計算機アーキテクチャ 107－9
（1994．7．21）

並列計算機用要素プロセッサ EMC-Y の基本性能評価

坂根広史 児玉祐悦 佐藤三久 山名早人 坂井修一† 山口喜教

電子技術総合研究所 †新情報処理開発機構

並列計算機 EM-X の要素プロセッサ EMC-Y は、通信と演算の強い結合によって高い並列処理性能が得られるよう独自に設計されたシングルチップ VLSI である。要素プロセッサが並列計算機においていかに効率よく機能しうるかを評価するためには、並列処理のための種々の機能や性質とともにプロセッサ単体での演算処理性能を把握しておくことが必要である。本稿ではまずプロセッサの概略および、主として RISC 演算処理部のアーキテクチャについて述べ、ベンチマークプログラムの逐次実行結果によりその性能について評価する。

Fundamental performance evaluation of a processing element EMC-Y for a parallel computer

Hirofumi SAKANE Yuetsu KODAMA Mitsuhisa SATO Hayato YAMANA
Shuichi SAKAI† Yoshinori YAMAGUCHI

Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305 Japan
†Real World Computing Partnership

In this paper, we evaluate an instruction execution performance of a processing element EMC-Y which is used in a parallel computer EM-X. To consider the performance of parallel computers, it is important to understand the performance of sequential execution as well as the performance of primitive operations on parallel processing. Since the sequential performance is a basis of parallel execution performance, we must grasp it accurately. We describe an architecture of EMC-Y, especially a RISC pipeline for instruction execution and evaluate results of well-known benchmark programs.

1 はじめに

並列計算機システムの処理能力を評価するためには、並列アプリケーションによって要素プロセッサ相互の並列協調動作を調べるだけではなく、各種並列処理プリミティブの性能を把握することが不可欠である。並列処理プリミティブとしては通信インターフェースのスループット、リモート要求に対するレイテンシ、メモリバンド幅、同期性能、演算処理速度等が挙げられる。その最も基本となるものの一つがプロセッサ単体の演算処理性能であり、他の並列処理プリミティブを評価するための基準となるため、できるだけ正確に把握することが必要である。

並列計算機EM-X とその要素プロセッサ EMC-Y [3][4] では、すでに開発し評価をおこなっている EM-4/EMC-R の基本アーキテクチャを引続き採用し、その評価で明らかとなった問題点の改良を試みた [1][2]。改良点をアーキテクチャ上および実装上の観点で挙げると、前者はリモートメモリアクセス機構の独立化、優先度別パケット処理、マッチング機構の改良、メモリアクセス調停方法の改良、命令セットの改良、トラップ機構の採用等があり、後者はパケットバッファの増設、レジスタの増設、浮動少数点演算回路の内蔵、メモリバンド幅の拡大、クロック周期の短縮等の実現である。

EMC-R/EMC-Y は強連結枝モデルに基づき、通信と演算をパイプラインレベルで融合している。この融合パイプラインによって通信処理と演算処理を同程度の基本処理単位時間で行うことにより、効率のよい細粒度並列処理を行うことができる。強連結枝モデルは、簡単なハードウェアで高速な命令発火を行うことができるダイレクトマッチングの採用や、パケットアーキテクチャの簡素化とパケット処理を逐次演算パイプライン内で実行する融合パイプラインによって EMC-R 上で初めて実現され、EMC-Y へ引き継がれた。演算処理部には RISC アーキテクチャを採用し、強連結ブロック内の高速な逐次演算実行を可能にした。それとともに強力な通信サポート機構として通信の単位であるパケットを 1 クロックで出力できる命令を持つ。なおネットワーク転送レートは 2 クロックにつき 1 パケットであり演算と通信の高い整合性を実現するが、さらに両者のレート差を吸収するため、ネットワークインターフェース部には FIFO パケットバッファを備える。RISC 演算パイプラインは 2 段に抑えている。これは通信循環パイプラインとの整合性のため [4]、および遅延実行命令を極力減らしハードウェアの複雑化を避けるためであるが、その結果メモリバンド幅の向上とあわせて、ロード / ストア命令は 1 クロックで実行される。アドレッシングおよびメモリアクセスは基本的にワード単位で行う。バイトアドレスを扱う命令もあるが、実際のメモリアクセスはワード単位でおこない、バイト操作はレジスタ上で行う。1 プロセッサあたりのアドレス空間は 1M ワードで、実記憶空間のみサポートする。逐次演算がすべて強連結ブロック内でかつパケット操作を行わなければ、EMC-Y を一般的な RISC プロセッサとして扱うことも可能である。

