサーバアプリケーションでは，数多くのクライアントからの要求があるため，「要求レベルの並列性」が高い．その結果として，マルチスレッド型のサーバは，この並列性を抽出活用して高性能を得ることができる．このようなアプリケーションを処理するサーバの性能指標としては，「定常的にクライアント要求を受け付けるスループット」が最も適当である．

さらに，サーバが発展するとデータセンタのような計算集約的なものになる．そこでは，サーバに電力を供給するコストとサーバから熱を逃がすコストが，センタの運用経費の中で大きなものとなる．Google によると，ラックに入ったクラスタサーバは 1 平方フィートあたり 400 ～ 700 W の電力密度が必要になるということである [2)]．これは，商用データセンタで普通に供給される 1 平方フィートあたり 70 ～ 150 W という電力密度 [2)] をはるかに上回る．現在のサーバクラスタに使われている ILP プロセッサのクロック周波数を下げれば消費電力も下がるが，これに比例して性能も下がってしまうため，この方法はありがたくない．すなわち，商用サーバで単位ワットあたりの性能を改善することが必要である．単一スレッド性能を最適化したマシンでは，効率的に単位ワットあたりの性能をあげることができない．

商用サーバアプリケーションは，ワーキングセットが大きく，メモリ参照の局所性があまりないため，キャッシュミス率があがって，ILP は低くなる傾向にある．これに加えてデータ依存のある分岐は予測がしにくく，プロセッサは予測のはずれたパスの実行を全部破棄しなければならない．ロード命令間の依存性も問題になるが，これは，命令発行時にハードウェアで検出することができず，実行してから破棄することになる．有効利用できる ILP が低いこととキャッシュミス率が高いことから，メモリアクセス時間が性能のボトルネックとなってしまう．結果として，**図 - 1** に示すとおり，単一命令発行プロセッサに対して複雑な ILP プロセッサは，性能上さほど有利なものではなくなってしまう．一方で，ILP プロセッサは，電力と複雑さの点で多大の犠牲を払っている．

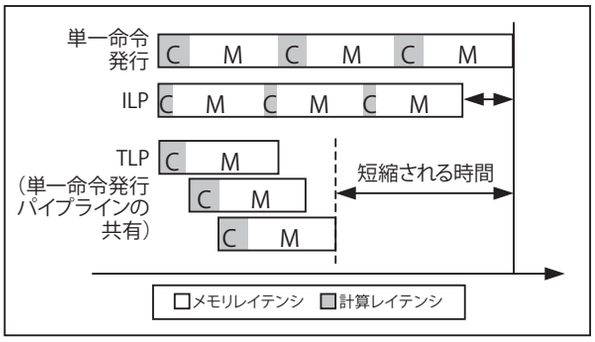

図-1 商用サーバ用にTLPとILPを最適化したプロセッサの動作: 単一命令発行マシンに比べ, ILPプロセッサでは主に計算時間が短縮される. その結果, メモリアクセス時間がアプリケーションの性能を決める. TLPの場合は, 複数のスレッドが単一命令発行パイプラインを共有し, スレッド間で実行のオーバーラップが起こって, マルチスレッドアプリケーションの性能が向上する.

サーバアプリケーションには，巨大な TLP を持つものが多い．したがって，TLP を有効利用できるマシン，すなわち単一スレッドプロセッサ群とコヒーレンスのための相互接続機能からなる共有メモリマシンが，高い性能をあげることができる．しかし，ILP の利用を中心に設計されたプロセッサからなる SMP を使った場合，電力性能比も悪く，コスト性能比も悪いものになる．それよりも，ダイの上に単純なプロセッサコアと共有キャッシュを載せ，チップ外の大きなメモリへのバンド幅を確保してコンピュータを作るほうが効率的である．すなわち，1 つのチップ上に SMP サーバ全体を作り込む，ということである．これには，副産物として，コア間の通信のレイテンシが短くなって，商用サーバアプリケーションで効率的にデータ共有をすることができるようになる効果もある．

