REVISTA DA ACADEMIA DE CIÊNCIAS DO PIAUÍ

ATHENA: Um Acelerador Reconfigurável, Dinâmico e Autônomo para Aplicações de Propósito Geral

ATHENA: A Reconfigurable, Dynamic and Autonomous Accelerator for General Purpose Applications

Francisco Carlos Silva Junior[1], Ivan Saraiva Silva[2] e Ricardo Pezzuol Jacobi [3]

[1] Universidade de Brasília; juninho.ufpi@hotmail.com; ORCID: 0000-0003-1867-8744
[2] Universidade Federal do Piauí; ivan@ufpi.edu.br; ORCID: 0000-0002-5705-6932
[3] Universidade de Brasília; jacobi@unb. br; ORCID: 0000-0002-4520-7641

Resumo: Arquiteturas reconfiguráveis têm sido amplamente utilizadas como aceleradores de propósito geral provendo melhoria no desempenho e na eficiência energética no sistema. Contudo, esses ganhos vêm ao custo de um significante custo adicional de área devido à grande quantidade de unidades funcionais que geralmente são utilizada nesse tipo de acelerador, o que pode tornar proibitivo sua integração em sistemas embarcados que possuam restrições fortes de área, como os dispositivos móveis e *IoT* (*Internet of Things)*, por exemplo. Com intuito de reduzir o custo adicional inserido pela CGRA no sistema, este artigo propõe a ATHENA (***A THin rEcoNfigurable Array***). A ATHENA mapeia dinamicamente trechos das aplicações para serem executadas na CGRA. O mapeamento é feito através de um hardware dedicado (gerador dinâmico de configuração) que é implementador em cinco estágios de pipeline. A computação na ATHENA pode ser distribuída temporalmente, permitindo que as unidades funcionais sejam reutilizadas mapeando operações para uma mesma unidade em diferentes ciclos. A ATHENA foi implementada no simulador gem5 e avaliada utilizado o benchmark *mibench*. A ATHENA também foi sintetizada utilizando o software de síntese da cadence com a tecnologia de 45 nm. Os resultados mostram que a ATHENA foi capaz de acelerar em até 1,72x enquanto economiza 37% adicionando apenas 2,4% de custo adicional em um processador superescalar *8-wide*. Dessa forma, a ATHENA se mostra com uma solução promissora para dispositivos móveis e embarcados.

Palavras-chave: Acelerador reconfigurável**.** Arquitetura reconfigurável. Sistemas embarcados.

Reconfigurable architecture has been successfully used as general-purpose accelerators providing improvement in both performance and energy efficiency. However, these gains come at the cost of a significant area overhead due to the large number of functional units that are generally used in the CGRAs, which can make its integration in embedded systems with hard area constraints prohibitive. In order to reduce the area overhead of CGRAs, this

work proposes ATHENA (A THin rEcoNfigurable Array). ATHENA dynamically maps the applications' kernels to be executed on the CGRA. The mapping is done through dedicated hardware (dynamic configuration generator), which is implemented in a five-stage pipeline. ATHENA distributes its computation over time, allowing functional units to be reused by mapping operations to the same unit, but in different cycles. ATHENA is implemented in the gem5 simulator and uses the mibench benchmark to evaluate the proposed system. ATHENA was also synthesized using the cadence synthesis tool with 45 nm technology. The results show ATHENA was able to accelerate up to 1.72x while saving 37% in energy, on average, by adding only 2.4% of area overhead on an 8-wide superscalar processor. Thus, ATHENA is a promising solution for mobile and embedded devices providing performance and energy improvements at low area cost.

Keywords: Reconfigurable Accelerator. Reconfigurable Architecture. Embedded systems.

1 Introdução

A crescente complexidade das aplicações tem exigido cada vez mais desempenho dos processadores. Avanços tecnológicos e microarquiteturais eram os principais responsáveis pela significante melhoria de desempenho a cada geração de processador. Melhorias microarquiteturais como *pipelining,* organização superescalar e unidades de predição de salto complexas permitiam ganhos de desempenho através da exploração de ILP (*Instruction Level Parallelism)* que anteriormente não eram possíveis. No entanto, devido a limitações de potência (NAFFZIGER, 2005), os ganhos no desempenho devido ao aumento da frequência de relógio e do aumento da densidade do circuito desaceleraram (HENNESSY,2011). A alternativa encontrada pela indústria para continuar provendo melhorias no desempenho dos processadores foi a adoção de processadores *multicore*.

