論　文

範囲検査アドレッシングを有するFLATS2アーキテクチャと性能評価

正　員　市川　周一[†]　　非会員　佐藤　三久[†]　　非会員　後藤　英一[††]

The Evaluation of Range-Checking Addressing Modes and the Architecture of FLATS2

Shuichi ICHIKAWA [†], *Member*, Mitsuhisa SATO [†], *Nonmember* *and* Eiichi GOTO [††], *Nonmember*

あらまし　FLATS2は，資源共有型MIMDの一種である循環パイプライン方式の計算機である．本論文では，FLATS2の命令セット設計とその実現について述べ，配列処理等について効率的な処理が可能であることを示す．更にFLATS2での評価結果を示し，考察を加える．FLATS2の命令セットは，既存のスカラ命令の拡張である．FLATS2のアドレシングモードには範囲検査機能が付加されており，更に検査結果に応じて分岐を行う機能がある．アクセスアドレスが指定範囲内にある場合，実際にメモリオペランドに対して演算が実行され，指定番地への分岐が発生する．範囲外であれば，分岐せず次命令へ進む．これらすべての操作を1命令で実行することによって，数値計算や記号処理において配列処理を効率良く実行することができる．例えば，ループによる繰返し処理においては，ループ制御の時間をメモリアクセスや演算の実行に重畳させて隠すことができる．結果的に，算術演算器には連続的にデータが供給され，高い数値演算性能が実現される．更にFLATS2では，複合命令によって1命令当り複数の演算処理を行うことを許した．これによって，更に演算性能が改善される．

1. まえがき

FLATS2は，数値処理や記号処理などメモリ上の配列を扱う応用に適した命令セットをもつ，循環パイプライン計算機である．循環パイプラインは，ジョセフソン素子等のラッチ機能つき論理素子に適したアーキテクチャとして提案された，一種の資源共有型MIMD方式である[(1)]．筆者らは，循環パイプライン方式の有効性を確認するため，既存のシリコン技術を用いて循環パイプライン方式の計算機FLATS2を設計し，実現した[(2),(3)]．本論文では，FLATS2の命令セットとその実現を概観し，数値計算応用での配列処理についてその有効性を評価する．

循環パイプライン方式では，複数の命令流を用いることによって，多段のパイプラインから高いスループットを引き出そうとする．しかし各命令流の性能を改善するためには，これとは別の方法を用いる必要がある．FLATS2でも，データのバイパス等の先行制御を用いて命令流内での命令の重畳度を上げ，命令流当りの性能を改善している．更にFLATS2では，命令レベルの並列性を利用するために，BLアドレシングという方式を命令セットに導入した[(4)]．BLアドレシングは通常のアドレシングモードの拡張で，計算された実効アドレスが命令で指定した範囲内にあるかどうか検査し，検査結果に応じて条件分岐を行う．これによって，オペランドのフェッチ，データの演算，条件分岐などを1命令で指定することができる．

配列処理に関しては，従来，専用アーキテクチャとしてベクタ方式が採用されているが，これは配列全体に対して均一な処理を行う場合に適する方式である．一方BLアドレシングは，基本的にスカラ命令であり，ベクタ方式より応用範囲が広い汎用の方式であるため，ループ制御によらない非均一な処理にも適用できる．また，ループ内でアクセスする配列要素間にデータ依

† 新技術事業団，東京都
Research Development Corporation of Japan, Tokyo, 158 Japan
†† 東京大学理学部情報科学科，東京都
Faculty of Science, The University of Tokyo, Tokyo, 113 Japan

存性がある場合でも，本方式ではスカラ処理の延長であるためプログラム上特別な工夫を必要としない．更にBLアドレシングでは，ベクタレジスタ等を用いずにメモリ上のデータを直接扱うので，配列長によらず同じ手続きで処理が可能である．FLATS2の場合，ベクタ化で利用できる並列性は，既に循環パイプラインによるMIMD化で使用している．

スカラ処理と配列処理を統合する試みとしては，N. P. Jouppiらによる MultiTitan の Unified Vector/Scalar Floating Point Architecture[5] がある．これは，FPU演算命令にベクタ長フィールドを設け，FPUレジスタファイルをスカラ処理とベクタ処理で共用するアーキテクチャである．Unified Vector/Scalarでは，スカラ命令とはベクタ長1のベクタ命令のことになる．ベクタ処理をスカラ処理に融合することにより，スカラ命令と同じレジスタインタロック機構を利用して，ベクタ命令を実装することができる．結果的にベクタ化できる応用が広がり，多くの応用で性能が改善できると言う．

Unified Vector/ScalarとBLアドレシングは，拡張されたスカラ命令を用いて配列処理を高速化するという点では共通しているが，その実現には大きな違いがある．MultiTitanのFPU演算命令は結局ベクタ命令を採用したため，配列の各要素に対して同じ処理を行う場合にしか適用できない．また，オペランドがFPUレジスタに限られるため，各命令で最大16要素までしか処理することができない．また，命令で扱う配列長は，命令のベクタ長フィールドで静的に定まっている．不連続なアドレスに対するアクセスも，スカラ的にしか処理できない．これに対してBLアドレシングは，Lispのようなベクタ化できない処理にも利用でき，メモリに対する処理なので配列長の制限もなく，オペランドのアドレス計算についても自由度が大きい．結局，Unified Vector/Scalarはベクタ命令であるが，BLアドレシングはスカラ命令であるという違いがある．

