OR Parallel Execution of Horn Clause Programs Based on WAM and Shared Control Information

by

Khayri A M Ali

SICS R88010
Research Report
ISSN 0283-3638

When the system starts up, one processor say p1 will be in the Computation mode and all the other processors will be in the Recomputation mode to follow p1. When a user query is read, all processors in the system start in parallel processing the query. When p1 gets the first choice point, it creates a Split frame, takes the first branch for processing and leaves the other branches on the frame. When any of the other processors gets the first choice point, it has to wait until p1 has created the respective Split frame, then it takes one of the untried branches, if there is any. Processors that have not got untried branches will follow p1 (or one of the other processors that took a branch) to the next choice point.

Another possible way to start up the system is to use another principle in the start phase. For instance, all processors can start processing the top level query in parallel and the one, which gets the first choice point first in time, creates the respective Split frame. Processors that get the first choice point later on take one of the untried branches, if there is any, or follow one of the processors that have got a branch of that Split frame. That is, all processors start in the Computation mode until the first choice point is reached. Then processors that do not find untried branches will switch to the Recomputation mode.

9. Reporting Results

Only processors in the Computation mode report the results.

10. Modifications to WAM Instructions

Information in the global memory is shared between all processors in the system. Some operations require mutual exclusion of more than one global object. A simple implementation for a system with a few tens of processors is to allow one processor at a time to manipulate that information. Here we assume one global lock for the all shared information. The underlaying hardware should support efficient locking operations on shared memory.

As mentioned above each processor can be either in the Computation mode, or in the Recomputation mode. The semantics of each of the four operations: TRY, RETRY, TRUST, and FAIL depends of the processor mode. In the following we describe the semantics of TRY and FAIL operations in the Computation and Recomputation modes. RETRY and TRUST operations are executed within FAIL.

TRY Instruction:

When a processor is in the Computation mode, it executes the TRY instruction as follows.

> *Lock*
> Create a SF of reference s
> set RC, NUB and the respective bit of the current processor on the SF
> Push (1,s) in the HP-stack
> *Unlock*
> Set up a local CP
> Process the first branch

When a processor is in the Recomputation mode, it executes the TRY instruction according to the following actions. Assume a processor p is following another processor q and the depth of p's HP-stack is i.

- p stops itself when either p's Stop flag is "On" or it reaches a choice point at which the processor q has backtracked from and q in the Recomputation mode. Then it looks for a processor in the computation mode to follow .

- If p is following q and p reaches the next CP before q, p has to wait until q reaches that point.

- If the next CP is processed by q only and q is processing the last branch of the CP, p processes that branch of the CP without creating a local CP frame.

- If the next CP has untried branches, p takes one of them and creates a local CP frame.

- If the next CP has no untried branches and there are more than one processor processing different branches of that CP, p creates a local CP frame and follows q.

The above actions can be specified as follows.

```
Lock
If (q in the Recomputation mode And q's HP-stack depth ≤ i) Or p's Stop flag is "On"
Then
        Reset the respective bit in the q's Recomputation vector
        Reset p's Stop flag (if it is On)
        Loop
            While the SFA field of the top element of the HP-stack = NOSF DO pop
            If the SF refers to a processor in the Computation mode
            Then
                Insert itself in the respective Recomputation vector
                Unlock
                Update the B# field according to the selected branch
                Reset the state from CP
                process the selected branch until the next TRY instruction
                BreakLoop
            Else
                Reset its respective bit and decrement RC field of the SF
                Deallocate the SF if RC = 0
                Unlock
                Remove the CP
                lock
Else
        If q's HP-stack depth  = i
        Then Unlock, try again later on
        Else
            If SFA field of the i+1th element of q's HP-stack = NOSF
            Then
                Push the i+1th element of q's HP-stack in the p's HP-stack
                Unlock
                process the branch without creating a local CP
            Else
                Add itself and inc. RC in the SF referred by the i+1th element of q's HP-stack
                If there is untried branches
                Then
                    Update NUB field of the SF
                    Push the i+1th element of q's HP-s. with the selected branch in p's HP-stack
                    Switch to Computation mode
                    Remove itself from q's Recomputation vector
                    Unlock
                    Create a local CP
                    Process the selected branch
                Else
                    Push the i+1th element of q's HP-stack in the p's HP-stack
                    Unlock
                    Create a local CP
                    Process the pushed branch
```

FAIL Instruction

A processor p in the Recomputation mode can execute the FAIL instruction only in the following case: when p is following say q, both p and q processing the same branch, and p fails before q. (P, otherwise, processes a part of the successful path processed by q.) In this case, p performs the following actions.

- P stop following q.

- P backtracks to the latest SF that refers to a processor in the Computation mode which is not q. P follows that processor.

The above actions can be specified as follows.

