Exploração de paralelismo em nível de instruções e de tarefas em uma arquitetura de processamento não convencional

Aparecida Lopes de Medeiros¹

Ana Luisa Medeiros²

Márcio Eduardo Kreutz¹

Resumo: Impulsionadas pelos constantes avanços da indústria de semicondutores, as aplicações computacionais têm evoluído consideravelmente, uma vez que podem fazer uso de circuitos eletrônicos eficientes e com alto poder computacional. Nesse sentido, a busca por maior desempenho pode ser conseguida por meio do aumento da frequência de operação, o que permite instruções executarem em tempo menor, ou pela execução concorrente das operações, por meio de técnicas de paralelismo. Este artigo apresenta a implementação da técnica de paralelismo chamada *software pipelining* em uma arquitetura de processamento não convencional. Adicionalmente, são mostrados resultados de simulação, que comprovam a eficiência da técnica quando o desempenho é medido em versões da arquitetura com e sem a implementação do *software pipelining*.

Palavras-chave: Arquiteturas de processamento não convencionais, Paralelismo, Processadores Superescalares, Software *Pipelining*.

Abstract: Driven by the increasingly advances in the semiconductor industry, computational applications could evolve substantially, since they can profit from high computational power offered by electronic circuits. Going in this direction, the search for performance improvements could be obtained either by increasing the operation frequency, which allows instructions to execute in less time, or by concurrent execution of operations through parallelism techniques. This paper presents the implementation of a parallelism technique, called software pipelining, on a non-conventional processing architecture. Simulation results are provided, which comprove the efficiency that can be achieved with this technique when performance on versions with and without the technique are compared.

Keywords: *Non-conventional processing architectures, Parallelism, Superscalar Processors, Software Pipelining.*

1. Introdução

Historicamente, arquitetura de processadores de propósito geral tem avançado, visando atender à demanda das aplicações por desempenho, uma vez que os desenvolvedores podem criar aplicações com funcionalidades que demandam mais recursos. Nesse sentido, a busca por desempenho pode ocorrer pelo aumento da frequência de operação, e por modificações na arquitetura do processador. Essas modificações necessitam de técnicas de paralelismo, as quais focam em execução concorrente daquelas operações que não apresentem dependências de dados entre si.

¹ Programa de Pós-Graduação em Sistemas e Computação, Departamento de Informática e Matemática Aplicada, UFRN, Natal (RN) - Brasil

{aparecidalp@gmail.com, kreutz@dimap.ufrn.br}

² Programa de Pós-Graduação em Engenharia Elétrica, Centro de Engenharia Elétrica e Informática, UFCG, Campina Grande (PB) – Brasil

{analuisafdm@gmail.com}

O paralelismo é um termo utilizado para indicar que uma arquitetura permite várias execuções simultâneas, bem como pode ser obtido no tempo e no espaço. A diferença entre esses dois tipos de paralelismo se verifica na “granularidade” das operações que de fato, executam em paralelo. No entanto, é necessário que se utilize uma granularidade de referência, que num processador de propósito geral, corresponde a uma *instrução* de seu conjunto de instruções. Dessa forma, quando o paralelismo se dá em nível de *micro-instruções*, o paralelismo é dito *temporal*, pois a granularidade de referência - as instruções - executam sequencialmente. Nesse contexto, a concorrência se dá pela execução simultânea das unidades da parte operativa do processador. A técnica que implementa paralelismo temporal é chamada de *pipeline* [8].

O *pipeline* representa uma forma de paralelismo temporal, caracterizado pela existência de execução de eventos sobrepostos no tempo. A tarefa a ser executada é dividida em sub-tarefas, cada uma dessas sendo executada por um estágio de *hardware* especializado, que trabalha de maneira concorrente com os demais estágios envolvidos na computação.

De outra forma, quando o paralelismo ocorre em nível de instruções, este é chamado de *espacial*, já que para as instruções completas (com todas as suas *micro-instruções*) executarem concorrentemente, é necessário que unidades funcionais da parte operativa do processador sejam replicadas. Adicionalmente, o paralelismo espacial também é conhecido por paralelismo em nível de instrução (ILP - *Instruction Level Parallelism*).