演算の結果に対して範囲検査を行う方式としては，VAX-11[6] のINDEX命令(Compute Index)などが知られている．しかし，INDEX命令では検査結果をユーザが使用することはできない(例外/割出しが発生する)．一方BLアドレシングでは，範囲検査の結果を利用して分岐を行うことができるため，ユーザプログラムで検査結果に応じて処理を変更することができる．このため，ループの制御からLISPでの型判定にまで広く応用することができる[4]．VAXのACB命令(Add, Compare, and Brancc)等は，演算と比較，条件分岐を1命令で行うが，アドレス計算やメモリアクセスは行わない．FLATS2では，2メモリオペランド型の命令セットを採用して，メモリオペランドのフェッチと演算までを1命令で実行することによって，命令内で指定する処理の量を増やしている．

循環パイプライン計算機ではパイプラインを複数の命令流が共有しているため，命令実行時にパイプライン内で動的に命令処理の重畳度を上げるより，1命令内で静的に多くの処理を指定する方が実装上有利である．その意味で，BLアドレシングは循環パイプラインに適した方式であると言える．しかし，BLアドレシング自体は汎用的な方式なので，通常のCPU-FPU構成にも適している．FLATS2自体も，CPU-FPU構成で集積化できる設計になっている．

本論文では，特に数値演算応用などの配列処理について，FLATS2の方式を評価する．以下，2.でFLATS2のアーキテクチャを概観し，3.でFLATS2における配列処理の実現方法を説明する．4.では，いくつかのベンチマークテストを用いてFLATS2の方式を評価する．

2. BLアドレシングとFLATS2のアーキテクチャ

本章では，最初にBLアドレシングの概念について説明し，次にFLATS2命令セットにおける実現を述べる．

2.1 BLアドレシング

BLアドレシングは，通常のアドレッシングモードを拡張して，範囲検査機能と条件分岐機能を追加したものである．BLアドレシングでは，オペランドの実効アドレスを計算した後，実効アドレスが命令で指定した範囲内(BaseとLimitの間)にあるかどうかを検査し，

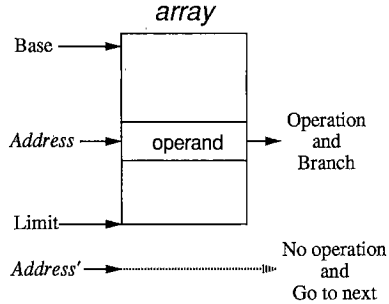


図1 BLアドレシングの概念
Fig. 1 BL addressing.

検査結果に応じて条件分岐を行う．すなわち，アドレスがBaseとLimitの間にあれば分岐を実行し，必要があれば副作用としてレジスタの値を更新する．アドレスが指定範囲外だった場合，その命令の実行を中断して，分岐も行わない(次の命令を実行するかトラップする)．図1に，BLアドレシングの概念を示す．

配列処理プログラムでは，BLアドレシングを用いてループの終了判定を行うことができる．配列の下限アドレスと上限アドレスをそれぞれBaseとLimitに使用し，アクセスするアドレスが配列内であるか否かで分岐する．アドレシングの副作用でアクセスするアドレスを順次変えると同時に，配列要素のアクセスとループの終了判定，そして分岐を1命令で同時に行うため，効率の良い処理が可能になる．これについては，以下でFLATS2のアーキテクチャを述べた後，具体的な例も示しながら詳しく説明する．

更に，BLアドレシングを用いることによって，配列添字の実行時検査を容易かつ効率良く実行することができる．通常の計算機では，配列添字の実行時検査のためにコンパイラが検査用のコードを生成するが，そのためにコードサイズが膨張し実行時間が増加してしまう．しかしBLアドレシングでは，あらかじめ配列の上限アドレスと下限アドレスを用意しておくことによって，性能に影響を及ぼすことなく配列添字の検査を行うことができる．配列添字に関するプログラムミスは非常に多く発生するので，これを防ぐことはプログラムの信頼性を向上し，デバッグの労力を軽減する上で効果が大きい．

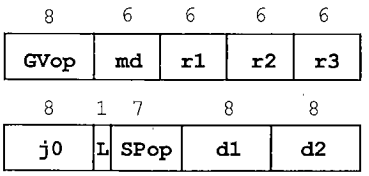
2.2 FLATS2の命令形式

FLATS2の命令では，1命令でメモリオペランドを二つまで指定することができる．更に，BLアドレシングを用いて指定したオペランドに対して，演算処理を1命令サイクルで行う．従って，FLATS2では1命令で以下のような操作を指定することができる：

(1) オペランドの実効アドレス計算，
(2) 実効アドレスの範囲検査と分岐，
(3) アドレシングの副作用の書込み，
(4) オペランドのフェッチ，
(5) 演算の実行．

