Como executar sua simulação em mútiplos processadores sem modificar nem seus módulos nem o SystemC

Tiago R. C. Falcão[1], **Liana Duenha** [2], **Rodolfo Jardim de Azevedo**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 – 13.083-852 – Campinas – SP – Brasil

[2]Faculdade de Computação – Universidade Federal do Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brasil

{tiago.falcao,rodolfo}@ic.unicamp.br, lianaduenha@facom.ufms.br

Abstract. *Simulation is one of the main stages in the validation process in systems design; in this stage, system architects can verify the correctness, behavior, and performance of the target system. SystemC is a System-level Description Language (SLDL), a C++ language extension that supports different abstraction levels. The downside is its sequential simulation model that does not take advantage of the parallel processing capabilities. This paper proposes a generic technique that allows the simulation of a set of SystemC components by encapsulating each one in a process, which can be scheduled over cores or distributed on a cluster. The main advantage of this approach is that it parallelize SystemC-TLM2 simulators using the original SystemC Kernel and models.*

Resumo. *A simulação é uma etapa importante no desenvolvimento de sistemas computacionais, por meio de qual os sistemas em desenvolvimento são testados e tem comportamento e desempenho avaliados. SystemC é uma Linguagem de Descrição de Sistemas (SLDL), uma extensão da linguagem C++ com suporte a diferentes níveis de abstração. O SystemC simula sequencialmente todo o sistema sem aproveitar possível potencial de processamento paralelo. Esse artigo propõe uma abordagem genérica para permitir a simulação de cada componente num processo distinto, que pode ser escalonado em diferentes processadores, locais ou distribuídos. A principal vantagem é paralelizar simuladores descritos em SystemC sem a necessidade de modificar os modelos ou o SystemC.*

1. Introdução

Mais componentes continuarão a ser adicionados num *chip* mesmo considerando possíveis revisões da Lei de Moore[Moore 1975], como ocorrido no passado, quando o prazo para duplicar o número de componentes num *chip* foi modificado de um para dois anos. Hoje, o mercado vivencia uma estimativa de 18 meses [Mack 2011]. O fato é que o número de componentes presentes em um chip cresce rapidamente e continuará crescendo. Nos processadores, isto representa mais núcleos e funcionalidades de propósito específico, de tal forma que esse acúmulo de funções justifica serem chamados de *System on Chip* (SoC). Essa crescente complexidade requer adaptações no processo de simulação destes sistemas para correta compreensão do comportamento do SoC nos estágios iniciais do projeto, antes que o mesmo seja produzido fisicamente.

Linguagens de Descrição de Sistemas (do inglês, *System Level Description Languages* - SLDLs) fornecem um conjunto de bibliotecas, tipos de dados, funcionalidades, núcleo de simulação e componentes de descrição de hardware para modelagem e simulação de um sistema em alto nível de abstração. SystemC [Sys 2012] é uma SLDL com modelo de descrição baseado na linguagem C++ e utilizada para modelagem e verificação de sistemas em diferentes níveis de abstração. A temporização pode ser feita de forma vaga (*loosely-timed*), com melhor desempenho, ou de forma aproximada (*approximately-timed*), mais próxima a um hardware real, utilizando sincronização baseada em eventos e anotação de tempo nas transações. Recentemente, SystemC se tornou uma escolha popular dos projetistas de SoCs e processadores embarcados [Ezudheen et al. 2009].

SystemC-TLM2 é um conjunto de funcionalidades para modelagem da camada de comunicação entre componentes do sistema e é parte integrante do padrão SystemC [Aynsley 2009]. TLM2 introduz o conceito de desacoplamento temporal, no qual as *threads* de todos os módulos possuem relógios locais que podem avançar em relação ao tempo global da simulação, o que reduz gargalos e fornece melhor desempenho em troca de uma perda de precisão na temporização. A comunicação utiliza troca de mensagens, chamadas de *payloads*, que encapsula os dados da requisição em si e extensões que podem ser especializadas pelo usuário.

Podemos simular um sistema simples composto por um módulo processador e uma memória conectados diretamente por TLM2 (Figura 1). Cada leitura/escrita deve ser encapsulada em um *payload* que contenha todos os dados da requisição e enviada a partir do núcleo pelo seu soquete inicializador para o soquete de destino do componente de memória. Assincronamente, o *payload* é processado pela memória, que deve ler ou escrever na posição indicada pela mensagem, e respondida pelo mesmo caminho.

