Projeto de um Processador de Rede *Intra-Chip* para Controle de Comunicação entre Múltiplos *Cores*

Henrique C. Freitas[1], Flávio R. Wagner[2], Philippe O. A. Navaux[3]
Grupo de Processamento Paralelo e Distribuído[1, 3]
Laboratório de Sistemas Embarcados[2]
Programa de Pós-Graduação em Computação, Instituto de Informática
Universidade Federal do Rio Grande do Sul
{hcfreitas, flavio, navaux}@inf.ufrgs.br

Carlos Augusto P. S. Martins[4]
Grupo de Sistemas Digitais e Computacionais[1, 4]
Programa de Pós-Graduação em Engenharia Elétrica, Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais
capsm@pucminas.br

Resumo

O projeto de processadores com arquiteturas multi-core é a atual alternativa para o aumento de desempenho demandado pelo grande volume de processamento de informação. Estes projetos levam à necessidade da definição de mecanismos eficientes de comunicação entre os núcleos do chip. Neste artigo é apresentado o projeto de um processador de rede intra-chip responsável por gerenciar e controlar a comunicação entre os múltiplos núcleos do chip. Para análise do desempenho do processador de rede foi utilizada a linguagem ArchC, que permitiu a simulação do comportamento da arquitetura com precisão de ciclos. Na conclusão verificou-se, através dos experimentos, que a nova arquitetura apresenta um ganho de desempenho em relação aos processadores R2NP e IXP1200, devido à organização interna e do controle dos periféricos responsáveis por estabelecer a comunicação.

1. Introdução

A diminuição do tempo de resposta das aplicações em computadores pessoais ou em servidores é um dos objetivos quando se estuda melhorar o desempenho destes sistemas. Vários projetos de arquiteturas de processadores utilizam técnicas [5, 10, 15, 20] como *pipeline*, superescalaridade e *multithreading* para exploração de paralelismo como alternativas para melhorar o desempenho.

Um processador tradicional executa uma *thread*, ou seja, um fluxo de instruções. Processadores com suporte a *pipeline* de instruções, superescalaridade e múltiplas *threads* simultâneas [11] possuem mais de um caminho para o fluxo de instruções. Portanto, um único processador com dois fluxos de instruções é um processador com dois núcleos lógicos de processamento.

Algumas soluções exploram o paralelismo entre as *threads* através de projetos de processadores com vários núcleos físicos de processamento relativamente simples [13, 17]. Neste caso, em cada núcleo é aplicada a técnica de *pipeline* de instruções, não há superescalaridade e as *threads* simultâneas são obtidas através das execuções de cada uma das *threads* em seus respectivos núcleos de processamento.

Arquiteturas *multi-core* requerem um alto desempenho na comunicação interna. São vários núcleos em paralelo processando *threads* e cada um destes núcleos podendo, ou não, suportar *threads* simultâneas [20]. A necessidade de comunicação entre *threads* é alta e deve ser suportada por um sistema de comunicação eficiente, de baixa latência, alta vazão de dados e que privilegie o paralelismo.

Normalmente as soluções para interconectar [16] cada um dos núcleos internos ao *chip* são os barramentos e as chaves *crossbar* através de memória compartilhada. Alternativas para os processadores com

um grande número de núcleos são as *Networks-on-Chip* (NoCs) [2, 14]. As NoCs são sistemas de comunicação mais complexos, já que possuem uma rede de interconexão e roteadores responsáveis pelos caminhos que cada pacote deve seguir entre um núcleo e outro. Estes roteadores são blocos de *hardwares* dedicados constituídos de *buffers* para alocação dos pacotes e devem, além de definir a rota, controlar o fluxo de pacotes recebidos e enviados.

O principal problema na comunicação entre os diversos núcleos está em encontrar o melhor sistema de comunicação associado à melhor técnica para roteamento dos dados, controle de fluxo, redução da latência e aumento da vazão de dados.

O objetivo deste artigo está em apresentar o projeto de um processador de rede *intra-chip* capaz de gerenciar e controlar a comunicação entre os múltiplos núcleos da arquitetura em conjunto com a *Reconfigurable Crossbar Switch* (RCS) apresentada no WSCAD 2005 [8].

A principal contribuição está na organização interna e no conjunto de instruções de rede que permite um menor tempo de residência dos pacotes no sistema de comunicação *intra-chip*.

Para obtenção dos resultados foram utilizadas linguagens de descrição de arquitetura (ArchC) [18] e de sistema (SystemC) [1].

Nas seções seguintes serão apresentados trabalhos correlatos, proposta, resultados e conclusões.

2. Trabalhos Correlatos

Nesta seção são apresentados trabalhos correlatos que abordam propostas de sistemas de comunicação *on-chip* demandadas pelas arquiteturas *multi-core*.

