ESTUDO DA INTEGRAÇÃO DE UM *ARRAY* ADAPTÁVEL EM PROCESSADORES MULTICORE

Francisco Carlos Silva Junior, Ivan Saraiva Silva
Departamento de Computação, Universidade Federal do Piauí (UFPI), Brazil
juninho.ufpi@hotmail.com, ivan@ufpi.eud.br

***Abstract*: Reconfigurable architectures have been successfully used as accelerators on single core processor or as stand-alone computer engine. However, on the multicore era, it is necessary to modify the way as reconfigurable array is used. Traditionally, a set Reconfigurable and Functional Unit are used the same way they are used to single core. Unfortunately, it results on considerable area overhead. Adaptable architectures emerged from the development of reconfigurable architectures. Adaptable architectures are systems able to adapt to applications run on it. This paper proposes and validates an adaptable architecture. A minimal configuration is proposed to accelerate threads from a single core. Also, a configuration to accelerate threads from a multi-core is discussed. Using the minimal configuration the array is able to accelerate 39% the inner loop of a matrix multiplication. Synthetic applications were also used to observe how data dependencies affect the system performance.**

***Keywords:* Reconfigurable Architectures; Reconfigurable Computing; Adaptable Architectures; multicore;**

I. INTRODUÇÃO

Nos últimos anos, o aumento na complexidade das aplicações tem exigido cada vez mais desempenho dos processadores. Via de regra, as aplicações atuais utilizam algoritmos complexos e têm como característica uma computação *data-intensive*. Para tais aplicações, duas abordagens têm sido utilizadas desde o surgimento da tecnologia de desenvolvimento de circuitos integrados. Na primeira abordagem, os processadores de propósito geral (GPP – *General Purpose Processors*) são utilizados para a execução de qualquer aplicação. Na segunda abordagem, hardware dedicado, ou circuitos integrados de aplicação específica (ASIC – *Application Specific Integrated Circuits*) são utilizados para a execução de aplicações também específicas. Nesta abordagem, o ganho de desempenho é obtido por intermédio da especialização das estruturas de hardware. OS GPPs são flexíveis o suficiente para executar qualquer aplicação, entretanto, não conseguem prover o desempenho necessário para lidar com a complexidade crescente das aplicações atuais. Por outro lado, os ASICs oferecem alto desempenho e flexibilidade muito baixa, pois possuem estruturas de hardware específicas para a aplicação para a qual foram desenvolvidas.

As arquiteturas reconfiguráveis (AR), viabilizadas a partir da popularização dos dispositivos programáveis (FPGA – *Field Programmable Gate Array*), emergiram como uma solução arquitetural capaz de unir a flexibilidade dos GPPs com o desempenho dos ASICs [1]. Arquiteturas reconfiguráveis são normalmente constituídas de um *array* de unidades funcionais reconfiguráveis (UFR) ou elementos de processamento (EP), interconectados por intermédio de um subsistema de interconexão. Tanto o desempenho quanto a flexibilidade das ARs são obtidos por intermédio da exploração do paralelismo intrínseco, resultante da disponibilidade de múltiplas unidades funcionais reconfiguráveis.

Apesar das vantagens citadas, a implantação de uma arquitetura reconfigurável pode levar a um considerável aumento da área e do consumo de energia do sistema [1]. Esse aumento se deve ao grande número de recursos computacionais (ULA, multiplexadores, banco de registradores, rede de interconexão, etc) que a arquitetura reconfigurável necessita. Além disto, a exploração dos recursos da AR requer a utilização de algoritmos específicos que realizam o mapeamento da aplicação na arquitetura reconfigurável.

O sucesso obtido pelas arquiteturas reconfiguráveis deu origem ao modelo arquitetural que se convencionou chamar de arquiteturas adaptáveis. As arquiteturas adaptáveis são baseadas em um conjunto de unidades programáveis, interconectadas por um subsistema de interconexão que também é programável. Tais estruturas são exploradas por intermédio de configurações que escalonam no espaço (intra arquitetura), e no tempo, operações a serem realizadas pelas unidades programáveis. Ao contrário das arquiteturas reconfiguráveis, as arquiteturas adaptáveis não modificam sua estrutura de hardware.