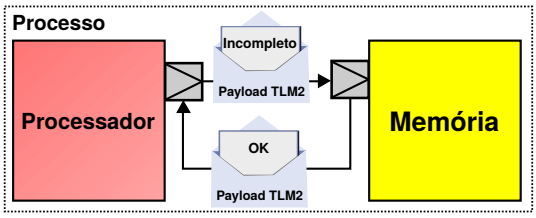

Figura 1. Comunicação TLM2 entre componentes de núcleo e memória

Um fator limitante para acelerar simuladores SystemC é que seu núcleo de simulação é sequencial e baseado em eventos discretos. O escalonador seleciona um processo por vez, que será executado até o seu final ou até que seja feita uma chamada da função `wait` para que o próprio processo suspenda sua execução e volte para a fila do escalonador. Nas simulações de sistemas com vários processadores, o núcleo do SystemC ocupa aproximadamente 50% do tempo de simulação e apenas 30% são destinados ao comportamento de todos os núcleos de processamento [Duenha et al. 2012]. O SystemC escalona a execução de um módulo por vez, mesmo que a máquina base tenha suporte à execução paralela de múltiplos processos [Ezudheen et al. 2009]. Por exemplo, uma plataforma virtual com 16 processadores será simulada em apenas um núcleo de processamento da máquina real, mesmo que esta contenha múltiplos processadores. Por que não utilizar todos os núcleos ou mesmo um *cluster* de servidores?

Atualmente, pesquisas para aumento de desempenho de simuladores descritos em SystemC estão focadas na sua distribuição da simulação em diferentes processa-

dores e os melhores ganhos foram obtidos no particionamento manual da simulação [Ezudheen et al. 2009, Huang et al. 2008, Mello et al. 2010, Chopard et al. 2006]. Essas técnicas não modificam o escalonador padrão do SystemC.

Esse artigo propõe uma solução que é:

Paralela Tenta usar o máximo poder de processamento disponível no sistema computacional onde a simulação é executada.

Simples Uma prova de conceito que funciona e ainda demonstra possibilidades para futuras otimizações específicas para um determinado sistema modelado;

Modular Um módulo SystemC que pode ser adicionado ou removido sem necessitar modificar os módulos existentes e que requeira o mínimo de configuração para tal fim;

Validável Não modifica o núcleo SystemC, que é amplamente utilizado e devidamente validado.

1.1. Trabalhos Relacionados

O modelo TLM com tempo distribuído (TLM-DT) [Maia et al. 2006] propõe a modificação do núcleo SystemC para uma redução do tempo de simulação para sistemas modelados com *transaction-level modeling* (TLM). Seguindo o princípio de simulação paralela baseada em eventos discretos (PDES), cada processo tem sua própria definição de "tempo local". Esta nova abordagem usa um mecanismo de simulação paralela, chamado SystemC-SMP, capaz de expor o poder computacional de máquinas com múltiplos processadores. Neste modelo, cada processo é explicitamente paralisado enquanto aguarda por algum evento. Esse tipo de mecanismo de exclusão mútua permite ao escalonador executar outro processo. Quando um processo paralisado é notificado do evento esperado, o relógio local é atualizado e o processo volta a executar.

Como extensão do trabalho anterior, foi apresentada uma estratégia de modelagem para SoC com quantidade massiva de processadores (MPSoCs) baseada em memória compartilhada utilizando a abordagem TLM-DT e o mecanismo do SystemC-SMP [Mello et al. 2010]. Do ponto de vista do núcleo do simulador, a plataforma em TLM-DT é vista como um conjunto de `SC_THREAD`s que conhecem como se comunicar e sincronizar com as demais. Porém, o mapeamento destas *threads* deve ser explicitamente controlado pelo projetista do sistema por diretivas de configuração. A limitação dessa abordagem é que todos sistemas devem ser devidamente adaptados para esta versão modificada do SystemC, o que não é uma tarefa trivial.

Chopard et al. [Chopard et al. 2006] permite o particionamento de processos SystemC, com mínimas mudanças no núcleo original do SystemC e na própria sintaxe da linguagem de modelagem. Nesta abordagem, uma cópia do escalonador executa em cada nó de processamento e simula um subconjunto dos módulos do sistema modelado. A consistência dos dados é garantida através dos eventos entre os processos. Para assegurar a sincronização, um nó, definido como principal, é responsável por receber as anotações de tempo do próximo evento esperado para cada nó e atualiza o tempo global da simulação. Desta forma, os demais nós podem atualizar seus relógios locais assim que recebam as informações temporais do nó principal.

Baseado nos mesmos conceitos descritos acima, Huang et al. [Huang et al. 2008] propôs ganho de eficiência com a distribuição geográfica dos modelos SystemC. Neste

contexto, o mecanismo consiste de um conjunto de sistemas Linux em rede e cooperam para computar a mesma simulação. A principal contribuição deste trabalho é implementar uma biblioteca para simulações SystemC distribuídas, chamada de SCD (*SystemC Distribution*), que paraleliza as simulações e não modifica a biblioteca original.

