

# HeMPS 8.x Tutorial

Por: Marcelo Ruaro, Guilherme Madalozzo, Alzemiro Silva, Anderson Sant'Ana, Fernando Gehm Moraes

Contato: marcelo.ruaro@acad.pucrs.br

Versão atual: **8.5**

Data da versão: 11/08/2017

## Recursos suportados:

- *Reclustering*
- Repositório de aplicações unificado
- Contagem de instruções
- Simulação em SystemC
- DMNI
- Simulação VHDL
- Hold
- Mapeamento
- Migração de Tarefas

## ➤ Tutorial de execução local

1. Ambiente necessário:
  - a. Ubuntu
  - b. Java 7 ou superior (Java usado somente na depuração)
2. Clonar o repositório da hemps do github:  
#git clone <https://github.com/GaphGroup/hemps>
3. Entrar no diretório **hemps**
4. Entrar no diretório **hemp8.5**
5. Executar o comando **tree**. Observar a árvore de diretórios.

### Descrição da árvore de diretórios:

A HeMPS é composta por 6 diretórios principais:

**applications:** contém o código fonte (.c) das aplicações;

**build\_env:**

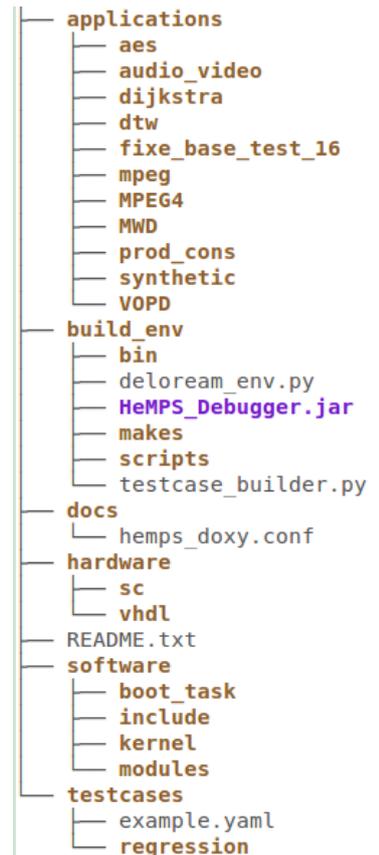
**bin:** contém arquivos executáveis úteis para a configuração da memória e depuração;

**scripts:** contém os scripts úteis e o script *hemp.pl*, que é executado para configurar a plataforma através de parâmetros informados em tempo de projeto. Estes parâmetros se encontram nos arquivos de testecase (.hmp);

**hardware:** contém a descrição dos componentes hardware em VHDL e SystemC RTL;

**software:** contém os códigos fontes da camada de software;

**testcase:** contém os testcases que serão utilizados pelo script *hemp-run*



6. Para criar as variáveis de ambiente, utilizar os comandos especificados no endereço:

[http://www.inf.pucrs.br/hemps/getting\\_started.html](http://www.inf.pucrs.br/hemps/getting_started.html)

ou no ambiente das máquinas do laboratório:

```
# module load systemc  
# module load modelsim  
# module load mips-elf/4.1.1-hemps
```

7. Adicionar os caminhos dos binários da hemps e do mips-elf ao PATH:

```
export HEMPS_PATH=$HOME/hemps/hemps8.5  
export PATH=$PATH:$HEMPS_PATH/build_env/bin
```

\*Conforme o caminho da pasta

Dica: Implementar os passos 6 e 7 no arquivo “.bashrc”

## ➤ Criação de nova aplicação

1. Os próximos passos permitem criar um novo cenário de testes, chamado **teste**, contendo duas aplicações que serão criadas neste tutorial, chamadas: **pc1** e **pc2**
2. **Criação do arquivo de configuração do cenário de teste:** No diretório **testcase** deve-se criar o arquivo de configuração (**teste.yaml**), adicionando as 2 aplicações que serão criadas para serem executadas. Para configurar o **.yaml**, adicione o seguinte texto ao arquivo:

```
hw:  
  page_size_KB: 32  
  tasks_per_PE: 1  
  repository_size_MB: 1  
  model_description: sc  
  noc_buffer_size: 8  
  mpsoc_dimension: [4,4]  
  cluster_dimension: [2,2]  
  master_location: LB  
  
apps:  
  - name: pc1  
    start_time_ms: 1 ms  
  - name: pc2  
    start_time_ms: 5 ms
```

Pode-se notar que foram adicionadas duas aplicações que iniciam em **1 e 5 ms**.

### A estrutura básica de um testcase é descrita como segue:

#### [tasks per pe]

Número de tarefas por PE, um valor típico configurado pela equipe do GAPH é uma faixa entre 1 e 6 tarefas (1-6).

