Scratchpad Memory Management Using Data-Prefetching

Ivan Saraiva Silva
Department of Computing
Federal University of Piauí
Teresina, Brazil
ivan@ufpi.edu.br

Hildebrando Segundo
Computer Science Graduate Program
Federal University of Piauí
Teresina, Brazil
hildebrando@ufpi.edu.br

***Abstract*— Nowadays, multicore processors including dozens of cores are a reality and the challenge of designing microprocessors with hundreds of cores does not appear far from being achieved. However, some questions remain open, despite the proposals found in the literature. Memory latency, data locality and programmability are some of the more critical ones. As the number of cores in multi-core processor architectures increases, so does the need of more efficient memory hierarchy and resource management scheme. The design of efficient memory hierarchy and memory management policy becomes a very important research subject in multi core architecture field. The usage of resource allocation strategies becomes increasingly necessary. This paper presents the design and implementation of a multi-core node with support to memory configuration and hardware-based scratchpad memory management. The node consists of 9 cores (8 slave cores and a master core) and a configurable local memory. The proposed architecture was implemented with VHDL and prototyped with FPGA development boards. Experimental results show that the proposed hardware-based scratchpad memory management unit can achieve 99,0 of hits in data address prediction and 97,1% of hit when searching data-frames in the local memory.**

Keywords—Multicore; Scratchpad Memory; Memory Configuration; Data-Prefetching;

I. INTRODUÇÃO

Nos últimos anos o aumento do número de núcleos de processamento em um único chip tornou-se a chave para o sucesso da industria de microprocessadores. Neste período, por razões tecnológicas, o foco mudou do continuo aumento da frequência do relógio para o aproveitamento do continuo aumento do número de transistores. Segundo esta estratégia, em 11 anos a esta industria foi capaz de evoluir, do desenvolvimento e comercialização do microprocessador POWER4 [1], o primeiro microprocessador multicore comercialmente disponível, (174 milhões de transistores), para o microprocessador Xehon Phi, que integra 10 núcleos de processamento e requer aproximadamente 5 bilhões de transistores [2]. Arquiteturas de hardware tais como Tile64 [3] e Cell Processor [4] integram 64 e 9 núcleos de processamento, respectivamente. Observando estes avanços percebe-se que a estratégia adotada pela industria converteu-se em absoluto sucesso tecnológico. De fato, do ponto de vista da industria a adoção de das arquiteturas multicore foi necessária para manter o continuo aumento de desempenho oferecido a seus clientes e para garantir os investimentos que permitem o continuo aprimoramento da tecnologia.. Entretanto, nesta área muitos problemas ainda necessitam de soluções tecnológicas ou arquiteturais adequadas.

O desempenho dos microprocessadores historicamente evolui mais rapidamente que o desempenho dos sistemas de memória. Tipicamente, o desempenho dos microprocessadores evolui em 50% ao ano, enquanto o desempenho das memória evolui apenas entre 9% e 20% no mesmo período [5]. Este problema, que tornou-se conhecido como *memory wall*, foi originalmente anunciado para ilustrar o descompasso entre o crescimento do desempenho dos microprocessadores e as memórias DRAM [6]. Vinte anos depois, já na era dos microprocessadores multicore, é possível observar que o desempenho de aplicações tradicionais e emergentes da computação de alto desempenho, degrada quando o número de núcleos de processamento aumenta [7, 8]. Uma solução para este problema seria o desenvolvimento de hierarquias de memória específicas para arquiteturas multicore. Tais hierarquias devem prover maior integração entre a memória e os núcleos de processamento. Também é necessário integrar recursos que ofereçam suporte de hardware para o gerenciamento da memória.

O problema que ficou conhecido como *power wall* [9] também está associado com a maior disponibilidade de transistores, portanto com a maior disponibilidade de núcleos de processamento nas modernas arquiteturas multicore. Considerando o que se convencionou chamar de *orçamento energético* (ou *de potência*) do projeto, alguns autores têm considerado que em um futuro próximo: (i) se torne impossível integrar todos os transistores que a tecnologia oferece para uma dada área de silício [10] e; (ii) microprocessadores não conseguirão utilizar todos os seus núcleos ao mesmo tempo, na frequência máxima permitida pelo projeto [11]. Em sistemas embarcados em particular, as limitações associadas ao contínuo suprimento de energia, tornará impossível prover energia para alimentar todos os transistores ao mesmo tempo. Os problemas mencionados aqui (*power wall* e *memory wall*) estão diretamente relacionados, sabe-se, por exemplo, que aproximadamente 40% do consumo energético em sistemas embarcados deve-se aos acessos à memória [12]. Soluções para estes problemas envolvem o uso

Este trabalho é parcialmente suportado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPQ.

