IPNoSys: uma nova arquitetura paralela baseada em redes em chip

Sílvio R. Fernandes
UFRN
silvio@lasic.ufrn.br

Bruno C. Oliveira
UFRN
bcruz@lasic.ufrn.br

Miklécio Costa
UFRN
miklecio@lasic.ufrn.br

Ivan S. Silva
UFRN
ivan@dimap.ufrn.br

Resumo

A tecnologia de integração tem avançado a ponto de transformar os processadores multi-core em uma realidade de mercado nos dias atuais. Nesse cenário, as redes de interconexão têm uma função essencial quando o número de núcleos de processamento cresce, uma vez que o uso de soluções baseadas em barramento torna-se impossível. Algumas soluções de interconexão têm sido empregadas, entretanto, são custosas em relação à área e potência dissipada. Este artigo apresenta uma solução utilizando redes em chip, não apenas como interconexão, mas também como sistema de processamento. Simulações mostraram que o tempo de execução na arquitetura proposta é 3,5 vezes menor que a mesma aplicação executada em uma plataforma virtual MP-SoC.

1. Introdução

A tecnologia de integração tem transformado a microeletrônica em nanoeletrônica, e nessa na era dos bilhões de transistores não só é possível encolher os *chips,* mas principalmente, desenvolver sistemas maiores e mais complexos dentro de um único chip. Esse foi um passo decisivo para a consolidação dos sistemas *multi-cores*, provavelmente a futura geração dos computadores [1], [2]. Já é uma realidade no mercado os processadores *dual*, *quad*, *eighth-core* e até mais que isso [3], [4]. Aliados a maior capacidade de integração estão os projetos de redes em chip como subsistemas de interconexão.

As redes em chip, ou NoC (*Network-on-Chip*), se tornaram um subsistema de interconexão que ao mesmo tempo representa uma solução e um problema. Como solução as NoCs apresentam alta escalabilidade, comunicação paralela, estrutura reusável e interconexão ponto-a-ponto. Como problema, elas são associadas ao aumento de área em chip e da potência dissipada pelo sistema, apesar de serem usadas técnicas para redução do consumo de energia.

Um projeto tradicional baseado em NoC direta consiste em um conjunto de roteadores interconectados entre si e a núcleos do sistema (*core*). As NoCs são responsáveis apenas pela transmissão dos pacotes com requisições e respostas entre os núcleos que realizam o processamento da aplicação.

Assim, esse artigo apresenta uma proposta de um novo modelo de arquitetura baseada em NoC, o sistema IPNoSys. Nesse sistema os roteadores, além de rotear os pacotes, também são responsáveis pela execução das instruções que formam as aplicações.

O artigo segue com a Seção 2 apresentando os trabalhos relacionados bem como os conceitos gerais sobre NoC e máquinas em fila, que são a base teórica para o sistema IPNoSys. A Seção 3 apresenta a arquitetura IPNoSys e seus principais conceitos. A Seção 4 apresenta os cenários e os resultados de simulação. As conclusões e trabalhos futuros são apresentados na Seção 5.

2. Trabalhos relacionados

O tema redes em chip (NoC) é muito recorrente na literatura, onde se tem apresentado inúmeras contribuições técnico-científicas. No entanto, tais contribuições estão relacionadas apenas a subsistemas de interconexão, os projetos de NoC não incorporam capacidade de processamento do ponto de vista de execução de aplicações durante o roteamento.

Os mecanismos de interconexão são fundamentais para o desempenho e escalabilidade dos sistemas ao qual fazem parte, principalmente no ambiente *multi-core*. Os barramentos ainda estão sendo utilizados como sistema de interconexão enquanto a quantidade de *cores* é limitada, já que esta é uma solução antiga, com baixíssima escalabilidade e constantemente identificada como o elemento central do chamado "Gargalo de Von Neumann" [5]. Para minimizar o problema da baixa escalabilidade, as hierarquias de barramentos foram utilizadas com algum êxito, mas são sempre uma solução *ad hoc* para um sistema, impossibilitando seu reuso.

Dessa forma, as NoCs emergiram como a solução para o problemas de baixa escalabilidade e reusabilidade. O modelo de comunicação de uma NoC, geralmente, é baseado na troca de mensagens encapsuladas em pacotes. A transmissão acontece da

fonte até o destino dos pacotes através dos roteadores adjacentes desse caminho, de forma paralela e em *pipeline* [6].