Niagara の概要

Niagara は，商用サーバアプリケーションのスループットを高めることによって，プロセッサが処理するスレッド数を大幅に増やし，高いバンド幅に見合ったメモリサブシステムを提供する．Niagara のハードウェアでは，32 スレッドが実行される．本アーキテクチャでは，4 つのスレッドがひとまとまりになってスレッドグループを作る．スレッドグループは，「SPARC パイプ」と呼ばれるパイプラインを共有する．Niagara は 8 個のスレッドグループを処理するため，CPU 全体で 32 スレッドが走ることになる．各 SPARC パイプには，1 次命令キャッシュと 1 次データキャッシュが含まれる．あるスレッドでメモリストールやパイプラインストールを起こす場合でも，同じスレッドグループの他のスレッドにペナルティなしで切り替えることで，オーバーヘッドは発生しない．図 -1 は，共有パイプラインを使い回すことで高いスループットが達成されることを示している．

32 個のスレッドは，3 MB の 2 次キャッシュを共有する．2 次キャッシュは，4 ウェイのバンク構造をとり，アクセスをパイプライン化してバンド幅をあげている．これは，12 ウェイのセットアソシアティブで，多くのスレッドからのアクセスによる競合ミスを減らしている．商用サーバ用コードでは，データ共有が起こり，コヒーレンスミス率が高くなりがちである．従来の SMP では，独立したプロセッサをコヒーレント制御つきで相互接続しており，コヒーレンスミスの信号が，プロセッサ内部に比べて動作の遅いオフチップバスやリンクに出ていき，結果としてレイテンシが大きくなってしまっていた．Niagara では，これらのミスを起こらなくし，かわりに低レイテンシの共有キャッシュの通信に置き換えている．

クロスバー接続によって，SPARC パイプ，2 次キャッシュバンク，その他 CPU の共有資源間で通信リンクが張られる．クロスバー接続のバンド幅は 200GB/s を超える．通信のソース／デスティネーションの組ごとに，2 つの入り口を持つキューがある．キューは，クロスバーのそれぞれの向きに 96 個のトランザクションを入れることができる．クロスバーは，I/O サブシステムとの通信用のポートも提供する．デスティネーションポートが競合した場合の調停には，単純な古い順優先の方式が採られており，これによってすべての接続要求への公平なスケジューリングを保証している．さらに，クロスバーは，Niagara のメモリアクセスに順序づけをしている．

メモリインタフェースは，デュアルデータレート 2 (DDR2) DRAM 用の 4 チャネルであり，20 GB/s を超えるスループットを提供し，128 GB までの容量に対応する．**図 -2** に Niagara プロセッサのブロック図を示す．

SPARC パイプライン

以下で，SPARC パイプの実装について述べる．先に述べた通り，このパイプは，4 つのスレッドを同時に処理する．各スレッドは，独自にレジスタ群，命令群，ストアバッファを持つ．スレッドグループの中では，1 次キャッシュ，トランスレーションルックアサイドバッファ（TLB），実行ユニットとほとんどのパイプラインレジスタが共有される．単一命令発行パイプラインは，6 ステージ（フェッチ，スレッドの選択，デコード，実行，メモリ，およびライトバックの各ステージ）からなる．