Com o aumento da granularidade do paralelismo, ocorre a execução concorrente de um conjunto de instruções, ou tarefas (TLP - *Task Level Parallelism*). Esse tipo de paralelismo é caracterizado pela execução concorrente de instruções pertencentes à diferentes tarefas em uma mesma parte operativa. Aplicação de TLP está presente em processadores *superescalares* ou com palavra de instrução muito larga (VLIW - *Very Large Instruction Word*). No entanto, essas são arquiteturas de processamento convencionais, no sentido de que seguem o padrão de *Von Neumann*, onde dados e instruções são buscados separadamente, normalmente fazendo uso de memórias e caminhos de dados distintos.

Nesse contexto, este artigo propõe a exploração de paralelismo em nível de instruções (ILP) e de tarefas (TLP) em uma arquitetura não convencional, ou seja, que não segue o modelo proposto por *Von Neumann*. Desse modo, como técnica de paralelismo a ser empregada, optou-se pelo *software pipelining*, devido à sua eficiência em arquiteturas convencionais. Além disso, este artigo objetiva verificar como esta técnica pode ser aplicada em outro tipo de arquitetura e avaliar o seu impacto no desempenho quando instruções e tarefas são executadas.

A arquitetura a ser utilizada neste artigo é a IPNoSyS (*Integrated Processing Network-on-Chip System*) [1]. Esta arquitetura apresenta diversos núcleos para a execução de operações, portanto, emprega paralelismo espacial por construção. Além disso, possui um modelo de execução de instruções baseado em pacotes, cuja natureza compatibiliza com a repetição de instruções, o que vem de encontro com a lógica de execução do *software pipelining*. Dessa forma, mais de um pacote pode ser executado simultaneamente, permitindo a execução de paralelismo em nível de tarefas.

Na arquitetura IPNoSyS, cada *pacote* contém dados e instruções, o que caracteriza uma arquitetura de processamento não convencional, uma vez que estes não necessitam espaços de endereçamento distintos, nem tampouco serem buscados por canais de comunicação diversos.

O *software pipelining* é uma técnica que tenta aproveitar ao máximo os recursos disponíveis da arquitetura para execução de instruções paralelas, permitindo a execução de instruções de iterações diferentes em um mesmo ciclo de execução. Originalmente, a arquitetura IPNoSyS não apresenta a técnica de *software pipelining* implementada. Portanto, neste artigo, além das comparações de desempenho, foram realizadas modificações na arquitetura original, a fim de oferecer suporte a essa técnica.

Este artigo está estruturado da seguinte forma: a Seruturado da seguinte forma fim de ofer; a Seruturado da seguinte forma fim de oferecer suportnais de comunica permitindo a; na Seuturado apresentada a implementação; e na Seturad, os resultados dos experimentos; por fim, na Ses dos, sfim, na Ses dos as conclusões e as sugestões de trabalhos futuros.

2. Trabalhos relacionados

Há diversos trabalhos que exploram paralelismo em nível de instruções, em especial para processadores VLIW (*Very Long Instruction Word*) e Superescalares, uma vez que dispõem de várias unidades funcionais para a execução de instruções. Neste artigo, foi utilizada a técnica de paralelismo *software pipelining*, pela é possível utilizar eficientemente as unidades funcionais disponíveis em processadores na execução instruções que se repetem no tempo, ou seja, *laços de repetição*.

Em [6], a técnica de *software pipelining* é aplicada na arquitetura *Jetpipeline*. Essa arquitetura integra duas arquiteturas, uma VLIW e outra superescalar, onde as instruções são escalonadas por um compilador da arquitetura VLIW e executadas pela arquitetura superescalar. Essa arquitetura explora paralelismo em nível de instrução, tendo quatro unidades funcionais, as quais podem executar instruções em paralelo. Além disso, as simulações foram realizadas com um, três, cinco, sete e nove núcleos, e, em todos os casos, obtiveram menos ciclos de execução com o uso da técnica. A Tabela 1 mostra os resultados obtidos nas simulações. Com a avaliação dos testes, a técnica *software pipelining* se mostrou uma técnica com potencial para proporcionar aumento de desempenho em arquiteturas de processadores.

