Parallelism in Simple Algebra Systems

DAVID BARTON
Florida International University

The paper presents a multiprocessor computer suitable for certain simple kinds of algebraic manipulation and shows how the machine yields near linear speed improvements over a single processor machine while allowing the storage of huge expressions.

INTRODUCTION

In an earlier paper[1] a multiprocessor computer was proposed as a device suitable for certain simple kinds of algebraic manipulation. The primary purpose of the machine was to enable very large algebraic expressions to be stored in immediate access memory, and to achieve this some 1000 CPUs each with 128K bytes of memory communicating on a single bus were employed. With such a huge number of processors it was to be expected that substantial parallel computation would be possible and significant speed improvements compared with a single processor would be realised, together with the expanded available memory. However, the machine obtained only a small fraction of the speed improvement possible because too great an emphasis was placed on efficient utilisation of storage; indeed using 1000 CPUs speed was improved only by a factor a little greater than 2. The present paper addresses the same problem in manipulative algebra using a similar but not identical machine and, by making less efficient use of the memory, shows how speed improvement factors near 1000 are possible for certain kinds of manipulative operations. Thus true parallelism is achieved with a large number of processors that results in a near linear speed improvement over the single processor model.

THE EXPRESSION SET AND STORAGE MECHANISM

The class of algebraic expressions discussed in this paper is the ring of Poisson Series PS, where

$$\mathrm{PS} = \{y \mid y = \Sigma\, \mathrm{P}(x_1, \ldots, x_n) \begin{matrix}\sin\\ \cos\end{matrix} (L(u_1, \ldots, u_m))\} \quad (1)$$

and $P(x_1, \ldots, x_n)$ is a polynomial over the rationals in x_i $(i = 1, \ldots, n)$ and $L(u_i, \ldots, u_m)$ is a linear expression in u_j $(j = 1, \ldots, m)$ with integer coefficients. The summation is finite and the canonical form for the Poisson Series in which products of sine and cosine are linearized is employed. Taking a particular polynomial term from this expression we have

$$\frac{p}{q} x_1^{r_1} x_2^{r_2}, \ldots, x_n^{r_n} \begin{matrix}\sin\\ \cos\end{matrix} (k_1 u_1 +, \ldots, + k_m u_m),$$

where $p, q, r_1, r_2, \ldots, r_n, k_1, \ldots, k_m \in Z$. Thus an expression of the form (1) can be represented as a collection of vectors $\mathbf{v} \equiv (r_1, r_2, \ldots, r_n, k_1, \ldots, k_m, \gamma, p, q)$ $\gamma = 0$ or 1 as the term is sine or cosine. For applications of serious interest the values r_i and k_j are typically restricted as follows

$$0 \leqslant r_i \leqslant 10 \quad \text{and} \quad -10 \leqslant k_j \leqslant 10 \quad \text{with} \quad n = m = 6.$$

Thus an element v can be represented in about 25 bytes of storage. The algebraic manipulative problem is simply to add and multiply such expressions as (1) while preserving their canonical form. On a single processor with small expressions this is of course a simple problem. However, when expressions with millions of terms are involved it is of greater interest.

It was proposed in Ref. 1 that these huge expressions be distributed over about 1000 separate computers, each with a processor and local storage in such a way that the entire expression was as nearly equally divided, on a term-for-term basis, as possible between the CPUs. It was precisely the constraint that terms be equally divided between processors that led to the poor CPU utilization that occurs with the method. To understand this point it is necessary to examine the details of the addition and multiplication procedure described in Ref. 1. When two expressions were to be combined under either addition or multiplication each machine would generate from the component parts of the operands stored in its memory a sequence of new terms of the result. To ensure total cancellation and equal distribution of these terms, each term as it was produced was broadcast to *all* the other processors. All processors received every term of the result individually and searched to see if the term would cancel with any term already present in their memories. This search takes the majority of the computation time. If a match was found then cancellation took place, but failing that the terms were handed out to processors on a round-robin basis. In order to reduce the inter-CPU communication on the bus the machines were maintained in synchronization by a pattern of interrupts so that a single cycle of operation during which a term was broadcast, received, processed for cancellation and finally accepted by a CPU appeared as described in Fig. 1. This cycle is divided into three parts.

Part 1. Terminating when one machine broadcasts a single term to all the others. During this time all the CPUs can in principle be working on genuine parallel computation. That is to say all the machines are busy doing *different* useful computation such as the preparation of the next term of the result or the cancellation of existing terms in the result.

