Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Igor Felipe Gallon

Framework para estudos de mapeamentos inteligentes de grafos de multiaplicações em Arquiteturas Many-Core

São Carlos
2026

Gallon, Igor Felipe

Framework para estudos de mapeamentos inteligentes de grafos de multiaplicações em arquiteturas many-core / Igor Felipe Gallon -- 2026.
89f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Emerson Carlos Pedrino
Banca Examinadora: Paulo Rogério Politano, Mauro Masili
Bibliografia

1. Many-core. 2. Network-on-Chip. 3. Otimização multiobjetivo. I. Gallon, Igor Felipe. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Arildo Martins - CRB/8 7180

Folha de Aprovação

Defesa de dissertação de mestrado do(a) candidato(a) Igor Felipe Gallon, realizada em 01/04/2026

Comissão Julgadora

Prof(a) Dr(a) Emerson Carlos Pedrino (UFSCar)

Prof(a) Dr(a) Paulo Rogério Politano (UFSCar)

Prof(a) Dr(a) Mauro Masili (USP)

O relatório de defesa assinado pelos membros da comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação

Igor Felipe Gallon

Framework para estudos de mapeamentos inteligentes de grafos de multiaplicações em Arquiteturas Many-Core

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Arquiteturas de Computadores (SDARC)

Orientador: Emerson Carlos Pedrino

São Carlos
2026

Dedico este trabalho à minha amada Marina e aos meus pais, Adriana e Márcio, pelo apoio, incentivo e compreensão ao longo desta jornada.

Agradecimentos

Agradeço, primeiramente, a Deus, pois todas as coisas foram feitas por intermédio d'Ele e, sem Ele, nada do que foi feito se fez.

Ao professor Emerson, pela orientação, dedicação e pelos valiosos ensinamentos, tanto acadêmicos quanto de vida, que foram fundamentais para a realização deste trabalho.

Resumo

O aumento da demanda por processamento de dados em aplicações intensivas, como sistemas inteligentes e aplicações em fluxo (*stream*), tem impulsionado o uso de arquiteturas Multicore e Many-Core como alternativa aos sistemas de núcleo único. Nesse contexto, as arquiteturas baseadas em Network-on-Chip (NoC) destacam-se como uma solução eficiente para comunicação entre múltiplos núcleos. Entretanto, o problema de mapeamento de tarefas nessas arquiteturas, especialmente em cenários com múltiplas aplicações concorrentes, permanece desafiador, uma vez que envolve a otimização simultânea de diversas métricas, como consumo de energia, latência, balanceamento de carga e tolerância a falhas.

Diante desse cenário, este trabalho propõe uma *framework* para o estudo de mapeamentos de tarefas em arquiteturas many-core baseadas em NoC, com foco em ambientes de multiaplicações. A abordagem integra um módulo de otimização multiobjetivo, desenvolvido em MATLAB, responsável pela geração de soluções de mapeamento e análise da curva de Pareto, com um simulador de NoC de alto nível, implementado em Python, que permite avaliar o impacto dos mapeamentos por meio da simulação de tráfego de pacotes na rede. A ferramenta proposta possibilita a exploração sistemática do espaço de projeto, permitindo a análise comparativa de diferentes estratégias de mapeamento, a visualização de métricas de desempenho e a obtenção de resultados estatísticos. Além disso, a abstração adotada no simulador favorece a experimentação rápida e a avaliação de cenários complexos, sem a necessidade de modelagem detalhada em nível de hardware.

Os resultados obtidos indicam que a *framework* é eficaz no suporte a estudos de mapeamento multiobjetivo em arquiteturas many-core, contribuindo como uma ferramenta flexível e acessível para análise de desempenho, ensino e desenvolvimento de novas estratégias de mapeamento em sistemas baseados em NoC.

Palavras-chave: Many-Core. Network-on-Chip. Otimização Multiobjetivo. Framework. Mapeamento de Tarefas. Multiaplicações. Simulação.

Abstract

The increasing demand for data processing in compute-intensive applications, such as intelligent systems and streaming applications, has driven the adoption of Multicore and Many-Core architectures as an alternative to single-core systems. In this context, Network-on-Chip (NoC)-based architectures have emerged as an efficient solution for communication among multiple cores. However, the problem of task mapping in these architectures, especially in scenarios involving multiple concurrent applications, remains challenging, as it requires the simultaneous optimization of several metrics, such as energy consumption, latency, load balancing, and fault tolerance.