Em [4] é proposta uma chave *crossbar* com largura de banda dos barramentos reconfigurável para aplicação em redes de interconexão *on-chip*. O principal objetivo é aumentar a vazão de dados associados aos vários pacotes. Em função da demanda de largura de banda de cada pacote os barramentos da chave *crossbar* são reconfigurados.

O projeto ParIS [2] apresenta uma arquitetura de roteador para sistemas *on-chip* através de uma rede de interconexão parametrizável. Este projeto é parte do SoCIN (*System-on-Chip Interconnection Network*) que desenvolve arquiteturas para interconexão de *cores* em um SoC. O ParIS é uma extensão do projeto RASoC (*Router Architecture for SoC*). Os principais componentes da arquitetura do ParIS são: controladores de link (tráfego), *buffers*, controladores de escalonamento, *switches* (unidades de chaveamento) e uma matriz *crosspoint*. Por se tratar de uma arquitetura parametrizável, que traz como conseqüência flexibilidade de reconfiguração, os resultados que são tratados com uma maior ênfase são referentes aos diferentes tamanhos da implementação do ParIS e a relação custo/desempenho.

Em [19] é apresentada uma rede de interconexão *on-chip* escalável e que pode se adaptar às necessidades de cada aplicação ou padrão de comunicação. O projeto apresenta um circuito de roteamento parametrizável para o número de entradas e saídas e para diminuir a latência é utilizada a técnica *cut-and-through*. Entre os resultados obtidos é possível ressaltar: tamanho da rede implementada no FPGA (*Field Programmable Gate Array*) e largura de banda. Cada um destes resultados foi obtido em função de quatro tipos de redes implementadas: hipercubo, árvore binária balanceada, *torus* e *crossbar*. Segundo os autores, a reconfiguração é feita através da implementação de uma nova topologia no FPGA.

As soluções da Intel [11] apresentam barramentos para interconexão interna. O processador Pentium Extreme Edition possui dois núcleos físicos que suportam *hyperthreading* (*Simultaneous Multithreading*) [20]. O suporte *hyperthreading* provê dois caminhos para os fluxos de instruções em cada núcleo. Portanto, logicamente este processador possui quatro núcleos. Os processadores Pentium D e Core Duo, são compostos por dois núcleos, mas sem suporte a *hyperthreading*. Permanece o suporte a múltiplas *threads*, mas não no mesmo ciclo (simultaneamente). Portanto, fisicamente e logicamente são dois núcleos internos interconectados por barramentos.

As propostas da Sun Microsystems [17], entretanto, estão voltadas para grandes servidores e cada núcleo corresponde a um processador físico relativamente simples. A arquitetura do UltraSPARC-T1 possui oito núcleos físicos e, portanto, oito núcleos lógicos. Uma chave *crossbar* é utilizada como meio para interconexão entre os núcleos, memórias *caches* L2 e demais periféricos.

Processadores de rede [3, 6], como o Intel IXP1200 [12], são projetados com múltiplos núcleos heterogêneos. Em função do contexto específico, cada núcleo é responsável por uma parte do processamento de rede.

O IXP1200 não foi projetado para sistemas de comunicação *on-chip,* mas possui sete núcleos internos, sendo seis *microengines* e um StrongARM. O StrongARM é responsável pelo gerenciamento e processamento de tarefas mais complexas do IXP e as *microengines* de tarefas específicas de rede, tais como o roteamento.

A comunicação interna e externa é feita através de barramentos dedicados para acessos aos registradores

de transferência, às memórias compartilhadas e para acessos aos periféricos.

Na Seção 3 apresentamos a proposta de um processador de rede *intra-chip* para gerenciamento e controle de comunicação em arquiteturas *multi-core*. Esta proposta, em conjunto com a RCS [8, 9], compõe um sistema de comunicação programável, com maior flexibilidade, capaz de reduzir a latência de comunicação para arquiteturas com um grande número de núcleos.

3. Processador de Rede *Intra-Chip* Proposto

A arquitetura apresentada nesta seção é uma evolução do processador de rede R2NP (*Reconfigurable RISC Network Processor*) [7]. O que permanece é a idéia de controle da chave *crossbar* reconfigurável e dos *buffers* de entrada de pacotes. Portanto, a arquitetura do conjunto de instruções e organização interna dos blocos construtivos fazem parte de um novo projeto para o contexto *intra-chip*.

A Figura 1 ilustra o processador de rede proposto (NPoC: *Network Processor on Chip*) em uma arquitetura *multi-core*. O controle e gerenciamento da comunicação entre os diversos núcleos são feitos através da chave *crossbar* reconfigurável (RCS – *Reconfigurable Crossbar Switch*), que provê a interconexão, e dos *buffers* de entrada dos pacotes.