978-1-4673-9143-6/15/$31.00 ©2015 IEEE

de memória local de rascunho. Memórias locais de rascunho mais eficientes tanto do ponto de vista energético quando do ponto de vista do desempenho, pois não incluem lógica para verificação da disponibilidade do dado e geração dos sinais pertinentes (*Hit* e *Miss*).

A previsibilidade e escalabilidade do desempenho oferecido são as maiores vantagens associadas ao uso das memórias locais de rascunho [13], entretanto, elas necessitam ser explicitamente gerenciadas. O acesso à memória é normalmente controlado pelo programador ou pelo compilador, isto devido a ausência de lógica de gerenciamento do conteúdo da memória em tempo de execução[14]. A alocação de dados em memórias locais de rascunho pode ser estática ou dinâmica. Na alocação estática o compilador o compilador gera informação sobre os dados que precisam ser pré-carregados. Na alocação dinâmica, um mecanismo de transferência de dados em tempo de execução para a memória local de rascunho precisa ser oferecido.

Este artigo apresenta o projeto e implementação de microprocessador multicore dotado de memória local de rascunho configurável e gerenciável com o auxílio de recursos de hardware. Isto significa que o microprocessador suporta a configuração da memória local utilizada por cada tarefa em execução, bem como provê recursos de hardware que auxiliam a alocação e transferência de dados em tempo de execução. A arquitetura proposta usa mecanismos de pré-busca de dados e translação de endereços como métodos de gerenciamento e acesso a memória local de rascunho.

O artigo está organizado como segue. Na seção 2 alguns trabalhos relacionados são apresentados e discutidos. A seção 3 descreve a arquitetura proposta, enquanto a seção 4 apresenta e discute os experimentos realizados e os resultados obtidos. Finalmente, a última seção traz as conclusões e uma breve discussão sobre trabalhos futuros.

II. TRABALHOS RELACIONADOS

Um número consideravelmente grande de publicações recentes abordam o uso e gerenciamento de memória local de rascunho (*Scratchpad Memory* - SPM). A maioria destas publicações abordam algoritmos, métodos e técnicas de compilação para dar suporte à alocação de dados. Em [15] é possível encontrar a apresentação de um método utilização de memórias locais de rascunho baseado reuso de dados. Também é possível encontrar uma relativamente extensa análise de que abordam os mais variados aspectos do uso e exploração de SPM, publicados na década de 2000. Do mesmo modo, em [16] é possível encontrar uma extensa lista de trabalho que analisam às SPMs sob o ponto de vista da previsibilidade de tempo de execução e análise de pior caso.

Outra abordagem com base na qual as SPMs são frequentemente analisada é o que se convencionou chamar de *software-cache* ou *software-managed cache*. Neste caso, as SPMs são tratadas como memória cache cujo gerenciamento é realizado por software. Neste contexto, em [17] encontra-se a proposta de uma SPM que é gerenciada como *software-cache* multidimensional. As múltilas dimensões são utilizadas para melhor tratar as peculiaridades de aplicações multimídia.

Até onde foi possível verificar, os trabalhos [18] e [19] são os que melhor se relacionam com a proposta apresentada neste artigo. Em [18] uma memória local de rascunho e usada para armazenar os dados do conjunto de trabalho das aplicações, mantendo a consistência entre o espaço de endereçamento lógico e o espaço de endereçamento físico. Uma unidade de hardware é então proposta para realizar operação de acesso a memória (*Load* e *Store*) com previsibilidade de tempo de execução. Em [19] propõe-se e implementa-se uma unidade de hardware cujo objetivo é gerenciar os acesso (*Load* e *Store*) à memória local de rascunho garantindo a previsibilidade do tempo de acesso. Em um estudo de caso, a unidade proposta é integra a um microprocessador Microblaze. Neste dois trabalhos [18] e [19] a unidade de gerenciamento da memória local de rascunho realiza operações de translação de endereço físico em endereços lógicos como parte da estratégia de gerenciamento.

A abordagem apresentada neste artigo difere das abordagens apresentadas em [18], [19] em quatro aspectos fundamentais: (i) A unidade de gerenciamento proposta neste artigo foi desenvolvida para lidar com os requisitos de acesso à memória encontrados em arquiteturas multicore, deste modo é necessário que requisições de duas ou mais tarefas sejam atendidas; (ii) A memória local proposta neste artigo pode ser adequadamente configurada de modo a possibilitar sua utilização, de forma concorrente, por múltiplas tarefas de múltiplas aplicação, (iii) o gerenciamento da SPM é baseada em técnicas de previsão de endereços de acesso à memória, e; (iv) a unidade de gerenciamento da SPM proposta neste artigo é responsável unicamente pela previsão de endereços, outras unidades da arquitetura realizam a translação de endereços e a transferência propriamente dita.

