Evaluation and Proposal of a Lightweight Reconfigurable Accelerator for Heterogeneous Multicore

Francisco Carlos Silva Junior, Ivan Saraiva Silva, Ricardo Pezzuol Jacobi

***Abstract*—Nowadays, with the spread of mobile devices, energy efficiency and performance have become important requirements in system design. These devices need enough performance to run increasingly complex applications at the lowest possible energy cost to extend battery life. Heterogeneous systems have proven to be a promising solution to meet such demands. Generally, such systems consist of cores that offer either energy efficiency or high performance. This work proposes a reconfigurable accelerator, called ALLEGRI, and evaluates its impact on performance and energy consumption considering three different versions of a multicore processor, varying the superscalar processor issue width (single, dual and four-issue). The parmibench suite was used to evaluate the system. The results show that the version that has the best energy efficiency and the best performance was the single-issue multicore + ALLEGRI and four-issue + ALLEGRI, respectively.**

***Index Terms*—Multicore processors, Reconfigurable architecture, Processor design exploitation, Binary translation, MPSoC.**

I. Introdução

Nos últimos anos o aumento da complexidade das aplicações tem exigido cada vez mais desempenho dos processadores. Ao mesmo tempo, com a popularização de dispositivos móveis, tais como *smartphones* e *smartwatches*, aumenta também a demanda por eficiência energética. Com isso, o projeto de sistemas embarcados atualmente requer um compromisso entre energia e desempenho.

Arquiteturas reconfiguráveis (AR) surgiram como uma solução arquitetura para diminuir o *gap* de desempenho e energia entre o processador de propósito geral (PPG) e os ASICs. Os PPG possuem uma alta programabilidade, mas oferecem baixo desempenho e alto custo energético. Por outro lado, os ASICs possuem alto desempenho a baixo consumo energético, mas não oferecem nenhuma programabilidade. As ARs, por meio da exploração de suas unidades funcionais reconfigurável através de configurações, conseguem oferecer programabilidade e um melhor desempenho e eficiência energética que os GPPs.

As ARs têm sido amplamente utilizada como aceleradores em processadores *single-core* [1], [2], [3], [4]. No entanto, com a barreira da potência (*power-wall*), a indústria tem investido principalmente em arquiteturas *multicore*. Com isto ARs voltadas para multicore também têm sido propostas [5], [6], [7].

Francisco Carlos Silva Junior, UnB, juninho.ufpi@hotmail.com
Ivan Saraiva Silva, UFPI, ivan@ufpi.edu.br
Ricardo Pezzuol Jacobi, UnB, jacobi@unb.br.

Em [5] e [6] uma AR é integrada de forma dedicada para cada núcleo. A AR é uma extensão de um trabalho previamente proposto, voltada para processador *single-core* [8]. Enquanto que em [5] a AR é a mesma para todos os núcleos, em [6] a AR empregada para cada núcleo possui diferente quantidade de unidades funcionais. Ambas utilizam um processador *single-issue in-order* com 5 estágios como PPG. No entanto, a maioria dos processadores atuais, mesmo os embarcados, são superescalares fora de ordem [9]. A menor versão da AR proposta em [6] possui 27 ULAs, 3 multiplicadores e 6 unidades de *load/store* e em [5] a AR é composta por 144 ULAs, 48 load/stores e 16 multiplicadores. Apesar do custo de área não ser discutido nesses dois trabalhos, é esperado que ele seja alto.

Em [7], visando mitigar a sobrecarga de área causada pela AR no multicore, uma AR compartilhada é proposta em um multicore inspirado no ARM big.LITTLE [10]. No entanto, como a AR é compartilhada, pode haver perda de desempenho por contenção de recursos.

Esse trabalho propõe e avalia uma AR leve, denominada de ALLEGRI (***A L**ightweight Pipe**L**ined r**E**confi**G**urable a**R**ch**I**tecture*), que foi projetada para ser integrada em um *multicore*. Diferentemente de [5] e [6], que adaptou uma AR voltada para *single-core*, A ALLEGRI foi proposta pensando em reduzir a sobrecarga de área causada sem impactar muito no desempenho. Devido a ALLEGRI ser consideralvemente menor que as AR encontradas na litetura, isso possibilitou a sua integração de forma dedicada em cada núcleo. Ao invés de reduzir o custo de área por usar uma AR compartilhada como em [7], foi proposta uma arquitetura com menos recursos e que não opera de forma totalmente combinacional como em [5], [6], [7]. A ALLEGRI é integrada em um *multicore* composto de 4 núcleos superescalares RISC-V. Foram desenvolvidas 3 versões do processador *multicore* e avaliada a relação desempenho/energia de cada uma delas. Com os dados obtidos, é possível propor um MPSoC (*Multicore Processor System-On-Chip*) heterogêneo "ideal"composto por núcleos de alta eficiência energética e de alto desempenho. Para avaliar o sistema proposto foi utilizado um subconjunto do *benchmark* Parmibench [11]. As principais contribuições deste trabalho são:

- Proposta de uma arquitetura reconfigurável com pouco custo adicional de área no sistema *multicore*;
- Avaliação de como o desempenho e a energia variam conforme se modifica a largura de despacho dos núcleos superescalar em um sistema multicore homogêneo aco-

plado a ALLEGRI;
- Mostrar que um *multicore* com núcleos superescalares *single-issue* + ALLEGRI fornece 1,04x melhor desempenho e gasta 1,22x menos energia que um multicore com núcleos superescalares *four-issue*;
- Os dados obtidos permitem definir a arquitetura que provê maior desempenho e eficiência energética para projeto de MPSoCs heterogêneos.

Este artigo está organizado em 5 seções. A Seção II apresenta os trabalhos relacionados. A Seção III descreve o sistema proposto. A seção IV apresenta a metodologia utilizada nos experimentos e os resultados de energia e desempenho. Por fim, a seção V apresenta as conclusões e discute os trabalhos futuros.

