2004－ARC－159　(13)
2004／7／30

排他実行マルチスレッド実行モデルに基づくFuceプロセッサの評価

松崎 隆哲† 雨宮 聡史‡ 泉 雅昭‡ 雨宮 真人†

†九州大学 大学院 システム情報科学研究院
‡九州大学 大学院 システム情報科学府

排他実行マルチスレッドモデルとは、他プログラムからの中断を受けずに排他的に走りきるプログラム片をスレッドと定義し、スレッドを並列に実行するスレッド実行モデルである。Fuceプロセッサは多数のスレッドを複数のスレッド実行ユニットで効率良く実行するために、データフローを基盤とする継続概念をスレッドの並列実行モデルに採用し、排他実行マルチスレッドモデルを実現している。本稿では、Fuceプロセッサの構成について述べ、シミュレーションによってプロセッサの性能を評価する。

The design and evaluation of a processor based on Exclusive Multi-threaded Execution Model

Takanori Matsuzaki, Satoshi Amamiya, Masaaki Izumi, and Makoto Amamiya

Graduate School of Information Science and Electrical Engineering,
Kyushu University

In the exclusive multi-threaded execution model, programs are constructed with non-preemptive threads to execute multiple threads concurrently. The core concept of exclusive multi-threaded execution model is the *continuation*-based computing model derived from the dataflow computing. We have been developing the Fuce processor to realize the exclusive multi-threaded model. In this paper, we discuss the architecture design of the Fuce processor and evaluate the effectiveness of the Fuce processor.

1　はじめに

近年の半導体技術の向上によって、プロセッサアーキテクチャの進歩はとどまるところを知らない。近年の代表的なプロセッサであるスーパスカラプロセッサは、その構造を複雑化するとともに多大なる性能向上を果たしている。しかしながら、単一プロセス実行または単一スレッド実行における命令レベルの並列性の抽出には限界があり、スーパスカラプロセッサでは並列性を十分に活かしきれていないという問題もある。その対策として、複数のプロセスまたは複数のスレッドを同時実行させて、スループットの向上をはかるSMT (Simultaneous Multi Threading) プロセッサが提案[4]され、実用化[3]されている。また、複数のコアをチップに搭載したCMP (Chip Multi Processor) の研究[5]も進められている。

我々は今後の集積回路技術の進歩を見据え、メモリとプロセッサを同一チップに搭載し、マルチスレッド実行方式にもとづいたFuceプロセッサ[2][7]を提案してきた。命令レベルの並列性の利用には限界があるとの観点から、Fuceプロセッサはスレッドレベルの並列性に着目してきた。多数のスレッドを効率よく実行するために、複数のスレッド実行ユニットを搭載したチップマルチプロセッサを想定し、データフロー計算モデルを基盤とした継続概念[1]をスレッドの並列実行モデルに採用することで、スレッドの並列実効性能を追求している。

本稿では、データフローモデルを基盤とする継続概念を利用したプロセッサの構成を述べ、プロセッサの性能評価によって、並列性能が得られることを

示す。

2　排他実行マルチスレッド実行モデル

2.1　継続モデル

Fuce プロセッサのスレッド並列実行モデルは、データフロー計算モデルに基づいた継続という概念を核としている。継続は、スレッド間のデータ依存関係によって定義される。図 1 に継続の概念を示す。図 1(a) は 3 つのスレッド A,B,C の依存関係を示している。B は A の結果を必要とし、C は B の結果を必要としている。これら 3 つのスレッドを実行するためには、A は計算結果とともに B に対して通知を送り、B は計算結果を C に通知しなければならない。この結果の通知を継続と呼び、A は B に継続し、B は C に継続すると言う。図 1(b) は継続するスレッドが複数の場合を示している。スレッド B と C は依存関係がないので並列走行が可能であり、スレッド D は B と C から継続されないと走行できない。

あるスレッドに対して継続される元のスレッドの数を fan-in 値、継続する先のスレッドの数を fan-out 値と呼ぶ。スレッド B の fan-out は 2 であり、スレッド D の fan-in は 2 である。

スレッドレベルの実行制御は継続によって制御される。スレッドは継続されるたびに fan-in 値をデクリメントされ、0 になった時に実行可能となり発火・実行される。そのスレッドは消滅するまでいかなる干渉も受けずに走行する。なお、概念上、スレッドは fan-out 値が 0 となり、継続すべきスレッド全てに継続し終わったときに消滅する。

