九州大学学術情報リポジトリ
Kyushu University Institutional Repository

# スレッドレベル並列処理プロセッサFUCEのハードウェア構成とその評価

松崎, 隆哲
九州大学大学院システム情報科学研究院知能システム学部門

雨宮, 聡史
九州大学大学院システム情報科学府知能システム学専攻 : 博士後期課程

泉, 雅昭
九州大学大学院システム情報科学府知能システム学専攻 : 修士課程

雨宮, 真人
九州大学大学院システム情報科学研究院知能システム学部門

https://doi.org/10.15017/1516229

出版情報：九州大学大学院システム情報科学紀要. 11 (1), pp.39-44, 2006-03-24. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第11巻 第1号 平成18年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.11, No.1, March 2006

# スレッドレベル並列処理プロセッサFUCEのハードウェア構成とその評価

松崎 隆哲* ・ 雨宮 聡史**・ 泉 雅昭***・ 雨宮 真人*

# Hardware Design and Evaluation of the FUCE Processor for Thread Level Parallel Processing

Takanori MATSUZAKI, Satoshi AMAMIYA, Masaaki IZUMI and Makoto AMAMIYA

(Received December 9, 2005)

**Abstract:** We are developing the FUCE processor based on the dataflow computing model. FUCE means FUsion of Communication and Execution. In order to execute many threads with multiple thread execution units efficiently, the FUCE processor executes multiple threads using the exclusive multi-thread execution model. The core concept of the exclusive multi-thread execution model is continuation based multi-thread execution, which is derived from dataflow computing. The FUCE processor unifies processing inside the processor and communication with processors outside as events, and executes the event as a thread. In this paper, we introduce the thread-programming model and the architecture of the FUCE processor and evaluate the hardware cost of a FUCE processor and the concurrency performance of a FUCE processor which we described in VHDL. As a result, we understood that the processor has concurrency capability when there is sufficient thread level parallelism.

**Keywords:** Thread level parallelism, Multi-thread, Dataflow, Continuation model

## 1. はじめに

今日のプロセッサアーキテクチャは，プロセッサ内部で命令レベルの並列性(Instruction Level Parallelism:ILP)を利用することで性能向上を果たしてきた．特に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，近年の半導体技術の進歩によって単一チップに搭載可能なトランジスタ数が増大したことにより，プロセッサ構造の複雑化が可能となった．これによって，スーパースカラプロセッサは，投機的な命令実行や複数命令の同時発行といった命令レベルの並列性を利用するための機構をプロセッサに搭載し，プログラムの並列性をハードウェアによって抽出することで性能向上を果たしてきた．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は限界があるため[7]，スーパスカラプロセッサは並列性を十分に活かすことができないという問題がある．また，命令レベルの並列性を抽出するためプロセッサアーキテクチャが複雑化するといった問題もある．その対策として，複数のプロセスや複数のスレッドを同時に実行させることで，スーパスカラプロセッサの並列性を活用しスループットの向上を図る手法として，スレッドレベルの並列性（Thread Level Parallelism:TLP）が着目されている．

スレッドレベルの並列性を利用するプロセッサとして，SMT（Simultaneous MultiThreading）プロセッサが提案[4]され，実用化[3]されている．また，複数のプロセッサコアをチップに搭載したCMP（Chip MultiProcessor）の研究[6]も進められている．しかしながら，これらの研究は基本的にノイマン型逐次実行モデルを拡張したものである．これらは，元々単一の命令流（逐次的な命令列）を効率良く扱うためのプロセッサアーキテクチャの拡張であるため，複数の命令流を効率良く扱うことが困難であり，並列分散処理には向いていないという問題がある．

我々は，ノイマン型逐次実行モデルを利用した命令レベルの並列性の抽出には限界であるとの立場から，並列処理と親和性の高いデータフロー計算モデルを基盤としたFUCEプロセッサを開発している．FUCEプロセッサは，一般的なノイマン型プロセッサアーキテクチャにデータフローの概念を取り入れたスレッド並列実行メカニズムをプロセッサに搭載することで，スレッドレベルの並列性を利用するプロセッサである．スレッドレベルの並列性を利用するため，FUCEプロセッサではプロセッサ内部にスレッドの実行制御を行う付加回路を追加することで，継続概念[1]に基づいたプログラミングモデルを実現している．