O ArchSC [Ziyu et al. 2009] é ambiente SystemC paralelo construído sobre uma plataforma de simulação em nível de sistema de larga escala e paralela, chamado ArchSim, que fornece ferramentas para gerenciar o particionamento e a troca de mensagens entre os módulos. O sistema é dividido e distribuído em múltiplos processadores, cada subsistema consiste de um simulador independente, logo escalonadores independentes. Esse trabalho não se aplica ao TLM.

Roth et al. [Roth et al. 2013] apresenta uma metodologia baseada em SystemC-TLM para acelerar a simulação de MPSoCs com *Network on-Chip*, que combina ambas vantagens: modelagem em diferentes níveis de abstração e execução paralela em máquinas com múltiplos núcleos; integra o paradigma paralelo de modelagem de eventos discretos com o conceito de escalonadores minimalistas. O resultados apresentados demonstram que a abordagem alcança significativa melhoria de duas ordens de grandeza versus a execução sequencial RTL, enquanto preserva a analisabilidade e exibe moderada perda de precisão.

Peeters et al. [Peeters et al. 2010] apresenta uma abordagem para distribuir simulações em processadores de um *cluster* com uma política de sincronização híbrida, utilizando a *Message Passing Interface* (MPI). Duas instancias devem sincronizar somente quando elas tem alguma dependência (como, por exemplo, compartilhamento de dados). Assim, dois componentes que não tenham dependências, não precisam sincronizar entre eles. Porém, a sincronização global acontece em determinados pontos durante a simulação.

Outra metodologia de paralelização usa uma abstração mista em CPUs e GPUs para alcançar melhor desempenho da simulação [Sinha et al. 2012]. Dado um sistema em SystemC, parte dos modelos são adaptados para rodar em GPUs CUDA. Consequentemente, um novo SystemC é implementado. Baseado na mesma metodologia, Vinco et al. [Vinco et al. 2012] apresenta uma abordagem de simulação para RTL-SystemC em GPUs incluindo técnicas de escalonamento estático para redução da quantidade de eventos de sincronização. Um grafo de dependências é construído baseado nos sinais de entrada e saída para cada processo SystemC, e então é gerado o escalonador estático.

Os trabalhos mais recentes, Sniper [Carlson et al. 2011], Graphite [Miller et al. 2010], ZSim [Sanchez and Kozyrakis 2013] and PriME [Fu and Wentzlaff 2014], alcançam expressivos resultados utilizando o Pin [Luk et al. 2005] como núcleo da tradução dinâmica de binários (DBT), assim somente funcionam em arquiteturas Intel, tanto a arquitetura simulada quanto a hospedeira.

Os trabalhos relacionados não resolvem o problema de simular múltiplos componentes, não somente processadores, numa infraestrutura moderna com múltiplos processadores distribuídos sem a necessidade de sensíveis mudanças no código fonte. Muitas das soluções propostas reduzem o tempo de simulação, porém muitas se baseiam em uma versão modificada do SystemC ou necessitam que os componentes do sistema modelado sejam explicitamente escritos para a abordagem. Qualquer modificação nos modelos re-

quer um grande conjunto de testes de regressão a fim de evitar novos erros possam ser incluídos nas adaptações necessárias. Além disso, modificações no SystemC significam uma ruptura da implementação padrão que produz um código sem o devido estudo e verificação como o original apresenta.

2. TLM2 sobre TCP

O padrão TLM2 define um protocolo para simulação em nível de transação via troca de mensagens assíncronas, suportando nativamente barramentos de memória e é também extensível para suportar outros modelos de barramento. A troca de mensagem é amplamente utilizada em programas modernos, e quando assíncronos, oferecem uma maior facilidade para paralelização. O requerente pode enviar uma mensagem e concorrentemente continuar a realizar trabalho enquanto não recebe a resposta.

Nossa proposta vai além, utilizar o particionamento lógico dos módulos em TLM2 para separar a simulação em vários processos, com isso nós oferecemos mais tempo de processador para executar o comportamento de cada módulo. Metade do tempo de execução de um processo SystemC é utilizada pelo núcleo da biblioteca, independente do número de componentes simulados [Duenha et al. 2012]. Desejamos que apenas um subconjunto de componentes do sistema como um todo concorra pela outra metade, assim cada subconjunto terá o custo do núcleo mas de forma geral terão mais tempo de processador para seus comportamentos. Como prova de conceito, adotamos soquetes TCP que oferecem comunicação entre processos, distribuídos ou não, com ordenação e correção de erros.