Em [4], a técnica *software pipelining* foi empregada em uma arquitetura multi-core, a qual aproveita todos os núcleos dessa arquitetura para processamento. Foi utilizada a técnica SSP (*Single-dimension Software-Pipelining*), que é método de *software pipelining* com recursos limitados, aplicado a repetições perfeitas e imperfeitas. Além disso, para avaliação de aplicações foi proposto um método com base em duas operações: "*wait*" que é colocada antes da repetição, e "*signal*", que é colocada após; essas operações servem como um sinal para determinar se a instrução pode ou não executar naquele determinado momento. O *laço* é escrito em uma linguagem sequencial, onde o método citado anteriormente extrai automaticamente as *threads*, e escalona as instruções dentro dessas *threads*. Cada *thread* é atribuída a um *core*, e que são responsáveis pela sincronização e gerenciamento da comunicação. Com a utilização deste método o escalonamento está livre de bloqueio e respeita a dependência e restrições de recursos.

Em [7], é explorada a eficiência da tensão e frequência para processadores e a comunicação, com e sem a técnica *Software Pipelining* na arquitetura da NoC RAW-like. Essa arquitetura é uma NoC *mesh*, que tem módulos de CPU, uma *cache* para instruções e outra para dados, um *switch* que conecta os nodos e um *buffer* para armazenamento. Nesse trabalho, foram mostradas a economia e vazão de energia obtida por meio da tensão e frequência, explorando apenas ILP. Depois, foi proposta uma abordagem para identificar o impacto da vazão e economia de energia, aplicando a técnica *software pipelining* na NoC. Essa abordagem foi realizada da seguinte forma: dado um código sequencial, o compilador extraiu o paralelismo do código em partições e mapeou o código para múltiplos processadores da NoC; a partir do código paralelo, foram analisadas as estimativas da velocidade da CPU e da comunicação (ou seja, períodos ociosos que podem ser utilizados para economia de energia por meio da frequência e tensão); com base nessas informações, foram inseridas instruções de controle apropriadas à voltagem da CPU/comunicação no código e obtém, assim, o código final paralelizado. A partir do resultado obtido pela execução, verificou-se que a exploração de *software pipelining* pode melhorar o desempenho de forma significativa (20% neste caso) devido à extração de paralelismo em nível de instrução e ao uso eficaz dos recursos disponíveis na arquitetura.

Como podem ser verificadas, todas as abordagens acima elencadas referem-se à aplicação da técnica de *software pipelining* em arquiteturas convencionais. O que se busca com esse trabalho é a avaliação da eficiência dessa técnica no desempenho relacionado à execução de instruções em uma arquitetura não convencional.

Tabela 1: Resultado experimental da simulação

Nº de núcleos	Nº de ciclos sequencias	Nº de ciclos com software pipelining	Percentual de redução com software pipelining
1	19032	5024	73%
3	9019	3344	63%
5	3083	1212	61%
7	11503	3111	73%
9	6110	1650	73%

3. Plataforma IPNoSyS e a Técnica de *Software Pipelining*

Essa seção mostra as características da plataforma IPNoSyS e da técnica de paralelismo *Software Pipelining*.

3.1. IPNoSyS

A IPNoSyS é uma arquitetura multiprocessada, onde diversos núcleos de processamento são conectados por um conjunto de canais ponto a ponto, formando uma topologia matricial. Assim sendo, apesar de ser concebida como uma arquitetura de processamento de propósito geral (para a execução de algoritmos), a IPNoSyS pode ser também interpretada como uma arquitetura do tipo Rede em Chip (*Network-on-Chip* - NoC) [10], uma vez que, devido ao seu modelo de execução ser baseado na transmissão de pacotes entre os núcleos de processamento, esses implementam também operações típicas encontradas em roteadores de redes em chip.

Nesse contexto, a IPNoSyS é definida pelas seguintes características:

- Topologia em grelha 2D de dimensão quadrada;
- Roteamento XY com *Spiral Complement* (ver Figura 2);
- Chaveamento *wormhole*;
- Controle de fluxo baseado em créditos;
- Arbitragem distribuída; e
- Armazenamento na entrada.

No entanto, no contexto desse artigo, a IPNoSyS é compreendida como uma arquitetura de processamento de propósito geral, cuja capacidade de paralelismo é avaliada por meio da técnica de *software pipelining*.

A arquitetura da IPNoSyS é composta por um modelo de execução baseado em pacotes, onde dados e instruções trafegam juntos pelos canais de comunicação internos. Isso caracteriza uma arquitetura de processamento não convencional.