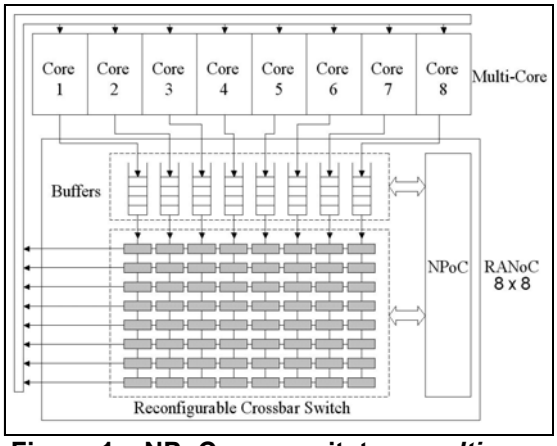

Figura 1 – NPoC em arquitetura *multi-core*

A Figura 2 apresenta a arquitetura do processador R2NP. Alguns blocos construtivos desta arquitetura como os *buffers* reconfiguráveis, os multiplexadores, as *microengines* e a unidade de reconfiguração não permanecem no projeto do NPoC. A primeira mudança é reduzir o tamanho e a complexidade da arquitetura em função do contexto *intra-chip*. No entanto, para estabelecer a comunicação entre os núcleos, são necessários a RCS e *buffers* de entrada em conjunto com o NPoC. A este conjunto foi dado o nome de RANoC (*Router Architecture for Network-on-Chip*), conforme ilustrado pela Figura 1.

A idéia presente na ilustração da Figura 1 é que, para um número muito grande de núcleos em um *chip*, um processador de rede se faz necessário para controlar e gerenciar a comunicação interna. Desta forma, é possível aumentar a flexibilidade do roteador de uma NoC, que conforme descrito na Seção 1, é composto basicamente por *hardwares* dedicados e não programáveis. A utilização de múltiplos RANoCs em cascata traz para dentro do *chip* uma estrutura com roteadores para múltiplos núcleos, ao contrário de uma NoC tradicional onde cada roteador é específico para apenas um núcleo.

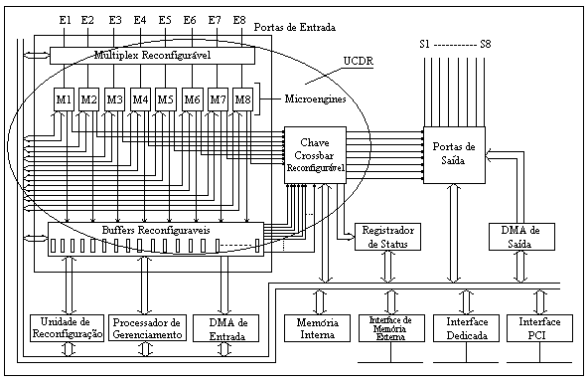

Figura 2 – Arquitetura do R2NP [10]

3.1. Arquitetura do NPoC

O projeto da arquitetura do NPoC segue o modelo tradicional de *pipeline* de cinco estágios. No entanto, em função da arquitetura dedicada do processador, o quarto estágio possui acesso a periféricos diferentes, conforme ilustrado pela Figura 3.

No quarto estágio são realizados acessos à memória, chave *crossbar* e *buffers*, conforme descrição a seguir:

- MEM: acesso à memória pelas instruções do tipo *load* e *store*.
- BCTU: acesso aos *buffers* e à chave *crossbar* através das instruções de rede.
- SCH: acesso ao escalonador que controla o tráfego de pacotes dos *buffers* através de instruções de rede.
- REC: acesso ao registrador de reconfiguração da chave *crossbar*.

Os demais estágios seguem o modelo tradicional de *pipeline*:

- IF: Busca de instrução
- ID: Decodificação de instrução
- EX: Execução de instrução
- WB: Escrita em banco de registradores

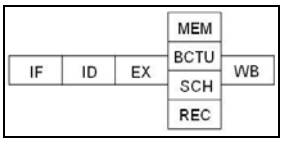

Figura 3 – Estágios de *pipeline* do NPoC

A Figura 4 apresenta a organização interna dos blocos construtivos do NPoC. Através desta figura é possível descrever o caminho de dados de todas as instruções projetadas com alguns sinais de controle mais significativos como J (*jump* – desvio) e de controle dos MUXs de entrada da ALU (*Arithmetic and Logic Unit*). A descrição em linguagem de arquitetura ArchC foi realizada tendo como referência a organização da Figura 4.

O NPoC é um processador RISC (*Reduced Instruction Set Computing*) [10]. Portanto, tem instruções de formato regular e fixo e com execução em apenas 1 ciclo quando o *pipeline* está cheio.