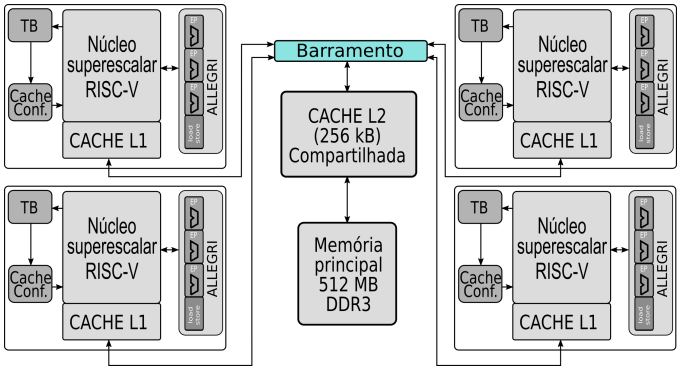

Fig. 1. Visão geral do sistema proposto.

II. Estado da Arte

Arquiteturas reconfiguráveis têm sido amplamente investigadas na literatura [12][13]. Inicialmente, essas arquiteturas eram acopladas aos processadores de núcleo único, como a arquitetura Morphosys [1], Chimaera [3], PipeRench [4] e DynaSpAM [14]. No entanto, com a barreira da potência, os processadores *multicore* se tornaram dominantes no mercado de processadores de propósito geral (PPG) [15]. Dentro desse contexto, começaram a surgir arquiteturas reconfiguráveis para processadores *multicore* [5][6][16][7].

CReAMS [5] e HARTMP [6] adaptam uma AR *single-threaded* para processadores *multicore*. Os processadores são *single-issue* e fazem despacho em ordem. A AR é reconfigurada de forma transparente através de um hardware de tradução binária. CReAMS propõe um *multicore* homogêneo, enquanto que o HARTMP propõe um *multicore* onde os recursos disponíveis na AR serão diferentes e, portanto, o *multicore* é heterogêneo. Com essa organização heterogênea, os autores esperam melhorar a utilização dos recursos reconfiguráveis, pois as *threads* com maior carga de trabalho serão alocadas aos núcleos com maior capacidade de processamento.

Diferentemente do HARTMP e do CReAMS, o trabalho aqui proposto integra um AR a um processador superescalar fora de ordem, mais complexo e mais compatível com os processadores que estão no mercado nos dias atuais, como intel i7 e ARM cortex A15. Além disso, realiza-se uma exploração do espaço de projeto avaliando-se energia e desempenho para cada versão do PPG do *multicore*. Ao invés de adaptar uma AR *single-thread* para um processador *multicore*, como no HARTMP e no CReAMS, este trabalho propõe uma AR que foi projetada para ter um baixo custo em área no *multicore*. No entanto, apesar do baixo custo em área, a AR deve ter recursos suficiente para acelerar as aplicações. Esse baixo custo em área viabiliza a integração da AR de forma dedicada para cada núcleo.

TransRec [7] é a primeira proposta de integrar uma AR em um *multicore* heterogêneo de ISA única. O *multicore* é inspirado no big.LITTLE da ARM [10]. A AR é compartilhada temporalmente entre os dois núcleos. Apesar de propor um *multicore* heterogêneo, o sistema é simulado em um ambiente *single-core* onde o processador *single-core* vai ter uma das configurações: LITTLE, *big* ou *bigger*.

No trabalho aqui proposto, as simulações são feitas em um ambiente *multicore*. As aplicações foram paralelizadas utilizando o padrão POSIX *Threads* (*pthreads*). Desse modo, todos os impactos devido à utilização de um multicore são levados em consideração, tais como: custo de comunicação, coerência de *cache* e sincronização.

Trabalhos mais recentes [17], [18], [19] têm focado em eficiência energética, sacrificando precisão na computação, em aplicações que suportam essas imprecisões, em troca de economia energética e de área. A ALLEGRI também foca em eficiência energética, mas ao invés de propor unidades de computação aproximada, propõe um AR que usa consideravelmente menos unidades funcionais que os trabalhos encontrados na literatura. Isso, além de diminuir o custo energético, também reduz o custo em área, que é muito importante principalmente nos *multicores*, pois isso viabiliza a integração de forma dedicada da AR em cada núcleo do processador.

III. Sistema Proposto

A arquitetura proposta para avaliação é um processador *multicore* homogêneo composto por quatro núcleos. Os núcleos do processador foram modificados para prover reconfiguração transparente. Todos os núcleos são compostos por um processador superescalar RISC-V com execução fora de ordem. A cada núcleo foi acoplado um hardware para realizar a tradução binária (TB), uma *cache* de configuração e a arquitetura reconfigurável ALLEGRI. Uma visão geral do sistema proposto pode ser visto na Fig. 1. Os componentes serão explicados em detalhes nas próximas seções.

Cada núcleo do processador possui *caches* L1 privadas. Ambas as *caches*, L1-Icache e L1-Dcache, são associativas por conjunto de 4 vias e possuem 16 KB cada. É utilizado um mecanismo baseado em diretório para prover coerência de *cache*. Foi utilizado o protocolo MOESI (*Modified*, *Owned*, *Exclusive*, *Shared*, *Invalid*), detalhes desse protocolo podem ser vistos em [20]. O modelo de consistência sequencial proposto por Lamport [21] foi implementado para garantir uma visão consistente da memória que é compartilhada entre os núcleos.

A. O Núcleo de Processamento

O núcleo de processamento é composto por um processador superescalar Risc-V, um tradutor binário (TB), *cache* de configuração, um controlador de configuração e a arquitetura ALLEGRI. A Fig. 2 mostra em mais detalhes a organização do núcleo de processamento. Os blocos em azul são os estágios do *pipeline* de um processador superescalar convencional e os blocos em amarelo são as modificações propostas neste trabalho.