すでに述べた理由で RISC 演算処理部の演算能力を明らかにするために、逐次演算ベンチマークテストを行った。ベンチマークプログラムは主に EM-C コンパイラ [7] を用いて実行オブジェクトコードを得た。DHRYSTONE ベンチマークにより主に整数演算性能を、LINPACK ベンチマークにより浮動少数点演算性能を測定した。また実効的な最大浮動少数点演算性能を測定するためにアセンブラ命令で記述した行列乗算プログラムで評価を行った。

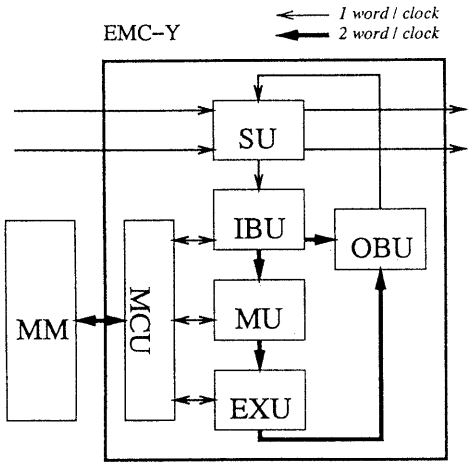
2 EMC-Y

EMC-Y は並列計算機 EM-X のために新たに設計されたシングルチップの要素プロセッサである。EMC-Y の諸元を表 1に、内部構成を図 1示す。データパスを示す線の太さは転送バンド幅を反映している。このほかにプロセッサ内部のフリップフロップを個別に読み書きするためのメンテナンス回路がある。

テクノロジ	CMOS 1.0 ミクロン ゲートアレイ (LSI Logic. LCA100K)
クロックサイクル	50ns
メモリサイクル	25ns
ワード長	メモリ 38bit ネットワーク 39bit
アドレス空間	1Mword
ゲート数	80872 ゲート
パッケージ	299pin PGA
信号ピン数	254pin
ALU,MPY	32bit
FALU,FMPY	IEEE754-1985 32bit
命令実行性能	20MIPS,40MFLOPS
パイプライン段数	2
レジスタ数	38bit x 32
ポート数	Inport:1word x 2port Outport:1word x 2port

表 1: EMC-Y の諸元

要素プロセッサに要求される要件として、並列処理のための通信プリミティブを高スループット低レイテンシで提供し、与えられた演算処理を十分高速に実行する必要がある。それを実現するための EMC-Y のアーキテクチャの特徴を以降に述べる。



SU : Switching Unit
IBU : Input Buffer Unit
MU : Matching Unit
EXU : EXecution Unit
OBU : Output Buffer Unit
MCU : Memory Control Unit
MM : Memory Module

図 1: EMC-Y の内部構成

2.1 通信機能

パケットアーキテクチャ

EMC-Y のパケットは 39bit × 2word の固定長で、1word のアドレス部と 1word のデータ部で構成される。パケットアドレス部には PE アドレスとローカルメモリアドレスを持ち、グローバルアドレスとして EM-X の全メモリ空間を指定できる。アドレス部にはこのほかパケットタイプ、待ち合わせモードフラグを持つ。ネットワークおよびメモリインターフェース部のデータ幅が 1word なのに対し、IBU:MU 間 および EXU:OBU 間では 2word 幅とし命令実行パイプラインとの整合性を図るとともにレイテンシ低減を実現している。

ネットワークインターフェース

EMC-Y は通信処理を強力にサポートする機構として、ネットワークスイッチ (SU)、パケットバッファ (IBU, OBU)、マッチング部 (MU) を内蔵しており、強連結枝モデルを効率良く実現する。

SU は 3 × 3 のクロスバスイッチで、サーキュラオメガネットワークを構成するためのセルフルーティングアルゴリズムにより動作する。各ポートは 1word 幅であるが、バーチャルカットスルー転送によりプロセッサ間レイテンシを 1 ホップあたり 1 クロックとし、レイテンシの低減を図っている。