Geralmente o uso de NoC está associado ao aumento da área em chip e da potência dissipada. E outros problemas como coerência e consistência de dados são encontrados quando elas são usadas nos MP-SoC (*Multiprocessor SoC*) [7]. Entretanto, o uso de NoC pode ser uma solução e não uma fonte de problemas. Tomando como referência a prototipagem em FPGA de um sistema com o microprocessador NIOS II/f [8] e do roteador ParIS da NoC SoCINfp [9] percebe-se que a área em chip do processador é 90,23% maior que a do roteador [10]. Em conseqüência, é provável que o processador também dissipe mais potência que o roteador. Assim, a simplificação dos elementos de processamento, sem perda de desempenho do sistema, pode contribuir para redução de área e potência.

Baseado nesse fato, é possível projetar um sistema baseado em NoC onde os roteadores são os principais responsáveis pela execução das aplicações, permitindo reduzir significativamente os núcleos a eles ligados. Nessa proposta o modelo de computação aproveita a comunicação paralela e em forma de *pipeline* dos pacotes pelos roteadores.

Um modelo de computação semelhante é utilizado com eficiência nas máquinas em fila (*queue machines*) [11]. Tais máquinas fazem referência a uma fila de operandos, de modo que operandos e operações são removidos do início da fila para sua computação, e o resultado é, geralmente, inserido no final da fila.

Pesquisas indicam que máquinas em fila podem ser usadas para construir eficientes máquinas escalares [12] ou uma nova alternativa para arquiteturas embarcadas [13]. Apesar do modelo de computação semelhante ao proposto nesse artigo, as máquinas em fila utilizam um único processador, não um sistema *muti-core*, e também não utilizam NoC.

Uma patente recente propõe uma arquitetura *dataflow* baseada em rede em chip [14]. Essa arquitetura é formada por um tradicional processador Von Neumann e duas redes em chip com topologia em malha (uma rede de dados e uma rede de instruções). O processador é responsável pela busca de instruções e operandos. Instruções dentro de laços de repetição devem ser executadas nas NoCs, nesse caso o processador configura os elementos de processamento da rede instruções de acordo com o grafo de fluxo de dados da aplicação gerado por um compilador. Os dados são distribuídos para a rede de dados através de um barramento para que a rede de instruções processe-os. É notável que essa arquitetura tem escalabilidade comprometida pelo barramento, os grafos de fluxo de dados têm tamanho limitado pelas dimensões da NoC e que o processamento pelas NoCs é dependente da busca e configuração das instruções pelo processador. Além disso, o próprio autor afirma que esta arquitetura tem potencial apenas quando existem instruções dentro de laços de repetição. Portanto, não sendo uma arquitetura onde a NoC é a principal elemento de processamento das aplicações.

3. Arquitetura IPNoSys

Considerando as hipóteses levantadas na seção anterior, este artigo propõe uma arquitetura baseada em NoC, denominada IPNoSys (*Integrated Processing NoC System*). Nessa arquitetura a NoC deixa de ser um elemento passivo do processo de execução da aplicações para se tornar o principal responsável pela execução.

A NoC adotada nessa arquitetura tem topologia do tipo malha ou grelha-2D quadrada e usa as seguintes características: roteamento XY, chaveamento que combina VCT (*Virtual-Cut-Through*) e *wormhole*, dois canais virtuais, controle de fluxo baseado em crédito, arbitragem distribuída e memorização na entrada.

No sistema IPNoSys, os roteadores se tornaram Unidades de Processamento e Roteamento (UPR), pois também possuem uma Unidade Lógica e Aritmética (ULA) e uma Unidade de Sincronização (US) para executar as instruções das aplicações. Os processadores ou núcleos de processamento ligados a cada roteador em uma NoC direta foram substituídos por apenas quatro simples módulos denominados Núcleo de Acesso à Memória (NAM). Cada NAM é ligado a um módulo de memória, localizadas nos cantos da NoC, e são os únicos a acessarem essas memórias. Como ilustrado na Figura 1.

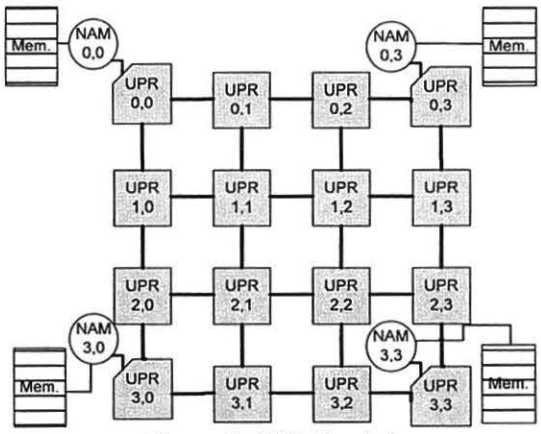

Figura 1 - IPNoSys 4x4

O sistema IPNoSys não utiliza registradores de contador de programa (PC - *Program Counter*), nem faz busca de instruções em uma pilha, nem mantém apenas um único fluxo de execução. Portanto, esse sistema não faz uso de processadores Von Neumann, uma vez que aplicações são executadas nas UPRs/NAMs por onde os pacotes são encaminhados.