A execução das instruções que compõem as aplicações é realizada dentro de cada núcleo de processamento (roteador), seguindo uma rota de execução definida pela política de roteamento adotada. Cada roteador tem internamente uma "Unidade Lógica Aritmética" (ULA) para a execução de instruções, podendo ser caracterizado como uma "Unidade de Roteamento e Execução", RPU (*Routing and Processing Unit*).

A arquitetura tem quatro unidades de acesso à memória (*Memory Access Units* - MAUs) onde são armazenadas as instruções e os dados. As RPUs são responsáveis por montar e injetar os pacotes na arquitetura. O sistema de entrada e saída (*IONode* e *MemArbiter*) também é ligado a uma das MAUs, conforme mostra a Figura 1.

Cada MAU tem uma estrutura de memória onde os programas são armazenados e enfileirados para que possam ser executados na arquitetura, por meio da injeção dos *flits* do pacote na RPU, a qual está conectada. Cada pacote é formado por um conjunto de *flits*, que correspondem à quantidade de bits que é injetada simultaneamente nos canais de comunicação. Nas Figuras 4 e 5, cada linha corresponde a 1 *flit*.

As RPUs possuem a função de executar as instruções e transmitir os pacotes. A RPU pode ter três ou quatro portas de entrada/saída, dependendo da sua localização na arquitetura; há dois *buffers* do tipo FIFO em cada uma dessas portas, sendo um para entrada e outro para saída, e mais um *buffer* interno para armazenar os resultados das instruções executadas. Adicionalmente, cada porta de saída tem um árbitro que além de controlar as disputas pelo canal de transmissão (usando a política de arbitragem *round-robin*), também é responsável por remover a instrução corrente do pacote e inserir o resultado no mesmo pacote durante a transmissão.

Figura 1: Arquitetura da IPNoSyS [2]

Adicionalmente, na IPNoSyS as aplicações precisam estar descritas em formato de pacotes, que contêm instruções e operandos a serem processados durante o percurso do pacote na arquitetura. Nesse sentido, os pacotes são armazenados nas memórias nos cantos da arquitetura. As MAUs são responsáveis tanto pelo gerenciamento das memórias quanto pela injeção dos pacotes na RPU (ver Figura 1).

A IPNoSyS implementa um modelo de execução orientado a pacotes: cada instrução executa em uma RPU, e é retirada do pacote; caso a instrução gere dados a serem utilizados por instruções subsequentes no fluxo de execução, esses dados são armazenados em um *buffer* interno da RPU e acessados pelas próximas instruções. A execução do programa termina quando é executada a instrução do *flit* terminador. As quatro unidades de acesso à memória (MAUs) permitem que sejam injetados até quatro pacotes simultâneos na arquitetura. Dessa forma, cada pacote pode representar uma tarefa de uma aplicação, o que implementa paralelismo em nível de tarefas.

O algoritmo de roteamento utilizado na IPNoSyS é o *Spiral Complement*. Esse algoritmo foi desenvolvido para essa arquitetura, combinando o roteamento estático "XY" e o tráfego complementado (*complement*). De acordo com [2], o algoritmo foi criado para que se possam rotear os pacotes enquanto houver instruções para serem executadas, sendo que seu trajeto segue a forma de uma "espiral complementada", daí o seu nome.

Figura 2: As 4 rotas possíveis espirais

A Figura 2 exemplifica as rotas das quatro possíveis espirais definidas pelo algoritmo de roteamento *Spiral Complement*, que podem co-existir em uma arquitetura de tamanho 4x4. As espirais sempre começam nos cantos da arquitetura (RPU's "0,0"; "0,3"; "3,0" e "3,3") porque nesses cantos estão conectadas as RPU's, as quais injetam os pacotes. Na Figura 4(a), a espiral inicia na RPU "0,0", enquanto que nas Figuras. 4(b), 4(c), 4(d) nas RPU's "3,0", "3,3" e "0,3" respectivamente.

Note que, dependendo do tamanho da arquitetura e do pacote, a espiral irá se autobloquear em algum momento. Isso pode ser verificado, por exemplo, na RPU "2,2" da Figura 4(a): se o pacote for suficientemente grande, quando *flits* chegarem na RPU "2,2", ainda existirão *flits* na RPU "3,2". Como os *flits* de ambas RPU's seguem pela mesma porta de saída (oeste), *flits* da RPU "2,2" ficam bloqueados.