パケットバッファは、IBU には 8 パケット分の FIFO を優先度別に 2 本、OBU には 8 パケット分の FIFO を 1 本持つ。IBU にはさらに FIFO が溢れた場合の外部メモリへの退避、リモートメモリアクセスのハードウェアサポート機能、マッチングのサポート機能がある。

MU は強連結枝モデルに基づくダイレクトマッチング処理と強連結ブロック先頭の命令発火 (フェッチ) を行う。

演算部

後述の演算部 (EXU) ではパケット出力命令を持ち、1 パケットを 1 クロックで出力できる。MU での高速マッチング / 命令発火機構とともに、融合パイプライン実現の要となっている。

2.2 メモリインターフェース

EMC-Y のアドレス空間は単一実記憶空間を提供し、メモリ階層は主メモリと内部レジスタの 2 種類のみで、仮想記憶やその他のキャッシュメモリはもたない。これによりメモリインターフェース回路を簡略化している。メモリバンド幅を向上させるため、システムクロックサイクルの 2 倍の速度で主メモリアクセスを可能としている。これは 1 チップでプロセッサを実現するためピン数およびゲート数の制限を回避するための手段である。これによりアーキテクチャ設計上の柔軟性を確保し、多くの改良と機能追加を可能にした。

MCU は各ユニットのリクエストをふたつのグループに分け、互いのアクセスを分離することにより回路の簡素化を図っている。例えば演算部では命令フェッチとメモリ参照命令の実行が同一クロックサイクル内で可能である。パケットとの整合と、柔軟な命令セット設計のためメモリを始めとして 1word は 38bit となっている。なお簡単なメモリプロテクション機能を持ち、主メモリ上のパケットバッファのアクセスは制限される。

2.3 演算部 (EXU)

EMC-Y の演算部では RISC アーキテクチャを採用し、ハードウェアの簡素化と命令実行の高速化を図っている。命令セットは独自に設計され、一般の演算命令のほかパケット出力命令を備えることで通信機能が強化されている。EMC-Y の用意しているデータタイプをビット数により分類すると以下のようになる。バイトオーダは big endian である。

38bit	ワード (6bit タグ +32bit 整数)
32bit	整数ワード
32bit	単精度浮動少数点
16bit	ショートワード
8bit	バイト

2.3.1 演算ユニット

整数演算ユニットとして 32bit ALU、論理演算ユニットおよびバレルシフタを備えている。また、IEEE754-1985 準拠の 32bit 単精度浮動少数点演算ユニットをチップに内蔵しているが、複雑さを避けるため non-pipelined の Floating ALU、Floating Multiplier を用いたことにより 1 クロックで演算結果が得られる。整数演算および浮動少数点演算は下位の 32bit が基本単位であり、上位 6bit はデータタグとして通常の命令では第 1 オペランドの値が継承されるほか専用の命令で扱うことができる。

2.3.2 レジスタセット

EMC-Y は表 2に示す 38bit レジスタを 32 個を備えている。これらは通常の命令のオペランドに指定できるが、このうち r0 から r25 の 26 個が汎用レジスタであり、残りは専用の用途を持つ特殊レジスタである。さらに特殊レジスタのうち r26 から r29 までは汎用レジスタとしても使える。なおこれらのレジスタは整数演算と浮動少数点演算に共通で使用できる。

制御レジスタおよび浮動少数点演算結果レジスタを表 3に示す。これらは通常の命令ではアクセスできず、制御レジスタはパイプラインの制御やプロセッサの状態を保存するためのレジスタで、浮動少数点演算結果レジスタは積和演算命令 (maaf) で連続演算を行うための専用補助レジスタとして用いられる。

r0 ～ r25	汎用レジスタ。以下の r26 から r29 までのレジスタも汎用レジスタとして使える。
r26(imr0)	即値レジスタ。即値ロード命令により 1 クロックでワードデータをロードできる。
r27(ap,imr1)	アドレスレジスタ。実効アドレスを自動格納したり演算の補助レジスタとして使われる。即値ロードも可。
r28(pr0), r29(pr1)	パケットデータレジスタ。強連結ブロックを起動したパケットのデータ部が格納される。
r30(fp)	パケットアドレスレジスタ。強連結ブロックを起動したパケットのアドレス部が格納される。
r31(zr)	ゼロレジスタ。内容はゼロに固定され、読みだし専用である。