No entanto, existem dependências de dados e de controle que podem impossibilitar que o *pipeline* fique cheio todo o tempo. A alternativa para reduzir o efeito destas dependências foi o adiantamento de dados capaz de aumentar o número de vezes que o *pipeline* permanece cheio. No caso das instruções de desvio, quando de um desvio verdadeiro, as instruções presentes no *pipeline* são descartadas ocasionando bolhas. A previsão de desvio não está implementada e faz parte dos trabalhos futuros.

O NPoC possui duas classes de instruções: uma de propósito geral e outra específica de rede. As instruções de propósito geral são referentes ao acesso à memória, aos desvios e à unidade lógica e aritmética.

A Tabela 1 ilustra algumas instruções de propósito geral. Existem outras instruções que não foram apresentadas e que seguem o mesmo tipo de descrição tais como: *sub, and, or, nor, nand, div, xori, addi, subi, ori, andi, jdi, jle, shiftr* e *shiftl*, entre outras.

A Tabela 2 apresenta instruções de rede com formatos iguais às de propósito geral, porém com acessos a periféricos diferentes. Estas instruções utilizam a ALU apenas para cálculo de endereço conforme será descrito a seguir, mas o propósito principal é o acesso aos periféricos *buffers* e chave *crossbar*. As instruções *read* e *write* acessam os *buffers* e registradores da chave *crossbar* através da unidade de transferência (BCTU: *Buffers and Crossbar Transfer Unit*). As instruções *send, block* e *erase* especificam para o escalonador quais os pacotes devem ser enviados, bloqueados ou apagados, respectivamente. No entanto, o escalonador não depende destas instruções para escalonar. A instrução *reconf* é usada para transferência de dados entre um registrador de propósito geral e o registrador de reconfiguração da chave *crossbar*. Este registrador de reconfiguração repassa o conteúdo ao decodificador da RCS que implementa a topologia necessária.

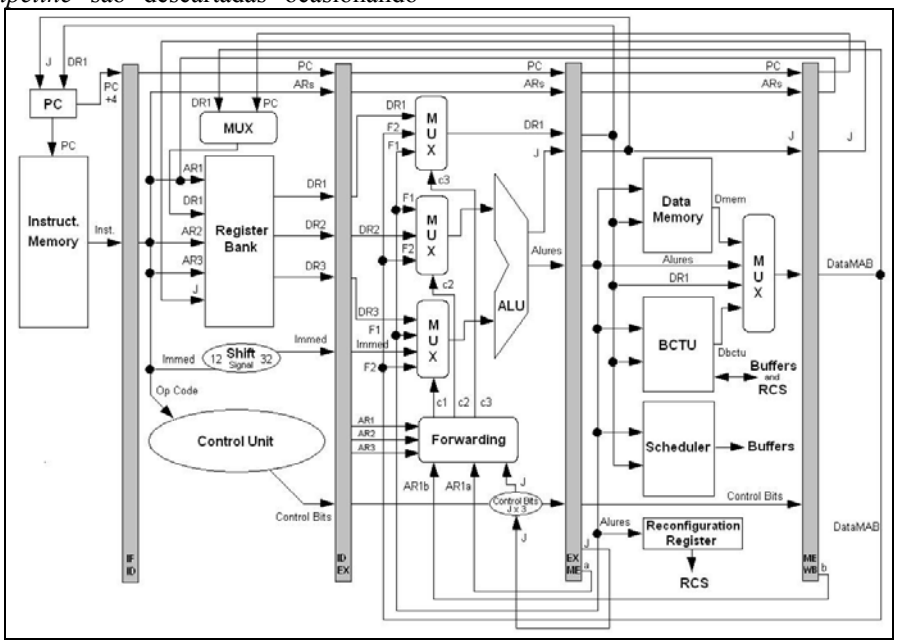

Figura 4 – Organização do NPoC

Tabela 1 – Algumas instruções de propósito geral

Instrução	Descrição
add r1, r2, r3	r1 = r2 + r3
mul r1, r2, r3	r1 = r2 x r3
ori r1, r2, imed	r1 = r2 + imed
not r1, r2	r1 = r2'
load r1, r2, imed	r1 = conteúdo(End[r2 + imed])
store r1, r2, imed	End[r2 + imed] = r1
jump r1, r2, imed	PC = r2 + imed, r1 = PC
jeq r1, r2, r3	PC = r1, se r2 = r3

Tabela 2 – Instruções de rede

Instrução	Descrição
read r1, r2, imed	r1 = conteúdo(End[r2 + imed])
write r1, r2, imed	End[r2 + imed] = r1
send r1, r2, imed	Buffer (r1), pacote (r2 + imed)
block r1, r2, imed	Buffer (r1), pacote (r2 + imed)
erase r1, r2, imed	Buffer (r1), pacote (r2 + imed)
reconf r1, r2	Reg. reconfiguração = r2

O formato das instruções é especificado pela figura 5. A palavra do NPoC é de 32 bits e o campo de *opcode* corresponde a 6 bits para todas as instruções. Os últimos campos de bits com um número seguido por x não são utilizados pelas instruções. Sendo assim, instruções com três operandos registradores não utilizam os cinco últimos bits da palavra.

