2007－ARC－174 (3)
2007／8／1

並列 2Dimensioanl Queue Processorの提案

玄田雅孝† Ben A. Abderazek† 曽和将容†

並列処理に向いている，命令長が短くプログラムサイズが小さい，偽の従属性である逆依存関係と出力依存関係が生じないのでレジスタリネーミングする必要がないなどの性質から，キュープロセッサが研究されている．しかしプログラムによってはサイズの大きなキューやキューに加えランダムアクセスレジスタを要求するということもある．そこでこれらの問題を解消するために複数の Queue を持つプロセッサ Parallel Multi Dimension Queue Processor(MDQ) が提案されている．本論文では現在設計中の MDQ の一つである Parallel 2Dimensional Queue Processor について述べる．

Proposal of the Parallel 2Dimensional Queue Processor

Masatka Genda,† Ben A. Abderazek† and Masahiro Sowa †

Queue-based processors use a FIFO queue to perform operations. The characteristics of this scheme make queue processors suitable for parallel processing. Instructions implicitly access the queue through pointers located at the head and the tail of the queue. Thus, instructions have no operands making them short. The size of the programs is small and instructions are free of false dependencies. Furthermore, program is generated by a level-order traversal that extracts maximum parallelism. However, some programs require a long queue and random access registers. To solve these problems, we propose the Parallel Multi-Dimensional Queue Processor (MDQ) which features multiple queues. This paper describes the Parallel 2-Dimensional Queue Processor, a subset of MDQ.

1. はじめに

マイクロプロセッサは 1970 年代に初めて製造されて以来，半導体技術やプロセッサのアーキテクチャの発展により，目覚まし勢いで性能を向上させてきた．近年，マイクロプロセッサは PDA，携帯電話，家電製品に代表される組み込み機器などありとあらゆる場所で使用されている．そのような背景もあり低消費電力，小面積でありながら処理能力が高いプロセッサというものが強く望まれており，今もなお多くの研究がなされている．プロセッサの処理能力の向上を達成する上で最も重要な技術の一つとして並列化がある．その代表的なものとしてはスーパースカラプロセッサ[1)2)] などがあげられる．スーパースカラプロセッサではプログラムの実行時に並列実行可能な命令を抽出し，並列実行している．

既存のスーパースカラプロセッサでは計算の途中結果を格納する中間記憶にランダムアクセスレジスタを用いている．ランダムアクセスレジスタにアクセスするにはレジスタ番号を命令中に明示的に記述する必要が有る．そのためレジスタ数が増えると命令長が長くなってしまうという問題がある．そのためレジスタ数が不足した場合はレジスタを最利用することで解決する．レジスタを最利用することでレジスタの競合が起き，本来のプログラムには存在しない，偽の従属性である逆従属関係，出力依存関係が生じてしまう．これを解消するためにスーパースカラプロセッサではレジスタを内部で増加させ，レジスタリネーミングを行っている．その結果，並列性を抽出するハードウェアの機構を複雑にし処理能力の向上を難しくしている．

中間記憶に FIFO(First In First Out) のキューを用いたプロセッサであるキュープロセッサについて研究が行われている．我々の研究室ではこれまでにキュープロセッサに存在する 3 種類の計算モデルを明かにし，FPGA による並列キュープロセッサのプロトタイプ実装やコンパイラの開発などを行ってきた[3)4)5)7)]．キュープログラムは並列実行可能な順番に命令が並んでいて (グループ ILP) インオーダ実行で並列性を最大限に引き出せる．

またキューへのアクセスにはヘッド (QH) とテール (QT) に対して行われるので命令中にレジスタ番号を明示的に記述する必要がない．そのため問題の持つ並列性をすべて表現できる．

- 命令長が短いのでプログラムサイズが小さくなる

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communication

- 偽の従属性が生じないのでプログラムの持つ並列性を最大限に抽出できる
- レジスタリネーミングする必要がない
- 並列実行命令の発見が容易
- キュー語数とプログラムに独立性がある