() 内は別名

表 2: レジスタセット

PC	プログラムカウンタ。実行中の命令のアドレスを保持する
nPC	ネクスト PC。次に実行される命令のアドレスを保持する
RA0,1	例外処理復帰アドレスレジスタ
SR	ステータスレジスタ。割り込み制御フラグや演算結果フラグを持つ
FAR	浮動少数点演算結果レジスタ
FMR	浮動少数点乗算結果レジスタ

表 3: 制御レジスタ

2.3.3 命令セット

命令セットは表 4に示す 8 グループからなり、命令語長は 38bit である。以下にそれぞれのグループについて述べ、代表的な命令の例を EMC-Y アセンブラ言語表記で示す。*src n* はソースオペランドレジスタを、*dst* は結果レジスタを表す。多くの命令は第 2 ソースオペランドレジスタの代わりに埋め込み型即値 (符合付き 17bit) を指定できるが、その詳細は省略する。

命令グループ	命令
整数演算	add, sub, addc, subb, and, or, xor, xnr
シフト演算	lsl, lsr, lsrr, lslr, asl, asr, rol, shlc, shrc, sets, shas, shac
整数乗除	mul, divx, divi, divs, dive, divr, diviu, divsu, diveu, cvtdf, cvtfd, fsign, swap
浮動少数点演算	addf, subf, subrf, minf, maxf, negf, cvtif, cvtfi, divf, absf, mulf, maaf
メモリ参照	ldr, ld, st, xchg, lrr, lr, sr, xchgr, enq, deq, enqr, deqr, stb, sts
分岐	*bcc*, jl, jlr, strap, reti
パケット処理	*send*, *senda*, *lpa*, *alup*, *sendc*
その他	lddt, anddt, stdt, chgdt, setmt, ldmt, extsb, extb, insb, extss, exts, inss, alutst, ldi

(斜体は小グループの総称)

表 4: 命令セット

整数演算命令

32bit 整数の算術演算および論理演算を行う。

例: add *src0*, *src1*, *dst*: [*dst*] ← [*src0*] + [*src1*]

シフト演算命令

32bit の論理シフト、算術シフト、ローテート操作を行う。

例: lsl *src0,src1,dst*: [*dst*]←[*src0*] << [*src1*]

整数乗除命令

32bit 整数の乗算および除算補助演算を行う。

例: mul *src0,src1,dst*: [*dst*]←[*src0*] × [*src1*]

浮動少数点演算命令

32bit 単精度浮動少数点演算を行う。

例: addf *src0,src1,dst*: [*dst*]←[*src0*] + [*src1*]
maaf *src0,src1,dst*: [*dst*],FAR←FAR + FMR
FMR←[*src0*] × [*src1*]

maaf 命令は1クロックで浮動少数点の加算と乗算を同時に行うことができ、これにより EMC-Y の理論最大演算性能は 40 MFLOPS となる。

メモリ参照命令

メモリデータのロード/ストアを行う。

例: ldr *src0,src1,dst*: [*dst*]←MEM[[*src0*]+[*src1*]]
sr.a *src0,disp,src1*:
MEM[[*src0*]+*disp**4]←[*src1*]
ap←[*src0*]+*disp**4

メモリ参照命令では、命令フィールドの auto bit(アセンブラ命令では .a) によりメモリ参照の実効アドレスをアドレスポインタレジスタ (ap) に格納する機能の使用を選択できる。これによりメモリ参照とポインタの更新を同時に実行できる。

分岐命令

条件分岐、無遅延分岐、レジスタ間接分岐を行う。jl 命令以外はすべて遅延分岐となる。表中 *bcc* は branch on condition 命令グループの総称である。

例: beq *src0,src1,label* : delayed branch to *label*
if [*src0*]==[*src1*]
blt.a *src0,src1,label* : delayed branch to *label*
if [*src0*]<[*src1*]
cancel delay slot if annulled condition

遅延分岐を行う命令では命令フィールドの annulled bit = 1(アセンブラでは .a) でかつ分岐条件が満たされると delay slot の命令がキャンセルされる。これにより条件分岐命令を連続して実行することができる。