Lock
Remove itself from q's Recomputation vector
Reset p's Stop bit if it is "ON"
Backtracks with popping the HP-stack until finding an element of the HP-stack referring to a SF
If the SF refers to a processor in the Computation mode which is not q
Then
    Insert p in the respective Recomputation vector
    Unlock
    Update the B# field of the top element of the HP-stack
    Reset the state from the CP
    Process the selected branch until the next TRY instruction
Else
    Reset its respective bit and decrement RC of the SF
    Unlock
    Loop
        Reset p's Stop bit if it is "ON"
        Backtracks with popping the HP-stack until finding an element of the HP-stack referring to a SF
        Lock
        If the SF refers to a processor in the Computation mode which is not q
        Then
            Insert itself in the respective Recomputation vector
            Unlock
            Update the B# field of the top element of the HP-stack
            Reset the state from the CP
            Process the selected branch until the next TRY instruction
            BreakLoop
        Else
            Reset its respective bit and decrement RC of the SF
            Unlock

A processor in Computation mode processes the FAIL performs the following actions.

- It backtracks to the latest SF.

- If the SF refers to untried branches, it takes one of them. If that SF is accessible by only the current processor and there is no more untried branches, the local CP frame and SF are deallocated.

- If the SF does not refer to untried branches, but refers to processors in the Computation mode, one of these processors is followed.

- If no untried branch is found and no processor in the Computation mode is followed, one SF after the other is investigated until a SF having either untried branches or processors in the Computation mode is found.

The above actions can be specified as follows.

Lock
While the SFA field of the top element of the HP-stack = NOSF **DO** pop
If there is untried branches of the SF referred by the current top element of the HP-stack
Then
    Case NextUntriedBranch **Of**
    RETRY:
        Update NUB field of the SF
        Update the B# field of the top element of the HP-stack
        Unlock
        Reset the state from the CP
        Process the branch
    TRUST:
        Case RC of the SF **Of**
        1 :
            Update the top of HP-stack to (Branch#, NOSF)
            Deallocate the SF
            Unlock
            Reset the state from the CP
            Deallocate the current CP frame
            Process the branch
        > 1:
            Update NUB field of the SF
            Update the B# field of the top element of the HP-stack
            Unlock
            Reset the state from the CP
            Process the branch
Else
    Put itself in the Recomputation mode
    If All processors in the Recomputation mode **Then** <u>*Exit*</u>
    If The SF refers to processors following it
    Then set Stop flags of these processors
    If the SF refers to a processor in the Computation mode
    Then
        Insert itself in the respective Recomputation vector
        Unlock
        Update the B# field of the top element of the HP-stack
        Reset the state from the CP
        Process the selected branch until the next TRY instruction
    Else
        Reset its respective bit and decrement RC of the SF
        Unlock
        Loop
            Backtracks with popping the HP-stack until finding an element of the HP-stack referring to a SF
            Lock
            If The SF refers to processors following it
            Then set Stop flags of these processors
                If the SF refers to a processor in the Computation mode
            Then
                Insert itself in the respective Recomputation vector
                Unlock
                Update the B# field of the top element of the HP-stack
                Reset the state from the CP
                Process the selected branch until the next TRY instruction
                BreakLoop
            Else
                Reset its respective bit and decrement RC of the SF
                Unlock

11. Start Up

When the system starts up, p1 will be in the Computation mode and all the other processors will be in the Recomputation mode as shown in Figure 7. When a user query is read, all processors start in parallel processing the query. P1 only is allowed to create the first SF when it gets the first CP, i.e. when p1 executes the first TRY instruction. Any one of the other processors will suspend within TRY instruction if it executes the first TRY instruction before p1.

Implementation of the other idea is as follows. Assume that, there is at least one choice point in the user program and there is a global cell. Initially, the global cell contains a special value, say "0", which is not a processor name. When the user query is read, all processors process the query in parallel. The first in time, say p, that gets the first TRY instruction, writes its name in the global cell and creates the first SF. When any of the other processors gets its first TRY instruction, it finds the name of the processor that has created the first SF in the global cell. Then it can access that SF via the first entry in the p's HP-stack.

12. Bounded Global Storage

Split frames and HP-stacks reside in the global memory. The number of used SFs is equal to the number of active CPs in the search tree. This number depends on the user program. The maximum size of a HP-stack is equal to the number of CPs in the deepest path of the search tree. It also depends on the user program. In this section we explain how the system works when the global storage is limited.

Assume that the global storage is bounded to M SFs and n HP-stacks of size L each. One possible solution is as follows.

- When a processor in the Computation mode is going to create a new SF and detects shortage of SFs or "L-1" entries in its HP-stack, it pushes a special entry (**Stuck**) in the HP-stack, pushes **Mark** entry in the local CP stack, and switches to sequential execution mode. A processor in the sequential mode processes its current job sequentially on the local memory only as conventional WAM. When it finishes that job and backtracks to the Mark entry, it switches to parallel execution mode and works again on the global SFs.

