A UTILIZAÇÃO DO TRANSPUTER® [1] T800 NO SISTEMA P3

L. A. G. Rolim, P. R. F. A. Luz
CPqD - TELEBRÁS
Caixa Postal 1179, 13085 - Campinas - SP

RESUMO

Utilização do transputer T800 como coprocessador da placa U-32 do Processador Paralelo P3, onde é responsável pela comunicação entre módulos e operações com ponto flutuante, focalizando-se o problema da sincronização dos dois processadores. Uma abordagem "software" do problema apresenta as estruturas de dados bem como os algoritmos na U-32 e no T800, respectivamente em CHILL [2] e OCCAM, utilizados pelo Sistema Operacional PP-SO/P[3].

ABSTRACT

Utilization of the T800 as U-32 coprocessor of the Processador Paralelo P3, where it is responsible for the intermodule communication and floating-point operations focusing the two processors's synchronization problem. A software approach of the problem presents the data structures and algorithms at U-32 and T800, respectively in CHILL and OCCAM, used by the PP-SO/P Operating System.

1. INTRODUÇÃO

O Processador Paralelo P3 [2] é um sistema para processamento paralelo baseado na placa U-32 e no sistema operacional PP-SO/P, desenvolvidos no CPqD - Telebrás. A placa U-32 compõe-se basicamente de :

- microprocessador iAPX 386 - 25 MHz (5 MIPS).
- coprocessadores numéricos 80387 (0.85MFlops) e WTL1167 (1.1 MFlops).
- módulo de comunicação, implementado como placa aérea, baseado no transputer T800 (15 MIPS)

O sistema P3 é composto , em sua configuração típica, por 16 módulos de processamento e um módulo de gerência, sendo que cada módulo contém até 8 placas U-32. O T800 é utilizado para implementar a comunicação entre placas U-32 de diferentes módulos.

O ambiente de concorrência entre várias tarefas, executando na U-32 [1] e solicitando operações ao T800, torna necessário um mecanismo que garanta a sequencialização dos pedidos como forma de sincronização dos dois processadores.

A solução para o sincronismo entre o iAPX 386 e o T800, proposta neste artigo, é genérica; embora analisado o problema particular do Sistema P3.

2. O TRANSPUTER T800

O T800 [6] combina capacidade de processamento, memória e intercomunicação em um simples circuito integrado VLSI. É um microprocessador RISC de 32 bits, de alta capacidade (2.5 MFLOPS) e típicos 15 MIPS com uma performance de 6000K Whetstones/seg na versão 30Mhz; características úteis para sistemas de alta capacidade de processamento. Outras características importantes desse microprocessador são as facilidades relativas a:

- controle, a nível do hardware, da execução de tarefas, permitindo eficiente gerenciamento da execução de processos concorrentes.
- temporização implementada internamente, propiciando o controle de tempo sem a necessidade de temporizadores externos.
- canais de comunicação serial, bidirecional, de alta velocidade que possibilitam a transferência de mensagens a uma taxa de até 20 Mbits/seg.
- memória interna de 4Kbytes que possibilita a carga e execução, eficiente, de "software" de funções a serem executadas paralelamente com a U-32.

[1] Transputer, OCCAM - marcas registradas do grupo INMOS
[2] CHILL [3]: CCITT High Level Language
[3] Sistema Operacional desenvolvido no CPqD - TELEBRÁS para aplicações de Processamento em Tempo Real e em Sistemas Distribuídos.

O T800 utilizado como coprocessador da U-32, devido às suas características, libera-a das transferências de mensagens entre processadores assim como da execução de complexas operações de ponto flutuante, executando mais rapidamente e em paralelo com ela, permitindo ainda concorrência na execução das operações disponíveis.

3. COMUNICAÇÃO ENTRE U-32 E O T800

A utilização dos recursos oferecidos pelo T800 é feita através de primitivas do Sistema Operacional, que garantem a sequencialização das operações solicitadas ao T800 através de dados colocados em endereço de memória previamente determinado seguido da geração de uma interrupção.

