ハイパフォーマンス
コンピューティング 48－3
（１９９３．８．１９）

並列計算機AP1000用数値演算アクセラレータの構成と評価

清水 俊幸　石畑 宏明*
(株)富士通研究所　並列処理研究センター†
飯野 秀之　木村 雅春‡
富士通(株)

並列計算機AP1000の数値演算アクセラレータオプション(NCA: Numerical Computation Accelerator)を開発した．NCAによりAP1000のプロセッサエレメント(Cell)にベクトル処理機構を付加し，計算能力を高めた．NCAでは,ベクトル演算器とスカラ演算器の間にコマンドFIFOと呼ぶバッファを設け,ベクトル処理とスカラ処理のオーバラップを可能とした.オーバラップによりベクトル演算器とスカラ演算器の処理速度の差に起因する演算器の利用率の低下を防ぎ,トータルな処理時間の短縮を実現した．NCAのアーキテクチャと基本性能,並列処理性能について述べる．

An architecture and performance evaluation of a Numerical Computation Accelerator (NCA) for the AP1000

Toshiyuki Shimizu, Hiroaki Ishihata
Parallel Computing Research Center, FUJITSU LABORATORIES LTD.
Hideyuki Iino, Masaharu Kimura
FUJITSU LIMITED

A Numerical Computation Accelerator (NCA) is developed to accelerate numerical applications for the AP1000. The NCA attaches a vector processing unit to a processing element, or cell, of the AP1000. The NCA has a FIFO buffer between a cell and a vector processing unit. The FIFO buffer can accept several vector processing commands while the vector processing unit is busy. It prevents inefficiency from the difference of processing speeds between scalar and vector processing units and keeps the vector processing unit busy. In this paper, the NCA architecture and its preliminary performance evaluation are presented.

*E-mail:{toshi, hata}@flab.fujitsu.co.jp
†211 川崎市中原区上小田中1015
‡E-mail:kimura@gmd.ed.fujitsu.co.jp

1 はじめに

計算機の性能向上にともない,詳細な科学技術計算(シュミレーション等)が行なわれるようになり,アプリケーションの要求する計算能力も増加の一途をたどっている.膨大な計算能力に対する要求に,ベクトル処理を高速化することによって応えるアプローチと,複数の演算器を並列に動作させることによって応えるアプローチがある.

ベクトル化,並列化はアプリケーションによって適用した場合に,処理速度の短縮に差が生じる.ベクトル処理と並列処理を組み合わせて,アプリケーションに応じた高速化手法を適用することによって,最適効率で問題を解くことを検討する必要がある.我々は,並列計算機AP1000用にベクトル処理機構(数値演算アクセラレータ:NCA)を開発し,この問題に取り組んでいる.

アプリケーションをベクトル化し,ベクトル演算器で高速実行すると,加速されないスカラ部の比率が高まり,全体の処理が加速されないのは,良く知られた問題である.この原因には,ベクトル処理とスカラ処理の間に依存関係があって実行の順番に制約がある場合と,ベクトル演算器に十分演算命令を供給できずにベクトル演算器の利用率が低くなってしまう場合の2つがある.

依存関係による実行順序制約を緩和するには,アルゴリズムの変更を行なう必要がある.これに対して,演算器に十分命令が供給できない問題に対しては,ベクトル演算器が命令を実行中にでも命令のキューイングが可能な構成として,スカラ演算器の命令供給能力不足を隠すことができる.

一般に,アプリケーションではいろいろな長さのベクトル演算が実行される.ガウスの消去法(LINPACKの一部)等ではベクトル長がマトリックスのサイズから1まで変化していく.このような変化を見せる場合には,ベクトル長の長い演算の初期には,多くのベクトル命令をキューイングさせておくことができ,ベクトル演算器の利用率を高く維持できる.スカラ演算器とベクトル演算器のスピードが大きく異なる場合には,このようなキューイングの機構が大切である.

本稿では,NCAの構成と基本的な性能評価について述べる.設計指針,動作モデルについて検討した後,アーキテクチャについて述べ,基本性能および,並列処理性能について述べる.最後に今後の課題について考察する.

2 デザインコンセプト

並列計算機AP1000の数値演算アクセラレータ(NCA)を設計するにあたり,次の三点に注力した.

