マルチCPU・LISP処理系 SYNAPSEについて

中西正和，加藤良信，伊藤精二，松井祥悟
（慶應義塾大学　理工学部　数理科学科）

1. はじめに

ハードウェアのコストが年々急激に下がって、従来の資源の有効利用の目的が少しずつ変化していることは動かし難い事実である。今までは1台のCPUの時間を多くの利用者で分割して、高価なCPUを有効に利用してきた。しかし今では1人の利用者にCPUを1つ与えてしまうほうが、ソフトウェアのコストなどの面ではるかに有利になるばかりでなく、数台のCPUを与えることで計算時間の短縮も期待できるようになった。しかし、メモリに関しては、CPUと異なり、コストが下がればそれだけ大容量のものが要求されてくるので、メモリを利用者毎に分割してしまうのは必ずしも得策ではない。特にLispにおけるメインメモリの必要量は組み合わせ数学的に増加する傾向がある。また、CPUの共有で利用者が受ける論理的なサービスはほとんどないが、メモリの共有は、利用者間通信等の多くの利点が存在する。マルチCPUのLispマシンSYNAPSEは、このような考え方のもとに設計された。

大体の構成を図1に示す。L.P.U. はList Processing Unit, G.C.U. はGarbage Collection Unitである。共有メモリは，具体的には，一本のfree listである。G.C.U.はこれを管理する。L.P.U.に付属しているメモリは，利用者固有のものである。また，各L.P.U.はプラグインで共有メモリに接続され，独立して他の共有メモリ，または個人用メモリに接続することができるように設計されている。（図2）各L.P.U.およびG.C.U.のためのCPUとしては，MC68000を採用している。

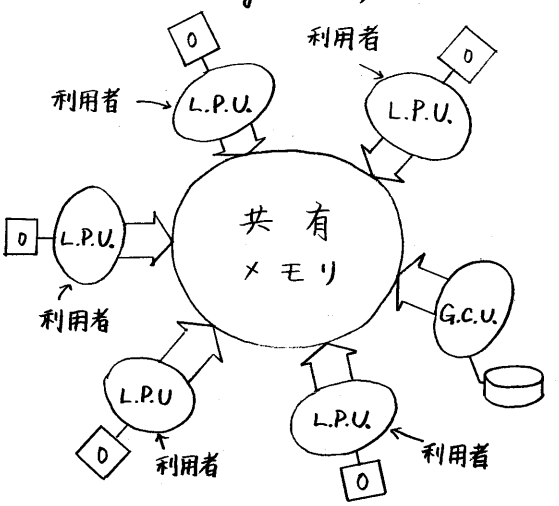

図1　SYNAPSEの概要

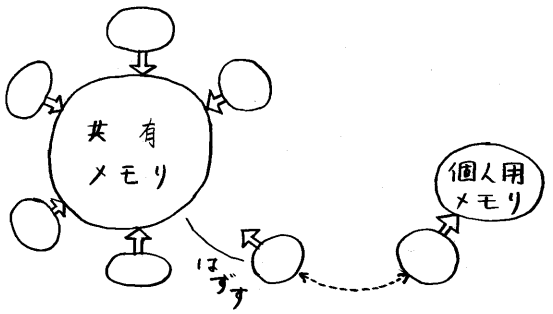

図2　L.P.U.のパーソナル化

2. SYNAPSEのハードウェア

SYNAPSEは，MC68000　16ビットマイクロプロセッサを中心としたLISP専用システムである。ハードウェア，ソフトウェア，ともに何の制

約もなく，一から手がけたという経過から，MC68000の特徴であるアドレス空間の広さ，処理スピードを十分に生かし，ソフトウェアとしてのLISPインタプリタを十分考慮したハードウェア構成となっている。