という特徴がある．これまでに消費順序遵守キューコンピューティングモデル，生産順序遵守キューコンピューティングモデルなどの計算法が提案されており[6]，またキュープロセッサの柔軟性を増すために，キューとレジスタ間での演算を許すQRP並列キュープロセッサや複数のキューを用意する並列マルチディメンジョナルキュープロセッサMDQが提案されている[8]．MDQは柔軟性を兼ね備えた並列キュープロセッサである．MDQの内2個のキューからなるマルチディメンジョナルキュープロセッサが最小数のキューを持つMDQである．これを我々は並列Two dimensional queue processor(2DQ)と呼んでいる[9]．2DQはレジスタという概念をもたず，すべての演算をキューで行う柔軟性のある並列プロセッサである．本論文では生産順序遵守コンピューティングモデルに基づく2DQに関してその原理，命令セット，アーキテクチャについて述べる．

2. 2DQ

キューコンピューティングはキューの先頭（キューヘッド）からデータを取り出し，それに演算を施し，その結果をキューの末尾（キューテール）に格納する計算方法であるので，プログラムに中間結果格納用のレジスタ名が出現しない計算方法である．それゆえ，プログラムは，偽従属をもたない，プログラムはレジスタ数から独立している（スケーラビリティ），問題の持っている並列性すべてを表現できる（最大並列表現機能），プログラムサイズが半分になるなど（短プログラム長）など多くの特徴を備えている．しかしながら，実際のプロセッサを実現しようとすると，スタックポインタやフレームポインタ，アドレス修飾用レジスタなどどうしてもレジスタを用意する必要性がでてくるし，また，プログラムの書き方によっては長いキュー長を準備する必要も出てくる．

2DQは二つのQA,QBというキューを持つ並列マルチディメンジョンプロセッサでQAは主に演算用であり,QBはキューとして使うとともに，スタックポインタSP，フレームポインタFP，アドレス修飾レジスタ，一時的なデータ待避用のレジスタとしても用いる．

2DQではキューからのデータを読み出す際にヘッドからだけでなく，ヘッドとオフセットを加えた位置のデータが読み出せる．またデータの書込みにはテールだけでなく，テールとオフセットを加えた位置へ書込みを行うことができる．但し命令ごとにオフセットが適用可能かが決まっていてすべての命令がオフセットを使用できるわけではない（例えば二項演算では第二読み出しデータのみにオフセットが適用可能）．キューへのアクセスを行うと,QH,QTは更新される．QH,QTの更新方法についての詳細な説明は後の章で述べる．図1はab/(a+c)の計算をする2DQのプログラムである．　図1 (a)はy=ab/(a+c)を計算する2DQのプログラムであり，同図(b)は実行に伴うキューの変化を示す."ld B, a"はQBのテール(qtB)にa番地の内容をロードする命令で，"ld A,b"はQAのテール(qtA)にb番地の内容をロードする命令である."mul A,B,A"はQAのキューヘッド(qhA)とQBのキューヘッド(qhB)からデータを取り出しそれに演算を施し，その結果をQAのキューテール(qtA)に書き込む命令である." add A,B-1,A"はQBのキューヘッドの一つ前のデータをQAのキューヘッドの値を加えあわせその結果をQAのテールに書き込む命令である．" add A,B-1,A"の-1はキューヘッドからの相対位置を示す数値でこれをオフセットと呼んでいる．QTはデータがロードされたとき右に1移動し，QHはデータがアクセスされたとき右に1移動する．このプログラムでは"ld B, a"，"ld A,b"，"ld A,c"が同時に実行でき，また，"mul A,B,A"，"add A,B-1,A"が同時にできるので，4ステップでプログラムの実行が終わる．キュープログラムではこのように同時に実行できる命令は必ず連続して表れること，また問題の持っている並列性のすべてを表現できることと，オペランドが必要ないかまたは少量ですむので命令長が短くなるなどの特徴がある．　キュープロセッサではQHやQTの移動を命令で制御することもできる．移動をストップするとオフセット名をレジスタ名のように扱え，キューは実質的にレジスタとほぼ同じ動作をするようになる．この方法でアドレス修飾用レジスタ，スタックポインタなどを実現している．