パケット処理命令

パケット出力およびパケットアドレス部を生成する。EMC-Y の命令セットアーキテクチャの中で特に特徴的なもので、一般の RISC プロセッサと決定的に異なるのがこのパケット処理命令である。

例: send0 *src0,src1,dst,disp,wcf* : send a packet

src0 のデータを *src1* 以下で指定するパケットアドレスに出力する。パケットアドレスには PE アドレスおよびメモリアドレスからなるグローバルアドレスが生成される。

その他

例: ldi *imm_tag,imm_data,dst*:
[*dst*]←[*35bit immediate data*]

ldi 命令の *dst* には imr0 と imr1 のみ指定可能。

2.3.4 メンテナンス回路

EMC-Y はメンテナンスバスをもち、チップ外部から内部のほとんどのフリップフロップをアクセスできる。また命令セットにもメンテナンス操作命令を持ち、プログラムからメンテナンス操作を行える。

2.3.5 例外処理

トラップの種類は MU でのマッチングエラー、IBU オーバフロー、メモリアクセス例外、メモリパリティエラー、strap 命令によるソフトウェアトラップである。strap 以外は hard wired によりトラップルーチンへ制御が移る。

2.3.6 命令実行パイプライン

パイプライン段数を2段に抑えることでハードウェアが簡略化された。命令フェッチとメモリ参照命令によるメモリアクセスが1クロック内で実行でき、その結果遅延ロードの制御やライトバックステージが不要となっている。ほとんどの命令が1クロックで実行でき、命令実行ステージで即結果が得られる。ただしネットワーク混雑により OBU 内の出力待ちパケット数が OBU 容量に達している状態で、さらにパケット出力命令を実行しようとしたときにはパイプラインが OBU ready 待ちで停止する。このほかにパイプラインが止まる要因としては、deq, xchg 命令で2度のメモリ参照のため実行に2クロックかかる時だけである。

図 2に実行パイプラインの動作例を示す。1クロックは2つのメモリアクセスフェーズにわかれている。ここで前半フェーズを p(rimary)、後半フェーズを s(econdary) とする。p は命令フェッチ、s はメモリ参照命令のためのメモリアクセスを行う。メモリアクセスにはほかに IBU, MU との競合があるが、ここでは省略する。システムクロック立ち上がりエッジによりレジスタの書き込みを含むプロセッサの状態遷移が行われる。

FD はフェッチ&デコードステージ、EX は命令実行ステージを表す。Addr, Data はメモリアクセスを表す。Addr の A*ins* は命令 *ins* をフェッチする命令アドレス、Aw/Ar はメモリ参照命令にともなうメモリアドレスを意味する。Data の I*ins* は命令フェッチで、A*ins* のアドレスが示すメモリの内容である。R/W はメモリ参照命令にともなうリード/ライトを意味する。命令フェッチアドレスは FD ステージ直前の s 終了時に現れる。p で命令がフェッチされ、続いて s でデコードが行われる。

次のクロックではデコードした命令を EX ステージで実行するとともに、FD ステージで次命令のフェッチを行う。実行する命令がメモリアクセス命令ならば p で実効アドレス計算とアドレス出力が行われ、s でメモリデータの

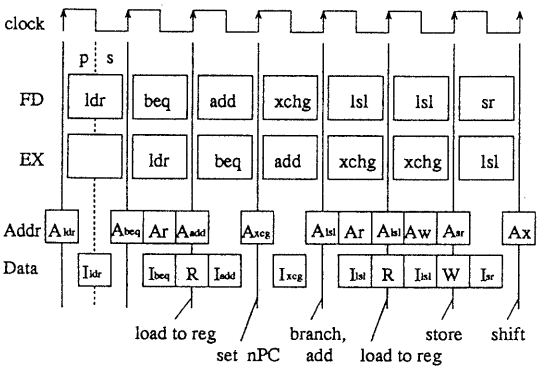


図 2: 命令実行パイプラインの流れ

リード / ライトを行う。xchg 命令のように 2 度メモリアクセスを行う場合、最初のクロックでメモリリードを行い、次でメモリライトを行う。その間パイプラインは停止し命令フェッチは中断される。条件分岐命令ではその実行中にフェッチされている命令が次のクロックで実行された後で分岐が行われる。なお jl 命令はデコード時に nPC へ書き込む準備ができるので遅延なしに分岐する。