- When a processor in the Recomputation mode reaches a Stuck entry of a HP-stack, it stops following any processor and removes itself from all SFs accessible by it. This processor never gets a new job.

- Processors that do not discover shortage of global storage proceed in parallel.

This solution reduces the number of working processors in the system and restricts the parallelism depending of the values of M and L, and the user program. When a processor switches to the sequential execution mode, it processes its job on the local memory with the same performance as sequential WAM.

Other solutions are possible. For instance, a solution that is based on processors in the Recomputation mode that discover shortage of global storage may backtrack and follow other processors in the Computation mode. Hopefully, when they find a job, some storage is freed up by the other processors.

If the first solution is taken, the TRY and FAIL instructions are specified as follows. Added parts to the above specification are underlined.

TRY Instruction:

When a processor is in the Computation mode, it executes the TRY instruction as follows.

```
If processor in the parallel execution mode
Then
    If There is a free SF And HP-stack depth < L-1
    Then
            Lock
            Create a SF of reference s
            set RC, NUB and the respective bit of the current processor on the SF
            Push (1,s) in the HP-stack
            Unlock
            Set up a local CP
            Process the first branch
    Else
            Push( , Stuck) in the HP-stack
            Unlock
            Switch to sequential mode
            Push(Mark) in the local CP-stack
            Executes WAM's TRY instruction
Else
            Executes WAM's TRY instruction
```

When a processor is in the Recomputation mode, it executes the TRY instruction as follows. Assume a processor p is following another processor q and the depth of p's HP-stack is i.

```
Lock
If (q in the Recomputation mode And q's HP-stack depth ≤ i) Or p's Stop flag is "On"
Then
    Reset the respective bit in the q's Recomputation vector
    Reset p's Stop flag (if it is On)
    Loop
        While the SFA field of the top element of the HP-stack = NOSF DO pop
        If the SF refers to a processor in the Computation mode
        Then
            Insert itself in the respective Recomputation vector
            Unlock
            Update the B# field according to the selected branch
            Reset the state from CP
            process the selected branch until the next TRY instruction
            BreakLoop
        Else
            Reset its respective bit and decrement RC field of the SF
            Deallocate the SF if RC = 0
            Unlock
            Remove the CP
            lock
 Else
    If q's HP-stack depth  = i
    Then Unlock, try again later on
    Else
        If SFA field of the i+1th element of q's HP-stack = Stuck
        Then
            Reset the respective bit and decrement RC field in all accessible SFs
            Remove itself from the Recomputation vector
            Unlock
        Else
            If SFA field of the i+1th element of q's HP-stack = NOSF
            Then
                Push the i+1th element of q's HP-stack in the p's HP-stack
                Unlock
                process the branch without creating a local CP
```

```
    Else
        Add itself and inc. RC in the SF referred by the i+1th element of q's HP-stack
        If there is untried branches
        Then
            Update NUB field of the SF
            Push i+1th element of q's HP-s. with the selected branch in p's HP-s.
            Switch to Computation mode
            Remove itself from q's Recomputation vector
            Unlock
            Create a local CP
            Process the selected branch
        Else
            Push the i+1th element of q's HP-stack in the p's HP-stack
            Unlock
            Create a local CP
            Process the pushed branch
```

FAIL Instruction

A processor in Computation mode processes the FAIL instruction as follows.

```
If processor in the sequential execution mode
Then
    If Mark entry in the CP stack is not reached on sequential WAM's FAIL
    Then Process the next branch
    Else
        Pop Mark entry from the CP stack
        Switch to parallel execution mode
        Lock
        While the SFA field of the top element of the HP-stack = NOSF DO pop
        If there is untried branches of the SF referred by the current top element of the HP-s
        Then
            Case NextUntriedBranch Of
            RETRY:
                Update NUB field of the SF
                Update the B# field of the top element of the HP-stack
                Unlock
                Reset the state from the CP
                Process the branch
            TRUST:
                Case RC of the SF Of
                1 :
                    Update the top of HP-stack to (Branch#, NOSF)
                    Deallocate the SF
                    Unlock
                    Reset the state from the CP
                    Deallocate the current CP frame
                    Process the branch
                > 1:
                    Update NUB field of the SF
                    Update the B# field of the top element of the HP-stack
                    Unlock
                    Reset the state from the CP
                    Process the branch
        Else
            Put itself in the Recomputation mode
            If All processors in the Recomputation mode Then Exit
            If The SF refers to processors following it
            Then Set Stop flags of these processors
            If the SF refers to a processor in the Computation mode
            Then
                Insert itself in the respective Recomputation vector
                Unlock
```

Update the B# field of the top element of the HP-stack
Reset the state from the CP
Process the selected branch until the next TRY instruction