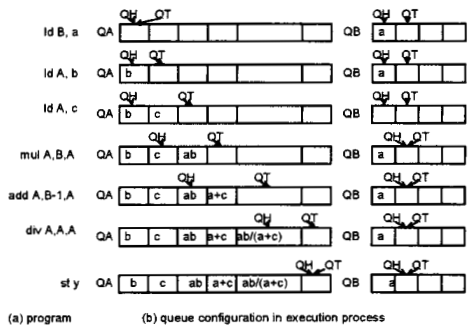

図1　y=ab/(a+c)を計算する2DQのプログラム

2.1 キューによるランダムアクセスレジスタの実現

キュープロセッサではQHやQTの移動を命令で制御することもできる．キューにはストップモードとい

うモードがあり, このモードではQH,QTを更新しない. このモードはキューをレジスタとして使用するために用意されている. キューをランダムアクセスレジスタとして使うにはqhBとqtBを同じ場所へセットする. そしてQBをストップモードへ設定する. qhB,qtBが固定されているのでオフセットがアドレスとなり, ランダムアクセスできる. 値のインクリメント, デクリメントをするにはincr命令 (ヘッド, テールに共通のオフセットを使用する即値加算命令) を使用する. 別のレジスタの値をコピーするための操作にはmove命令 (ヘッド, テールにそれぞれ別のオフセットが使用可能な命令) を使用する. こうすることでSPのインクリメント, デクリメント,FPへSPを代入する事が1命令で実行可能となる. 図2ではSPは3番地,FPは4番地にあるとし,qhB,qtBを8番地にセットしている. SP=SP+4としたい場合,incr B,-5,4とすればよい. またFP=SPとしたい時はmove B,-4,-5とすればよい. このようにキューを制御することでランダムアクセスレジスタを持たなくてもランダムアクセスレジスタのように使用できる.

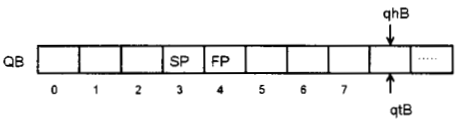

図 2 QB の使い方

3. 命令セット

2DQの命令セットの命令長はすべて2バイトであり,

- 条件分岐, 無条件分岐命令
- ロード, ストア命令
- 算術論理演算命令
- Move命令
- QH,QT操作命令
- NOPなどの特殊命令

の47命令よりなっている. 命令長は2バイト固定であるが基本処理単位は4バイトの32ビットである. 命令フォーマットを図3に示す. 命令フォーマットは図3の(a)〜(g)の7種類と特殊命令用フォーマットで計8種類である. この図でopcodeは命令指定部分で6ビット, qはキュー指定部で1から3ビットからなる. offsetは3から7ビット, address指定部は5から10ビットからなる. 命令にはaddやjmpなどの命令の他, キュー間のデータ転送を行うmvqqやキューレジスタの制御を行うmsqh, アドレス拡張を行うcovopなどのキュー独自の命令が含まれる.

3.0.1 算術論理演算命令

算術論理演算命令のフォーマットは(e)であり, レジスタ指定に3ビット, offset指定に7ビット用意されている. このオフセット部によりQHから127ワード離れた位置を指定できる. たとえばsub A,B, A+4はQBのヘッドの値とQAのヘッドから4つ手前の値を減じ, その結果をQAのテールに格納することを意味する. すなわち(qtA)← (qhB) -(qhA- 4)である.

(a) mvqq	operation(6)	offset0(4)	offset0(4)	q(2)	6/2/4/4
(b) ld,st	operation(6)	address(5)	offset(3)	q(2)	6/2/3/5
(c) jmp	operation(6)	address(6)	offset(3)	q	6/1/3/6
(d) msqh	operation(6)	sn(6)	s(2) c	q	6/1/2/1/6
(e) add,mul	operation(6)	offset(7)	q(3)		6/3/7
(f) branch	operation(6)	address or imm(9)		q	6/1/9
(g)covop,etc	operation(6)	address or function(10)			6/10

図 3 2DQ の命令フォーマット

3.0.2 ロード, ストア命令

ロード命令, ストア命令のフォーマットは(b)タイプである. ロード命令 ldq q2,ofst,addr は(qtqw)←m[(qhq-ofst)+4*addr] をあらわす. ここでldqやqhqのqはキューAかBを表し, ofstはオフセット, qhはキューヘッドを表す. qhq-ofstで示されるキューの内容に4*addr部を加算してできたアドレスのメモリの内容がメモリから取り出されキューテールに入れられる. qhq-ofstにより指定されるキューワードはアドレス修飾用レジスタの役割をしている. ストア命令 stqw q,ofst,addr は m[(qhq-ofst)+4*addr]←(qhq) と, キューヘッドの内容を(qhq-ofst)+4*addrで表されるメモリアドレスに格納する.