Além da demanda por desempenho, com a popularização de dispotivos móveis e de novas tecnologias alimentadas por bateria (*smartphones, smartwatches, IoT – Internet of Things*), a energia se tornou um requisito importante no projeto de um sistema computacional. Dessa forma, os sistemas atuais requerem desempenho e eficiência energética. Para lidar com esses requisitos, tradicionalmente, tem-se utilizado aceleradores dedicados, também conhecidos como ASIC (*Application-Specific Integrated Circuit).* Os ASICs implementam um algoritmo em hardware dedicado para a aplicação alvo e, dessa forma, oferece uma solução ótima em termos de energia e desempenho através da especialização de

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

sua estrutura de hardware. Contudo, as aplicações atuais dos sistemas embarcados possuem diferentes demandas e estão em constante modificação. Com isso, reprojetar um ASIC para cada uma atualização de uma aplicação ou projetar um novo ASIC para cada nova aplicação ou funcionalidade se torna muito custo. Por este motivo, os projetistas de hardware têm buscado soluções mais flexíveis.

As arquiteturas reconfiguráveis de granularidade grossa (CGRAs – *Coarse-Grained Reconfigurable Architectures*) surgiram como uma solução arquitetural que visa acelerar aplicações e oferecer ganhos energéticos em relação à execução no processador de propósito geral (PPG). As CGRAs, diferentemente dos ASICs, proveem programabilidade através da reconfiguração de suas unidades funcionais reconfiguráveis.

Um dos grandes desafios enfrentados pelas CGRAs é a redução no custo de área que elas causam no sistema devido ao uso de dezenas de unidades funcionais para explorar o paralelismo das aplicações, algo que se agrava ainda mais se for considerado um ambiente multicore. Com a ideia principal de reduzir o custo adicional em área causada pelas sem degradar o desempenho e a a eficiência energética que pode ser provida pelas CGRAs, este trabalho propõe a ATHENA (***A THin rEcoNfigurable Array***). A redução na área na ATHENA foi obtida através do uso de menos unidades funcionais e de um modelo de execução diferente das CGRAs tradicionais que permite a reutilização das unidades funcionais. Além disso, foi constatado que com poucas unidades funcionais, a ATHENA consegue prover melhorias em desempenho e energética. Isso mostra que as CGRAs composta de muitas unidades funcionais podem estar subutilizando seus recursos.

Este trabalho está organizado em 6 seções. Na seção 2 é apresentado o estado da arte das arquiteturas reconfiguráveis. Na seção 3 a arquitetura ATHENA é explicada em detalhes. Na seção 4 os resultados da arquitetura ATHENA são avaliados. Por fim, a seção 5 conclui o trabalho e discute trabalhos futuros.

2 Trabalhos Relacionados

As primeiras CGRAs surgiram nos anos 90 (TESSIER,2015). Desde então, muitas CGRAs têm sido propostas, como pode ser verificado em diversos *surveys* (WIJTVLIET,2016), (LIU, 2019) e (PODOBAS,2020). Devido ao grande número de trabalhos na área, esta seção se limitará a apresentar as propostas que são diretamente correlatas com a arquitetura ATHENA.

Inicialmente, quando as primeiras CGRAS foram propostas, seu uso era quase que totalmente voltado ao meio acadêmico. Um dos principais motivos que

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

tornava o uso das CGRAs complicado era a falta de ferramentas para automatizar o uso das CGRAs e que utilizasse padrões já reconhecidos na indústria (WIJTVLIET,2016). Visando mitigar esse problema, várias CGRAs focaram em desenvolver um compilador para a CGRA que eram propostas, como PADDI (CHEN,1992), rDPA (HARTENSTEIN,1995), MATRIX (MIRSKY,1996) e muitas outras CGRAs. Com isso, o mapeamento na CGRA era feito de forma automatizada, em tempo de compilação, e o não havia necessidade de programar a CGRA pelo programador. Contudo, essa abordagem possuía algumas desvantagens: i) maior *time-to-market*, pois há a necessidade do desenvolvimento de compilador especial; ii) A manunteção é muito custosa, pois modificações na CGRA implicam em modificações no compilador e iii) falta de compatibilidade de software, pois para códigos antigos usufruírem dos benefícios da CGRA ele deve ser recompilado.

O *Warp processor* (LYSECKY,2004) foi um dos primeiros trabalhos a propor um mecanismo para mapeamento dinâmico de trechos para executar em uma arquitetura reconfigurável. Nessa abordagem, um hardware dedicado para esse propósito é adicionado ao sistema para realizar a detecção e mapeamento de trechos a serem executados na arquitetura reconfigurável. As principais vantagens dessa abordagem são: i) poder utilizar de informações que são disponibilizadas somente em tempo de execução para otimizar o mapeamento e ii) mapeamento de forma transparente de operações para execução na CGRA sem necessidade de reconfiguração ou qualquer modificação no binário da aplicação. O *Warp processor* é composto de dois processadores: um para execução da aplicação e outro para execução o algoritmo que realiza o mapeamento dinâmico. Além desses processadores, a arquitetura conta com uma FPGA para executar as regiões críticas, que são limitadas a laçõs de repetições.