SYNAPSEは，64KダイナミックRAMで構成された8Mbyteの大容量共有メモリとそれを共通にアクセスできる数個のL.P.U.と一個のG.C.U.とから成るマルチCPUシステムである。（図3） 各L.P.UとG.C.UはMC68000を中心とし，ローカルメモリ，I/Oなどから構成され，バス・アービタを通して，共有メモリをアクセスする。（図4） 各L.P.U.は，共有メモリをLISPデータ領域としてリスト処理を行ない，G.C.U.は各L.P.U.が出すゴミを一手に引き受けて，ガーベージコレクションを行なう。バス・アービタは，共有メモリを，各L.P.U.とG.C.U.に対し平等にアクセスできるよう，優先順位なしにバスを割りふりする。必要ならば，非可分な処理を実現するため，一定時間バスを一つのユニットに固定できる。MC68000は，非同期バスで，メモリアクセスは一種のハンドシェイクと考えられる。したがって，バス・アービタによる共有メモリへのアクセスの待ち時間は，各CPUに対してバス・サイクルを伸ばすという方向に働き，したがって，各CPUから見ると，少々アクセスタイムの長いメモリ装置へのアクセスと見える。

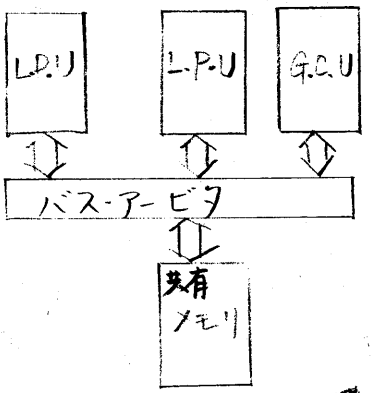

図3 SYNAPSEの構成

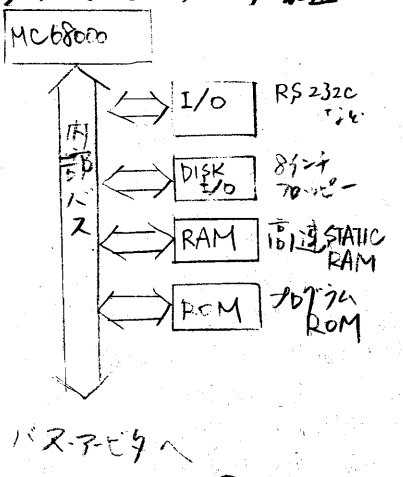

図4. L.P.Uの構成

各L.P.U.は，外部記憶装置として，8インチ・フロッピ・ディスク装置を持つ。またG.C.U.はハード・ディスク装置を持つ。共有メモリは，1byteを8bit＋パリティ1bitの9bitも構成とした1ボード512Kbyteの RAM・UNITを16枚使用した8Mbyteメモリで，ボード内でパリティ・チェックを行なっている。MC68000のアドレス空間は24bit×1byte，すなわち16Mbyteの空間をもつが残り8Mbyteは，将来の拡張にそなえている。

3. ガーベージコレクション

ハードウェアの構成で述べた様に，SYNAPSEではガーベージコレクタ，LISPインタプリタに対して，それぞれ1つずつのCPUを割り当てている。この為，ガーベージコレクションのアルゴリズムは並列動作を前提にして検討された。

現在，並列ガーベージコレクションのアルゴリズム[6,7]として知られているものには，Steal[1]，Dijkstra[2]，Kung & Song[3]などがあるが，これらはいずれも実

装されてはいない。我々は、これら3つに検討を加え，最も効率の良いKung & Songのアルゴリズムを実験的にインプリメントする事にした。

4. Kung & Songのアルゴリズム

Kung & Songのアルゴリズムは，1) root insertion phase，2) marking phase，3) collecting phase　の3つのphaseから成り立っており、マーキング法は、基本的にはスタックを用いた再帰的リンクたどり法である。ただし，マークとして，そのノード状態に対応して4つの色を使っている。（表1）

表1．Kung & Song法のノードと色

色	状態
white	garbage または unmark
off-white	free node
gray	marking中．link書き換え先となったnode
black	生きているnode