O auto-bloqueio implica que, a partir desse momento, o pacote não pode mais avançar pelos canais de comunicação, e implica a execução localizada, ou seja, as instruções passam a ser executadas sequencialmente na RPU, onde ocorreu o bloqueio até esse terminar ou até terminar o pacote.

Em [3, 9], foi proposto um novo algoritmo de roteamento para a IPNoSyS, chamado de *Zigue-Zague*, que segue um modelo determinístico e com rotas de mesmo tamanho nas quatro possibilidades de injeção (RPU's "0,0", "3,0", "0,3" e "3,3"). Essa nova política de roteamento evita a situação de execução localizada (como ocorre no roteamento *Spiral Complement*) e busca uma melhor distribuição da execução das instruções nas RPU's, evitando a ociosidade das unidades de processamento. A Figura 3 mostra um exemplo de rotas definidas pelo algoritmo de roteamento *Zigue-Zague* em uma arquitetura de tamanho 4x4. Além disso, é possível verificar que o autobloqueio não ocorre, uma vez que os caminhos não se cruzam.

Figura 3: Exemplo de roteamento Zigue-zague

3.2. Software Pipelining

Software pipelining aparece como uma das técnicas mais importantes para a melhoria do desempenho de uma aplicação por meio de paralelismo, dado o domínio do tempo de execução de um *laço* sobre o tempo de execução da aplicação.

Nesse contexto, *Software Pipelining* é uma técnica que reorganiza as instruções, reestruturando *laços de repetição* de modo que várias iterações sejam sobrepostas, ou seja, executadas ao mesmo tempo. Segundo [5], *software pipelining* utiliza a ideia de que o corpo do *laço* pode ser reformado de modo que uma iteração do *laço* possa iniciar antes da iteração anterior ter terminado, obtendo, assim, maior paralelismo e aumento significativo na taxa de execução.

Essa técnica é implementada usualmente em arquiteturas VLIW e Superescalares, por serem arquiteturas que possuem várias unidades funcionais que podem ser usadas para a execução em paralelo das instruções de cada

iteração do laço: instruções pertencentes a iterações diferentes podem executar em um mesmo ciclo, mapeadas para diferentes unidades funcionais da arquitetura.

4. Implementação

A implementação da técnica de paralelismo *software pipelining* foi realizada com o objetivo de se minimizar o número de modificações na arquitetura original da IPNoSyS. Com isso, busca-se compatibilidade com a arquitetura original em relação ao conjunto de instruções, com os formatos de pacotes e com o código objeto gerados.

O modelo de execução original da IPNoSyS é orientado a dados, ou seja, cada instrução que executa, pode deixar dados para a instrução seguinte do fluxo de execução, na RPU onde executou; em seguida, o código da instrução é retirado do *flit* do pacote a que pertence, o qual segue para a próxima RPU.

Durante a implementação, verificou-se que essa característica do modelo de execução da IPNoSyS é favorável à implementação do *software pipelining*, pois no caso de uma instrução que acabou de executar não ser retirada do pacote, a qual será executada novamente na próxima RPU para onde o pacote será roteado, o que implementa uma lógica de *repetição*.

Portanto, a implementação do *software pipelining* na IPNoSyS pôde ser realizada sem impactar significativamente a arquitetura original. Essencialmente, teve que ser modificado um campo do pacote para indicar a presença de instruções em laços; criado um contador para o número de iterações dos laços; e implementada uma lógica em hardware dentro de cada RPU para controlar o descarte de *flits* em pacotes: instruções caracterizadas como pertencendo a um laço somente tem seu *flit* descartado do pacote quando o contador acusa que o número de iterações para esse laço foi atingido.

4.1. Formato do pacote

O pacote para instruções pertencentes a *laços de repetição* é muito semelhante ao original proposto para a IPNoSyS (Figura 4). Isso se deve à tentativa de se manter ao máximo a compatibilidade com a linguagem original da arquitetura, bem como implementar o paralelismo com o mínimo de modificações arquiteturais. As alterações realizadas se referem à inserção de mais um *flit* de instrução contendo informações sobre: 1) o número de iterações do *laço* e 2) o número de vezes que cada instrução em *laço* deve executar. A Figura 5 exemplifica o novo formato de pacote.