Part 2. During which each machine searches the set of result terms so far stored in its memory to discover if the newly broadcast term will cancel with any of those present. Because each processor has *at all times* almost the same number of terms of the result as any other and assuming, for large expressions, an even distribution of terms that cancel this search will take almost the same length of time in each processor. To the extent that the times are unequal, and since the duration of this activity is limited by the time for a CPU to find a match or the

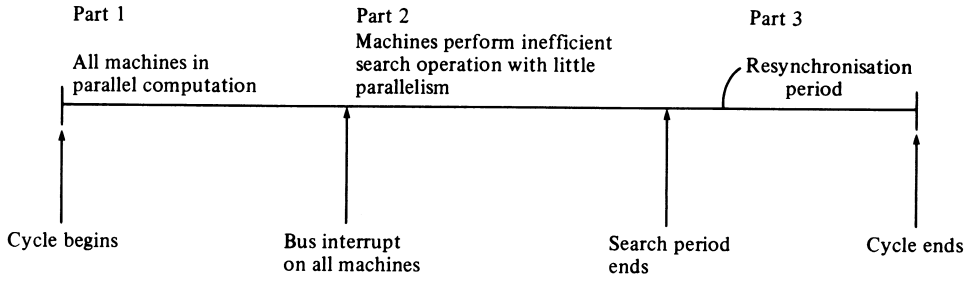

Figure 1.

time of the slowest one to finish, whichever is shorter, some CPUs may be able to use the remaining time to do useful parallel work. However, given the uniform distribution of terms, the time available for such activity is very short. The parallel activity during the actual search, however, is not fully rewarding. Assuming q terms of the result per machine, each CPU will perform about $\log_2 q$ comparisons before ending its search. On a single processor with $1000q$ terms only $\log_2 1000q$ comparisons are needed. With $q = 4000$ for a typical large expression we see that the improvement in search time is only a factor of about 2 using all 1000 processors.

Part 3. During this time, which is very short, processors simply resynchronise and arrange to begin the cycle again.

Thus during Part 1 of the cycle full parallel processing is obtained while in Part 2 very little parallelism is realized. The remainder of the paper describes what can be done about this.

COMPUTER CONFIGURATION AND BUS CHARACTERISTICS

Following Ref. 1 the computer configuration employed here again comprises a set of N computational units each connected to a single bus B and inter-connected by a bus L in a loop. The latter bus allows concurrent DMA input and output simultaneously to all units so that they can function as a huge circular shift register. This arrangement is presented in Fig. 2. The facilities provided by Bus B, while superficially similar to those proposed in Ref. 1, differ substantially from the earlier machine. They are the following:

1. Interrupt facilities
   - 1.1 A 'B' interrupt generated on a single computer when the bus is given to that computer by the bus contention discrimination logic so that the unit can transmit a significant amount of information.
   - 1.2. An 'X' and 'Z' interrupt generated on all units simultaneously when initiated by any single unit. Used to broadcast small quantities of information from one to all units.
   - 1.3. A 'D' interrupt generated on a single unit when a transfer initiated via a 'B' interrupt and bus B to that unit is done.
   - 1.4. A 'Y' and 'F' interrupt generated on every unit simultaneously and initiated when all units have requested it. Used to synchronise activity on all units.
   - 1.5. An 'S' interrupt generated on a single unit when requested by that unit and used to prepare the bus for information transfer via an 'X' interrupt.
2. The interrupt enabling facilities are as follows:
   - 2.1. Each unit can place its vote to enable interrupts in all units itself included. Interrupts can be enabled either singly or all together. That is to say a unit can call for only 'X' interrupts to be allowed anywhere or for interrupts of all types to be allowed anywhere. An interrupt of any type can occur only if all units have voted to enable it.
   - 2.2. Interrupts occur according to a priority order among those requests outstanding at the instant the bus becomes available. The priority order from most to least significant is:
     D, X, Z, Y, F, S, B
3. When an interrupt occurs then
   - 3.1. Votes by computers that are interrupted are cancelled. Interrupts are inhibited on all machines and the interrupted units must vote before any unit can receive another interrupt.
   - 3.2. At any time any unit can request an interrupt of any kind and the request will be remembered and will be processed only when interrupts of that kind are enabled.
   - 3.3. Requests for interrupts may be cancelled only by the requesting unit and this may be done at any time prior to the request being granted. Arbitration logic assigns the bus in a round robin fashion among those units with an outstanding request subject to the priority scheme.
4. Address and data lines are provided on the bus that may be written by the unit to which the bus is currently assigned and that may be read by any unit at any time. The value on these lines determines which machine receives the 'D' interrupt if a transfer is initiated via a 'B' interrupt.