CCA (CLARK,2004), diferentemente do *warp processor,* propôs uma CGRA que podia ser mapeamenda tanto estaticamente (via compilador) quanto dinamicamente. A CGRA possui um formato triangular para execução de subgrafos *dataflow*, possuindo até 7 linhas composta de unidades funcionais heterogêneas. O algoritmo de mapeamento dinâmico utiliza uma heurística gulosa e foi baseado no *framework* replay (PATEL,2001).

DIM (*Dynamic Instruction Merging*) (BECK,2008) propõe um CGRA fortemente acoplada a um processador MIPS pipeline e um tradutor binário. O tradutor binário (TB) é um bloco de hardware capaz de gerar configurações em tempo de execução a partir da execução das aplicações no processador e, portanto, realiza o mapeamento dinâmico na CGRA. A CGRA proposta é organizada em uma estrutura de matriz de unidades funcionais, onde cada instrução é alocada em uma interseção entre uma linha e uma coluna. Instruções alocadas em uma mesma linha são executadas em paralelo, enquanto instruções

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

alocadas em linhas diferentes são executas de forma sequencial. No entanto, execução de forma sequencial não significa em ciclos diferentes, pois os EPs utilizadas na CGRA conseguem executadas até 3 instruções dependentes em um único ciclo.

Diferente do DIM que implementa um hardware dedicado, o TB, para realizar o mapeamento dinâmico das operações na CGRA, o DynaSpaM (LIU,2015) modificou o escalonador de instruções de um processador superescalar para também alocar operações em uma CGRA. A CGRA utilizada no DynaSpaM é inspirada no *PipeRench* (GOLDSTEIN,1999) e possui 32 unidades funcionais. Além disso, a arquitetura conta com um detector de *traces*, um hardware para selecionar os trechos que devem ser mapeados na CGRA.

Assim como o *Warp processor,* o DORA (WATKINS,2016) utiliza um co-processador dedicado para realizar o mapeamento das operações para a arquitetura reconfigurável. No entanto, ao invés de utilizar uma FPGA, o DORA utiliza a CGRA DySER (GOVINDARAJU,2012) como arquitetura reconfigurável.

Mais recentemente, as aplicações de *deep learning* têm se tornado bastante popular em diversas áreas, como visão computacional, reconhecimento de voz etc. CGRAs com foco na aceleração desse tipo de aplicação têm sido propostas (CHEN,2016)(TU,2017)(LIANG,2018)(MÁRIO,2020). Nessa abordagem, a flexibilidade da CGRA é mais limitada para poder atender aos padrões de computação apresentados nessas aplicações. Outra área que tem sido explorada recentemente pelas CGRAs é a de *approximate computing* (AKBARI,2018)(AKBARI,2019)(ZERVAKIS,2020), que troca a precisão da computação em aplicações que suportam imprecisões por desempenho e eficiência energética.

Assim como a grande maioria das CGRA mencionadas nesta seção, a ATHENA propõe uma CGRA que é dinamicamente reconfigurada. De maneira similar ao DIM e ao CCA, A ATHENA utiliza um algoritmo guloso para, dinamicamente, mapear instruções para ser executadas na CGRA através do gerador dinâmico de configuração. No entanto, diferentemente de todas as CGRAs apresentadas, que distribuem sua computação espacialmente, a ATHENA faz uma distribuição tanto espacial quanto temporal da computação para reduzir o custo de área da CGRA no sistema. A distribuição espacial ocorre quando operações são escalonadas para execução em paralelo em um mesmo ciclo de relógio. Por outro lado, a distribuição temporal, que contribuiu para a redução de área da CGRA proposta, permite que as unidades funcionais sejam reutilizadas através do tempo alocando-se operações em uma mesma unidade, mas em ciclos diferentes.

3 A Arquitetura ATHENA

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

A arquitetura ATHENA (***A THin rEcoNfigurable Array***) é composta por cinco componentes principais: o gerador dinâmico de configuração (GCD), a cache de configuração, a CGRA ATHENA, o controlador de configuração e o processador de propósito geral. Uma visão geral do sistema reconfigurável proposto pela ATHENA pode ser visto na Figura 1. Cada um desses componentes é detalhado nas próximas subseções.