3 ベンチマークによる性能評価

使用したベンチマークテストは DHRYSTONE、LINPACK、行列乗算の 3 種類である。

EMC-Y はすでに設計とタイミングチェックを終えているが本稿執筆時点ではメーカからの出荷待ちのため、ベンチマークの実行はソフトウェアシミュレータによって行った。このシミュレータはパラメータチューニング [5] やテストベクタ作成などプロセッサ設計支援にも用いられ、実際のプロセッサと相互結合網の動作をレジスタ転送レベルで忠実にシミュレートできる。すでにシミュレータ上の 80 台版 EM-X システムでいくつかの並列プログラムが動作している [6] が、本実験では単一プロセッサを評価するために逐次ベンチマークプログラムのみを用いた。

性能評価の参考とするため、各ベンチマークについて SUN SPARCstation2 で実行した結果をあわせて示す。実行条件は次の通りである。

SPARCstation 2	
OS :	SunOS 5.3
Compiler :	cc -O
CPU :	SPARC 40MHz
公称 MIPS :	28.5
公称 FLOPS :	4.2MFLOPS

現在、EM-X システムの言語処理系としては EM-C コンパイラが用意されている。EM-C は C 言語の並列プログラミング用に上位互換拡張した言語であり、EM-4 ではその細粒度並列機構へのアクセスを容易にし、マルチスレッド処理に適していることが実証されている [7]。並列性を陽に記述することにより最適な並列化を実現可能であり、核プログラミング言語として使用されている。EM-X でもすでにそのコンパイラが使用可能で、後述のベンチマークテストも EM-C を使用した。

DHRYSTONE : V2.1

code	dhrystones/second	VAX MIPS
standard lib.	12286	6.99
optimized lib.	17779	10.12
SPARC	43353	24.67

表 5: DHRYSTONE の結果

EM-C の並列プログラミングのサポートは言語仕様とマルチスレッドライブラリの両方により行われている。今回は逐次プログラムのコンパイルのみに使用したが、EM-C の関数コールのインプリメントはパケットによる起動方式がとられており、並列プログラミングへの移行は容易である。

3.1 DHRYSTONE

DHRYSTONE は主に整数演算やストリング操作性能を表すためのベンチマークである。時間測定部分以外はソースプログラムに手を加えず、コンパイラのオブジェクトをそのまま用いた。その結果を表 5に示す。VAX MIPS は VAX 11/780 の性能との相対値を表す。

プログラムの中でライブラリ関数 strcpy, strcmp が使われているのでそのライブラリの実行性能がベンチマークテストに影響する。標準ライブラリを使用した場合、DHRYSTONE ループ 1 回につき 1628 クロックかかった。それぞれ 30 バイトの文字列のコピーと比較を 1 回ずつ行っているが、strcpy, strcmp の実行クロック数は 1 回につきそれぞれ 452 クロック、373 クロックであり、実行時間の 51 % をこれらの関数が占めていることになる。EM-C の標準ライブラリはナイーブに C で記述したものであるが、それを使用したオブジェクトと、ワード単位で効率よく処理するようアセンブラ言語で記述した最適化ライブラリを用いたオブジェクトを実行して比較した。

最適化したライブラリを用いることで DHRYSTONE 値が約 45 %向上している。現時点で EM-X 用 EM-C コンパイラのオプティマイズ機能は使用範囲に制限があり今回は使用しなかった。SPARC の 約 0.41 倍の値が得られたが、EMC-Y は公称 MIPS 比で言えば SPARC の 0.7 倍の性能を持っているはずである。コンパイラとライブラリの性能に大きく左右されるためあくまで参考値であるが、

それらの改善によってさらに近付くことができる。EM-4でのEM-C使用経験と改善された命令セットから考えて、今回得られた数値のさらに1.5倍を上回る演算性能が引き出せることが期待できる。

