



Visualizações de Plataformas com OVP

Anderson R. P. Domingues
anderson.domingues@acad.pucrs.br
Porto Alegre, 8 de Julho de 2018

Sumário

1 Introdução	1
2 Requisitos	2
3 Estrutura de uma Visualização	2
3.1 Frontend	3
3.2 Backend	4
Referências	6

1 Introdução

A ferramenta OVP permite monitorar dados de simulação de plataformas através do módulo de visualizações. Este módulo utiliza de um servidor HTTP que provê uma interface WEB para o monitoramento destes dados. Quando a simulação é iniciada, o servidor HTTP é instanciado, possibilitando monitorar a simulação com o auxílio de um web browser.

Existem dois tipos de dados que podem ser monitorados por estas interfaces. O primeiro tipo corresponde aos dados extraídos dos modelos da simulação, como por exemplo status do processador. Estes dados são específicos para cada modelos. O segundo tipo de dado é referente à API do próprio simulador. Alguns dados, como por exemplo a contagem de instruções, são comuns a todos os modelos de processadores. Para ambos os tipos de dados, o programador pode manipulá-los, utilizando a linguagem C, antes de enviá-los ao módulo de visualização. Um exemplo de visualização é mostrado na Figura 1.

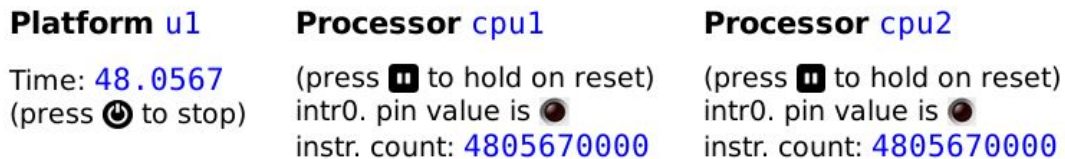


Figura 1. Exemplo de visualização de uma plataforma multiprocessada. São mostrados o tempo de simulação, status e contagem de instruções para dois processadores.

Além de permitir o monitoramento de informações em tempo de execução, as visualizações também permitem interagir com a plataforma simulada e com o simulador. Por exemplo, é possível forçar com que um certo processador permaneça no estado de reset. Também é possível atribuir valores a pinos específicos dos modelos. No exemplo da Figura 1 há três botões: um para terminar a simulação e outros dois que mantêm os processadores cpu1 e cpu2 no estado de reset. Para estes últimos, é possível pressioná-los novamente para que os processadores retornem a seu estado de operação usual.

As próximas seções mostram

2 Requisitos

Existem diferentes formas de se explorar visualização nas plataformas simuladas com OVP. Este documento aborda apenas visualizações na forma de harness. Para que uma simulação possa exibir a visualização, o parâmetro `--httpvis` precisa ser passado ao harness. Sem este parâmetro o servidor HTTP necessário à visualização não é instanciado. Nenhuma dependência externa é necessária, porém é possível utilizar bibliotecas de terceiros para aprimorar a visualização.

3 Estrutura de uma Visualização

A visualização de uma plataforma depende de dois componentes. O primeiro componente é o backend da visualização, isto é, o código que roda por trás das requisições HTTP executadas pelo browser. Ao iniciar a simulação, as bibliotecas do OVP geram um servidor HTTP que recebe requisições de um web browser e as responde com um documento HTML que contém as informações de visualização. Assim, é preciso que o simulador saiba como formatar os dados antes de enviá-los ao browser. O segundo componente é o frontend da visualização. O frontend é composto minimamente por um ou mais documentos HTML, que juntos compõem a organização da visualização no browser. A programação destes documentos HTML seguem os mesmos princípios de qualquer página web, exceto por alguns elementos especiais, que são específicos do OVP. A interação entre estes componentes é mostrado na Figura 2.