In this context, this work proposes a framework for the study of task mapping in NoC-based many-core architectures, with a focus on multi-application environments. The approach integrates a multi-objective optimization module, developed in MATLAB, responsible for generating mapping solutions and analyzing the Pareto front, with a high-level NoC simulator implemented in Python, which enables the evaluation of mapping strategies through packet-level network traffic simulation. The proposed tool enables systematic design space exploration, allowing comparative analysis of different mapping strategies, visualization of performance metrics, and extraction of statistical results. Furthermore, the abstraction level adopted in the simulator supports rapid experimentation and the evaluation of complex scenarios without requiring detailed hardware-level modeling.

The obtained results indicate that the proposed framework is effective in supporting multi-objective task mapping studies in many-core architectures, contributing as a flexible and accessible tool for performance analysis, education, and the development of new mapping strategies in NoC-based systems.

Keywords: Many-Core. Network-on-Chip. Multi-objective optimization. Framework. Tasks mapping. Multi-application. Simulation.

Lista de ilustrações

Lista de tabelas

Lista de siglas

APG Application Process Graph (Grafo de Processos da Aplicação)

CI Circuito Integrado

DR Diagonal Raster (Varredura Diagonal)

DS Diagonal Snake ("Cobra"diagonal)

GUI Graphical User Interface (Interface Gráfica do Usuário)

HLS High-Level Synthesis (Síntese de Alto Nível)

HR Horizontal Raster (Varredura Horizontal)

HS Horizontal Snake ("Cobra"Horizontal)

LT Left-Turn

NoC Network-on-chip

NF Negative First

NL North Least

OE Odd Even

PE Processing Element (Elemento de Processamento)

RTL Register-Transfer Level (Nível de transferência de registradores)

SoC Systems-on-Chip (Sistema em um chip)

WF West First

Sumário

Capítulo 1

Introdução

A frequência máxima de operação de processadores de núcleo único tem atingido limites práticos devido à dissipação de energia, restrições térmicas e outros fatores físicos. Com a redução dos ganhos obtidos por escalonamento de tensão e frequência nas gerações tecnológicas mais recentes, a indústria passou a explorar o paralelismo em nível de tarefas por meio de arquiteturas multicore e many-core (HILL; MARTY, 2008; BORKAR, 2007; JERRAYA; WOLF, 2005). Nessas arquiteturas, múltiplos núcleos de processamento operam em paralelo, aumentando a taxa de transferência (*throughput*) do sistema sem depender exclusivamente do aumento da frequência de operação.

Ao mesmo tempo, a demanda computacional de aplicações modernas — como processamento de vídeo, análise de grandes volumes de dados e aprendizado de máquina — tem crescido substancialmente e exige cada vez mais capacidade de processamento paralelo (JANG et al., 2021; LIU; KATO; EDAHIRO, 2019; ACHBALLAH; OTHMAN; SAOUD, 2017; HASSAN; MORGAN; EL-KHARASHI, 2017). Como consequência, plataformas multicore e many-core tornaram-se um paradigma dominante em Systems-on-Chip (Sistema em um chip) (SoC), exigindo infraestruturas de comunicação eficientes e escaláveis entre os diversos elementos de processamento.

Com o avanço tecnológico, o número de núcleos integrados em um mesmo Circuito Integrado (CI) tem aumentado rapidamente, frequentemente combinando diferentes tipos de núcleos para atender requisitos funcionais e não funcionais do sistema. Avanços na nanotecnologia indicam que será possível integrar milhares de núcleos em um único chip, caracterizando a chamada era dos sistemas *many-core* (MIAO et al., 2025; BORKAR, 2007). Nesse contexto, aplicações são tipicamente modeladas como conjuntos de tarefas menores que podem ser distribuídas entre múltiplos núcleos e executadas em paralelo (KOBBE et al., 2011; FARUQUE; KRIST; HENKEL, 2008).

À medida que o número de elementos de processamento cresce, o projeto de infraestruturas de comunicação escaláveis torna-se um desafio fundamental. Mecanismos tradicionais de interconexão em chip, como barramentos compartilhados e crossbars, apresentam limitações significativas de escalabilidade, complexidade de interconexão e consumo de energia quando aplicados a sistemas com grande número de núcleos. Nesse cenário, as NoCs emergem como um paradigma eficiente de comunicação em arquiteturas many-core, fornecendo uma rede estruturada composta por roteadores e enlaces responsáveis pela troca de dados entre os elementos de processamento (GUPTA; BHARGAVA; INDU, 2021; HENKEL; WOLF; CHAKRADHAR, 2004; BENINI; MICHELI, 2002). Ao adotar comunicação baseada em pacotes, as NoCs permitem maior escalabilidade e melhor suporte aos requisitos de desempenho e eficiência energética desses sistemas.