O modelo de computação do sistema IPNoSys aproveita o *pipeline* e o paralelismo das transmissões dos pacotes entre as UPRs. Desse modo, os pacotes passam a funcionar como seqüências de instruções, onde a operação corrente é aquela que está no início do pacote e os resultados podem ser inseridos em qualquer parte do pacote servindo de operandos de instruções futuras. Nesse modelo, cada UPR executa a primeira instrução e encaminha o restante do pacote para a próxima UPR, inserindo seu resultado no momento da transmissão da palavra correspondente. Desse fato, observa-se que o pacote diminui à medida que é transmitido e as instruções contidas nele são executadas, como ilustra a Figura 2.

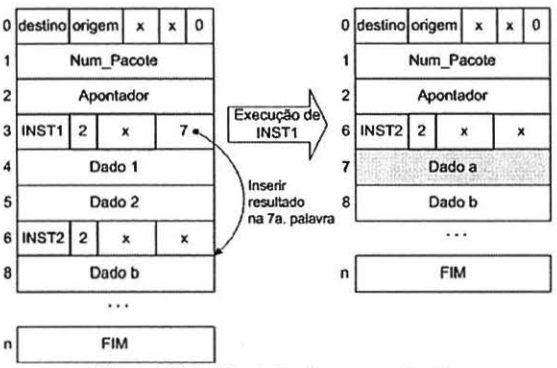

Figura 2 – Modelo de computação

O número de instruções e operandos nas aplicações, descritas em forma de pacotes, é variável. Dessa forma o tamanho dos pacotes não é fixo e, em geral, cada UPR executa apenas uma instrução por vez, de um pacote. Assim, o destino de cada pacote deve ser suficientemente distante da origem para que todas as instruções contidas nele possam ser executadas. No entanto, em uma NoC de dimensões limitadas sempre haverá uma aplicação que possua mais instruções que o número de UPRs do maior caminho que o pacote possa trafegar. Uma solução é rotear novamente o pacote que chegou ao seu destino e que ainda possui instruções a serem executadas. Entretanto nenhum algoritmo de roteamento atual está preparado para isso, portanto foi desenvolvido, junto com o sistema IPNoSys, o algoritmo *spiral complement* com esse objetivo.

3.1. Algoritmo *spiral complement*

As bases do algoritmo *spiral complement* são o roteamento XY e no padrão de tráfego *complement*.

As aplicações são descritas em pacotes, os quais são armazenados nas memórias e injetados pelos cantos da rede através dos NAMs. O primeiro destino do pacote injetado deve ser aquele mais distante em uma rede com dimensões quadradas (o complemento da fonte). Se esse caminho não for suficiente para executar todas as instruções do pacote, um novo destino é então calculado e o pacote novamente roteado a partir do destino anterior. O novo destino deve ser o mais distante possível, no entanto, deve ser diferente da fonte anterior desse pacote. Para isso, o algoritmo reduz a rede, virtualmente, em uma linha ou uma coluna, e adota o novo destino como sendo o canto da rede reduzida. Dessa forma, os novos destinos fazem o pacote percorrer caminhos em espiral, como mostrado na Figura 3.

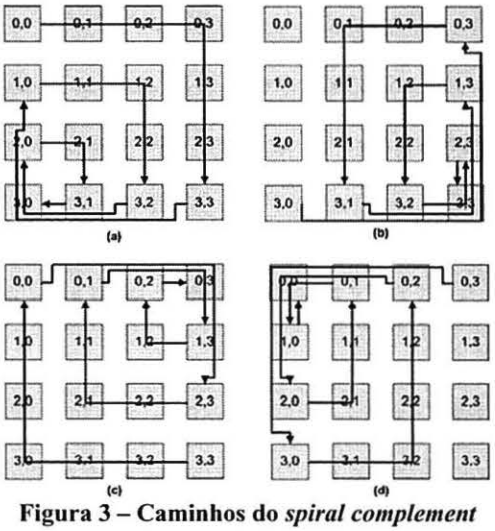

Figura 3 – Caminhos do *spiral complement*

A Figura 3(a) mostra quando o pacote é injetado no canto superior esquerdo. Na Figura 3(b) quando o pacote é injetado no canto inferior esquerdo. O caminho da Figura 3(c) para pacotes injetados pelo canto inferior direito e da Figura 3(d) injetados pelo canto superior direito. Quando um pacote passa por toda a primeira espiral (Figura 3(a)) e ainda restam instruções a serem executadas, o pacote segue a segunda espiral (Figura 3(b)) como se tivesse sido injetado por este canto. Da segunda espiral o pacote pode seguir a terceira, e desta seguir para a quarta espiral. Após a quarta espiral o pacote pode seguir a primeira espiral e reiniciar todo o processo.