1. オーバラップ実行
   AP1000のCellプロセッサであるSPARCとベクトル処理演算器μVPが互いに並列に動作できること.同期が必要な時にとれ,不要な待ち合わせが起きないようにして演算器の利用率を高く維持する.
2. 短いスタートアップ時間
   ベクトル処理性能に強く影響をおよぼすスタートアップ時間を短くすること.並列計算機では,問題が分割されて,ベクトル長が短くなる可能性があり,その場合においても高い性能を出せることが重要である.
3. 広いメモリバンド幅
   ベクトル処理に本質的なメモリアクセス性能を高く実現する.メモリのパイプライン化によって,高いスループットを実現する.

NCAでは,オーバラップ実行の実現とスタートアップ時間の短縮のために,コマンドFIFOと呼ぶキューをSPARCとμVPの間に持たせた.また,メモリバンド幅を広く実現するため,大容量SRAMを2バンク構成で持たせ,リード4段/ライト1段のパイプラインメモリとした.

3 実行モデル

μVPとSPARCを動作させる場合の実行モデルについて検討する.図1にいろいろな実現方法におけるタイミングを示す.ここでT_{SPARC}はSPARCが,例えば次のベクトル処理のためのアドレス計算やループの処理といった,スカラ処理を行なっている時間である.T_{STR}はCellとNCAを接続するバス(LBUS)を介してμVPのステータスを読む時間である.このステータスによって次のコマンドが発行できるかどうか判断する.T_{CMDW}は同様にLBUSを介してμVPにコマンドを書き込む時間である.T_{VP}はμVPがベクトル処理をしている時間である.この例では,処理時間(ベクトル長)は徐々に短くなっている.各シーケンスにおいて上段はμVPの資源を使用する処理であり,下段はSPARCの資源を使用する処理である.

No_overはベクトル演算ライブラリを呼び出すイメージによる最も簡単な実現である.μVPを起動するとその処理が終了するまで,SPARCはWAIT

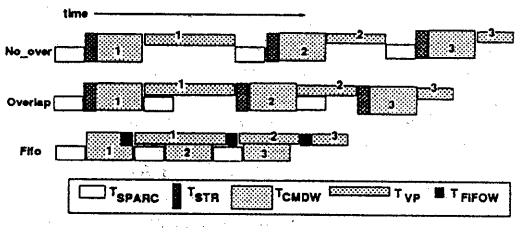

図 1: 実行モデル

する．この実現はμVPがアクセスするベクトル処理対象データ(ベクトルデータ)とSPARCがアクセスするスカラ処理対象データ(スカラデータ)とのデータ依存関係について考慮する必要が無くベクトル化が容易であるが,逆に一時に動作できるのは，SPARCかμVPの一方のみである．実行時間は$T_{no_over} = T_{STR} + T_{CMDW} + T_{VP} + T_{SPARC}$である．

OverlapはμVPの起動後はSPARCはμVPと並行に動作してスカラ処理を行ない,次のベクトル演算の起動の準備まで並列に動作できる．次のベクトル演算を起動するにはμVPの実行終了を待つ必要がある．この実現の場合,ベクトルデータとスカラデータの依存関係について解析し,必要であれば待ち合わせをする必要がある.依存関係がない場合の実行時間は,定常状態を求めるために,はじめのT_{SPARC}をスタートアップ時間，$T_{base} = T_{SPARC}$として無視すれば，$T_{overlap} = T_{STR} + T_{CMDW} + max(T_{VP}, T_{SPARC})$となる．このモデルを実現する場合には，SPARCによるメモリ参照とμVPによるメモリ参照のアービトレーションに配慮する程度で,アーキテクチャ的にはno_overと差はない．

Fifoはベクトル演算の起動準備が終了する都度，SPARCとμVPの間に設定したFIFOにコマンドを書き込めば，次のスカラ処理を続けることができるモデルである．FIFOに書き込まれたコマンドは，μVPがレディになるとμVPに書き込まれ，再びベクトル処理が開始される.ステータスの読み出しの必要はない．ここで，T_{FIFOW}はコマンドFIFOからμVPにコマンドを書き込む時間である．このモデルは，overlapに対して，FIFOの段数分コマンドを先出しできる点，ステータス読み出しが必要ない点，SPARCが書き込むのに比べ短い時間でコマンドをμVPに書き込むことができる点が有利である．FIFOが溢れず,データ依存関係がなければ,実行時間T_{fifo}は，overlapと同様に$T_{base} = T_{SPARC}$として無視すれば，$T_{fifo} = max(T_{CMDW} + max(T_{SPARC}, T_{VP}), T_{VP} + T_{FIFOW})$となる．