Each computational unit consists as before of a processor, memory, transmit queue and a receive queue. The machine configuration is presented in Fig. 3.

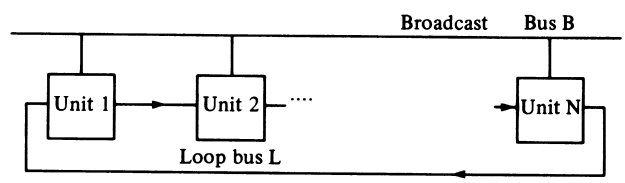

Figure 2.

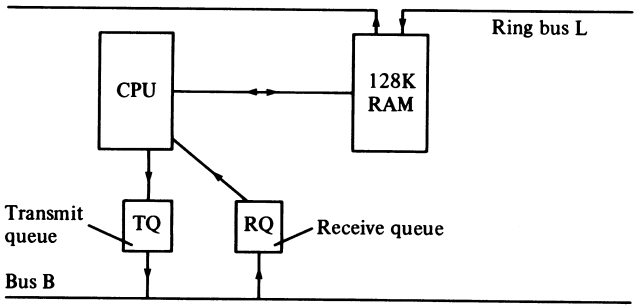

Figure 3.

EXPRESSION STORAGE AND MANIPULATIVE TECHNIQUES

As explained earlier the expressions PS are each represented as a set of vectors

$$v = \{r_1, r_2, \ldots, r_n, k_1, k_2, \ldots, k_m, \gamma, p, q\}$$

stored on the machine of Fig. 2. Each expression is stored in the canonical form to aid simplification, and that canonical form is the form in which all products of trigonometric functions are linearised, all algebraic cancellation has taken place and each individual term (vector v) is stored in processor n_v where the processor number is generated by a hash function H.

$$n_v = H(r_1, r_2, \ldots, r_n, k_1, k_2, \ldots, k_m, \gamma)$$

This canonical form cannot guarantee to distribute the vectors v uniformly over the processors and consequently the aglorithms presented here cannot be as space-efficient as those presented in Ref. 1. But these algorithms are much faster. The expressions PS form a ring and consequently it is only necessary to consider the two operations of additions and multiplication.

Clearly addition is very simple with the proposed canonical form. Each computational unit simply adds the two fragments of the two operands stored in its memory to yield its fragment of the result. No data transmission is necessary, all units operate in parallel on distinct useful calculations and a speed improvement factor comparable with the number of units is achieved. Multiplication is a little more difficult. To form the produce of two expressions, A and B, each unit generates terms of the product locally and concurrently transmits these terms one at a time to the receiving unit determined by the Hash function. Concurrent with the generation of terms each unit will receive terms generated by other units for inclusion in its portion of the product. These terms must be added to the local partial result so far and any cancellation must be implemented. When all units have completed the calculation of the partial results then one of the expressions is rotated via Loop L and the product continues until the entire calculation is finished.

The algorithm is simple enough provided that there is adequate memory to store the result and the intermediate expressions that may be generated. That is, however, a significant reservation. It is perfectly possible for terms to be generated during the course of a product that completely fill a particular unit to which they are allocated by the Hash function and thus prevent that unit from receiving additional terms. Further, if the storage limitation could be ignored terms subsequently allocated to that unit might cancel with existing terms thus causing the space problem to disappear. It is clear that some procedure must be included in the algorithm to ensure that this case, which is by no means unusual, does not lead to failure. Consequently another constraint should be added; namely that the multiplication procedure should fail only when either the intermediate product fills all units, or the final result cannot be stored in the units according to the canonical form. The latter condition simply means that the Hash function is not perfect.

In an attempt to satisfy the proposed constraint the multiplication algorithm can be modified as follows. Any unit while in operation is in one of three states. State S_1 is the state in which a unit is available to receive any term that is transmitted to it by any other unit (itself included). The transmitting unit can discard the term after transmission. State S_2 is the state in which a unit is nearly full. It is available to receive terms only for cancellation purposes. A unit transmitting to another unit in State S_2 must retain the transmitted term until the receiving unit either accepts or rejects it. State S_3 is the state in which a unit is full and unable to accept any new terms at all. This condition is temporary only.