O algoritmo *spiral complement* cria caminhos para execução de todas as instruções em pacotes com qualquer tamanho independente das dimensões da rede. Além disso, o algoritmo promove uma distribuição do tráfego de dados por toda a rede, evitando a inconveniente concentração na região central, comum em redes em malha usando o padrão complemento. No entanto, dependendo do tamanho do pacote e das dimensões da rede, uma situação de *deadlock* pode acontecer, então o sistema deve tratá-lo.

3.1.1. Tratamento do *Deadlock*. Considerando um pacote, injetado pelo canto superior esquerdo de uma rede 4x4 (Figura 3(a)), que após as primeiras quinze UPRs, tem o cabeçalho do pacote chegando à UPR 3,2. O pacote deveria ser encaminhado da UPR 3,2 para a UPR 3,1, mas se o pacote for tão longo que o restante do pacote continua sendo transmitido da UPR 3,3 para a UPR 3,1, o cabeçalho deve esperar pela alocação do canal. Contudo, o restante do pacote não pode ser transmitido enquanto o cabeçalho estiver esperando, caracterizando a situação de *deadlock*.

Geralmente situações de *deadlock* em NoCs são resolvidas com canais virtuais. No caso do sistema IPNoSys o número de canais virtuais corresponderia ao número máximo de vezes que um mesmo pacote passa por um canal no mesmo sentido, ou seja, seriam necessários três (veja a Figura 3). Entretanto, se o pacote fosse longo o suficiente para percorrer as quatro espirais, o *deadlock* voltaria a acontecer quando o pacote retornasse à primeira espiral e todos os canais virtuais estivessem alocados transmitindo o restante do pacote.

Portanto, a solução adotada, denominada execução localizada, resolve definitivamente esse problema. Nessa solução a UPR que detecta o *deadlock* (aquele que tenta transmitir o cabeçalho) continua executando a próxima instrução até que possa transmitir o pacote.

3.2. Formato do pacote

O pacote do sistema IPNoSys é formado por um conjunto variável de palavras de 32 *bits* de largura, onde as três primeiras palavras formam o cabeçalho, seguidas pelas palavras com instruções e operandos, terminado por uma palavra que indica o fim do pacote. Além dos 32 *bits*, são usados mais 4 *bits* sinalizadores que informam o tipo de palavra. A Figura 4 apresenta o formato do pacote junto aos *bits* de controle.

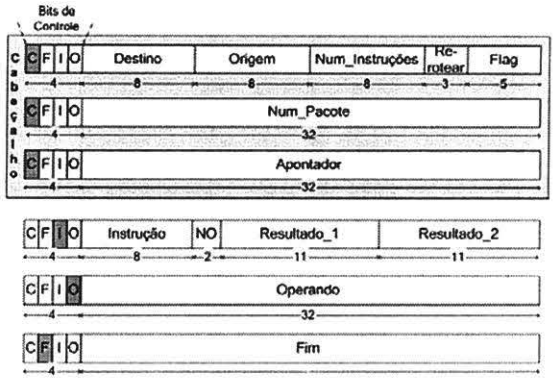

Figura 4 - Formato do pacote IPNoSys

Em cada palavra apenas um dos quatro *bits* do controle pode estar ativo. Na primeira palavra é indicado a fonte e o destino atual do roteamento, além do número de instruções no pacote seguido pelos bits que indicam como calcular o novo destino e pelo *flag* que indica o tipo de pacote. Atualmente há dois tipos de pacote: regulares ou de controle. Pacotes regulares têm suas instruções executadas em qualquer UPR por onde são roteados e são novamente roteados quando necessário. Pacotes de controle carregam instruções para serem executadas apenas no NAM indicado na própria instrução e novos destinos não são calculados. A segunda palavra é um identificador único do pacote e a terceira um apontador para a próxima instrução a ser executada no pacote. O apontador indica a quantidade de palavras transmitidas antes da instrução corrente, isso permite uma contagem global para inserção dos resultados por qualquer UPR que execute a instrução.

A palavra de instrução possui um identificador da instrução seguido pela quantidade de operandos (até dois) necessários para executá-la. Ainda possui dois campos usados pela maioria das instruções para indicar as palavras onde inserir o resultado. Instruções de NAM utilizam esses campos para o endereço do NAM e a quantidade de operandos (quando acima de dois).