Nesse ambiente, surge o desafio de particionar e mapear eficientemente as tarefas das aplicações nos diversos núcleos disponíveis. O particionamento envolve identificar tarefas, estabelecer mecanismos de sincronização, gerenciar o acesso à memória e garantir a correta execução paralela do código (CENG et al., 2008). O mapeamento de tarefas define o posicionamento das tarefas na arquitetura e os padrões de comunicação entre elas, visando otimizar métricas de hardware como consumo de energia, desempenho computacional, tensão e frequência (XU; LEPPäNEN, 2016). Trata-se de um problema complexo, fortemente dependente das características da arquitetura many-core considerada (MITTAL, 2016; TENDULKAR, 2014; SINGH et al., 2013).

Encontrar um mapeamento ótimo que satisfaça simultaneamente todos os requisitos do sistema é extremamente difícil na prática. De fato, o problema de mapeamento de tarefas em sistemas many-core é geralmente classificado como NP-difícil (SINGH et al., 2013). Consequentemente, heurísticas e meta-heurísticas baseadas em conhecimento do domínio da aplicação são frequentemente empregadas para encontrar soluções eficientes. À medida que o número de aplicações e tarefas cresce, o espaço de busca de possíveis mapeamentos aumenta exponencialmente, tornando inviável uma exploração exaustiva. Por essa razão, o problema de mapeamento de tarefas permanece como um dos desafios centrais no desenvolvimento de sistemas many-core eficientes (MARWEDEL et al., 2011; MARCULESCU et al., 2009).

As metodologias de mapeamento podem ser classificadas de acordo com o momento em que as decisões são tomadas. Em cenários com cargas de trabalho estáticas, os mapeamentos são geralmente definidos em **tempo de projeto**, utilizando conhecimento global sobre os padrões de computação e comunicação das aplicações. Em ambientes dinâmicos, por outro lado, são empregadas técnicas de mapeamento em **tempo de execução**, nas quais as tarefas são alocadas e eventualmente migradas durante a operação do sistema (ORSILA et al., 2007; THIELE et al., 2007). Nesses casos, um ou mais núcleos podem ser responsáveis pelo escalonamento de tarefas, gerenciamento de recursos e controle de configuração do sistema, podendo-se adotar estratégias centralizadas, distribuídas ou

híbridas (HONG et al., 2009; THEOCHARIDES et al., 2009; FARUQUE; KRIST; HENKEL, 2008). Enquanto abordagens em tempo de projeto tendem a produzir mapeamentos altamente otimizados, abordagens em tempo de execução oferecem maior flexibilidade e adaptabilidade às condições dinâmicas do sistema.

Geralmente, mapeamentos em tempo de projeto utilizam técnicas de computação evolutiva visando-se otimizações que implicam em altos custos computacionais, o que pode ser inaceitável para problemas de larga escala. Entretanto, fornecem soluções eficientes de mapeamento para sistemas de pequena escala dentro de tempos aceitáveis. Em contrapartida, mapeamentos em tempo de execução devem levar em consideração o tempo necessário para configurar cada tarefa, o qual se somará ao tempo total de execução da referida aplicação. Assim, todas as tarefas são mapeadas uma a uma, ao contrário do caso estático, em que todas as tarefas são mapeadas de uma só vez, olhando-se para o sistema de forma global. Logo, algoritmos determinísticos (LIU; TEMPESTI, 2024; KHULLER; YANG, 2018; BINKERT et al., 2011) são usados para mapeamentos eficientes nessa abordagem, visando-se otimizar as métricas de desempenho pretendidas pelo usuário, conforme já sobredito. Algumas vantagens desse tipo de mapeamento são: adaptabilidade aos recursos disponíveis, habilidade de evitar núcleos defeituosos. Os mapas criados por esse método não são tão bons quantos aos mapas criados pelo mapeamento em tempo de projeto. Como alternativa, há técnicas de mapeamento híbrido que exploram os benefícios de ambas as abordagens (EMERETLIS et al., 2022; POURMOHSENI et al., 2020). Portanto, um conjunto de mapas ótimos são gerados em tempo de projeto para serem utilizados em tempo de execução de forma mais eficiente, de acordo com as restrições impostas pelo usuário (BONNEY et al., 2016). No entanto os mapas precisam ser gerados novamente em caso de novas aplicações. Além do problema de mapeamento já mencionado, outro em aberto e que ainda é desafiador é o de geração de mapas de alocação de tarefas utilizando-se várias aplicações rodando em sistemas de múltiplos/muitos núcleos tentando otimizar vários objetivos simultaneamente.