2.2　スレッドと関数インスタンス

前述の継続モデルを実現するために、Fuce プロセッサでは関数とスレッドを中心にしてプログラミングモデルを定義している。図 2 にスレッドと関数インスタンスの関係を示す。一般に、関数は複数のスレッドで構成され、その関数の実行環境 (命令列とデータ領域) として関数インスタンスを持つ。スレッドは関数インスタンスをコンテキストとして利用する。同一の関数に属するスレッドは、関数インスタンスを共有する。

Fuce プロセッサにおけるスレッドの定義を以下に述べる。

- スレッドは同期値 (fan-in 値) を持つ。他のスレッドによる継続命令によって、同期値がデクリメントされ、同期値が 0 になると実行可能状態になる。
- 他のスレッドとの同期は、スレッドが実行可能状態になった時点で解決されており、スレッド実行中に同期待ちを行わない。
- 実行終了命令を実行するまで中断することなく排他的に走りきる。
- スレッドの大きさは、扱うデータがレジスタに収まるサイズとする。

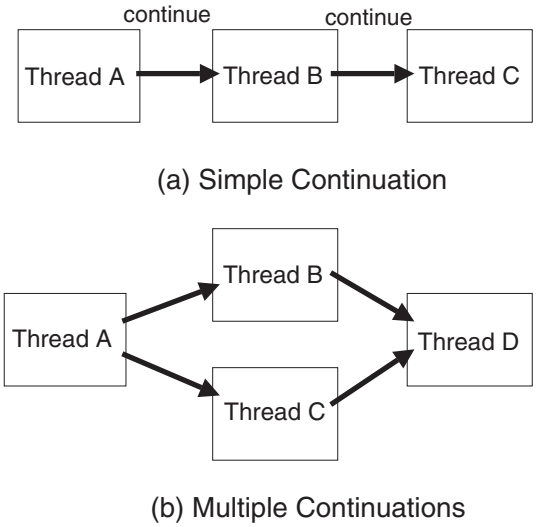

図 1: スレッド間の継続

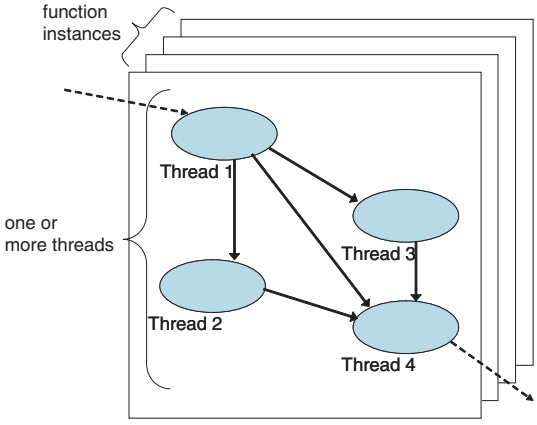

図 2: スレッドと関数インスタンス

3　Fuce プロセッサ

3.1　基本概要

Fuce(Fusion of communication and execution) プロセッサは、その名が示すように通信処理と通常の処理を同様に扱うという観点から設計されている。これは、プロセッサ内部におけるすべての処理を排他的に実行可能なプログラム断片に細分化し、これを多重化して並列に実行することで実現している。また、割り込み処理を含めたすべての処理をスレッドとして扱い、中断無しに実行する。これによって、通信や入出力などの外部イベントの処理や内部計算処理を効率的に行うことができる。

Fuce プロセッサは、マルチメディア処理のような大容量データやストリーム処理のような再利用性の低いデータを処理することを念頭においている。そのため、大容量メモリをプロセッサ内部に統合することで、大容量データを異なるスレッドで同時に扱えるプロセッサ構造を採用している。