Else

Reset its respective bit, decrement RC of the SF <u>and deallocate the SF if RC=0</u>
Unlock
Loop
Backtracks with popping the HP-stack until finding an element of the HP-stack referring to a SF
Lock
If The SF refers to processors following it
Then Set Stop flags of these processors
If the SF refers to a processor in the Computation mode
Then
Insert itself in the respective Recomputation vector
Unlock
Update the B# field of the top element of the HP-stack
Reset the state from the CP
Process the selected branch until the next TRY instruction
BreakLoop
Else
Reset its respective bit and decrement RC of the SF <u>and deallocate the SF if RC=0</u>
Unlock

13. Locking Strategies

So far we have used a very simple locking strategy in which only one processor at a time manipulates the global control information. In this section we present two locking strategies that allow parallel manipulation of the global control information.

The global control information can be divided into two sections: *shared* and *unshared* sections. The shared section is the part of the global control information shared by more than one processor. The unshared section is the part of the global control information accessible by only one processor. Figure 8 shows the part of a search tree which is associated with shared control information.

The **first locking strategy** is as follows. Whenever no processor is manipulating control information of the shared section, processors can manipulate control information of the unshared section in parallel. When a processor is manipulating a part of global control information of the shared section, no any other processor is allowed to manipulate any part of the global control information.

In the **second locking strategy**, many processors can manipulate control information in the unshared section in parallel with one processor at a time can manipulate control information in the shared section.

Implementation of the first locking strategy

The first strategy can be implemented by a global lock and a global counter. The counter counts the number of processors simultaneously manipulating the unshared section. Only when the counter reading is zero, it is possible for only one processor at a time to do global lock. Initially, the global counter is zero and the global lock is unlocked. The definition of TRY and FAIL instructions are modified as follows.

Assume the following:

- Allocating or deallocating of a SF is an atomic operation.

- ***LocalLock*** operation increments the global counter by one and returns only when the global lock is unlocked.

- ***LocalUnlock*** operation decrements the global counter by one.

- ***Lock*** operation locks the global lock only when it is unlocked and the global counter is zero.

- ***Unlock*** operation unlocks the global lock.

TRY Instruction:
When a processor is in the Computation mode, it executes the TRY instruction as follows.

> Create a SF of reference s
> set RC, NUB and the respective bit of the current processor on the SF
> *LocalLock*
> Push (1,s) in the HP-stack
> *LocalUnlock*
> Set up a local CP
> Process the first branch

When a processor is in the Recomputation mode, it executes the TRY instruction exactly the same as defined before with the new definitions of *Lock* and *Unlock* operations.

FAIL Instruction
Modifications to the above definition of the FAIL instruction (Computation mode) are as follows.

- It first requests a local lock.

- It backtracks to the latest SF.

- If the SF is accessible by the current processor only, the processor manipulates the SF.

- If the SF is accessible by more than one processor, locking of the global lock is requested.

- The rest is exactly the same as defined before.

```
LocalLock
While the SFA field of the top element of the HP-stack = NOSF DO pop
If RC of the SF = 1
Then
        Case NextUntriedBranch Of
        RETRY:
                Update NUB field of the SF
                Update the B# field of the top element of the HP-stack
                LocalUnlock
                Reset the state from the CP
                Process the branch
        TRUST:
                Update the top of HP-stack to (Branch#, NOSF)
                LocalUnlock
                Deallocate the SF
                Reset the state from the CP
                Deallocate the current CP frame
                Process the branch
Else
        LocalUnlock
        Lock
        If there is untried branches of the SF referred by the current top element of the HP-stack
        Then
                Update NUB field of the SF
                Update the B# field of the top element of the HP-stack
                Unlock
                Reset the state from the CP
                Process the branch
        Else
                Put itself in the Recomputation mode
                If All processors in the Recomputation mode Then Exit
                If The SF refers to processors following it
                Then    set Stop flags of these processors
                If the SF refers to a processor in the Computation mode
                Then
                        Insert itself in the respective Recomputation vector
                        Unlock
                        Update the B# field of the top element of the HP-stack
                        Reset the state from the CP
                        Process the selected branch until the next TRY instruction
                Else
                        Reset its respective bit, decrement RC of the SF,and deallocate the SF if RC=0
                        Unlock
                        Loop
                                Backtracks with popping the HP-stack until finding an element of the HP-stack
                                referring to a SF
                                Lock
                                If The SF refers to processors following it
                                Then set Stop flags of these processors
                                If the SF refers to a processor in the Computation mode
                                Then
                                        Insert itself in the respective Recomputation vector
                                        Unlock
                                        Update the B# field of the top element of the HP-stack
                                        Reset the state from the CP
                                        Process the selected branch until the next TRY instruction
                                        BreakLoop
                                Else
                                        Reset its respective bit, decrement RC of the SF and deallocate
                                        the SF if RC = 0
                                        Unlock
```

Implementation of the second locking strategy
We implement the second strategy by using one global lock and n local locks, one for each processor. Each HP-stack is associated with an additional pointer, called Last Shared Entry (LSE), that points to the latest entry in the HP-stack containing a reference to the latest SF shared by other processors. Entries in the bottom of the HP-stack to the entry pointed by LSE refer to SFs shared by more than one processor (see Figure 9). The other entries in the HP-stack refer to SFs accessible by only one processor. When LSE and Current Branch pointers of a processor p refer to the top entry of p's HP-stack, all entries in the stack refer to shared SFs. The difference between the LSE and Current Branch pointers of a HP-stack, called Displacement, indicates the number of unshared SFs in that stack.