#### [model\_description]

Descrição da infraestrutura de hardware. **sc** - simulação em SystemC (GCC), **scmod** - simulação em SystemC utilizando a ferramenta ModelSim, **rtl** - simulação em VHDL utilizando a ferramenta ModelSim

#### [noc\_buffer\_size]

Parâmetro com número inteiro que permite configurar a profundidade dos buffers dos roteadores da NoC, um valor típico é **8**.

#### [dimensions]

Dois parâmetros numéricos inteiros que permitem definir a dimensão do MPSoC. Não é suportado uma dimensão menor que 2x2. Ex: [6,6]

#### [cluster\_size]

Dois parâmetros numéricos inteiros que permitem definir a dimensão de cada cluster do MPSoC. Não é suportado uma dimensão menor que a dimensão do próprio MPSoC, além disso o número de processadores do cluster deve ser divisível (com resto igual a 0) pelo número de processadores do MPSoC. Ex: [3,3]

#### [masters location]

Localização onde o processador mestre global irá ser gerado. Atualmente a única posição suportada é LB (Left Bottom).

Os próximos parâmetros estão relacionados as aplicações que executarão no sistema, a mesma estrutura deve ser utilizada para cada aplicação que irá executar.

#### [name]

Nome da aplicação que será executada na plataforma, este nome deve ser o mesmo das aplicações presentes no diretório *applications*.

#### [start\_time\_ms]

Tempo de início de execução da aplicação descrita acima, este valor numérico inteiro é representado em **um** (microsegundos -  $\mu$ m) ou **ms** (milissegundos). Esse parâmetro é opcional, caso não estiver presente a aplicação irá iniciar assim que o sistema puder admiti-la.

Exemplo de instanciação da aplicação mpeg, que inicia em 1ms.

apps:

```
- name: mpeg
  start_time_ms: 1 ms
```

### 3. Agora, deve-se criar as duas aplicações que foram informadas no arquivo de configuração.

- A. Ir no diretório **applications** e criar dois diretórios, cada um, com o nome de cada aplicação.

```
cd applications
mkdir pc1
mkdir pc2
```

- B. Criando a aplicação pc1 (produtor-consumidor 1)

```
cd pc1
gedit taskA.c
#include <api.h>
#include <stdlib.h>

Message msg;

int main(){

    int i, j,t;

    Echo("task A started.");
    Echo(itoa(GetTick()));

    for(i=0;i<500;i++){
        msg.length = 30;
        for(j=0;j<30;j++) msg.msg[j]=i;
```

```
        Send (&msg, taskB);
    }

    Echo (itoa (GetTick ()));
    Echo ("task A finished.");
    exit ();
}

gedit taskB.c
#include <api.h>
#include <stdlib.h>

Message msg;

int main () {
    int i;

    Echo ("task B started.");
    Echo (itoa (GetTick ()));

    for (i=0; i<500; i++) {
        Receive (&msg, taskA);
    }

    Echo (itoa (GetTick ()));
    Echo ("task B finished.");
    exit ();
}
```

C. Criando a aplicação pc2 (produtor-consumidor 2): basta copiar os arquivos .c de pc1 para pc2

```
cd ../pc2/
cp ../pc1/* .
```

## Execução do cenário de teste criado

1. Ir até o diretório **testcases**
2. Executar o comando (hemp-run <nome do cenário> <tempo em ms>):

```
hemp-run teste.yaml 100
```

## Depuração

1. A simulação em SystemC irá começar, a janela do *HeMPS Debugger* irá abrir automaticamente.
2. Ler o texto introdutório, abaixo, sobre o *HeMPS Debugger*:

### Ferramenta de depuração: HeMPS Debugger

A ferramenta de depuração foi desenvolvida em Java e o seu executável (.jar) encontra-se no diretório */bin* da plataforma. A ferramenta é totalmente gráfica e possui diversas funções que permitem avaliar a simulação verificando o comportamento das tarefas, serviços e comunicação. Entre suas principais funções destacam-se:

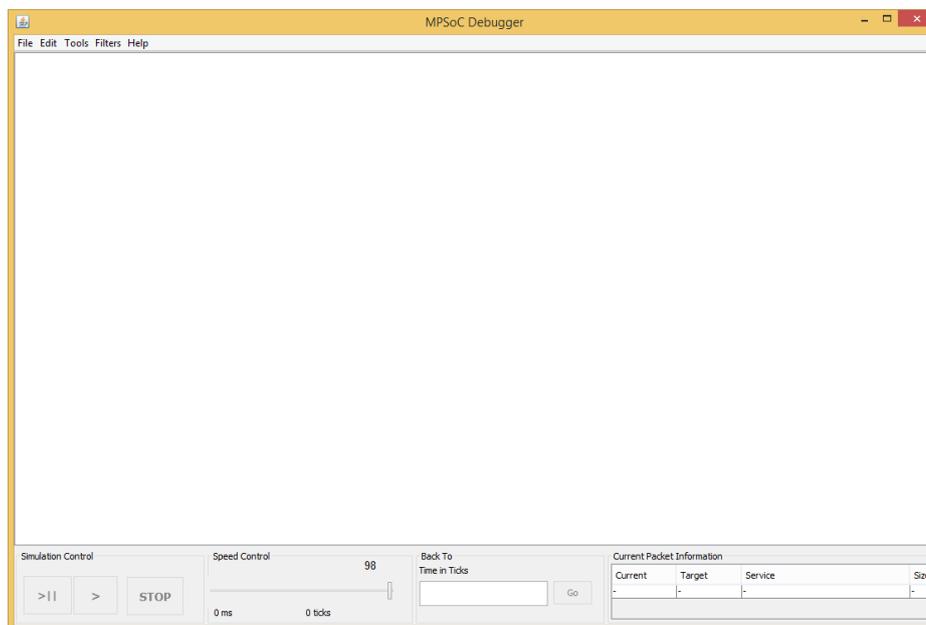
- Acompanhamento da execução em tempo de simulação;
- Visualização do mapeamento das tarefas;

- Verificação dos protocolos, tanto de comunicação quanto de gerenciamento;
- Verificação dos percentual de utilização de cada link entre um par de roteadores;
- Verificação do percentual de comunicação em cada roteador, tanto no nível de serviço quanto no nível de flits;
- Medição do tempo de cada protocolo;
- Visão geral da distribuição de serviços executados por cada PE;
- Verificação do sincronismo de comunicação entre tarefas baseado no protocolo *read-request*;
- Suporte multiplataforma (testado em Windows, Linux e MAC OS);
- Visualização dos logs das tarefas (Delorean).

O HeMPS Debugger é executado automaticamente na simulação por SystemC (GCC). Quando configurar um testcase com simulação por ModelSim, deve-se dar um clique duplo sobre o ícone HeMPS\_Debugger.jar presente no diretório */bin* da plataforma.

OBS: É necessário ter a JVM (*Java Virtual Machine*) instalada na versão 6 ou acima.

Ao abrir a ferramenta a seguinte tela é exibida:

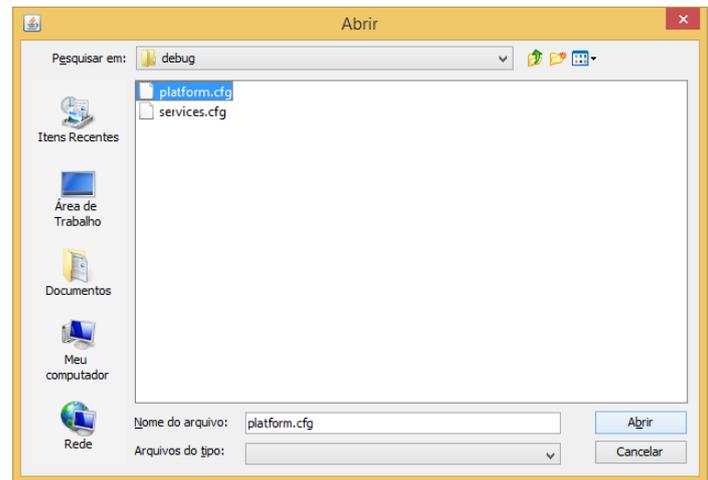


Nesta tela além do menu superior, há os seguintes painéis de controle de uso:

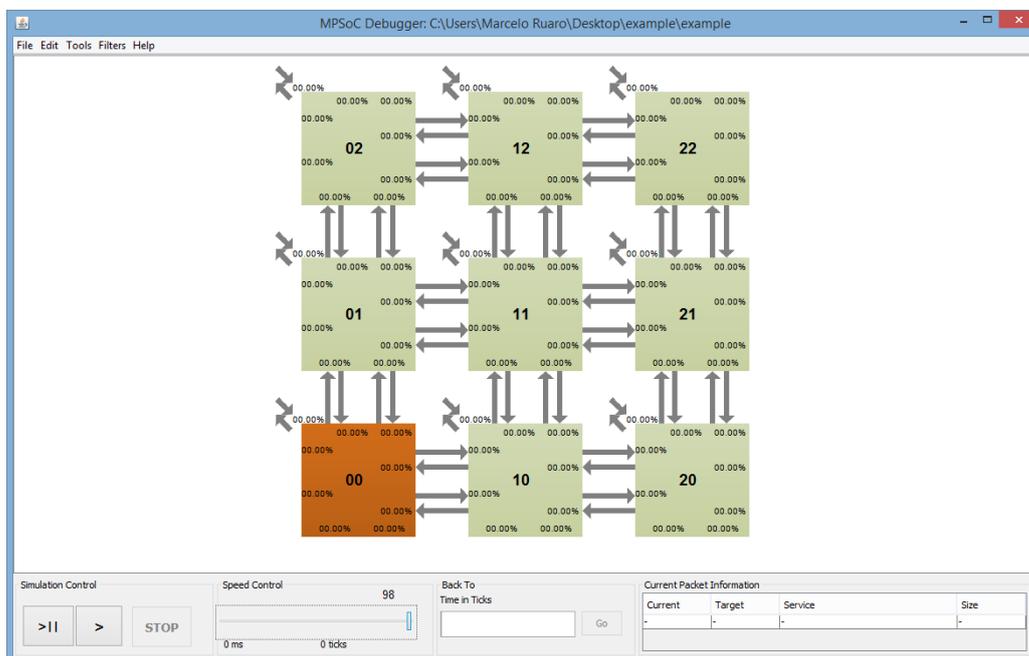
- Controle de simulação (botões passo-a-passo, executar e parar);
- Controle da velocidade de simulação (em porcentagem de 0 a 100%);
- Campo para voltar a um determinado tempo de simulação;
- Informações sobre o pacote atual: roteador atual, roteador alvo, serviço e tamanho do pacote em flits e largura de banda alocada.