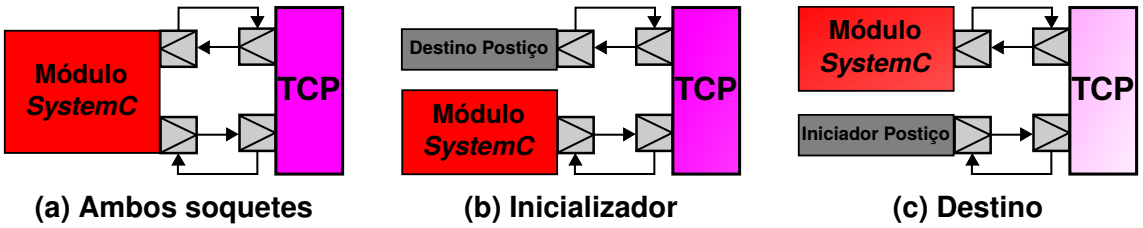

Figura 2. Conexões com o módulo TCP

O módulo de TLM2 sobre TCP conecta-se com os módulos preexistentes utilizando os soquetes de inicialização e destino (Figura 2a), isso possibilita que possa ser conectado a processadores, memórias, roteadores de NoC, etc. Conforme o estado de um *payload* o módulo determina qual soquete TLM2 deverá ser utilizado. Os módulos costumam apresentar apenas um tipo de soquete TLM2, inicializador ou destino, nestes casos, por requerimentos do SystemC, um módulo postiço deve ser conectado ao soquete não utilizado. Deve-se aproveitar a oportunidade para adicionar código de verificações nestes módulos extras, uma vez que nenhum *payload* do sistema deve passar por eles. Junto com o módulo TCP, fornecemos módulos de inicialização (Figura 2b) e de destino (Figura 2c).

Normalmente, os componentes TLM2 de uma simulação SystemC usam a memória compartilhada do processo para transmitir o *payload* apenas por referência. Porém, quando separamos a simulação em processos distintos, o objeto de *payload* e todos conteúdos referenciados (dado e extensões) precisam ser recriados em cada novo contexto. Quando um *payload* deixa pela primeira vez seu processo de origem, ele recebe uma extensão com um marcador único para registrar o contexto original dele. Todos

os *payloads* recebidos via TCP são checados se são pacotes retornando ao processo original ou são estrangeiros. Os estrangeiros são devidamente alocados neste contexto e construídos para evitar qualquer inconsistência na estrutura de dados, no momento que deixam o processo transitório, toda memória relacionada é liberada. Os pacotes quando retornam aos seus processos de inicialização, são identificados e apenas a estrutura de dados original é atualizada/sobrescrita com os valores recebidos.

O *payload* não contém diretamente todas as informações: o dado é apontado por um ponteiro para a memória do processo e qualquer extensão é um objeto referenciado (Figura 3a). Quando comunicados a outros processos, todos os dados devem ser enviados explicitamente para a correta reconstrução da estrutura no processo destino. Nesta implementação, adotamos um empacotamento simples (Figura 3b) que cria um bloco que contém todos dados necessários do *payload* e das extensões testadas. Neste caso, a região do dado é o único campo com tamanho variável, ele é estrategicamente colocado no final do pacote e precedido do seu comprimento, para uma decodificação rápida e simples. Campos opcionais, como extensões, são adicionadas manualmente no bloco a ser transmitido.