III. A ARQUITETURA PROPOSTA

A arquitetura proposta e composta por um núcleo mestre, oito núcleos escravos, uma unidade de gerenciamento da memória (*Memory Management Unit* - MMU), duas unidades de acesso direto à memória (*Direct Access Memory* - DMA), uma memória local de rascunho e uma memória externa. A Figura 1 apresenta uma visão esquemática da arquitetura proposta.

A unidade de gerenciamento da memória (MMU) é responsável pela translação de endereços e pela configuração da memória local de rascunho. A MMU inclui uma unidade de gerenciamento da memória local de rascunho (*Scratchpad memory Managemente Unit* - SPMU), que é responsável pela alocação de espaço na memória local de rascunho para a execução das tarefas. As duas unidades de acesso direto à memória (DMA) são responsáveis pela transferência de dados. O DMA_1 realiza as transferências memória externa e a memória do núcleo mestre, enquanto que o DMA_2 realiza as transferências entre a memória externa e a memória local de rascunho.

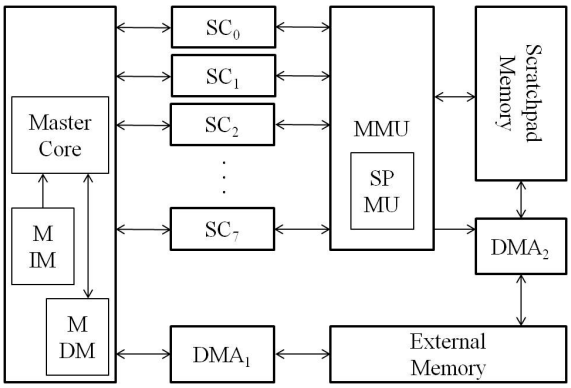

Fig. 1. Arquitetura proposta para o microprocessador multicore

O núcleo mestre possui memórias privadas de instrução e dados e é responsável pela execução de tarefas de gerenciamento da arquitetura e pela distribuição de tarefas para os núcleos escravos. Os núcleos escravos são responsáveis unicamente pela execução das tarefas e têm acesso direto exclusivamente a memória local de rascunho. Tanto o núcleo mestre quanto os núcleos escravos são microprocessadores MIPS. O microprocessador MIPS utilizado foi entretanto modificado com a adição de um subconjunto de instruções utilizadas para configuração e alocação de espaço na memória local de rascunho, bem como para a realização de troca de mensagens e sincronização inter-processos.

A arquitetura proposta foi projetada de modo a também possibilitar sua utilização como um nó de processamento em um multiprocessador baseado em redes em chip (*NoC-Based Multiprocessor*). Para tanto, os módulos DMA_1 e DMA_2 e o modulo *External Memory*, da figura 1, devem ser substituídos por uma interface de redes em chip.

A. Hierarquia de Memória e Configuração

A arquitetura proposta utiliza dois níveis de hierarquia de memória. O primeiro nível é a memória externa e o segundo nível é a memória local de rascunho. A princípio não há restrição quanto ao tamanho relativo destas duas memórias, entretanto, a unidade de transferência entre eles deve ser fixa. A unidade de transferência é idêntica a menor porção alocável na memória local de rascunho (a partir deste pondo neste artigo denominada como *Allocation Frame* - AL). Na implementação proposta neste artigo a unidade de transferência escolhida foi de 32 palavras (128 bytes). A memória local de rascunho foi desenvolvida com 128 *Allocation Frames* (ALs), ou seja, 16 K bytes. A memória externa utilizada foi a memória SRAM de 2 MB disponível na placa de desenvolvimento DE2-115.

A configuração da hierarquia de memória é realizada pelo núcleo mestre para cada tarefa a ser executada pelos núcleos escravos. Três instruções foram adicionadas ao conjunto de instrução do núcleo mestre para o processo de configuração da memória. As instruções adicionadas foram:

- **SETID**: Esta instrução define identificadores para a aplicação e tarefa a ser executada. Alocações de memória são realizadas para uma aplicação e tarefa específica;
- **REQMEM**: Esta instrução é utilizada para alocar um número inteiro de allocation frames para a aplicação e tarefas atuais (definidas pela instrução SETID);
- **RELEASE**: Esta instrução é utilizada para liberar toda a memória alocada para a aplicação e tarefas atuais;

Quando a instrução REQMEM é executada, a MMU seleciona na memória local de rascunho o número de *allocation frames* solicitados, e os aloca para a aplicação e tarefa atuais. A alocação consiste em criar em uma tabela de translação de endereços (*Translation Table* - TT) uma associação entre os IDs da aplicação e da tarefa e os *allocation frames* da memória local de rascunho. Acessos futuros à memória, realizados por aquela aplicação e tarefa, implicam na necessidade de transladar o endereço requisitado para um endereço específico em um *allocation frame* na memória local de rascunho.