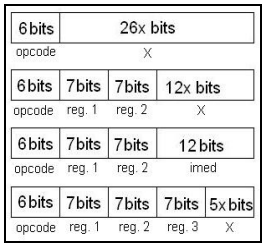

Figura 5 – Formato das instruções

Instruções que possuem um operando com valor imediato utilizam o bloco *shift signal* da arquitetura apresentada pela Figura 4, que é responsável por deslocar o sinal até o trigésimo segundo bit da palavra para que possa ser utilizada pela ALU.

A descrição da arquitetura do conjunto de instruções e organização interna com precisão de ciclos do NPoC foi descrita em ArchC. A Seção 3.2 faz uma breve abordagem.

3.2. Descrição em ArchC

ArchC [18], além de uma linguagem de descrição de arquitetura de processadores, é um ambiente capaz de criar simuladores das arquiteturas descritas. Estes simuladores criados automaticamente, com base na linguagem SystemC [1], são capazes de ler um código executável, ou de montagem, realizar a simulação e fornecer resultados de desempenho.

A descrição em ArchC requer três arquivos, sendo cada um responsável por:

- Descrição da arquitetura geral, com quantidade de registradores, quais são os periféricos, estágios de *pipeline*, etc.
- Descrição do formato das instruções.
- Descrição do comportamento com precisão de ciclos de cada instrução.

O arquivo da descrição do comportamento é o mais complexo e possui todas as especificações do comportamento de cada instrução em cada estágio de *pipeline*. Neste arquivo é descrito o adiantamento de dados, conforme exemplo ilustrado pela Figura 6.

```
//forwarding for j (instruction branch)
if ( C_value == 0 ){
  //forwarding for r2
  if ( (EX_MEM.regwrite == 1) && (EX_MEM.r1 != 0 ) &&
(EX_MEM.r1 == ID_EX.r2) )
    ID_EX.data2 = EX_MEM.alures.read();
  else if ( (MEM_WB.regwrite == 1) && (MEM_WB.r1 != 0 ) &&
(MEM_WB.r1 == ID_EX.r2) )
    ID_EX.data2 = MEM_WB.datamab.read();
  else
    ID_EX.data2 = RB.read(r2); }
else {
   ID_EX.data2 = RB.read(r2);
   ID_EX.regwrite = 0; }
```

Figura 6 – Exemplo de adiantamento de dados

O exemplo da Figura 6 pode ser expandido para todas as instruções, ressaltando que para algumas são necessários o adiantamento para os registradores r1 e r3, além do r2.

Os resultados apresentados na Seção 4 são referentes à verificação do funcionamento da descrição, da comparação com o R2NP e com o IXP1200.

4. Resultados

Os resultados procuram verificar o funcionamento do conjunto de instruções e da organização descrita. Sendo assim foi escolhido um programa que realiza cálculo de fatorial. Este programa foi escrito sem e com recursão para que seja possível realizar uma comparação de desempenho e utilização de um maior número de tipos de instruções.

Na versão sem recursão são utilizadas apenas instruções aritméticas e de desvio. Na versão com recursão são necessárias instruções de *load* e *store* para armazenamento de resultados em pilha (memória) e a instrução de desvio *jump* que guarda o valor de retorno

no registrador r1, além da instrução de desvio condicional que também é usada na versão sem recursão.

Durante as simulações foi possível corrigir erros na descrição em ArchC, relativos ao descarte de instruções e adiantamento de dados, por exemplo.

A Figura 7 apresenta o trecho inicial da simulação do cálculo do fatorial de 4 sem recursão. Vale ressaltar que no primeiro ciclo é feita a leitura de uma instrução, no segundo ciclo esta instrução está no estágio de decodificação e uma nova instrução no estágio de leitura. Este processo se repete até o quinto ciclo, onde o *pipeline* se encontra cheio e a primeira instrução escreve no banco de registradores seu resultado.

```
----- PC=0 ----- 0
addi r4, r0, 4
addi r4, r0, 4
----- PC=0x4 ----- 1
addi r2, r0, 1
addi r4, r0, 4
addi r2, r0, 1
----- PC=0x8 ----- 2
addi r3, r0, 12
addi r4, r0, 4
addi r2, r0, 1
addi r3, r0, 12
----- PC=0xc ----- 3
mul r2, r2, r4
addi r4, r0, 4
Result Reg. = 0x4
addi r2, r0, 1
addi r3, r0, 12
mul r2, r2, r4
----- PC=0x10 ----- 4
addi r4, r4, 65535
```

Figura 7 – Trecho da simulação do fatorial

Através da simulação em ArchC foi possível gerar estatísticas referentes aos acessos à memória e ao banco de registradores apresentados pela Figura 8.