FLATS2の命令セットには四つの基本命令形式がある(それぞれI/J/K/M形式と呼ばれている)が，BLアドレシングを用いる演算命令はM形式をとる．図2に，そのM形式の命令を示す．図中，GVop (GVU operation) とmd (mode) はオペランドのアドレシングモードを指定するフィールドである．同様に，SPop (SPU operation) はオペランドに対する演算の種類を指定するフィールドで，整数/浮動小数点演算器に対するオプコードになる．r1〜r3はアドレス計算に使うレジスタ番号，j0はBLアドレシングで使う分岐オフセット．d1とd2はアドレス計算またはオペランドに使用されるイミディエイト値(displacement)である．Lは長形式の命令を意味するフラグで，Lが1の場合は命令の後に各32ビットのイミディエイトD1とD2が続き，d1とd2の代わりに使用される．

M形式で使用できるアドレシングモード(実効アドレス)を表1に示す．アドレシングモードでは，BLアドレシングのBaseレジスタ番号だけを指定する．Limit



GVop : GVU operation,
md : operand select mode,
r1,r2,r3 : GV register no.,
j0 : branch offset,
L : long format flag,
SPop : SPU operation,
d1,d2 : displacements.

図2 M形式命令
Fig. 2 M format instruction.

表1 FLATS2のアドレシングモード

記法	実効アドレス	副作用
#imm	イミディエイト	
grn, vrn	汎用レジスタ	
S,P,T,Q,R,U	浮動小数点レジスタ	
b:imm(p)	$p + imm$	
b:>imm(p)	$p + imm$	$p = p + imm$
b:<imm(p)	p	$p = p + imm$
b@x	$b + x$	
b@>x	$b + x$	$b = b + x$
b@<x	b	$b = b + x$
b@imm	$b + imm$	
b@>imm	$b + imm$	$b = b + imm$
b@<imm	b	$b = b + imm$

表中，*imm*はイミディエイト値，*n*はレジスタ番号(数字)．*b,p,x*は汎用レジスタ．

レジスタには，Baseレジスタの偶奇ペア†が使用される．

2.3 FLATS2における配列処理の例

ここで，FLATS2の命令を用いて配列の処理を行う方法を，例を挙げて示す．M形式命令では，例えば次のような命令が使用できる．ここで，vrは汎用レジスタ，Sは浮動小数点演算器のレジスタである．

```
add3. f. j vr0 : 4<(vr2), S, S, L001
```

"vr0 :"は，vr0がBaseアドレス，vr1がLimitアドレスであることを示す．"4<(vr2)"は，vr2の内容がアクセスするアドレスで，更に命令実行後には副作用としてvr2に4を加えることを表す．vr2の値がvr0とvr1の値の間にあれば，メモリがアクセスされ，メモリ上の単精度浮動小数点数とSレジスタの値が加算されて(add3.f)，演算結果がSレジスタに書き込まれる．更に，アクセスが成功した場合，vr2に副作用として4を加えてから，ラベルL001へ分岐する．アクセスが失敗すると分岐は発生せず，次の番地の命令を実行する．

この命令を使って，配列の総和を計算することができる．以下に，配列arrayの総和を計算するプログラムを示す：

```
        mov. f      # 0f0. 0, S
                            ; Sの初期化
        lea         @array, vr0
                            ; Baseの初期化
        lea         @array+END, vr1
                            ; Limitの初期化
        movw        vr0, vr2
                            ; Pointerの初期化
L002 : add3. f. j vr0 : 4<(vr2), S, S, L002
                            ; ループ本体(自己ループ)
```

レジスタを初期化した後でループ本体を実行し，vr2を副作用で変更しながら総和を計算して，vr2が配列外に出たらL002の次の命令に進む(ループの脱出)．このループでは，ループ本体の演算実行とループの終了検査および条件分岐が，すべて1命令内に統合されている．FLATS2は，この命令をパイプライン的に処理することによって，1命令サイクルに1回ずつこのループを実行する．

この例と同様の方法で，FLATS2は多くの典型的な配列処理を効率よく実現することができる．4.で，FLATS2命令セットを用いてLinpack (BLAS)やLivermoreループを評価した結果を示す．

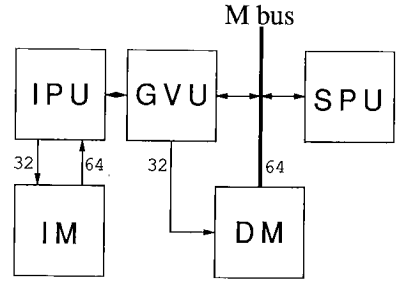
3. FLATS2の実現

3.1 基本構成

FLATS2 CPUの基本構成(概念図)を，図3に示す．CPUは，命令のデコードと順序制御を行う命令処理ユニット(IPU)，汎用レジスタ間の整数演算とアドレス計算を行うGVユニット(GVU)，メモリとレジスタ間の整数/浮動小数点演算を行うSPユニット(SPU)の三つのユニットからなる．命令は命令メモリ(IM)に，データはデータメモリ(DM)に格納され，それぞれ図3のようにIPU, GVU, SPUに接続されている．IM, DMは1マシンサイクルにつき64 bitのデータを転送することができる．すなわち命令の転送幅は1.23×10^6 Byte/s, M busの転送幅も同じく1.23×10^6 Byte/sである．FLATS2の主要緒元を表2に示す．

3.2 GVU(アドレス演算部)の構成

2.で述べたような命令動作を実現するために，GVUには範囲検査用のハードウェアが組み込まれている．GVUのブロック図を図4に示す．図中のReg1, Reg2,



IPU : Instruction Processing Unit,
GVU : Gloval/Variable register Unit,
SPU : Sum and Product Unit,
IM : Instruction Memory,
DM : Data Memory.