The definition of the lock operations can be defined as follows.

- *Glock:* locks the global lock.

- *Gunlock:* unlocks the global lock.

- *Lock(p):* locks p's lock
  If Displacement > 0
  Then return(TRUE)
  Else return(FALSE)

- Unlock(p): unlocks p's lock.

As a rule, before manipulating the two pointers of a HP-stack, the respective local lock has to be locked first. The definition of TRY and FAIL instructions are modified as follows.

TRY Instruction:
When a processor is in the Computation mode, it executes the TRY instruction as follows. Once the current processor has succeeded to lock its lock, it can push a new entry of its HP-stack safely.

Create a SF of reference s
set RC, NUB and the respective bit of the current processor on the SF
Lock(me)
Push (1,s) in the HP-stack
Unlock(me)
Set up a local CP
Process the first branch

When a processor is in the Recomputation mode, it executes the TRY instruction according to the following actions. Assume a processor p is following another processor q and the depth of p's HP-stack is i.

- P locks the global lock and q's lock first.

- P stops itself when it reaches a point the processor q has backtracked from or if p's Stop flag is "On". Then p unlocks q's lock and looks for a processor in the computation mode to follow.

- If p is following q and p reaches the next CP before q, p has to wait until q reaches that point. In this case, p releases both global and q's locks and requests them later on.

- If the next CP is processed by q only and q is processing the last branch of the CP, p processes that branch of the CP without creating a local CP frame. The global and q's locks are unlocked.

- If the next CP has untried branches, p takes one of them and creates a local CP frame. P advances q's LSE pointer to point to the HP-stack entry that corresponds to the CP, if q's LSE pointer is not already pointing to that entry. The global and q's locks are unlocked.

- If the next CP has no untried branches and there are more than one processor processing different branches of that CP, p creates a local CP frame and follows q. The global and q's locks are unlocked.

The above actions can be specified as follows.

```
Glock
Lock(q)
If (q in the Recomputation mode And q's HP-stack depth ≤ i) Or p's Stop flag is "On"
Then
        Unlock(q)
        Reset the respective bit in the q's Recomputation vector
        Reset p's Stop flag (if it is On)
        Loop
              While the SFA field of the top element of the HP-stack = NOSF DO pop
              If the SF refers to a processor in the Computation mode
              Then
                    Insert itself in the respective Recomputation vector
                    Gunlock
                    Update the B# field according to the selected branch
                    Reset the state from CP
                    process the selected branch until the next TRY instruction
                    BreakLoop
              Else
                    Reset its respective bit and decrement RC field of the SF
                    Deallocate the SF if RC = 0
                    Gunlock
                    Remove the CP
                    Glock
Else
        If q's HP-stack depth  = i
        Then Unlock(q), Gunlock, try again later on
        Else
              If SFA field of the i+1th element of q's HP-stack = NOSF
              Then
                    Push the i+1th element of q's HP-stack in the p's HP-stack
                    Unlock(q)
                    Gunlock
                    process the branch without creating a local CP
              Else
                    Add itself and inc. RC in the SF referred by the i+1th element of q's HP-stack
                    If there is untried branches
                    Then
                          Update NUB field of the SF
                          Push the i+1th element of q's HP-s. with the selected branch in p's HP-stack
                          Advance q's LSE pointer, if p and q are only in the SF (i.e., if RC of the SF = 2)
                          Unlock(q)
                          Switch to Computation mode
                          Remove itself from q's Recomputation vector
                          Unlock
                          Create a local CP
                          Process the selected branch
                    Else
                          Push the i+1th element of q's HP-stack in the p's HP-stack
                          Unlock(q)
                          Gunlock
                          Create a local CP
                          Process the pushed branch
```

FAIL Instruction

A processor p in Computation mode processes the FAIL performs the following actions.

- P locks its local lock.

- If p has unshared SFs, it backtracks to either the latest unshared SF which has untried branches or the latest shared SF.

- If the SF refers to untried branches, it takes one of them. If that SF is unshared and there is no more untried branches, the local CP frame and SF are deallocated.

- If the SF does not refer to untried branches, but refers to processors in the Computation mode, one of these processors is followed.