Como a primeira versão não tem recursão, não é necessário guardar dados de cálculos intermediários e endereços de retorno em pilha. O acesso à memória é feito apenas durante a leitura de instrução. Sendo assim, este valor permanece constante nos três cálculos de fatorial (20 acessos).

Os demais valores crescem à medida que o número usado para cálculo também cresce. Para a versão com recursão o acesso ao banco de registradores aumenta bastante em função de um maior número de instruções aritméticas e de *load* e *store* que também fazem o acesso.

A outra estatística gerada é referente ao tempo de simulação e ao número de instruções executadas. Em função das características dos dois programas, os números da versão com recursão se apresentaram bem maiores do que na versão sem recursão, conforme apresentado pela Figura 9.

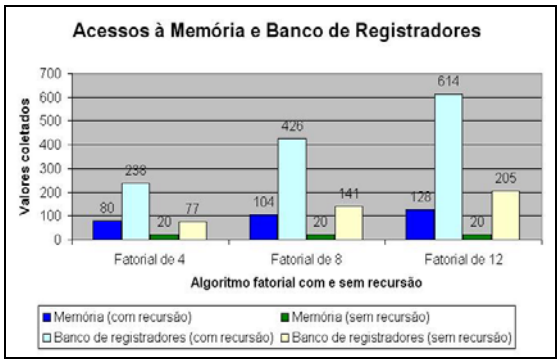

Figura 8 – Acessos à memória e registradores

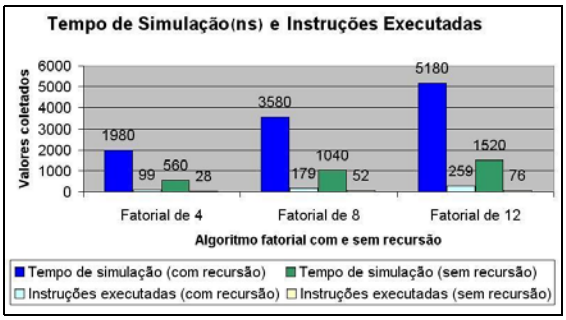

Figura 9 – Simulação / instruções executadas

Estas simulações foram importantes para a verificação do funcionamento das instruções e da descrição com precisão de ciclos em ArchC.

A Figura 10 apresenta a comparação entre o uso de instruções de rede do NPoC e do R2NP. Alguns resultados do R2NP foram extraídos de trabalho anterior [7], onde o R2NP foi submetido a três programas com as seguintes características:

- Implementar a topologia anel unidirecional para comunicação entre oito computadores através da chave *crossbar* reconfigurável.
- Estabelecer a comunicação em uma topologia hipercubo bidirecional de grau 3, através de um algoritmo de roteamento e acessos à chave *crossbar* reconfigurável.
- Estabelecer a comunicação em uma topologia árvore binária bidirecional através de um algoritmo de roteamento e acessos à chave *crossbar* reconfigurável.

Nestes três casos o R2NP tinha que acessar a RCS e implementar uma topologia ou estabelecer a melhor rota entre os nós de origem ou destino.

A diferença entre os programas escritos para o R2NP e para o NPoC está na quantidade das instruções de rede necessárias. O R2NP não possuía uma instrução capaz de reconfigurar a RCS de uma vez, para isto era usada uma unidade de reconfiguração dedicada. Uma única instrução do R2NP reconfigurava apenas um único nó.

O novo projeto da RCS [8, 9] contém um decodificador de palavra de 32 bits para bits de reconfiguração e, por conseqüência, para implementação de uma nova topologia. O NPoC possui a instrução *reconf* capaz de entregar ao registrador de reconfiguração da RCS a palavra de 32 bits. Sendo assim, uma única instrução do NPoC pode reconfigurar toda a RCS, ou uma linha, uma coluna, um sub-conjunto de nós ou apenas um único nó.

A Figura 10 apresenta a quantidade de instruções de rede usadas pelos dois processadores. Para o NPoC, o caso 1 é referente ao uso de instruções para leitura e escrita em *buffers*, conforme demandam os algoritmos de roteamento citados para o hipercubo e árvore. O caso 2 é referente ao uso de uma instrução para implementação da topologia.

