UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

Vali_Sync: Um módulo para a Instrumentação e Execução de Testes Temporais em Programas Paralelos

Mário Santos Camillo

Adenilso da Silva Simão

Simone do Rocio Senger de Souza

Nº 288

RELATÓRIOS TÉCNICOS

São Carlos - SP
Dezembro/2006

SYSNO 1562832
DATA / /
ICMC - SBAB

1 Introdução

A atividade de teste de software é crucial para garantir a qualidade do software, entretanto, o alto custo de aplicação compromete sua utilização prática. Este custo ocorre devido à falta de pessoas qualificadas, de ferramentas de apoio e de técnicas adequadas para a condução desta atividade. Esses problemas se agravam quando são considerados softwares paralelos. Muitas das questões exploradas e caracterizadas para programas seqüenciais não são suficientes para programas paralelos, visto que esses programas possuem características adicionais que precisam ser consideradas durante a atividade de validação, como: não determinismo, sincronização, concorrência e comunicação.

Ambientes de passagem de mensagens fornecem mecanismos para o desenvolvimento de programas paralelos para serem utilizados em um grupo de computadores conectados (conhecido como cluster), os quais formam uma máquina paralela virtual [5]. Ambientes de passagem de mensagens bastante difundidos são: PVM - Parallel Virtual Machine [3] e os baseados no padrão MPI - Message Passing Interface, tais como MPICH e LAM MPI [6].

A `ValiMPI` é uma ferramenta de teste de programas paralelos para o ambiente de passagem de mensagem `MPI 2.0`. A arquitetura da ferramenta `ValiMPI` foi desenvolvida para atuar no paradigma de passagem de mensagem e, portanto, auxiliar na aplicação dos critérios pertinentes, dando apoio para os critérios de fluxo de dados e controle

Dentro desse contexto o módulo `ValiSync` foi desenvolvido com o objetivo de estender as funcionalidades da ferramenta de teste `ValiMPI`[1]. A nova funcionalidade implementada pelo módulo é a de testes temporais dos programas paralelos, ou seja, o `ValiSync` testa os programas paralelos para erros que podem ser causados pela ordem em que as sincronizações entre `sends` e `receives` ocorrem.

2 Integração do Módulo `ValiSync`

A organização da ferramenta `ValiMPI` e a integração do módulo `ValiSync` podem ser vistos na Figura 1.

Os módulos da ferramenta `ValiMPI` foram desenvolvidos de forma a possuir um baixo acoplamento, sendo que a integração entre eles é realizada por meio da troca de dados. Em geral, esses dados são lidos e/ou escritos em arquivos de propósito específico. Como pode ser observado, o módulo está integrado à ferramenta, necessitando da saída dos módulos `Vali_Exec` (caminhos executados e saídas dos teste) e `Vali_Inst` (Informação sobre fluxo de dados e GFCs) para obter as sincronizações de cada caso de teste e também do módo de reexecução do módulo `Vali_Exec`.

Na seção seguinte o funcionamento do módulo e sua execução são descritos de forma mais detalhada.

3 Descrição do Módulo

O módulo `ValiSync` foi criado tendo como função obter as sincronizações interprocessos entre as chamadas de `send` e `receive` do MPI e realizar a reexecução de cada uma dessas sincronizações de forma a testar o programa paralelo para erros temporais.

Para isso foram implementados um programa em C (`gera_sync`) e um `script` em `Bash` (`vali_sync`). O `gera_sync` cria uma lista de nós que executam as primitivas de