A avaliação de diferentes estratégias de mapeamento de tarefas e padrões de comunicação em arquiteturas many-core requer ferramentas adequadas de exploração do espaço de projeto. Nesse contexto, frameworks e ferramentas voltadas ao estudo de mapeamentos em tempo de projeto desempenham papel fundamental na análise sistemática de desempenho, no desenvolvimento e comparação de estratégias de alocação de tarefas e na investigação de compromissos entre múltiplos objetivos de otimização, como latência, throughput e consumo de energia (XU; LEPPäNEN, 2016).

Tais ferramentas permitem explorar diferentes configurações arquiteturais, algoritmos de roteamento e políticas de mapeamento antes da implementação do sistema. Entretanto, muitos simuladores de estado da arte são fortemente orientados à modelagem de baixo nível, frequentemente implementados em linguagens como C/C++ ou SystemC e baseados em simulação ciclo-a-ciclo. Embora ofereçam alta fidelidade em relação ao comportamento

do hardware, essas ferramentas geralmente exigem descrições arquiteturais detalhadas e processos de configuração complexos, o que limita sua utilização em estudos conceituais, exploração rápida de soluções e contextos educacionais.

Com o objetivo de facilitar a investigação metodológica de estratégias de mapeamento em arquiteturas many-core, este trabalho propõe uma framework voltada ao estudo de mapeamentos em tempo de projeto, integrando ferramentas de otimização multiobjetivo e simulação de comunicação em NoC. A geração de soluções de mapeamento é realizada por meio de algoritmos de otimização multiobjetivo implementados em *MATLAB*, responsáveis por explorar o espaço de soluções e produzir conjuntos de mapeamentos candidatos de acordo com diferentes métricas de desempenho. Para avaliar o impacto desses mapeamentos no comportamento da rede de interconexão, a framework incorpora também um simulador de NoC de alto nível, denominado *SimpyNoC*, implementado em Python. O simulador abstrai detalhes microarquiteturais de baixo nível e prioriza flexibilidade e facilidade de modificação, permitindo a rápida implementação e comparação de diferentes algoritmos de roteamento e estratégias de comunicação. Dessa forma, a framework proposta fornece um ambiente integrado para geração, avaliação e análise de mapeamentos de tarefas em arquiteturas many-core.

1.1 Motivação

Após a revisão da literatura, constatou-se que, apesar da existência de diversos trabalhos dedicados ao problema de mapeamento de tarefas em tempo de projeto para arquiteturas Many-Core, ainda são escassos os trabalhos que oferecem um *framework* integrado, com Graphical User Interface (Interface Gráfica do Usuário) (GUI) nativa, que permita ao pesquisador realizar estudos de técnicas e parametrizações de mapeamento de forma mais objetiva, flexível e interativa, além de facilitar a visualização e análise dos resultados obtidos. A disponibilidade de ferramentas com essas características é particularmente importante para apoiar a exploração do espaço de projeto e a comparação sistemática de diferentes estratégias de mapeamento.

Além disso, observa-se que muitos trabalhos na literatura consideram cenários simplificados de mapeamento, frequentemente envolvendo aplicações isoladas e desconsiderando a execução simultânea de múltiplas aplicações sobre a mesma infraestrutura de comunicação baseada em NoC. Outro aspecto relevante é que diversas técnicas de mapeamento encontradas na literatura concentram-se na otimização de uma única métrica de desempenho, como latência ou consumo de energia, não explorando adequadamente o caráter multiobjetivo do problema. Na prática, arquiteturas Many-Core frequentemente exigem a consideração simultânea de múltiplos critérios de otimização, tais como consumo de energia, balanço de carga, desempenho e tolerância a falhas.

Adicionalmente, existe também uma lacuna relacionada à disponibilidade de simula-

dores de NoC de alto nível voltados à exploração conceitual e metodológica de estratégias de mapeamento. Muitos simuladores amplamente utilizados na literatura são fortemente orientados à modelagem de baixo nível e à simulação ciclo-a-ciclo, o que, embora proporcione alta fidelidade de hardware, pode dificultar sua utilização em estudos exploratórios, prototipagem rápida de algoritmos e contextos educacionais. Dessa forma, ferramentas de simulação mais leves e flexíveis podem desempenhar papel importante no apoio à análise de estratégias de comunicação e de mapeamento em arquiteturas many-core.