Para abrir uma nova depuração é necessário ir em Menu->New Debugging (Ctrl + N) e informar o arquivo *platform.cfg* do *testcase* pelo qual se deseja depurar. Esse arquivo encontra-se no diretório `<nome_testcase>/debug`, e é criado somente após a simulação ter iniciado (GCC ou ModelSim).

Ao realizar essa etapa a ferramenta irá carregar e gerar uma visualização gráfica do MPSoC, como representado pela figura abaixo. O mestre global está representado em laranja, em azul os mestres locais e em verde os escravos.



As setas que compõe a representação gráfica indica um enlace entre dois roteadores (setas maiores) ou entre um roteador e o elemento de processamento (setas menores nos cantos superior esquerdo). Como essa ferramenta foi inicialmente projetada para um MPSoC com canais duplicados, a comunicação entre roteadores é duplicada. Porém, em um MPSoC que utiliza uma NoC com canal simples, a ferramenta colorirá apenas uma seta (canal).

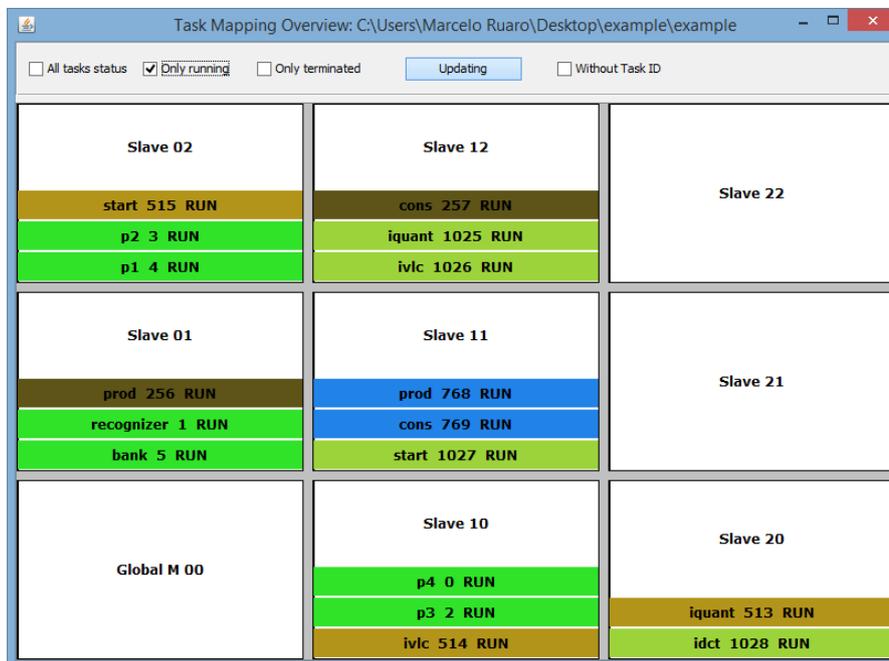


Com o MPSoC carregado, pode-se verificar o andamento da simulação configurando a velocidade de representação das comunicação, através da barra *speed control* e clicando no botão *play* do painel *Simulation Control*. Um valor abaixo ou igual a 98%, além de avançar a simulação, irá colorir em vermelho a seta (enlace) pelo qual o pacote está sendo enviado. Além disso, os valores representados com o símbolo de %, em cada enlace, representa a porcentagem de largura de banda utilizada para aquele enlace. Além da execução automática, pode-se optar por verificar passo-a-passo o envio de pacotes. Para isso, primeiramente a simulação

deve ser parada e depois deve-se utilizar o botão “passo-a-passo” para analisar cada pacote transmitido de PE por PE.