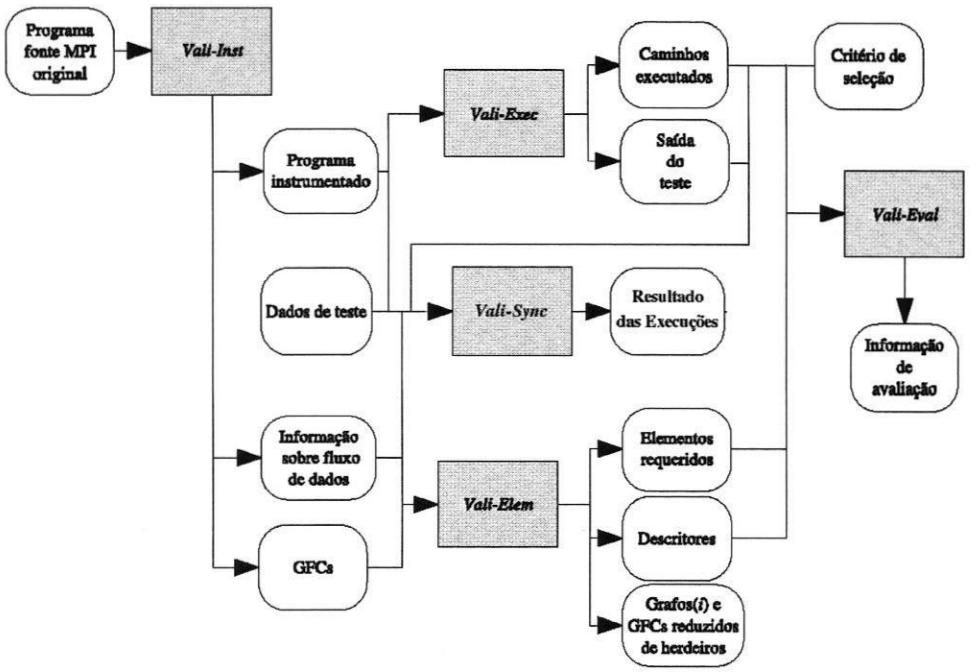

Figura 1: Arquitetura da ferramenta ValiMPI com a integração do novo módulo.

comunicação (ou seja, chamadas `send` ou `receive`) para cada uma das funções presentes no programa paralelo sendo testado, utilizando-se dos arquivos de grafo de fluxo de controle (arquivos `C.<nome da função>.dot`) criados pela ferramenta `ValiMPI` [1]. Um exemplo desse arquivo pode ser visto na Figura 2. O arquivo exemplificado é o arquivo `C.main.dot` do programa `GCD` utilizado durante os testes do módulo (a porção correspondente do arquivo gcd-mpi.c pode ser vista na Figura 5). Nele podemos ver a sintaxe dos arquivos `.dot`, onde as linhas começadas por "node" descrevem um nó e as linhas seguintes entre "/* */" indicam o que acontece no nó em questão. Por exemplo, na primeira linha da figura temos a descrição do nó 11, e as duas linhas seguintes indicam que ocorre um `send` da variável `buf` com tamanho 2, do tipo `MPI_INT`, para o processo 1, com a tag 1, utilizando o comunicador `MPI_COMM_WORLD`, e também ocorre um `s-uso`[1] de `buf`.

Após a criação da lista de primitivas de comunicação, o `gera_sync` lê os arquivos de caminho de execução (arquivos `trace.<nome da função>.p<número do processo>`) de cada função de cada processo de forma a obter uma lista de nós que executam chamadas `send` ou `receive`. A Figura 3 contém parte do arquivo de caminho de execução do caso de teste 001 do programa GCD. Cada par de números separados por um hífem significa o número do nó e o número do processo que executou esse nó, o que torna fácil saber quando houve um `send` ou `receive` por parte do processo ao qual o arquivo se refere. Por exemplo, o dois primeiros pares de números, 1-0 2-0, indicam que primeiramente foi executado o nó um do processo zero e o próximo nó executado foi o nó dois do processo zero.

Com essas duas listas, pode-se então fazer a interconexão das listas de nós dos processos, conectando cada nó que execute um `receive` com os nós que executam `sends` em outros processos que possam sincronizar com ele (`sends` que tenham a mesma `tag` e que tenham como destino o processo executando o `receive`). As listas interconectadas criam

```
1  node [shape = circle] 11;
2  /* m_send of buf,2,MPI_INT,1,1,MPI_COMM_WORLD at 11 */
3  /* susage of buf at 11 */
4  node [shape = circle] 12;
5  /* definition of buf at 12 */
6  /* cusage of y at 12 */
7  node [shape = circle] 13;
8  /* definition of buf at 13 */
9  /* cusage of z at 13 */
10 node [shape = circle] 14;
11 /* m_send of buf,2,MPI_INT,2,1,MPI_COMM_WORLD at 14 */
12 /* susage of buf at 14 */
13 node [shape = circle] 15;
14 /* m_recv of buf,1,MPI_INT,MPI_ANY_SOURCE,2,
15                 MPI_COMM_WORLD,&status at 15 */
16 /* definition of buf at 15 */
17 /* derefdefinition of status at 15 */
18 node [shape = circle] 16;
19 /* definition of x at 16 */
20 /* cusage of buf at 16 */
21 node [shape = circle] 17;
22 /* m_recv of buf,1,MPI_INT,MPI_ANY_SOURCE,2,
23                 MPI_COMM_WORLD,&status at 17 */
24 /* definition of buf at 17 */
25 /* derefdefinition of status at 17 */
26 node [shape = circle] 18;
27 /* definition of y at 18 */
28 /* cusage of buf at 18 */
29 node [shape = circle] 19;
30 /* pusage of x at 19 */
31 /* pusage of y at 19 */
```