Este trabalho propõe e avalia uma arquitetura adaptável para processadores multicore. O *array* proposto tem como objetivo, oferecer recursos computacionais para aceleração de múltiplas *threads* (ou processos), sendo executadas simultaneamente em múltiplos núcleos do processador multicore. Adicionalmente, o *array* deve apresentar-se como uma solução que requer um número significativamente menor de elementos de processamentos (EP) que outras arquiteturas adaptáveis ou reconfiguráveis encontradas atualmente na literatura, como [1] [2] [3]. Este artigo mostrará que a configuração para o *array* adaptável é capaz acelerar o laço principal de uma *thread*, sendo executado em um núcleo MIPS pipeline, em 39%. O artigo mostrará também como evoluir esta configuração, de modo que ela possa ser acoplada à arquitetura de um processador multicore e prover aceleração de múltiplas *threads* executando simultaneamente nos núcleos deste processador. Uma análise da variação da quantidade de Elementos de Processamentos na coluna do *array* também é feita observando a redução de área e o impacto na capacidade do *array* adaptável de explorar ILP.

Este trabalho está organizado em 6 seções. A seção II define o que é arquitetura reconfigurável e como ela é classificada. A seção III mostra algumas arquiteturas reconfiguráveis propostas e destaca a contribuição deste trabalho. A seção IV descreve a arquitetura adaptável proposta neste trabalho. A seção V apresenta os resultados que puderam ser obtidos através da execução de aplicações sintéticas e reais na arquitetura adaptável. O trabalho é finalizado com a seção VI, apresentando as conclusões e os trabalhos futuros.

II. ARQUITETURAS RECONFIGURÁVEIS

Arquiteturas reconfiguráveis são sistemas integrados que permitem customização da unidade de hardware para satisfazer os requisitos computacionais de diferentes aplicações [2]. Tais sistemas integrados são classificados segundo um conjunto de critérios específicos, os principais são: granularidade, acoplamento e mecanismo de reconfiguração.

Granularidade se refere ao tamanho do dado que pode ser operado pelas unidades funcionais reconfiguráveis da AR. Unidade funcional reconfigurável é um bloco lógico cuja funcionalidade pode ser reconfigurada (redefinida). Em arquiteturas de granularidade fina (FGRA – *Fine-Grained Reconfigurable Architectures*), as operações feitas na UFR são a nível de bits e são constituídas por componentes de hardware cuja configuração geram operadores a nível do bit, tais como *flip-flops* e *look-up tables*. Por outro lado, nas arquiteturas reconfiguráveis de granularidade grossa (CGRA – *Coarse-Grained Reconfigurable Architectures*), as operações são feitas no nível de palavras, e as UFRs são constituídas de componentes de hardware mais complexos, tais como as ULAs (Unidades de Lógica e Aritmética), blocos de memória, etc.

Arquiteturas reconfiguráveis são normalmente utilizadas como aceleradores de aplicação. Muitas aplicações, como já demonstrado por Amdahl [4], possuem apenas partes de seu código cuja execução por aceleradores é vantajosa. Assim sendo, ARs são frequentemente acopladas a um processador de propósito geral. Este é o caso das arquiteturas propostas em [2] [3]. O acoplamento indica como a AR se comunica com o GPP. Em arquiteturas fortemente acoplados, a AR é implementada como uma unidade funcional dentro do processador. Nos sistemas fracamente acoplados, a AR é implementada como um co-processador e a comunicação se dá através de um barramento.

O mecanismo de reconfiguração define a forma como múltiplas configurações podem ser geradas e carregadas na AR. Dois mecanismos de reconfiguração são normalmente utilizados: reconfiguração estática e a reconfiguração dinâmica. Na reconfiguração estática, as configurações são geradas, normalmente por um compilador, que identifica as partes da aplicação que podem ser aceleradas pela AR. Na reconfiguração dinâmica, a reconfiguração da AR pode ser efetuada com o sistema em operação. Deste modo é possível que um bloco de hardware, acoplado à AR e ao GPP, gere as configurações em tempo de execução e, eventualmente, carregue uma nova configuração na AR.

Os conceitos aqui apresentados também se aplicam para arquiteturas adaptáveis.

III. TRABALHOS RELACIONADOS

Trabalhos relatando propostas de implementação e uso de arquiteturas reconfiguráveis começaram a surgir de forma mais regular em meados da década de 1990. Desde então, um número considerável de propostas foram publicadas. Dada a amplidão do tema, este artigo abordará um número limitado de arquiteturas.