- If no untried branch is found and no processor in the Computation mode is followed, one SF after the other is investigated until a SF having either untried branches or processors in the Computation mode is found.

The above actions can be specified as follows.

```
Loop1
        If Lock(me)
        Then
                If the SFA field of the top element of the HP-stack = NOSF
                Then
                        Pop
                        Unlock(me)
                        ContinueLoop1
                Else
                        Case NextUntriedBranch Of
                        RETRY:
                                Update NUB field of the SF
                                Update the B# field of the top element of the HP-stack
                                Unlock(me)
                                Reset the state from the CP
                                Process the branch
                                Return
                        TRUST:
                                Update the top of HP-stack to (Branch#, NOSF)
                                Unlock(me)
                                Deallocate the SF
                                Reset the state from the CP
                                Deallocate the current CP frame
                                Process the branch
                                Return
        Else
                Unlock(me)
                Glock
                If there is untried branches of the SF referred by the top element of the HP-stack
                Then
                        Update NUB field of the SF
                        Update the B# field of the top element of the HP-stack
                        Gunlock
                        Reset the state from the CP
                        Process the branch
                        Return
                Else
                        Put itself in the Recomputation mode
                        If All processors in the Recomputation mode Then Exit
```

If The SF refers to processors following it
Then set Stop flags of these processors
If the SF refers to a processor in the Computation mode
Then
    Insert itself in the respective Recomputation vector
    Gunlock
    Update the B# field of the top element of the HP-stack
    Reset the state from the CP
    Process the selected branch until the next TRY instruction
    Return
Else
    Reset its respective bit and decrement RC of the SF
    Pop
    Gunlock
    Loop2
        Glock
        Backtracks with popping the HP-stack until finding an element of the HP-stack referring to a SF
        If The SF refers to processors following it
        Then set Stop flags of these processors
        If the SF refers to a processor in the Computation mode
        Then
            Insert itself in the respective Recomputation vector
            Gunlock
            Update the B# field of the top element of the HP-stack
            Reset the state from the CP
            Process the selected branch until the next TRY inst.
            Return
        Else
            Reset its respective bit and decrement RC of the SF
            Gunlock
            ContinueLoop2

NB: The Pop operation updates the LSE and Current Branch pointers of the respective HP-stack, if both pointers point to the top element.

14. Hardware Architectures

This section gives suitable architectures that support the model. Three architectures are shown in Figure 10. Each processing element has its local memory (LM). The global memory (GM) is physically distributed.

In Figure 10 (a), each processor has a part of global memory (GM). Accessing a local GM is performed directly while accessing remote GM is through the common bus. In Figure 10 (b), cache memory (CM) is used as in Sequent for caching global information. In Figure 10 (c), GM is exactly the same as in (a) but a communication processor (CP) in each node is used for communicating and processing nonlocal requests. Requests to the local part of global memory can be performed directly by the corresponding processor (p).

15. A Method for Nonshared Memory Multiprocessor

In this section we outline a method with the same principle presented in this report for nonshared memory multiprocessor. In the method outlined in this section, we combine the history path idea from this report with multisequential machine idea [Ali86].

Assume a system with a number of processing elements (PE) connected by some medium that provides communications from every PE to all other PEs. Every PE has a local memory for holding a copy of program code and environment. Every PE is exactly the same as WAM with an additional HP-stack.

The method is outlined as follows.

1. All PEs in the system start simultaneously processing the top-level query.
2. Every PE independently selects a part of the search tree for processing exactly the same as in [Ali86].
3. Every PE maintains its current path from the top-level query to the current branch in its History-Path stack.
4. When a PE p has completed processing its part of the tree and there is another PE q with unprocessed job, q sends a path to p specifying an unprocessed branch. The PE p compares the received path from q with its History-Path stack. In general, there is a common part starting at the root. The PE p backtracks to the common part, then it processes the received uncommon part.

We notice that, PEs communicate only information specifying a path and not a computation state. Communications occur only when there are idle PEs and busy PEs with unprocessed job. The required communication capacity is not high. The performance of every PE is almost as the WAM.

Improvement

Since recomputation time is long, a PE q that gave a branch to another processor p may finishes its job before p starting processing that branch. In this situation, it is more effective to allow the PE that reaches the unprocessed branch first to process it. That is, we need a mechanism that makes a PE that gave a job to change its mind when it wishes.

Our solution is as follows. Assume that every PE is associated with a bit-vector of size N, where N is the number of processors in the system. That is, there are N bit-vectors in the system; one vector in each PE. Initially, all bits are OFF. When a PE q gives another PE p a branch i at a choice point c, q turns ON the p's bit in its local bit-vector indicating that one of my branches is sent to p. The branch i at choice point c is marked as it has sent to p. Now either p or q can process that branch. If q has finished its job and backtracked to c before p, q checks p's bit. If it is ON, q turns it OFF (and possibly signal p) and processes the branch.