If a term is rejected by a unit in State S_2 because it has insufficient room to store it, the transmitting unit must attempt to retransmit it to other units by directing it at the pth attempt to unit

$$r_{v,p} = H'(r_1, r_2, \ldots, r_n, k_1, k_2, \ldots, k_m, \gamma) + p \bmod N$$

where H' is a hash function distinct from H. Thus each unit receives its own terms via hash function H and may be called upon to assist its neighbours by accepting terms via H' and p. A unit that has accepted terms via H' must attempt to transmit them either via H or via H' to units with a lower p. In this way as saturation (all machines nearly full) approaches the system as a whole will ensure cancellation takes place and ensure that terms are shared across all units.

A problem of indefinite delay can occur close to saturation as follows. Unit A wishes to transmit a term to another unit B but whenever unit A gets the bus it just happens that B is in State S_3 (temporarily unavailable) because B has just received a transfer from a third unit C and has not yet rejected it. A relinquishes the bus only to have C seize it again and transfer to B thus leaving B busy before A tries again. A solution to this is to have any unit that has a term rejected by a machine in State S_2 wait for some randomly determined interval before it tries again. This is, of course, a well-known collision avoidance procedure in broadcast networks.

When multiplication proceeds according to this algorithm the entire system will settle into one of three final states.

Final State 1

All terms in their correct units as defined by H and no unit having anything to transmit. In this case the calculation is complete and the answer is correctly stored. The algorithm has performed successfully.

Final State 2

A non-trivial subset of units exactly full with some terms *not* in their correct units as defined by H. These units are

```
Initialise the local state table (all units in state S1)
Initialise the retransmit and result areas both empty; set TQ and RQ
empty
SEARCHING = False; SEARCH-COMPLETE = False; vote for all
interrupts;
Repeat
  If current state of this unit differs from state shown in local table
    then request S interrupt;
  If (on this unit there are no more terms to generate and RQ is empty
  and the retransmit area is empty)
    or SEARCHING-COMPLETE
    then request F interrupt;
  If ((Result area is full or the retransmit area is full)
    and not SEARCHING)
    or (on this unit there are no more terms to generate)
  then request Y interrupt;
  else drop request for Y interrupt;
  If TQ not empty
    then request B interrupt;
    else drop request for B interrupt;
While RQ not empty do
  Begin
    Get a term from RQ;
    If the term was sent via hash function H
      and (the term cancels with one in the local result area or the term
      can be incorporated into the result area)
      then absorb the term into the result area;
  If the term as sent via hash function H'
    and (the term cancels with one in the retransmit area or the term
    can be incorporated into the retransmit area)
    then absorb the term into the retransmit area;
  Determine the new local state;
  If new state differs from state in local table
    then request S interrupt;
  If term requires an acknowledgement
  {it was received while unit was in state S2 according to the local table}
    then begin
      {in the Transmit status field of RQ}
      Mark its acknowledgement status;
      {in same field}
      Mark it ready for acknowledgement in RQ;
      request the B interrupt;
      end;
    else begin
      Disable all interrupts
      Update the RQ output pointer; {remove term from RQ}
      Enable previous interrupt state on the unit;
      end;
  end
{to avoid collisions on bus all terms in TQ have a brief real time delay
associated with them}
If TQ not full and it is time to release a term from the retransmit area
to TQ
  then begin
    Mark the copy of term in retransmit area as 'in transmit'}
    Place term in TQ together with its destination number
    and p integer;
    Disable interrupts;
    Update TQ input pointer; {insert term into transmit queue}
    Enable previous interrupt state on this unit;
    end
If retransmit area not full and there are more terms of the result
  to generate
  then begin
    Generate a term;
    Disable interrupts;
    If retransmit area not full
      then place term in retransmit area for immediate release;
      else discard term;
    Enable previous interrupt state on this unit;
    end;
Until FINISHED;
```

Figure 4. Task Ω running on all units.

attempting to transmit the incorrectly located terms to their final home units but they are never accepted as the receiving units have no space. By definition of this state no unit has any new term to generate that is to say the product is finished. Clearly this case is improbable but the algorithm has failed and this has to be regarded as a case where the product is too large to store.