3.1. U-32 → T800

A U-32 pode solicitar ao T800, dois tipos básicos de operação:

- Intercomunicação - transmissão/recepção de dados através de cada um dos quatro links de comunicação do T800.
- operações em ponto flutuante.

Neste item se descreve, na linguagem CHILL, as estruturas de dados e os algoritmos necessários para a sequencialização, na U-32, das solicitações de tarefas para o T800. A sequencialização de operações visa garantir que o T800, nao obstante possa executar concorrentemente tranferências de dados através dos links seriais e operações em ponto flutuante, despacha um tipo de operação por vez.

3.1.1. Mecanismo de acesso

A comunicação entre a U-32 e o T800 é realizada através de memória partilhada e interrupções em ambos os sentidos. No lado do T800 é acionada a linha *EVENTREQ* todas as vezes que a U-32 solicita algum serviço do mesmo. O acionamento desta linha é feito através de um ciclo de escrita em uma porta de E/S.

O software inicialmente carregado pela U-32 no T800, provê um processo de alta prioridade que trata a interrupção associada à linha *EVENTREQ* . A função desse processo é de despachar, o mais rápidamente possível, o pedido da U-32 para o processo específico de modo a tornar desnecessário a espera da U-32 pelo reconhecimento do acionamento da *EVENTREQ* ; o que é factível se for levado em conta a diferença de velocidade entre os processadores.

Os dados relativos à operação solicitada, através da linha *EVENTREQ* , são obtidos em um endereço fixo conhecido por ambos os processadores e dispostos de acordo com a estrutura `param_struct` mostrada na *Figura. 1* . É importante salientar que o campo `ptr_dad` não corresponde ao par seletor/offset do iAPX 386 e sim ao enderço físico correspondente pois o T800 utiliza palavras de 32 bits para endereçar os dados referentes à operação solicitada pela U-32.

```
NEWMODE
  operation_code =
    SET (txlink0, rxlink0, txlink1, rxlink1,
         txlink2, rxlink2, txlink3, rxlink3,
         float ),

  type_operation = SET (nowait, wait, complete),

  pbuffer = /* ptr+tam dados a TX / RX */
     ROW CHAR (32767),

  param_struct =
     STRUCT(operation operation_code,
        CASE operation OF
          (tklink0, rxlink0,
           txlink1, rxlink1,
           txlink2, rxlink2,
           txlink3, rxlink3): ptr_dad pbuffer,
          (float):  type type_operation,
                    code INT,
                    ptr_par REF float_parms
        ESAC) ;

   DCL T800_param param_struct ;
```

Figura 1: *Controle para sequencialização de tarefas*

O Sistema operacional da placa U-32 deve assegurar que o acionamento da linha *EVENTREQ* e as alterações na estrutura `param_struct` sejam realizadas de forma congruente. Esse mecanismo é apresentado mais adiante.

A interrupção no sentido T800 → U-32 é realizada de forma semelhante. O T800 realiza um ciclo de escrita em um determinado endereço de memória e com isso gera uma interrupção para a U-32. Esse procedimento é centralizado, no T800, em um único processo pois neste caso precisa-se atualizar uma estrutura de dados que armazena as várias comunicações de interrupção enviadas para à U-32. Essa estrutura, deste ponto em diante referenciada como `Result_struct` - *Figura. 4* , é também alocada em um endereço conhecido por ambos os processadores. Através das informações contidas em `Result_struct` o processo tratador de interrupção na U-32 poderá reativar os processos que estejam aguardando o término da operação solicitada ao T800.

3.1.2. Sequencialização de pedidos

A sequencialização de pedidos ao T800 é realizada no S.O. através de Algoritmos e Estruturas de dados, a seguir descritos e deste ponto em diante referenciado como CS-T800 (Controlador Software), que garantem a utilização, de forma congruente, do mecanismo de interrupção e interface anteriormente apresentados.