3.2 LINPACK

LINPACKは浮動少数点演算性能を表すために一般的に用いられているベンチマークである。その演算時間のほとんどは図3に示すような単純な行列演算の最内周ループ部分が占めており、浮動少数点演算回数はほぼ n^3 のオーダとなる。この最内周ループを最適化すれば演算時間を大きく短縮でき、ハードウェアの能力を有効に引き出すことができる。そこでループの実行時間を可能な限り短縮することに注意しながら、このループを実行する関数daxpyをハンドコンパイルした。そのアセンブラコードを図4に示す。配列の操作はポインタで行い、インデックスの更新は実効アドレス自動格納機能を利用することにより、ループを6命令で実現した。それぞれの命令は1クロックで実行されるためループ1回の実行時間は6クロックである。浮動少数点演算を2個含んでいるのでnが十分大きければ最大6.66 MFLOPS (= 20MHz/(6/2)) で実行できる。m重のアンローリングを行なえば条件分岐命令が減り1ループあたり $5+1/m$ クロック ($m=8$ で最大7.80 MFLOPS) となる。なお今回作成したdaxpy関数のアセンブラルーチンにおいて引数処理やループ前処理にかかる時間は16クロックである。

```
for (i = 0; i < n; i++)
    y[i] = y[i] + a * x[i];
```

図 3: 最内周ループ (daxpy)

```
loop:           ;      -----  comment  -----
 ldr.a ap,r11,r0  ; dx[i]          ; r0<-[ap+r11], ap+=r11
 ldr.a ap,r12,r1  ; dy[i]          ; r1<-[ap+r12], ap+=r12
 mulf  r10,r0,r2  ; da*dx[i]         ; r2<-r10*r0
 addf  r2,r1,r1   ; dy=dy[i]+da*dx[i] ; r1<-r2+r1
 blt   ap,r9,loop ; index check      ; goto loop if ap<r9
 st    ap,0,r1    ; dy[i]; delay slot ; [ap]<-r1
```

図 4: daxpy のアセンブラコード

EM-Cコンパイラのみで得たオブジェクトによる実行結果(normal)と、daxpyを最適化して得た実行結果(daxpy)を表6に示す。daxpy4とdaxpy8はそれぞれdaxpyループの4重および8重アンローリングを行ったものである。浮動少数点演算はすべて単精度、配列サイズは $n=100$ である。

100×100 単精度		
code	MFLOPS	unrolling
normal	1.451	4
daxpy	4.991	–
daxpy4	5.437	4
daxpy8	5.505	8
SPARC	3.806	4

表 6: LINPACK の結果

最内周ループを最適化したコードで置き換えた結果、3.4 倍から3.8倍の演算速度向上が見られた。また、ループサイズが小さいのでアンローリングの効果が大きく、8重の場合で 1.10 倍向上した。8重のアンローリングではループ部分の理論最大値 7.80 MFLOPS の 0.7 倍の性能が引き出せた。これはnが大きくなるにつれ漸近的に理論値に近付く。また SPARC に対して最大1.45倍の性能となった。この優位性は、EMC-Y の浮動少数点演算命令が他の命令と同じく1クロックで実行できることによると考えられる。

3.3 行列乗算

行列の乗算プログラムを作成する際、最内周の積和の処理はインデックス幅の異なる2本のベクトルを処理する必要があるが、EMC-Y ではLINPACKのコードのようにインデックス更新に実効アドレス格納機能を用いることができる。オートインクリメント・デクリメントとは違い、任意の増減幅がとれる点で有用である。またmaaf命令は浮動少数点積和演算を1クロックで実行でき、行列の乗算やベクトルの内積演算では演算時間に大きく寄与する。

図5に $n \times n$ 次元行列乗算のプログラムと最内周ループ部分のアセンブラコードを示す。アセンブラコードの各命令オペランドはレジスタを別名で表してある。maaf命令は浮動少数点演算結果レジスタ (FAR,FMR) を使用するためその前後処理としてループ外部に5命令必要である。

実行クロック数および単位時間あたりの浮動少数点演算数を表7に示す。配列サイズは $n=100$、浮動少数点演算は単精度である。code の欄の normal はコンパイラのオブジェクトのみ用いたことを表し、mat はアセンブラコード(アンローリングなし)、mat8 はアセンブラコードで8重アンローリング処理したものである。

最適化コードにより、2.6倍から3.1倍演算速度が向上した。ループコードのサイズがLINPACK(daxpy)より小さいにもかかわらず向上の幅が小さいのは daxpy ループ

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++) {
    t = 0;
    for (k = 0; k < n; k++)
        t = t + A[i][k] * B[k][j];
    C[i][j] = t;
  }
```