図3 FLATS2 CPUの基本構成
Fig. 3 Block diagram of FLATS2 CPU.

表2 FLATS2主要諸元

マシンサイクル	65 ns
語長	32 ビット(データ)＋1 ビット(タグ)
実記憶容量	5.5 MB (現状)
論理素子	ECL10K, 10KH, 100K, FAST
基板	18.3 inch × 12.7 inch (26枚)
筐体	960(W) × 1200(H) × 1075(D)

† レジスタ番号の下位1ビットを反転した番号のレジスタ．例えば，3番に対しては2番，10番に対しては11番．

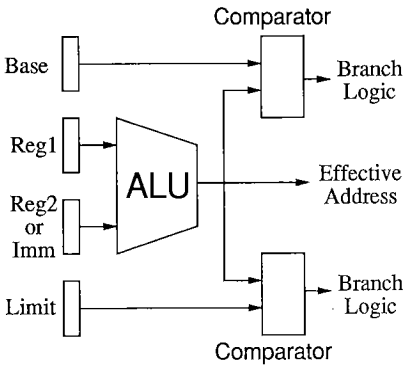


図4　GVUの構成
Fig. 4　Block diagram of GVU.

Base/Limit対には，M形式命令のr1～r3で指定される，GVUの汎用レジスタが使用される．Immediateは，命令で指定するイミディエイト値である(M形式のd1, d2またはD1, D2)．ALUは32 bitの固定小数点ALUで，アドレス演算と汎用レジスタのレジスタ間演算に使用される．

図4からもわかるように，FLATS2のアドレシングでは，実効アドレスにReg1, Imm, Reg1＋Imm, Reg1＋Reg2のいずれかを使用する．このため，FLATS2では配列アクセスで定数添字も変数添字も同じように扱うことができる．また，ループ制御において，インデックスの増分が定数であっても変数であっても，BLアドレシングをループの脱出判定に用いることができる．

ALUの出力は，実効アドレスの範囲検査のために比較器でBaseおよびLimitと比較され，比較結果がIPU内の分岐制御論理に通知される．FLATS2のような命令仕様の場合，分岐を1命令サイクルで実行する方法が問題となるが，FLATS2ではアドレス計算と範囲検査の遅延時間が他の命令流の実行時間に吸収されてしまうため，特別な工夫は必要ない．単一命令流内では，分岐方向が決まった後で分岐先の命令をフェッチしている(**3.4**参照)．

BLアドレシングではアクセスするアドレスの範囲を検査するが，範囲検査用のハードウェアはそれ以外の命令にも利用できる．例えば，VAX-11のACB (Add, Compare, and Branch)命令のような演算型分岐命令は，BLアドレシングで用いるハードウェア資源をそのまま利用して実現することができる．FLATS2でも，このような演算型分岐命令を実装することによって，BLアドレシングの分岐制御を利用できない場合にも，ループ制御等を効率良く実行できるよう配慮した．

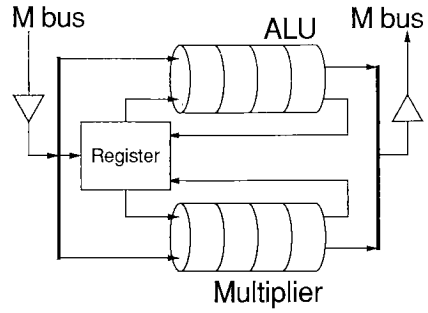


図5　SPUの構成
Fig. 5　Block diagram of SPU.

3.3　SPU (浮動小数点演算部)の構成

SPUは，メモリオペランドに対する整数/論理演算と浮動小数点演算を実行する演算ユニットである．SPUのブロック図を図5に示す．SPUは，図3，図5でもわかるように，メモリからのデータバス(M bus)に接続された付加プロセッサの形式をとる．M busの転送速度は，1基本クロックにつきデータ64 bitである．

SPU内部には，それぞれ4段にパイプライン化されたALUおよび乗算器と，レジスタファイルがある．ALUおよび乗算器はそれぞれ64ビット幅で，整数/浮動小数点演算を処理する．レジスタファイルは仮数部(整数部)64ビット，タグ/指数部が16ビットで，演算器のオペランドとして働く．演算器は，M busからのデータまたはレジスタ上のデータを用いて演算を行い，レジスタまたはM busに結果を転送する．