本稿では，データフロー計算モデルを基盤とする継続概念を用いたスレッドレベル並列処理プロセッサのハードウェア構成を述べ，プロセッサ構成に要するコストの試算とプロセッサの性能評価を行う．

平成17年12月9日受付
* 知能システム学部門
** 知能システム学専攻博士後期課程
*** 知能システム学専攻修士課程

## 2. FUCE アーキテクチャ

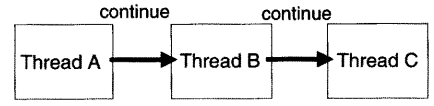
FUCEアーキテクチャはデータフロー計算モデルを利用したプロセッサアーキテクチャであり，継続概念に基づくマルチスレッド処理を行うFUCE プロセッサと継続概念に基づいたプログラミングモデルから構成される．本章では，FUCEアーキテクチャを構成するプログラミングモデルについて述べる．

### 2.1 継続モデル

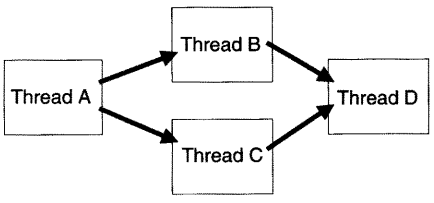
FUCEアーキテクチャのスレッド並列実行モデルは，データフロー計算モデルに基づいた継続概念を核としている．継続はスレッド間のデータ依存関係によって定義される．**Fig. 1**に継続概念を示す．**Fig. 1** (a)は三つのスレッドA,B,Cの依存関係を示している．BはAの結果を必要とし，CはBの結果を必要としている．これら三つのスレッドを実行するためには，Aは計算結果とともにBに対して通知を送り，Bは計算結果をCに通知しなければならない．この結果の通知を継続と呼び，AはBに継続し，BはCに継続するという．**Fig. 1** (b)は継続するスレッドが複数の場合を示している．スレッドBとCは依存関係がないので並列実行が可能であり，スレッドDはBとCから継続されないと実行できない．

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．**Fig. 1** (b)において，スレッドAのfan-out値は2であり，スレッドDのfan-in値は2である．

FUCEアーキテクチャにおけるスレッド並列実行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実行される．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．




(a) Simple Continuation




(b) Multiple Continuations

**Fig. 1**　Thread Continuation.

### 2.2 スレッドモデル

FUCEアーキテクチャは継続モデルを実現するために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．**Fig. 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用いる．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．

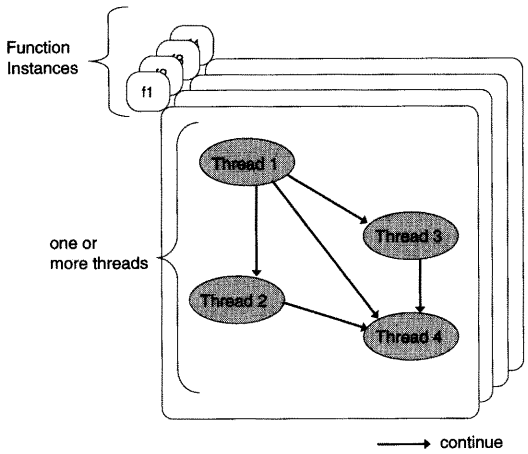



**Fig. 2**　Threads and Function Instance.

FUCEアーキテクチャ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（粒度）は，扱うデータがレジスタに収まるサイズとする．

## 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，プロセッサのすべての処理を排他的に実行するスレッドとして扱うことによって，プロセッサ内部におけるすべての処理を分割多重化して並列に行うことで実現している．また，プロセッサ内部の演算やシステムコールを内部イベント，従来のシステムで割り込みと呼ばれる処理