Todos os blocos adicionais propostos dão suporte a reconfiguração da arquitetura ALLEGRI. O TB gera as configurações, lendo as instruções que são processadas no estágio de *commit* do processador. Ao final do processo de geração da configuração, que será detalhado mais a frente, o TB salva em *cache* de configuração o mapeamento dos blocos básicos na arquitetura ALLEGRI. O controlador de configuração garante a coerência entre o banco de registradores local da ALLEGRI e do processador e gerencia a execução na ALLEGRI.

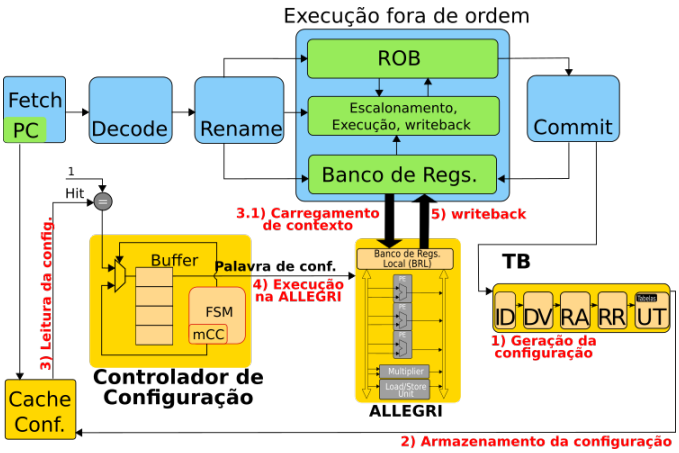

Fig. 2. Visão detalhada do núcleo de processamento.

B. A Arquitetura ALLEGRI

A ALLEGRI possui três tipos de unidades funcionais: i) Elementos de Processamentos (EPs); ii) Multiplicador e iii) Unidade de *load/store*. Esses elementos estão organizados em uma coluna, utilizados através de configurações que são enviadas a cada ciclo, funcionando de forma semelhante a um *pipeline*. Um banco de registradores local (BRL) também foi adicionado à ALLEGRI para armazanemante temporário dos dados. Portanto, para realizar as computações, a ALLEGRI lê seus operandos do BRL e escreve seu resultado também no BRL. Na versão atual da ALLEGRI não há suporte para operações de ponto flutuante.

O Elemento de Processamento é capaz de realizar operações lógicas e aritméticas inteiras e é composto por uma ULA (Unidade de Lógica e Aritmética). As operações realizadas no EP possuem latência de 1 ciclo do processador. O multiplicador realiza multiplicações inteiras e possui latência de 3 ciclos. A unidade *load/store* realiza as operações de acesso à memória e possui 2 ciclos de latência em caso de *cache hit*. Caso um *cache miss* aconteça, a arquitetura ALLEGRI ficará aguardando até que o *cache miss* seja resolvido para que possa continuar sua execução.

C. A Configuração e o Controlador de Configuração

Uma configuração mapeia a computação a ser realizada por um dado trecho de código para a ALLEGRI. Neste trabalho, a configuração é dividida em palavras de configuração e cada palavra de configuração define as operações realizadas na ALLEGRI em um único ciclo de relógio. Essa definição inclui o código das operações a serem executadas nos EPs, bem como os endereços dos registradores que contêm os operandos fonte, o registrador destino e uma constante para operações que operem com valores imediatos, como por exemplo *addi*, *load* e *store*. Como mencionado anteriormente, as unidades funcionais da ALLEGRI são exploradas ciclo a ciclo, reusando os mesmos recursos, de forma similar a um *pipeline*. Deste modo, a quantidade de recursos que uma configuração pode utilizar é delimitada pela quantidade de palavras de configuração que uma configuração pode conter. Neste trabalho foi adotado o máximo de 72 palavras de configuração por configuração, pois com essa quantidade de palavras o TB consegue mapear todos os *kernels* das aplicações utilizadas para avaliar a arquitetura. Dessa forma, uma configuração pode mapear até 216 operações de lógica e aritmética, 24 multiplicações e 36 operações de acesso à memória na ALLEGRI.

O controlador de configuração gerencia a execução na ALLEGRI e é responsável por enviar as palavras de configuração no tempo apropriado. Para fazer isso ele possui um *buffer* FIFO com 4 entradas, uma máquina de estados finita e um mCC (*micro Configuration Counter*). O *buffer* é usado para armazenar temporariamente as configurações encontradas na *cache* de configuração e que ainda não atingiram o estágio de *commit*. A máquina de estado possui quatro estados: ocioso, carregando contexto, executando e salvando contexto. No estado executando, o mCC é usado para acessar e enviar a palavra de configuração apropriada para a ALLEGRI. Note que o mCC funciona como um contador de programa para a configuração, endereçando a próxima palavra de configuração a ser enviada.

D. O Tradutor Binário e a Cache de Configuração

O TB dinamicamente detecta sequências de instruções para serem executadas na ALLEGRI. Na implementação proposta em [22], o TB é implementado em um pipeline de 4 estágios. No trabalho proposto aqui o TB foi extendido para 5 estágios para adicionar o estágio de *register renaming* entre os estágios de alocação de recurso e atualização de tabelas. Este estágio adicional foi projetado para tratar falsas dependências, como WAW (*Write After Write*) e WAR (*Write After Read*), que são resolvidos pela técnica de *renaming*. A implementação proposta em [22] também trata falsas dependências, mas somente em arquiteturas baseadas em barramentos.

O algoritmo de tradução binária é um algoritmo guloso que aloca as operações na ALLEGRI. O processo inicia quando a primeira instrução é processada no estágio de *commit* do processador. A instrução será alocada na ALLEGRI considerando a dependência de dados, latência da operação e os recursos

disponíveis na ALLEGRI. A geração de um configuração é finalizada em três casos: i) uma instrução não suportada pela ALLEGRI foi recebida pelo TB; ii) A ALLEGRI não possui recurso para alocar a instrução ou iii) o número de blocos básicos mapeados atingiu seu limite.