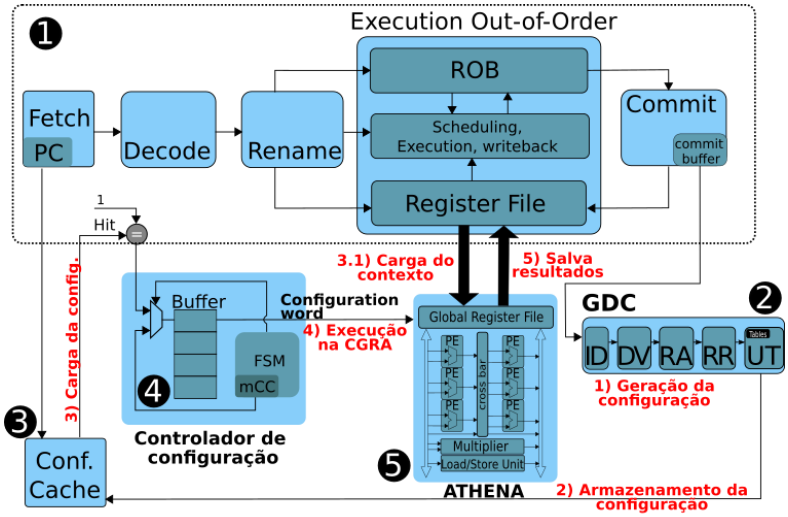

FIGURA 1. VISÃO GERAL DO SISTEMA PROPOSTO.

3.1 O Gerador Dinâmico de Configuração

O Gerador dinâmico de configuração (GCD), bloco 2 na Figura 1, dinamicamente mapeia uma sequência de instruções para serem mais tarde executadas de forma mais eficiente na ATHENA. O algoritmo guloso utilizado no GDC, que escalona as operações para serem executadas o mais cedo possível, foi baseado no DIF (*Dynamic Instruction Formatting*) (NAIR, 1997) e é bastante semelhante ao aplicado ao tradutor binário do DIM (BECK, 2008) e ao escalonador dinâmico do CCA (CLARK,2004).

O GCD é responsável por gerar configurações para a ATHENA em tempo de execução. Para que isto seja possível, ela avalia cada instrução no *commit buffer* do processador de propósito geral. Sendo assim, a entrada para o GCD é um conjunto de instruções que representa um fluxo de execução que foi enviado ao *commit buffer* do processador. Para o GCD gerar um mapeamento e, consequentemente, uma configuração, uma série de etapas são realizadas, como verificação de suporte da operação pela ATHENA, verificação de dependência de dados e verificação de recursos disponíveis na ATHENA.

O GCD foi implementado em um pipeline de 5 estágios, sendo eles: ID (*Instruction Decode*), DV (*Dependency Verification*), RA (*Resource Allocation*), RR

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

(*Register Renaming*) e TU (*Table Update*). Esse pipeline é acoplado ao estágio de *commit* do processador superescalar (ver Figura 1). A comunicação entre o estágio de *commit* do processador e o GDC é fetia através de um *buffer* adicionado ao estágio de *commit,* chamado de *commit buffer*. O estágio de *commit* ao finalizar a execução de uma instrução, armazena essa instrução no *commit buffer* e essa instrução é depois consumida pelo GDC para geração da configuração. O nome de cada estágio descreve bem o que cada estágio do GDC desempenho. No estágio ID, a instrução é decodificada e seus registradores alvos e operandos e o tipo da instrução são retornados. No estágio DV, é feita a verificação de dependência de dados entre as instruções previamente alocadas e a instrução que está sendo alocada. O estágio RA realiza a alocação de uma unidade funcional na ATHENA para a instrução que está sendo processada. O estágio RR faz a renomeação de registradores para eliminar falsas dependências do tipo WAW (w*rite after write*) e WAR (*write after read*). E, por fim, o estágio TU atualiza as tabelas que são utilizadas para manter o estado da configuração que está sendo gerada.

Todos esses passos são repetidos até ser encontrada uma condição de encerramento da configuração. Uma configuração pode ser encerrada por três motivos: i) instrução não suportada pela ATHENA; ii) falta de recursos na ATHENA e iii) a quantidade máxima de blocos básicos que podem ser agrupados em uma configuração foi atingido. Em todos os casos, a configuração é encerrada e armazenada na cache de configuração.

A cache de configuração é responsável por armazenar as configurações geradas pelo GDC. As configurações são salvas e indexadas pelo endereço de PC (*Program Counter*) da primeira instrução que foi mapeada naquela configuração. A cache de configuração utilizada neste trabalho possui 128 entradas e é associativa por conjunto de 4 vias.

3.2 A Arquitetura Reconfigurável ATHENA

A ATHENA é um acelerador reconfigurável de propósito geral que visa prover aceleração e economia energética em processadores superescalar com um baixo custo de área adicional. Para reduzir o custo em área, foi adotado um modelo de execução similar aos *arrays* lineares, onde as unidades funcionais são reutilizadas através da alocação de operações em diferentes ciclos na mesma unidade funcional.