B. Acesso a Memória e Translação de Endereços

A translação de endereços é realizada na MMU com o auxilio da *Translation de Table* (TT). A TT é composta de 128 entradas, cada uma delas referente a um *allocation frame* da memória local de rascunho. Cada entrada da TT contem: O endereço de uma porção de dados na memória da memória externa, de tamanho idêntico a um *allocation* frame (a partir deste ponto neste artigo, esta porção de dados na memória principal será identificada como *Data Frame*); O ID da aplicação e; O ID da tarefa. A TT é capaz de atender a dezesseis requisições simultâneas, cada uma delas oriunda de um núcleo escravo.

Quando uma requisição de acesso à memória chega, a TT verifica se há alocação de memória para os IDs informados. Se a alocação existir a TT verifica se o *data frame* solicitado está na armazenado na memória local de rascunho (ou seja, verifica se este *data frame* foi previamente movido da memória externa para a memória local de rascunho). Dois tipos de erros podem ocorrer durante o processo de translação:

1. ***Allocation fault***: Este erro ocorre quando não há alocação de memória para os IDs informados, ou seja, quando a instrução REQMEM não foi executada previamente;
2. ***Frame fault***: Este erro acontece quando, apesar da alocação, o *data frame* solicitado não está presente na memória local de rascunho;

O primeiro erro é fatal e encerra a execução da tarefa. O segundo erro gera um pedido de configuração do DMA_2 para a transferência do *data frame* da memória externa para a memória local de rascunho.

Quando o processo de translação não gera erros, ou seja, quando há alocação de espaço e o *data frame* foi previamente transferido, a MMU realiza a operação de acesso à memória (leitura ou escrita). A unidade de gerenciamento da memória local de rascunho (*Scratchpad Memory Management Unit* - SPMU) foi projetada com o objetivo de minimizar o número

de *frame faults*. A próxima seção apresenta a arquitetura da SPMU.

IV. A SPMU

A unidade de gerenciamento da memória local de rascunho (SPMU) apresentada neste artigo utiliza técnicas de pré-busca de dados para prever quais *data frames* serão utilizados em um futuro próximo. A arquitetura adotada usa recursos semelhantes aos apresentados em [20] para manter o histórico do fluxo de execução das tarefas em execução. Nesta abordagem, duas tabelas, a *Branch Target Buffer* (BTB) e a *Frame Predict Buffer* (FPB), são utilizadas para prever os endereços alvo dos futuros acessos à memória.

A BTB usa métodos de previsão de desvios para manter o histórico do fluxo de execução. Cada entrada da BTB armazena informações sobre cinco blocos básicos a serem executados. Como pode ser visto na figura 2, para economizar bits na implementação da BTB, o endereço alvo (*Target*) de um desvio condicional foi também usado como endereço inicial (*PCBegin*) do próximo bloco básico.

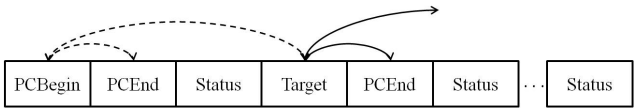

Fig. 2. Entrada da *Branch Target Buffer* (BTB).

A partir do inicio da execução de uma tarefa (primeiro bloco básico) a BTB é dinamicamente inicializada e atualizada. Cada entrada reflete a probabilidade de tomada de cinco desvios no fluxo de execução. Em outras palavras, cada bloco básico acessível devido a uma falha na verificação da condição de desvio inicia uma nova entrada na BTB. O campo *Status* indica a previsão de desvio, assim, tem-se:

- Se a previsão for pela tomada do desvio, o endereço alvo do fluxo de execução está no próximo campo *Target* da tabela e o fluxo ainda é previsto pela entrada atual;
- Se a previsão for por não tomar o desvio, o endereço alvo é obtido realizando a operação "*PCEnd + 4*" e uma nova entrada da tabela (indexada pelo valor "*PCEnd + 4*") prevê o fluxo de execução.