Final State 3

Similar to Final State 2 except that at least one unit of the subset has more terms of the product to generate. That is to say the multiplication is not finished and the algorithm has failed.

It is necessary to be able to detect and distinguish the final states. Final State 1 in which no unit has anything to generate or to transmit is easy to detect with a common interrupt. Final States 2 and 3, however, are only attained when every unit has transmitted every term in its transmit area to every other unit allowed by the algorithm and had it rejected by that unit during a period of time when no unit has accepted a term from anyone. That condition is harder to detect. Each unit must measure this time interval itself and it need do so only while it is in State S_2 or State S_3 or when it believes it is finished. While the system is in this mode each unit must keep track of the unit numbers to which each individual term available for

```
{Interrupt handling sequences}
{S-interrupt; generated on a single unit to enable that unit to transmit
its new state via the bus to all other units.
Following the 'S' interrupt all units receive an X interrupt requested
by this procedure}.
Copy current state of this unit to data lines on bus;
Copy unit number of this unit to address lines on bus;
Request an 'X' interrupt;
Vote to allow 'X' interrupts only;
Exit interrupt routine;

{X-interrupt; generated on all units simultaneously to enable them to
read the new state of some individual unit. This interrupt follows
immediately after the 'S' interrupt on the unit broadcasting its change
of state}.
Read new state from the data lines of bus;
Read unit number from the address lines of bus;
Update the table of units/states in the unit;
Vote for interrupts at all levels;
Exit interrupt routine;

{F-interrupt; generated on all units simultaneously when requested by
all units simultaneously. A unit requests this interrupt when it believes
that the multiplication algorithm has completed}.
Set FINISHED flag true;
disable all interrupts;
exit interrupt routine;

{Y-interrupt; generated on all units simultaneously when requested by
all units simultaneously. A unit requests this interrupt when it believes
either
  (a) that because its own local storage areas (RES and RT) are full
      it is possible that the system is stuck through lack of space. When
      all units agree the SEARCHING flag is set to indicate that the
      entire system should record any successful cancellation.
or
  (b) that it has finished and has nothing more to do. However other
      machines may be in trouble so the unit must call for 'Y' to enable
      other units who really need it to obtain relief}.
Set flag SEARCHING true;
Initialise counts for each term in the local retransmit area (RT);
Vote for all interrupts;
Exit interrupt routines;
```

Figure 5a. Interrupt Sequence ω

```
{Z-interrupt; generated on all units when requested by any one unit. If
the entire system has entered the SEARCHING mode via a 'Y'
interrupt and a single unit accepts a transmission of a term then that
unit terminates the SEARCHING mode by requesting this interrupt}.
Set SEARCHING-COMPLETE flag false
Set SEARCHING flag false;
Vote for all interrupts;
Exit interrupt routines;

{D-interrupt; generated on a single unit at the end of a transmission
to that unit, such transmission having been initiated by the 'B'
interrupt}.
  If received term has acknowledgement mark
    then begin
      if the acknowledgement is accept and SEARCHING is true
        then begin
          request Z interrupt;
          Vote for Z interrupt;
          end
        else vote for all interrupts;
      Update records and counts in retransmit area to take account
      of the acknowledgement;
      If necessary set SEARCHING-COMPLETE true;
      Exit interrupt routine;
    end
  else begin
    Mark current state of unit from local state table
    in received state field of RQ;
    place received term in RQ;
    drop request for F interrupt if any;
    If RQ full
    then begin
      Set the unit in state S3;
      request S interrupt;
      Vote for S interrupt;
      end
    else vote for all interrupts;
    Exit interrupt routine
  end
```

Figure 5b. Interrupt Sequence ω

```
{B-interrupt; generated on a single unit following a request by that unit
for the bus. The interrupted unit initiates a bus transfer}.
If RQ contains a term that is ready to be acknowledged {marked in
transmit status field of RQ}
  then Begin
    Place the address of the remote unit on bus address lines;
    Place term on data lines;
    Place acknowledgement status (accept/reject) on data lines;
    Initiate bus transfer;
    Decrement RQ pointer; {remove term from RQ}
    end
else if TQ contains a term whose destination is not presently
  in state S3 according to the local state table
  then Begin
    If destination is in state S1 according to local state table
      then delete term from retransmit area;
    Place the destination address on bus address lines;
    Place term on data lines;
    Initiate bus transfer;
    Decrement TQ pointer; {remove term from TQ}
    end
If TQ is empty and there are no terms in RQ requiring acknowl-
edgement
  then drop request for B interrupt;
  else request B interrupt;
Vote for D interrupt;
Exit interrupt routines.
```

Figure 5c. Interrupt Sequence ω

transmission has been sent and this record must be re-initialised each time any unit accepts any term at all. When a unit determines that it has sent all its terms to all allowed units and had them rejected then it can call for the recognition of Final States 2 or 3 via the common interrupt mechanisms and when all units agree the appropriate interrupt will be generated.

