計算機アーキテクチャ 85－9
（1990．11．22）

データ駆動計算機ＥＭ－４の関数分散方式

児玉祐悦 坂井修一 山口喜教
（電子技術総合研究所）

データ駆動計算機ＥＭ－４は、1000 台規模の並列処理を目指した高並列計算機である。このＥＭ－４における静的および動的関数分散方式について述べ、80 台の要素プロセッサＥＭＣ－Ｒからなる ＥＭ－４プロトタイプによる評価を行なう。静的関数分散方式としてＥＭ－４では、全ＰＥから乱数や巡回によって選び出す方式の他に、相互結合網での局所性を利用する方法が可能であり、分割統治問題による評価によりその有効性を確認した。動的関数分散方式としてＥＭ－４では相互結合網の局所性を利用した特殊パケット処理による方式を提案し、分割統治問題による評価によりその効果を確認した。

Load Balancing by Function Distribution Methods in a Highly Parallel Dataflow Machine EM-4

Yuetsu Kodama Shuichi Sakai Yoshinori Yamaguchi
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibalaki 305, Japan

The EM-4 is a highly parallel dataflow machine and its target structure has more than a thousand processing elements(PEs). This paper presents load balancing by function distribution methods in the EM-4 and evaluation of its methods by executing simple program on the EM-4 prototype, which consists of 80 PEs. The EM-4 can statically distribute a function invocation not only using random or round methods but also using locality of the network. The EM-4 can dynamically distribute a function invocation using MLPE packets which round local PEs and found local-minimum load PE. These merits of both function distribution methods are evaluated by executing a divide-and-conquer program.

1 はじめに

負荷分散の問題は、高並列計算機では同期・通信・並列性の抽出などと同様に重要な問題である。負荷分散問題とは、与えられた問題をより速く解くための処理順序の制御や、処理を行なう要素プロセッサの決定、処理データの分散を決めることである。本稿ではこの負荷分散問題のうち、特に関数分散方式、つまり関数呼び出しを行なう際にその関数をどの要素プロセッサで実行させるかを決定する方式について述べる。この関数分散方式は大きく分けて、プログラム実行以前に決めておく静的関数分散方式と、実行時に決定する動的関数分散方式とがある。80台の要素プロセッサからなるEM－4プロトタイプにおける両方式の実現方式について述べ、実際にサンプルプログラムを走らせて評価を行なう。

本稿では、まず関数分散の観点から見たEM－4プロトタイプの概要について紹介する(第2節)。次に、静的負荷分散方式について、その方式と評価を行なう(第3節)。さらに、動的負荷分散方式について、その方式と評価を行ない、両方式について比較検討する(第4節)。最後にまとめと今後の課題について述べる(第5節)。

2 EM－4プロトタイプ

データ駆動計算機EM－4は、1000台規模の並列処理を目指した高並列計算機である[2]。現在、第一段階として、要素プロセッサの全機能をシングルチップに納めたデータ駆動型プロセッサEMC－R[1]を試作し、このプロセッサを80台結合したEM－4プロトタイプを作成した。本プロトタイプは1990年4月から稼働中であり、実際のプログラムで999MIPSを達成した[3]。

関数分散を考える上で重要な項目として、結合網の構造・速度・距離、通信や同期のオーバーヘッド、演算処理の速度があげられる。EM－4プロトタイプの結合網は、プロセッサ結合型オメガ網を採用している。図1にノード数12の場合のオメガ網を示す。この結合網は、要素プロセッサ（PE）の台数をNとした時、