Diante desse cenário, as principais motivações deste trabalho incluem a proposta de um *framework* para estudos de mapeamento de múltiplas aplicações do tipo *stream* de dados, permitindo que diferentes aplicações sejam mapeadas simultaneamente em uma arquitetura baseada em NoC. A abordagem proposta também explora o uso de algoritmos evolutivos para otimização multiobjetivo, considerando simultaneamente métricas como consumo de energia, balanço de carga e tolerância a falhas.

Inicialmente, o *framework* foi implementado em MATLAB, utilizando a *toolbox* PlatEMO para execução de algoritmos de otimização multiobjetivo aplicados ao problema de mapeamento em arquiteturas Many-Core. Em uma etapa complementar, foi desenvolvido um simulador de NoC de alto nível em Python, denominado *SimpyNoC*, juntamente com uma GUI para visualização e análise dos resultados. Essa integração possibilitou a realização de experimentos de mapeamento mais realistas por meio da simulação de tráfego de pacotes na rede e da execução paralela de tarefas, mantendo ao mesmo tempo a flexibilidade necessária para experimentação e avaliação de diferentes estratégias de mapeamento.

1.2 Objetivos Gerais

O objetivo geral deste trabalho é desenvolver uma *framework* para o estudo de mapeamento de tarefas em arquiteturas *many-core* baseadas em NoC. A framework proposta visa permitir a exploração sistemática de diferentes estratégias de mapeamento, para diferentes aplicações, possibilitando a investigação e otimização simultânea de múltiplas métricas relevantes ao sistema, como consumo de energia, tolerância a falhas e balanço de carga entre os elementos de processamento. Além disso, busca-se viabilizar a análise do impacto dessas estratégias sobre o desempenho da rede de interconexão, por meio da avaliação de métricas de comunicação e comportamento da NoC. Dessa forma, a ferramenta desenvolvida pretende apoiar estudos metodológicos e experimentais voltados à análise e comparação de técnicas de mapeamento em arquiteturas *many-core*.

1.3 Objetivos Específicos

Para alcançar o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

- ❏ Investigar técnicas de mapeamento de tarefas em arquiteturas *many-core* baseadas em NoC, com ênfase em abordagens de mapeamento em tempo de projeto (Seção 2).
- ❏ Desenvolver um módulo de otimização multiobjetivo para geração de mapeamentos de tarefas, utilizando algoritmos evolutivos implementados em MATLAB por meio da *toolbox* PlatEMO (Seção 3.3).
- ❏ Integrar à *framework* mecanismos para avaliação de múltiplas métricas de otimização, como consumo de energia, tolerância a falhas e balanço de carga entre os elementos de processamento (Seção 3.3).
- ❏ Desenvolver um simulador de NoC de alto nível em Python, permitindo a simulação de comunicação baseada em pacotes e a análise do comportamento da rede sob diferentes estratégias de mapeamento e configurações da rede NoC (Seção 3.5).
- ❏ Implementar mecanismos de visualização e análise dos resultados experimentais, permitindo a comparação entre diferentes estratégias de mapeamento e seus impactos nas métricas de desempenho da rede (Seção 4).

1.4 Organização do texto

Este trabalho está estruturado em 4 capítulos, além desta introdução, organizados de forma a apresentar, de maneira progressiva, os conceitos, a metodologia proposta e os resultados obtidos.

No Capítulo 2, são discutidos os trabalhos relacionados, destacando abordagens existentes para mapeamento de aplicações e simuladores de NoCs disponíveis na literatura.

No Capítulo 3, é descrita a metodologia proposta, incluindo o módulo de otimização multiobjetivo e o simulador NoC desenvolvido.

No Capítulo 4, são apresentados os experimentos realizados, bem como a análise dos resultados obtidos, com foco na avaliação do desempenho dos mapeamentos e do simulador.

Por fim, no Capítulo 4.2.3, são apresentadas as conclusões do trabalho, suas limitações e possíveis direções para trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta uma revisão da literatura sobre mapeamento de tarefas em arquiteturas *many-core* baseadas em NoC e sobre ferramentas utilizadas para avaliação dessas arquiteturas. Na Seção 2.1, são discutidas diferentes estratégias de mapeamento propostas na literatura. Em seguida, a Seção 2.2 apresenta uma revisão de simuladores de NoC, classificados de acordo com seu nível de modelagem. Essa análise permite contextualizar as principais ferramentas utilizadas na área e identificar limitações relacionadas à complexidade de modelagem e à exploração de estratégias de mapeamento em arquiteturas *many-core*.