Quando o processo de geração da configuração termina, a configuração é salva na *cache* de configuração, indexada pelo PC da primeira instrução mapeada na ALLEGRI pertencente àquela configuração. A *cache* de configuração armazena até 128 configurações e usa a LFRU (*Least Frequent Recently Used*) como política de substituição.

Para permitir melhor exploração de ILP (*Instruction Level Parallelism*) entre os blocos básicos, a ALLEGRI suporta execução especulativa. A especulação é feita mapeando mais de um bloco básico na mesma configuração. A especulação sempre assume o mesmo caminho tomado pelo salto quando a configuração está sendo gerada. Para dar suporte a execução especulativa, um bit para cada unidade funcional foi adicionado na palavra de configuração para informar se operação é especulativa ou não. Quando uma configuração erra sua especulação, todas as computações que foram executadas de forma especulativa são apagadas (*flushed*) e, portanto, não são computadas. Se uma configuração erra sua predição duas vezes consecutivas ela é apagada da *cache* de configuração. Desse modo, uma nova configuração é gerada na próxima execução e, provavelmente, com uma melhor predição.

E. Exemplo de Funcionamento da ALLEGRI

Esta seção explica como funciona a execução em um núcleo com a ALLEGRI. A explicação é dividida em três partes: a execução no processador, geração da configuração e a execução na ALLEGRI.

1) Execução no Processador: A execução no processador ocorre como em um processador superescalar convencional, onde a instrução passa por todos os estágios do processador: *fetch, decode, rename, dispatch, issue, execute* e *commit*.

2) Geração da Configuração: Em paralelo com a execução do processador, O TB mapeia as instruções para serem executadas na ALLEGRI (Fig. 2 passo 1). Para realizar a comunicação entre o TB e o estágio de *commit* do processador superescalar, foi adicionado um buffer adicional, *Commit buffer*, ao processador superescalar. No estágio de *commit*, antes de remover a instrução do ROB, a instrução é inserida no *Commit buffer*, que é então lido pelo TB para geração da configuração, conforme explicado na seção III-D. Uma vez finalizada a geração da configuração, o TB armazena-a na *cache* de configuração (Fig. 2 passo 2).

3) Execução na ALLEGRI: No estágio de *fetch* do processador, para cada novo bloco básico, ele irá verificar se existe uma configuração para o PC desse bloco básico. Caso não exista, a execução será realizada no processador. Caso contrário, a configuração é carregada no *buffer* de configuração no controlador de configuração (passo 3 da Fig. 2). Adicionalmente, é adicionado uma *flag* nessa instrução para informar que ela marca o início da execução de uma configuração. Nesse ponto também se inicia o carregamento dos operandos necessários para execução na ALLEGRI (passo 3.1 da Fig. 2). O carregamento dos operandos é realizado pelo controlador de configuração que lê os valores dos registradores do processador e escreve-os no BRL. Quando a instrução que marca o início da configuração chega no estágio de *commit*, este interrompe o processador (*stall*) e sinaliza o início de execução ao controlador de configuração. Para dar *stall* no pipeline do processador o ROB é todo invalidado, o que remove todas as instruções que estão em andamento no pipeline e serão executadas na ALLEGRI. Além disso, o *commit* notifica o *fetch* para aguardar sinalização da ALLEGRI para continuar a fazer a busca de novas intruções. O controlador gerencia a execução na ALLEGRI como descrito na seção III-C. Ao final da execução, os registradores escritos durante a execução da configuração são copiados de volta para o banco de registradores do processador (passo 5 da Fig. 2). Além disso, o controlador de configuração informa para o processador o PC para onde ele deve retornar sua execução.

IV. Resultados

A. Metodologia

Para avaliar o desempenho, o sistema proposto foi implementado no simulador gem5 [23] utilizando o modelo *out-of-order CPU*, que fornece precisão de ciclo nas simulações. A ISA Risc-V foi escolhida por ser uma arquitetura de código aberto e estar ganhando aceitação na indústria e academia.

A estimativa de área e energia do processador foi obtida usando o McPAT [24]. O CACTI [25] foi usado para estimar a área e energia da *cache* de configuração. A ALLEGRI foi sintetizada usando a biblioteca Cadence GSCLIB045 (*Cadence 45nm Generic Std Cel*). Com essa implementação, foi possível obter a de frequência de 1,4 GHz. Sendo assim, o CACTI e o McPAT consideram a tecnologia de 45nm e a frequência de 1,4 GHz. Essa metodologia de análise de desempenho e de consumo energético é a mesma utilizada em outros trabalhos que avaliam uma AR, como TransRec [7], DynaSpAM[14] e DYSER [26]. Para avaliar a eficiência energética, foi utilizado o EDP (*Energy-Delay Product* [27]. Essa métrica foi adicionada ao estudo para permitir uma comparação com o TransRec [7].

O sistema foi avaliado considerando três diferentes processadores de propósito geral utilizado no *multicore*: i) *single-issue* (SI), ii) *dual-issue* (DI) e iii) *four-issue* (FI). Depois, a ALLEGRI foi acoplada no sistema *multicore*, totalizando seis configurações possíveis: i) *multicore* SI, ii) *multicore* SI + ALLEGRI, iii) *multicore* DI, iv) *multicore* DI + ALLEGRI, v) *multicore* FI e vi) *multicore* FI + ALLEGRI. Os parâmetros utilizados para cada um desses processadores podem ser vistos na Tabela I. A hierarquia de memória foi a mesma para todas as versões e seus parâmetros podem ser vistos na Tabela II.