3.2　プロセッサの基本構成

図 3 にプロセッサの概要を示し、並列処理の観点からプロセッサの特徴を述べる。

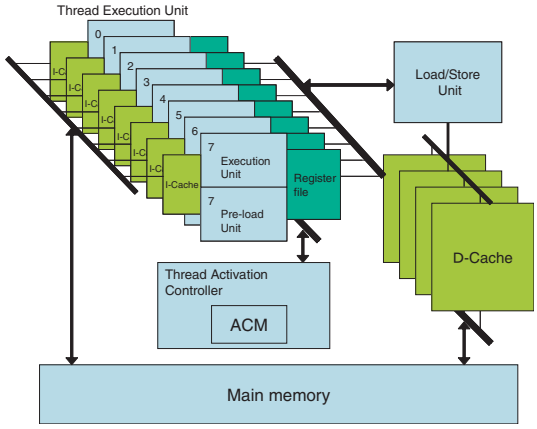

図 3: Fuce プロセッサ概要

- スレッド実行ユニット

  スレッド実行ユニットはスレッド命令列を処理するユニットである。Fuce プロセッサはスレッド実行ユニットを 8 個持ち、同時に 8 個のスレッドを実行することができる。スレッドを効率的に実行するために、スレッド実行ユニットは演算ユニットとプリロードユニットの二つで構成されている。演算ユニットは単純な RISC プロセッサであり、プリロードユニットは演算ユニットのサブセットとしてデータのロードに関する命令のみサポートする。

- レジスタファイル

  レジスタファイルは 2 つのセットによって構成されている。一方は、Current Register File (CRF) と呼ばれ、演算ユニットで用いられる。他方は、Alternate Register File (ARF) と呼ばれ、後述するスレッドコンテキストの先読みを行う際にプリロードユニットで用いられる。CRF と ARF のそれぞれの役目はスレッド切り替え時に交替する。

- Thread Activation Controller

  Fuce プロセッサは、スレッドの同期管理と起動を制御する機能ユニット Thread Activation Controller(TAC) を持ち、スレッドレベル並列実行を制御する。TAC の詳細については後述する。

3.3　スレッドコンテキスト先読み

Fuce プロセッサでは、プリロードユニットとレジスタファイルを用いて、スレッドコンテキストの先読みを行う。このことによって、スレッド実行中のメモリアクセスレイテンシを隠蔽することができる。

図 4 にスレッドコンテキスト先読みの概要を示す。これは、演算ユニットが CRF を利用してスレッド A の処理を行っている間にプリロードユニットが ARF にスレッド B のデータを読み込むことで実現する。先読み機構によって、演算ユニットにてスレッドの処理を開始する際に必要なデータをレジスタに整える。

データの読み込みが終了したスレッド B は、スレッド A の処理が終了するのを待つ。スレッド A の処理が終了した時、スレッド B の処理が演算ユニットで開始される。その際にレジスタファイルの CRF と ARF の役目が入れ替わる。すなわち、スレッド B のデータを読み込んだ ARF が、CRF として演算ユニットで利用され、CRF として利用されていたレジスタファイルは ARF としてプリロードユニットで利用されることになる。

スレッドコンテキストの先読みは、コンパイラの命令スケジュールによって、スレッドコードの先頭部分にロード命令列、残りの部分に演算命令とストア命令が配置されることを前提としている。

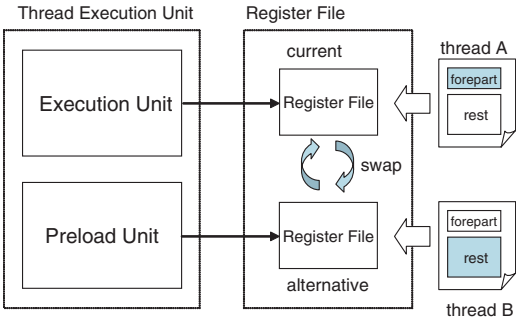

図 4: スレッドコンテキスト先読みの概要

3.4 Thread Activation Controller (TAC)

Fuce プロセッサは、Thread Activation Controller (TAC) を持ち、TAC 内部の Activation Cotrol Memory (ACM) に関数インスタンス毎の情報を保持してスレッドの管理を行う。TAC はスレッド実行ユニットで発行されたスレッド制御命令を処理し、ACM を書き換える。図 5 に ACM の構造を示す。ACM は OS の仮想記憶システムと同様のページ構造となっている。ACM の各ページは関数インスタンスに対応して割り付けられ、関数インスタンスの情報を保持する。ACM の各エントリには、関数内のスレッドの情報 (スレッドの現在の同期値 (fan-in 値)、同期値の初期値 (f-init)、スレッドの先頭アドレス (Code-entry)) が保持される。関数内のスレッドは ID で特定される。スレッドの ID は、仮想記憶システムと同様に、ACM のページ番号とページ内オフセットで表す。TAC 内部には、キューが用意され同期が完了して実行可能となったスレッドをキューに保持する。プリロードユニットに空きができると、キュー内のスレッドがスレッド実行ユニットに割り当てられ、プリロードユニットによって、スレッドコンテキストの先読みを行う。