ここで,実際にこれらのパラメータの値について検討してみる．μVPおよびその周辺を50MHzで動かした場合は，1ワード(32ビット)，2ワードの書き込みに要する時間はそれぞれ，$T_{FIFO1W} = 40(ns), T_{FIFO2W} = 60(ns)$である．SPARCからの読み書きはLBUSの性能値で，$T_{STR} = 560(ns)$, $T_{CMD1W} = 240(ns), T_{CMD2W} = 280(ns)$である．

μVPのDAXPYループにおけるタイミングはベクトル長をnとした時，$T_{VP} = 33 \sim 36 + 3n(clock) \simeq 700 + 60n(ns)$である．

これに対して，μVPの起動に必要なコマンドの数は1ワード7回，2ワード1回であるから，$T_{FIFOW} = 340(ns)$, $T_{CMDW} = 1960(ns)$である．

スカラ演算の実行時間は正確に見積もることができないが，$T_{SPARC} = 3000(ns)$の場合について,各モデルの所要時間と，ピーク性能の半分を出せるベクトル長$N_{1/2}$を求めた結果を表1に示す．

表 1: 性能見積り(DAXPY)

	実行時間 (ns)	$N_{1/2}$
T_{no_over}	$6220 + 60n$	127
$T_{overlap}$	$5520 (n \leq 38)$	
	$3220 + 60n (n > 38)$	53
T_{fifo}	$4960 (n \leq 38)$	
	$2660 + 60n (n \leq 65)$	41
	$1040 + 60n (n > 65)$	

以上のように,常に$T_{no_over} \geq T_{overlap} \geq T_{fifo}$の関係が成り立っている.また,overlap, fifoでは,それぞれ$n < 38, n < 65$の範囲でμVPの処理が先に終了してしまい，μVPの利用率が低くなっていることがわかる．T_{SPARC}が短くなるか，T_{VP}が長くなれば，μVPの利用率が上がり，$N_{1/2}$が小さくなって性能の立ち上がりが良くなっていくことがわかる．

4 NCAシステム

4.1 システム構成

AP1000は，Cell毎にメモリを持つメッセージパッシング型並列計算機である[3]．AP1000はCellの内部バス(LBUS)を介して,オプションを接続できる．NCAは,このオプション接続機構を利用して接続する.図2にNCAを接続した場合のCellのシステム構成を示す．

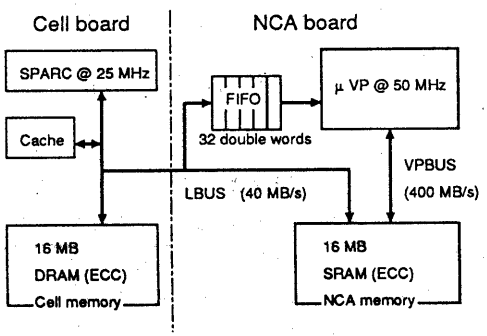

図 2: NCA接続時のCellのシステム構成

4.2 μVP

μVP(MB92831)は8Kバイトのベクトルレジスタと, ADD, MLT, DIV, LOAD/STOREパイプを1本づつ,計4本持つベクトル演算器である[2]. 演算パイプライン段数はADD, MLTは2段, DIVは8段である. 演算ピーク性能は, ADD, MLT, DIVパイプが並列動作時, 50MHz動作で, 106 MFLOPS(倍精度), 206 MFLOPS(単精度)である.

データバスは64ビット幅で, LOAD/STOREとも400 MBのバンド幅を持つ. μVPのレジスタの読み書きによって処理の起動や終了の確認を行なう. μVPは内部に命令バッファ(VCB)を256語持ち, VPBUSを介してVPコードをフェッチし,単独で処理を継続することも可能である. 倍精度実数/単精度実数/整数の四則演算,整数の論理演算, マクロ演算(最大値,最小値のサーチなど)を備える.