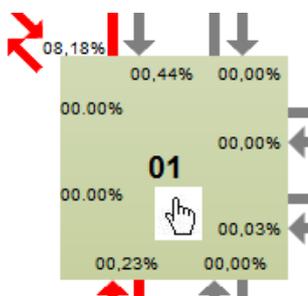
### Mapeamento e Estado das Tarefas

A aba de mapeamento e estado das tarefas pode ser aberta clicando em Tools->Task Mapping Overview (Ctrl + t). A figura abaixo apresenta a tela exibida para o *testcase example*. Nesta tela, as aplicações são representadas por cores, no caso da figura pode-se perceber que há 5 aplicações em execução, o PE slave 10 executa 3 tarefas simultaneamente, o PE 20 executa 2 tarefas, e assim por diante.

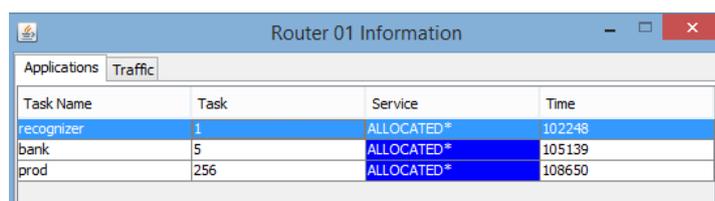


### Sincronismo de Comunicação entre Tarefas

Outra funcionalidade da HeMPS Debugger é o mecanismo de verificação da sincronização entre MESSAGE\_REQUEST e MESSAGE\_DELIVERY, que ocorre entre um par de tarefas comunicantes. Esse recurso pode ser acessado através de um clique simples sobre um PE, na tela principal.



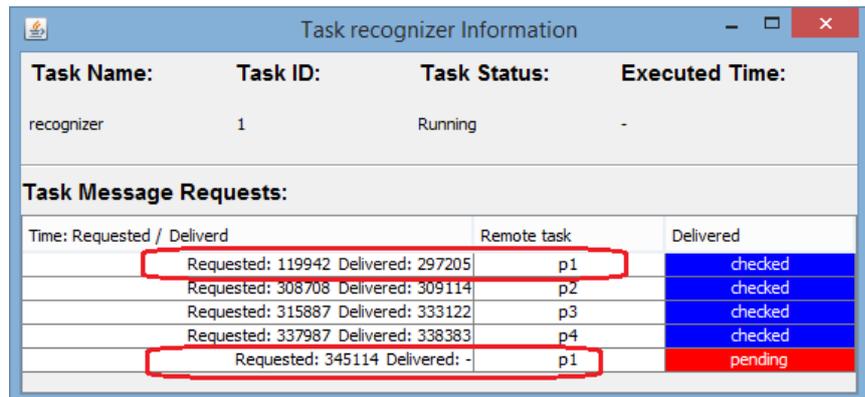
Essa ação irá abrir uma tela onde pode-se verificar em detalhes o estado de cada tarefa, bem como o volume de comunicação do PE.



The screenshot shows a window titled "Router 01 Information" with a "Traffic" tab selected. It contains a table with the following data:

Task Name	Task	Service	Time
recognizer	1	ALLOCATED*	102248
bank	5	ALLOCATED*	105139
prod	256	ALLOCATED*	108650

Ao dar duplo clique em uma tarefa que contenha o *Service* ALLOCATED, a tela de verificação de sincronismo de comunicação de tarefa (no caso acima, *recognizer*) será exibida. Nesta tela, representada pela figura abaixo, pode-se perceber que, por exemplo, a tarefa *recognizer* enviou um MESSAGE\_REQUEST para a tarefa *p1* no tempo de 119.942 (medido em ciclos de *clock* ou *ticks*), e obteve o respectivo MESSAGE\_DELIVERY da tarefa *p1* aos 297.205 *ticks*. Ou seja, ficou bloqueada por 177.263 *ticks*. Além disso, é possível perceber que a última linha da tabela representa o envio de um MESSAGE\_REQUEST para a tarefa *p1*, mas que ainda não foi recebido. Portanto, a tarefa está em estado de WAITING aguardando a mensagem.