Quanto às modificações na arquitetura, foram adicionados dois componentes arquiteturais: (i) um contador para o número de execuções de cada instrução pertencente a um laço; e (ii) um pequeno circuito de controle que compara o valor atual do contador (após a execução da instrução) com o número de iterações previstas para o laço.

Figura 4: Estrutura do Pacote IPNoSyS original.

Figura 5: Estrutura do Pacote IPNoSyS para instruções que executam em laços de repetição

4.2. Modelo de execução

Na seção 3, é mostrado o modelo de computação e execução da IPNoSyS, em que cada RPU executa uma instrução de cada pacote por vez e, após essa execução, a instrução e os operandos são descartados do pacote, de modo que o restante do pacote é transmitido para a próxima RPU.

Na IPNoSyS com *software pipelining*, esse modelo de execução foi modificado para que as RPUs executem em paralelo, de modo que, se a RPU estiver disponível e possuir instrução pra ser executada, essa execução é feita. Essa nova versão do modelo de execução proporciona paralelismo em nível de instruções, acelerando a execução das operações.

O modelo de execução proposto para a IPNoSyS com *software pipelining* estabelece que cada RPU irá ler o código da operação, decodificá-la e executá-la; em seguida, a RPU testa se essa instrução é uma instrução do tipo *laço*; caso não seja, o código da operação é retirado do pacote e este enviado para a próxima RPU (exatamente o que ocorre na arquitetura original). Se a instrução pertencer àum laço de repetição, o controle interno da RPU verifica se esta instrução já executou a quantidade de vezes prevista para o laço; se já tiver sido executado, essa instrução e seus respectivos operandos são descartados do pacote, se não tiver sido executado, o campo do contador do número de execuções no *flit* de instrução do pacote é incrementado e a instrução com seus operandos são transmitidos para a próxima RPU, a fim de executarem novamente. Esse processo é repetido até se chegar ao número de repetições desejadas para cada laço da aplicação. A execução da aplicação termina quando não existirem mais *flits* no pacote que contém as instruções.

A Figura 6 exemplifica a sequência de execução da IPNoSyS com *Software Pipelining* em: a) Pacote na Memória; b) *Flits* de Cabeçalho, instrução e operando na RPU; c) Execução da 1ª iteração da 1ª instrução e geração do resultado; d) Transmissão do cabeçalho, instrução e operandos para a próxima RPU; e) Execução da 1ª iteração da 2ª instrução e execução da 2ª iteração da 1ª instrução, f) Execução da 2ª iteração da 2ª instrução, geração do resultado e descarte do cabeçalho; g) Transmissão do *flit* do terminador e f) Pacote concluído e descartado da arquitetura. Quando houver algum resultado após a execução da instrução, esse resultado é guardado no *buffer* de resultados da RPU que o gerou e depois é inserido, durante a transmissão, na posição que precisará deste valor. Note que, na Figura 6(e), tem-se duas instruções executando concorrentemente, o que reflete a implementação do *software pipelining*.

Figura 6: Modelo de computação e execução da IPNoSyS com "Software Pipelining".

5. Resultados Obtidos

Esta seção apresenta os resultados dos testes de simulação realizados com a IPNoSyS com arquitetura original e com a versão da IPNoSyS com *software pipelining*. O conjunto de testes realizados corresponde a aplicações sintéticas de repetição, as quais foram desenvolvidas para que pudessem ser realizadas análises com base na comparação dos resultados obtidos com a duas versões da IPNoSyS. As comparações são realizadas em termos do número de ciclos de relógio necessários para a execução da aplicação.

Os testes escolhidos são referentes à análise da solução adotada para a exploração de paralelismo em nível de instruções. Todos os resultados foram obtidos a partir do simulador alto nível desenvolvido para a arquitetura da IPNoSyS. Os testes foram realizados em variações do algoritmo de roteamento e do modo de execução (original e *software pipelining*). Para o modo de execução original foi utilizado o algoritmo de roteamento *Spiral Complement* e para o modo de execução com *software pipelining* os algoritmos de roteamento *Spiral Complement* e zigue zague.