3.0.3 比較命令

比較命令 cpr q, uofst は (qhq) と (qhq-uofst) を比較してその結果の大小, 等しいをテール qtq に入れる. このシステムでは決まったコンディションコードレジスタを持たず, コンディションコードは演算結果と同じようにキューの1語に入れられる. これにより並列に比較命令が実行できる命令体系となっている.

3.0.4 条件分岐命令, 無条件分岐命令

ジャンプ命令 jmp uofst,addr は pc←(qhb - ofst) + 2*addr のように実質的なレジスタアドレス修飾によりジャンプする. このとき使うキューはQBに固定されている. addr部は6ビットと少ないが, メモリアドレス部はアドレス拡張命令コボップ covop addr1 により, 16ビットまで拡張できる. このように本命令セットでは命令のaddr部はcovoop命令で拡張できる. 分岐命令 blt q,addr はqhqに格納されているコンディションコードにしたがって, 条件分岐する.

3.0.5 Move 命令

キュー間でデータを移動する命令 mvqq q,ofstt,ofsth はキューテールからのオフセットを持った命令でキューヘッド近辺A(qhq-ofsth)の値をキューテールqtq-ofstt

表 1 2DQ の構成

QA,QB のワード長	33 bit
同時命令発行数	4
パイプライン段数	7
命令長	2byte 固定長
整数演算ユニット	4
ロードストアユニット	1
分岐ユニット	1
QA のサイズ	64 word
QB のサイズ	16 word

に移動する命令である.

3.0.6 QH,QT 操作命令

キューヘッドやキューテールなどのキューポインタ移動命令 msqh q1,s,c,sn はキューポインタを自動更新モードにしたりストップモードにしたり, また, 任意の数だけ移動させるための命令である. この命令により非常に柔軟なキュー使用が可能になる.

3.0.7 アドレス拡張命令

命令のメモリアドレス指定部は 5 から 10 ビットと少なめであるが, ここはアドレス拡張命令コボップ covop addr1 により, 10 ビット分拡張できる.

4. アーキテクチャ

現在設計中の 2DQ の構成を表 1 に示す. 2DQ は 32bit プロセッサで中間記憶に QA,QB の二つの循環キューを持つ. キューのエントリーはデータ 32bit とデータが利用可能かを示すレディビット 1bit の計 33bit からなる. そして QA,QB のサイズはそれぞれ 64 ワード,16 ワードで QH,QT はそれぞれ 6bit とする ($\log_2(64)$). キューのポートはそれぞれ読み込みポート数 8, 書込みポート数 4 としている.

2DQ のパイプライン構成を図 4 に示す. 各ステージでは

- IF ステージ:4 命令を一度にメモリから読み込む.
- ID ステージ:4 命令をデコードする.
- QCU:このステージは並列キューコンピュータ独得なものである. ここでは QH,QT を元に各命令のキューへの書き込み, 読み込みを行うキューアドレスを生成する. 命令実行後の QH,QT の値を計算し, 更新する.
- DISPATCH:命令間のデータ依存関係を調べ, 依存関係の無い命令を命令ウィンドウへ送る.
- ISSUE:命令ウィンドウの先頭から使用するデータと演算器が用意できている命令を複数個発行する.
- EXE:キューへアクセスしデータを読み込み, 実際に計算を行う.
- WB:実行結果をキューへ書き込む.

を行う. レジスタ名が無く, 並列実行可能な順序で命令が並ぶというキュープログラムの性質により,2DQ のアーキテクチャには

- ID ステージの際にレジスタリネーミング機構が必要ない
- 並列実行できる命令が順番に並んでいる (グループ ILP) ため,ISSUE ステージのインストラクションウィンドウが小さくて良い
- グループ ILP によりインオーダ実行で十分なのでリオーダバッファ機構が必要無い
- 逆従属関係が存在しないので実行可能命令はデータ従属性だけのチェックで十分である

という特徴があり, その分だけハードウェア量を削減できる.

4.1 QCU の設計

IF ユニット,ID ユニット,DISPATCH ユニット,ISSUE, EXE ユニット,WB ユニットは通常のプロセッサのそれらと大きな違いはない. ここではキューユニット特有の QCU ユニットについて述べる. QCU ユニットは

- キューアドレス生成
- QH,QT 更新

モジュールからなる QPC(Queue Pointer Calcuration) を 4 個接続したものである (図 5). QA,QB の QH,QT は QCU 内部で管理される. 図中の inst_info は命令の制御信号,qRegAddrs はキューへの読み書きを行う為のアドレスである. アドレス生成では QH,QT を元に命令がデータを読み書きするかを調べ, 読み書きが行われる場合は選択された QH,QT へオフセット値を使い, キューのアドレスを生成する. そして QH,QT の更新方法により QH,QT を更新をする.QH,QT の更新は次の章で述べる. 更新された QH,QT は次の命令が使用する QH,QT となる.