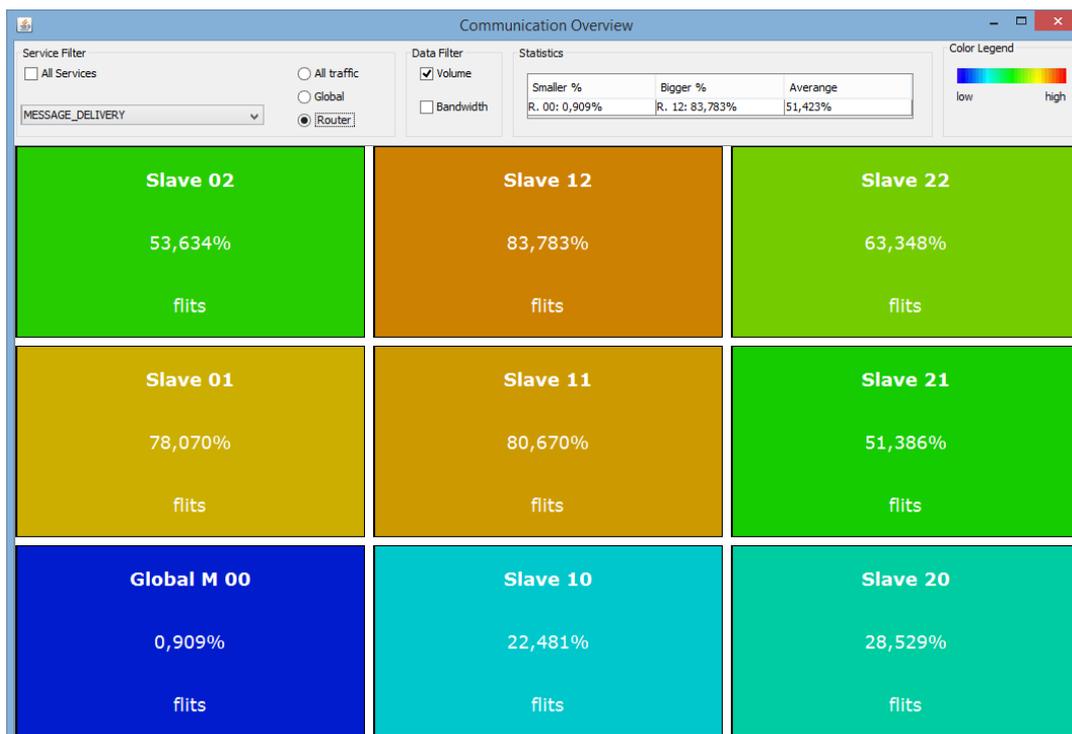
Task Name:	Task ID:	Task Status:	Executed Time:
recognizer	1	Running	-

Task Message Requests:			
Time: Requested / Delivered	Remote task	Delivered	
Requested: 119942 Delivered: 297205	p1	checked	
Requested: 308708 Delivered: 309114	p2	checked	
Requested: 315887 Delivered: 333122	p3	checked	
Requested: 337987 Delivered: 338383	p4	checked	
Requested: 345114 Delivered: -	p1	pending	

### Distribuição da Comunicação

A tela de distribuição de comunicação é exibida através da opção Tools->Communication Overview. Nesta tela é possível avaliar a distribuição de comunicação no nível de serviço entre os PEs. Diferentes cores, de acordo com a frequência de cor, representa o volume de comunicação do PE. É possível selecionar o serviço pelo qual se deseja verificar a distribuição ou também considerar todos os serviços (*All Services*). Além disso, é possível considerar três pontos de vista: (i) *All traffic*, que relaciona o serviço escolhido em relação a todo tráfego de comunicação contendo todos os serviços trafegados na rede; (ii) *Global*, que relaciona o serviço selecionado com todo o tráfego na NoC, que contenham aquele serviço; e (iii) *Router*, que relaciona o serviço selecionado com todo o tráfego de comunicação do roteador, contendo todos os serviços que foram recebidos por ele.



### Debugger Log Reader for MPSoC (Delorean)

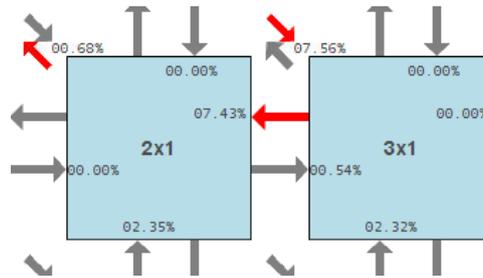
O Delorean é uma ferramenta de usuário que visa simplificar, em uma interface gráfica, a análise de log de cada tarefa simulada no MPSoC. A figura abaixo apresenta tal ferramenta com 3 aplicações. 2 aplicações (mpeg e



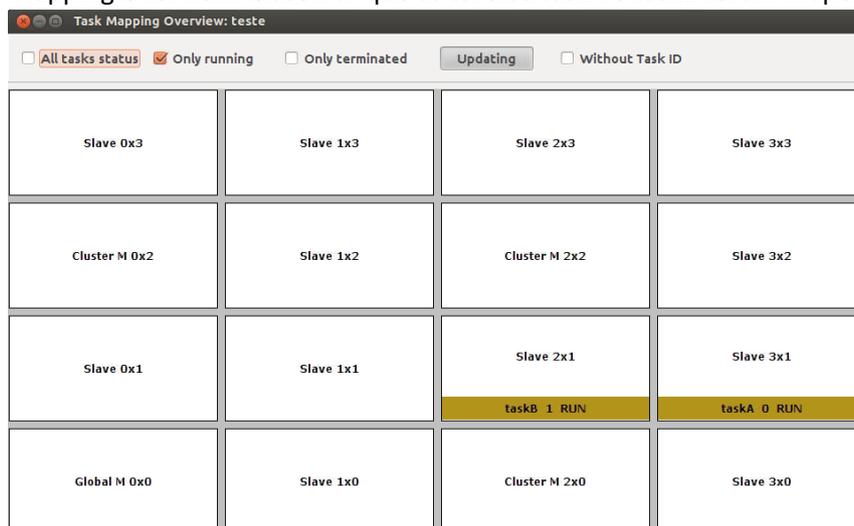
4. Clicar no botão (Play). **NOTA: A simulação gráfica pode ser reiniciado a qualquer momento em File->Reset Simulation.** Para **parar** a simulação clicar em **STOP**, o botão >|| mostra o pacote roteador por roteador (hop por hop)