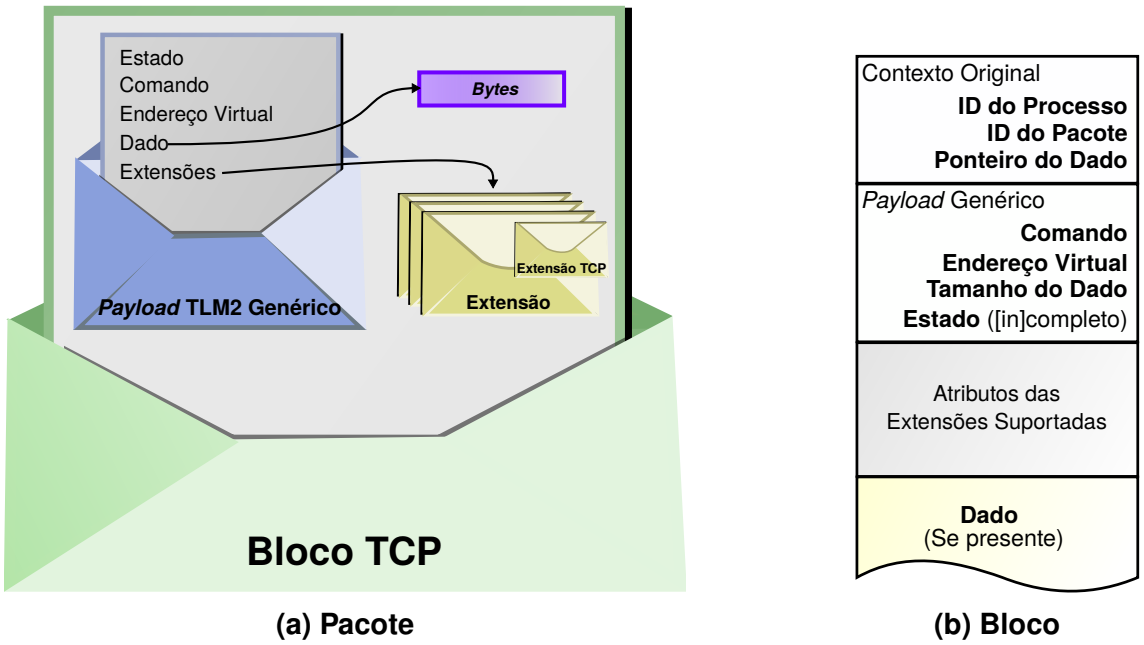

Figura 3. A mensagem TCP

Nem todos atributos dos payloads são propagados via TCP (Figura 3b). Do *payload* original são copiados o comando, o estado da resposta, o endereço virtual e o tamanho do dado. O ponteiro original para o dado é armazenado nos dados de anotação do contexto, junto com o marcador único do processo origem e o identificador do *payload*, já que o endereço de memória apontado somente tem valor lógico para o contexto original. Nos testes realizados, adicionamos uma extensão relativa ao sistema modelado para armazenar informações do roteamento em uma NoC, sendo adicionada ao bloco transmitido antes do dado.

Configuramos os soquetes TCP para enviar assim que possível os pacotes no *buffer* para redução da latência, mesmo assim, raramente um bloco é enviado isolado e completo

em um pacote TCP. O módulo TCP é responsável por armazenar os pacotes TCP, montar um bloco completo e reconstruir o *payload* e suas extensões e, se necessário, alocar a memória para o dado. Para testes, podemos forçar exatamente um bloco por comunicação TCP completando um pacote TCP com zeros.

Os testes integração criam uma rede em malha 2D de produtores e consumidores, e cada configuração é testada sem (Exemplo 1) e com (Exemplo 2) os módulos TCP. Foram testadas diversas configurações que tentam forçar os canais de comunicação com simples *ping-pong*, utilizando janelas de 3 a 10 mensagens antes de um *wait* (escalonamento de *threads* SystemC), e disparadas 100 mensagens seguidas de cada nó para todos demais sem aguardar o retorno da resposta. Os testes são executados em cada compilação do código fonte do módulo TCP, como descritos na Tabela 1 com sistemas de até 200 componentes.

```
1     //Binding South with North
2     southRouter->N_init_socket.bind(northRouter->S_target_socket);
3
4     //Binding North with South
5     northRouter->S_init_socket.bind(southRouter->N_target_socket);
```

Exemplo 1. Trecho da conexão de dois roteadores sem o módulo TCP

```
1     //Binding South
2     southTCP = new tlm_tcp("SouthTCP");
3     southRouter->N_init_socket.bind(southTCP->tsocket);
4     southTCP->isocket.bind(southRouter->N_target_socket);
5     southTCP->start_server(6801);
6     southTCP->connect("127.0.0.1", 6802);
7
8     //Binding North
9     northTCP = new tlm_tcp("NorthTCP");
10    northRouter->N_init_socket.bind(southTCP->tsocket);
11    northTCP->isocket.bind(northRouter->N_target_socket);
12    northTCP->start_server(6802);
13    northTCP->connect("127.0.0.1", 6801);
```

Exemplo 2. Trecho da conexão de dois roteadores usando o módulo TCP

Não é necessária uma barreira global para iniciar a simulação, cada processo chama a função *wait_connection()* antes de iniciar sua simulação para aguardar as conexões de volta apenas dos seus vizinhos. Cada módulo pode começar a produzir suas requisições e ir preenchendo as filas: do protocolo TLM2, interna do módulo TCP e da implementação da pilha TCP/IP.

2.1. Exemplo: Sistema com 1 Processador

O exemplo com um processador demonstra como dividir a simulação em múltiplos processos com o módulo TCP. Na Figura 1, é mostrado um exemplo simples da comunicação TLM2 entre um núcleo e uma memória. Para dividir essa simulação em processos distintos utiliza-se o módulo TCP para criar uma camada de comunicação transparente entre os dois processos (Figura 4).