フェッチステージでは，命令キャッシュと命令 TLB

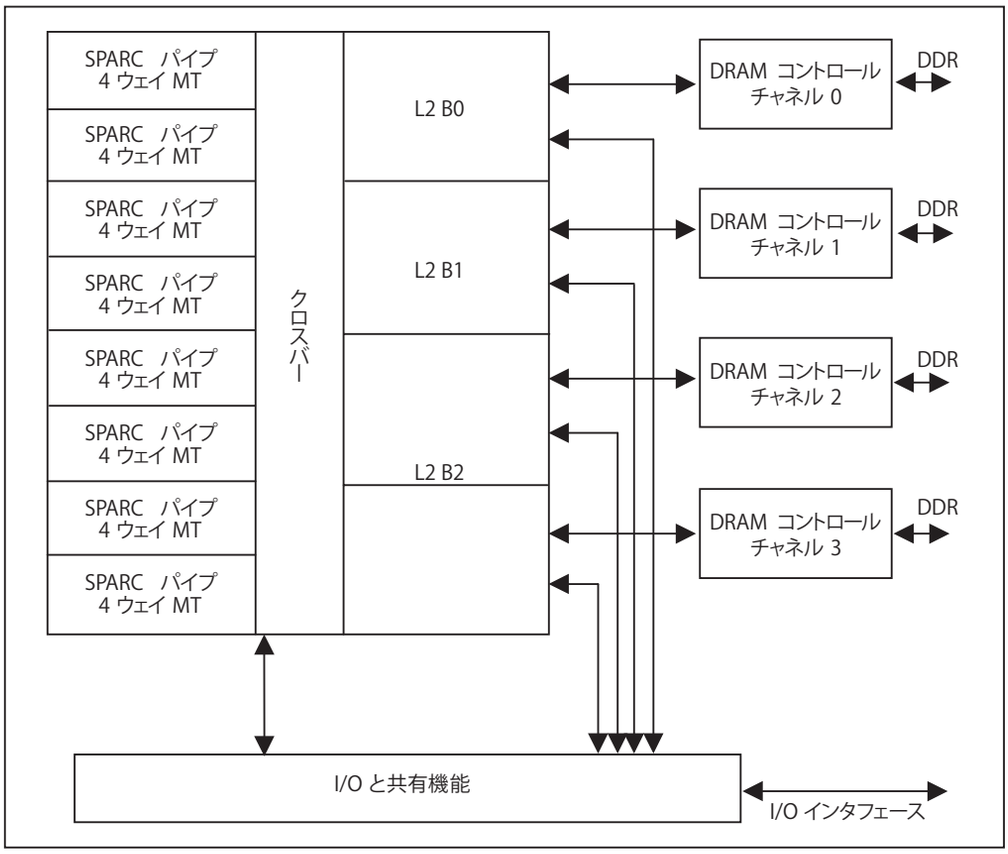

図-2　Niagaraブロック図

(ITLB) がアクセスされる．次のステージでは，ウェイを選ぶことでキャッシュアクセスが完了する．ここでのクリティカルパスは，64 エントリのフルアソシアティブ ITLB のアクセスにある．スレッド選択マルチプレクサが，4 つのスレッドのプログラムカウンタ（PC）のうち，どれでアクセスするかを決める．本パイプラインでは，1 クロックサイクルで 2 つの命令をフェッチする．キャッシュの中のプレデコードビットは，長いレイテンシがかかる命令を示す．

スレッド選択ステージでは，スレッド選択マルチプレクサが処理可能なスレッドを 1 つ選んで，命令を発行し，以後のステージに送り込む．このステージでは，命令バッファの管理運用も行われている．フェッチステージでキャッシュからフェッチされた命令は，以後のステージでまだ使えない場合に，その命令の含まれるスレッドの命令バッファに入れておく．最初の 2 つのステージのパイプラインレジスタは，スレッドごとに別のものが用意されている．

選ばれたスレッドの命令は，デコードステージに移るが，このステージでは，命令デコードとレジスタファイルアクセスが行われる．実行ユニットには，算術論理演算ユニット（ALU），シフタ，乗算器，除算器などが入っている．バイパスユニットは，実行中の命令がレジスタファイルを更新する前に，これに依存関係のある命令に実行結果を渡す．ALU とシフト命令のレイテンシは 1 クロックで，実行ステージで結果が生成される．乗算と除算のレイテンシは長く，スレッドスイッチを引き起こす．

ロードストアユニットは，データ TLB（DTLB），データキャッシュ，ストアバッファなどからなる．DTLB とデータキャッシュのアクセスは，メモリステージで行われる．フェッチステージで ITLB アクセスがクリティカルパスであったのと同様に，64 エントリのフルアソシアティブ DTLB アクセスがクリティカルパスとなる．ロードストアユニットは，スレッドごとに 8 エントリのストアバッファを 1 本ずつ，計 4 本を持っている．ストアバッファ中のタグをチェックすることで，ロードとストアの間にリードアフターライト（RAW）ハザードを検出することができる．ストアバッファは，データをロード命令にバイパスして RAW ハザードを解決する．ストアバッファのタグのチェックは，TLB アクセスの後，ライトバックステージの中の早いタイミングで行われる．ロードデータは，ライトバックステージの中の遅いタイミングで，依存関係のある命令間でバイパスする形で渡される．ADD などの 1 サイクル命令は，ライトバックステージでレジスタファイルを更新する．