3.3. Unidade de processamento e roteamento

As Unidades de Processamento e Roteamento (UPR) são roteadores com capacidade adicional de realizar operações lógico-aritméticas e instruções de salto. As UPRs possuem quatro portas para comunicação com as UPRs adjacentes, e as UPRs dos cantos possuem uma quinta porta para comunicação com o NAM correspondente.

Quando um pacote chega por uma das portas de entrada da UPR, é verificado o tipo de pacote através da *flag* no cabeçalho. Pacotes de controle são apenas

roteados usando roteamento XY sem nenhuma modificação no pacote. Pacotes regulares podem ter um novo destino calculado, usando *spiral complement*, caso o pacote tenha atingido o destino atual, em seguida o pacote é roteado usando XY.

Na porta de saída o árbitro resolve as disputas pelo canal de transmissão e também requisita a execução da instrução corrente no pacote. Caso a instrução seja lógico-aritmética ou salto, a ULA é requisitada, caso seja instrução de NAM, é requisitada a Unidade de Sincronização (US). A ULA executa a instrução e o resultado é armazenado pelo árbitro. Durante a transmissão do pacote o árbitro mantém um contador das palavras para inserir o resultado no momento correto. Instruções de NAM provocam a criação de pacotes de controle pela US. Nesse caso, o árbitro o transmite por um canal virtual exclusivo para esse tipo de pacote. Esse canal virtual é livre de *deadlock*, uma vez que os pacotes de controle são curtos e são roteados apenas usando o algoritmo XY sem movimentos circulares.

Nos casos de espaço insuficiente no *buffer* receptor, canal de transmissão já alocado ou situações de *deadlock* o árbitro realiza a execução localizada. Assim, ele continua requisitando a ULA ou US para que as instruções dos pacotes sejam executadas até que a transmissão possa ser realizada.

3.4. Núcleo de acesso a memória

O modelo de memória do sistema consiste em quatro módulos compartilhados localizados nos cantos da NoC (Figura 1). Tais módulos de memória são acessados apenas pelos Núcleos de Acesso à Memória (NAM) aos quais estão ligados.

As aplicações no sistema IPNoSys são descritas através de um ou mais pacotes no formato apresentado na Seção 3.2, que são armazenados nos módulos de memória assim como os dados da aplicação. Desse modo, os NAMs são responsáveis pela injeção dos pacotes no sistema IPNoSys e pela execução de instruções de sincronismo e instruções que lêem ou escrevem dados na memória.

Tais instruções chegam ao NAM através dos pacotes de controle e correspondem à: leitura e escrita da memória (*Load* e *Store*, respectivamente); ordem de injeção imediata de um pacote que está na memória (*Exec*); injeção de um pacote de forma sincronizada (*SincExec*), isto é, após receber um ou mais sinais de sincronismos de outros pacotes; sinalização de sincronismo (*Sinc*); e envio de um valor para ser inserido em um outro pacote (*Send*).

Portanto, instruções não executadas nas UPRs são executadas nos NAMs e com isso não sendo necessário o uso de processadores Von Neumann no sistema IPNoSys.

4. Resultados

A arquitetura IPNoSys foi implementada em SystemC [15] no nível de precisão de ciclos. Para simular a execução de aplicações nessa arquitetura também foi desenvolvida uma linguagem de descrição de pacotes, definida por macros que especificam os valores dos campos dos pacotes.

Para descrever uma aplicação em alto nível para a linguagem de descrição de pacotes é utilizado o grafo de fluxo de dados (DFG - *data-flow graph*) como linguagem intermediária. Tal grafo determina a dependência de dados, permitindo identificar quantos pacotes são necessários para executar a aplicação e entender a relação de sincronismo entre eles. Essas informações são úteis não apenas para a precisão da execução, mas também para formular um modelo para melhor desempenho.

Para validar e avaliar o sistema IPNoSys foram realizadas simulações de dois estudos de casos. O primeiro é de um simples contador com o objetivo de avaliar a solução do *deadlock* (execução localizada) através de dois cenários de simulação (seqüencial e paralela). O segundo estudo de caso é a transformada discreta do cosseno (DCT - *Discrete Cosine Transform*) em duas dimensões. As simulações dessa aplicação foram realizadas em instâncias do sistema IPNoSys com diferentes dimensões da rede e também foram comparados com uma plataforma virtual MP-SoC.