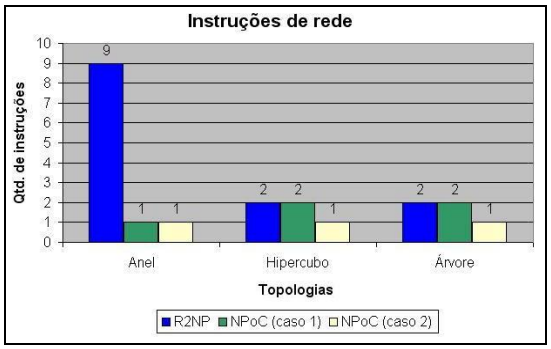

Figura 10 – Acessos a RCS

Para a topologia anel, o R2NP usou 9 instruções para implementar a topologia na RCS. O NPoC precisa de apenas uma instrução para implementá-la.

Para o hipercubo e árvore, os algoritmos fazem leitura dos *buffers* e solicitação de envio de pacotes. Tanto o R2NP e o NPoC (caso 1) podem fazer a mesma tarefa usando a mesma quantidade de instruções. No entanto, o NPoC pode usar apenas uma instrução para acessar o registrador de reconfiguração (caso 2), implementar a topologia e deixar que o escalonador e a RCS encaminhem o pacote. Isto é possível porque além do decodificador a RCS possui também um analisador de cabeçalho que permite a passagem ou não de um pacote por um nó implementado para a topologia.

A grande vantagem do NPoC sobre o R2NP está no projeto das instruções de rede e dos periféricos. À medida que a demanda por comunicação e implementação de uma nova topologia aumenta, o ganho do NPoC em relação ao R2NP, que pela Figura 10 chegou ao máximo de 9 vezes, pode se tornar ainda maior.

Usando instruções de rede só para implementação de topologias é possível verificar o ganho do NPoC em relação ao R2NP. A Figura 11 apresenta a quantidade mínima de instruções necessárias para as três topologias, onde para hipercubo bidirecional há um ganho de 24 vezes favorável ao NPoC.

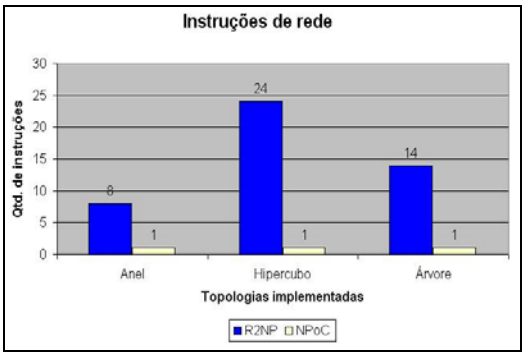

Figura 11 – Implementação de topologias

As *microengines* do Intel IXP1200, apresentado na Seção 2, utilizam barramentos para comunicação interna e externa e possuem um conjunto de instruções para gerência dos *buffers* de entrada e saída referentes às interfaces de comunicação e da memória usada para alocação de pacotes. Apesar do IXP1200 não ser um processador de rede para gerência de comunicação *intra-chip*, é possível estabelecer uma comparação (Figura 12) entre a quantidade de tarefas necessárias para uma transferência de pacotes se uma versão *intra-chip* baseada neste processador fosse projetada. O IXP1200 necessita de um número mínimo de tarefas:

- Buscar o pacote do *buffer* de entrada.
- Alocar o pacote em memória ou registradores de transferência.
- Processar e/ou descobrir destino do pacote.
- Enviar o pacote para o *buffer* de saída.

Estas tarefas devem ser executadas durante o recebimento e envio de todos pacotes.

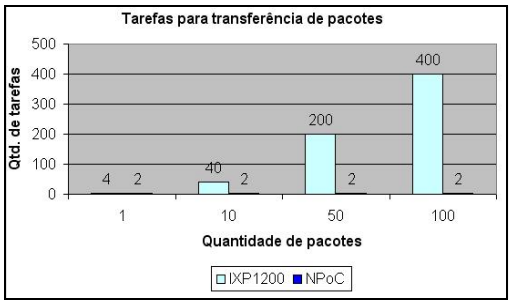

Figura 12 – Transferência de pacotes

O número mínimo de tarefas do NPoC:

- Definir o padrão de comunicação.
- Implementar a topologia na chave *crossbar* reconfigurável.

Estas tarefas são executadas uma única vez para todos os pacotes.

A Figura 12 apresenta uma comparação da quantidade de tarefas necessárias para a transferência de pacotes. A definição do padrão de comunicação, entre todas as tarefas, é a que pode levar um maior tempo. No entanto, uma vez estabelecido o padrão e implementada a topologia, todos os pacotes são encaminhados através da RCS. Este padrão de comunicação pode ser definido antes das transferências de pacotes (estaticamente) ou dinamicamente em função de mudanças neste padrão.