2.1 Tipos de Mapeamentos

Na literatura relacionada ao problema de mapeamento de aplicações em arquiteturas Many-Core, diversas abordagens têm sido propostas visando otimizar simultaneamente diferentes métricas do sistema. Entre essas abordagens, destacam-se técnicas baseadas em algoritmos genéticos (AG) multiobjetivos (HARADA; ALBA, 2020), amplamente utilizadas para gerar automaticamente soluções de mapeamento considerando restrições energéticas e o escalonamento eficiente de aplicações em sistemas Multi e Many-Core. O crescente uso de dispositivos móveis e a preocupação com sustentabilidade energética também têm motivado o desenvolvimento de soluções voltadas à redução do consumo de energia, conforme discutido na revisão sistemática apresentada em (MURALIDHAR; BOROVICA-GAJIC; BUYYA, 2022).

Diversos trabalhos exploram técnicas evolutivas para resolver o problema de mapeamento ou particionamento de tarefas. Em (AYARI et al., 2016), é proposto um AG híbrido que avalia o impacto de diferentes operadores de cruzamento, introduzindo um

novo operador guiado por escalabilidade que apresenta vantagens em relação aos operadores clássicos. De forma semelhante, em (ZHU; DING, 2017) é proposto um AG aprimorado para particionamento de tarefas em arquiteturas multicore heterogêneas, com foco na redução do consumo de energia. Já em (JIANG; XU; CHEN, 2021), os autores utilizam um algoritmo genético para o escalonamento de tarefas em sistemas multicore heterogêneos integrados com FPGA, tendo como objetivo minimizar o tempo total de execução da aplicação.

Além das abordagens baseadas em algoritmos evolutivos, diversos trabalhos investigam técnicas de mapeamento estático em tempo de projeto. Em (ROCHA et al., 2020), é proposta uma abordagem que utiliza o grafo de comunicação da aplicação para posicionar tarefas com alta dependência de comunicação em regiões próximas da malha da NoC, reduzindo o custo de comunicação e melhorando o desempenho da rede. Em (BOLCHINI et al., 2013), os autores geram uma base de dados contendo mapas otimizados em tempo de projeto para diferentes aplicações, os quais podem ser posteriormente ajustados em tempo de execução de acordo com o estado corrente do sistema, visando melhorar a eficiência de comunicação ou o tempo de vida da arquitetura. De forma semelhante, em (XU; LIU; YANG, 2018), é apresentado um método de otimização multiobjetivo para o projeto de arquiteturas NoC específicas para aplicações, considerando simultaneamente métricas como potência, área e atraso.

Outra linha de pesquisa considera aspectos de confiabilidade e tolerância a falhas no processo de mapeamento. Em (CHOU; MARCULESCU, 2011), núcleos sobressalentes são distribuídos ao longo da arquitetura para reduzir o consumo de energia de comunicação e melhorar o desempenho do sistema em caso de falhas. Em (LE et al., 2015), um algoritmo baseado em enxame de partículas é utilizado para balancear simultaneamente confiabilidade de chaveamento de núcleos, consumo de energia e tempo de comunicação. Mais recentemente, em (REDDY; RAHMAN; LAY-EKUAKILLE, 2024), é proposta uma estratégia de mapeamento tolerante a falhas em arquiteturas many-core baseadas em NoC, na qual o mapeamento das tarefas e a seleção de núcleos de reserva são definidos em tempo de projeto, permitindo remapeamento em tempo de execução quando falhas permanentes são detectadas.

Outras abordagens investigam o uso de técnicas de aprendizado de máquina para o problema de mapeamento. Como exemplo, em (TONETTO et al., 2020) é apresentado um método baseado em redes neurais artificiais que realiza o mapeamento de aplicações em tempo de execução em sistemas multicore heterogêneos, visando maximizar a resiliência do sistema em termos da carga de trabalho média entre falhas. Embora trabalhos dessa natureza possuam objetivos relacionados ao presente estudo, eles não são explorados nesta proposta por não utilizarem algoritmos genéticos como estratégia principal de mapeamento.