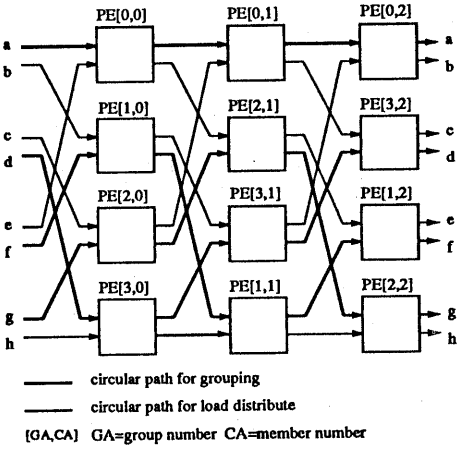


図1: オメガ網

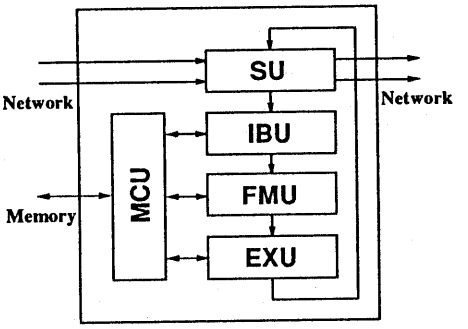


図2: EMC-Rの機能ブロック

ハードウェア量はO(N)、PE間距離はO(log N)以下といった望ましい性質を持っている。PE内にはパケット交換を行なうユニット（SU）を内蔵しており、このユニットは相互結合網との入出力2ポート、プロセッサとの入出力1ポートの3×3のクロスバースイッチとしてプロセッサとは独立に動作する。また、このユニットにはデッドロックを回避する機能や後で述べる動的負荷分散のための機能が含まれている。

通信・同期に関しては循環パイプラインが採り入れられており、少ないオーバーヘッドで各処理が行なわれる。図2にEMC－R内のブロック図を示す。ネットワークからやってきたパケットは上で述べたSUによってプロセッサ内に取り込まれ、パケット処理が行なわれる

までＩＢＵと呼ばれる入力パケットバッファで蓄えられる。ＩＢＵには３２ワード分の２ポートＲＡＭが内蔵されており、ＦＩＦＯ処理が高速に行なわれる。このＦＩＦＯが溢れるとパケットは外部メモリ上のバッファ（ＭＩＢ）に退避され、必要になるとＦＩＦＯに読み戻される。ＦＭＵではデータの待ち合わせ処理が行なわれ、データが揃っていればＥＸＵで実行される。この待ち合わせ処理には直接マッチングと呼ばれる連想記憶を使用しない高速な方式を採用している。

また、演算処理の高速化のため、上記の循環パイプラインの他にレジスタベースの先行制御パイプラインを採り入れ、両パイプラインを融合することにより効率のよい演算を実現している。

3 静的関数分散

静的関数分散とは関数呼び出しを行なうＰＥ (caller PE) とその呼び出された関数を実行するＰＥ (callee PE) との関係が、プログラム実行以前に決まっている方式である。 Callee PE の決定には、相互結合網の構造や距離、引数と結果の数、関数の大きさなどを考慮する必要がある。

3.1 ＥＭ－４における静的関数分散方式

ＥＭ－４では相互結合網としてプロセッサ結合型オメガ網を採用している。本結合網には各ＰＥから見て２つの循環路が存在する [4] が、ＥＭ－４ではこの内の１つをグループ内循環路として使用することにより相互結合網内でのグループ化を行なっている。すなわち、各ＰＥは［グループ番号、グループ内メンバー番号］の２つ組でアドレスづけされる。

ＥＭ－４では、静的関数割り付け方式として、１）乱数方式、２）巡回方式、３）絶対番地指定方式の他に、caller PE に対する callee PE の相対番地を指定する方式が可能である。すなわち、グループ内隣、グループ外隣、あるいはそれらの組み合わせといった caller PE から見た相対的な距離で callee PE を指定するわけである。また、絶対番地指定方式では、全ＰＥを一次元的につなげたハミルトン閉路上の隣などのようにＰＥからＰＥへの任意の対応を指定することが可能である。これらの指定は、プログラム言語内およびコンパイル時に任意に指定することができる。

また、ＥＭ－４の相互結合網は一方向（図１の左から右）にのみパケットが流れるため、引数を渡す時の caller PE から callee PE までの距離と結果をもらう時の callee PE から caller PE までの距離が異なる。このため関数呼びだし相手を決定するときには引数の数と結果の数とを考慮しなければならない。例えば下で述べる fib プログラムの場合、引数が２個・結果が１個であるので、caller PE からみてできるだけ近くのＰＥに割り付けるのが有利である。