Um conjunto de 6 aplicações do *parmibench suite* [11] foram utilizadas para avaliar o sistema proposto. O *parmibench* foi escolhido pois é um *benchmark* de código aberto para processadores *multicore* embarcados e disponibiliza aplicações em cinco domínios de aplicação: automação; controle industrial; redes; escritório e segurança. As aplicações são paralelizadas usando *pthreads*. Além dessas aplicações, foi implementada a multiplicação de matrizes, por ser um exemplo

clássico de aplicação altamente paralelizável com *threads* regulares. Todas as aplicações foram compiladas para Risc-V utilizando a *flag* de otimização -O2 usando o compilador gcc.

TABELA I
PARÂMETROS MICROARQUITETURAIS DAS DIFERENTES VERSÕES DO PROCESSADOR.

Versão	Parâmetros
SI	single-issue; fora de ordem; Fetch buffer: 8 insts.; Entradas no ROB: 40; Load buffer: 16; Store buffer: 16; Preditor de salto: Bi-Mode
DI	dual-issue; fora de ordem; Fetch buffer: 16 insts.; Entradas no ROB: 40; Load buffer: 16; Store buffer: 16; Preditor de salto: Bi-Mode
FI	four-issue; fora de ordem; Fetch buffer: 16.; Entradas no ROB: 60; Load buffer: 16; Store buffer: 16; Preditor de salto: Bi-Mode

TABELA II
PARÂMETROS DA HIERARQUIA DE MEMÓRIA DO PROCESSADOR MULTICORE.

Cache	Tipo	Procolo de Coerência	Modelo de Consistência	Tamanho
L1 de dados	Privada	MOESI	-	16KB
L1 de instrução	Privada	MOESI	-	16KB
L2 de dados e instruções	Comparti.	MOESI	Sequencial	256KB

B. Desempenho do Processador

O IPC (*Instruction Per Cycle*) das diferentes versões simuladas é mostrado na Tabela III. Conforme aumenta-se a largura do superescalar (*issue size*) do processador superescalar, o IPC tende também a aumentar, pois o processador é capaz de explorar mais ILP. No entanto, o aumento no IPC variando a largura do superescalar vai depender de características da aplicação. Os principais fatores que impactam são: quantidade de instruções de acesso à memória (desambiguação de referências à memória é um gargalo no superescalar) e dependência de controle. Na aplicação *bitcount*, por exemplo, por ser uma aplicação *control-flow* e possuir muita dependência de controle, o aumento na largura do superescalar teve pouco impacto no IPC. Por outro lado, a aplicação *susan smoothing*, que possui apenas 8,2% das instruções executadas sendo saltos, aumentou seu IPC em 1,6x e 1,39x comparado a versão SI e DI e a versão DI e FI, respectivamente.

A Fig. 3 mostra o tempo de execução para cada versão simulada do sistema normalizado pelo tempo de execução no processador multicore SI. Como pode se observar na Fig. 3, as aplicações que mais melhoraram seu IPC também foram as que mais tiveram seu tempo de execução reduzido conforme variou a largura do superescalar do processador. No *Susan Smoothing*, utilizando a versão FI sem a arquitetura reconfigurável, o tempo de execução foi reduzido quase pela metade quando comparado com a versão SI. Por outro lado, o *bitcount* teve praticamente o mesmo tempo de execução nas diferentes versões do processador.

TABELA III
IPC DOS DIFERENTES MULTICORES E OPERAÇÕES REALIZADA EM CADA APLICAÇÃO.

	ULA int. (%)	**Load/ Store (%)**	**Branch (%)**	**IPC SI**	**IPC DI**	**IPC FI**
bitcount	78,59	3	18	2,16	2,20	2,20
mult. de m.	65,16	26,11	8,72	1,88	2,44	2,60
patricia	59,21	21,88	18,75	1,68	2,08	2,20
sha	74,73	18,54	6,71	1,64	2,12	2,24
susan c	21,56	73,05	4,85	3,2	3,52	3,68
susan e	52,14	41,69	4,53	2,08	3,36	3,96
susan s	69,58	21,99	8,2	2,4	3,84	4,56

C. Melhoria no Desempenho com a ALLEGRI

Depois dos experimentos utilizando somente os diferentes processadores multicore, a arquitetura ALLEGRI foi acoplada ao multicore e avaliou-se o ganho de desempenho que ela consegue prover. A arquitetura ALLEGRI melhora o desempenho em quase todos os cenários, como pode ser visto na Fig. 3. A ALLEGRI consegue melhorar o desempenho por explorar mais ILP que o processador, no caso dos processadores SI e DI, e conseguir maior vazão de instruções por pular os estágios de *fetch*, *decode*, *rename* e *scheduling* usando o mapeamento gerado em execuções anteriores daquele trecho.

As aplicações que a ALLEGRI mais acelerou foram: *bitcount*, *sha* e *susan smoothing*. Essas três aplicações também são as que possuem mais instruções de operações lógicas 72,59%, 74,73% e 69,58% (Tabela III), respectivamente. Apesar da grande quantidade de *branches* na aplicação *bitcount*, esses saltos possuem um comportamento altamente regular, o que resultou em um taxa de erro de predição na ALLEGRI de apenas 3,8%. Com isso, a ALLEGRI conseguiu acelerar, através da execução especulativa, trechos com muita dependência de controle. As aplicações *sha* e *susan smoothing*, além de muitas operações lógicas e aritméticas, possuem blocos básicos maiores. Com blocos básicos maiores, maior é o espaço para explorar ILP e melhores são os mapeamentos na ALLEGRI, portanto, maior a aceleração provida pela arquitetura reconfigurável. A aplicação *sha* e *susan smoothing* possuem, em média, 13,89 e 11,18 instruções por bloco básico, respectivamente. Por outro lado, uma aplicação *control-flow* como o *bitcount* possui, em média, apenas 4,44 instruções por bloco básico.

A única exceção onde a ALLEGRI não conseguiu prover ganho de desempenho foi na aplicação *susan corners* na versão FI. Essa aplicação possui muitos acessos à memória, 73,05% das instruções executadas. Como as operações de acesso a memória na arquitetura reconfigurável são feitas de forma conservadora, sem predição de dependência, esse tipo de operação é executada de forma mais eficiente no processador superescalar, que implementa predição de dependência de memória utilizando *store sets* [28].