A FPB mantém informações sobre as instruções de acesso à memória para calcular os futuros endereços alvo. Na ISA MIPS as instruções de acesso à memória seguem os padrões apresentados em (1) e (2)

$$\text{Lw \$s1, Imm (\$s2)} \quad (1)$$

$$\text{Sw \$s1, Imm (\$s2)} \quad (2)$$

onde Lw e Sw são mnemônicos para *Load Word* e *Store Word*, respectivamente, $s1 e $s2 são registradores do banco de registradores do microprocessador MIPS e *Imm* é um valor imediato. Assim sendo, as entradas da FPB devem manter informações sobre o valor do registrador $s2, em tempo de execução, bem como deve registrar o valor imediato *Imm*. Nos dois casos apresentados em (1) e (2), o endereço alvo do acesso à memória é calculado pela adição do conteúdo do registrador $s2 com o valor do campo imediato. As entradas da FPB foram definidas como mostra a figura 3,

PCBegin	PC Instruction	Reg. Value	Offset	Immediate	Prediction	Accuracy

Fig. 3. Entradas da *Frame Predict Buffer* (FPB).

Na figura 3 os campos *RegValue* e *Offset* indicam, respectivamente, o valor do registrador utilizado na instrução de acesso a memória e de quanto ele é modificado a cada execução do bloco básico. O campo *Immediate* indica o valor imediato codificado na instrução e o campo *Prediction* indica o valor previsto para o endereço alvo na próxima execução. O campo *Accuracy* indica a precisão da previsão. O valor do campo *Accuracy* é binário indicando apenas erro ou acerto.

A. A Arquitetura da SPMU

Para inicializar e atualizar a BTB e FPB é necessário que a SPMU acompanhe o fluxo de execução dos núcleos de processamento, mantendo informações sobre instruções (incluindo endereços e operandos imediatos). Assim, é necessário que a SPMU receba dos núcleos escravos informações tais como:

- Conteúdo do registrador Contador de Programa (PC);
- Conteúdo do Registrador de Instruções (IR);
- Conteúdo dos registradores utilizados no cálculo de endereços de acesso à memória de dados;

O conteúdo do PC é utilizado como indexador de acesso tanto à BTB quanto FPB. Os valores do PC que interessam à BTB são os endereços iniciais de blocos básicos e os endereços das instruções de desvio condicional ou incondicional. Endereços iniciais de blocos básicos são os endereços seguintes às instruções de desvio e o endereço da primeira instrução executada de uma tarefa. Na FPB os endereços de interesse são os endereços das instruções de acesso a memória. O conteúdo do IR é utilizado para a decodificação da instrução em execução. A figura 4 mostra uma representação esquemática da SPMU e sua conexão com a *Translation Table*.

Na figura 5 é possível observar que a operação da SPMU inicia na unidade de decodificação da instrução (*Instruction Decode Unit* - IDU). Esta unidade é responsável pela decodificação das instruções em execução nos núcleos escravos e pela identificação de endereços de início e fim de bloco básico. A cada instrução executada pelo núcleo escravo, a IDU informa tanto a BTB quanto a FPB, seu tipo e endereço. No caso da execução de instruções de acesso a memória, a IDU também informa o registrador usado para o cálculo de endereço. Para instruções de desvio condicional o estado da condição de desvio (Taken/notTaken) também é informado.

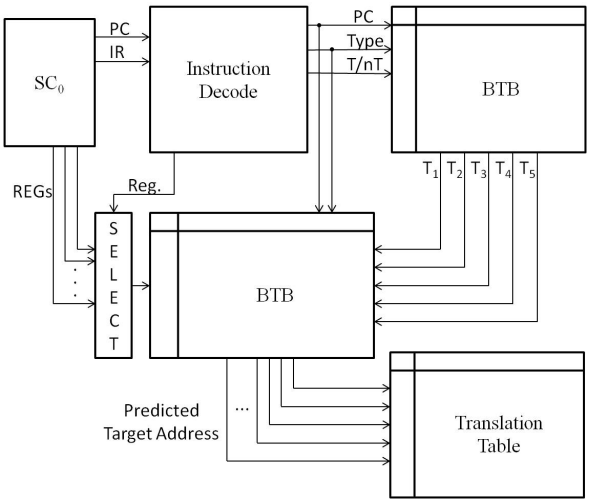

Fig. 4. Arquitetura da SPMU

A BTB é ativada quando a entrada *Type* indica que o valor do PC corresponde a um endereço de início de bloco básico ou de fim de bloco básico (instruções de desvio). Nestes casos a BTB usa as informações recebidas da IDU para inicializar ou atualizar suas entradas. Quando o endereço corresponde a um início de bloco básico a BTB gera os endereços de cinco blocos básicos seguintes no fluxo de execução e os envia à FPB. Quando o endereço corresponde a uma instrução de desvio (fim de bloco básico) a BTB atualiza a previsão de desvio (campo *Status*, ver figura 2), utilizando a informação da entrada *Taken/notTaken*, e gera endereços de cinco novos blocos básicos se o fluxo de execução foi alterado.

