# Javaオンチップマルチプロセッサの設計*1

6K－1

前田剛志*2
大阪工業大学大学院情報科学研究科

高橋義造*3
大阪工業大学情報科学部

## 1．はじめに

近年半導体の集積度が飛躍的に向上し、チップ上に複数のCPUを搭載させたオンチップマルチプロセッサの開発が可能となった[1]。そこで、本研究ではマルチスレッド機能を持つJavaに注目し[2]、Javaプログラムのバイトコードを直接実行できる複数台のプロセッサJava Virtual Machine[3][4][5][6]（以下JVMとする）を結合した、オンチップマルチプロセッサを設計し、FPGAに実装し、プロセッサ数台の規模でTSP問題を処理させた場合の性能について検討した。

## 2．アーキテクチャ

### 2.1.基本構成

本マルチプロセッサでは複数のJVMを結合し、複数のスレッドを並列に実行するための結合方式と同期機構を開発した。

### 2.2.マルチスレッドの実行

Javaにはオブジェクトにスレッドを複数実装することにより簡単にマルチスレッドを実現できる機能がある。そこでこのマルチスレッドを利用し、各スレッドを別々のプロセッサで並列に実行できるようなマルチプロセッサシステムを設計した（図1）。このシステムは、巡回セールスマン問題（以下TSPとする）などの探索問題に対して大きなパフォーマンスの向上を図ることができる。

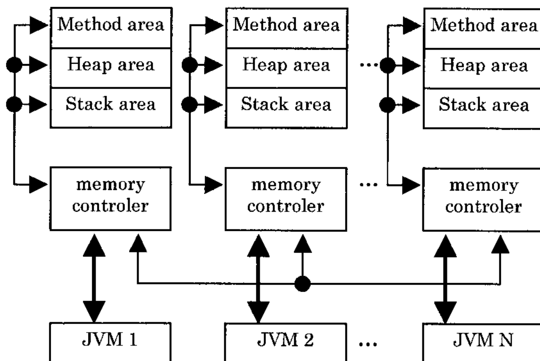

図1　マルチプロセッサの構成

### 2.3.同期機構

スレッドとは、お互いに独立して実行可能なプログラムの一部であるが、同期が必要な場合がある。これはスレッド間でデータを共有している場合、ロックなどを行い排他的に処理させる必要があるためである。Javaには同期制御のため、メソッドにsynchronizedというアクセスフラグが用意されており、これを付加することで同期メソッドを実現することが出来きる。この同期メソッドはマルチスレッド間で排他的に実行されることが保証される。同期メソッドを実行する際、他のスレッドによりロックされていない場合は実行できるが、ロックされている場合には、ロックが解除されるまで待たなければならない。

今回作成したマルチプロセッサでは、これらの同期の問題を解決するためHeap内に同期の必要なメソッドやフィールドを制御するメソッド制御テーブル（図2）を作成し、これを参照することにより同期メソッドを予め定められたプロセッサで実行するようにした。

メソッド制御テーブル

| method名 | type | access flag | pno | ADR |
|---|---|---|---|---|
| GetRange | ()myRange | synchronized | 0 | 2424 |
| SyncData | (int int)int | synchronized | 1 | 343 |
| handle | (int,int,myState)V | private | 0 | 3456 |
| tsp | (int,int,myState)V | public | 0 | 563 |

図2　メソッド制御テーブルの構成

図2において、method名はプロセッサがこの名前を参照することで必要なメソッドの情報を探すためのものである。typeはメソッドの引数や戻り値の情報である。access flagは、そのメソッドがpublicであるかsynchronizedであるかなどの情報を示すもので、この値を調べることでメソッドのアクセスを制御する。pnoはメソッドを実行するプロセッサの番号であり、コンパイラにより決定される。ADRは自分のメモリ上のmethodのアドレスである。

メソッド制御テーブルは新しいメソッドを読み出す命令であるinvokevirtualを実行する場合に使用される（図3）。invokevirtualを実行する時、プロセッサは起動したいメソッド名のaccess flagを調べ、読み出したいメソッドが同期の必要があるかどうかを確認する。同期の必要が無いとわかった場合、メソッド制御テーブルに格納されているメソッドのアドレスを参照し、新しいメソッドを実行する。同期メソッドだと分かった場合、pnoに格納されている番号のプロセッサに引数をパケットとして送信する。そしてパケットを受け取ったプロセッサは実行中のフレーム処理が終わり次第、要求されたメソッドの処理を行い、結果をパケットで元のプロセッサに返す。なおpnoが自分自身の番号なら、自分で同期メソッドを実行する。

さらに同期メソッドを他のプロセッサで実行させ結果を待っている間に、stackエリアに積まれたフレームのうちにデータの依存性などがなく、先に処理することが可能なフレームを実行させることで待ち時間を隠蔽するようにした。

*1 Design of a JAVA On-Chip Multiprocessor

*2 Takashi Maeda, Graduate School of Information Science, Osaka Institute of Technology
*3 Yoshizo Takahashi, Faculty of Information Science, Osaka Institute of Technology