Considerando o sistema com a ALLEGRI acoplada ao processador, percebe-se uma tendência de menor tempo de execução conforme aumenta-se a largura do superescalar. Esse comportamento é esperado, pois, como o processador de propósito geral consegue prover maior desempenho conforme

aumenta-se sua largura, os trechos executados no processador serão executados mais rápido e os trechos executados na ALLEGRI manterão seu desempenho.

No entanto, no *bitcount* esse comportamento não foi observado, onde a versão DI + ALLEGRI teve um tempo de execução maior que a versão SI + ALLEGRI. Isso pode acontecer porque a configuração gerada depende de como o fluxo de instruções é executado no processador. Nesse caso específico, as configurações geradas pelo TB na versão SI + ALLEGRI são muito melhores, pois conseguiram mapear de forma mais eficiente os *kernels* da aplicação.

A versão que teve o melhor desempenho foi a FI + ALLEGRI que apresenta o menor tempo de execução considerando todo o *benchmark*.

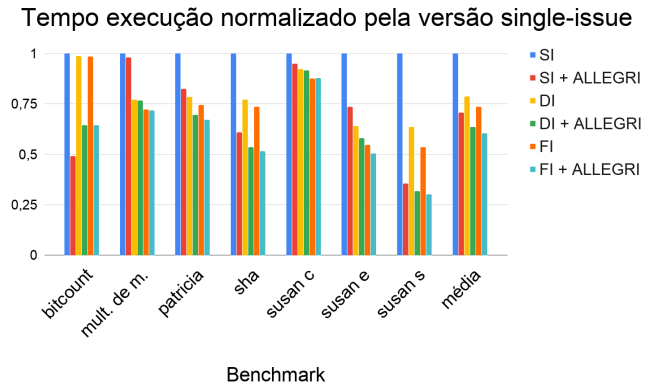

Fig. 3. Tempo de execução normalizado pela versão SI.

D. Consumo Energético

O consumo energético das diferentes configurações do sistema pode ser visto na Fig. 4. Considerando as três versões do *multicore*, sem a adição da ALLEGRI, nas aplicações onde o aumento da largura do superescalar trouxe pouco benefício de desempenho (*bitcount* e *susan corners*), houve aumento no consumo energético conforme aumentou-se a largura do superescalar do processador. Esse aumento é causado por conta do ganho de desempenho não compensar o aumento da potência do processador superescalar ao aumentar sua largura.

A ALLEGRI conseguiu reduzir o consumo energético para todas as aplicações. A ALLEGRI, quando está executando, economiza a energia que o processador gastaria passando pelos estágios de *fetch*, *decode*, *rename* e pelo *scheduling* para executar aquele mesmo trecho de código. Isso é possível porque a ALLEGRI utiliza informações de execuções anteriores e salva, em configuração, o mapeamento daquelas instruções na arquitetura reconfigurável. Além disso, a ALLEGRI consegue reduzir o tempo de execução das aplicações. Em média, a ALLEGRI reduziu em 30%, 21% e 23% o consumo energético nas versões SI, DI e FI, respectivamente.

Para todo o benchmark avaliado, a configuração do sistema que gasta menos energia para executar as aplicações é a versão SI + ALLEGRI. Nessa configuração temos um processador que dissipa a menor potência entre os 3 processadores simulados e uma arquitetura reconfigurável para acelerar os *kernels* das aplicações. Portanto, essa versão apresenta-se com uma boa solução em termos de economia de energia.

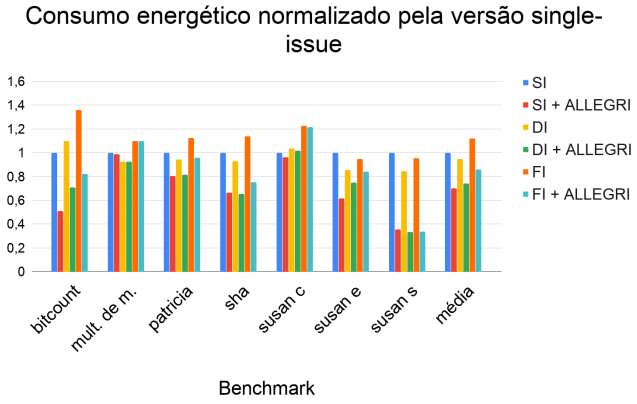

Fig. 4. Consumo energético normalizado pela versão SI.

E. Análise da Eficiência Energética

Para avaliar a eficiência energética foi utilizado o EDP (*Energy-Delay Product*) [27]. A Fig. 5 apresenta a melhoria do EDP das diferentes versões do multicore quando comparado com a versão SI.

Considerando somente as versões do multicore sem a ALLEGRI, a versão DI foi a que teve o melhor EDP. A execução na versão DI é 1,42x mais rápida e consome 1,06x menos energia, em média, quando comparado com a versão SI, provendo um EDP 1,36x melhor. A versão FI possui melhor EDP que a versão SI, no entanto, não consegue prover melhor EDP que a versão DI, pois a versão FI executa 1,06x mais rápido e consome 1,51x mais energia que a versão DI. Com isso, seu EDP é 1,2x pior. A única aplicação onde o melhor EDP é provido pela versão SI é na aplicação *bitcount*. Como já explicado anteriormente, nessa aplicação o aumento na potência dissipada pelo processador causado pelo aumento na largura do superescalar não compensa o ganho de desempenho, por isso, o melhor EDP é da versão SI.