スレッド選択ロジックは，フェッチステージとスレッド選択ステージにおいて，与えられたサイクルでどのス

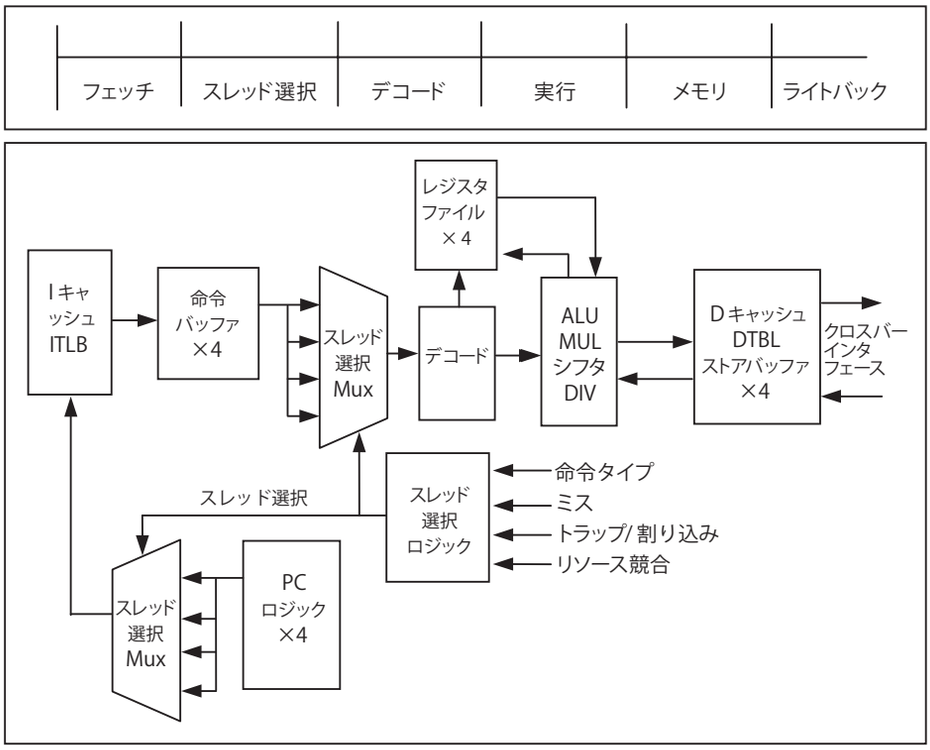

図-3　SPARCパイプラインのブロック図：4つのスレッドが，ローカルな命令の実行機構とデータキャッシュの入った6ステージの単一命令発行パイプラインを共有する．マシンの他の部分との通信は，クロスバーインタフェースを介して行われる．

レッドがアクティブかを決定する．**図 -3** に示すように，スレッド選択の信号は，フェッチステージとスレッド選択ステージで共通である．そのため，スレッド選択ステージでスレッドが選択されてデコードステージに送る命令が発行されると，フェッチステージでも同じ命令によってキャッシュアクセスが起こる．スレッド選択ロジックでは，さまざまなパイプラインステージの情報を利用して，あるスレッドをいつ選択し，いつ選択解除するかを決める．たとえば，スレッド選択ステージでは，命令キャッシュの中のプレデコードビットを参照して命令タイプを決めることができるが，いくつかのトラップはそれ以後のパイプラインステージにならないと検出できない．すなわち，命令タイプ信号はスレッド選択ステージでスレッドを選択解除できるが，メモリステージで起こされるトラップは，トラップ処理の間に当該スレッドのそれ以後の命令をすべてフラッシュし，自分自身を選択解除することが必要となる．

パイプラインインターロックとスケジューリング

ADD（加算）のように 1 サイクルで実行できる命令に対しては，Niagara は，同一スレッドの先行命令からの完全なバイパスを実装しており，これによって RAW 依存は解消される．前述したように，ロード命令のレイテンシ，すなわち，ロード命令の結果が後続の命令によって利用可能になるまでのサイクル数は 3 である．このような長いレイテンシを持つ命令はパイプラインハザードを引き起こし，ハザードが解消されるまで当該スレッドをストールさせる必要がある．したがってロード命令の場合には，同一スレッドの後続の命令は，ハザードが解消されるまで 2 サイクル待つことになる．