ALUと乗算器は，それぞれ1基本クロックにつき一つ演算を行うことができる．更に，演算器がレジスタ間演算を実行しているときは，演算と並行してM busからレジスタファイルにデータを転送することができる．このように，SPU内で複数の処理を並行して行うことによって，数値演算を効率良く実行することができる．以上のように複数の操作をSPUに指示する命令を，FLATS2では複合命令と呼ぶ．

3.4　FLATS2のパイプライン制御

FLATS2では，M形式の命令は10段にパイプライン化されて処理される．図6に，その場合のパイプラインチャートを示す．IMR, GVR, DMRは，それぞれIM，汎用レジスタ，DMを読み出すサイクルである．IPUは命令のデコードサイクル，GVUは，GVUでの整数/アドレス演算サイクルである．SPUでの演算処理は，**3.3**でも述べたように4段にパイプライン化され

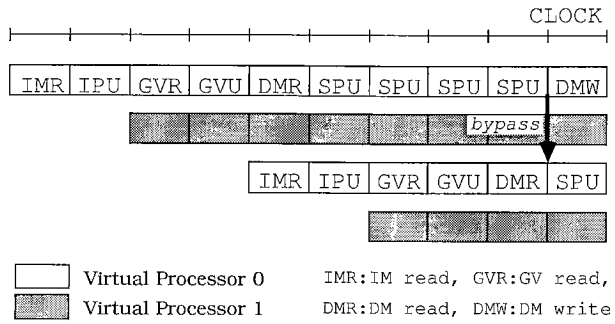


図6 FLATS2のパイプラインチャート
Fig. 6 Pipelining of virtual processors.

ている．DMWは，演算結果をDMに書き込むサイクルである．

FLATS2は循環パイプライン方式の計算機であるから，パイプラインは複数の命令流(仮想プロセッサ；Virtual Processor)によって共有され，各命令流の命令は4基本サイクルに一つずつパイプラインに投入される．従って，先行する命令がGVUでBLアドレシングの範囲検査を終了した後，次の命令がIMRフェーズでフェッチされる．このためFLATS2では，BLアドレシングに伴う分岐制御でも，予測分岐等の方法は用いていない．

しかし，循環パイプライン以外の方式でも，先行制御によってBLアドレシングを効率良く実現することができる．以下に，その理由を述べる．

M形式では基本的にj0フィールドの指定する番地に分岐するので，分岐予測が容易にできる(常にj0の指す命令に分岐すると仮定できる)．また，多くの場合j0フィールドは次命令を指しているので，命令のプリフェッチが有効に働く．更に，配列処理などで小さなループをM形式で実現する場合，飛び先がキャッシュまたはバッファ内にある可能性が高いので，ジャンプによるオーバヘッドは小さいと考えられる．一方，ループが大きい場合は，ジャンプのオーバヘッドが実行時間内に占める割合が小さくなるため，実行時間に及ぼす影響も小さくなる．BLアドレシングがメモリアクセスと並行して範囲検査を行うことを考えれば，範囲検査と分岐の遅延時間はデータメモリのアクセス時間に重畳するため，実行時間に与える影響は少ないと言える．

FLATS2のパイプライン制御については，文献(2)，(3)に報告済みであるので，そちらも参照して頂きたい．

4. 実装と評価

4.1 FLATS2の実装

FLATS2は市販のSSI/MSI(主としてECL 10K/100K)を用いて実装されており，現在クロックサイクル65 nsで動作している．メモリにはCMOS-SRAMが5 MBほど実装されており，更にDRAMを用いた拡張メモリが計画されている．ほぼA3大の多層基板にはチップが各200～400個実装されており，これらの基板(計26枚)がバックプレーンに接続されて，1 m×1 m×1 m程度の筐体に納められている．LSIなどの集積技術を使用していないために，物理的に大きくなってクロックにも制限が出ているが，これはやむを得ないであろう．アーキテクチャ的には，CPU-FPU構成で集積化できるような構成になっている．

現在ソフトウェア環境として，CとFORTRANのクロスコンパイラとライブラリー，更にローダを含む簡単な実行環境が実機上で稼動している．また，LISPシステムとオペレーティングシステムが近々稼動する予定である．

4.2 仮想プロセッサのスカラ性能

FLATS2の仮想プロセッサを一つだけ用いて，いくつかのベンチマークテストを行った結果を表3に示す．比較のために，他のプロセッサでの結果も併せ示す．

表中，Clockの項は公称のクロック周波数であるが，この項はプロセッサごとにクロックの定義が異なっているため直接比較することができない．MIPSの項は公称MIPS値である．

Dhrystoneの項には，C言語版ver. 2. 1の結果を示す(ver. 1. 1ではFLATS2は6.5 KD/sであった)．得られた結果は，ほぼ公称MIPS値に比例している．Dhrystoneは整数演算や文字列操作を主としたベンチ

表3 ベンチマークテストの結果

アーキテクチャ CPU + FPU	クロック MHz	公称 MIPS	Dhrystone Kdhry/s	Whetstone Mwhet/s	Linpack MFLOPS
68020 + 68881	20	3	4.3	1.2	0.12
FLATS2 (1 proc.)	7.7	3.6	6.0	3.1	2.5
SPARC + Weitek	16.7	10	19.	3.9	1.1
R2000 + R2010	16.7	12	25.	9.0	1.8