Estrutura de dados

O CS-T800 deve sequencializar os pedidos ao T800 para cada uma das possíveis operações. A estrutura básica para a sequencialização de cada operação consiste de uma fila simplesmente ligada, `link-list` *Figura. 2* , onde são colocados os processos que esperam atendimento e a identificação do processo, através de seu TSS [1] , que eventualmente esteja usando os serviços do T800.

[1] Task State Segment - identificação, unívoca, de um processo (task) no iAPX 386 (modo protegido) utilizado pelo **PP-SO/P**

```
NEWMODE
  TSS = INT,
  link_list = STRUCT
   ( id_current_process,
     begin_link,  /* link p/ inicio da fila */
     end_link TSS /* link p/ fim da fila */ ) ;

  DCL float_operation type_operation,
     control_list
       ARRAY(operation_code) link_list ;
```

Figura 2: *Estrutura de dados do CS-T800*

Algoritmos

Dois algoritmos são descritos neste item. O primeiro deles refere-se ao despacho de pedidos para o T800 e a sequencialização dos mesmos para cada uma das operações que podem ser executadas paralelamente. O segundo algoritmo descreve o processo de tratamento da interrupção que o T800 gera no momento em que a operação for completada.
As operações em float-point podem ser solicitadas de três maneiras distintas, de acordo com o campo `type` da estrutura `param_struct` (*Figura. 1*):

- `nowait` - indica que o solicitante (processo) não deseja esperar o término da mesma.
- `wait` - indica que o solicitante deve esperar até o término de uma operação em execução.
- `complete` - indica que o processo solicitante deve ser suspenso até o término da mesma.

Despacho e sequencialização

O despacho dos pedidos para o T800 é feito através de uma primitiva do S.O. Sua função é basicamente verificar a congruência dos dados de entrada e em seguida despachar o pedido para o T800 ou suspender o processo requisitante se a tarefa não puder ser despachada no momento da chamada da primitiva. O algoritmo básico da primitiva de despacho de operações, para o T800, é apresentado na *Figura. 3* .

```
Dispatcher_trans:

   PROC (parameters param_struct)
       EXCEPTIONS (PRIMITIVEFAIL) ;

   /* Verifica se a operacao pode ser despachada */

   DO WITH control_list(parameters.operation)
     IF id_current_process = 0 THEN
        T800_param := parameters ;
        generate_EVENTREQ () ;
        id_current_process = current_TSS () ;
        IF parameters.operation = float THEN
           float_operation := parameters.type ;
           CASE parameters.type OF
             (nowait):
                 RETURN ;
             (wait):
                 IF id_current_process =
    current_TSS () THEN
                    float_operation := complete ;
              ELSE
              ESAC ;
           FI ;
      ELSE
        link_list (parameters.operation, current_tss) ;
      FI ;
   OD;
   deschedule () ;

END Dispatcher_trans ;
```

Figura 3: *Despachador de opera cões para o T800*

Tratamento de interrupção

Ao final de cada operação executada pelo T800, é gerada, através de um ciclo de escrita em um registro de E/S, uma interrupção para U-32. Os dados relativos à interrupção são previamente colocados em área de memória reservada para comunicação e cujo formato é o da estrutura `operation_status` mostrado na *Figura. 4* .

A variável `operation_status` é uma dupla palavra que deve ser sempre acessada com o barramento em LOCK. No lado da U-32 a sequência de instruções para a leitura da variável `operation_status` é mostrado no trecho de código, em linguagem de montagem, da *Figura. 5* .

```
NEWMODE op_result = /* Erro em operacao */
    ARRAY(0:2) BOOL PACK,
  Result_struct   =
    ARRAY (operation_code) op_result PACK;

  DCL operation_status Result_struct;
```

Figura 4: *Estrutura de dados relativa aos resultados.*

```
        XOR     EAX,EAX
        XCHG    EAX,operation_status
```