並列ガーベージコレクションの特徴の1つとして，marking中に新たに、rootから到達可能となったノードを、インタプリタが、ガーベージコレクタへ知らせなければならないことがあげられるが、スタックを用いた並列ガーベージコレクションでは，この操作はインタプリタが，新たにrootから到達可能となったノードを，スタックへ積んでやるという操作に置き換えられる。この操作のため，marking中にはインタプリタ，ガーベジコレクタの両者がスタックをアクセスすることになり，アクセス競合が起こりうる場合がある。

Kung & Songは，このアクセス競合の問題をdequeによって解決している。つまり，インタプリタ，ガーベジコレクタが別の口をアクセスすることによって競合をさけている訳である。ところがこのアルゴリズムでは，dequeの実現ならびにそのemptinessの判定に問題がある。そのうえ，インタプリタが複数個ある場合には，dequeのインタプリタ側の入り口で，やはり競合が起こる事になり，SYNAPSEの最終的な構成では，このままの形では使用できない。このため，我々はハードウェアの支援の下に，アルゴリズムに変形を加えインプリメントした。

5. アルゴリズムの拡張と実現

スタックのアクセス競合を解決するため，Kung & Songはdequeを使用したが，SYNAPSEでは，3つ以上のプロセッサが接続されるため，dequeでは解決できない。結局，我々は，スタックアクセスをクリティカルセクションとして行なうという方法を選ぶ事になった。しかし，セマフォなどを使った方法では，効率の点で問題があり，ハードウェアの支援を求めることにした。幸い，ハードウェアで比較的簡単に，非可分操作が実現できたので，陽にクリティカルセクションを設けずに，非可分操作を許し，かつ，個別封鎖が起こらないよう，ハードウェアを工夫した。この他，非可分操作が許されたため，アルゴリズム中に，いくつか改良の余地が見つかった。最終的なアルゴリズムをappendix Iに付ける。

6. LISPインタプリタ

今回インプリメントされた処理系は、変数bindingの実験用として，PDP11 UNIX(V6)上に作られているKLISP[4]のshallow binding版をもとに作製された。特徴としては，Full Funargをサポートするため，Baker[5]によるa-listを残したshallow bindingを実現している点があげられている。

KLISPは，もともとミニコン用の処理系であるため，メガセル級のSYNAPSE用としては，当然力不足である。このため，効率の評価が終った後に，新しい処理系を作る予定である。

7. SYNAPSE ファイルシステム

SYNAPSEのためにUNIX風の階層ディレクトリをサポートしたファイルシステムを作製した。外部記憶としては，256K程度のフロッピディスクから10M程度のハードディスクまでサポートできる。

SYNAPSE ファイルシステムでは，ファイルの拡張子名が"DIR"であるファイルをディレクトリファイルとし，親のディレクトリのもとに通常のファイル同様に作成可能にすることにより，階層ファイルシステムを実現した。

OPEN，READ，WRITE，CLOSE等の通常のファイル操作の他に現在のディレクトリを変更するためのCHDIRと，新しくディレクトリを作成するためのMKDIRをシステムコールとして用意した。

8. 企画中のもの

現在，共通メモリまわりのものとしてはL.P.U.とG.C.U.だけであるが，共通にサービスするリスト処理用の基本プロセッサを配置することで効率を上げることを考えている。また，共有メモリといっても，各ユーザに属するものは，他のユーザに属するものとはなり得ないが，ユーザシステムが作成したリストをROM化する（ROMリストという概念を導入する）ことで情報の有効な活用を期待することも考えている。