4.3 NCA

SPARCとμVPの並列動作を実現するためにダブルワード(64ビット) 32段のコマンドFIFOを用意し, μVPが動作中の場合でも, SPARCからμVPへの命令の発行が可能である.

メモリは, 4MビットSRAMを使用した2バンク構成で, 32ビット単位のECCを使用する. メモリはアドレスのビット3が連続して同じ値でアクセスされると(2ダブルワードおき)バンクコンフリクトが生じる.

メモリ制御およびLBUSインタフェースのゲートアレイは, 同じチップを2個, モードを切り替えて使用する. コマンドFIFOはこのチップに32ビットづつ持たせる. 表2にNCAボードの諸元をまとめる.

表 2: NCAボード諸元

NCA unit数	2 unit/ボード
動作周波数	50 MHz
演算性能(単精度)	206 MFLOPS/unit
(倍精度)	106 MFLOPS/unit
メモリ容量	16 Mbytes/unit (SEC-DED)
メモリアクセス	400 Mbyte/s (μVP)
	40 Mbyte/s (SPARC)
コマンドFIFO	32 × 64 bits queue
寸法	380mm × 243mm
消費電力	20 W/unit

5 ソフトウエア

NCAボードを動作させるためには,コンパイラにμVPのコード生成を任せるか,システムに用意された演算ライブラリを呼び出すか,あるいはユーザがμVPのプログラムを独自に開発する必要がある. コンパイラサポートについては今後の課題である. ここでは,演算ライブラリの実現方法について述べる. ユーザがライブラリ以上のレベルでμVPプログラムを作成する場合も同様に実現できる.

ライブラリを実現するためには, μVPプログラム(VPコード)と, それを起動するためのプログラム(ドライバ)を用意する. List1, List2は,DAXPYを計算するVPコードと, ドライバである[1].

5.1 VPコード

VPコードはラベル_vdaxpy (4行目)から実行が開始される. まず配列dx[]をベクトルレジスタvr0にベクタ長(VLEN個)ロードし(5), これにスカラ値(vsr28)を乗算してvr8に入れる(6). 配列dy[]をvr16にロードし(7), vr8と加えた結果をvr24に入れる(8).その結果をdy[]に書き戻す(9). μVPの実行はvstopで終了する(10).

List 1 VPコードの実現例(daxpy)

```
1       .seg    "data"
2       .global _vdaxpy,_vdaxpye
3       .VPU 0
4   _vdaxpy:
5       vld64  vsr4, vsr2,  vr0
6       vmulsd vr0,  vsr28, vr8
7       vld64  vsr5, vsr3,  vr16
8       vaddd  vr16, vr8,   vr24
9       vst64  vr24, vsr5,  vsr3
10      vstop
11  _vdaxpye:
12      .ENDV
```

μVPは,これらロードストア命令,演算命令の他,レ

[1]例では簡単のため,ベクタ長は128迄に限定し,演算方向も順方向のみである

ジスタ比較や, ブランチ命令を持ち,ベクトル長がレジスタ長を越えた場合のストリップマイニング処理を単独で行なうことが可能である.

5.2 ドライバ

ドライバは上記VPコードの実行に先立ち,必要なパラメータ(daの値や, 配列dx[], dy[]のアドレス, ベクタ長など)をμVPにセットし, その後μVPをスタートする. List 2は fifo の動作モデルで動く.

List 2 ドライバの実現例(daxpy)

```
1   daxpy(n,da,dx,incx,dy,incy)
2   double dx[],dy[],da;
3   int incx,incy,n;
4   {
5     int vcb_pos;
6     /* location of VP code */
7     extern int *vdaxpye, *vdaxpy;
8
9     /* set VP code to VCB */
10    vcb_pos=nca_set_code(&vdaxpy,
11                         &vdaxpye-&vdaxpy);
12
13    NCA_FIFO(8);
14    VPUIO(UVSTS)= NE_RESET;
15    VPUIO(UVLEN)= n;      /* vector length */
16    VPUDIO(UVSR(28))=da; /* set in double */
17    VPUIO(UVSR(2))= (int)(&dx[0]);
18    VPUIO(UVSR(3))= (int)(&dy[0]);
19    VPUIO(UVSR(4))= incx; /* stride of dx */
20    VPUIO(UVSR(5))= incy; /* stride of dy */
21    /* VP start from vcb_pos */
22    VPUIO(UVSTA)  = VPU_START|vcb_pos;
23  }
```

まず, nca_set_code()によって, VPコードをμVPの命令バッファ(VCB)にセットし, VPコードの先頭インデックスを返す.必要ならばリロケーションが行なわれる. この時,既にVCBにVPコードがセットされていれば, VCBへのセットは行なわず,インデックスのみを返す.