Feng [1] propôs uma CGRA voltada para Processamento Digital de Sinais (PDS). A arquitetura proposta é composta por 16 elementos de processamento, quatro registradores de configuração e uma rede de interconexão compartilhada. Os elementos de processamento são distribuídos em 4 estágios, cada estágio com 4 elementos de processamento. Segundo os autores, a principal contribuição do trabalho foi a redução do custo de implementação, considerando a área em chip como unidade de medida. A redução da área deveu-se à utilização de um subsistema de interconexão simplificado.

MorphoSys [2] é uma CGRA fracamente acoplada a um processador TinyRisc. A comunicação com o processador é feita através de uma controladora de DMA (*Direct Memory Access*) e um *frame buffer*. A arquitetura é composta de 64 células reconfiguráveis, arranjadas em um *array* 8x8, divido em 4 quadrantes 4x4. O subsistema de interconexão é composto por três níveis hierárquicos, denominados: Conectividade com o vizinho mais próximo, Conectividade intraquadrante e Conectividade interquadrante. Novas instruções foram inseridas no processador TinyRISC para permitir a comunicação e execução de trechos de código. A arquitetura apresenta bom desempenho para segmentos regular de código com computação intensiva, mas requer um custo alto de hardware

Os trabalhos citados anteriormente são voltados para ambientes mono processados (*single-core*), mas arquiteturas reconfiguráveis voltadas para *multicore* têm sido propostas também, como em [5] [6]. Essas arquiteturas visam explorar as vantagens dos processadores multicore e das arquiteturas reconfiguráveis e acelerar as aplicações devido ao ILP e TLP (*Thread Level Paralelism*).

Yan [5] propôs um RMC (*Reconfigurable Multi-Core*) que é acoplado a um *multicore* homogêneo. O RMC é composto por: RPUs (*Reconfigurable Processor Units*), um *crossbar* reconfigurável e um gerenciador de RPUs. O *crossbar* é utilizado na comunicação de um núcleo do processador com uma RPU. Para permitir a execução na RPU, 7 instruções especiais foram adicionadas ao conjunto de instruções do processador. A RPU possui granularidade fina e é composta, principalmente, por: um *array* de CLBs (*Configurable Logic Block*), um LB (*Local Buffer*), um CC (*Configuration context*) e Multiplexadores de Selecão de Configuração. Foram utilizadas as aplicações 3-DES, AES e JPEG_ENC para verificar o desempenho da arquitetura. Resultados de simulações mostraram que a RMC foi capaz de acelerar, em média, 2,34 vezes as aplicações e que o custo adicional de transferência de dados e controle foi aceitável.

Dellagostin [6] propôs um *array* reconfigurável para sistemas multiprocessados, chamado de CReAMS (*Custom Reconfigurable Array for Multiprocessor Systems*) pelo autor. A arquitetura é capaz de explorar TLP replicando as DAPs (*Dynamic Adaptive Processor*). O DAP é um arquitetura reconfigurável de granularidade grossa fortemente acoplada ao processador. A comunicação entre os DAPs é feita através de uma rede 2D-\textit{mesh} utilizando roteamento XY. O DAP é composto de 3 blocos principais: Caminho de dados Reconfigurável, Núcleo do processador e um Hardware de detecção dinâmica de configuração. O caminho de dados

reconfigurável é organizado em uma estrutura de matriz, onde as linhas é o máximo de operações que podem ser feitas em paralelo, enquanto que as colunas limita a quantidade de instruções dependentes que podem ser executadas. Foram geradas versões homogêneas e heterogêneas da arquitetura para verificar o desempenho da arquiteturas nos dois ambientes. Para gerar a versão heterogênea foi gerada DAPs com diferentes quantidade de unidade funcionais. O trabalho mostrou o potencial da CReAMS em sua versão heterogênea e que houve uma melhora no desempenho da arquitetura, porém, ainda não alcançou o ganho desejado pelos autores na maioria das aplicações.

A arquitetura proposta neste trabalho usa menos EPs/Unidade Funcionais que os trabalhos [1] [2] [5] e [6], apenas 5 EPs e 1 unidade de *Load/Store* por núcleo, e possui um subsistema de interconexão simples. Além disso, o sistema se comporta como um sistema heterogêneo, onde a quantidade de EPs utilizado por um determinado núcleo será definido pela configuração que é enviada, o que faz a arquitetura adaptável executar com eficiência mesmo quando os núcleos tem cargas de trabalho diferentes, pois o *array* adaptável de cada núcleo irá se adaptar à carga de trabalho de cada núcleo. Esse comportamento da arquitetura adaptável é alcançada porque não há limitações de níveis de configuração que pode ser executada no *array* adaptável, como se tem em [6].