マルチスレッドパイプラインでは，同一の資源を共有するスレッド間で構造ハザードも発生する．1 命令／サイクルのスループットを持つ ALU のような資源に対しては構造ハザードが起こらないが，1 命令／サイクルより低いスループットを持つ除算器では，新たなスケジューリングが必要となる．そのような場合，DIV（除算）命令を実行するスレッドは，除算器が空くまで待つ必要がある．スレッドスケジューラは，実行が中断してから一番長い時間が経ったスレッド（LRU スレッド）に対して高い優先順位を与えることによって公平性を保証する．なお，除算器が使用されているときにも，他のスレッドは ALU やロードストアユニットなどの空いている資源を使用することができる．

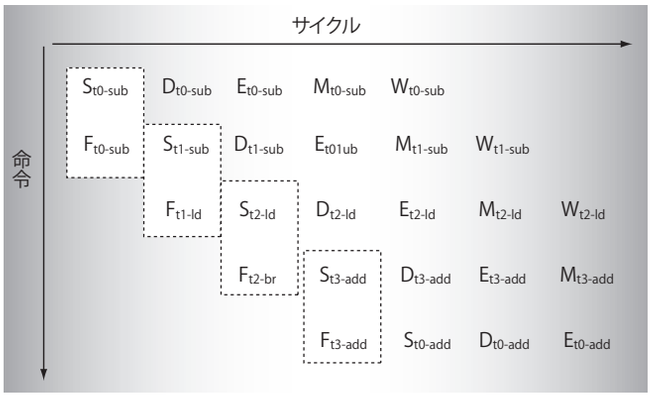

図-4　スレッド選択：すべてのスレッドが利用可能な場合

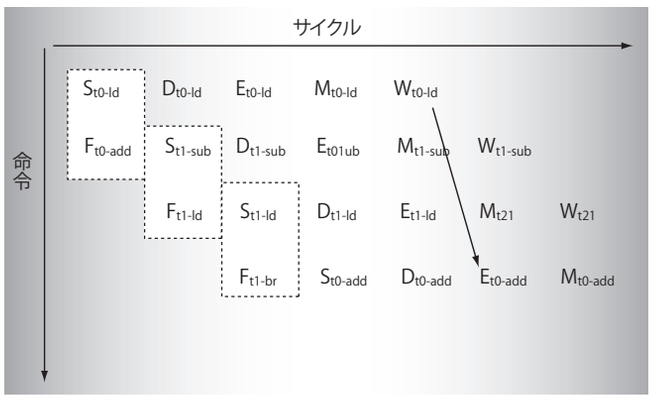

図-5　スレッド選択：2 つのスレッドのみが利用可能な場合．ロード命令のヒット／ミスが決定する前に，thread0 のADD命令が投機的にパイプラインに投入される．

スレッド選択のポリシー

スレッド選択ポリシーは，LRU スレッドに優先権を与えた上で，毎サイクル，利用可能なスレッド間で切り替えを行うというものである．スレッドは，ロード，分岐，乗除算などの長レイテンシ命令によって，また，キャッシュミス，トラップ，資源競合のために発生したパイプラインストールによって，利用不可になる．スレッドスケジューラは，ロード命令がキャッシュにヒットすると想定して，それに依存する命令を投機的にスケジューリングする．そのような投機的なスレッドには，投機的でないスレッドより低い優先順位を与える．

図 -4 に，すべてのスレッドが利用可能であるときのパイプラインの動作の様子を示す．図中，1 つの命令のパイプライン処理は，左から右に進む．命令キャッシュからパイプラインに新たにフェッチされた命令は，上から下へ順番に登場する．また，$S_{t0\text{-}sub}$ という記述は，*thread0* の減算（SUB）命令がパイプラインの S ステージにあることを表している．図の例では，最初のサイクルで，t0-sub が選択され，発行されている．次のサイクルでは *thread1* の命令 t1-sub が，その次のサイクルでは *thread2* の t2-ld が選択されている．さらにその次のサイクルでは *thread3* の t3-add が選択され，すべてのスレッドが 1 回ずつ選択されたことになる．したがって次のサイクルには，LRU スレッドである *thread0* の命令 t0-add が選択されている．なお，スレッド選択ステージとフェッチステージでは，同じスレッドが選択されていることに注意されたい．