Pseudo code for a program to perform multiplication according to this algorithm is presented in Fig. 4 and 5(a, b, c). It consists of identical programs running in all units and in each unit there is a task Ω to perform background work and a set of interrupt routines ω to handle communication with the bus.

The data flow that is controlled by task Ω and the interrupt routines is illustrated in Figure 6. Data areas that are held in common by Ω and the interrupt routines need to be protected in the usual way to avoid race conditions. Such program has been omitted from the pseudo code to aid simplicity. To help understand the pseudo code it should be remembered that in real time on the bus a 'B' interrupt is always followed immediately by a 'D' interrupt; an 'S' interrupt is always followed immediately by an 'X' interrupt; 'Y' interrupt is usually followed by a 'Z' interrupt but there can be substantial other interrupt activity between these two; an 'F' interrupt is generated when all units have finished or given up. Fig. 7 shows the detailed contents of the transmit and receive queues manipulated by the pseudo code.

PERFORMANCE OF THE SYSTEM

Let us now examine the expected performance of this network of machines and the proposed multiplication algorithm upon the assumptions that the hash function H is near perfect and that bus transmission speeds are significantly higher than processor speeds. That is to say terms are evenly distributed and a unit can get rid of terms much quicker than it can generate them or absorb them. Then, with N units on the bus each containing p and q terms respectively of large expressions P and Q, we see that to generate the product PQ each unit will perform Npq term generations and relocations via the bus and each will receive Npq terms from other units for inclusion in its fragment of the product. Reception of a term implies the processor performs a search to determine if cancellation is possible but, since the unit stores only a fragment of the result, the search will involve fewer comparisons then would be performed by a single processor working on the entire product PQ. Since all N units can work in parallel on all these activities it is clear that a speed improvement over a naively programmed single processor machine will be slightly better than a factor N. In fact approximately $N(1+\log_2 N/\log_2(Nqp))$. Against this must be set the effects of saturation should it occur and the very light interrupt load.

The storage allocation algorithm broadly described above enables significant parallelism unless saturation approaches. As the entire system begins to fill up the multiplication operation will slow down owing to the repeated relocation of terms and the repeated rejection of terms by units already full. It is of course normal for space allocation algorithms to experience difficulty when space is nearly exhausted. This configuration of machines is essentially designed to ensure that very large expressions can be manipulated and consequently with a fixed N saturation will in due course take place. It should be emphasized that the appropriate solution to this problem is to add more units to the bus thereby removing the saturation condition and increasing the parallelism.

The high degree of parallelism can only be maintained using a very-high-speed bus. Indeed a simple calculation shows that the bus must be able to transmit N terms, each to a different destination during the average time it takes for a unit to generate a new term of the product and