A ATHENA tem três tipos de unidades funcionais: i) elementos de processamento (EPs); ii) multiplicador e iii) unidade de *load/store*. Um banco de registradores global (BRG) é utilizado na ATHENA para realizar a comunicação entre instruções dependentes que estão alocadas em ciclos diferentes. Além disso, o BRG serve também para realizar a comunicação com o processador de propósito geral. Uma rede *crossbar* é utilizada para rotear os dados entre os EPs em diferentes colunas em um mesmo ciclo.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

Os EPs estão organizados em duas colunas, pois cada EP possui a latência de metade do ciclo de relógio do processador. Cada EP possui uma unidade lógica e aritmética (ULA) e suas entradas vêm do BRG ou de outro EP através da rede *crossbar*. A ULA utilizada no EP é mais simples que a ULA utilizada pelo processador de propósito geral, contando somente as operações mais comuns nos *kernels* das aplicações. Por conta disso, a ATHENA pode executar até duas operações dependentes em um único ciclo.

A unidade de *load/store* leva dois ciclos para executar uma operação de acesso à memória em caso de cache *hit.* Caso um cache *miss* aconteça, a ATHENA fica aguardando até o cache *miss* ser resolvido para que sua execução continue. Por fim, o multiplicador leva três ciclos para realizar uma multiplicação.

3.3 O Controlador de Configuração

O Controlador de configuração gerencia a execução na ATHENA e é responsável por enviar as palavras de configuração para a ATHENA. Ele possui um *buffer* de 4 entradas, uma máquina de estados finito e um contador de micro configuração (mCC). O *buffer* armazena temporariamente configurações que foram encontradas na cache de configuração, mas ainda não chegaram ao estágio de *commit* do processador.

A máquina de estados finito possui quatro estados: *ocioso, carregando contexto, executando e salvando contexto*. Quando o controlador está carregando o contexto, ele envia sinais para carregar os operandos necessários para execução da configuração do banco de registrador do processador para o banco de registrador da ATHENA. No estado *executando*, o mCC é utilizado para acessar e enviar a palavra de configuração para a ATHENA. Note que o mCC funciona como um contador de programa para a configuração, endereçando a próxima palavra de configuração para ser enviada. Ao final da execução, no estado *salvando contexto*, os registradores alterados pela configuração são salvos de volta para o banco de registradores do processador. Adicionalmente, o controlador de configuração retorna o controle da execução para o processador, informando o PC de onde deve reiniciar sua execução.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

4 Resultados

Para avaliar o desempenho da arquitetura ATHENA, ela foi implementada no simulador de arquitetura gem5 (BINKERT,2011). Esse simulador também é utilizado para avaliar outras CGRA na literatura como o LASER (BALASUBRAMANIAN,2018) e TransRec (BRANDALERO,2019). Dentre as ISA disponibilizadas no simulador gem5, foi escolhida a ISA Risc-V devido sua ampla aceitação tanto na indústria quanto na academia. O processador de propósito geral utilizado possui uma organização superscalar com despacho de até 8 instruções por ciclo (*8-wide*). Um subconjunto de 9 aplicações do *benchmark* do *mibench suíte* (GUTHAUS,2001) foi usado para avaliar o sistema. Todo o benchmark foi compilado usando o gcc 8.3.0 com a *flag* de otimização -O3 ativada. Através de experimento variando o a quantidade de blocos que podiam conter em uma única configuração, foi escolhida agrupar até 4 blocos básicos, pois essa configuração teve os melhores resultados de desempenho na média.

Para os resultados de área e energia foram utilizadas o CACTI (THOZIYOOR,2008), McPAT (LI,2009) e a ferramenta de síntese da Cadence. Para a estimação da energia e área do processor foi utilizado o McPAT. O CACTI foi utilizado para estimar a área e a energia da cache de configuração. A CGRA ATHENA foi sintetizada usando a biblioteca *GSCLIB045* (*Cadence 45nm Generic Std cell*). O CACTI e o McPAT também utilizaram a tecnologia de 45nm.

4.1 Desempenho

A ATHENA foi capaz de acelerar em até 73% quando comparado com a execução somente no processador, como pode ser visto na Figura 2. A aplicação *bitcount* apresentou a melhor aceleração na ATHENA, executando 1,73x mais rápido, mesmo sendo uma aplicação *control-flow*, onde 23% das intruções executadas são saltos condicionais (*branches*). Isso ocorre porque a execução especulativa permite gerar configuração maiores, mapeando mais de um único bloco básico na CGRA. Além disso, essa aplicação tem muita dependência de dados em seus *kernels*, o que torna a execução no processador consideravelmente mais lenta no processador do que na CGRA que usa lógica combinacional para executar até duas instruções dependentes em um único ciclo. A ATHENA também uma aceleração similar ao *bitcount* nas aplicações *sha* e *susan s* , executando 1,71x e 1,69x mais rápido que a execução somente no processador, respectivamente. Ambas aplicações possuem blocos básicos com muitas instruções (18 instruções por bloco básico em média), o que possibilita melhores mapeamentos para a CGRA devido ao escopo maior para exploração de ILP.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