NCA_FIFO(8)によってNCAのコマンドFIFOに8ワード分の空きがあるかどうかを調べ,空きがなければ空くまで待つ. FIFOが用意できると, まずμVPのステータスをリセットするコマンド(NE_RESET)を書き込み, ベクタ長 n をVLENにセットし, 続いて, da および各種パラメータをμVPのレジスタにセットした後, μVPのVSTAレジスタにスタートコマンド(VPU_START)を書き込む. これらの一連のコマンドが, FIFOを経由してμVPに書き込まれると, μVPは実行を開始する. ここで, VPUIO(), VPUDIO()はどちらもマクロでそれぞれ指定したパラメータのアドレスへのintの書き込み, doubleの書き込みである.

6 基本性能

NCAの基本性能として,メモリのアクセス速度とDAXPYの性能を測定した.

6.1 メモリアクセス

List 3に示した測定プログラムをベクトル化して測定した. 結果を図3, 表3にまとめる.

List 3 メモリアクセステストプログラム

```
1   double a[], b[];
2
3   /* direct */
4   for (i=0; i<N, i+=1) a[i] = b[i];
5
6   /* indirect read */
7   for (i=0; i<N, i+=1) a[i] = b[l[i]];
8
9   /* indirect write */
10  for (i=0; i<N, i+=1) a[l[i]] = b[i];
```

表 3: Memory access speed

Mode	Read Setup, Rate (μs, MB/s)	Write Setup, Rate (μs, MB/s)	l[i]
Direct	6.42, 345		
L1	5.93, 234	5.86, 236	i
L5	7.14, 204	7.03, 205	1

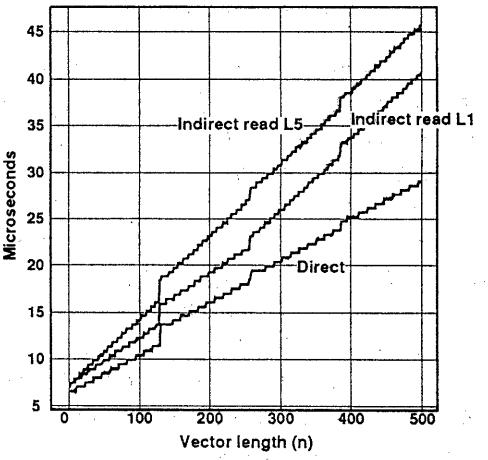

図 3: メモリアクセス速度

μVPのレジスタ設定の時間を含み全てのデータ転送が終了するまでの時間である. Directは順次メモリをアクセスするため,バンクコンフリクトが起きず, 最も高い転送レートを示す. Indirectアクセスは,はじめにリストベクトルをフェッチするために,見かけ上の転送レートは低くなっている. L1はバンクコンフリクトが生じない場合, L5は必ずバンクコンフリクトを起こす例である. Indirectライト

も同様な性能が得られた．全てのアクセスパターンで,最大スループット400MB/sの半分以上である．

6.2 DAXPY

DAXPYを例としてコマンドFIFOの効果を測定した結果を図4に示す．List 1, List 2を基本にドライバ側で128要素単位にストリップマイニングを施している．R_{fifo}, T_{fifo}, R_{over}, T_{over} はそれぞれ3節のfifoとoverlapの性能値である．

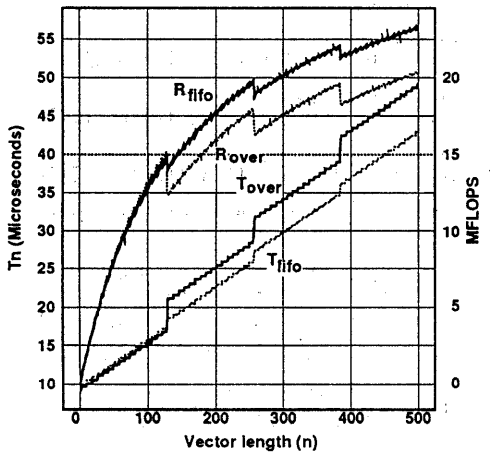

図 4: FIFOの効果(DAXPY)