また、Fuce プロセッサは、TAC を利用して、スレッド間のデータ受け渡しや排他制御を行うことができる。8 個のスレッド実行ユニットでスレッドを並列実行する際のスレッド間のデータ受け渡しに共有メモリ領域を利用するとコストが大きくなる。Fuce プロセッサでは、TAC の排他制御機構を用い、

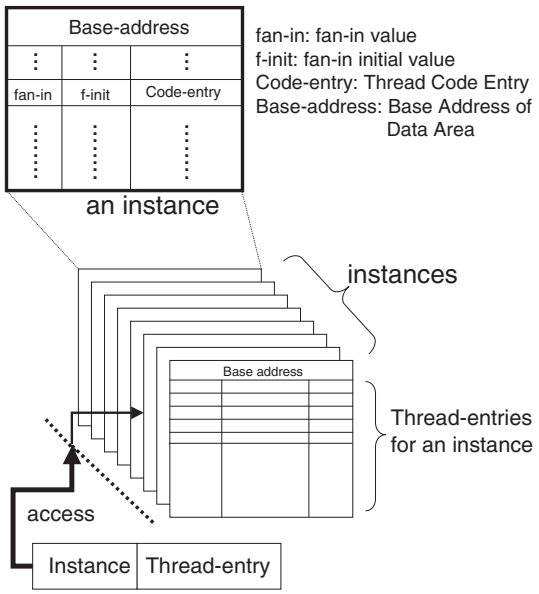

図 5: ACM(Activation Control Memory) の概要

また ACM を高速なデータ受け渡し領域として用いることによって、高速化なスレッド間排他実行を可能とする。

4 性能評価

本節では、Fuce プロセッサの並列性能に関し、スレッドコンテキスト先読み (プリロード) と TAC を利用したデータ受け渡しの効果について評価する。なお、評価は VHDL にて記述した Fuce プロセッサを ModelSim 上で動作させて行った。今回利用した Fuce プロセッサのシミュレーション環境は、8 個のスレッド実行ユニットを持ち、TAC のアクセスレイテンシは 1 サイクルとし、メモリアクセスレイテンシは可変である、ただし、データキャッシュは搭載されていない。

評価では、4096 個の整数の最大値探索プログラム、および 8-Queen の全解探索プログラムを、Fuce アセンブラによって作成した。それぞれのプログラムについて、プリロードを行った場合と行わなかった場合についてメモリアクセスレイテンシを変化させて実行した。なお、両プログラムとも、TAC の排他制御機構を用い、ACM をデータ受け渡し領域として用いてる。