A FPB é ativada quando a entrada *Type* indica que o valor do PC corresponde a uma instrução de acesso à memória ou quando a BTB gera endereços de blocos básicos. Quando uma instrução de acesso à memória é executada a FPB inicializa ou atualiza a entra correspondente aquela instrução. Quando a FPB recebe endereços de blocos básicos, endereços alvo de acesso à memória são gerados e enviados à *Translation Table*.

Endereços alvo de acesso à memória são gerados com a indexação da FPB com cada um dos endereços de blocos básicos e leitura do campo *Prediction* (ver figura 3). Para reduzir o tamanho e a complexidade da FPB, na implementação apresentada neste artigo a FBP foi limitada à geração de dois endereços alvo de acesso à memória por bloco básico. Assim o tamanho total da FPB é de oitenta entradas.

O tamanho da BTB deve ser suficiente para guardar informações sobre todos os blocos básicos das aplicações e tarefas. Também para limitar o tamanho e a complexidade de implementação a BTB implementada foi limitada a oitenta entradas. Como cada entrada pode guardar informações de até 5 blocos básicos tem-se a possibilidade de guardar informações de 400 blocos básicos no total.

V. EXPERIMENTOS E RESULTADOS

A arquitetura apresentada neste artigo foi inteiramente descrita com o auxílio da linguagem VHDL. A descrição VHDL foi utilizada para geração de um protótipo do microprocessador multicore (tal como apresentado na figura 1). Para a prototipagem foi utilizada a placa de desenvolvimento DE2-115, dotada de um FPGA da família Cyclone IV da Altera. O núcleo mestre foi desenvolvido com memórias de instruções e dados separadas, cada memória com 8 K Bytes (4 K palavras). A memória local de rascunho foi desenvolvida com 128 *Allocation Frames* (ALs), ou seja, 16 K bytes. A memória externa utilizada foi a memória SRAM de 2 MB disponível na placa de desenvolvimento DE2-115. Mesmo com as limitações de memória, imposta pela limitações do FPGA utilizado, a arquitetura desenvolvida é complemente funcional.

Esta seção apresenta os resultados de implementação. Também apresenta e discute os resultados de avaliação de desempenho da SPMU, obtidos por intermédio da execução de um conjunto de aplicações.

A. *Resultados de Implementação*

Para a síntese da descrição VHDL foram utilizadas as ferramenta da Altera [21]. O FPGA usado foi o dispositivo EP4CE115F29C7 da família Cyclone IV, também da Altera. Os resultados da síntese para os blocos mais importantes da arquitetura podem ser vistos na tabela I.

TABLE I. IMPLEMENTATION RESULTS.

Unidade	Tamanho (em LE)	Frequência máxima	Memória (em Bytes)
MIPS	2225	79,00 MHz	-
MMU	4.247	97,25 MHz	5.230
Memória Local	-	-	16.384
Memória Externa	-	Board limited	2 M
Multicore	32.480	67,93 MHz	28.672

B. *Desempenho das Aplicações*

Para avaliar o desempenho da arquitetura proposta, em particular da SPMU, um conjunto de aplicações foi desenvolvido e executado no protótipo. A execução no protótipo é mais rápida que a simulação da descrição VHDL, entretanto, a análise detalhada dos resultados é praticamente impossível. Considerando esta dificuldade um conjunto de monitores de hardware foi desenvolvido e integrado à arquitetura. Os monitores mediram o desempenho da SPMU em particular e o tempo de execução em ciclos.

O conjunto de aplicações é composto por: Três aplicações do *benchmark* ParMiBench [22]; Multiplicação de Matrizes; Ordenação com *Bubble Sort*; Ordenação com *Merg Sort* e; Alinhamento de sequência genéticas com o algoritmo *Smith and Waterman*. As três aplicações de ParMiBench foram *Bitcont*, *Susan* e Stringsearch. Considerando as limitações de memória do protótipo, principalmente no que se refere à memória local de rascunho, o conjunto de dados utilizado foi limitado. Na tabela 2 podem ser encontrados os resultados de avaliação de desempenho. A coluna "*#Cores*" indica o número de núcleos usados para a execução do benchmark. A coluna

"*Address Hit*" indica o percentual de acerto da previsão de endereços de acesso a memória. A coluna "*Branch Predict. Hit*" indica a taxa de acertos da previsão de desvio associada ao usa da BTB. A coluna "*Efect. Hit*" leva em consideração os erros de previsão de endereço alvo de acesso a memória e os erros de previsão de desvio e mostra os acertos efetivos de previsão de endereço de acesso à memória. Por fim, a coluna "*Exec. Cicles*" mostra o tempo de execução dos benchmarks em ciclos.