ここで, 3節の検討と比較すると, SPARCで行なうストリップマイニング処理がT_{SPARC}にあたり, 無視した1回目の$T_{SPARC}(T_{base})$は, List 2の10行目等の時間を含むことになる．

fifoの場合, ストリップマイニングの1セグメント分のスタートアップは, T_{fifo}のスタートアップタイムだから, $T_{start} = 1040(ns)$ である．

$T_{base} = 8.7(\mu s)$程度と考えれば, コマンドFIFOを使用した場合の性能パラメータT_{fifoS}は, $T_{fifoS} = 8.70 + \lceil n/128 \rceil * 1.040 + 0.06n(\mu s)$であると予測される.これは,測定結果に完全に一致し, 動作モデルに従っていることが示された．

overlapについては, $T_{start} = 3220$であり,性能パラメータは, $T_{overlapS} = 8.70 + \lceil n/128 \rceil * 3.220 + 0.06n(\mu s)$となるが,実際には$T_{start} = 3400(ns)$である．この差はLBUSサイクルでいえば2サイクルであり無視できる．

7 アプリケーション

ハンドコンパイルでLINPACK, SCG, SLALOMについてベクトル化して性能を測定した．LINPACKでは, 単体性能についてコマンドFIFOの効果について検討する．SCG, SLALOMでは, NCAを用いて並列実行を行なった場合とNCAを使用しない場合の比較を行ない, 台数効果を比較する．

7.1 LINPACK

NCAを使用した場合のCell単体のLINPACK性能を, 3節における実行モデルに基づいて fifo, overlap, no_over で測定した．その結果を図5に示す．pivot行の交換をベクトル化するため, コードをループの外に出した以外はLINPACKベンチマークと同等のアルゴリズムを使用している[2]．

SPARCによるLINPACK性能は,倍精度,単精度それぞれ1.67 MFLOPS, 2.42 MFLOPS (100 × 100)であるのに対し, NCAをfifoモデルで使用すると 23.1 MFLOPS, 23.5 MFLOPSが得られた.それぞれ 14倍, 9.7倍の速度向上が示された．

LINPACKでは三角行列の処理を行なうため,ベクトル長が次第に短くなる．ベクタ長の長い前半で,コマンドFIFOにベクトル処理コマンドを貯めておくことによって, 後半で$T_{SPARC} < T_{VP}$となってもμVPとSPARCの処理が並行して動作する．

問題が小さな場合(マトリックスサイズ,精度)ほど, セットアップ時間が性能に与える影響は大きく, コマンドFIFOの効果や, μVPとSPARCの並列実行の効果が顕著である．

100 × 100の時,Singleと Doubleで性能に差が無いのは, Singleでは問題サイズが小さくT_{VP}が短い($T_{SPARC} < T_{VP}$の期間が短い)ため, 性能は演算ではなくμVPへの命令供給速度によって抑えられてしまうためである．200 × 200の時にもこの傾向がみられる．

ライブラリの単位で演算終了を確認する実現(no_over)やoverlapでは, μVPの性能を十分に利用できないことがわかる．

7.2 SCG

2次元のポアソン方程式をSCG[4]で解くプログラムをベクトル化し, 並列実行性能を測定した．

NCAの台数と性能の関係を図6, 図7に示す．ここで, VP6, VP8, VP10, VP12 は,それぞれ使用したNCAの数を示し, IU64, IU100, IU144, IU196は, NCAを使用しない時に使用したCell数を示す．

[2] ガウス消去法の部分ではDAXPYの係数ベクトルが再利用される事などを利用してロードストアの回数を減らす工夫をしている．

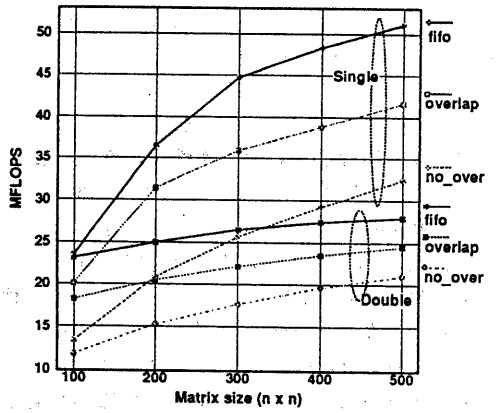

図 5: 問題サイズと単体性能(LINPACK)

X軸は問題の大きさを示し, 50 × 50, 100 × 100, 200 × 200, 400 × 400のメッシュについて計算を行なった場合の結果である.実行モデルにはfifoを用いた.