を外部イベントと呼び，両者を区別なくイベントとして扱う．このスレッド実行モデルを実現するため，FUCEプロセッサはプロセッサ内部にスレッド管理機構を持ちイベントの管理をハードウェアで行う．

データフロー計算モデルを基にした並列プロセッサアーキテクチャは，メモリアクセスが頻発しかつそのメモリアクセスに関連性が低いため，参照の局所性を効果的に利用できずキャッシュの効率が悪いという問題がある．FUCEプロセッサは，データフロー計算モデルをスレッド実行方式に用いているため，高速なメモリアクセスを実現することがプロセッサの構成上大変重要である．そこで，プロセッサ内部にメモリを搭載することで高速なメモリアクセスを実現する．

### 3.2 プロセッサの基本構成

**Fig. 3**にプロセッサの概要を示し，スレッドの並列処理の観点からプロセッサの特徴を述べる．

FUCEプロセッサは，スレッドの実行を行う命令実行ユニット（スレッド実行ユニット:Thread Execution Unit）を複数個搭載しているチップマルチプロセッサである．また，継続概念に基づいたスレッド実行を実現するために，プロセッサ内部にスレッド管理機構（Thread Activation Controller）を持つ．FUCEプロセッサは，スレッド管理機構を用いることで，排他的マルチスレッド実行[11]や継続概念に基づいたスレッド実行を実現する．

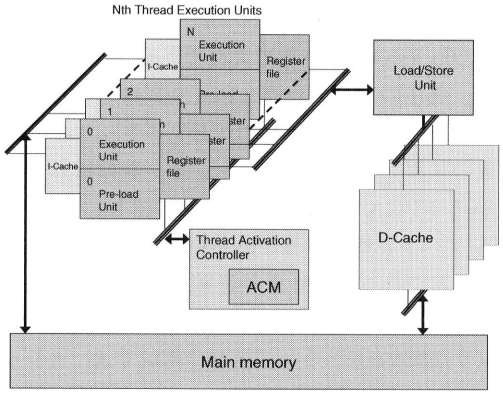



**Fig. 3** Overview of FUCE processor.

- 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複数個持ち，スレッド実行ユニットの個数分のスレッドを同時に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（Execution Unit）とプリロードユニット（Pre-load Unit）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二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 (TAC)
  TACはスレッドの同期管理と起動を制御することで，データフロー計算モデルに基づいたスレッドレベル並列実行を実現する．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の先読み

FUCEプロセッサはデータフロー計算モデルを基にしたマルチスレッド実行モデルを利用しているためスレッドの切り替えが多発する．さらに，キャッシュの効果が低いためスレッド切り替えのメモリアクセスコストが問題となる．そこで，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，スレッドコンテキストの先読みを行うことによってスレッド切り替えのオーバヘッドの削減を実現する．また，このことによってスレッド実行中のメモリアクセスを隠蔽することができる．

**Fig. 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，演算ユニットでスレッドの処理を開始する前に必要なデータをレジスタに事前に用意する．

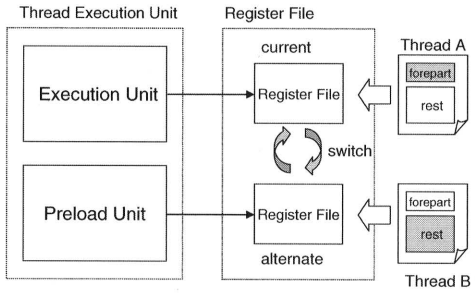



**Fig. 4** Overview of Preload Thread Context.

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．

### 3.4 Thread Activation Controller

Thread Activation Controller(TAC)は，データフロー計算モデルに基づいたFUCEプロセッサのスレッド並列実行を制御するハードウェアスレッド管理の機能ユニットである．TACは内部にスレッドに関する情報を保持する Activation Cotrol Memory (ACM) を持ち，関数インスタンスやスレッドの情報を ACM に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．**Fig. 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(sync-count値)，同期値の初期値（fan-in)，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．