表4 LINPACKサブプログラム(BLAS)

名前	意味	cycle/loop	FLOP/cycle
dasum	$\sum \lvert x_i \rvert$	2	1
daxpy	$y_i = y_i + x_i$	2	1
dcopy	$y_i = x_i$	1	1
ddot	$\sum x_i y_i$	1	2
drot	2次元の回転	6	1
dscal	$x_i = a x_i$	1	1
dswap	x_i と y_i の交換	3	2/3
idamax	max $\lvert x_i \rvert$ のiを返す	3 ～ 5	2/3 ～ 2/5

マークであるが，FLATS2の実現では，バイトのロード/ストア命令を実行するのに各2命令サイクルを要するため，若干FLATS2には不利である．

Whetstoneの項は，C言語版の倍精度whetstoneの結果を示す．Whetstoneでは三角関数や指数・対数関数を多く使用するが，これらの関数はハンドコーディングでFLATS2の数値演算用命令を使用して最適化してある．これらの関数をCコンパイラでコンパイルした場合には，性能が12％ほど低くなった．

FLATS2では，MIPS値に対するWhetstone値が高い．これは，FLATS2の浮動小数点演算の遅延時間が他の仮想プロセッサに重畳して隠れるため，各仮想プロセッサの1命令サイクルごとに1浮動小数点演算が可能だからである．

4.3 Linpack (BLAS)

Linpackで使用する最下位ルーチンの集まりを，BLAS (Basic Linear Algebra Subprograms) と呼ぶ．これらのルーチンは，小さなループを主体とした単純な配列処理を行う．FLATS2の命令セットでは，2.でも例を示した通り，BLアドレシングを用いてこれらのループを非常にコンパクトかつ高速に実現することができる．表4に，BLASの最内周ループがそれぞれ何命令サイクルで実現されるかを示す．更に，それぞれのループ内でサイクル当りいくつの浮動小数点演算が行われているか(浮動小数点演算密度)を示す．

表4からわかるとおり，多くのルーチンで演算密度が1になっている．これは，BLアドレシングによって，ループの終了検査と条件分岐，オペランドの供給などが，演算自体に完全にオーバラップして実行されることを示している．内積計算のルーチンddotで演算密度が2になっているのは，複合命令である内積命令(rip命令)を用いてALUと乗算器を並列に使用しているためである．idamaxで演算密度値に幅があるのは，最内周ループ内に条件文が含まれるためである．BLアドレシングはスカラ命令の拡張であるので，条件分岐を含むループにも適用できる．また，BLアドレシングでは増分が変数である場合にも，定数の場合と全く同じ性能が得られるという利点がある．

BLアドレシングを用いて人手で最適化したBLASによってLinpackベンチマークを実行した結果は，2.5 MFLOPSであった．ここでも，MIPS値(3.5 MIPS)に比べてMFLOPS値がかなり高いことが確認された．

Livermore Kernelは24あるが，ここでは先頭の14(原型リバモアループ)について，FLATS2で実行した結果を示す．

まず，各kernelの最内周ループをBLアドレシングを用いてハンドコーディングで最適化した結果を，図7にまとめる．縦軸は演算密度(FLOP/cycle)で，横軸がkernelの番号である．演算密度は，サイクル時間で割ることによって演算速度に換算できる．現状のFLATS2では，演算密度1は3.8 MFLOPSに相当する．

ここで，FLATS2の命令セットの効果を確かめるために，使用する命令を制限しながら以下の3段階に分けて評価した．

(1) 基本命令セット(basic)

FLATS2の通常命令とBLアドレシングだけを使用した場合．すなわち，メモリオペランドは読出し一つ，書込み一つまでを使用し，浮動小数点演算は1命令で一つだけ行う場合．

(2) 内積命令を付加した場合(+rip)

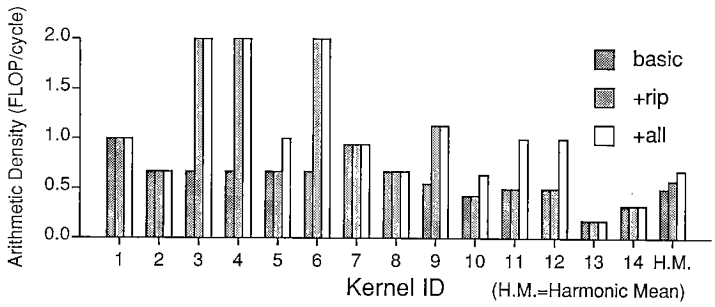


図7 最内周での演算密度
Fig. 7 Peak arithmetic density.

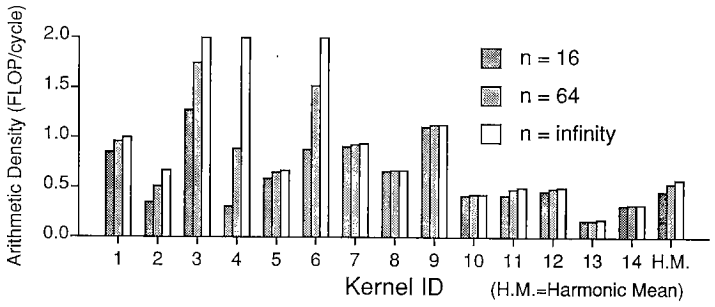


図8 実測された演算密度
Fig. 8 Measured arithmetic density.