9. おわりに

本プロジェクトは1981年度より始まり，初年度にはハードウェア及びソフトウェアの基本設計，開発ツールの製作が行なわれた。今年度は，ガーベジコレクタプロセッサ，インタプリタプロセッサの2つのプロセッサを持つ，マルチCPU，LISPマシンを製作すべく，プロジェクトが進められた。現在ソフトウェアの開発はPDP11/60 UNIX上で行なわれており，SYNAPSEとPDP11/60は16bitパラレルI/Oで接続されている。また，RS232Cにより，CP/Mマシンとも補助的に接続されている。ハードウェアは基本的設計と基本実験が終了した段階で，共有メモリ＋CPU1つという構成で開発を行っている。我々が考えている最終的な構成は，インタプリタプロセッサを複数個，有するものであるが，これは来年度末に完成する予定である。

今回の発表は，プロジェクトの概要，進行状況，また，今後の計画などを紹介した。

参考文献

1) Steele, G.L.: Multiprocessing Compactifying garbage collection, CACM, 18(a), PP.495-508(1975).

2) Dijkstra, E.W. etal.: On-the-fly garbage collection: An exercise in Cooperation, Lecture note in computer science, 46, PP.43-56, spring-verlag. New york, (1976).

3) Kung, H.T. & Song, S.W.: An efficient parallel garbage collection system and its correctness proof, Proc. 18th Annual Symposium on Foundation of Computer Seince, PP.120-131(1977).

4) 大田一久: shallow bindingによるLISP, 学位論文, 慶應義塾大学工学部数理工学科(1988).

5) Baker, H.G.Jr: Shallow binding in LISP 1.5, CACM, 21(7),

PP.565-569(1978).
6) Cohen, J: Garbage Collection of Linked data structure, Computing Surveys, 13(3), pp.34-367(1981).
7) 日比野 靖: ガーベージコレクションとそのハードウェア, 情報処理, 23(8), pp.730-741(1982).

appendix I

```
GC1. [ROOT INSERTION PHASE].

begin
    for i<-1 until R
        do PUSH(i) od
end
```

```
GC2. [MARKING PHASE].

begin
    MARKING:=true;
    while the stack is empty and stack save flag is on
        do
            POP(n);
            s:=n.left;
            if s!=NIL and s.color!=black then
                begin
                    s.color:=black;
                    PUSH(s);
                end
            s:=n.right;
            if s!=NIL and s.color!=black then
                begin
                    s.color:=black;
                    PUSH(n);
                end
        od
    MARKING:=false;
end
```

```
GC3. [COLLECTING PHASE].

begin
    for i<-R+2 until M
        do
            if i.color=white then
                APPEND(i);
            else if i.color!=off-white then
                i.color:=white;
        od
end
```

```
PROCEDURE APPEND(n)

begin
    n.color:=off-white;
    n.left:=NIL;
    n.right:=NIL;
    [FREE.right].right:=n;
    FREE.right:=n;
end
```

```
PROCEDURE PUSH(n)

begin
    INDUSBL:=true;
    if the stack is full then
        begin
            call exception;
        end
    push n onto the stack;
    INDUSBL:=false;
end
```

```
PROCEDURE POP(n)

begin
    INDUSBL:=true;
    n:=the cell at the top of the stack;
    n.color:=black;
    remove n from the stack;
    INDUSBL:=false;
end
```

```
OPERATION LPa:Add a left (or right) outgoing edge from
              an active cell m to an active cell n

begin
    m.left (or right):=n;
    if MARKING and n.color!=black then
        begin
            INDUSBL:=true;
            n.color:=black;
            PUSH(n);
            INDUSBL:=false;
        end
end
```

```
OPERATION LPc:Make a free cell active

begin
     CREATE(n);
     if MARKING then
          begin
               n.color:=black;
               PUSH(n);
          end
end



PROCEDURE CREATE(n)

begin
    LOOP: INDUSBL:=true;
     if FREE.left=FREE.right then
          begin
               INDUSBL:=false;
               go to LOOP;
          end
     n:=FREE.left;
     FREE.left:=[FREE.left].right;
     n.left:=NIL;
     n.right:=NIL;
     n.color:=gray;
     INDUSBL:=false;
end
```