Ao acoplar a ALLEGRI, o EDP é melhorado significativamente. Em média o EDP com a ALLEGRI no sistema é 4x melhor que o EDP da versão SI. A ALLEGRI melhora o EDP em todas aplicações do *benchmark*. Os ganhos mais significativos foram nas aplicações onde a ALLEGRI proveu os maiores ganhos de desempenho (*bitcount* e *susan smoothing*). O multicore FI com a ALLEGRI conseguiu prover um EDP de até 9,75x melhor quando comparado com o multicore SI. Esse ganho é possível porque a ALLEGRI consegue melhorar o desempenho quando tem paralelismo disponível ao mesmo tempo que economiza energia executando de forma mais eficiente que o processador superescalar.

F. Custo de Área da ALLEGRI

A Tabela IV mostra os resultado de área dos diferentes processadores e da ALLEGRI. A ALLEGRI possui uma área de 7,28 mm^2, incluindo a *cache* de configuração (3,64 mm^2).

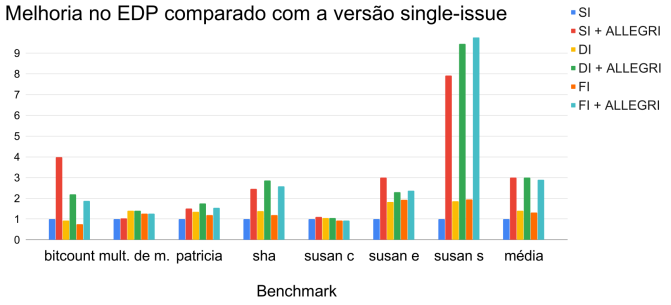

Fig. 5. Melhoria no EDP considerando as diferentes versões.

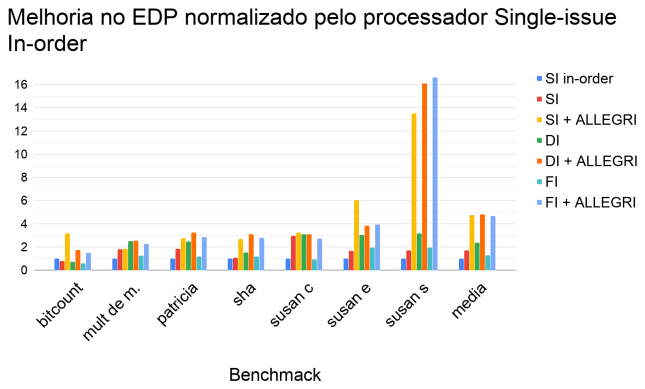

Fig. 6. EDP com relação ao processador *single-issue in-order*.

Os processadores *single-issue*, *dual-issue* e *four-issue* possuem áreas de 41,05 mm^2, 41,82 mm^2 e 55,35 mm^2, respectivamente. O custo adicional de área causada pela inserção da ALLEGRI é de apenas 17,7% no pior caso. No melhor caso, quando integrada ao processador *four-issue*, é de apenas 13,15%.

G. Relação com o Estado da Arte

Uma relação do trabalho aqui proposto com o Transrec [7] foi realizada considerando o EDP. Uma comparação direta com este mesmo trabalho não foi possível, pois não trazeria conclusões seguras devido a diferenças na frequência de operação do sistema, metodológicas e de tecnologia. Esse trabalho foi selecionado por utilizar a mesma ISA. A métrica EDP foi utilizada, pois é a única métrica apresentada em [7] e, por este motivo, não utilizou-se outras métricas como desempenho, energia e área.

Como o EDP apresentado no Transrec é normalizado com base no EDP de um processador *single-issue in-order*, foram feitos novos experimentos considerando um *multicore* que usa um processador *single-issue in-order*. Foi utilizado o modelo de processador *TimingSimpleCPU* do gem5 para modelar o multicore *single-issue in order*. Os resultados podem ser vistos na Fig. 6.

A ALLEGRI conseguiu melhorar, em média, 4,74x o EDP quando comparado com processador *multicore single-issue in-order*. Por outro lado, o Transrec melhorou, em média, 2,72x o EDP, considerando sua melhor versão em termos de EDP, a *bigger + CGRA*.

O trabalho apresentado em [7], além de muito recente, é, muito provavelmente, o mais relacionado com este artigo. Entretanto, os resultados apresentados incluem somente EDP, o que dificulta uma comparação precisa, pois o EDP agrupa informação de energia e desempenho em uma só métrica. Ademais, alguns parâmeros escolhidos (frequência de operação e tecnologia) diferem substancialmente daqueles usados em ALLEGRI. De toda forma, com um array menor, com uma frequência menor a ALLEGRI conseguiu melhores resultados de EDP quando comparado com os apresentados em [7].

TABELA IV
INFORMAÇÃO DE ÁREA DO PROCESSADOR DA ALLEGRI

Versão	Área do processador (mm^2)	ALLEGRI + Cache de conf. (mm^2)	Custo adicional (%)
SI + ALLEGRI	41,05	7,28	17,7%
DI + ALLEGRI	41,82	7,28	17,4%
FI + ALLEGRI	55,35	7,28	13,15%

V. Conclusão e Trabalhos Futuros

Este trabalho propôs e avaliou diferentes processadores *multicores* acoplados à arquitetura reconfigurável ALLEGRI. Foram avaliadas três versões do processador *multicore*, variando a largura de despacho do processador superescalar que compõe cada núcleo: i) *single-issue* (SI), ii) *dual-issue* (DI) e iii) *four-issue* (FI). Cada versão do *multicore* foi simulada considerando dois cenários: *standalone* (sem a ALLEGRI) e com a ALLEGRI, totalizando assim 6 versões. A análise dessas versões do *multicore* foram feitas considerando consumo energético, desempenho e o EDP (*Energy-Delay Product*). Para o conjunto de aplicações simuladas, as versões que tiveram o melhor consumo energético, desempenho e EDP são SI + ALLEGRI, FI + ALLEGRI, SI + ALLEGRI, respectivamente. Adicionalmente, também foi avaliado o custo adicional em área da integração da ALLEGRI no processador *multicore*. A integração da ALLEGRI, no pior caso (SI + ALLEGRI), teve um custo adicional em área de apenas 17,7%, enquanto que melhora, em média, para esta mesma versão, o consumo energético e o desempenho em 30% e 42,85%, respectivamente.