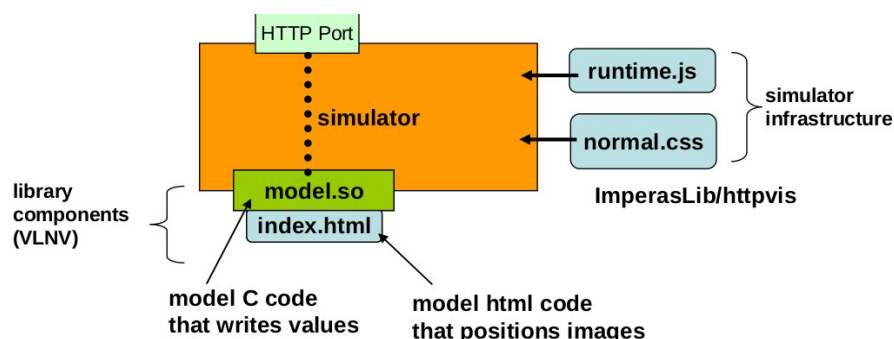


Figura 2. Interação entre os componentes de visualização.


3.1 Frontend

O documento HTML utilizado para a visualização (chamados de `index.html` na Figura 2) possui classes de marcações especiais, que são substituídas pelos valores das variáveis provenientes do monitoramento pelo backend. Estas marcações podem ser do tipo `ovptext`, `ovppower`, `ovpled`, `ovpreset`, entre outros. Para fins de praticidade, as marcações mais utilizadas são brevemente descritas abaixo.

OVPTTEXT: Esta é a marcação mais genérica, e permite que qualquer valor, numérico ou de texto, seja exibido na visualização. Todas os caracteres em azul da Figura 1 são exibidos através desta classe de marcação. Qualquer elemento do documento HTML que possui a classe `ovptext` será substituído pelo valor correspondente ao seu identificador. Por exemplo, o elemento abaixo será substituído pelas informações da variável `cpuName1`. A substituição, como dito anteriormente, é realizada pelo componente backend.

Marcação	Resultado
<code></code>	instr. count: 4805670000

OVPPPOWER (ex.,): Esta marcação adiciona um botão de power à visualização. Este botão, no exemplo da Figura 1, é utilizado para terminar a simulação. Entretanto, o comportamento pode ser adaptado. No exemplo abaixo a marcação é substituída por um botão de power que, ao pressionado, altera o estado da variável `power0`. Note que o ícone do botão, tal qual qualquer estilo associado, são construídos pela biblioteca `runtime.js` e `normal.css`, que são padrões do OVP. Porém, é possível reescrever o comportamento e estilo da marcação.


Marcação	Resultado
<code></code>	

OVPLED: Este componente apresenta um led na visualização. Este led é mostrado como ligado ou desligado de acordo com o valor da variável associada. No exemplo abaixo, o led ficará aceso quando o valor do pino

interrupt0 possuir o valor 1.

Marcação	Resultado
<code></code>	

OVPRESET: A marcação reset permite adicionar um botão de dois estados à visualização. Existem outras marcações de dois estados, como por exemplo ovpswitch, porém todas apresentam o mesmo comportamento, diferenciando-se apenas no aspecto visual. Para o exemplo abaixo, utilizado no exemplo da Figura 1, o botão apresenta o símbolo de pause para quando o processador está em execução e o símbolo de resume para quando o processador está parado no estado de reset.

Marcação	Resultado (running e paused)
<code></code>	

Em adição às marcações, existem alguns outros parâmetros da visualização que podem ser configurados através do frontend. O mais importante deles é o intervalo de atualização da visualização. Um intervalo pequeno entre as atualizações permite monitorar as informações de simulações quase em tempo real, porém pode fazer com que a simulação fique lenta. Em contrapartida, um intervalo muito grande implica na exibição de dados defasados e, portanto, pode não ser útil para a visualização. O parâmetro de atualização é configurado no elemento body do arquivo index.html, conforme abaixo.

```
<body onload="startRefresh('/', 400);">
```

3.2 Backend

No componente backend é programado o comportamento do simulador ao receber e enviar requisições HTTP para um browser. Existem dois tipos de requisição: get e post. As requisições do tipo get são utilizadas pelo browser para solicitar a página de visualização para o simulador. Isto acontece tanto no momento em que a página de visualização é enviada para o navegador, quanto para quando os dados são atualizados em tela, conforme o intervalo configurado, explicado anteriormente. As requisições do tipo post são utilizadas pelo navegador para enviar dados para o simulador. As marcações do tipo ovpreset, no exemplo da Figura 1, quando pressionados, enviam uma requisição ao simulador solicitando que um determinado processador fique ou não preso no estado de reset.