4.1. Contador simples

A aplicação do contador é um conjunto de adições unitárias e acumulações sucessivas com o resultado final armazenado na memória. Essa aplicação foi usada para avaliar o tratamento do *deadlock*. Para isso, o contador foi simulado com um número variável de instruções de adição nos pacotes. Pacotes com 8, 16, 32, 64, 128 e 256 instruções de adição foram gerados para execução paralela e seqüencial. Na execução seqüencial, um único pacote, incluindo todas as instruções, foi injetado pelo canto superior esquerdo da rede. Na execução paralela, quatro pacotes foram injetados paralelamente cada um por um dos cantos da rede. Em todos os casos foi usada uma rede 4x4.

A Figura 5 mostra a quantidade de bytes de instruções executadas em cada UPR nas simulações do contador. Como os pacotes foram injetados pelo canto superior esquerdo em uma rede 4x4, a situação de

deadlock acontece na UPR 3,2 nas simulações com mais de 32 instruções. Essa figura mostra que a execução localizada resolve eficientemente o problema do *deadlock*, entretanto pode ser necessário grandes *buffers* nos árbitros para armazenar os resultados temporariamente.

As mesmas simulações foram executadas paralelamente, através de quatro pacotes injetados pelos cantos da rede. A distribuição da carga na rede foi melhor balanceada, como apresentada na Figura 6, exigindo *buffers* menores para o armazenamento.

Conseqüentemente, com o crescimento dos pacotes também cresce o número de instruções executadas em todas as UPRs. A Figura 6 também mostra que a maior quantidade de instruções executadas acontece nos cantos da rede, provocada pelo algoritmo *spiral complement* e assim evitando a inconveniente concentração na região central da rede.

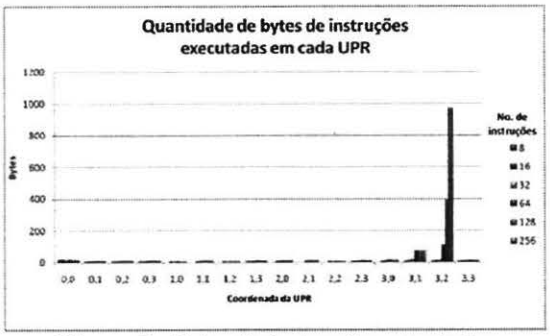

Figura 5 - Bytes de instrução executadas sequencialmente

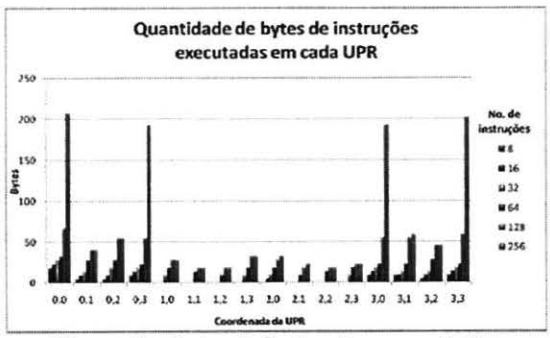

Figura 6 - Bytes de instruções executadas paralelamente

Esses resultados também demonstram a capacidade de processamento paralelo do sistema IPNoSys, o qual permite reduzir o número máximo de bytes de instruções executadas em uma única UPR em até 80%, comparando as execuções paralela e seqüencial.

4.2. Transformada discreta do cosseno

O sistema IPNoSys é capaz de executar todas as instruções de uma aplicação independente das dimensões da rede. Para avaliar essa propriedade foi implementada a transformada discreta do cosseno em duas dimensões (DCT-2D) que utiliza a propriedade da separabilidade [16]. Nessa avaliação a mesma implementação da DCT foi executada em cinco instâncias de IPNoSys com as seguintes dimensões da rede: 2x2, 3x3, 4x4, 5x5 e 6x6. Foram considerados dois aspectos: tempo de execução e número máximo de instruções executadas por UPR, mostrados nas Figuras 7 e 8, respectivamente.

Figura 7 - Tempo de execução da DCT-2D

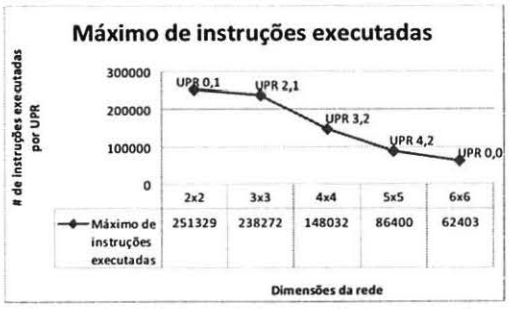

Figura 8 - Máximo de instruções executadas

O tempo de execução tende a aumentar com o aumento das dimensões da rede, no entanto torna-se estável a partir da dimensão 4x4. Esse comportamento está relacionado com a quantidade de RPUs que armazena parte dos pacotes, executa pelo menos uma instrução e transmite o restante do pacote.