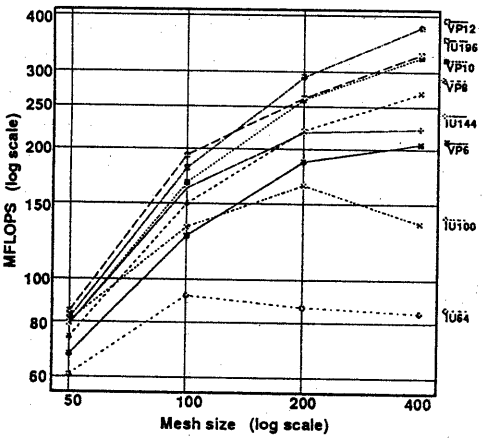

図 6: 問題サイズと並列実行性能(SCG)

ベクトル長VLはメッシュの大きさをn, セル台数をpとすると, $VL = n^2/p$.であるから,ベクトル長で200から27000程度の問題である. Y軸はその時のFLOPS値であり,図6では,システム全体の性能, 図7では,その時の単体あたりの性能を示している.

図7から判るようにNCAを使用しない場合,問題サイズが大きくなると,ある時点で単体あたりの性能が低下する. この性能低下の別れ目は,IU64では100,IU100では200とCellの台数が増えることにより, 問題サイズが大きなところとなる. これは, 1 Cell当りのデータ量がある程度以上となると, キャッシュのヒット率が低下するためである.

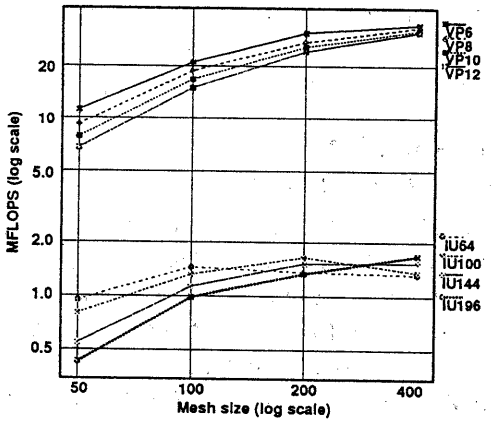

図 7: 並列実行時の単体性能(SCG)

NCAを使用した場合キャッシュのヒット率は性能に影響しない. 問題が大きくなるとベクトル長が長くなり演算ルーチンのピーク性能に近付いて行く. NCAを利用した場合,使用しなかった場合に比べNCA 1台で約18 Cellから19 Cell分の性能が得られていることが図6から読みとれる.

7.3 SLALOM

SLALOM[1]をベクトル化し並列実行した場合の性能を表4に示す. SLALOMでは,ベクトル化が容易で,しかも全体に占める時間の多くを費やしているルーチン(Solver)のみをベクトル化した. このプログラムは,もともとシステムのコンフィグレーションが$(2^n, 2^n)$という構成でのみ実行可能なため, 2 × 2と4 × 4の構成のみで測定している.

表 4: SLALOM in seconds

Patches	1020	693
NCA (2×2)	59.9,(20.5, 39.4)	27.2,(8.7, 18.5)
Cell (2×2)	171,(131, 41)	59.2,(40.4, 18.8)
Cell (4×4)	43.2,(32.2, 11.0)	14.1,(8.9, 5.2)

表4において, NCAはNCA付きの性能であり, バッチ数1020個で約60秒を要す. 処理の内訳は, (Solver, Other)で,それぞれ, Solverで要した時間, その他のルーチンで要した時間である.いずれも単位は秒. 同じ大きさの問題をSPARCのみで解くと, 同じ2 × 2の構成では, 171秒を要し,処理の内訳でSolverの部分で約6倍の時間がかかっていることがわかる. 4 × 4のSPARCで解いた場合には,総時間は43.2秒となる.この時,内訳は, Solver, OtherともCellの増加分の4倍程度の時間短縮がなされている.