図 -5 は，2 つのスレッドだけが利用可能である状況を示している．この例では，*thread0* と *thread1* は利用可能だが，*thread2* と *thread3* は利用不可である．最初のサイクルでスレッド選択ステージにいる t0-ld 命令は，長レイテンシの命令であるため，*thread0* は選択対象から外される．ただし，t0-ld 命令自体は発行される．2 番目のサイクルでは，*thread1* が利用可能であるため，スケジューラはこれに切り替える．その次のサイクルでは，*thread0* が選択対象から外されているため，*thread1* のほかに利用可能なスレッドはない．ここで，前のサイクルに選択された t1-sub は 1 サイクルの命令なので，*thread1* は次のサイクルにも選択することができる．次のサイクルでは，*thread0* は投機的に実行可能なので選択することができる．最初の t0-ld がヒットなら，データはバイパスを通して t0-add に渡される．ミスなら，t0-add はフラッシュされ，2 次キャッシュからデータが戻ってきた時再発行されることになる．

整数レジスタファイル

整数レジスタファイルは，リードポート 3 個とライトポート 2 個を持つ．リードポートは，通常の単一命令発行のマシンのそれと同じものである．3 つ目のリードポートは，ストア命令をはじめとするいくつかの 3 オペランド命令のために用意されている．2 本のライトポートは，それぞれ，1 サイクルの命令と，長レイテンシの命令によって使い分けられる．複数の長レイテンシ命令（ロード，および，乗除算命令）は同時に結果を生成し得るので，それらの命令は調停される． SPARC V9 アーキテクチャでは，**図 -6** に示すようなレジスタウィンドウが定義されている．1 つのウィンドウは，それぞれ 8 本の，*in*，*local*，*out* レジスタ群からなる．スレッドからは同時にそれらすべてを読み書きできる．あるウィンドウの *out* レジスタは，次のウィンドウの *in* レジ

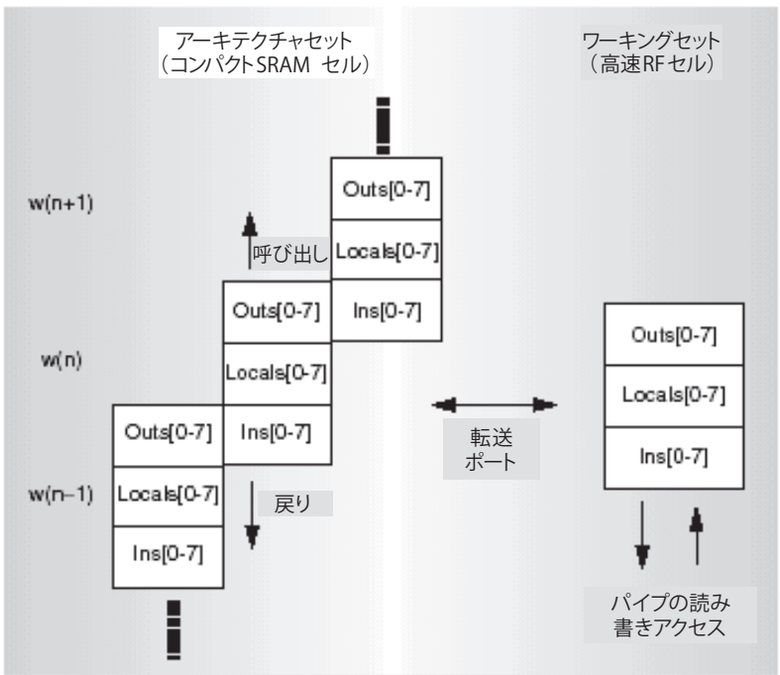

図-6　ウィンドウ化されたレジスタファイル：W(n-1), w(n), およびw(n+1)の各ウィンドウはこの順で隣りあっている．ウィンドウ変更イベントによって，当該ワーキングセットと新(あるいは旧)のウィンドウの間でデータ転送が起きる．そのとき，パイプラインアクセスがワーキングセットの中で行われる．この構造は，4つのスレッドに対してそれぞれ独立に作られている．