Figura 5: *Acesso, em "lock", à variável de controle.*

A instrução XCHG é executada com o barramento em LOCK, o que garante o acesso exclusivo à variável `operation_status` para obter as informações passadas pelo T800. Cada 3 bits de `operation_status` corresponde a um campo que é o STATUS de uma das possíveis operações executadas pelo T800. O bit mais significativo deste campo indica o final da operação enquanto que os menos significativos indicam se houve erro na operação efetuada (zero se houve sucesso).
Após obter essa variável, o processo tratador de interrupção do T800 verifica que operações finalizaram, reativa os respectivos processos (se for o caso) que aguardavam pelo seu término, solicitando nova(s) operação (ões) caso exista(m) outra(s) pendente(s). Esse processo deve tratar o caso em que `operation_status` = 0, ou seja : o T800 gera uma interrupção indicando a finalização de uma operação que já foi comunicada à U-32 numa interrupção anterior. O algoritmo básico do processo de tratamento de interrupção é apresentado na *Figura. 6* .

3.2. T800 → U-32

Como descrito anteriormente, a sequencialização de tarefas para o T800 ocorre naturalmente com a execução de primitivas que garantem o acesso indivisível à area de memória, onde o T800 vai buscar os dados após ter sido interrompido pela U-32. A interrupção causa

```
Interrupt_T800:

  PROC () ;

  DCL current_status LONG_INT ;

      current_status := get_status () ;
      IF current_status /= 0 THEN
         DO FOR i := txlink0 TO float ;
            IF finish (i) THEN /* Reativa */
               reactive (i) ;
               next_operation (i) ;
            FI ;
         OD ;
      FI ;

   END Interrupt_T800 ;
```

Figura 6: *Tratamento da interrupção gerada pelo T800.*

no T800 a ativação da linha *EVENTREQ* que causa a imediata ativação do processo responsável pelo tratamento da mesma. Esse processo nada mais faz que enviar imediatamente para o processo responsável pela operação, uma mensagem com o conteúdo da estrutura de dados referente à operação solicitada.

3.2.1. Tratamento da interrupção

O trecho de código, em OCCAM, mostrado na *Figura. 7*, ilustra em linhas gerais o processo responsável pelo tratamento da interrupção, causada pela U-32, que solicita ao T800 a execução de uma operação.

Uma vez despachada a mensagem com os dados referentes à operação solicitada para o processo tratador, o despachador fica apto a atender nova solicitação (é suspenso) e o controle do T800 é passado ao próximo processo que esteja pronto para execução. Os processos executores têm o formato ilustrado na *Figura. 8*.

Note-se que os processos que executam as diferentes operações solicitadas pela U-32, após lerem os dados relativos à operação, executam de forma concorrente e em paralelo com a U-32, concorrendo esporádicamente com esta pelo controle do barramento ao acessar os dados referentes a uma operação. Tendo em vista as diferentes operações implementadas no T800, o trecho

```
DEF  max.op = 8 --- No. de operacoes - 1
DEF  tam.info = 7 --- Tam - 1 de T800_param
  CHAN Evento:
--- Processos tratadores das tarefas.
  CHAN Processo[0:max.op]:
--- Status das operacoes
  VAR   status[0:max.op]:
  SEQ i = [0 FOR max.op]
      status[i] := 0
  PRI
     PAR
 --- parms_struct vista pelo T800
     VAR T800_param[0:tam.info]:
     WHILE TRUE
       Evento ? ANY --- Espera int da U-32
       op := T800_param[BYTE 0] --- Op. solicitada
       IF status[op] = 0  --- Processo livre ?
          status[op]:= 1 --- Indica op em execucao
          Processo[op] ! T800_param [1 FOR tam.info]
       TRUE
          Notifica.U32(op,em.andamento)
```

Figura 7: *Tratamento de interrupção EVENT_REQ.*

```
WHILE TRUE --- Do for ever
  VAR x
    Processo[n] ? x

     --- Executa a operacao

    Notifica.U32(n, cod.erro)
```

Figura 8: *Formato geral dos processos executores.*

de código em OCCAM da *Figura. 9* ilustra uma forma de implementação.