```
loop:
 ldr.a  var_pa,4,tmp0          ; tmp0<-[pa+4], pa<-pa+4
 ldr    var_pb,0,tmp1          ; tmp1<-[pb]
 add    var_pb,var_n4,var_pb   ; pb+=n*4
 blt    var_pa,var_toa,loop    ; goto loop if pa<toa
 maaf   tmp0,tmp1,tmp          ; delay slot, tmp+=tmp0*tmp1
```

図 5: 行列乗算プログラム

に比べて配列参照が少ないためである。アンローリングの効果は 8 重で 1.19 倍となり、ループの小さい行列乗算の方がその効果が大きい。その場合、理論最大値 9.70 MFLOPS の 96.8 % まで到達した。これは浮動少数点演算の割合が大きいためと、最内周ループとその外側のループの間のループ変数および定数処理のオーバヘッドが小さいためである。SPARC に対して最大 1.56 倍の演算性能が得られた。LINPACK より差が大きい理由の一つとして配列データサイズがより大きいため SPARC ではキャッシュミスを多発することが考えられる。

このほか、ベクトルの内積演算を想定した場合、2 本のベクトルのインデックス増分が同じとするとインデックス操作をさらに減らすことができ、EMC-Y では配列要素のポインタ更新付きロード 2 命令、積和演算 1 命令、条件分岐 1 命令の 4 クロックのループで実現できる。m 重アンローリングした場合は 3+1/m クロックとなる。ベクトルが長くなるにつれて実行時間はそのループが持つ理論最大値に漸近する。m = 8 の場合最大 12.80 MFLOPS となる。

$C = AB$: 100×100　単精度

code	clock	MFLOPS	unrolling
normal	13100K	3.053	–
mat	5060K	7.904	–
mat8	4260K	9.389	8
SPARC	0.332sec	6.024	–

表 7: 行列乗算の結果

4　おわりに

EMC-Y はクロックあたりの単体演算性能は十分高速であると言え、EM-X のように強力な細粒度通信機構を持つ並列計算機の適用範囲をより広げることに結びつく。EM-4 ではデータパラレル言語が EM-C をターゲットとして構築されており、配列操作の基本演算のような処理をより高速に実行できる EMC-Y を持つ EM-X ではさらにその有効性が期待できる。

現在我々は並列計算機 EM-X の言語処理系、ソフトウェア開発環境およびシステムソフトウェア等の整備を進めている。また本稿で言及した単体演算性能をはじめ、各種ベンチマークプログラムによる評価も行っており、種々の並列処理能力について調べた結果を今後随時発表していく予定である。

謝辞

本研究を遂行するにあたり御指導、御討論いただいた太田情報アーキテクチャ部長ならびに計算機方式研究室の同僚諸氏に感謝いたします。

参考文献

[1] Kodama,K., Koumura,Y., Sato,M., Sakane,H., Sakai,S. and Yamaguti,Y. EMC-Y : Parallel Processing Element Optimizing Communication and Computation, to appear Proc. of ICS'93, (1993).

[2] 児玉, 佐藤, 坂根, 坂井, 山口. 高並列計算機 EM-X のアーキテクチャ, 情報処理学会 計算機アーキテクチャ研究会 101-7, (1993), pp.49-56.

[3] Yamaguchi,Y., Sakai,S., Hiraki,K., Kodama,Y. and Yuba,T. An Architectural Design of a Highly Parallel Dataflow Machine, Proc. of IFIP 89, (1989), 1155-1160.

[4] 坂井、平木、山口、児玉、弓場. データ駆動計算機のアーキテクチャ最適化に関する考察, 情報処理学会論文誌, Vol.30, No.12, pp.1562-1572, (1989)

[5] 坂根、児玉、佐藤、山名、坂井、山口. 並列計算機 EM-X のプロセッサ・ネットワークインターフェースの最適化の検討, 情報処理学会 計算機アーキテクチャ研究会 104-14, (1994), 105-112.

[6] 児玉, 坂根, 佐藤, 坂井, 山口. 高並列計算機 EM-X のリモートメモリ参照機構の評価, JSPP '94, (1994), pp.225-232.

[7] 佐藤, 児玉, 坂井, 山口. 並列計算機 EM-4 の並列プログラミング言語 EM-C, JSPP '93, (1993), pp.183-190.