以上のように,ベクトル化を行なったコアの部分

では, 約6倍程度の性能が得られるものの,ベクトル化していない部分が全体に占める割合が増加し,全体としての処理時間の短縮は2倍から3倍である.

8 考察と課題

8.1 データ配置の影響

性能評価は,ベクトルデータを全てμVPが直接アクセス可能なNCAメモリ上に置き, SPARCがアクセスするデータとのコヒーレンシを保証するために, キャッシュ不可として行なった. このため,ベクトルデータに対して, SPARCが頻繁にアクセスすれば,キャッシュミスのため性能が低下する. これを避けるため, SPARCによるベクトルデータアクセスを極力抑えて実現した.

しかし,全てのアプリケーションに対して, この様なオプティマイズを行なうのは難しい. ベクトル演算の前にキャッシュされているデータをフラッシュしたり, NCAメモリにコピーしたりする実現について, キャッシュミスとフラッシュ/コピーのオーバヘッドを考慮して検討する必要がある.

8.2 通信とベクトル処理

並列処理に必要なCell間通信は,データアクセスのローカリティを意識してオプティマイズされたキャッシュや汎用プロセッサにとってローカリティを崩す要因である.一般に受信したデータはキャッシュにのっていないか,あるいは受信時にキャッシュが無効化される.これは, フェッチ時にキャッシュミスペナルティとなり, また無効化時にもバスを使う.

ベクトルデータがキャッシュ不可領域にあること, データ依存関係が見切れていることを利用し, ベクトルデータはコピーすることなく送信し,受信することが望まれる. アクティブメッセージ等コピーを避けるメッセージ通信機構を取り入れることによって処理の高速化が期待できる.

9 まとめ

NCAを使用することは実数演算性能を要求するアプリケーションにおいて, 効果的であることを示した.

NCAはコマンドFIFOを備え,ベクトル処理とスカラ処理をオーバラップさせることにより,ベクトル処理命令の発行が複数可能となり,命令供給速度とベクトル処理速度のアンバランスが吸収され,結果としてセットアップ時間の短縮につながり,処理時間の大幅な短縮が実現できた.

また,広いメモリバンド幅を用意したことによって, バンクコンフリクトが起きない場合,ロードストア処理時間はμVPのピーク性能に等しくなり,バンクコンフリクトが生じた場合にでも, 少なくともピーク性能の1/2を維持することができた.

NCAを付加することによる性能向上は,ベクトル化の程度,問題の大きさ, アプリケーションの種類によって異なり, 2倍から20倍となっている. NCAのコストはCellの約4倍であり,アプリケーションによって性能メリットが出る場合と出ない場合がある. 今後,多くのアプリケーションに対して検討を進める必要がある.

今回の評価は,数台のNCAオプションを使用して行なったため,数百台規模の並列処理において, ベクトル処理を採り入れた場合の効果について検討することはできなかった. 今後,台数を増やし評価を行ないその効果について検討する必要がある.

コマンドFIFOを用いたオーバラップ処理を実現するためには, 単にBLAS1等のライブラリを提供するだけでは十分でない. SPARCが参照するデータと, μVPが参照するデータの依存関係を解析し, また,ベクトルコードを自動生成するコンパイラ等の開発することにより,これらの課題が解決される.

謝辞

ハードウエアの開発を行なって頂いた富士通デバイス(株)渡辺氏, 栗原氏, プロジェクト推進に関し御指導,御協力頂いた(株)富士通研究所並列処理研究センター石井センター長,池坂主任研究員,同僚諸氏に深謝します.

参考文献

[1] J. Gustafson et al. Slalom update. *Supercomputing Review*, pp. 56–61, March 1991.

[2] Hideyuki Iino et al. A 289MFLOPS single-chip supercomputer. In *IEEE 39th International Solid State Cuircuits Conference*, 1992.

[3] 石畑宏明, 稲野聡, 堀江健志, 清水俊幸, 池坂守夫. 高並列計算機AP1000のアーキテクチャ. 電子情報通信学会論文誌 *D-I*, Vol. J 75-D-I, No. 8, pp. 637–645, 1992.

[4] 速水, 原田. 対角法スケーリングを施した共役勾配法のベクトル計算機における有効性について. 情処論文誌, Vol. 30, No. 11, pp. 1364–1375, 1989.