4096 個の整数の最大値探索に要した実行サイクル数を図 6 に、IPC を図 7 に示す。

8-Queen の全解探索問題に要した実行サイクル数

を図 8 に、IPC を図 9 に示す。

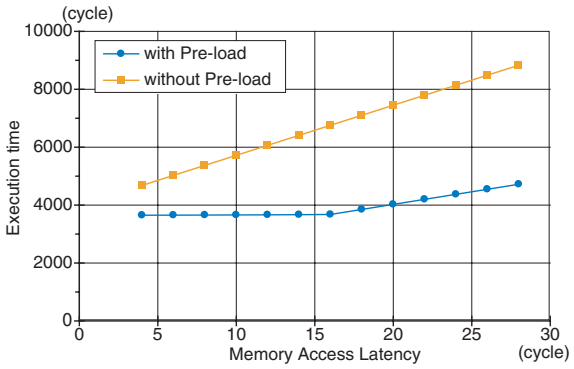

図 6: 最大値探索の実行サイクル数

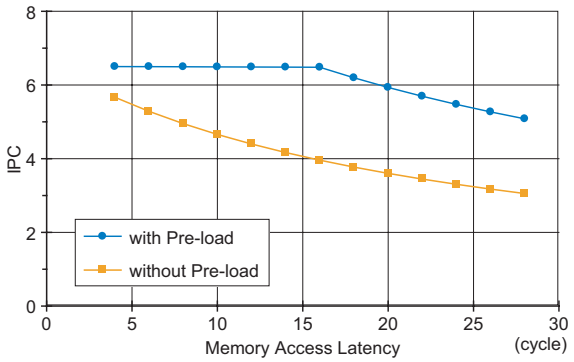

図 7: 最大値探索の IPC

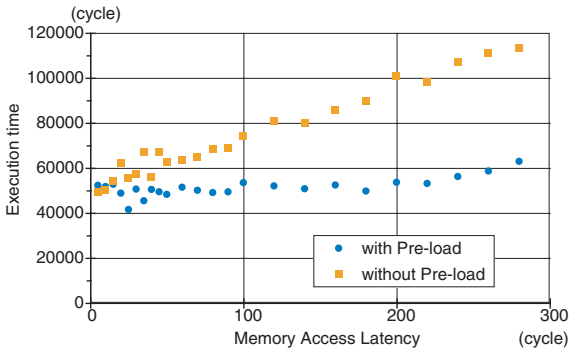

図 8: 8-Queen の全解探索の実行サイクル数

また、8-Queen の全解探索問題について、スレッド間のデータ受け渡しに TAC を用いた場合と共有メモリを用いた場合について、メモリアクセスレイテンシを変化させ比較した。図 10 に実行サイクル数の比較を示し、IPC の比較を図 11 に示す。

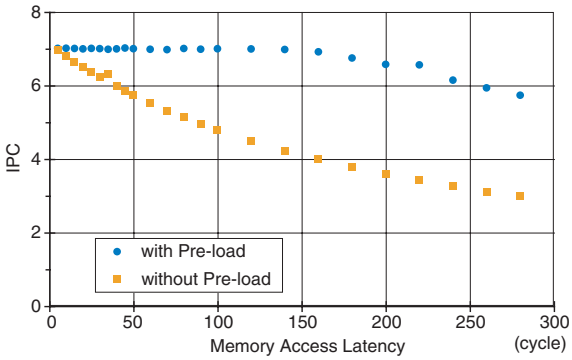

図 9: 8-Queen の全解探索の IPC

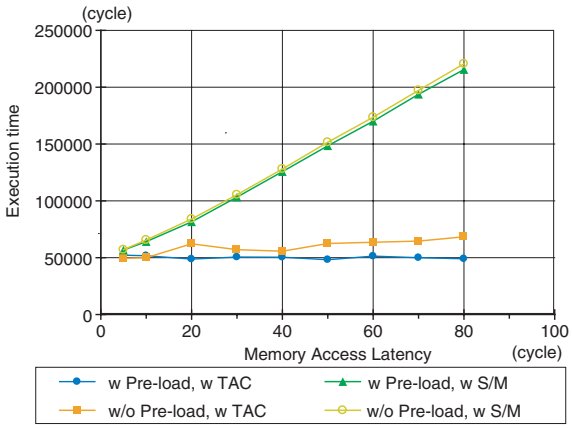

図 10: 8-Queen の全解探索の実行サイクル数
– TAC 利用と共有メモリ使用の場合比較 –

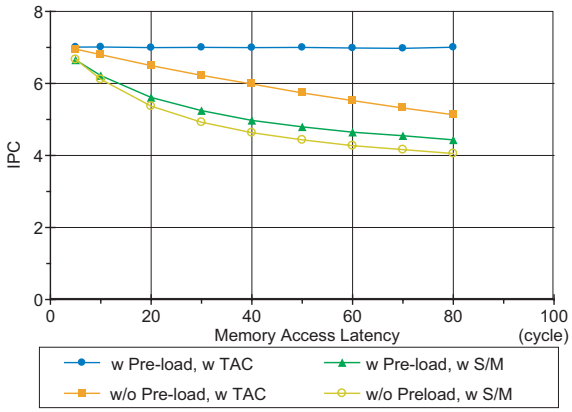

図 11: 8-Queen の全解探索の IPC
– TAC 利用と共有メモリ使用の場合比較 –

図 6 は、プリロードを行うことで、メモリアクセスレイテンシが 16 サイクル以下では実行サイクル

数が一定であることを示している。このことからスレッド実行中のメモリアクセスレイテンシを隠蔽できていることがわかる。メモリアクセスレイテンシが 16 以上のときには実行サイクル数が増加しているが、その増加率はプリロードを行なった場合の方が行わなかった場合の約 0.5 倍に抑えられている。同様に、図 8 からプリロードを行った場合、おおよそ 160 サイクル程度までメモリアクセスレイテンシを隠蔽できていることがわかる。メモリアクセスレイテンシが 160 サイクルを超えた場合の増加率もプリロードを行ったほうが行わなかった場合の約 0.5 倍に抑えられている。以上のことより、プリロードを行うことによって、メモリアクセスレイテンシの増加に対してある時点までは実行サイクル数を一定に保つことができ、また、実行サイクル数が増加する場合においても実行サイクル数の増加率を抑えることができる。