A execução mais rápida acontece na instância IPNoSys de menores dimensões (Figura 7), uma vez que essa instância (2x2) possui poucos enlaces e menos comunicação. Em outras palavras, quando a rede é menor o número de instruções executadas por UPR é maior, veja Figura 8. Entretanto, NoCs menores são mais restritas com relação ao paralelismo pois elas são mais suscetíveis as situações de *deadlock*.

Portanto, eliminando a possibilidade de ter uma arquitetura paralela com uma única UPR.

A DCT-2D também foi avaliada considerando o paralelismo através da implementação de três cenários, todos executados em uma instância 4x4 do sistema IPNoSys. No primeiro cenário todos os pacotes da aplicação são injetados pelo mesmo canto da rede seqüencialmente. No segundo cenário os pacotes que controlam os laços de repetição da aplicação (pacotes de *loop*) são injetados por um mesmo NAM e os pacotes responsáveis pelo cálculo da DCT injetados pelo NAM mais próximo onde a execução dos pacotes de *loop* finalizou. Finalmente, no terceiro cenário, os pacotes de *loop* são injetados por um NAM e os pacotes do cálculo da DCT são injetados paralelamente pelos quatro cantos da rede.

Para avaliar o desempenho nos três cenários a DCT-2D também foi executada em uma plataforma virtual MP-SoC – STORM [17]. STORM usa processadores SPARC V8, interconexão baseada em uma rede com topologia malha 2D [18] e versões de memória distribuída ou compartilhada (com ou sem *cache*). Em todas as versões foram executadas implementações paralela e seqüencial da DCT-2D e redes 4x4. Para comparar as duas arquiteturas considerou-se o tempo de execução e a quantidade de memória requerida.

A Figura 9 mostra a quantidade de memória requerida para cada instância da STORM e dos cenários do sistema IPNoSys. A memória requerida corresponde à soma dos dados globais, dados locais e código da aplicação, em *bytes*. O primeiro cenário IPNoSys (execução seqüencial) requer menos memória que os outros dois cenários, os quais são equivalente a STORM com memória distribuída, como era esperado.

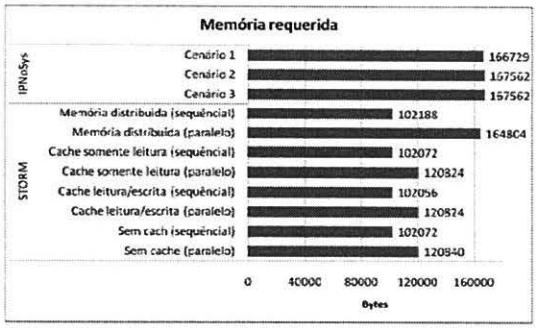

Figura 9 - Memória requerida DCT

Comparando o tempo de execução (Figura 10), o melhor caso da INoSys é o primeiro cenário que é ultrapassado apenas pelo melhor caso da STORM (execução paralela com cache leitura/escrita). A execução seqüencial no sistema IPNoSys tem menor tempo de execução devido as características da aplicação e o nível de paralelismo adotado (paralelismo a nível de instrução). A DCT-2D possui muita dependência de dados, que é o pior caso para paralelização em nível de instrução. Mesmo assim, o tempo de execução na implementação paralela (cenário 3) é consideravelmente baixo. Além disso, como o sistema IPNoSys não possui processadores, a área em chip pode ser substancialmente reduzida uma vez que a prototipagem em FPGA mostrou que a área de uma NAM é 84% menor que a o processador NIOS II/f.

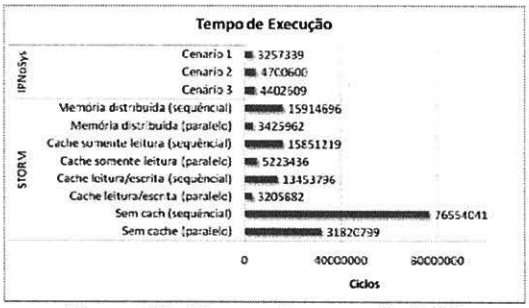

Figura 10 - Tempo de execução DCT

5. Conclusões e trabalhos futuros

Este artigo apresentou uma nova arquitetura baseada em NoC a qual não faz uso de processadores Von Neumann, denominada IPNoSys. Tal arquitetura possui simples estruturas, Núcleos de Acesso à Memória (NAM), apenas nos cantos da rede, e roteadores capazes de executar instruções, denominados Unidades de Processamento e Roteamento (UPR). Aplicações são descritas em um formato de pacote específico que inclui instruções e operandos que são executados quando o pacote é transmitido da fonte até o destino. O caminho dos pacotes é determinado por um algoritmo de roteamento próprio (*spiral complement*) que garante a execução de todas as instruções independente das dimensões da rede. Além disso, os pacotes vão diminuindo a medida que são transmitidos/executados permitindo um aproveitamento máximo dos canais de comunicação.