If p reaches that branch before q, p sends first a CHECK message to q to check whether that branch is processed or not. When q receives that message from p, it just checks p's bit in the local vector. If it is ON, q turns it OFF and responses with positive acknowledgement, otherwise with negative acknowledgement. When p receives a positive acknowledgement only, it will process the branch.

This solution requires one bit-vector of size N at every PE and extra information at choice points that have some of their branches at remote PEs. We could keep a list at each choice point containing name of remote PEs and the respective branches. Since every PE may give at most one path to each other PE at a time, the maximum number of entries in all lists on one PE is N - 1.

If we use one-word vector of size N instead of one bit-vector at every PE, we could use a pointer to a local choice point when a branch of that choice point is at a remote PE. If a pointer is not NULL, the respective branch is unprocessed. The pointer allows every PE to directly deallocate a list entry when the respective branch is processed.

16. Conclusion and Discussion

We have presented a method for OR-parallel execution of Horn clause programs on a shared memory multiprocessor system. The shared memory contains only control information that guide processors requesting a job to independently construct the required environment to get a new job without degrading the performance. The method utilises the technology devised for the sequential implementation of Prolog. The only overhead in comparison with the sequential WAM is the maintenance of the global control information: some extra overhead on TRY, RETRY, TRUST, and FAIL instructions only.

A general model for nonshared memory architectures, for a very large system, based on the same principle is outlined. The performance of each processor is almost the same as the sequential WAM.

If we compare our method that shares some control information with the complete global memory model [Over86], we expect that maintenance of binding environment (environment stack, Trail stack, and Heap) is much simpler in our method. The cost of getting a new branch in both methods is probably the same. This is because, in our method choice point stacks reside in the local memories while in the other method reside in the global memory. Our method requires one complete physical copy of the binding environment in each processor's memory, while the other method uses a multiple logical copies of the environment in the global memory. The granularity of atomic operations on shared information may be larger in our method. Some possible optimization can be done in our method to reduce granularity of these operations by using better representation of the shared data structures and faster algorithms.

It is planned to evaluate the two methods on the Sequent Balance 8000 by Milton Wong at the Royal Institute of Technology, Stockholm. Investigation of the treatment of cut and the extra-logical predicates of Prolog is an interesting future work. A general locking mechanism that allows parallel manipulation of the shared control information is under investigation.

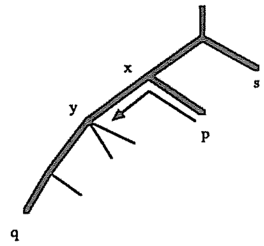

Figure 1: The Basic Idea

- p has finished its branch
- p backtracks to x and follows q
- p gets a branch from q at choice point y