図 10、図 11 より、スレッド間のデータ受け渡しに TAC を用いることで、大幅な性能改善効果を得ていることがわかる。本評価に利用した Fuce プロセッサのシミュレーション環境では、データキャッシュを利用していないためメモリアクセスのペナルティが大きくなる。しかしながら、データキャッシュが常に利用できている場合として想定できるレイテンシ 4〜10 サイクルの範囲では、スレッド間のデータ受け渡しに TAC を用いた方が効果的である。また、多数のスレッドが並列実行する場合はデータキャッシュの効果があまり期待できないと想定できる、その場合のレイテンシを 40〜50 サイクル程度と想定すると、スレッド間のデータ受け渡しに TAC を用いると 2 倍以上の性能向上が得られる。以上のことより、TAC はスレッドの実行制御だけではなく、スレッド間のデータ受け渡し領域としても利用でき、効果も期待できる。ただし、TAC をデータ受け渡し領域として利用する場合には、ACM の管理をプログラムで行う必要があり、そのようなコードを生成するコンパイラが必要となる。

5 おわりに

本稿では、Fuce プロセッサの実行モデルである排他実行マルチスレッド実行モデルについて述べ、プロセッサアーキテクチャについて述べた。また、Fuce プロセッサアーキテクチャが排他制御を伴う並列処理に有効であることをシミュレーションによって示した。

現在、このプロセッサはまだ実装段階にあるため、データキャッシュを利用した場合の性能を示すことができなかった。Fuce プロセッサは、データフローモデルを基盤としているため、データの局所性を生かしにくいという問題があるが、一方、Fuce プロセッサでは TAC のスレッド制御機構を利用することによって、スレッド間パイプライン実行のような従来手法とは異なったプログラミング手法 [6][8] を実現することができる。

今後は、本プロセッサを FPGA ボード上に実装し、その上で OS カーネルプログラムを開発して動作させ、プロセッサの性能評価を行う予定である。

謝辞 本研究は、文部科学省科学研究費補助金・基盤研究 (A)(2)「細粒度マルチスレッド処理原理による並列分散処理カーネルウェアの研究」(課題番号：15200002) による。

参考文献

[1] Amamiya, M., ”A New Parallel Graph Reduction Model and its Machine Architecture,” Data Flow Computing: Theory and Practice, Ablex Publishing Corporation, pp.445-467, 1991.

[2] Amamiya, M., Taniguchi, H. and Matsuzaki, T., ”An Architecture of Fusing Communication and Execution for Global Distributed Processing, ”Parallel Processing Letters, Vol.11, No.1, pp.7-24, (2001).

[3] Marr, D. T. , Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A. and Upton, M. ”Hyper-threading technology architecture and microarchitecture: a hyperhtext history,” Intel Technology J. 6,1 2002 (online journal).

[4] Lo, J. L., Eggers, S. J. , Emer, J. S., Levy, H. M., Stamm, R. L. and Tullsen, D. M., ”Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading, ” ACM Transactions on Computer Systems, Vol.15, No.3, pp.322-354, 1997.

[5] Sun Microsystems Inc. Throughput computinghttp://www.sun.com/processors/throughput/.

[6] 雨宮 聡史, 松崎 隆哲, 雨宮 真人, “排他実行マルチスレッド実行モデルに基づくオンチップ・マルチプロセッサの設計,” 情報処理学会研究報告，2003-ARC-155, pp. 51-56，(2003).

[7] 雨宮 真人 他, 通信・放送機構 (TAO) 研究成果報告書,「情報通信網の基盤技術に関する研究」, 平成 15 年 3 月.

[8] 泉 雅昭, 雨宮 聡史, 松崎 隆哲, 雨宮 真人, “継続モデルに基づくスレッドプログラミング手法の提案,” 情報処理学会研究報告，2004-ARC-159，(2004).