IV. Arquitetura Proposta

A arquitetura proposta é composta de uma coluna dotada de 5 elementos de processamento (EPs) e uma unidade de *Load/Store* de dois estágios. A arquitetura adaptável é controlada por uma máquina de estados (FSM - *Finite State Machine*) e tem um banco de registradores (BR). O diagrama de blocos da arquitetura pode ser visto na Figura 1.

Os dois estágios da unidade de *Load/Store* (ST1 e ST2) são responsáveis, respectivamente, por: i) calcular o endereço e ii) realização da operação de leitura ou escrita.

No início da execução de uma configuração do *array* adaptável, o banco de registradores recebe o contexto de entrada. O contexto de entrada é a cópia do conteúdo do banco de registradores de um núcleo de execução do processador multicore. Durante a execução de uma configuração, o banco de registradores armazena valores da computação da coluna. E, ao final da execução, o conteúdo do banco de registradores é enviado como resultado da execução da configuração, que então será copiado para o banco de registrador do processador.

Uma configuração define um conjunto de operações que devem ser executadas na coluna de EPs e na unidade de *Load/Store* durante um ou mais ciclos. Assim, uma configuração pode ser dividida em múltiplas palavras de configuração, sendo uma palavra de configuração a definição das operações que serão executadas em um ciclo.

Quando deseja-se iniciar a execução de uma configuração no *Array* Adaptável (AA), uma palavra de configuração é enviada da cache de configuração para o AA. Uma palavra de configuração de coluna deve ser enviada para o AA a cada ciclo. No primeiro ciclo, se a primeira palavra da configuração incluir alguma instrução de *Load* ou *Store*, uma ordem para executar o *Load* ou *Store* será enviada para o primeiro estágio da unidade de *Load/Store* (cálculo do endereço) e a execução das instruções que não são de acesso à memória serão executadas no próximo ciclo. Caso a palavra de configuração não inclua instrução de acesso à memória, um nop (*no operation*) será enviada para o primeiro estágio da unidade de *Load/Store*. Note que, isto faz com que a execução das instruções de *Load/Store* que estão na estágio 1 estejam adiantadas no fluxo de execução. A escrita do contexto de entrada no banco de registradores também ocorre no primeiro ciclo. No segundo ciclo, uma ordem de execução, relativa às instruções que não são de acesso à memória da palavra de configuração do ciclo anterior, é enviada para os EPs. Além disso, uma nova palavra de configuração é recebida da cache de configuração, a instrução de acesso à memória desta palavra de configuração é enviada para o primeiro estágio da unidade de *Load/Store* e a instrução que estava no primeiro estágio da unidade de *Load/Store* avança para o segundo estágio. A partir do final segundo ciclo, quando a primeira palavra de configuração foi executada no *array*, os resultados das computações dos EPs e de um *Load* ou *Store* no estágio 2 serão escritos no banco de registradores, ou na memória, ao final de cada ciclo. A partir do terceiro ciclo, repete-se o envio de novas palavras de configuração semelhantes às utilizadas no segundo ciclo, até que toda a configuração seja executada. Ao final da execução um contexto de saída (resultado) é gerado e enviado aos banco de registradores do processador.

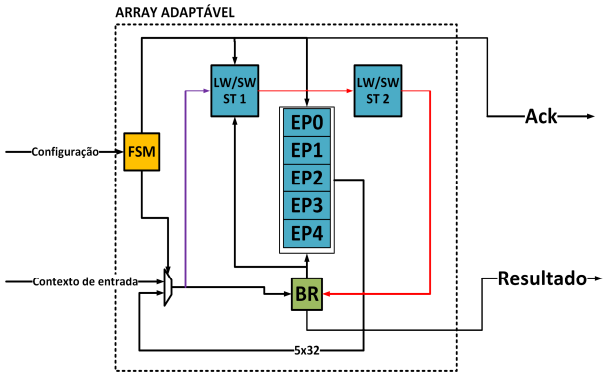

Figura 1 - Visão esquemática da arquitetura proposta.