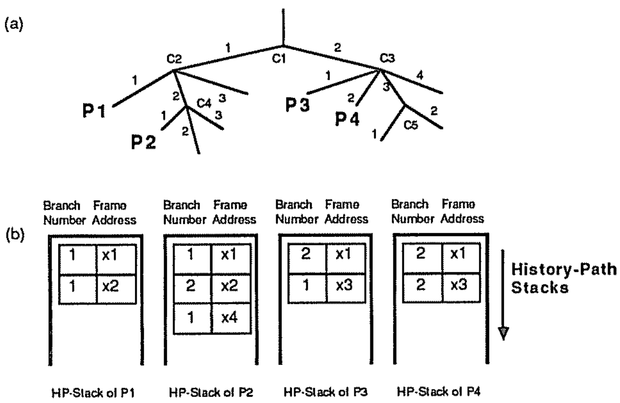

Figure 2 (a) Numbering of branches (b) History-Path stacks

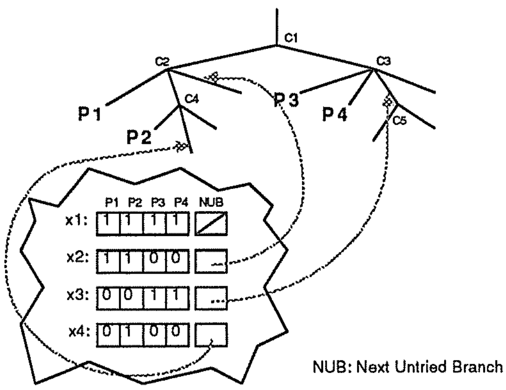

Figure 3: Split frames associated with the active choice points

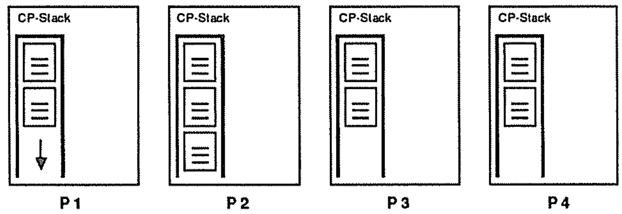

Figure 4: Local choice point stacks

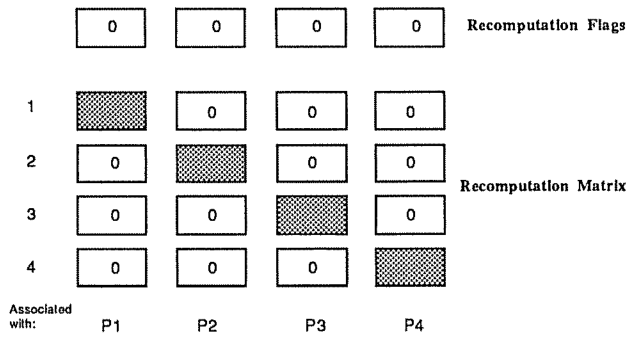

Figure 5: Recomputation flags and Recomputation matrix for the example shown in Figure 2.

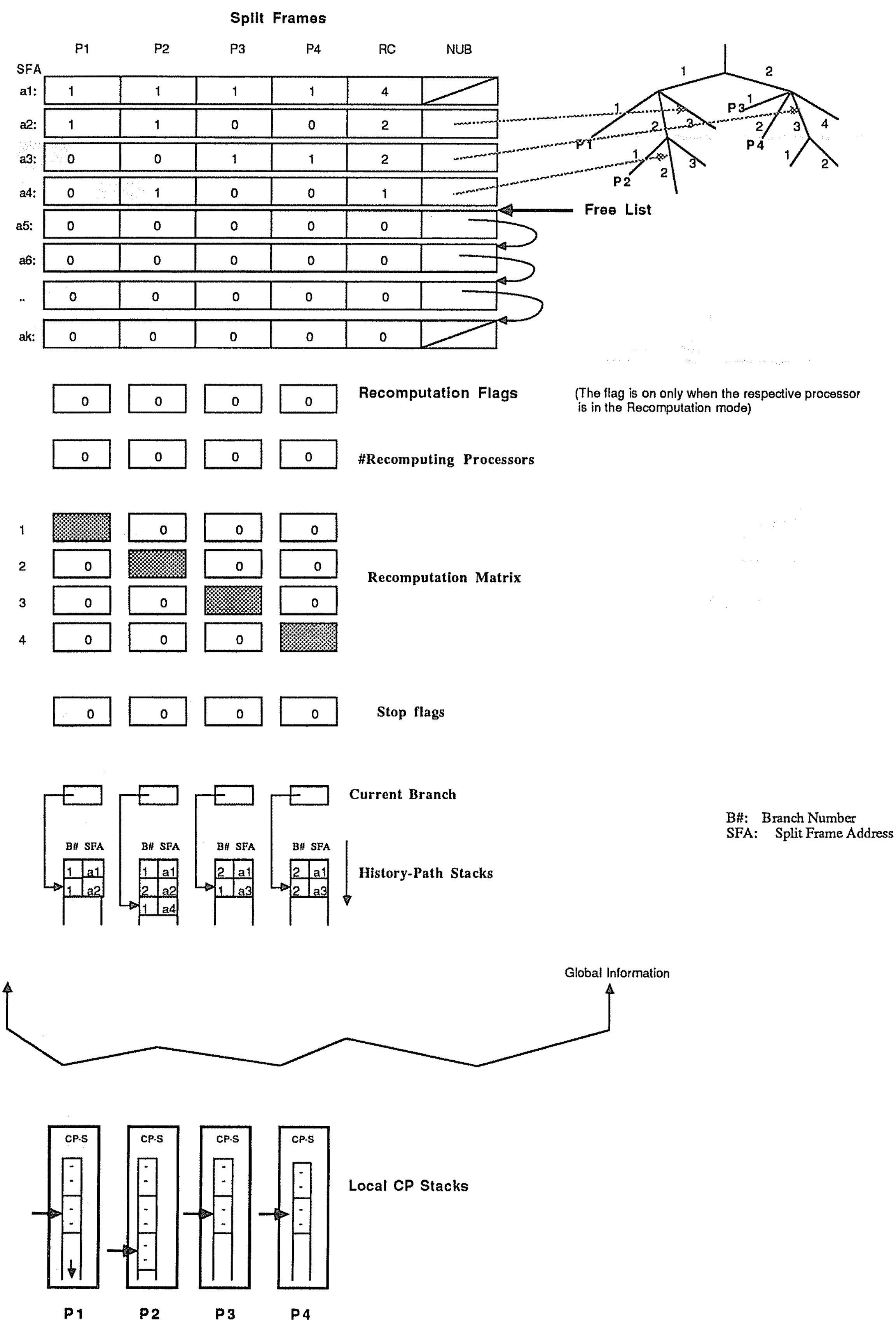

Figure 6: A snapshot of the global state of the example shown in Figure 2.