5. Conclusões

Neste artigo é apresentado um processador de rede *intra-chip* como parte integrante de um sistema de comunicação, capaz de gerenciar e controlar a comunicação entre múltiplos *cores* de um *chip*, através da programação e implementação de novas topologias na *Reconfigurable Crossbar Switch* (RCS).

Os resultados da simulação em ArchC verificaram o comportamento das instruções descritas e foram importantes para correções ou melhorias do projeto.

O ganho de desempenho favorável ao NPoC, em relação ao R2NP e ao IXP1200, é resultado de um projeto de instruções de rede que busca um menor tempo de execução pelo processador, permitindo que o tráfego de pacotes possam ocorrer com um menor tempo de residência no sistema de comunicação.

A contribuição deste trabalho está na arquitetura e organização interna dos blocos construtivos do NPoC como uma alternativa, para que em conjunto com a RCS, possa integrar uma *network-on-chip*, onde cada roteador possa gerenciar múltiplos núcleos.

Os trabalhos futuros são: o projeto da previsão de desvio, técnicas de compilação, integração do NPoC com a RCS e *buffers* para proposta de uma arquitetura de *network-on-chip*, avaliação da latência de comunicação, área ocupada consumo de energia e comparação com outras NoCs e sistemas de comunicação para arquiteturas *multi-core*.

Referências

[1] A. Ghosh, et al., *System modeling with SystemC*, International Conference on ASIC, pp.18-20, 2001

[2] C. A. Zeferino, et al., *ParIS: A Parameterizable Interconnected Switch of Networks-on-Chip*, Symposium on Integrated Circuits and Systems Design, 2004

[3] Comer, D. E., *Network Systems Design Using Network Processors*, Prentice Hall, 2003

[4] D. Kim, et al., *A Reconfigurable Crossbar Switch with Adaptive Bandwidth Control for Networks-on-Chip*, IEEE International Symposium on Circuits and Systems, pp. 2369 – 2372, 2005

[5] G. Dal Pizzol, et al., *Branch prediction topologies for SMT architectures*, International Symposium on Computer Architecture and High Performance Processing (SBAC-PAD), pp.118-125, October 2005

[6] G. Lawton, *Will Network Processor Units Live up to Their Promise?*, IEEE Computer, Volume 37, Number 4, pp.13-15, April, 2004

[7] H. C. Freitas, C. A. P. S. Martins, *R2NP: Processador de Rede RISC Reconfigurável*, III Workshop em Sistemas Computacionais de Alto Desempenho, Vitória ES, Brasil, pp. 60-67, 2002

[8] H. C. Freitas, et al., *RCS-2: Projeto de uma Chave Crossbar Reconfigurável,* VI Workshop em Sistemas Computacionais de Alto Desempenho, Rio de Janeiro RJ, Brasil, 2005

[9] H. C. Freitas, et al., *Reconfigurable Crossbar Switch Architecture for Network Processors*, IEEE International Symposium on Circuits and Systems, pp.4042-4045, May 2006

[10] Hennessy, J. L., D. A. Patterson, *Arquitetura de Computadores Uma Abordagem Quantitativa*, Editora Campus, 3a edição, 2003

[11] Intel, *IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, March 2006

[12] Intel, *IXP1200 Network Processor Family, Hardware Reference Manual*, December, 2001

[13] K. Olukotun, et al., *The Case for a Single-Chip Multiprocessor*, 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.2-11, 1996

[14] L. Benini, G. D. Micheli, *Network-on-chip architectures and design methods*, IEE Proceedings Computers & Digital Techniques, Vol. 152, Issue 2, pp.261-272, 2005

[15] L. Spracklen, S.G. Abraham, *Chip Multithreading: Opportunities and Challenges*, International Symposium on High-Performance Computer Architecture (HPCA), pp.248-252, February 2005

[16] R. Kumar, V. Zyuban, D.M. Tullsen, *Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling*, 32nd International Symposium on Computer Architecture, pp.408-419, June 2005

[17] S. Chaudhry, et al., *High-performance throughput computing*, IEEE MICRO, Vol. 25, Issue 3, pp.32-45, May-June 2005

[18] S. Rigo, et al., *ArchC: A SystemC-Based Architecture Description Language*, International Symposium on Computer Architecture and High Performance Processing, pp.66-73, October 2004

[19] T. A. Bartic, et al., *Topology adaptive network-on-chip design and implementation*, IEE Proc. Comput. Digit. Tech., Vol. 152, No. 4, July 2005

[20] T. Ungerer, et al., *A Survey of Processors with Explicit Multithreading*, ACM Computing Surveys, Volume 35, Issue 1, pp.29-63, March 2003