## 4. FUCE プロセッサのハードウェア量

我々は FPGA 実験ボード（Accverinos KM-1[5]）にFUCEプロセッサを実装し，評価している．そこで，実装したFUCEプロセッサの FPGA ゲート数を計算することで，スレッド管理に必要となるハードウェア量の試算を行った．なお，Accuverinous KM-1 は Xilinx社製 XC2V6000 を8個搭載しているため，XC2V6000をターゲットデバイスとして論理合成を行った．今回実装したFUCEプロセッサのスレッド実行ユニットは乗除算演算回路および浮動小数点演算回路を持っていない．また，命令キャッシュやACMを構成するメモリは，FPGA チップ上のBlockRAMにマッピングしたため，ハードウェア量はロジック部分のみ算出している．

FUCEプロセッサのハードウェア量を **Table 1**に示す．ハードウェア量試算の結果，8本のスレッド実行ユニットを持つ FUCEプロセッサは約15万 FPGA ゲートで構成

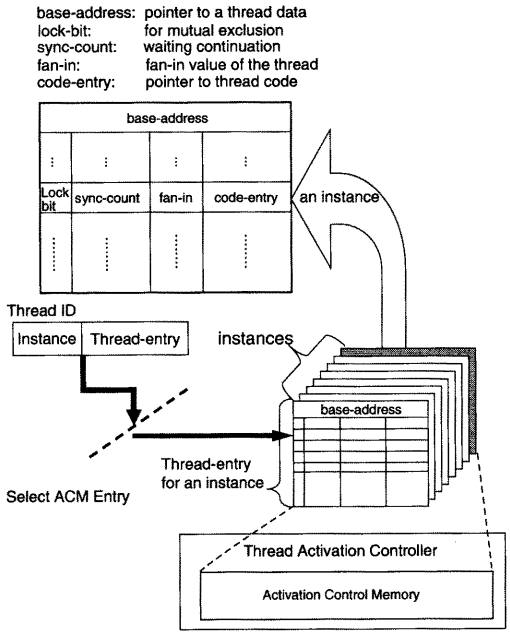



**Fig. 5**　Overview of ACM(Activation Control Memory).

することができ，その内スレッド管理を行うTACに必要なハードウェア量は約6%程度の約9000 FPGA ゲートであるとわかった．また，その他のハードウェア量のおよそ半数が複数のスレッド実行ユニットでスレッドを並列実行するために必要な回路である．それ以外の回路はメモリコントローラや外部インターフェスに利用されている．以上のことより，データフロー計算モデルに基づいたスレッド実行管理機構は，プロセッサ全体の10%程度の約15,000 FPGA ゲートで実現できる．

次に，現在のプロセッサとFUCEプロセッサのハードウェア量を比較するため，FUCEプロセッサのトランジスタ数の試算を行った．トランジスタ数の試算は，利用している FPGA ゲートの種類を確認し，それぞれの FPGA ゲートを構成するのに必要なトランジスタ数を計算することで行った．計算の結果，1 FPGA ゲートは約24トランジスタで構成することができるため，FUCEプロセッサは約360万トランジスタで構成でき，スレッド管理に必要なトランジスタ数は約36万個であると試算された．今回試算に利用したFUCEプロセッサは，乗除算と浮動小数点に関する演算回路を搭載していないため，プロセッサの回路規模は大変小さい物になっている．しかしながら，スレッド管理に利用される回路規模については演算回路が大きくなっても変化はない．そのため，8個の実行ユニットを利用してスレッドを並列に実行するために必要な回路は約36万トランジスタで実現できると見積ることができる．現在の一般的なプロセッサである，Pentium4のロジック部分は約2400万トランジスタであり，FUCEプロセッサと比較すると回路規模は大きい．FUCEプロセッサについても演算ユニットの機能を強化