(1)に加えて，複合命令であるrip命令と，メモリオペランドの先行ロードを許す場合．rip命令は，SPUの乗算器出力をALU入力にチェインして加算を行う命令で，1命令で二つの浮動小数点演算(乗算+加算)を指定する．また，メモリの先行ロードを指定するアドレシングモードは，次の命令で使用するメモリオペランドのロードを指示して，ロード操作を浮動小数点演算に重畳して行う．

(3) 性能上限(+all)

(2)に加えて，現在のFLATS2ハードウェアを変更することなく，マイクロコードの変更のみで対処可能なすべての命令をLivermore用に実装した場合の結果．これは，現状のFLATS2ハードウェアの性能上限の目安になる．

図7からわかるように，Livermore Kernelsでは，rip命令+先行ロードで大きな改善が見られるが，それ以外の命令はすべて用意しても，+ripと同程度の向上である．また，rip以外の特殊な命令については，Livermoreループから一般的に適用可能な命令は発見できなかったので，現状では実装していない．その意味では，+ripの項が現状のFLATS2の実現を表している．以降の評価はこの条件のもとで行った．

次に，FLATS2命令セットでのループ初期化のコストを評価するため，実際に繰返し回数(n)を変えながらループを実行し演算密度を測定してみた．その結果を図8に示す．[n=infinity]の項は，図7の+ripの項と同じである．nが十分大きいときは，最内周ループの効率が全体の結果を支配するので，[n=infinity]の項が実測され得る性能上限と考えられる．

図8からわかるように，FLATS2ではループの繰返し回数が減っても，性能の低下は緩やかである．すなわち，ループの初期化コストが小さい．これは，BLアドレシングがスカラ命令の拡張であることと，循環パイプラインが命令の遅延時間によるオーバヘッドを隠すためであると考えられる．結局，14の原型リバモアループでの演算密度の調和平均は，それぞれ0.58(n=∞), 0.54(n=64), 0.46(n=16)と極めて高い値を示した．これを性能に換算すれば，n=64で2.1 MFLOPSとなる．これはFLATS2のMIPS値に比べてもかなり高い値である．Linpackより複雑で，より現実の応用に近いLivermore Kernelでもこのように高い効率が得られるという結果から，本方式の応用範囲が非常に広いことが確認できた．

5. 考　　察

5.1 VLIW，スーパスカラとの対比

近年では，複数の操作を1サイクル内で実行することによってプロセッサの性能を向上させるという試みが，いろいろな形で行われている．その一つがスーパスカラ(Superscalar)やスーパパイプライン(Superpipeline)[7],[8]であり，またVLIW(Very Long Instruction Word architecture)[9],[10]と呼ばれるものである．これらは，複数の操作を1命令サイクル内で実現して性能を改善するという意味では，BLアドレシングと同じ目的をもつ．しかし，BLアドレシングが限定された適用範囲をねらって設計されているのに対し，これらのアーキテクチャは非常に汎用的な応用を指向している．スーパスカラやVLIWでは汎用的なハードウェア資源(ALUなど)が並行して使用可能で，しかもそれらの組合せに命令レベルでの制限が基本的にない．ハードウェアが割当て可能であれば，その操作は他の操作と並行して実行される．これに対してBLアドレシングでは，並行して行える操作の種類が固定されている(範囲検査と条件分岐など)．しかも命令レベルで「BLアドレシングモード」や「演算型条件分岐命令」として固定的に提供されているパターンでしか並行操作が実現できない．

しかしながら，いかに命令レベルで汎用的な並行操作が実現されたとしても，それが現実的に十分利用されなければ意味はない．SohiとVajapeyam[11]は，VLIWの命令セットにおいて，必ずしも多数の操作を完全に直交した形で提供する必要はないことを示した．比較的少数の操作を限定された命令セットで提供しても，大差のない性能が得られると言う．BLアドレシングのような限定的な手法は汎用的な手法に比べると実現コストが安いが，実際にはBLアドレシングで多くの応用について十分な性能を実現できる．

例えば，TRACE 7/200は命令長が256 bitで七つの操作が並列実行できるVLIW[10]である．FLATS2ハードウェアと同じ基本クロック(65 ns)で動作し，データの転送幅はFLATS2と同じ，命令の転送幅はFLATS2の2倍である．FLATS2よりもはるかに多くの汎用的並列操作機能を提供しているが，Linpackの性能は6.0 MFLOPSである．一方FLATS2では，単一仮想プロセッサでBLアドレシングを用いると2.5 MFLOPS，更に仮想プロセッサ二つを用いて並列化することにより5.0 MFLOPSの性能を発揮する．

表5　コードサイズ(byte)の比較

アーキテクチャ	Dhrystone	Whetstone	Linpack	Livermore
68020	2868	2600	9316	13676
FLATS2	6000	5288	18288	24712
比	2.1	2.0	2.0	1.8