Como trabalho futuro, pretende-se avaliar um MPSoC heterogêneo composto por um núcleo que obteve o melhor desempenho (FI + ALLEGRI) e por um núcleo que obteve a melhor eficiência energética (SI + ALLEGRI). Adicionalmente, pretende-se avaliar outras configurações possíveis do MPSoC para avaliar o quanto a solução proposta é melhor, em termos de energia e desempenho, que as outras versões em um ambiente heterogêneo.

Agradecimentos

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

REFERÊNCIAS

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

[2] F. Vahid, G. Stitt, and R. Lysecky, "Warp processing: Dynamic translation of binaries to fpga circuits," *Computer*, vol. 41, no. 7, pp. 40–46, July 2008.

[3] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The chimaera reconfigurable functional unit," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 2, pp. 206–217, Feb 2004.

[4] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "Piperench: a reconfigurable architecture and compiler," *Computer*, vol. 33, no. 4, pp. 70–77, April 2000.

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

[6] J. D. Souza, L. Carro, M. B. Rutzig, and A. C. S. Beck, "A reconfigurable heterogeneous multicore with a homogeneous isa," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1598–1603.

[7] M. Brandalero, M. Shafique, L. Carro, and A. C. S. Beck, "Transrec: Improving adaptability in single-isa heterogeneous systems with transparent and reconfigurable acceleration," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 582–585.

[8] A. C. S. Beck, M. B. Rutzig, G. Gaydadjiev, and L. Carro, "Transparent reconfigurable acceleration for heterogeneous embedded applications," in *2008 Design, Automation and Test in Europe*, March 2008, pp. 1208–1213.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[10] ARM, "big.little technology: The future of mobile," Tech. Rep., 2013.

[11] S. M. Z. Iqbal, Y. Liang, and H. Grahn, "Parmibench - an open-source benchmark for embedded multiprocessor systems," *IEEE Computer Architecture Letters*, vol. 9, no. 2, pp. 45–48, Feb 2010.

[12] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2016, pp. 235–244.

[13] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 642–649.

[14] F. Liu, H. Ahn, S. R. Beard, T. Oh, and D. I. August, "Dynaspam: Dynamic spatial architecture mapping using out of order instruction schedules," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 541–553.

[15] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, May 2005.

[16] F. C. Junior, I. Silva, and R. Jacobi, "A partially shared thin reconfigurable array for multicore processor," in *Anais do IX Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. Porto Alegre, RS, Brasil: SBC, 2019, pp. 113–118. [Online]. Available: https://sol.sbc.org.br/index.php/sbesc_estendido/article/view/8645

[17] G. Korol, M. Jordan, M. Brandalero, M. B. Rutzig, and A. C. S. Beck, "Power-aware phase oriented reconfigurable architecture," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 626–629.

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

[19] S. Xu and B. Carrion Schafer, "Deep: Dedicated energy-efficient approximation for dynamically reconfigurable architectures," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 587–594.

[20] D. A. Patterson and J. L. Henessy, *Computer Architecture: A Quantitative Approach*, 5th ed. Oxford, USA: Morgan Kaufmann, 2016.

[21] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers*, vol. C-28, no. 9, pp. 690–691, Sep. 1979.

[22] A. Beck, M. Rutzig, G. Gaydadjiev, and L. Carro, "Transparent reconfigurable acceleration for heterogeneous embedded applications," in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 1208–1213.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[24] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 469–480.

[25] R. B. N. Muralimanohar and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, Tech. Rep., 2009. [Online]. Available: http://www.hpl.hp.com/techreports/2009/HPL- 2009-85.html

[26] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "Dyser: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, 2012.

[27] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, Sep. 1996.

[28] G. Z. Chrysos and J. S. Emer, "Memory dependence prediction using store sets," in *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, July 1998, pp. 142–153.

Francisco Carlos Silva Junior É graduado em ciência da computação pela Universidade Federal do Piauí (UFPI), em 2015; Mestre em ciência da computação pela UFPI, em 2018. Atualmente é doutorando na Universidade de Brasília (UnB), desde de 2018. Atua e tem interesse na área de aceleradores e arquiteturas reconfiguráveis, processadores multicore e sistemas heterogêneos.

Ivan Saraiva Silva Possui graduação em Engenharia Elétrica e Mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba. Possui Diplôme d'Études Approfondies (Mestrado) em Microélectronique et Microinformatique pela Universidade Pierre et Marie Curie (Paris VI) e Doutorado em Informática também pela Universidade Pierre et Marie Curie (Paris VI). Foi de 1996 a 2009 professor da Universidade Federal do Rio Grande do Norte (Departamento de Informática e Matemática Aplicada), sendo atualmente professor Associado IV da Universidade Federal do Piauí (Departamento de Computação). Atua nas áreas de Arquitetura de Computadores e Concepção de Sistemas integrados e Sistemas Embarcados.

Ricardo Pezzuol Jacobi concluiu o doutorado em Ciências Aplicadas na Université Catholique de Louvain - Bélgica, em 1993. Foi professor do Instituto de Informática da UFRGS no período de 1989 a 1998. Desde 1998 é professor do Departamento de Ciência da Computação da Universidade de Brasilia. Foi Diretor do Instituto de Ciências Exatas da UnB no período de 2004 a 2007 e Vice-Diretor do Campus UnB-Gama, de 2008 a 2012. Fundou o Capítulo da IEEE Circuits and Systems em 2012 e foi Presidente da Seção Centro-Norte Brasil da IEEE no período de 2015 a 2018. Presidente do Conselho Brasil da IEEE em 2018 e 2019. Atua na área de Ciência da Computação e Microeletrônica, com ênfase em CAD, arquiteturas reconfiguráveis e sistemas monoliticos (SoC).