スタとしてアクセスされるが，それらは物理的には同じレジスタである．1つのスレッドは8つのウィンドウを持っており，4つのスレッドのそれぞれに対してこのようなレジスタセットが用意されている．したがって，レジスタファイルは全部で640本の64ビットレジスタを持ち，その容量は5.7KBにもなる．このような大容量のマルチポートレジスタファイルを提供することは，面積，アクセス時間の両面で，大きなチャレンジといえる．我々は，コンパクトなフットプリントと1サイクルでのアクセスを維持するため，新しい実現方式を採用した．

関数呼び出しには，新しいウィンドウが必要になる．ウィンドウはスライドアップする．*out*は*in*に変わる．リターンでは，スライドダウンが起こる．我々はコンパクトなレジスタファイルを実装するために，この性質を利用した．スレッドから見えるレジスタのセットを**ワーキングセット**と呼び，高速なレジスタファイルセルを用いて実装する．レジスタセット全体を**アーキテクチャセット**と呼び，コンパクトな6トランジスタSRAMセルを用いて実装する．2つのレジスタセットは転送ポート（transfer port）で結ばれる．ウィンドウを変えるようなイベントが発生すると，今のスレッドは選択対象から外され，転送が開始される．データ転送には，イベントのタイプによって，1ないし2サイクルかかる．転送が終わると，スレッドは再び選択対象となる．このデータ転送は他のスレッドによるレジスタファイルに対する読み書きとは独立に行われるので，他のスレッドは実行を続けることができる．加えて，あるサイクルにはたかだか1つのスレッドしか読み出しを行わないから，読み出し回路はすべてのスレッドのレジスタ間で共有されている．

メモリサブシステム

1次命令キャッシュの容量，連想度，ラインサイズは，それぞれ，16KB，4，32Byteとなっている．回路面積を削減するため，ランダム置き換えアルゴリズムを採用したが，大きな性能低下はない．命令キャッシュからは，毎サイクル2命令フェッチする．2つ目の命令が利用可能なら，次のサイクルでは命令キャッシュに空きスロットができる．このスロットは，パイプラインがストールなしでラインをフィルするのに使う．

1次データキャッシュの，容量，連想度，ラインサイズは，それぞれ，8KB，4，16Byteとなっており，ライトスルー方式をとっている．1次キャッシュは小容量ではあるが，そのミス率は10%以内であり，スレッドごとの平均メモリアクセス時間を大幅に削減することができる．商用のサーバアプリケーションは一般に大きなワーキングセットを持つので，ミス率を十分に小さくし

ようとすると，1 次キャッシュは巨大なものになる．これは回路面積が大きくなって好ましくない．ところで，本アーキテクチャではスレッドグループの 4 つのスレッドによって，1 次，2 次キャッシュのミスレイテンシを効率よく隠蔽することができる．したがって，ミス率，面積，レイテンシ隠蔽能力の間のよいトレードオフポイントとして，容量が 8KB となった．