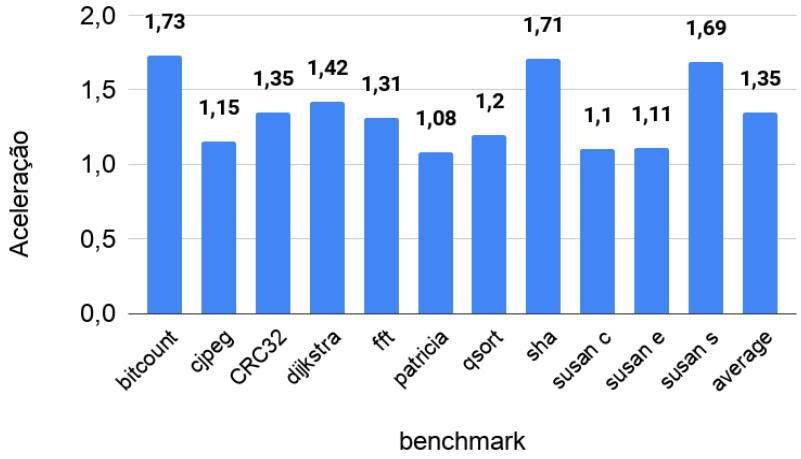

FIGURA 2. ACELERAÇÃO PROVIDA PELA ATHENA NO PROCESSADOR SUPERESCALAR PARA AS DIFERENTES APLICAÇÕES DO BENCHMARK.

Por outro lado, as aplicações *patrícia*, *susan corner* e *susan edges* tiver pouca melhoria no desempenho. *Patricia* possui muitos *kernels* distintos e essa é a principal razão por essa aplicação apresentar a aceleração mais baixa, somente 8%. Ter muitos *kernels* distintos implica em custos adicionais na comunicação devido ao carregamento de configuração e troca de contexto entre a CGRA e o processador, o que reduz os ganhos de desempenho provido pela ATHENA. Outro fator que impacta negativamente a aceleração provida pela ATHENA é a quantidade de operações de acesso à memória. A unidade de *load/store* da ATHENA são simples para manter a CGRA pequena e, por isso, não suporta desambiguação de memória. Já o processador superescalar geralmente usa *store-set* (CHRYSOS,1998) para desambiguação de memória e permite a execução fora de ordem de operações de acesso à memória. Assim, trechos com muitas operações de acesso à memória é executado de forma mais eficiente, do ponto de vista de desempenho, no processador superescalar e executar configurações com muitas operações de memória na CGRA pode levar a uma degradação do desempenho.

Considerando todo o benchmark, ATHENA acelerou em média em 35% a execução das aplicações. Os maiores ganhos da ATHENA foram nas aplicações com muitas operações lógicas e aritméticas (*bitcount, sha* e *susan s*). No entanto, ATHENA melhorou pouco o desempenho das aplicações que possuíam muitas operações de acesso à memória em seu *kernel*.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

4.2 Energia e Área

A Figura 3 mostram duas barras lado a lado. A primeira barra se refere ao consumo energético executando toda a aplicação no processador e a segunda barra ao consumo energético com a ATHENA integrada no sistema. Como pode ser observado, a ATHENA consegue reduzir o consumo energético em todas as aplicações. Em média, a ATHENA economizou 37% do consumo energético quando comparado com a execução somente no procesador. O fator que mais contribui para a economia de energia é que as instruções que são executadas na ATHENA utilizam uma estrutura de hardware muito mais simples que os estágios do pipeline do superescalar. Além disso, diferentemente dos processadores superescalares tradicionais, uma vez que uma configuração é gerada, ela armazena toda as informações sobre escalonamneto e dependência de dados na CGRA. Dessa forma, não é necessário realizar o escalonamento e verificação de dependência de dados como um processador superescalar geralmente faz. E, por fim, a lógica combinacional utilizada nos EPs da ATHENA reduz o número de leituras e escritas feitas no banco de registradores e, assim, também economiza energia.

A ATHENA economizou economizou mais energia nas aplicações onde a aceleração foi maior. Isso ocorreu porque nessas aplicações uma parte maior da aplicação é executada na ATHENA. Em outras palavras, a cobertura da ATHENA na aplicação foi maior.

A área da ATHENA e da cache de configuração é de 0,64 mm^2 e 0,31 mm^2, respectivamente. A área do processador é 38,77 mm^2. Dessa maneira, o custo adicional de área inserido pela ATHENA foi de apenas 2,4%.