O sistema IPNoSys também identifica e trata situações de *deadlock* através da solução denominada execução localizada. A eficiência dessa solução foi mostrada através de simulações com diferentes quantidades de instruções. Simulações do sistema IPNoSys com redes de diferentes dimensões foram realizadas executando a transformada discreta do cosseno (DCT) para confirmar a capacidade do sistema independente da quantidade de UPRs.

Finalmente, a DCT foi usada para avaliar o desempenho do sistema IPNoSys em relação a

plataforma STORM. Nas simulações o sistema IPNoSys utilizou tanta memória quanto a plataforma STORM. O tempo de execução no sistema IPNoSys é relativamente curto para diferentes estratégias de paralelismo, mesmo para o pior caso de paralelização.

Trabalhos futuros incluem: aplicações em outros níveis de paralelismo; utilização de canais virtuais para transmitir pacotes regulares; novos algoritmos de roteamento; configuração das UPRs indicando a quantidade de instruções a serem executadas; desenvolvimento de um compilador; prototipação em FPGA e avaliação de área em chip e potência dissipada.

10. Referências

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proceedings of the 41st annual conference on Design automation* San Diego, CA, USA: ACM, 2004.

[2] V. Borkar, "Platform 2015: Intel processor and platform evolution for the next decade," in *Intel Corporation white paper*, 2005, pp. 3-12.

[3] INTEL, "Product Brief: Intel IXP2850 Network Processor" www.intel.com/design/network/prodbrf/25213601.pdf. Access: 20 jan. 2008

[4] A. Artieri, V. D'Alto, R. Chesson, M. Hopkins, and M. C. Rossi, "NomadikTM Open Multimedia Platform for Next-generation Mobile Devices" http://www.st.com. Access: 20 jan. 2008

[5] J. Backus, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," in *Communications of the ACM*, 1978, pp. 613-641.

[6] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Destination network-on-chip," *EDA Tech Forum Journal,* pp. 6-7, 2005.

[7] G. Girão, B. C. Oliveira, R. Soares, and I. S. Silva, "Cache Coherency Communication Cost In A Noc-Based Mp-Soc Platform," in *20th Symposium on Integrated Circuits and Systems Design*, Rio de Janeiro, 2007, pp. 288-293.

[8] C. Aletra, "Nios II Processor Reference Handbook" http://www.altera.com/literature/lit-nio2.jsp. Access: Jan. 2008

[9] C. A. Zeferino and A. A. Susin, "SoCIN: A Parametric and Scalable Network-on-Chip," in *Proceedings of the 16th symposium on Integrated circuits and systems design*, 2003, pp. 169-174.

[10] S. R. F. d. Araújo, "Estudo da viabilidade do desenvolvimento de sistemas integrados baseados em redes em chip sem processadores: sistema IPNoSys," in *Departamento de Informática e Matemática Aplicada*. vol. Mestre Natal: UFRN, 2008, p. 87.

[11] B. R. Preiss and V. C. Hamacher, "Data Flow on Queue Machines," in *12th Int. IEEE Symposium on Computer Architecture*, 1985, pp. 342-351.

[12] H. Schmit, B. Levine, and B. Ylvisaker, "Queue Machines: Hardware Compilation in Hardware," in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 152-160.

[13] A. Canedo, B. Abderazek, and M. Sowa, "Queue Register File Optimization Algorithm for QueueCore Processor," in *19th International Symposium on Computer Architecture and High Performance Computing*, 2007.

[14] H. T. L. Nguyen, "Network-on-chip dataflow architecture," U. S. p. a. t. office, Ed. United States: Hanoi Tran Le Nguyen (VN), 2007, p. 9.

[15] OSCI, "SystemC" http://www.systemc.org. Access: 20 jan. 2008

[16] L. V. Agostini, "Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG," in *Instituto de Informática*. vol. Mestrado Porto Alegre: Universidade Federal do Rio Grande do Sul, 2002, p. 143.

[17] R. S. d. L. S. Rego, "Projeto e Implementação de uma Plataforma MP-SoC usando SystemC," in *Departamento de Informática e Matemática Aplicada*. vol. Mestrado Natal: Universidade Federal do Rio Grande do Norte, 2006, p. 144.

[18] R. Soares, I. S. Silva, and A. Azevedo, "When reconfigurable architecture meets network-on-chip," in *Proceedings of the 17th symposium on Integrated circuits and system design* Pernambuco, Brazil: ACM, 2004.