すると回路規模が大きくなると予測されるが，スレッド管理に利用される回路の規模を変化せずに，8個のスレッドを同時に実行できる．一方のPentium4は2個のスレッドのみ同時に実行可能であることを考えると，FUCEプロセッサは小規模の回路を追加することでスレッドを並列に実行できている．これは，FUCEプロセッサがスレッド実行モデルに並列処理と親和性の高いデータフローモデルを利用したからである．

**Table 1** Amount of gates of FUCE processor.

| Module Name | FPGA Gates |
|---|---|
| Thread Execution Units (×8) | 124,048 |
| Load/Store Unit | 5,555 |
| TAC | 9,112 |
| etc | 11,733 |
| all | 150,448 |

## 5. 性 能 評 価

本章では，FUCEプロセッサの並列性能とメモリアクセス速度による性能変化について評価を行う．なお，評価はVHDLで記述したFUCEプロセッサを利用して行った．

### 5.1 FUCEプロセッサの仕様

本評価で利用したFUCEプロセッサの仕様を**Table 2**に示す．今回利用したVHDLで記述したFUCEプロセッサは，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，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．また，命令キャッシュは各スレッド実行ユニットに1KBずつ持ち，プロセッサ全体で 8 KB搭載している．

**Table 2** Spec. of FUCE processor.

| | |
|---|---|
| Number of Thread Execution Unit | 8 |
| Memory Size | 1 MB |
| Data Cache | no |
| Instruction Cache | 4 KB × 8 |
| Memory Access Latency | 5-200 cycle |
| TAC Access throughput | 1 cycle |

### 5.2 シミュレーションによる性能の評価

性能評価はVHDLで記述したFUCEプロセッサをModelSim 上でシミュレートし，その上でベンチマークプログラムを実行することで行った．

ベンチマークプログラムとしてN-Queenの全解探索問題を利用した．ベンチマークプログラムはFUCEアセンブラによって作成した．N-Queenを利用した理由としては，元々のプログラムの並列度が確保しやすいため，プロセッサの並列性能を測定するのに適しているからである．そのため，作成したプログラムのスレッドレベル並列性は非常に高い．また，FUCEプロセッサはデータフロー計算モデルに基づき多数のスレッド並列に実行するため，キャッシュの効果をあまり得ることができない．そのため，スレッドコンテキストの先読みを行うことでメモリアクセスの隠蔽を行っている．そこで，メモリアクセスレイテンシが大きくなった際の性能評価を行うことで，スレッドコンテキスト先読みの効果について評価し，キャッシュの効果が得られない場合におけるFUCEプロセッサの性能について性能評価から予測する．性能評価は以下の項目について行った．

- スレッド実行ユニットの本数を変化させることによる性能変化
- 耐メモリアクセスレイテンシ

**Fig. 6**にスレッド実行ユニット（Units）の本数と並列効果の関係を示す．メモリアクセスレイテンシが10サイクルにおいてスレッド実行ユニットが1本の場合のクロックサイクル数を基準として，それぞれの実行性能の比を示している．**Fig. 6**を見ると，メモリアクセスレイテンシの値にかかわらず，スレッド実行ユニットの本数が増えると線形的に性能が良くなっている．また，メモリアクセスレイテンシが小さいほど，スレッド実行ユニット増加に対する性能の増加率が良くなっている．特にメモリアクセスレイテンシが10サイクルの場合は，スレッド実行ユニットが8本になると約7.8倍の実効性能が得られている．これらのことより，FUCEプロセッサはプログラムに十分な並列度が有る場合は，その並列度を十分活用できることがわかる．