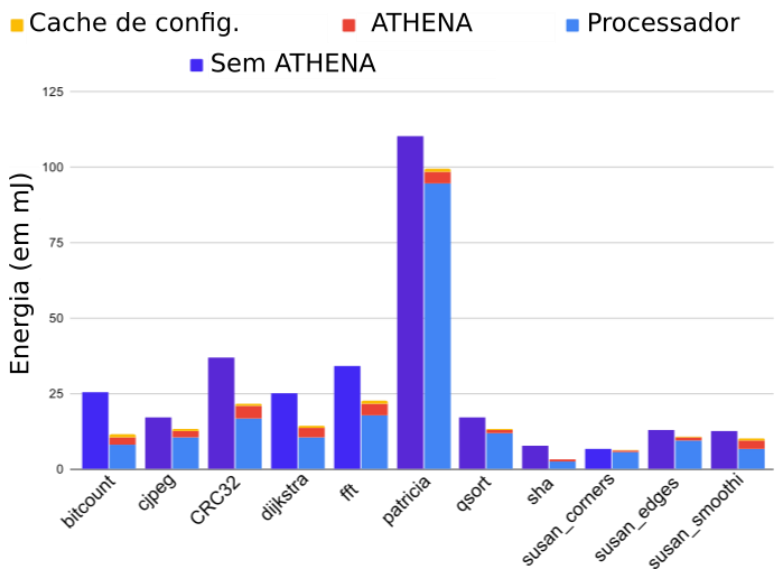

FIGURA 3. COMPARAÇÃO DO CONSUMO ENERGÉTICO DA VERSÃO COM E SEM A ATHENA.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

6 Conclusão e Trabalhos Futuros

Este trabalhou apresentou a CGRA ATHENA que foi integrada de forma transparente a um processador superscalar. O principal objetivo da ATHENA é acelerar as aplicações causando baixo custo adicional de área. A ATHENA foi capaz de acelerar até 1,73x para o benchmark avaliado, enquanto economizou 37% na energia, em média, ao custo de somente 2,4% de custo adicional de área. A ATHENA apresenta, portanto, uma solução promissora para dispositivos *low-power*, embarcados e móveis, onde as restrições de área são fortes. Mesmo com uma CGRA com poucas unidades funcionais, a ATHENA foi capaz de gerar melhorias no desempenho e na energia do processador.

Como trabalho futuro, pretende-se implementar desambiguação de memória nas unidades de acesso a memória da ATHENA, para possibilitar melhorar o desempenho da ATHENA em aplicações *memory-intensive*, que foi onde a ATHENA apresentou mais dificuldade para acelerar.

Referências bibliográficas

AKBARI, O., M. Kamal, A. Afzali-Kusha, M. Pedram e M. Shafique. **Px-cgra: Polymorphic approximate coarse-grained reconfigurable architecture**. Em 2018 Design, Automation Test in Europe Conference Exhibition (DATE), páginas 413–418, 2018.

AKBARI, O., M. Kamal, A. Afzali-Kusha, M. Pedram e M. Shafique. **X-cgra: An energy-efficient approximate coarse-grained reconfigurable architecture**. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, páginas 1–1,2019.

BALASUBRAMANIAN, M., S. Dave, A. Shrivastava e R. Jeyapaul. **Laser: A hardware/software approach to accelerate complicated loops on cgras**. Em 2018 Design, Automation Test in Europe Conference Exhibition (DATE), páginas 1069–1074, 2018.

BECK, A. C. S., M. B. Rutzig, G. Gaydadjiev e L. Carro. **Transparent reconfigurable acceleration for heterogeneous embedded applications**. Em 2008 Design, Automation and Test in Europe, páginas 1208–1213, 2008.

BINKERT, Nathan, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti et al. **The gem5 simulator**. ACM SIGARCH computer architecture news, 39(2):1–7, 2011.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

BRANDALERO, M., M. Shafique, L. Carro e A. C. S. Beck. **Transrec: Improving adaptability in single-isa heterogeneous systems with transparent and reconfigurable acceleration**. Em 2019 Design, Automation Test in Europe Conference Exhibition (DATE), páginas 582–585, 2019.

CHEN, Dev C e Jan M Rabaey. **A reconfigurable multiprocessor ic for rapid prototyping of algorithmic-specific high-speed dsp data paths**. IEEE Journal of Solid-State Circuits, 27(12):1895–1904, 1992.

CHEN, Yu Hsin, Tushar Krishna, Joel S Emer e Vivienne Sze. **Eyeriss: An energy efficient reconfigurable accelerator for deep convolutional neural networks**. IEEE journal of solid-state circuits, 52(1):127–138, 2016.

CHRYSOS, George Z e Joel S Emer. **Memory dependence prediction using store sets**. ACM SIGARCH Computer Architecture News, 26(3):142–153, 1998.

CLARK, N., M. Kudlur, Hyunchul Park, S. Mahlke e K. Flautner. **Application-specific processing on a general-purpose core via transparent instruction set customization**. Em 37th International Symposium on Microarchitecture (MICRO-37'04), páginas 30–40, 2004.

GOLDSTEIN, S. C., H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor e R. Laufer. **Piperench: a coprocessor for streaming multimedia acceleration**. Em Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367), páginas 28–39, 1999.