Além das estratégias mencionadas, uma linha importante de pesquisa investiga o

mapeamento de múltiplas aplicações em arquiteturas NoC. Em muitos casos, os trabalhos existentes concentram-se em aplicações individuais, explorando de forma limitada o grande número de núcleos disponíveis em sistemas many-core. Nesse contexto, em (SEPúLVEDA et al., 2011) é proposto o uso de um algoritmo imune multiobjetivo para otimizar simultaneamente latência e consumo de energia no mapeamento de múltiplas aplicações. De forma semelhante, em (YANG et al., 2010) é apresentada uma abordagem em duas etapas, na qual inicialmente é definida uma região da NoC para cada aplicação, seguida por um processo de otimização de latência e energia dentro dessa região.

Outros trabalhos também exploram estratégias para gerenciamento eficiente de múltiplas aplicações. Em (KHALILI; ZARANDI, 2012), é proposta uma técnica tolerante a falhas composta por duas fases: a primeira realiza o mapeamento das aplicações em núcleos funcionais, enquanto a segunda distribui núcleos ociosos entre esses núcleos ativos. Em (ZHU et al., 2014), os autores utilizam uma heurística eficiente para balancear a latência de pacotes e melhorar o desempenho do sistema em cenários de múltiplas aplicações. Em (KHASANOV; CASTRILLON, 2020), as aplicações podem alternar entre diferentes configurações de paralelismo de acordo com os recursos disponíveis, buscando minimizar o consumo energético sem violar restrições de tempo real.

Em (GE et al., 2021), é proposta uma abordagem de mapeamento em duas etapas: inicialmente são identificadas regiões potenciais da NoC por meio de análise geométrica, e posteriormente um algoritmo genético é utilizado para otimizar a latência dentro de cada região. Na etapa final, um algoritmo de simulated annealing baseado em B-Tree é aplicado para posicionar os blocos de forma a reduzir o consumo energético. Em (KUMAR; RAO; BEECHU, 2021), é apresentada a estratégia ERTEAM para mapeamento de multiaplicações embarcadas em tempo real, visando reduzir a distância média entre os núcleos e, consequentemente, a área ocupada no chip.

Outros trabalhos investigam estratégias dinâmicas, ou seja, em tempo de execução, para cenários de múltiplas aplicações. Em (ZHU et al., 2016), é proposta uma estratégia de mapeamento dinâmico que busca equilibrar simultaneamente a carga computacional dos núcleos e o tráfego de comunicação na rede. Em (LIU; TEMPESTI, 2024), os autores apresentam um método de remapeamento dinâmico baseado na previsão de envelhecimento do hardware, utilizando informações de temperatura e tráfego para redistribuir tarefas durante a execução. Finalmente, em (AMIN et al., 2023), é proposto um esquema híbrido de mapeamento que combina uma configuração inicial em tempo de projeto com adaptações em tempo de execução, considerando os recursos computacionais disponíveis para reduzir custos de energia e comunicação entre as tarefas.

2.2 Simuladores NoC

Ao longo das últimas duas décadas, diversos *frameworks* de simulação de NoC foram propostos, tendo como foco principal a validação microarquitetural e a avaliação orientada a hardware. Essas ferramentas podem ser classificadas em três categorias metodológicas: simuladores ciclo-a-ciclo, simuladores de alto nível e frameworks geradores de Register-Transfer Level (Nível de transferência de registradores) (RTL) (GAFFOUR et al., 2017). Cada categoria reflete um compromisso distinto entre fidelidade de modelagem, custo computacional, configurabilidade e usabilidade.

2.2.1 Simuladores ciclo-a-ciclo

Simuladores ciclo-a-ciclo concentram-se em precisão temporal, modelando pipelines de roteadores, controle de fluxo em nível de *flit*, *buffers*, mecanismos de arbitragem e contenção. Seu principal objetivo é a validação microarquitetural sob condições realistas de tráfego. Esse nível de detalhamento permite uma avaliação precisa de latência e vazão da rede, mas aumenta a complexidade da modelagem, o tempo de simulação e a complexidade de configuração.

Noxim (CATANIA et al., 2016), um dos simuladores de NoC de código aberto mais amplamente adotados, exemplifica essa categoria. Implementado em SystemC/C++, ele oferece ampla configurabilidade em algoritmos de roteamento, estratégias de arbitragem, dimensionamento de buffers e padrões de tráfego. De forma semelhante, **NoCTweak** (TRAN; BAAS, 2012) estende esse *framework* para incorporar análise de eficiência energética, enquanto **NIRGAM** (JAIN et al., 2007) fornece um ambiente modular e extensível para modelagem de redes com comutação por pacotes.