Niagara では，単純なコヒーレンスプロトコルが用いられる．1 次キャッシュはライトスルーであり，ロードミス時にはアロケートを行うが，ストアミス時にはアロケートを行わない．1 次キャッシュラインの状態は，valid/ invalid の 2 つである．2 次キャッシュは，1 次キャッシュのタグをシャドーするディレクトリを管理する．2 次キャッシュは，データを 64Byte 単位でバンク間インタリーブしている．1 次キャッシュでミスしたロード命令は，2 次キャッシュのソースバンクに，1 次キャッシュからリプレースされるウェイの情報とともに送られる．そこで，ロードミスしたアドレスはディレクトリの中の対応する 1 次タグの場所に入れられる．2 次キャッシュがアクセスされて，ミスしたラインのデータは 1 次キャッシュに戻される．ディレクトリは 1 次キャッシュライン単位で当該データの共有者のリストを管理する．異なる，または，同じ 1 次キャッシュから引き続くストアはディレクトリを検索し，当該ラインを持っている 1 次キャッシュに対する無効化要求をキューイングする．ストア命令は，2 次キャッシュが更新されるまで，ローカルのキャッシュを更新しない．この間，ストアデータは，同じスレッドには渡されるが，他のスレッドには渡されない．したがって，ストアは 2 次キャッシュのレベルで全体的な可視性を持つことになる．クロスバーが，同じ，あるいは，異なる 2 次キャッシュバンクからのメモリオーダを決定し，同じ順序で 1 次キャッシュへトランザクションが配送されることを保証する．2 次キャッシュはコピーバック方式である．I/O デバイスからの DMA は，2 次キャッシュを通して順序付けされる．4 つの DDR2 DRAM チャネルが 20GB/s を超えるバンド幅を提供する．

おわりに

Niagara システムは現在テスト中である．実験機上では，Solaris 向けに書かれた既存のマルチスレッドソフトウェアを変更なしに動作させることに成功した．パイプラインが単純であるため，コンパイラによる特別な命令スケジューリングの最適化は必要ない．現実の商用アプリケーションに対して，スレッド数にほぼ比例する性能が得られ，デザイン上のボトルネックは起こらないことを確認できた．Niagara プロセッサは，多数のハードウェアスレッドによって，商用のサーバアプリケーションに対して高いスループットと高いワットあたり性能を提供することができる．Niagara は，Sun のこうしたマイクロプロセッサの最初のものとして位置付けられる．スレッドリッチなアーキテクチャが利用可能になることによって，ソフトウェア開発者が効率的にアプリケーションの性能を向上させる新たな道が切り拓かれた．このアーキテクチャは，これまでのマイクロプロセッサのデザインに対するパラダイムシフトである．

謝辞　本稿は，才能とやる気にあふれた Niagara 開発チームの成果を記したものであり，著者一同はここに彼らの仕事を紹介できることを誇りに思う．

編集者謝辞　本解説の翻訳掲載を快諾された原著者の皆様，翻訳にあたってご尽力いただいたサンマイクロシステムズ（株）ならびに同社の近藤豪様に感謝申し上げます．なお，元の解説には囲み記事「Hardware multithreading primer」が入っていたが，これは前月号掲載の本特集の解説でカバーされているため，この記事では省略した．

参考文献

1）Kunkel, S. R. et al.: A Performance Methodology for Commercial Servers, IBM J. Research and Development, Vol.44, No.6, pp. 851-872 (2000).
2）Barroso, L., Dean J. and Hoezle, U. : Web Search for a Planet: The Architecture of the Google Cluster, IEEE Micro, Vol.23, No.2, pp. 22-28 (Mar.-Apr. 2003).
3）Olukotun, K. et al.: The Case for a Single Chip Multiprocessor, Proc. 7th Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS VII), pp.2-11 (1996).
4）Laudon, J., Gupta, A. and Horowitz, M. : Interleaving: A Multithreading Technique Targeting Multiprocessors and Workstations, Proc. 6th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS VI), ACM Press, pp.308-316 (1994).
5）Barroso, L. et al.: Piranha : A Scalable Architecture Based on Single-Chip Multiprocessing, Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA 00), IEEE CS Press, pp.282-293 (2000).
6）Kapil, S., McGhan, H. and Lawrendra, J. : A Chip Multithreaded Processor for Network-Facing Workloads, IEEE Micro, Vol.24, No.2, pp.20-30 (Mar.-Apr. 2004).
7）Hart, J. et al.: Implementation of a 4th-Generation 1.8 GHz Dual Core Sparc V9 Microprocessor, Proc. Int'l Solid-State Circuits Conf. (ISSCC05), IEEE Press, 2005, http://www.isscc.org/isscc/2005/ap/ISSCC2005AdvanceProgram.pdf

（平成 17 年 10 月 3 日受付）

■本稿に関する質問／コメントの連絡先：
Poonacha Kongetira, MS:
USUN05-215, 910 Hermosa Court, Sunnyvale, CA 94086;
poonacha.kongetira@sun.com