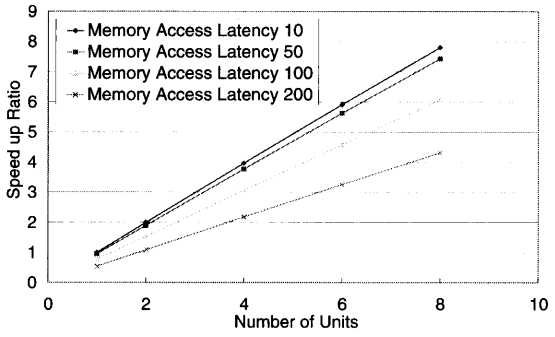



**Fig. 6** N-Queen: Number of Units and Speed up Ratio.

**Fig. 7**にメモリアクセスレイテンシを変化させた場合のクロックサイクル数を示す．8-Queenについて着目すると，メモリアクセスレイテンシが 50 サイクルを超えると

クロックサイクル数が急激に増加している．これは，3.3章で述べたスレッドコンテキスト先読みによって，メモリアクセスレイテンシが50サイクル程度までスレッド実行中のメモリアクセスを隠蔽できているため，クロックサイクル数の増加が抑えられているからである．50サイクルを超えると，スレッドコンテキスト先読みによってメモリアクセスレイテンシのすべてを隠蔽できないためクロックサイクル数の増加率が増えている．

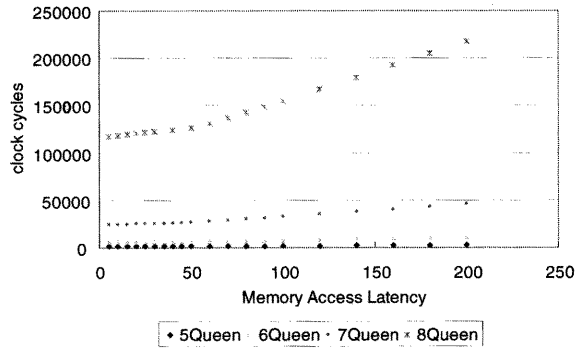



**Fig. 7** N-Queen: Memory Access Latency and clock cycles.

これらの結果から，FUCEプロセッサはプログラムが十分なスレッドレベル並列性を持っている場合は並列性能を発揮することができる．また，スレッドコンテキスト先読みを行うことで，メモリアクセスレイテンシの増加に対してクロックサイクル数の増加を抑えることができる．

## 6. おわりに

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スレッドレベル並列処理プロセッサの構成について述べ，FUCEプロセッサの評価を行った．

評価によって，FUCEプロセッサは小規模な回路の追加によってスレッド並列実行を実現でき，十分なスレッドレベル並列性を得られる場合は並列性能を発揮することができるとわかった．また，FUCEプロセッサはスレッド実行ユニットが増加するに従って並列性能は線形的に増加するとわかった．

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．しかし，メモリアクセスレイテンシを増やした際の性能評価によって，スレッドコンテキスト先読みの効果を示すことができた．FUCEプロセッサは，データフロー計算モデルを基にしたマルチスレッド実行モデルを用いているため，実行モデル自体にデータの局所性を生かしにくいという問題がある．スレッドコンテキストの先読みを行うことである程度メモリアクセスによる性能悪化を防ぐことができるが，キャッシュを効果的に利用するためにTACのスレッド割り当て機構を工夫する必要があると思われる．この点は，今後の課題である．

今後は，FPGA ボード上に実装したFUCEプロセッサを利用しての現実的なプロセッサ性能の評価，そしてソフトウェアシミュレータを利用しハードウェア構成を変更した際の性能の比較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) SK-Electronics CO.,LTD. Accverinous KM-1,http://www.accverinos.jp/
6) Sun Microsystems Inc. Throughput computing, http://www.sun.com/processors/throughput/.
7) D. W. Wall., "Limits of Instruction-Level Parallelism," In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, Vol. 26, pp. 176–189, 1991. ACM Press.
8) 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).
9) 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.
10) 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, pp. 79-84, (2004).
11) 泉雅昭, 雨宮聡史, 松崎隆哲, 雨宮眞人. 排他的マルチスレッド実行モデルにおける多段なループ処理へのスレッド間パイプライン並列実行方式の適用とその評価. FIT2005(第 4 回情報科学技術フォーラム), 第 C-012 巻, pp. 201–204, 9 2005.
12) 松崎 隆哲, 雨宮 聡史, 泉 雅昭, 雨宮 真人, "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," 情報処理学会研究報告, 2004-ARC-159, pp. 73-78, (2004).