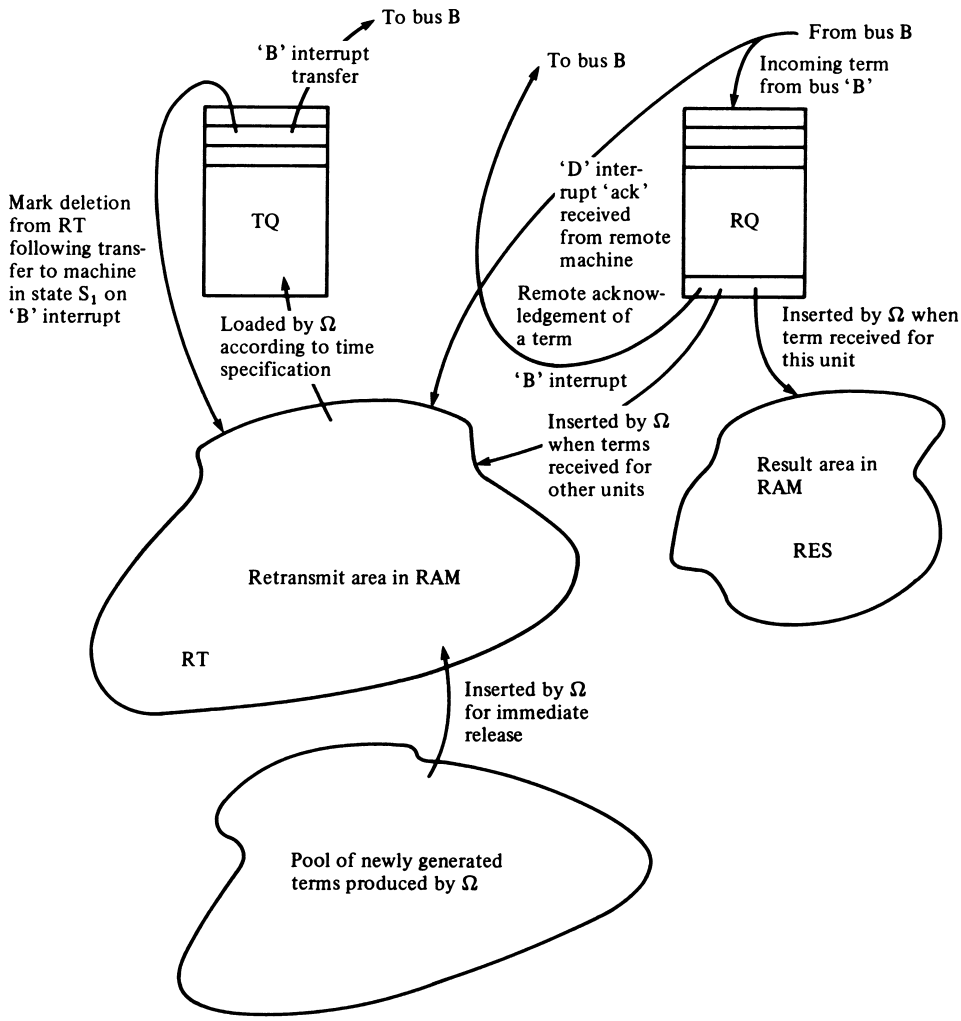

Figure 6.

Transmit Queue Entry
1. Term: 25 bytes of data
2. Transmit Status: (Constant value to mark a normal transmission of a term and not an acknowledgement)
3. P: integer
4. Destination machine number: integer

Receive Queue Entry
1. Term: 25 Bytes of data
2. Transmit Status: (normal transmission not an acknowledgement of a previous transmission, sender rejects previous transmission, sender accepts previous transmission)
3. P: integer
4. Sender machine number: integer
5. Receive State: (S_1, S_2)

S_1 The machine can receive terms freely
S_2 Currently either the result area or the retransmit area are unable to absorb the entire contents of RQ if it were filled.
P An integer specifying how this term must be forwarded.
$P \geqslant 0$ Implies term does not belong to this machine and must be relocated.
$P < 0$ This term belongs to this machine.

Figure 7. Contents of Transmit and Receive Queues

absorb a term generated elsewhere. Since a term and associated data (Fig. 7) consists of about 30 bytes of information it will take of the order of $10N$ memory cycles to move the N terms. The time taken for a unit to generate and absorb a single term is independent of N and of the order of 5000 memory cycles (all that is involved is a little integer arithmetic and a binary search to a depth of 10). Thus we see that full parallelism can be achieved only for fairly small N, about a hundred, and to improve this figure it is necessary to speed up the transmission of terms. A possible way to do this is to interface the bus to each unit in such a way that the two queues TQ and RQ are implemented in hardware using very-high-speed memory. If this can be done using memory 10 times as fast as the RAM on the units then 1000 units can achieve parallelism. The queues TQ and RQ can be quite small (1K bytes) and consequently this technique should not be prohibitively expensive.

The algorithms presented earlier can terminate successfully in Final State 1 or unsuccessfully in Final States 2 or 3. In the latter cases it is most probable that all units contain incorrectly located terms and that these states do indeed reflect the generation of expressions that are too large to store. It is however, possible that these failure states could be reached with some genuine subset of units full and the remaining machines having space available. Such a condition could only be reached if the hash functions H and H' tended to favor the subset of units unfairly when distributing terms of the product. The problem is easily dealt with by extending the pseudo code to use a different hash function when this case occurs but this extension has been omitted here for the sake of simplicity.

REFERENCES

1. D. Barton. *The Computer Journal* **27**, 2, 159–164.