A. A UNIDADE DE LOAD/STORE

O primeiro estágio da unidade de *Load/Store* é constituído de um somador de 32 bits e uma unidade de *forward*. O somador é utilizado para calcular o endereço de acesso à memória, conforme o padrão de instruções MIPS. Como foi mencionado anteriormente, a execução do primeiro estágio da unidade de *Load/Store* está adiantada no fluxo de execução, em relação às operações (pertencentes a mesma palavra de configuração do *Load* ou *Store*) a serem executadas nos EPs da coluna. Isso faz com que dependência de dados entre duas palavras de configuração executadas em ciclos sucessivos possam causar erros de execução. Para resolver esse problema, uma unidade de *forward* foi inserida para garantir que os dados operados pelo primeiro estágio da unidade de Load/Store estão sempre atualizados, pois pode ocorrer do dado lido do banco de registradores estar desatualizado, caso um dos EPs altere seu valor. A unidade de *forward* compara os registradores de

destino dos EPs com o registrador fonte utilizado no cálculo do endereço da unidade de *Load/Store*, como a unidade apresentada em [7].

A palavra de configuração da unidade de *LW/SW* possui um tamanho de 28 bits, sendo 16 bits para o campo imediato, 10 para mapear o registradores de base do cálculo do endereço e registrador destino e 2 bits para informar a operação (*lw,sw* ou *no operation*)

B. O ELEMENTO DE PROCESSAMENTO

O EP, como pode ser visto na Figura 2, é composto por um registrador de configuração, que contém todas as informações da palavra de configuração do EP, dois multiplexadores e uma ULA. A palavra de configuração armazenada no registrador de configuração define as entradas da ULA e sua operação. As ULAs realizam todas as operações lógicas e aritméticas necessárias à execução de instruções MIPS. A multiplicação, entretanto, por implicar na necessidade de inclusão de um operador crítico em termos de área, foi incluída somente em uma ULA, a ULA do EP0.

A palavra de configuração do elemento de processamento possui um tamanho de 36 bits, sendo 16 bits para o campo imediato, 15 bits para mapear os registradores rs,rd e rt e 5 bits de operação da ULA.

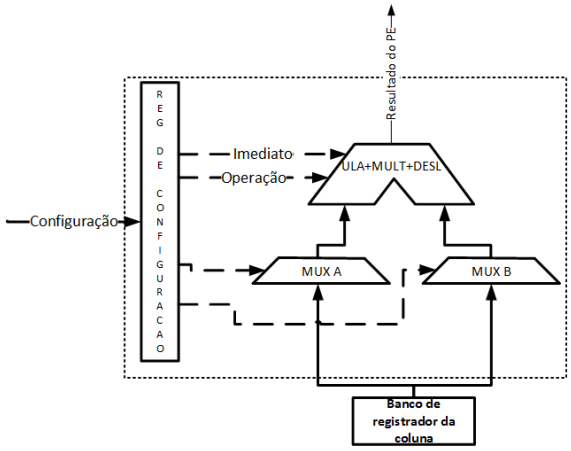

Figura 2 - Arquitetura do Elemento de Processamento.

C. INTEGRAÇÃO COM UM PROCESSADOR MULTICORE

O *array* adaptável será integrado a um processador multicore com 5 núcleos. Cada núcleo é um MIPS pipeline de 5 estágios como o descrito em [8]. Cada núcleo terá acoplado a ele um *array* adaptável, juntamente com uma cache de configuração. O *array* adaptável será fortemente acoplado ao núcleo MIPS pipeline, uma vez que ele precisa ter acesso ao banco de registradores do processador. A cada núcleo do processador será acoplado um *array* adaptável, como o apresentado na Figura 3. A integração do *array* adaptável com os 5 núcleos do processador multicore pode ser visto na Figura 4. Assim, em um processador multicore de 5 núcleos será necessário a utilização de um *array* de 5 colunas.

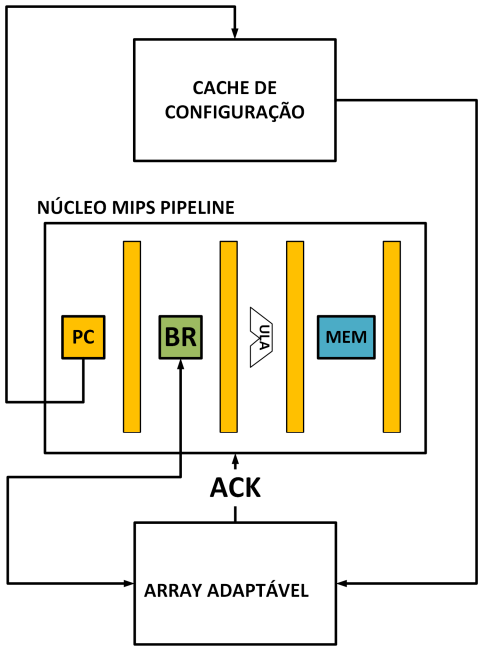

Figura 3 - Diagrama de blocos da integração do *array* adaptável com núcleo multicore.