É preciso que o comportamento do simulador para cada tipo de requisição seja definido. As definições se dão pela programação das funções OCL_HTTP_POST_FN e OCL_HTTP_GET_FN, que tratam, respectivamente, as requisições do tipo get e post.

O tratamento das requisições post é o mais simples. Quando uma requisição post é recebida, uma cópia do DOM (document object model) do HTML é enviado ao simulador. A partir daí é possível testar se um determinado elemento (ex., um botão), possui um determinado estado, (ex., ligado ou desligado), e tomar a ação correspondente (ex., interromper a simulação). A função OCL_HTTP_POST_FN para o exemplo da Figura 1 é mostrado abaixo, onde os estados dos botões de power e reset dos processadores cpu1 e cpu2 são testados, respectivamente.

```

static OCL_HTTP_POST_FN(post) {
    myDataP my = userData;
    if (strstr(body, "power0=clicked")) {
        opMessage("I", PREFIX "_SW", "PowerSwitch pushed.
        terminating simulation.");
        opModuleFinish(my->mi, 0);
    } else if (strstr(body, "reset1=clicked")) {
        opMessage("I", PREFIX "_SW", "RESET 1 pushed.");
        opNetWrite(my->reset1, (opNetValue(my->reset1) == 1) ? 0 : 1);
    } else if (strstr(body, "reset2=clicked")) {
        opMessage("I", PREFIX "_SW", "RESET 2 pushed");
        opNetWrite(my->reset2, (opNetValue(my->reset2) == 1) ? 0 : 1);
    }
}

```

As requisições get, por sua vez, são responsáveis pela substituição das marcações feitas no arquivo index.html pelos valores das variáveis associadas. Para facilitar a substituição, a biblioteca OCL do OVP fornece a função oclHTTPKeyPrintf. A função OCL_HTTP_GET_FN, utilizada no exemplo da Figura 1, é mostrada abaixo.

```

static OCL_HTTP_GET_FN(get) {
    myDataP my = userData;
    oclHTTPElementOpen(ch, "ovpelement");
    oclHTTPKeyPrintf(ch, "moduleInstName0", opObjectName(my->mi));
    oclHTTPKeyPrintf(ch, "moduleSimTime0", "%g",
(double)opModuleCurrentTime(opObjectRootModule(my->reset1)) );
    oclHTTPKeyPrintf(ch, "reset1", "%d", opNetValue(my->reset1));
    oclHTTPKeyPrintf(ch, "cpu1icount", FMT_64d, opProcessorICount(my->cpu1));
    oclHTTPKeyPrintf(ch, "cpu2icount", FMT_64d, opProcessorICount(my->cpu2));
    oclHTTPKeyPrintf(ch, "reset2", "%d", opNetValue(my->reset2));
    oclHTTPKeyPrintf(ch, "power0", "0");
    oclHTTPKeyPrintf(ch, "cpu1int", "%d", opNetValue(my->interrupt1));
    oclHTTPKeyPrintf(ch, "cpu2int", "%d", opNetValue(my->interrupt2));
    oclHTTPKeyPrintf(ch, "cpuName1", "%s", opObjectName(my->cpu1));
    oclHTTPKeyPrintf(ch, "cpuName2", "%s", opObjectName(my->cpu2));
    oclHTTPElementClose(ch, "ovpelement");
    oclHTTPSend(ch);
}

```

Ainda no arquivo de backend (harness.c), é necessário fazer a ligação dos métodos que farão o tratamento das requisições, habilitar a porta para o servidor web e apontar o arquivo de visualização. A ligação dos métodos (binding) é feita pela definição da variável mets, conforme mostrado abaixo. Também é mostrada abaixo a chamada da função opModuleHTTPOpen, que é responsável por abrir a porta do servidor

HTTP e carregar o arquivo de visualização, situado no subdiretório harness/httpvis. Note que o arquivo principal precisa ser chamado index.html.

```
octHTTPMethods mets = { .get=get, .post=post, .userData=my };  
opModuleHTTPOpen(mr, &mets, httpvisportnum, "harness/httpvis");
```