Para cada operação existe um processo associado, de prioridade inferior ao de atendimento da interrupção da U-32, concorrentemente com os demais processos executores gerenciados pelo "firmware" do T800.

3.2.2. Notificação do término de operação

Como as operações solicitadas pela U-32 ao T800 executam de forma concorrente e em paralelo com ela, torna-se necessário a sincronização do acesso à variável de controle das operações, que fica sujeita a acesso do tipo exclusivo. Devido ao T800 não possuir, à semelhança do iAPX 386, uma instrução do tipo XCHG execu-

```
PAR
   WHILE TRUE  --- Doforever env msg via link0
      SEQ
        Processo[txlink0] ? msg
          Operacao (txlink0)
          Notifica.U32(txlink0)
            .    .    .
   WHILE TRUE
     VAR info.op
        SEQ
           Processo[floatp] ? info.op
           Float(info.op)  --- Exec op solicitada
           Notifica.U32(floatp) --- Ver exp.
```

Figura 9: *Exemplo de processos executores.*

tada com o barramento em "lock", o módulo de comunicação implementa um esquema que garante, através de um registro "hardware", acesso exclusivo ao barramento por dois ciclos. O trecho de programa da *Figura. 10* ilustra como o "software" do T800, internamente à rotina `Notifica.U32` , garante a atualização do status das operações de forma congruente.

```
... garante "lock" para dois acessos
    LOAD NON-LOCAL  lock.bus
... le status com bus em " lock" (ciclo 1)
    LOAD NON-LOCAL  operation.status
... obtem o codigo de erro
    LOAD LOCAL      op.pattern
... Adiciona codigo de erro
    OR
... Escreve status em "lock" (ciclo 2)
    STORE NON-LOCAL operation.status
```

Figura 10: *Notificação, à U-32, de fim de operação.*

4. CONCLUSÕES

A utilização de transputer no Sistema P3, embora basicamente norteada pela capacidade de intercomunicação desses microprocessadores, libera a U-32 da execução de algumas funções importantes.

O módulo de comunicação baseado no T800, possui uma capacidade de processamento que torna o conjunto U-32 - T800, eficiente, flexível e consequentemente capaz de atender às futuras evoluções. A escolha do T800 fica justificada pela potencialidade do mesmo na execução de quaisquer funções que sejam necessárias, reduzindo a sobrecarga da U-32.
Neste artigo abordou-se específicamente o problema de sincronização dos processadores iAPX 386 da U-32 e o T800 do módulo de comunicação, com respeito ao coprocessamento por eles estabelecido, na tarefa de execução de software do Sistema Paralelo P3 [2].

Referências

[1] CPqD - TELEBRÁS, Especificação e características da U-32 - Projeto Processador Preferencial", Dezembro de 1987.

[2] A. Pestana e E. Cavalli, "Processador Paralelo P3", II SBAC-PP, Setembro 1988.

[3] CCITT, "CHILL Language Definition - Recommendation **Z.200**", Janeiro de 1985.

[4] **int$_e$l** Corporation,"iAPX 386 - Programmer's Reference manual".

[5] INMOS Corporation,"IMS T800 Transputer - Product Preview", 1987.

[6] M. Homewood, D. May, D. Sheperd, R. Sheperd, "The IMS T800 transputer", IEEE MICRO, pag. 10 -26, Outubro de 1987.

[7] INMOS Ltd, "IMS T414 transputer - Preliminary Data", Fevereiro de 1987.

[8] INMOS Ltd, "OCCAM Programming Manual".

[9] INMOS Ltd, "Transputer development System - IBM PC version - OCCAM IMPLEMENTATION", Setembro de 1985.