Todas as aplicações utilizadas nas simulações foram desenvolvidas como código objeto, ou seja, cada linha do código representa uma palavra do pacote, incluindo os bits de controle (o que equivale 36 bits). Arquivos de código objeto da IPNoSys utilizam a extensão ".ram", podendo ser descritos em arquivos de texto padrão. Um exemplo de aplicação é ilustrado na Figura 7(a), na qual é mostrado um laço na linguagem "C", e na Figura 7(b), código correspondente convertido na linguagem de pacotes, equivalente à linguagem de montagem: cabeçalhos indicam a rota dos pacotes pelos canais da arquitetura, seguidos pelas instruções e operandos, finalizando com a sinalização de um *flit* "terminador".

Figura 7: Exemplo de Aplicação de repetição: a) Código em alto nível; b) Código no formato de pacote.

Uma das grandes vantagens da arquitetura da IPNoSyS é o poder computacional proporcionado pela concorrência, devido ao emprego de varias unidades funcionais. Além disso, até quatro pacotes podem ser injetados simultaneamente, o que permite explorar TLP. A seguir, serão apresentados casos de testes, implementados para execução sequencial e para a execução paralela de instruções.

5.1. Experimento 1 – Fluxo Único de Execução

Nesse experimento, os pacotes foram injetados a partir da unidade de memória "0" (MAU "0") localizada no canto superior esquerdo, contendo apenas um pacote, com a arquitetura configurada na dimensão 4x4, utilizando os dois tipos de algoritmo de roteamento (*Spiral Complement* e Zigue-Zague) e variando o número de iterações do “laço”. Cada pacote contém dez instruções de natureza lógico/aritmética, de modo que cada instrução leva 1 ciclo de relógio para executar na unidade funcional da RPU.

Figura 8: Desempenho de aplicações na IPNoSyS Original, *Spiral Complement* e Zigue-Zague com um único pacote.

Os resultados mostrados na Figura 8 expressam o aumento de desempenho para as aplicações que executaram na IPNoSyS com *software pipelining*. Alftware pipelining dos na Figura 8 expressam o aumento de desempenho para as aplicaom a arquitetura configura*Spiral Complement*, pois as instruções se cruzam, levando à execução localizada. A partir da execução localizada, as instruções executam sequencialmente. Como o roteamento Zigue-Zague consegue diminuir o número de execuções localizadas, observa-se maior desempenho quando este é utilizado. No roteamento Zigue-Zague, caminhos de uma mesma rota não se cruzam por uma mesma RPU – evitando, assim, o bloqueio de pacotes na arquitetura.

5.2. Experimento 2 – Fluxo paralelo de execução

Para esse experimento, quatro pacotes foram injetados na arquitetura, a partir das unidades de memória: todas as MAUs são simultaneamente utilizadas. A arquitetura foi configurada para a dimensão 4x4, utilizando dois

tipos de algoritmo de roteamento e variando o número de iterações do “laço”. Os resultados são apresentados na Figura 9. É importante perceber que nesses resultados foram explorados simultaneamente paralelismo em nível de instruções (ILP) e em nível de pacotes/tarefas (TLP). Isso permite melhorar o desempenho pela exploração do paralelismo em granularidades diferentes. Nessa versão da IPNoSyS, o modelo de execução usa mais de uma RPU por vez.

Nesse sentido, na Figura 9, pode-se verificar em comparação com os resultados referentes à execução de um único pacote, que a injeção de mais um pacote para execução paralela impacta muito pouco no desempenho. Apenas na arquitetura original verificou-se um acréscimo de 12% em média, o que comprova a adequação da arquitetura para a execução simultânea de pacotes/tarefas. Isto se deve principalmente a duas razões: 1) pacotes seguem rotas distintas pelos canais de comunicação da arquitetura; e 2) a política de roteamento permite que as instruções ao se cruzarem (em uma mesma RPU) utilizem portas de entrada diferentes e por consequência, *buffers* de entrada e portas de saída também diferentes.

Figura 9: Desempenho de aplicações na IPNoSyS Original, *Spiral Complement* e Zigue-Zague com quatro pacotes.

5.3. Experimento 3 – Fluxo paralelo de execução com dependência de dados entre as instruções

Na Figura 10, os resultados com instruções que mantêm dependências de dados entre si são mostrados. Foi injetado 1 pacote com 10 instruções na MAU “0”, variando-se o número de iterações do “laço”.