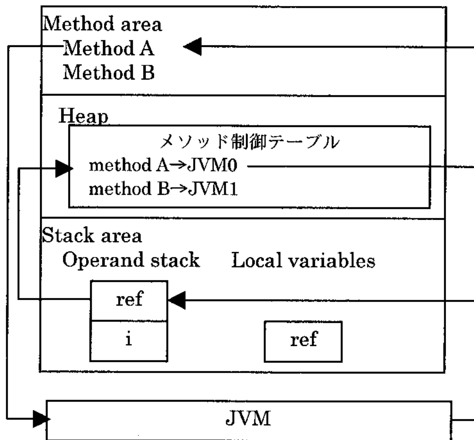


図３　invokevirtual methodA の実行

## 2.4. 結合方式

プロセッサ間の結合方式は、同期メソッドの引数と戻り値のやり取りをリング上でパケット（図４）を循環させることで行うようにする。引数の送信パケットは送り先のプロセッサ番号 pno、送り主のプロセッサ番号 rno、パケットの長さ length、メソッド名、引数から構成される。戻り値の送信パケットは送り先のプロセッサ番号 pno、パケットの長さ length、メソッド名、戻り値から構成される。

・引数の送信パケット

| pno | rno | length | method 名 | 引数１ | … | 引数 N |
|---|---|---|---|---|---|---|

・戻り値の送信パケット

| pno | length | method 名 | 戻り値 |
|---|---|---|---|

図４　パケットの構成

## ３．TSP 問題による性能評価

性能評価のため、TSP 問題を処理する Java のマルチスレッドプログラムをマルチプロセッサ上で実行させる場合を考察した（図５）。このプログラムは、巡回する都市のうち２番目に訪れる都市を各スレッドに割り当て、各自それ以降の探索を行うようになっている。例えば４つのスレッドで都市 A,B,C,D,E,F,G,H,G の最短経路を探索する場合、すべてのスレッドで最初に都市 A を訪れるものとし、次に Thread1 は都市 B,C 訪れ、Thread2 は D,E を訪れ、Thread3 は都市 F,G を訪れ、Thread4 は都市 H,G 訪れるようにし、それ以降の経路を各自並列に探索させるようにした。

TSP の処理のうち、各スレッドが２番目にどの都市を訪れるか決めるメソッド GetRange と、探索終了後、探索結果が最短経路かどうか確認し、最短経路なら経路情報と距離を保持するメソッド SyncData は同期が必要なため同期メソッドである。それ以外の都市を探索するメソッド handle,tsp は同期作業が必要なく、並列に処理させる。

TSP プログラムでは、各スレッドの処理のうち同期メソッドを実行する割合は都市数に対して指数関数的に減少する。都市数が１０を超えると、各プロセッサにおける同期制御に関するコストは１％以下になる。したがって十分大きな都市数ではプロセッサ数に見合う速度向上が得られる。

```
private synchronized myRange GetRange(){
    if(curLoop>=endLoop)return null;
    myRange ret = new myRange();
    ret.start=curLoop;
    curLoop+=(endLoop-startLoop)/Threads;
    ret.end=(curLoop<endLoop)?curLoop:endLoop;
    return ret;
}
public synchronized int SyncData(int Min,int ThreadNo){
    if(TrueMin>Min){
        TrueMin=Min;
        MinThreadNo=ThreadNo;
    }
    return TrueMin;
}
private  void handle (int start,int end,myState mine){
    for(int  i=start;i<end;i++){
        mine.t[1]=i;
        tsp(2,dist[0][i],mine);
    }
}
public void tsp(int n,int distance,myState mine){
    int i,j,len;
    int passed[]=new int[colMax];
    if(n==colMax){
        len=distance+dist[0][mine.t[n-1]];
        if(len<mine.min){
            mine.min=SyncData(len,mine.id);
            for(i=0;i<colMax;i++){mine.tour[i]=mine.t[i];}
        }
        mine.t[n-1]=0;
        return;
    }
    for(i=0;i<colMax;i++){
        if(passed[i]==0){
            mine.t[n]=i;
            len=distance+dist[i][mine.t[n-1]];
            if(len>=mine.min)return;
            tsp(n+1,len,mine);
        }
    }
}
```

図５　TSP マルチスレッドプログラム

## ４．まとめ

Java の特性を活かしたマルチプロセッサの設計を行った。今後は１０万ゲートの FPGA に４台程度のプロセッサを搭載することを目標に、VHDL による設計を進める予定である。

## 参考文献

[1] L. Hammond, et al, "The Stanford Hydra Cmp", IEEE Micro, Vol.20,No.2, pp-71-84, April,2000
[2] S. Oaks, H. Wong, JAVA Threads, O'REILLY, 1999
[3] J.Meyer,T.Downing,JAVA Virtual Machine,O'REILLY, 1997
[4] Andrew S. Tanenbaum, STRUCTURED COMPUTER ORGANIZATION Fourth Edition, PRENTICE HALL,1999
[5] J.Michael O'Connor, Marc Tremblay, PICOJAVA-1: The Java Virtual Machine in Hardware, IEEE MICRO, Vol.17, No.2, pp.45-53,1997
[6] H.McGhan,O'Connor,"PicoJava: A Direct Execution Engine For Java Bytecode",IEEE Computer,Vol.31,No.10,pp.22-30, 1998