GOVINDARAJU, V., C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam e C. Kim. **Dyser: Unifying functionality and parallelism specialization for energy-efficient computing**. IEEE Micro, 32(5):38–51, 2012.

GUTHAUS, M. R., J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge e R. B. Brown. **Mibench: A free, commercially representative embedded benchmark suite**. Em Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), páginas 3–14, 2001.

HARTENSTEIN, R. W. e R. Kress. **A datapath synthesis system for the reconfigurable datapath architecture**. Em Proceedings of ASP-DAC'95/CHDL'95/VLSI'95 with EDA Technofair, páginas 479–484, 1995.

HENNESSY, John L. e David A. Patterson. **Computer Architecture, Fifth Edition: A Quantitative Approach**. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA,5th edição, 2011, ISBN 012383872X

LI, Sheng, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen e Norman P Jouppi. **Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures**. Em Proceedings of the

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

42nd Annual IEEE/ACM International Symposium on Microarchitecture, páginas 469–480, 2009.

LIANG, M., M. Chen, Z. Wang e J. Sun. **A cgra based neural network inference engine for deep reinforcement learning**. Em 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), páginas 540–543, 2018.

LIU, F., H. Ahn, S. R. Beard, T. Oh e D. I. August. **Dynaspam: Dynamic spatial architecture mapping using out of order instruction schedules**. Em 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), páginas 541–553, 2015.

LIU, Leibo, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin e Shaojun Wei. **A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications**. ACM Comput. Surv., 52(6), outubro 2019.ISSN 0360-0300. https://doi.org/10.1145/3357375.

LYSECKY, Roman, Greg Stitt e Frank Vahid. **Warp processors**. ACM Trans. Des. Autom. Electron. Syst., 11(3):659–681, junho 2004, ISSN 1084-4309. https://doi.org/10.1145/1142980.1142986.

MÁRIO, Valter, João D. Lopes, Mário Véstias e José T. de Sousa. **Implementing cnns using a linear array of full mesh cgras**. Em Rincón, Fernando, Jesús Barba, Hayden K. H. So, Pedro Diniz e Julián Caba (editores): Applied Reconfigurable Computing. Architectures, Tools, and Applications, páginas 288–297, Cham, 2020. Springer International Publishing, ISBN 978-3-030-44534-8.

MIRSKY, Ethan, Andre DeHon et al. **Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources**. Em FCCM, volume 96, páginas 17–19, 1996.

NAFFZIGER, S., J. Warnock e H. Knapp**. Se2 when processors hit the power wall (or"when the cpu hits the fan")**. EmISSCC. 2005 IEEE International Digest of TechnicalPapers. Solid-State Circuits Conference, 2005., páginas 16–17, 2005.

NAIR, Ravi e Martin E. Hopkins. **Exploiting instruction level parallelism in processors by caching scheduled groups**. SIGARCH Comput. Archit. News, 25(2):13–25, maio 1997.

PATEL, S. J. e S. S. Lumetta: **replay: A hardware framework for dynamic optimization**. IEEE Transactions on Computers, 50(6):590–608, 2001

PODOBAS, A., K. Sano e S. Matsuoka. **A survey on coarse-grained reconfigurable architectures from a performance perspective.** IEEE Access, páginas 1–1, 2020.

TESSIER, R., K. Pocek e A. DeHon. **Reconfigurable computing architectures**. Proceedings of the IEEE, 103(3):332–354, 2015.

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4

THOZIYOOR, Shyamkumar, Naveen Muralimanohar, Jung Ho Ahn e Norman P Jouppi. **Cacti 5.1**. Relatório Técnico, Technical Report HPL-2008-20, HP Labs, 2008.

TU, Fengbin, Shouyi Yin, Peng Ouyang, Shibin Tang, Leibo Liu e Shaojun Wei. **Deep convolutional neural network architecture with reconfigurable computation patterns**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(8):2220–2233, 2017.

WATKINS, M. A., T. Nowatzki e A. Carno. **Software transparent dynamic binary translation for coarse-grain reconfigurable architectures**. Em 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), páginas 138–150, 2016.

WIJTVLIET, M., L. Waeijen e H. Corporaal. **Coarse grained reconfigurable architectures in the past 25 years: Overview and classification**. Em 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), páginas 235–244, 2016.

ZERVAKIS, G., H. Amrouch e J. Henkel. **Design automation of approximate circuits with runtime reconfigurable accuracy**. IEEE Access, 8:53522–53538, 2020.

Agradecimentos: Este trabalho foi financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES).

Revista da Academia de Ciências do Piauí, Ano 1, Nº 01, Jul. a Dez. de 2020. p.50-64 . ISSN:2675-9784 www.acipiaui.org.br
DOI:10.29327/261865.1.1-4