É sabido da literatura que esse tipo de instrução causa penalidade em *pipelines*, uma vez que a instrução que executa primeiro no *pipeline* precisa gerar os dados a serem utilizadas por instrução(ões) seguinte(s). No entanto, devido ao modelo de execução da IPNoSyS ser orientado ao fluxo de dados, naturalmente cada instrução pode “deixar” dados a serem utilizados por instruções subsequentes no *pipeline* em *buffers* internos às RPUs. Observa-se, no entanto, que os *buffers* internos de cada RPU precisam ser dimensionados de acordo com a quantidade de dados que as instruções geram para as instruções seguintes. O decréscimo verificado no desempenho (em torno de 20%) se deve a um ciclo a mais, o qual é necessário para a instrução escrever no *buffer* o dado a ser utilizado pela instrução seguinte.

Figura 10: Desempenho de aplicações na IPNoSyS Original, *Spiral Complement* e Zigue-Zague de um único pacote com dependência de dados.

6. Conclusões e trabalhos futuros

Neste trabalho, foi apresentada a implementação de uma técnica de paralelismo em nível de instruções em uma arquitetura de processamento não convencional, a qual é aplicada para avaliação de desempenho em nível de instruções e de tarefas.

Conforme os experimentos realizados, os resultados de simulação de diversas aplicações comprovaram a eficiência da técnica, uma vez que significativas melhorias no desempenho puderam ser observadas. Para todos os casos de teste, a IPNoSyS com *software pipelining* obteve tempo menor de execução. O aumento de desempenho obtido na execução das aplicações permite concluir que essa técnica é eficiente mesmo ao ser aplicada em arquiteturas não convencionais. Além disso, permite que sejam explorados diferentes níveis de concorrência: instruções e tarefas.

E outro ponto de vista, verificou-se uma boa adequação da arquitetura à exploração de paralelismo, pois com apenas pequenas modificações arquiteturais foi possível adequar o paralelismo proporcionado pela replicação de elementos de processamento em sua parte operativa, à técnica testada.

Como trabalhos futuros, pretende-se comparar a IPNoSyS com arquiteturas convencionais, em relação ao desempenho e à potência dissipada. Além disso, é importante que sejam adaptadas ferramentas de desenvolvimento, tais como compiladores, para a geração de instruções no formato de pacotes, o que permite uso mais amplo para a arquitetura.

Referências

- [1] ARAÚJO, S. R. F. *Estudo da Viabilidade do Desenvolvimento de Sistemas Integrados Baseados em Redes em Chip sem Processadores: Sistema IPNoSyS*. Dissertação (Mestrado em Sistemas e Computação) – Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, Natal, 2008.
- [2] ARAÚJO, S. R. F. *Projeto de Sistemas Integrados de Propósito Geral Baseado em Redes em Chip - Expandindo as Funcionalidades dos Roteadores para Execução de Operações: A Plataforma IPNoSyS*. Tese (Doutorado em Ciência da Computação) – Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, Natal, 2012.
- [3] CRUZ, M. O. *Roteamento em Zigue-zague para a plataforma IPNoSyS*. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, Natal, 2013.
- [4] DOUILLET, A.; GAO, G.R. *Software-Pipelining on Multi-Core Architectures*. In: 16th International Conference on Parallel Architecture and Compilation Techniques, Romania, Brasov, 2007. p. 39–48.
- [5] JONES, R. B.; VICK, I. H., A. *Software Pipelining: A Comparison and Improvement Microprogramming and Microarchitecture*. In: 23rd Annual Workshop and Symposium, 1999. p. 46 - 56.
- [6] MASAYUKI, K. et al. *Software Pipelining for Jetpipeline Architecture*. In: International Symposium on Parallel Architectures, Algorithms and Networks, Japan, 1977. p. 127 - 134.
- [7] KANDEMIR, f. Li; KOLCU I. *Exploiting software pipelining for network-on-chip architectures*. In: IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, Karlsruhe, 2006.
- [8] PATTERSON, D. A; HENNESSY, J. L. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design.
- [9] CRUZ, M. O; KREUTZ, M. E.; FERNANDES, S. *An experimental evaluation of a combination of features on the IPNoSys*. In: III Simpósio Brasileiro de Engenharia de Sistemas Computacionais, Niterói, 2013. p. 169 - 170.
- [10] ZEFERINO, C. *Redes-em-Chip: arquiteturas e modelos para avaliação de área e desempenho*. Tese (Doutorado em Computação). Departamento de Ciência da Computação da Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.