さらに、関数の大きさは１つの関数の複数ＰＥへ割り付け・並列化とも関連して重要であるが、ここでは重要性を述べるだけにとどめておく。

3.2 評価

例としてフィボナッチ数を再帰的に計算するプログラム fib を考えることにする。これは典型的な分割統治問題であり、１つの関数は２つの関数を呼び出すが、その呼び出す関数をどのＰＥに割り当てるかはプログラムの実行性能に大きく関わってくる。ここでは以下の割り当て方法について比較してみる。

1. ２つの循環路上隣のＰＥに割り当てる
2. 全ＰＥを乱数で割り当てる
3. 全ＰＥを巡回的に割り当てる
4. 循環路上のＰＥを乱数で割り当てる
5. 循環路上のＰＥを巡回的に割り当てる
6. 全ＰＥを一次元的につなげた閉路上隣とそのまた隣に割り当てる

乱数としては各ＰＥで同じ乱数列を使用するため、乱数からＰＥ番号の対応表が全ＰＥで同じであると、特定のＰＥへの関数割り当てが集中してしまう。これを防ぐため、各ＰＥでは自ＰＥ番号から順にメンバー番号、グループ番号を

表 1: fib(20) による静的負荷分散方式の比較

割り当て方式	実行時間 (ms)	ＰＥ使用率 (%)
1 循環路上隣	1.12	54.6
2 乱数	1.98	44.0
3 巡回	1.99	40.1
4 循環路上乱数	1.49	62.1
5 循環路上巡回	1.37	62.7
6 1次元閉路上隣	8.44	8.6

増加させた表を使用している。巡回方式でもこの対応表を使用している。また、循環路上乱数方式と循環路上巡回方式では、グループ内循環路隣とグループ外循環路隣のＰＥ番号を交互に並べた表を使用している。

それぞれの方法で fib(20) を実行した結果を表1 に示す。表で実行時間とはスタートパケットを入れてから結果パケットが返ってくるまでの時間、ＰＥ使用率とは実行時間内の各クロックで動作中のＰＥ台数の割合の平均を表している。乱数方式や巡回方式では、乱数や巡回指標などはプログラムの先頭では初期化していないため、実行前のＰＥの状態によって関数割り当てが変化し、実行時間などが変化する。そのため、何回か実行した中から一番良いデータを示した。

実行時間を見ると、循環路上隣方式が一番高速であり、次が循環路上巡回方式、循環路上乱数方式となっている。このように循環路を利用した方式の実行時間が短いのは、プログラムが二進木状に展開されていくことが、オメガ網のシャッフルされた構造と適合しており、効率良く負荷が分散されていくためである。

また、1次元閉路上隣の場合は他の方法と比べて8倍ほど性能が低い。これは、割り当て方式とプログラムの構造が適合せず、各ＰＥに割り当てられた関数が平均化されないためであり、全体の処理時間が関数が多く割り当てられた少数のＰＥの処理時間に左右されてしまったためである。

ＰＥ使用率を見ると、循環路上巡回方式が循環路上隣方式より高く、適切に負荷分散が行なわれていると思われる。にもかかわらず、前者が後者よりも実行時間が大きい理由は、方式固有の処理オーバーヘッドの違いであると考えられる。循環路上隣方式では caller PE からみた callee PE は固定であり、ＰＥ番号は単に定数として読み込まれるだけである。それに対し、巡回方式ではその指標の更新や対応表からのＰＥ番号の読み込みなどで1回の関数呼び出し当たり8クロックの処理が必要である。しかし、このオーバーヘッドは関数が大きくなるにつれて相対的に小さくなるため、大きなプログラムでは循環路巡回方式が有利になると考えられる。

静的関数割り当てではプログラムの構造と相互結合網の構造とをうまく適合させた割り当て方式を使用するかどうかが、プログラムの実行性能に大きく関わってくる。またそのときには、関数割り当てがうまく均等になることとともにその方式固有の処理オーバヘッドについても考慮する必要がある。