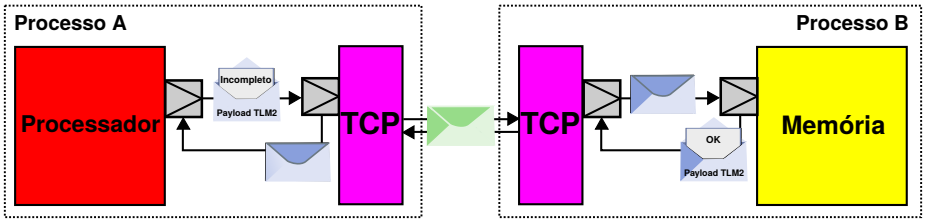

Figura 4. Comunicação TLM2 entre um núcleo e uma memória com módulo TCP

O processador envia um *payload* como incompleto com uma requisição do seu soquete inicializador para o soquete destino conectado. Não existe um conhecimento prévio do tipo do módulo conectado, apenas sabe-se que existe um soquete destino conectado.

O módulo TCP encapsula num bloco o *payload* e envia via conexão TCP preestabelecida para outro módulo TCP num processo distinto. O pacote TCP é lido e reconstruído como um *payload* com a requisição do processador.

A memória recebe o *payload* no seu soquete de destino com requisição e a processa exatamente como se a mesma tivesse vindo diretamente do soquete iniciador do processador. A resposta segue o caminho inverso, totalmente transparente para ambos módulos.

Se o processador desejar, ele pode enviar uma requisição de leitura com ponteiro nulo. A memória irá alocar e escrever o dado em seu espaço de endereçamento, retornando o resultado como *payload*. O módulo TCP irá gerenciar essa alocação, liberando a mesma do processo da memória e transferindo a mesma para o contexto original, corrigindo o ponteiro do objeto original.

Como descrito anteriormente, o processador e a memória necessitam dos módulos postiços para serem conectados aos soquetes não utilizados dos módulos TCP (Figura 2). O processador possui somente um soquete inicializador que é conectado ao soquete destino do módulo TCP, assim necessitando ocupar o soquete inicializador do módulo TCP. O mesmo ocorre com o soquete destino do módulo TCP da memória.

2.2. Exemplo: NoC - Malha 2x2 com 2 processadores

Um exemplo mais próximo a experimentos reais, uma *Network on Chip* (NoC) foi a inspiração original da técnica e é facilmente conectada com módulos TCP. Uma malha 2x2 (Figura 5) pode ser modelada com dois processadores, uma memória e um componente de exclusão mútua em *hardware*. Todas requisições são roteadas pelos módulos roteadores que determinam o destino com base no endereço, memória ou outro componente, e o sentido com base no estado do *payload*, incompleto ou completo.

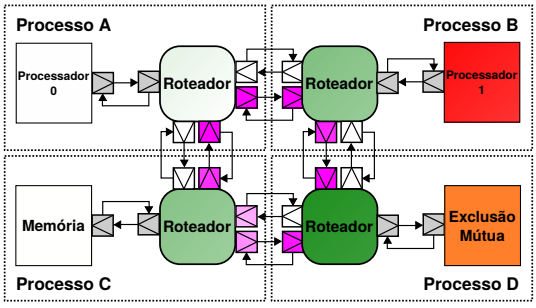

Figura 5. Exemplo de NoC 2x2

O módulo TCP pode ser utilizado para particionar esse sistema, um particionamento possível é cada *tile* (roteador e outro com-

ponente) seja executado em um processo diferente. Aqui adotamos roteadores de baixa complexidade, mas como qualquer conexão TLM2, poderíamos utilizar os módulos TCP entre os roteadores e o componente aumentando o número de processos.

Se o processador 1 solicitar uma leitura para uma posição de memória, o *payload* terá sua rota definida pelo primeiro roteador (o do processador 1) e seguirá sendo roteado pelos demais roteadores (e módulos TCP, transparentemente) até o roteador do seu destino, a memória. A resposta pode seguir a rota inversa ou qualquer outra determinado pela lógica do roteador. O módulo TCP não modifica o comportamento nem os módulos tem conhecimento se os *payloads* estão passando por módulos TCP.

3. Avaliação Experimental

Nós avaliamos a técnica em duas categorias de experimentos: artificiais e um benchmark para SoCs com multiprocessadores. Nesta seção, vamos descrever os testes e resultados obtidos.