As configurações serão geradas através de um bloco de hardware (tradutor binário) acoplado á cache de configuração e ao processador. O tradutor binário gera configurações em tempo de execução. Esta configuração é gerada através da análise, do programa/*thread* em execução, feita pelo tradutor binário, que verifica trechos de códigos candidatos a serem executados no *array* adaptável. Esse mecanismo de reconfiguração já foi proposto em [9].

A cache de configuração é necessária para armazenar as configurações geradas pelo tradutor binário e não foi implementado ainda.

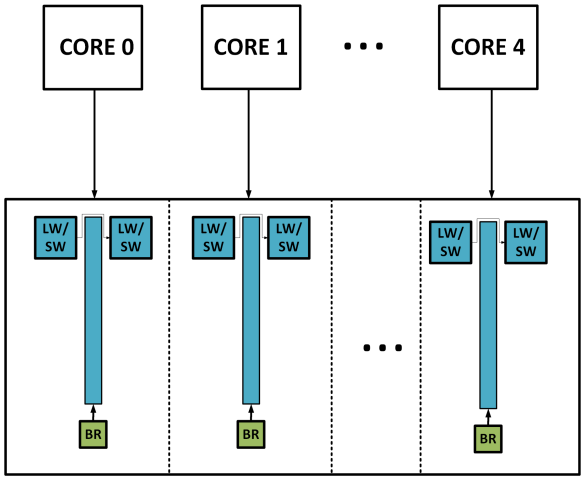

Figura 4 - Integração do processador multicore com o *array* adaptável.

O *array* adaptável proposto neste artigo traz como uma das vantagens, quando comparados com os outros trabalhos

encontrados na literatura, o fato de possuir somente uma coluna. Normalmente, os *arrays* adaptáveis são compostos por várias colunas e um sistema de interconexão que permite a comunicação entre os diferentes níveis de coluna. Sendo assim, o uso destes *arrays* adaptáveis em multicore, disponibilizando um *array* para cada núcleo, implicaria em um grande aumento da área do sistema. Utilizando o *array* adaptável proposto neste artigo, consegue-se amortizar esse aumento na área do sistema que se dá através da inserção de um *array* adaptável ao multicore.

V. Resultados Experimentais

Para validar a implementação da arquitetura proposta e avaliar o desempenho que ela oferece, aplicações sintéticas foram geradas e executadas em uma implementação VHDL do *array* adaptável. De modo a permitir a avaliação do efeito da dependência de dados no desempenho, cada aplicação sintética é constituída de 100 instruções e diferentes porcentagens de dependência de dados foram utilizadas. A porcentagem de dependência de dados indica quantas das 100 instruções da aplicação são dependentes entre si. Para a geração das aplicações a dependência de dados variou de 10% a 100%, com variação de 10 em 10.

Duas ferramentas foram desenvolvidas em Java para geração das aplicações sintéticas. A primeira ferramenta é utilizada para gerar as aplicações e suas configurações, enquanto que a segunda é utilizada gerar os binários a partir das configurações. A geração das aplicações sintéticas é feita a partir de dois parâmetros de entrada. O primeiro informa o número de instruções presentes na aplicação e o segundo informa a porcentagem de instruções dependentes. Como as aplicações são geradas de forma aleatória, garantindo apenas o número de instruções e a porcentagem de dependência, para cada porcentagem adotada, 30 versões de cada aplicação foram geradas.

A execução das aplicações na implementação VHDL do *array* adaptável foi simulada com o auxílio da ferramenta ModelSim da Altera [10]. Os resultados das simulações das aplicações sintéticas, sendo executadas podem ser vistos na Figura 5. Como esperado, com o aumento da dependência de dados o desempenho obtido piora, sendo necessário mais ciclos para executar a mesma aplicação. Isso se deve ao fato de que aplicações com maior dependência de dados, além de dificultar a exploração do paralelismo, geram configurações com profundidades maiores. No melhor caso, 0% de dependência de dados, as 100 instruções são executadas em 21 ciclos. No pior caso, 100% de dependência de dados, as 100 instruções são executadas em 101 ciclos.