4 動的関数分散

動的関数分散とはＰＥがプログラム実行時にその関数を実行するＰＥを割り当てる方式である。上で用いた乱数を使用する方法もプログラム中で乱数を計算しているが、乱数のシーケンスは静的であるため動的関数分散といえない。ここではプログラム実行中での他のＰＥの負荷などを考慮した動的な負荷分散方式について考える。このために必要なことは、負荷の見積り方法と負荷最小ＰＥの検出方法である。ＥＭ－4 におけるこれらの方法について述べ、実際の動的負荷分散方式について評価する。

4.1 ＥＭ－4における動的関数分散方式

負荷の見積りとしては1）入力パケットバッファ上のパケットの数、2）割り当てられている関数の数、3）構造メモリの使用量などを考慮する必要がある。

ＥＭＣ－Ｒでは図3に示したようにプロセッサの状況に応じて負荷の値が自動的に計算され

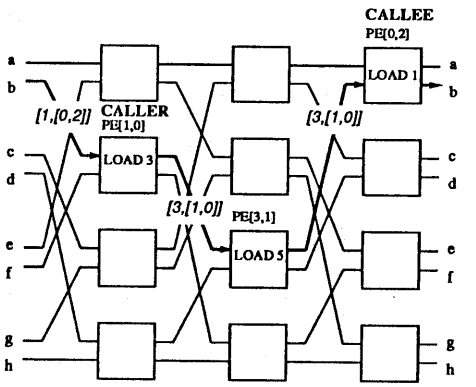
FIFO	0	1-7	8-F	10- 1F	*												
MIB	0				1- 1F	20- 3F	40- 5F	60- 7F	80- 9F	A0- BF	C0- DF	E0- FF	100- FFF	1000- 1FFF	*		
ESEN	8-1FFF													*	1-7	*	
ESTN	16-FFFF													*	1-15		
LOAD	0	1	2	3	4	5	6	7	8	9	A	B	C	F			

図 3: 負荷の見積り

るようになっている。ここで、FIFO とはチップ内蔵の FIFO 上の入力パケットの量、MIB はメモリ上の入力パケットの量、ESEN は使用可能な関数インスタンスの量、ESTN は使用可能な構造体メモリの量を表している。この見積り方法では2）、3）の数は、これ以上関数を割り当てられては資源が枯渇してしまうという危機的状況にのみ用いている。これは2）3）による負荷の増加は間接的にパケットの増加をもたらすため、主に1）のＩＢＵ上の待ちパケット数だけで十分に負荷量の指標になると考えられるからである。

図3 に示された負荷量は、プログラム内から読み出すことができ、これを用いたソフトウエアによる動的負荷分散が可能である。ただし、本方式は時間オーバーヘッドが大きいため、ＥＭ－４では通常用いない。

ＥＭ－４では、負荷最小のＰＥを検出するために MLPE という特殊パケットを用いる。このパケットはデータとして自分の負荷の値を持ったパケットである。MLPE パケットの行き先アドレスはコレを発行するＰＥ自身であるが、各ＰＥはこのパケットを例外的にＰＥ外に出力する。そして MLPE パケットは裏循環路を通って戻ってくる。この裏循環路を通過する間に各ＰＥの負荷の値とパケットデータの負荷の値とが比較され、もしＰＥの負荷の方が小さければその負荷の値とＰＥ番号によりデータ部を書き換える操作が行なわれる。これらの操作は通常のパケット転送のクロック内で実行されるため、余分なオーバーヘッドなしに裏循環路を一周する時間だけで、その中の負荷最小ＰＥを検出することができる。これらの処理を図４に示す。



[LD,[GA,CA]] は PE[GA,CA] が最小負荷 LD を持つ MLPE パケット示している

図 4: 最小負荷ＰＥの検出

しかし、関数を呼び出したい時に MLPE を出力して最小負荷のＰＥを検出するのでは手間がかかりすぎる。そのため、ＥＭＣ－Ｒでは先行取得した関数インスタンスの識別子（オペランドセグメント番号）を保持しておく特殊レジスタがあり、関数呼び出しの時はその値を使用し、次の関数呼び出しのために MLPE パケットを出力するという方法を取っている。