5. Observar o que acontece, diversas comunicações são realizadas entre os PEs, essa fase representa a inicialização do sistema, onde todos os PEs são inicializados e conhecem quem é o seu PE mestre. Após esse período inicial a comunicação concentra-se somente entre os PEs 2x1 e 3x1.



6. Ir em Tools->Task Mapping Overview. Observar que as tarefas taskA e taskB foram mapeadas nestes PEs.

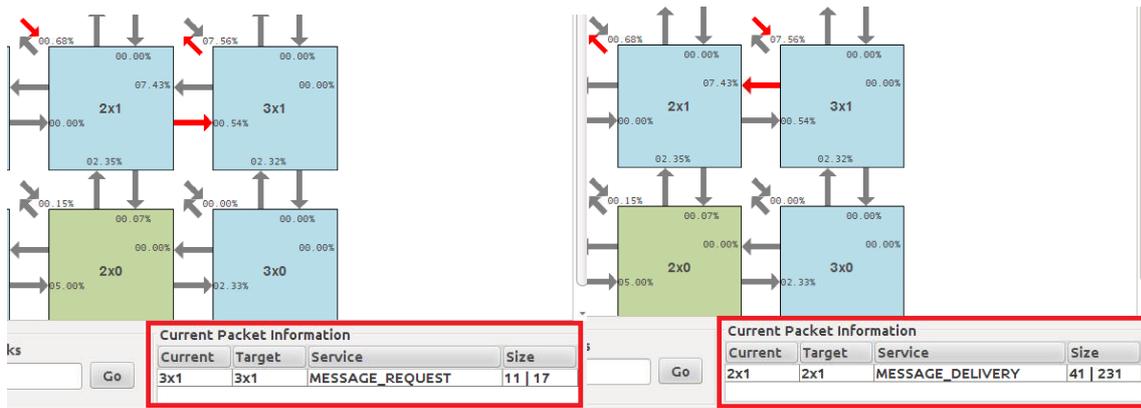


The 'Task Mapping Overview' window shows a grid of tasks. The 'Only running' checkbox is checked. The 'Updating' button is visible. The grid shows the following tasks and their status:

Slave 0x3	Slave 1x3	Slave 2x3	Slave 3x3
Cluster M 0x2	Slave 1x2	Cluster M 2x2	Slave 3x2
Slave 0x1	Slave 1x1	Slave 2x1	Slave 3x1
Global M 0x0	Slave 1x0	Cluster M 2x0	Slave 3x0

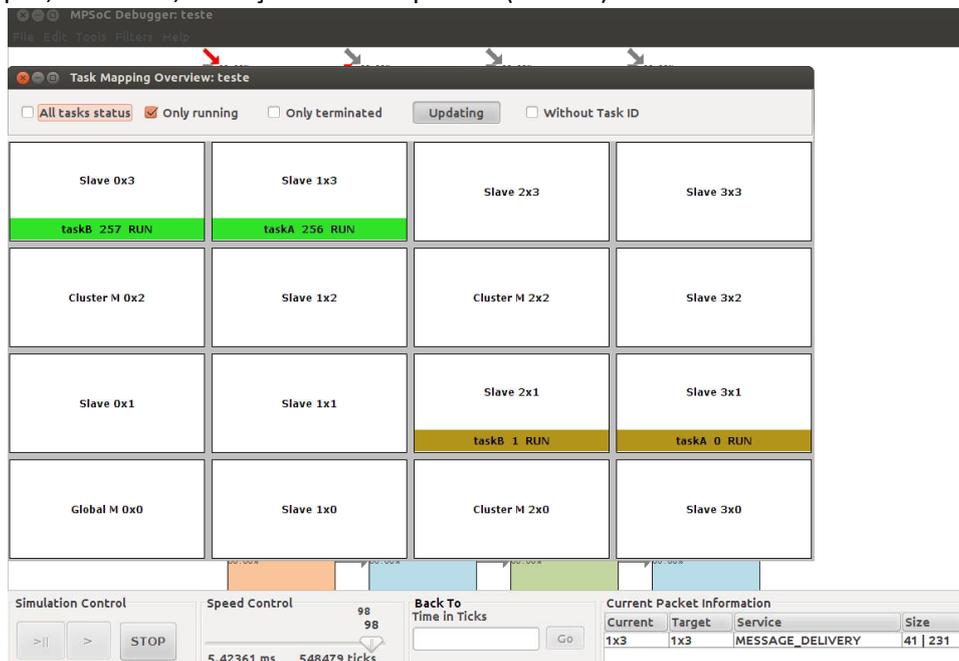
The 'taskB 1 RUN' and 'taskA 0 RUN' cells are highlighted in yellow.