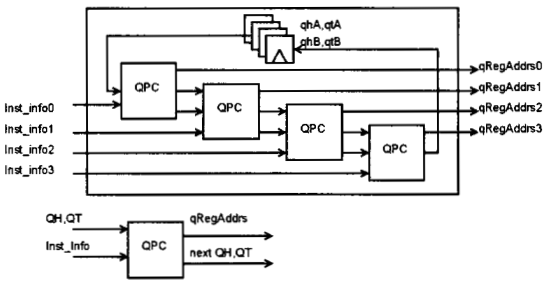

図 5 QCU の構成

4.1.1 QH,QT の更新方法

2DQ の QH,QT の更新方法について説明する. キューがストップモードの場合の場合,QH,QT は現在の値を維持する. QH を更新するためのフローチャートを図 6 に示す. QH の更新は以下の手順で行われる (図 6 内の番号に対応). 説明のためここでは第一オペランドで選択したキューは QX, 第二オペランドで選択したキューは QY とする.

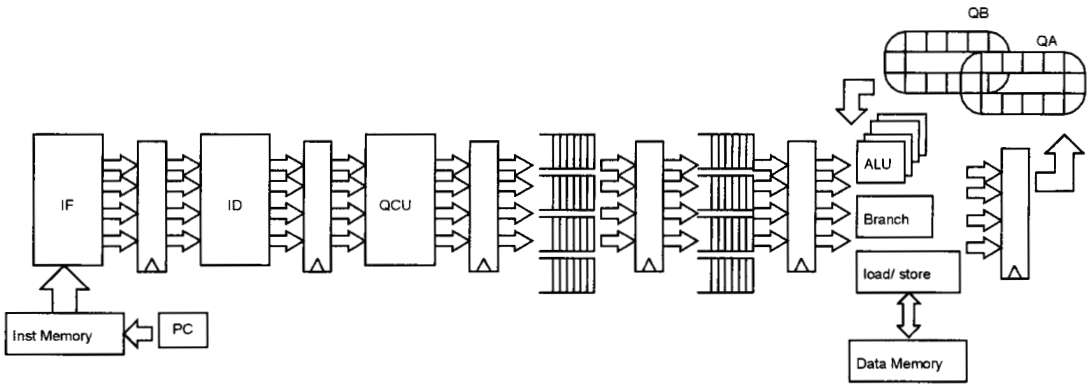

図 4　2DQ のパイプライン

(1)　キューがストップモードだったら現在の値を維持する. そうでなければ 2 へすすむ.
(2)　命令がデータを読み込むならば 3 へそうでなければ 4 へすすむ.
(3)　命令が二項演算ならば 5 へそうでなければ (一項演算の場合)6 へすすむ.
(4)　命令が QH 設定命令ならば qhx は即値 imm となる.
(5)　二項演算の読み込むデータが同じならば 7 へそうでなければ 8 へすすむ.
(6)　オフセットが 0 ならば qhx+1 そうでなければ現在の値を維持する.
(7)　オフセットが 0 ならば qhx+2 そうでなければ qhx+1 とする.
(8)　オフセットが 0 ならば第一オペランド, 第二オペランドで選択した qhx+1,qhy+1 とする. そうでなければ qhx+1 とする.

次に QT を更新するためのフローチャートを図 7 に示す. QT の更新は以下の手順で行われる (図 7 内の番号に対応).

(1)　キューがストップモードなら現在の QT を維持する. そうでなければ 2 へすすむ.
(2)　命令がデータを書き込むならば 4 へそうでなければ 3 へすすむ.
(3)　命令が QT 設定命令ならば QT は即値 imm となる.
(4)　オフセットが 0 ならば QT+1 そうでなければ現在の値を維持する.