この方法にはいくつか注意しなければならない点がある。第一に、全ＰＥ中の最小負荷ＰＥを見つけるのではなく、裏循環路という局所的な中での最小負荷ＰＥを見つけている点である。第二に、先行取得しようとしていても次の関数呼び出しが必要な時までに関数インスタンスの取得が間に合わないときがある点である。そのような場合の処理として静的関数分散方式

表 2: 一次元結合方式の fib(20) 実行結果

	static	MLPE
実行時間 (ms)	8.44	2.63
平均動作ＰＥ (%)	8.6	26.3

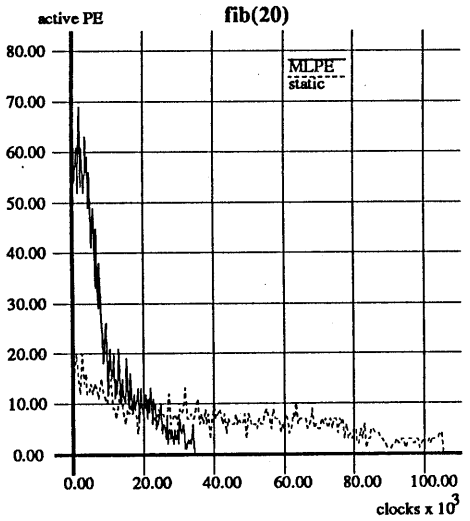


図 5: 一次元結合方式の実行ＰＥの時間的変化

などとの併用が必要である。第三に、最小負荷を見つけた時点とそのＰＥを利用する時点とでは負荷の分散状況に時間的差異が生じている点である。

4.2 例題による評価

静的負荷分散の評価と同様にフィボナッチ数を求めるプログラム fib を用いてＥＭ－４プロトタイプにおける MLPE を用いた動的負荷分散方式の評価を行なった。fib プログラムでは２つの関数を呼び出すが、そのうちの１つの関数呼び出しに MLPE パケットによる先取りオペランドセグメントを使用する。それ以外は静的関数分散で用いたプログラムと同じである。

最初に、この MLPE 手法を静的負荷分散方式で述べた全ＰＥを一次元上につなげた閉路上での隣に割り付ける方式に適用した場合の実行時間と平均動作ＰＥのデータを表2、各クロックでの実行ＰＥ台数の時間的変化を図５ に示す。この結果によると MLPE を用いた方が３倍ほど高速であり、特に計算の初期の段階での負荷分散が効果的に行なわれていることがわかる。また、静的負荷分散だけでは fib(21) を実行すると一部のＰＥで関数割り付けのオーバーフローが生じて実行できないが、MLPE 手法を適用すると fib(22) まで計算できることからも負荷分散が適切に行なわれていることがわかる。このように、プログラムの構造と整合性の良くない静的負荷分散方式にに対して MLPE を適用した場合の効果が確認された。

表 3: 循環路上隣方式の fib(24) 実行結果

	static	MLPE
実行時間 (ms)	7.16	7.10
実行命令数 ($\times 10^6$)	1.48	1.58
関数呼びだし	46367 (MLPE 2216)	
平均動作ＰＥ (%)	61.4	79.6
平均動作ＥＸＵ (%)	44.1	45.3

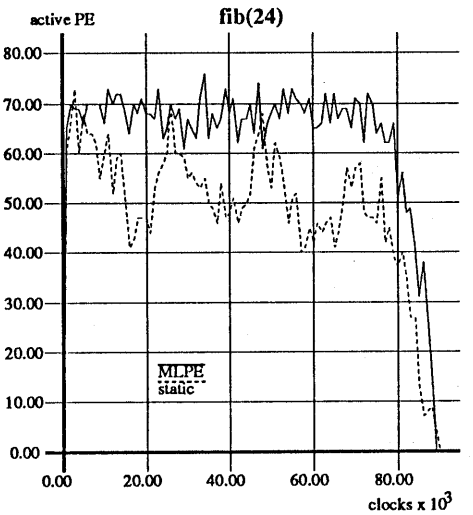


図 6: 循環路上隣方式の実行ＰＥの時間的変化

次に、オメガ網上で隣のＰＥに静的に関数割り付けを行なった場合とそれに MLPE による動的負荷分散を追加した場合とについて比較した。表３に実行時間などのデータを、図６に