Figura 2: Parte do arquivo `C.main.dot` do programa GCD.

```
1  1-0    2-0    3-0    4-0    5-0    6-0    7-0
2  8-0    9-0   10-0   11-0    0-1   11-0   12-0
3  13-0   14-0    0-2   14-0   15-0   46-2   15-0
4  16-0   17-0   46-1   17-0   18-0   19-0   20-0
5  21-0   22-0    0-3   22-0   23-0   46-3   23-0
6  24-0   25-0   32-0   33-0   37-0
```

Figura 3: Parte do arquivo `trace.main.p0` do caso de teste 001 do programa GCD.

```
1 recv_no 15 :send_no 46 processo  2 com_a_tag  2
2 recv_no 17 :send_no 46 processo  1 com_a_tag  2
3 recv_no 23 :send_no 46 processo  3 com_a_tag  2
```

Figura 4: Arquivo `seq_sync.p4` do caso de teste 001 do programa GCD.

um grafo de fluxo interprocessos que pode ser percorrido de forma a obtermos todas as sincronizações do programa paralelo sendo testado.

As sincronizações obtidas são então escritas em arquivos separados (um arquivo por conjunto de sincronizações de todos os nós que executam `receive` do processo), localizados no diretório `./valimpi/seq_syncs`, que seguem a sintaxe dos arquivos de sincronização utilizados pela ferramenta ValiMPI para a reexecução do caso de teste. A sintaxe desses arquivos é exemplificada na Figura 4. Cada linha do arquivo segue a sintaxe `recv_no <n> :send_no<m> processo <x> com_a_tag <t>`, que significa que o `receive` localizado no nó `<n>` do processo ao qual o arquivo se refere sincroniza com o `send` localizado no nó `<m>` pertencente ao processo `<x>` e que tem a tag de mensagem `<t>`.

O `vali_sync` gerencia toda a execução do módulo, ou seja, ele cria os arquivos temporários, assim como verifica a existência dos arquivos criados pela ferramenta `ValiMPI` necessários à execução do `gera_sync`. Além disso, o `vali_sync` gerencia a reexecução de cada uma das sincronizações, substituindo os arquivos de sincronização do caso de teste em questão (salvando o caso de teste original), pelos arquivos criados pelo programa e chamando o módulo de execução da `ValiMPI` (`ValiExec`), passando os parâmetros da reexecução.

Como o `gera_sync` obtém todas as sincronizações do caso de teste, inclusive as sincronizações não executáveis, é necessário que o `vali_sync` gerencie também o tempo máximo que cada sincronização pode executar, terminando o processo quando esse limite é ultrapassado. Por fim o `vali_sync` restaura os dados originais do caso de teste e guarda as mensagens de todas as reexecuções no arquivo `syncs.out` localizado no diretório do caso de teste em questão.

3.1 Aspectos Operacionais

Como anteriormente citado, o módulo utiliza-se de vários arquivos gerados pela execução de outros módulos da ferramenta `ValiMPI`, portanto, o programa deve ser preparado para passar pela ferramenta e os diferentes módulos tem que ser executados em ordem para que o `ValiSync` seja capaz de executar corretamente.

Primeiramente o programa a ser testado deve ser instrumentado e os grafos de controle e de fluxo de dados devem ser gerados através do `ValiInst`. Devido às limitações desse módulo, o arquivo `.c` sendo instrumentado não pode apresentar definição de variáveis com typedef, tipo de retorno de função implícito, definição de variáveis em escopos menores do que o da função e definição da variável junto de sua declaração.

Depois de instrumentado o programa deve ser compilado com o script `vali_cc` e o módulo `ValiElem` deve ser executado para criar os arquivos e pastas necessários para a execução do programa através do módulo `ValiExec`. Após a execução desses módulos todos os arquivos e pastas necessários ao funcionamento do `ValiSync` para um determinado caso de teste, previamente executado com o `ValiExec`, foram criados e o módulo

```
1         MPI_Send(buf, 2, MPI_INT, 1, 1, MPI_COMM_WORLD);
2
3         // envia y e z
4         buf[0] = y;
5         buf[1] = z;
6         MPI_Send(buf, 2, MPI_INT, 2, 1, MPI_COMM_WORLD);
7
8         MPI_Recv(buf, 1, MPI_INT, MPI_ANY_SOURCE, 2, MPI_COMM_WORLD,
9                 &status);
10        x = buf[0];
11
12        MPI_Recv(buf, 1, MPI_INT, MPI_ANY_SOURCE, 2, MPI_COMM_WORLD,
13                &status);
14        y = buf[0];
15
16        if (x > 1 && y > 1)
```

Figura 5: Parte da função `main()` do arquivo `gcd-mpi.c`.

pode ser iniciado.

3.1.1 Exemplo Completo de Execução

Usando como exemplo o programa `GCD`, cujo código pode ser lido, em parte, na Figura 5, serão demonstrados todos os passos necessários para a correta execução do `ValiSync`.