7. As tarefas se comunicam trocando pacotes do tipo MESSAGE\_REQUEST e MESSAGE\_DELIVERY.

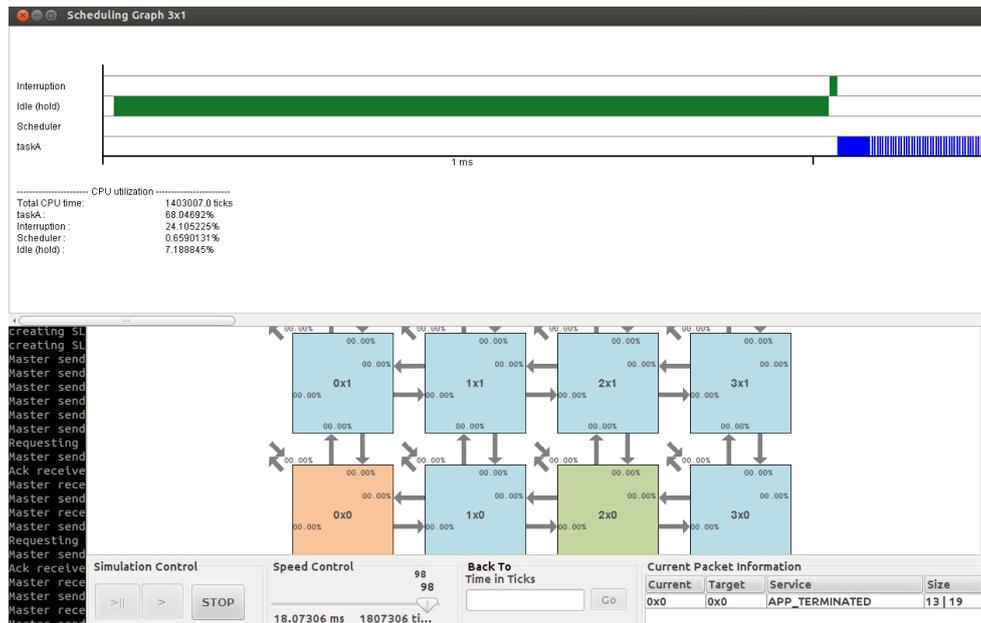


8. Entender o porquê da ordem dos pacotes, porquê somente o PE 2x1 que envia MESSAGE\_REQUEST e porquê somente o PE 3x1 que envia MESSAGE\_DELIVERY

9. Observar na tela de mapeamento de tarefas que a aplicação pc2 começa a executar, note o tempo em que as tarefas de pc2, em verde, começam a ser mapeadas ( 5ms !?).



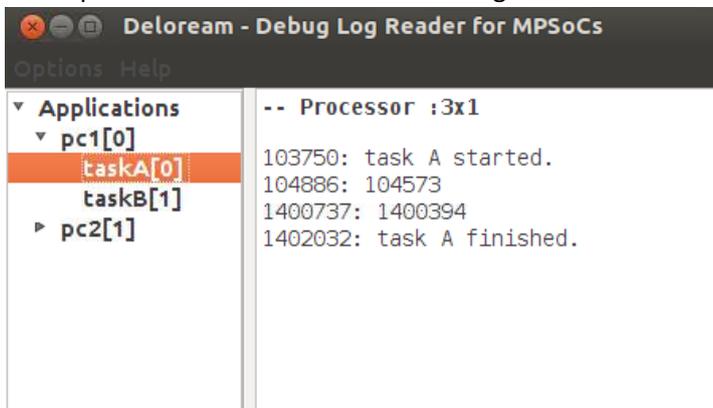
10. Explorar a depuração do uso da CPU do processador 3x1, para isso clicar sobre o processador, clicar na aba Scheduling e clicar no botão Open Scheduling Graph. Observar se o obtivo é semelhante à figura abaixo.



11. Explorar a ferramenta de carga de comunicação. Para isso clicar em **Tools->Communication Overview**. Observar a cor de cada PE.

## USO DO DELOREAM

12. O DELOREAM pode ser aberto em **Tools->Deloream**
13. Expandir a aplicação pc1, e clicar duas vezes sob a tarefa **taskA**. Observar que os logs gerados pela tarefa correspondem aos inseridos em seu código fonte.



**FIM DO TUTORIAL**