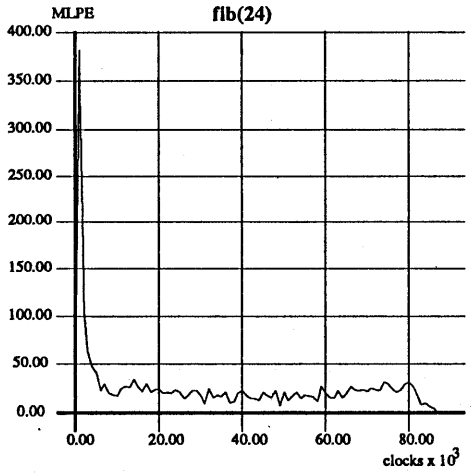


図 7: MLPE パケットの使用頻度

各クロックでの実行ＰＥ台数の時間変化を示す。この比較によると、MLPE を用いた方が多くのＰＥが動作し、またその時間変動も小さい。各方式の平均動作ＰＥは MLPE を用いた場合で 79.6%、用いない場合は 62.4% である。これにより動的負荷分散機構が適切に働いていることがわかる。

図 7 は使用された MLPE パケットの数を 1000 クロック毎に表したグラフである。この図によると、プログラム実行の立ち上がり時には多くの MLPE パケットが使用されているがその後はそれほど使われていないことがわかる。全関数呼び出しのうち MLPE を用いた関数呼び出しはほぼ５％である。これは全体の負荷が重くなり、ＩＢＵにパケットが常駐するようになると、MLPE パケットが負荷最小のＰＥを見つけて戻っても入力パケットバッファ（ＩＢＵ）内の他のパケットの処理が終るまで待たされるため、MLPE の利用サイクルが長くなるためである。

プログラムの実行時間は MLPE を用いた方がわずかに短い程度である。これは MLPE を用いることのオーバーヘッドが一因であるが、その増加分は実行命令数で見ると 7% にすぎない。このことの原因を探るため、各ＰＥの動作をより詳細に調べてみた。なお、これまでの議論ではＰＥが動作中であるとはＩＢＵ、ＦＭＵ、ＥＸＵのいずれかが動作している状態を指していた。

MLPE を用いたプログラムでＰＥ内のどの部分が動作しているかを調べた結果が図８ である。この図で細線がＥＸＵおよびＦＭＵが動作しているＰＥ台数、細線から点線までがＩＢＵだけが動作しているＰＥ台数、点線から太線までが結合網にパケットを出力できずにウエイトしているＰＥ台数を表している。ＰＥがＩＢＵだけで動作している割合は、平均 34.3%、結合網のビジーのためのウエイトは平均 13.4% となっている。一方、MLPE を用いないプログラムではＩＢＵの動作率は 17.3%、結合網によるウエイト率は 9.2% である。

ＩＢＵだけが動作している状態とは、ＰＥ内部の入力パケットバッファ（ＦＩＦＯ）が溢れた時のメモリ上の入力パケットバッファ（ＭＩＢ）への退避と、ＦＩＦＯが空になった時のＭＩＢからの回復を指している。この２つの場合はメモリアクセスがＩＢＵに限られるため、待ち合わせ処理や命令フェッチができずＦＭＵ、ＥＸＵは動作不能状態となる。このＩＢＵの動作は常に必要な動作ではなく、ＦＩＦＯが溢れた時にのみ必要な動作であり、１パケット当たり４クロックのオーバーヘッドとなる。すべてのパケットがこのＩＢＵ処理を行なうと仮定すると、fib プログラムの場合、実行時間中におけるＩＢＵ処理の割合は 47% となることが静的な解析によりわかる。MLPE を用いた場合はこのワーストケースにかなり近い状況であると考えられる。これらのことにより次のことが考察される。