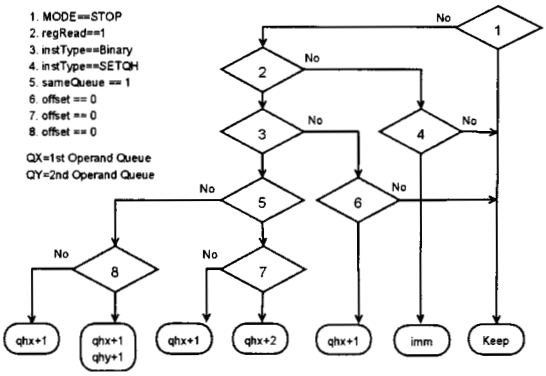

図 6　QH の更新方法

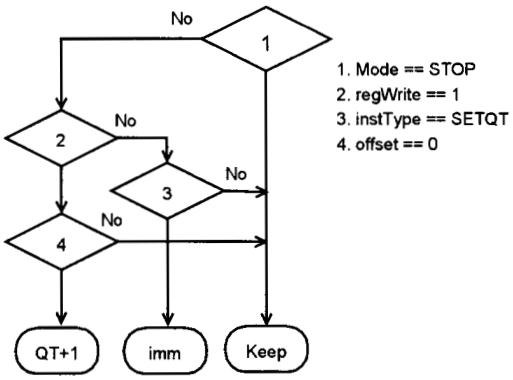

図 7　QT の更新方法

5. ま と め

キュープロセッサはプログラム中にレジスタ名が表れない, 単プログラム長, 並列がすべて含まれている, プログラム長の短い, 大並列実行性能を持った, ハードウエア複雑性の小さいプロセッサとして期待されている. また, 同時実行可能な命令が順番に並んでいる事から, 簡単な機構で並列性を抽出できることが知られている. 2DQ ではキューを実質的にレジスタのように制御することで, レジスタという概念をなくしたキューという概念だけで中間結果を格納するプロセッサとなっている. 2DQ にはこれまで培われたキュープロセッサの技術がすべてそそぎこまれており, そ

れだけに実用に近づいたプロセッサであるといえる．本論文では 2DQ の原理, 命令セット, アーキテクチャと QCU について述べた．実際にハードウェア記述言語の Verilog-HDL を使用して QCU を設計した結果,LogicELement 数は約 1700, クリティカルパスは 80.7[nsec] となった．現在 2DQ はこの命令セットをこのアーキテクチャをもとに残りの部分をハードウェア記述言語で実装中である．非常に近い将来全貌について紹介できるよう，現在全力をかけて開発を行っている．

参 考 文 献

[1] Kenneth C. Yeager, "The MIPS R10000 Superscalar Microprocessor", IEEE Micro, v.16 n.2, p.28-40, April 1996

[2] Subbarao Palacharla, Norman P. Jouppi, J. E. Smith, "Complexity-effective superscalar processors", Proceedings of the 24th annual international symposium on Computer Architecture, p.206-218, June 01-04, 1997, Denver, Colorado, United States

[3] B. A. Abderazek, M. Arsenji, S. Shigeta, T. Yoshinaga, M. Sowa, "Queue Processor for Novel Queue Computing Paradigm Based on Produced Order Scheme", Proc. of HPC, IEEE Computer systems press, 0-7695-2138-X/04, pp. 169-177, July 2004.

[4] M. Sowa , et al. "Parallel Queue Processor Architecture Based on Produced Order Computation Model", Journal of Supercomputing, ,Vol. 32,No. 3, pp. 217-229, June 2005.

[5] M. Sowa, B. A. Abderazek, S. Shigeta, K. Nikolova, and T. Yoshinaga, "Proposal and Design of a Parallel Queue Processor Architecture (PQP)", 14th IASTED International Conf. on Parallel and Distributed Computing and Systems (PDCS), Cambridge, USA, 2002

[6] Halcham kutluk, B. A. Abderazek, S. Shigeta, T. Yoshinaga and M. Sowa, "並列キュー計算モデルの基本特性評価", 電子情報通信学会, IEICE 信学技報, CPSY2004-15 p. 37-42, (2004-07)

[7] 石崎賀昭, Pavel Boytchv, 吉永努, 曽和将容, "GCC によるキューコンパイラ開発手法の提案", FIT2005(2005)

[8] 曽和将容, "マルチディメンジョナルキュープロセッサ", 特願 50600277736, 2006 年 2 月 14 日

[9] M. Sowa:"Over Queue 2DQ Produced Order Multi-Dimensional Parallel Processor Based on Produced Order Keeping Queue Computing",Sowa Lab. Technical Report SLL060560(2007)