3.1. Produtor - Consumidor

Nosso primeiro conjunto de experimentos são módulos artificiais para testar a escalabilidade e requerimentos. Os experimentos variavam a quantidade de um módulo produtor-consumidor TLM2, que apenas depende das bibliotecas do padrão SystemC-TLM2, para alcançar um grande número de processos distintos.

Para N módulos, criamos uma malha quadrada de $\left\lceil \sqrt{N} \right\rceil^2$ módulos roteadores, e preenchemos as $\left\lceil \sqrt{N} \right\rceil^2 - N$ últimas posições com componentes sem lógica. Cada módulo produtor-consumidor é configurado com um comportamento específico, quantos *payloads* deve ser enviados numa janela de envio sem aguardar resposta e quantos ele deve receber. Todos *payloads* gerados usam uma distribuição uniforme para determinar o endereço a ser solicitado. Com base neste, é gerado um código, para verificação de consistência, e armazenado no campo de dado para garantir que em qualquer módulo possa ser verificada a quantidade e a consistência dos *payloads*.

As configurações da Tabela 1 foram testadas com e sem o módulo TCP, que chegam a 200 processos e executam a troca de 120000 *payloads*.

3.2. MPSoCBench

Os testes mais realistas foram realizados com MPSoCBench [Duenha et al. 2014]. O MPSoCBench oferece um benchmark paralelo para SoCs com muitos processadores (até 64), e provê todos módulos: Processadores em 4 arquiteturas (ARM, PowerPC, MIPS e SPARC), memória, exclusão mútua em *hardware*, roteadores para NoCs e barramentos.

Baseado no ArchC [Azevedo et al. 2005, Rigo et al. 2011], os processadores incluídos são gerados de arquivos de descrição e simulam em nível de sistema as 4 arquiteturas descritas. A simulação em nível de sistema adiciona novas dificuldades no particionamento em múltiplos processos, o ArchC e os *softwares* são feitos para compartilhar não apenas a memória simulada, mas recursos reais do sistema e de um processo.

As chamadas de sistema (em inglês, *syscall*) são emuladas pelo ArchC, que assim como o SystemC, não tem conhecimento da divisão do simulador em processos distintos. Para garantir a consistência entre as chamadas de *syscalls* pelo software emulado

Tabela 1. Configurações testadas para o módulo Produtor-Consumidor

Tipo	Módulos	Roteadores	*Payloads*	Total de *Payloads*
Um para um	N	$\lceil\sqrt{N}\rceil^2$	P	P
0→0	1	1	100	100
0→1	2	4	100	100
1→0	2	4	100	100
0→2	3	4	100	100
2→0	2	4	100	100
0→3	4	4	100	100
3→0	4	4	100	100
0→0	9	9	100	100
0→2	9	9	100	100
0→6	9	9	100	100
0→8	9	9	100	100
2→0	9	9	100	100
6→0	9	9	100	100
8→0	9	9	100	100
8→8	9	9	100	100
Um para todos	N	$\lceil\sqrt{N}\rceil^2$	P	$(N * P)$
0⇒	9	9	100	900
4⇒	9	9	100	900
8⇒	9	9	100	900
Todos para um	N	$\lceil\sqrt{N}\rceil^2$	P	$(N * P)$
⇒0	9	9	100	900
⇒4	9	9	100	900
⇒8	9	9	100	900
Todos para todos	N	$\lceil\sqrt{N}\rceil^2$	P	$(N^2 * P)$
⇔	3	4	100	900
⇔	9	9	100	8100
⇔	144	144	10	207360
⇔	200	200	3	120000

rodando em vários processos na máquina real, criamos um controlador de *syscalls* externo a todos os processos de simulação e não dependente do SystemC, que centraliza a execução das *syscalls* em apenas um contexto. A pseudo *syscall* SBRK, para alocação dinâmica, precisa ser resolvida centralizada pelo controlador para que os processadores aloquem regiões distintas da memória simulada sem conflito. Todas chamadas relacionadas à manipulação de arquivos utilizam o controlador para compartilhar o mesmo descritor de arquivo (em inglês, *file descriptor*) do sistema operacional. Assim, esse identificador pode ser armazenado na memória simulada e utilizado pelo *software* emulado em qualquer processador.

Nossa técnica aumenta o custo de comunicação, o que é coerente com a discrepância de velocidades entre diferentes barramentos. Como em processadores modernos, o MPSoCBench depende de modelos com *caches* (e a devida coerência entre elas)

para alcançar melhores resultados reduzindo os custos excessivos de comunicação.