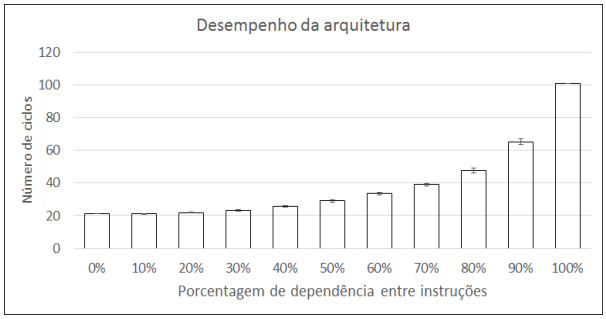

Figura 5 - Variação do desempenho da arquitetura à porcentagem de dependência.

Para verificar a exploração do paralelismo de uma aplicação real e simular uma execução de uma configuração do *array* reconfigurável acoplado a um multicore, destacou-se um trecho de código de uma multiplicação de duas matrizes 20x20. Essa multiplicação de matriz foi paralelizada. A paralelização foi feita dividindo a multiplicação em 5 *threads*, onde cada *thread* multiplica 4 linhas da matriz. Como para cada núcleo haverá uma coluna do *array* adaptável, as configurações das *threads* serão executadas paralelamente nos 5 *arrays* adaptáveis. O código da aplicação foi escrito em C e compilado para MIPS com o CrossCompiler GNU [11]. A configuração para o *array* adaptável foi gerada manualmente, analisando o assembly gerado e as dependências de dados. O trecho de código destacado corresponde a execução do laço mais interno da aplicação e possui 49 instruções com 65% de dependência de dados. Apesar do trecho apresentar 65% de dependência, a configuração gerada teve profundidade 39, isso se deve ao grande número de instruções de *load/store*, que também influencia na profundidade da configuração. A simulação da execução deste trecho de código no *array* adaptável levou 40 ciclos. O mesmo trecho de código foi também executado em uma implementação VHDL do processador MIPS, em sua versão pipeline. Detalhes da implementação do processador podem ser encontrados em [8]. A execução do trecho de código no processador MIPS pipeline levou 66 ciclos, o que mostra que a CGRA é capaz de acelerar este trecho de código em 39%.

O trecho de código destacado corresponde à operação de multiplica e acumula. Para cada elemento da matriz resultante, são feitas 20 somas, pois a matriz é quadrada e tem 20 colunas, então, cada elemento da matriz resultante necessitará executar esse trecho de código 20 vezes. Como a multiplicação foi dividida em 5 *threads*, cada *thread* irá multiplicar 80 elementos da matriz, portanto, isso faz com que o trecho de código seja executado 1600 (20x80) vezes, ao final da execução de cada *thread*.

As aplicações sintéticas também foram executadas na implementação VHDL do processador MIPS pipeline. Os resultados desta execução foram comparados com o melhor e pior caso da execução das aplicações no *array* adaptável. A comparação pode ser vista na Figura 6. Como se pode notar, mesmo com 100% de dependência de dados, onde é executada apenas uma instrução por coluna do *array* adaptável, e não se tem nenhum paralelismo entre as instruções, o *array* adaptável apresenta desempenho um pouco melhor que o processador MIPS pipeline. Para dependência de dados variando entre 10% a 60%, a aceleração no pior caso fica entre 60% e 78%. No melhor caso a aceleração fica entre 72% e 79%.

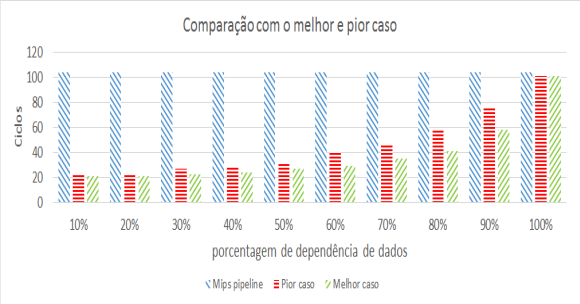

Figura 6 - Comparação do desempenho em pipeline MIPS com o melhor e pior caso no *array* adaptável.

Para realizar a análise de área, foram geradas duas versões adicionais em VHDL do *array* adaptável, uma com 4 EPs e outra com 3 EPs na coluna do *array*. Para realizar a comparação de área foi considerado o número de elementos lógicos para cada implementação do *array* adaptável. O número de elementos lógicos foi obtido através de resultados de síntese para a tecnologia FPGA com o auxílio da ferramenta *Quartus* II da Altera [10].