TABLE II. RESULTADOS DE DESEMPENHO DA SPMU

Benchmarks	**Resultados de Desempenho**				
	# Cores	***Address Predict. Hit (%)***	***Branch Predict Hit (%)***	***Efect. Hit (%)***	***# Exec. Cycles***
Bit Count	4	99,3	76,6	76,1	75.671
Susan	4	97,5	92,1	89,8	1.537.913
Stringsearch	8	96,8	89,9	87,0	786.185
Matrix Multi.	8	99,0	98,1	97,1	1.102.927
Quick Sort	4	98,9	85,4	84,5	90.693
Bubble Sort	4	94,5	93,1	88,0	200.436
Smith & Waterman	8	91,0	72,6	66,1	131.586

Os resultados da tabela 2 mostram que as aplicações mais regulares tiraram maior proveito da SPMU. Entretanto, mesmo a aplicação menos regular, o algoritmo *Smith and Waterman* para alinhamento local de sequências genéticas, ainda obteve um considerável taxa *hit* quando da previsão de endereços de acesso à memória. Observa-se também que os erros de previsão de desvio foram consideravelmente maiores que os erros de previsão de endereços. Isto se deve ao algoritmo de previsão de desvio usado que foi bastante simples.

Um processador multicore com mesma arquitetura, mas sem a inclusão da unidade de gerenciamento da memória local de rascunho também foi desenvolvido e prototipado. Nesta implementação as unidades de DMA foram utilizadas para transferência de data frames da memória externa para a memória local de rascunho quando *frame faults* eram detectados. As mesmas aplicações foram executadas neste segundo protótipo e os tempos de execução foram comparados com aqueles apresentados na tabela II. A figura 5 mostra tempo de execução obtido com os dois protótipos. É possível observar que com a SPMU as aplicações foram executadas mais rapidamente que sem a SPMU. Os resultados mostram que com a SPMU o tempo de execução é 25,8% menor no melhor caso (*Bit Count*) e 9,8% menor no pior caso (*Quick Sort*).

Comparações com o estado da arte são bastante difíceis, senão impossíveis, devido ao fato da única equipe que trabalha com implementação em hardware de unidade de gerenciamento de memória local de rascunho, ter focado seu trabalho em análise de pior caso e previsibilidade de tempo de acesso e execução. Em todo caso, este artigo é, até onde se pode verificar, o único a propor mecanismos de pré-busca de dados e translação de endereços como métodos de gerenciamento e acesso a memória local de rascunho em arquiteturas multicore.

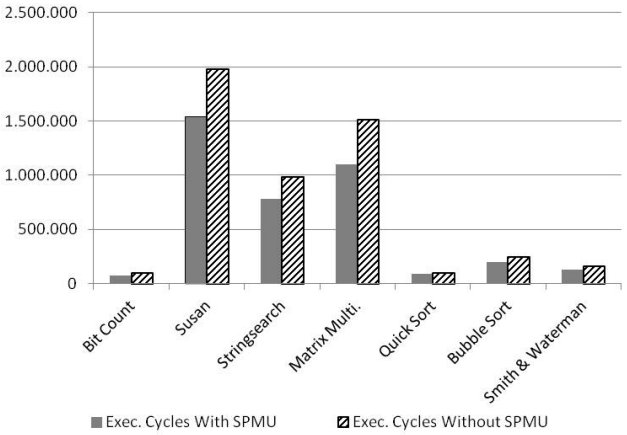

Fig. 5. Comparação do tempo de execução com e sem SPMU

VI. CONCLUSÕES E TRABALHOS FUTUROS

Este artigo mostrou a arquitetura, implementação e avaliação de desempenho de uma arquitetura multicore dotada de unidade de hardware responsável pelo gerenciamento da memória local de rascunho. O trabalho inovador no que se refere a utilização de previsão de endereços de acesso à memória como método de gerenciamento.

Resultados experimentais mostraram que a unidade pode oferecer bom desempenho no que diz respeito à predição de endereços de acesso à memória. Mostraram também que foi possível reduzir o tempo de execução das aplicações em até 29%, quando comparado com os tempos obtidos na mesma arquitetura sem o gerenciamento da memória.

Como trabalho futuro pretende-se experimentar outros algoritmos de predição de desvio e desenvolver uma implementação em SystemC para a realização de experimentos com mais aplicações e conjunto de dados mais significativos.

References

[1] William Stallings, *Computer Organization and Architecture*, 7th Edition, 2007.

[2] Intel, "Intel Delivers New Architecture for Discovery with Intel® Xeon Phi™ Coprocessors". Intel. Retrieved March 29, 2014, from ewsroom.intel.com/community/intel_newsroom.