Referências

- [1] Imperas Software. *Visualization used in Virtual Platforms | Open Virtual Platforms*. Online. Visitado em 9 de Julho de 2018. Disponível em: <http://www.ovpworld.org/visualization-used-in-virtual-platforms>
- [2] Imperas Software. *Installation, Getting Started with OVP, and Cross-Compiling Applications | Open Virtual Platforms*. Online. Visitado em 9 de Julho de 2018. Disponível em: <http://www.ovpworld.org/installation-getting-started-with-ovp-and-cross-compiling-applications>
- [3] Imperas Software. *Writing C Platforms and Modules using the OVP OP API | Open Virtual Platforms*. Online. Visitado em 9 de Julho de 2018. Disponível em: <http://www.ovpworld.org/writing-c-platforms-and-modules-using-the-ovp-op-api>
- [4] Imperas Software. *iGen Platform and Module Creation User Guide | Open Virtual Platforms*. Online. Visitado em 9 de Julho de 2018. Disponível em: <http://www.ovpworld.org/igen-platform-and-module-creation-user-guide>

Apêndice A - Exemplo de Visualização

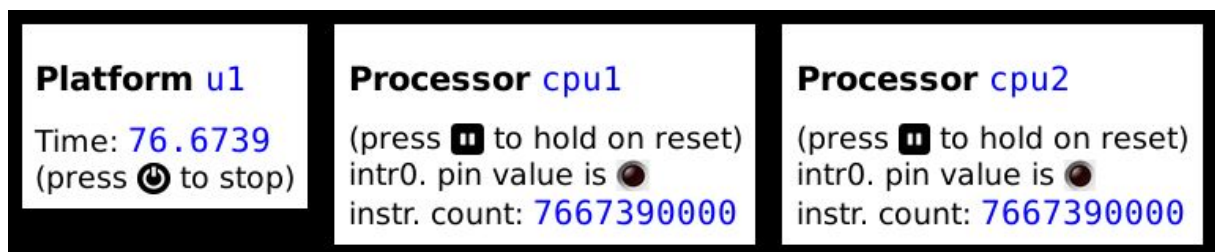
Para executar o exemplo em anexo, utilize a seguinte linha de comando.

```
$/example.sh
```

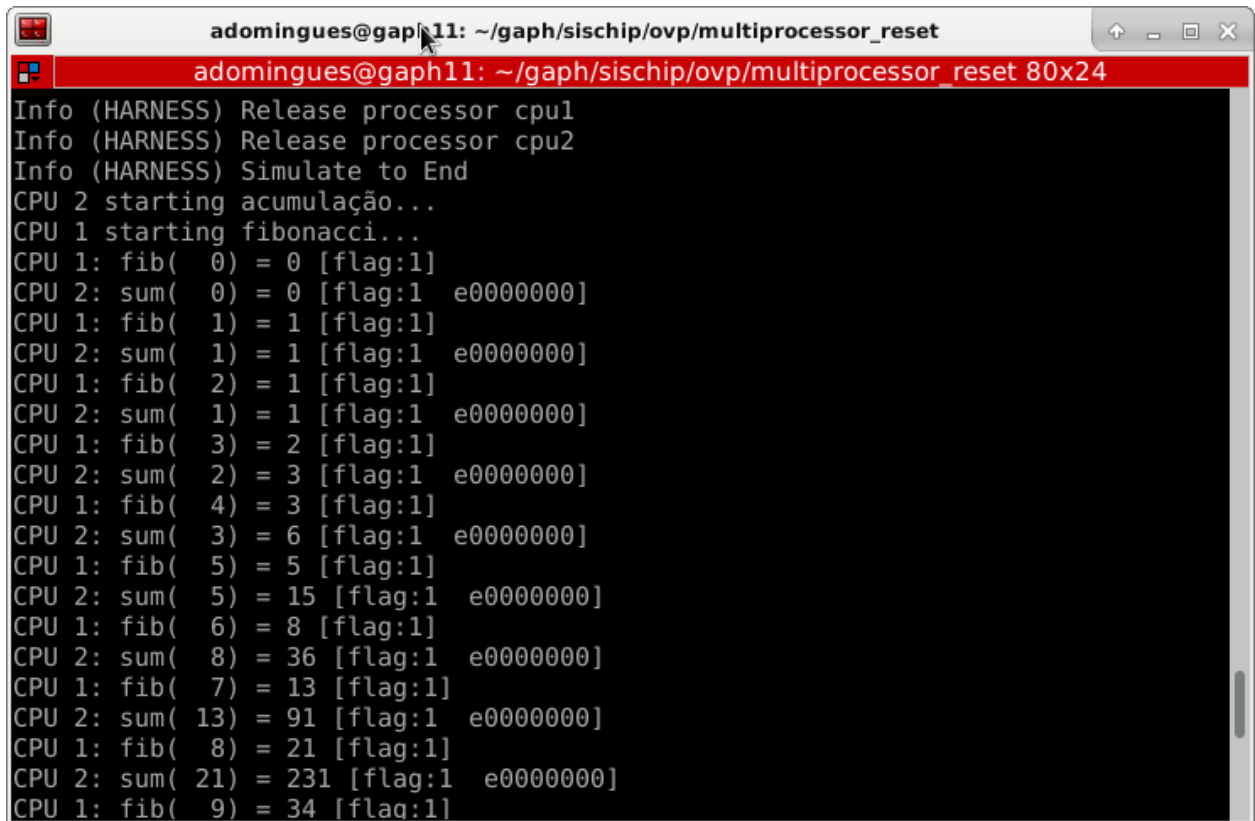
Quando perguntado "Do you want to open a browser to see the visualization [y/N]", pressione y e, em seguida, enter.

```
Do you want to open a browser to see the visualization [y/N]  
y
```