4. Conclusões

Apresentamos uma técnica genérica que permite particionar simulações SystemC em diferentes processos, não limitados a simulações de processadores de alguma arquitetura especifica. As comunicações utilizando o módulo TCP proposto é transparente aos demais módulos TLM2, que não necessitam de nenhuma modificação em seu código fonte. O que deixa a oportunidade para executar simulações dos mesmos módulos em ambientes distribuídos, locais, ou mesmo em SystemC puro sem o módulo proposto. Esse módulo TCP não requer nenhuma versão especial ou modificada da biblioteca SystemC, o que segue o padrão IEEE 1666-2011 [Sys 2012] e é a implementação amplamente utilizada na industria e academia.

Referências

(2012). IEEE Standard for Standard SystemC Language Reference Manual.

Aynsley, J. (2009). *OSCI TLM-2.0 language reference manual*. Open SystemC Initiative, ja32 edition.

Azevedo, R., Rigo, S., Bartholomeu, M., Araujo, G., Araujo, C., and Barros, E. (2005). The ArchC Architecture Description Language and Tools. *International Journal of Parallel Programming*, 33(5):453–484.

Carlson, T., Heirman, W., and Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation.

Chopard, B., Combes, P., and Zory, J. (2006). A Conservative Approach to SystemC Parallelization. In *Proceedings of the Workshop on Scientific Computing in Electronics Engineering*, pages 653–660.

Duenha, L., Azevedo, R., and de Azevedo, R. J. (2012). Profiling High Level Abstraction Simulators of Multiprocessor Systems. In *Proceedings of the Second Workshop on Circuits and Systems Design - WCAS 2012*.

Duenha, L., Guedes, M., Almeida, H., Boy, M., and Azevedo, R. (2014). MPSoC-Bench: A toolset for MPSoC system level evaluation. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 164–171. IEEE.

Ezudheen, P., Chandran, P., Chandra, J., Simon, B., and Ravi, D. (2009). Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machines. In *2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 80–87. IEEE.

Fu, Y. and Wentzlaff, D. (2014). PriME: A parallel and distributed simulator for thousand-core chips. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 116–125. IEEE.

Huang, K., Bacivarov, I., Hugelshofer, F., and Thiele, L. (2008). Scalably Distributed SystemC Simulation for Embedded Applications. In *Proceedings of the International Symposium on Industrial Embedded System (SIES 2008)*, pages 271–274.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 40 of *PLDI '05*, pages 190–200, New York, NY, USA. ACM.

Mack, C. A. (2011). Fifty Years of Moore's Law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202–207.

Maia, I., Greiner, A., and Pecheux, F. (2006). SystemC SMP: A parallel approach to speed up Timed TLM simulation. In *SoC-SIP-System on Chip-System in Package*.

Mello, A., Maia, I., Greiner, A., Pecheux, F., aind A. Greiner, I. M., and Pecheux, F. (2010). Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations. In *Proceedings of Design, Automation and Test in Europe - DATE*, pages 606–609. IEEE.

Miller, J. E., Kasture, H., Kurian, G., Gruenwald, C., Beckmann, N., Celio, C., Eastep, J., and Agarwal, A. (2010). Graphite: A distributed parallel simulator for multicores. *High Performance Computer Architecture, 2010. HPCA 16. The 16th International Symposium on*, pages 1–12.

Moore, B. G. E. (1975). Cramming more components onto integrated circuits. *Solid-State Circuits Newsletter, IEEE*, 38(8):33–35.

Peeters, J., Ventroux, N., Sassolas, T., and Lacassagne, L. (2010). A SystemC TLM Framework for Distributed Simulation of Complex Systems with Unpredictable Communication. In *Proceedings of Design and Architecture for Signed and Image Precessing - DASIP-2010*.

Rigo, S., Azevedo, R., and Santos, L. (2011). *Electronic System Level Design: An Open-Source Approach*. Springer Netherlands.

Roth, C., Bucher, H., Reder, S., Buciuman, F., Sander, O., and Becker, J. (2013). A SystemC modeling and simulation methodology for fast and accurate parallel MPSoC simulation. In *Proceedings of the 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6.

Sanchez, D. and Kozyrakis, C. (2013). ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, volume 41 of *ISCA '13*, pages 475–486, New York, NY, USA. ACM.

Sinha, R., Prakash, A., and Patel, H. D. (2012). Parallel simulation of mixed-abstraction SystemC models on GPUs and multicore CPUs. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 455–460.

Vinco, S., Chatterjee, D., Bertacco, V., and Fummi, F. (2012). SAGA: SystemC Acceleration on GPU Architectures. In *Proceedings of Design Automation Conference (ASP-DAC 2012)*, pages 115–120.

Ziyu, H., Lei, Q., Hongliang, L., Xianghui, X., and Kun, Z. (2009). A Parallel SystemC Environment: ArchSC. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 617–623. IEEE.