Todos os comandos descritos durante o exemplo devem ser executados na pasta onde se encontra o código fonte do programa `gcd-main`.

Primeiro Passo: é executado o `Vali_Inst` para instrumentar o programa e gere os grafos de fluxo de controle. Isso é feito através do seguinte comando:

`vali_inst gcd-mpi.c`

Saída: Arquivos `gcd-mpi.c_instrumentado.c`, que é o código fonte instrumentado, e `C.main.dot`, que é o arquivo de descrição do grafo de fluxo de controle da função "main" do código fonte.

Segundo Passo: O programa é compilado pelo script `vali_cc`.

`vali_cc gcd-mpi.c_instrumentado.c -o gcd-mpi`

Saída: Arquivo `gcd-mpi`, que é o executável do programa.

Terceiro Passo: Geração dos elementos requeridos e descritores de critérios pelo `Vali_Elem` (a execução desse módulo não está diretamente realcionada à execução do `ValiSync`, mas é necessária para a execução do `Vali_Exec`)

`vali_elem 4 "main(0,1,2,3)"`

Saída: Elementos requeridos e descritores dos critérios no diretório `./valimpi/res/`.

Quarto Passo: O `Vali_Exec` é usado para executar um determinado caso de teste.

`vali_exec 1 run 4 gcd-mpi "12 44 36"`

Saída: saída do programa `GCD` instrumentado na tela. Os seguintes arquivos são criados no diretório `./valimpi/test_case0001`:

- `progname`, com o nome do executável;
- `stdout`, com o conteúdo da saída padrão;
- `stderr`, com o conteúdo da saída de erro;
- `args`, com os argumentos de entrada do programa;
- `stdin`, com o conteúdo da entrada padrão;
- `trace.main.p[0...3])`, com os arquivos de caminho de execução de cada processo; e
- `seq sync.p[0...3]`, com os arquivos de sequência de sincronização de cada processo.

Caso a implementação `LAM/MPI` esteja disponível, é gerado o arquivo `xmpi.lamtr`, que serve para a visualização da execução do programa através da ferramenta `XMPI` (incluída na `LAM/MPI`).

Quinto Passo: Com todas as saídas necessárias já geradas é possível então executar o `ValiSync` através do comando:

`vali_sync gcd-mpi`

A execução do módulo gera os arquivos de sincronização e faz a execução de cada uma dessas sincronizações. São produzidos os arquivos `lsTrace` e `lsDot`, que são arquivos de apoio usados durante a execução do módulo. No diretório `./valimpi/seq_syncs` são criados os arquivos `seq_syc.p[0..3]_[0..6]`, que contêm as diferentes sincronizações possíveis encontradas pelo módulo.

4 Conclusão

O módulo ValiSync se portou de maneira esperada durante os testes, criando todas as sincronizações, fazendo a sua execução e interrompendo aquelas que executavam por mais de um dado intervalo de tempo. Os testes foram executados nos seguintes programas:

(i) `GCD` (um programa de cálculo do máximo denominador comum);

(ii) `Integral` (um programa para o cálculo da integral de uma função através do método dos trapézios); e

(iii) `Prod-Cons` (exemplo clássico de produtor-consumidor).

Porém, quando testado para programas paralelos mais complexos, tal como o jantar dos filósofos, no qual ocorrem muitos `sends` e `receives` (principalmente quando executados em laços muito extensos), o módulo apresentou problemas com o número de sincronizações, o que o torna ainda inapropriado para testes em programas reais.

Referências

[1] Hausen A. C. Valimpi: Uma ferramenta de teste estrutural para programas paralelos no ambiente de passagem de mensagens mpi. Master's thesis, Universidade de São Paulo, 2005.

[2] R.H. Carver and Lei Y. Reachability testing of concurrent programs. *IEEE Transactions on Software Engineering*, 32(6):382–403, June 2006.

[3] Geist G.A., Kohl J.A., Papadopoulos P.M., and Scott S.L. Beyond pvm 3.4: What we've learned what's next, and why. In *Fourth European PVM-MPI Conference - Euro PVM/MPI'97*, Cracow, Poland, 1997.

[4] C.H. Hwang, K.C. Tai, and T.L. Huang. Reachability testing: An approach to testing concurrent software. *International Journal of Software Engineering and Knowledge Engineering*, 5(4):493–510, 1995.

[5] Souza P.S.L. Máquina paralela virtual em ambiente windows. Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), São Carlos, SP, maio 1996.

[6] M. Snir, S. Otto, H. Steven, D. Walker, and J.J. Dongara. *MPI: The Complete Reference*. The MIT Press, Massachussets, 1996.