MLPE を用いた動的負荷分散方式では負荷の偏りがなくなり、すべてのＰＥに均等にパケットが分散された結果、すべてのＰＥでＦＩＦＯが溢れてしまい、ＰＥの実行効率が低下してしまう。

この問題を解決するためには、ハードウエアによる方法とソフトウエアによる方法がある。ハードウエアによる方法はＩＢＵを使用する外部メモリとＦＭＵ・ＥＸＵが使用するメモリとを分離して同時アクセスを可能にすることである。ソフトウエアによる方法は、強連結ブロックを大きめに設定することにより、ＩＢＵによ

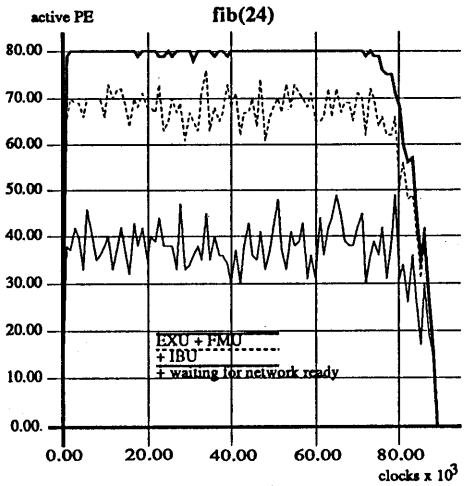


図 8: 動的負荷分散時の動作ユニットの割合

るオーバーヘッドを相対的に小さくすることである。

5 おわりに

ＥＭ－４の関数分散方式について述べ、例題を用いた評価を行なった。

静的関数分散方式では、その割り当て方法によりプログラムの性能に大きな違いが出てしまうため、応用プログラムにあった方法を取らなければならない。本稿で述べた fib プログラムでは問題が木構造状の展開を示すため、循環路上のＰＥに割り付ける方式が有利であった。

MLPE を用いた動的負荷分散方式により、負荷が均等化される事が確認できた。反面、ＩＢＵなどのオーバーヘッドの増加も確認され、これに対するハードウエア・ソフトウエア両面の解決方法について述べたが、これらの検証は今後の課題である。

プロトタイプ上での課題としては、ここでは簡単な分割統治型の問題だけを取り上げたが、他のより広範囲な問題に対してもその問題の性格に応じた負荷分散手法の開発とその有効性の検証を行なっていきたい。さらに、それらをコンパイラにより解析・選択する方法について研究していきたい。

１０２４台版ＥＭ－４へ向けての課題としては、MLPE をより有効的に利用するために本パケットの優先処理、先行取得オペランドセグメント番号の容量の増大、ＩＢＵのオーバーヘッドの軽減といったハードウエア的改良の方法とその有効性を検証していきたい。さらには、よりグローバルな負荷分散手法の開発を行なっていきたい。

謝辞

本研究を遂行するにあたりご指導、ご検討いただいた棟上情報アーキテクチャ部長、弓場知能システム部長、島田計算機方式室長ならびに研究室の同僚諸氏に感謝いたします。

参考文献

[1] Sakai,S., Yamaguchi,Y., Hiraki,K., Kodama,Y. and Yuba,T. An Architecture of a Dataflow Single Chip Processor, Proc. 16th Annual Symp. on Comp. Arch., (1989), 46-53.

[2] 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.

[3] Kodama,K., Sakai,S., Yamaguti,Y. A Prototype of a Highly Parallel Dataflow Machine EM-4 and its Preliminary Evaluation, Proc. of InfoJapan '90, (1990), 291-298.

[4] 坂井修一, 児玉祐悦, 山口喜教. プロセッサ結合型オメガ網を用いた並列計算機の構成, 信学技報 CPSY 89-31, (1989), 1-6.

[5] 坂井修一, 児玉祐悦, 山口喜教. 高並列データ駆動計算機ＥＭ－４における負荷分散方式, 信学技報 CPSY 90-47, (1990), 55-60.