O navegador padrão deve abrir, mostrando a visualização da plataforma conforme abaixo.



A saída da aplicação pode ser vista no terminal da simulação. Para este exemplo, é utilizada uma aplicação distribuída que calcula os 40 primeiros números da sequência de Fibonacci. Esta aplicação possui dois software, cada um executando em um processador distinto com memória privada. O primeiro calcula os valores da sequência e envia para o segundo processador. O segundo software recebe os valores computados e os acumula. A comunicação é feita por uma memória compartilhada.



```
adomingues@gaph11: ~/gaph/sischip/ovp/multiprocessor_reset
adomingues@gaph11: ~/gaph/sischip/ovp/multiprocessor_reset 80x24
Info (HARNESS) Release processor cpu1
Info (HARNESS) Release processor cpu2
Info (HARNESS) Simulate to End
CPU 2 starting acumulação...
CPU 1 starting fibonacci...
CPU 1: fib( 0) = 0 [flag:1]
CPU 2: sum( 0) = 0 [flag:1 e0000000]
CPU 1: fib( 1) = 1 [flag:1]
CPU 2: sum( 1) = 1 [flag:1 e0000000]
CPU 1: fib( 2) = 1 [flag:1]
CPU 2: sum( 1) = 1 [flag:1 e0000000]
CPU 1: fib( 3) = 2 [flag:1]
CPU 2: sum( 2) = 3 [flag:1 e0000000]
CPU 1: fib( 4) = 3 [flag:1]
CPU 2: sum( 3) = 6 [flag:1 e0000000]
CPU 1: fib( 5) = 5 [flag:1]
CPU 2: sum( 5) = 15 [flag:1 e0000000]
CPU 1: fib( 6) = 8 [flag:1]
CPU 2: sum( 8) = 36 [flag:1 e0000000]
CPU 1: fib( 7) = 13 [flag:1]
CPU 2: sum( 13) = 91 [flag:1 e0000000]
CPU 1: fib( 8) = 21 [flag:1]
CPU 2: sum( 21) = 231 [flag:1 e0000000]
CPU 1: fib( 9) = 34 [flag:1]
```

A execução acontecerá somente se o ambiente estiver devidamente configurado. Caso o ambiente não esteja configurado corretamente, execute as seguintes linhas de comando para configurá-lo.

```
$module load ovp/20170201
$source /soft64/imperas/ferramentas/64bits/Imperas.20170201/bin/setup.sh
setupImperas /soft64/imperas/ferramentas/64bits/Imperas.20170201
```