Os resultados de área podem ser vistos na Tabela 1. A remoção de um EP resulta em uma redução de 16% de elementos lógicos, aproximadamente. Porém, a remoção de EPs reduzem a capacidade de exploração de ILP do *array* adaptável e, consequentemente, pode vir a diminuir o desempenho do arquitetura. Essa redução da capacidade de exploração de ILP se dar de uma forma linear: dada uma versão do nosso *array* adaptável com *n* EPs, o máximo de ILP que ela consegue explorar é *n+1*, já que tem também uma unidade de LW/SW além dos EPs. Para o trecho de multiplicação que foi selecionado para verificar o ILP, o máximo de ILP obtido na configuração foi 2, ou seja, o desempenho seria o mesmo da versão original do *array* adaptável, porém utilizando menos elementos lógicos. Essa relação ILP x EPs mostra a necessidade de um estudo sobre o ILP obtidos em aplicações reais de modo que se possa escolher uma versão do *array* adaptável que maximize o uso de seus EPs e minimize o uso de elementos lógicos.

Table I - Styles

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

VI. Conclusões e Trabalhos Futuros

Muitas CGRAs têm sido propostas com o objetivo de fornecer recursos de hardware para aceleração de aplicações, contudo, de um modo geral, o *array* reconfigurável da arquitetura adaptável é, via de regra, constituído de um significativo número de elementos de processamento. A arquitetura proposta apresenta um modelo que usa menos elementos de processamento que os outros trabalhos já publicados. O sistema de interconexão proposto é bastante simples, constitui-se apenas de um banco de registradores que armazenam as computações da coluna do *array*. Além disso, a arquitetura é capaz de executar configurações com profundidade infinita, funcionando como um pipeline superescalar infinito.Como trabalho futuro, pretende-se integrar essa arquitetura ao caminho de dados do processador multicore. Neste caso, para cada núcleo de processamento uma coluna do *array* adaptável deve ser integrada. Um tradutor binário em hardware também deve ser integrado a arquitetura do par (Núcleo de Processamento; *array* adaptável). Este tradutor permitirá a geração de configurações em tempo de execução. As ferramentas já desenvolvidas devem evoluir de modo a se tornar um compilador capaz de compilar aplicações escritas em C para execução no processador multicore munido de um *array* adaptável. A ideia é que a arquitetura seja mista, podendo ser configurada tanto estaticamente quanto dinamicamente. Finalmente, um estudo de exploração de paralelismo de aplicações reais será realizado, de modo a permitir uma melhor avaliação da arquitetura proposta.

References

[1] C. Feng and L. Yang, "Design and evaluation of a novel reconfigurable alu based on fpga," in *Mechatronic Sciences, Eletric Engineering and Computer (MEC), Proceedings 2013 International Conference on*, Dec 2013, pp. 2286-2290.

[2] H. Singh, M. H. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Chaves Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on*, vol. 49, no. 5, pp. 465-481, May 2000.

[3] J. K. Paek, K. Choi, and J. Lee, "Binary acceleration using coarse-grained reconfigurable architecture," *SIGARCH Comput. Archit. News*, vol. 38, no. 4, pp. 33-39, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1926367.1926374

[4] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM,1967, pp. 483-485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[5] L. Yan, B. Wu, Y. Web, S. Zhang, and T. Chen, "A reconfigurable processor architecture combining multi-core and reconfigurable processing unit," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, June 2010, pp. 2897-2902.

[6] J. D. Souza, L. Carro, M. B. Rutzig, and A. C. S. Beck, "Towards a dynamic and reconfigurable multicore heterogeneous system," in *2016 Brazilian Symposium on Computing Systems Engineering*, Nov 2014, pp. 73-78.

[7] D. A. P. E. J. L. Henessy, *Computer Organization and design the hardware/software interface, 5th ed.* Oxford, USA: Morgam Kaufmann, 2014.

[8] F. C. J. Silva and I. S. Silva, "Designing a complete pipelined datapath to mips isa: Learning in pratice," in *Sforum 2014, Chip in Aracaju, 2014.*

[9] A. C. S. B. Filho, "Transparent reconfigurable architecture for heterogeneous applications," Ph.D. dissertation, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, junho 2008.

[10] Quartus, "Altera. Quartus ii handbook version 13.1 , volume 3," http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf, 2016, acessado em 14 de fevereiro de 2016.

[11] I. Cpmware, "Building a gnu gcc cross compile," www.cpnware.com/Docs/BuildingGcc.pdf, 2016, acessado em 14 de fevereiro de 2016.