Linpackより複雑なLivermore Loopを評価すると，TRACEが2.3 MFLOPSであるのに対して，FLATS2は最大性能の半分にあたる単一仮想プロセッサだけを使用して1.1 MFLOPSであった(まだ並列化による評価は行われていない)．

このように，BLアドレシング方式を実装したFLATS2は限定的な並列操作だけを命令レベルで提供するが，得られる性能は実装コストに比べてかなり高い．

5.2 コードサイズ

FLATS2の命令セットは，BLアドレシングを用いたため64 bit長とやや長めになっている．そこで，4.で性能評価に使用したベンチマークテスト本体について，オブジェクトコードのサイズを調べてみた．対照のためにコード密度のかなり高いプロセッサであるMC68020と比較して，表5に示す．コード長の比はほぼ一定で，FLATS2はMC68020の2倍前後である．BLアドレシングによる性能の向上を勘案すれば，悪くはない値であると考えられる．

6. む　す　び

範囲検査機能付きのアドレシングモードである，BLアドレシングについて説明し，BLアドレシングを命令セット内に実装した計算機FLATS2について報告した．FLATS2アーキテクチャの概要を説明し，FLATS2の命令セットを用いて配列処理を効率良く実現する方法を述べた．更に，いくつかのベンチマークテストを用いて評価を行った結果，BLアドレシングによって最内周ループで高い演算性能が得られ，本方式が有効に働くことを示した．

FLATS2は，資源共有型MIMDの一種である循環パイプライン方式の計算機である．しかし現時点では，FLATS2の仮想プロセッサを複数個用いた評価については，まだ終了していない．循環パイプラインの利点を生かして，効率の良いマルチプロセシングを実現する方法についても，更に検討を進める必要がある．

FLATS2は市販のMSI/SSIを用いて実装されているが，循環パイプラインは本来，将来のラッチ機能付

き論理素子を用いた多段のパイプラインに適した方式である．今後はFLATS2の評価を通じて，より多段で命令流数の多いプロセッサについても考察してゆく必要がある．

謝辞 FLATS2計画を実現するにあたって，御協力を頂いた東京大学および理化学研究所の諸氏と，新技術事業団の同僚諸氏に深く感謝します．FLATS2の製作を担当した三井造船および三井造船システム技研の諸氏に感謝します．

文　献

(1) Shimizu K., Goto E. and Ichikawa S. : "CPC (Cyclic Pipeline Computer)—An Architecture Suited for Josephson and Pipelined Memory Machines", IEEE Trans. Comput., **38**, 6, pp. 825-832 (1989).
(2) Ichikawa S. : "A Study on a Cyclic Pipeline Computer : FLATS2", 修士論文，東京大学理学部情報科学科 (1987).
(3) 市川周一，加藤紀行，後藤英一："循環パイプライン計算機FLATS2"，情処学研報，88-ARC-72-1.
(4) Sato M., Ichikawa S. and Goto E. : "Run-time Checking in LISP by Integrating Memory Addressing and Range Checking", Proc. 16th Annual Symposium on Computer Architecture, pp. 290-297 (1989).
(5) Jouppi N. P., Bertoni J. and Wall D. W. : "A Unified Vector/Scalar Floating-Point Architecture", Proc. 3rd Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS III), pp. 134-143 (1989).
(6) Digital Equipment Corporation : "VAX ARCHITECTURE HANDBOOK", Digital Equipment Corporation (1981).
(7) Jouppi N. P. and Wall D. W. : "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines", Proc. 3rd Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS III), pp. 272-282 (1989).
(8) Smith M. D., Johnson M. and Horowitz M. : "Limits on Multiple Instruction Issue", Proc. 3rd Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS III), pp. 290-302 (1989).
(9) Fisher J. A. : "Very Long Instruction Word Architectures and The ELI-512", Proc. 10th Annual Symposium on Computer Architecture, pp. 140-150 (1983).
(10) Colwell R. P. et al. : "A VLIW Architecture for a Trace Scheduling Compiler", IEEE Trans. Comput., **37**, 8, pp. 967-979 (1988).
(11) Sohi G. S. and Vajapeyam S. : "Tradeoffs in Instruction Format Design for Horizontal Architectures", Proc. 3rd Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS III), pp. 15-25 (1989).

(平成2年8月10日受付，11月20日再受付)

市川　周一

昭60東大・理・情報科学卒．昭62同大大学院修士課程了．現在，新技術事業団後藤磁束量子情報プロジェクト研究員．計算機アーキテクチャ，特にパイプライン計算機の研究と設計に従事．情報処理学会，ACM各会員．

佐藤　三久

昭57東大・理・情報科学卒．昭61同大大学院博士課程中退．現在，新技術事業団後藤磁束量子情報プロジェクト研究員．コンパイラ，システムプログラミング，計算機アーキテクチャの研究に従事．情報処理学会会員．

後藤　英一

昭28東大・理・物理卒．現在，東京大学理学部情報科学科教授．同大学大型計算機センター長．理博．現在の主要研究テーマ：数式処理，電子幾何光学，リスプマシン，ジョセフソン接合論理．