[3] Bell, S.; Edwards, B.; Amann, J.; Conlin, R.; Joyce, K.; Leung, V.; MacKay, J.; Reif, M.; Liewei Bao; Brown, J.; Mattina, M.; Chyi-Chang Miao; Ramey, C.; Wentzlaff, D.; Anderson, W.; Berger, E.; Fairbanks, N.; Khan, D.; Montenegro, F.; Stickney, J.; Zook, J. TILE64 - Processor: A 64-Core SoC with Mesh Interconnect, *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International* , vol., no., pp.88,598, 3-7 Feb. 2008

[4] Kahle, J.; Day, M.; Hofstee, H.; Johns, C.; Maeurer, T.; Shippy, D. Introduction to the Cell Multiprocessor. IBM Journal of Research and Development 49(4) (2005) 589-604.

[5] Byna, S; Sun, X – H.; and Chen, Y. Taxonomy of Data Prefetching Strategies for Multicore Processors. Journal of Computer Science and Technology (JCST), Volume 24, Number 3 / May, pp. 405-417, 2009

[6] Wulf, W. A. and McKee, S. A. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News* 23, 1 (March 1995), 20-24.

[7] Murphy, R. On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance. IEEE 10th International Symposium on Workload Characterization, pp. 35 – 43, Sept. 2007.

[8] Moore, S. K. Multicore is bad news for supercomputers. IEEE Spectrun, vol. 45, issue 11, pp. 15, November 2008.

[9] Kuroda, T. CMOS design challenges to power wall. Microprocesses and Nanotechnology Conference, 2001 International , vol., no., pp.6,7, Oct. 31 2001-Nov. 2 2001

[10] Esmaeilzadeh, H.; Blem, E.; St.Amant, R.; Sankaralingam, K.; Burger, D. Dark silicon and the end of multicore scaling. Computer Architecture (ISCA), 2011 38th Annual International Symposium on , vol., no., pp.365,376, 4-8 June 2011

[11] Eiblmaier, M.; Mao, R.; Wang, X. Power Management for Main Memory with Access Latency Control. Fourth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks.pp. 22 – 29, April 16, 2009, San Francisco, California, USA.

[12] Ahn I.; Goundling N.; Sampson, J.; Venkatesh, G.; Taylor M. and Swanson S. Scaling the Utilization Wall: The Case for Massively Heterogeneous Multiprocessors. CS2009-0947, September 3, 2009

[13] Kachris, C.; Nikiforos, G.; Papaefstathiou, V.; Xiaojun Yang; Kavadias, S.; Katevenis, M., Low-latency explicit communication and synchronization in scalable multi-core clusters," Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on , vol., no., pp.1,4, 20-24 Sept. 2010

[14] Wehmeyer, L.; Marwedel, P., Influence of memory hierarchies on predictability for time constrained embedded software. Design, Automation and Test in Europe, 2005. Proceedings , vol., no., pp.600,605 Vol. 1, 7-11 March 2005

[15] Cong, J., Huang, H., Liu C., Zou, Y. A reuse-aware prefetching scheme for scratchpad memory. In Proceedings of the 48th Design Automation Conference (DAC '11). ACM, New York, NY, USA, 960-965.

[16] Whitham, J.; Audsley, N. Studying the Applicability of the Scratchpad Memory Management Unit. Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE, vol., no., pp.205,214, 12-15 April 2010

[17] Azevedo, A., Juurlink, B. A Multidimensional Software Cache for Scratchpad-Based Systems. In S. Virtanen (Ed.), Innovations in Embedded and Real-Time Systems Engineering for Communication pp. 59-78. Hershey, PA.

[18] Whitham, J. Audsley, N. Implementing time-predictable load and store operations. In Proceedings of the seventh ACM international conference on Embedded software (EMSOFT '09). ACM, New York, NY, USA, pp. 265-274.

[19] Whitham, J. Audsley, N. The Scratchpad Memory Management Unit for Microblaze: Implementation, Testing, and Case Study. Technical Report YCS-2009-439, University of York, 2009.

[20] Panda, R.; Gratz, P.V.; Jimenez, D. A., "B-Fetch: Branch Prediction Directed Prefetching for In-Order Processors," Computer Architecture Letters , vol.11, no.2, pp. 41,44, July-Dec. 2012

[21] Altera Corporation. Design Entry and Synthesis. (2014). Retrieved March 29, 2014, from http://www.altera.com/products/software/quartus-ii/subscription-edition/design-entry-synthesis/qts-des-ent-syn.html.

[22] Iqbal, S. M. Z.; Liang, Y.; Grahn, H. ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems; Computer Architecture Letters , vol.9, no.2, pp. 45,48, Feb. 2010

[23] Smith, T. F. and Waterman, M. S., Identification of common molecular subsequences. J Mol Biol, 147(1):195-197, March 1981