BookSim (JIANG et al., 2013) expande ainda mais a configurabilidade ao oferecer suporte a múltiplas topologias, esquemas de roteamento e métricas de desempenho, visando a avaliação detalhada de redes de interconexão em sistemas many-core. Em um nível ainda mais alto de integração, **gem5** (BINKERT et al., 2011), por meio de seu modelo **Garnet** (AGARWAL et al., 2009), incorpora modelagem de NoC ciclo-a-ciclo dentro de simulação arquitetural de sistema completo, permitindo a execução de cargas de trabalho realistas e a análise de desempenho em nível de sistema (IPC, tempo de execução da aplicação, cache misses)

Semelhantemente, (JOSEPH; PIONTECK, 2014) propuseram um simulador de NoC ciclo-a-ciclo com suporte para modelagem abstrata de grafos de tarefas. Diferentemente de geradores de tráfego puramente sintéticos, esse framework integra grafos de tarefas em nível de aplicação com simulação detalhada em nível de roteador, permitindo a avaliação do comportamento microarquitetural sob diferentes padrões de cargas de trabalho de comunicação. Embora essa abordagem seja mais realística, ela também reforça a ênfase na fidelidade em nível de ciclo e na modelagem detalhada de hardware.

DarSim (LIS et al., 2010) é um framework de simulação ciclo-a-ciclo para arquiteturas NoC e many-core, implementado em C++ com um mecanismo de simulação de eventos discretos. Ele modela detalhadamente o comportamento de roteadores, buffers e efeitos de contenção, ao mesmo tempo em que oferece suporte a topologias de rede e estratégias de roteamento configuráveis.

MuchiSim é um framework de simulação escalável para sistemas many-core baseados em múltiplos chiplets (chip contendo vários chips especializados e interconectados) que fornece modelagem ciclo-a-ciclo da comunicação e do movimento de dados (ORENES-VERA et al., 2024). Implementado em C++, ele permite exploração arquitetural em larga escala enquanto reporta métricas de desempenho, energia, área e custo, apoiando a avaliação de trade-offs de projeto em arquiteturas many-core distribuídas.

2.2.2 Simuladores de alto nível

Simuladores de alto nível priorizam a exploração arquitetural e a escalabilidade em detrimento da fidelidade microarquitetural detalhada. Em vez de modelar explicitamente os estágios de pipeline dos roteadores e mecanismos de controle de baixo nível, esses frameworks abstraem detalhes de hardware para permitir simulações mais rápidas e uma exploração mais ampla do espaço de projeto.

HNOCS (BEN-ITZHAK et al., 2012), implementado sobre o mecanismo de simulação de eventos discretos OMNeT++, fornece modelagem de NoC heterogênea com tipos de roteadores e canais virtuais configuráveis, mas não segue uma semântica de temporização estritamente ciclo-a-ciclo característica de simuladores microarquiteturais de baixo nível.

Naxim (NAKAJIMA et al., 2013) é um framework híbrido de simulação de NoC que integra modelagem de NoC baseada em SystemC com emulação de processadores via QEMU para permitir a simulação concorrente tanto da comunicação na rede quanto do comportamento dos processadores. Cada PE é representado por uma instância do QEMU, enquanto a interconexão NoC é modelada em SystemC, com a comunicação entre os componentes sendo realizada por meio de interfaces de *sockets*. Ele suporta topologias mesh configuráveis e esquemas de roteamento, tornando-o adequado para exploração arquitetural e avaliação de desempenho de projetos de NoC com núcleos emulados.

Embora essas ferramentas reduzam a complexidade de modelagem em comparação com frameworks totalmente ciclo-a-ciclo, elas permanecem principalmente orientadas à pesquisa arquitetural e à avaliação de desempenho, em vez de à clareza instrucional ou a fluxos experimentais simplificados.

2.2.3 Frameworks geradores de RTL

Proteus (BAMBHANIYA et al., 2023) adota uma metodologia geradora de RTL, diferindo-se de simuladores exclusivamente baseados em software. Ao utilizar High-Level

Synthesis (Síntese de Alto Nível) (HLS), ele converte uma descrição de NoC em C++ em RTL sintetizável, permitindo tanto simulação em software quanto validação em hardware baseada em FPGA a partir de uma especificação unificada. Essa abordagem estabelece uma ponte entre modelagem arquitetural e prototipagem em hardware, mas inerentemente desloca o foco para a viabilidade de implementação e implantação em hardware, em vez de exploração leve ou uso educacional.

2.2.4 Perspectiva geral

De maneira geral, a tendência na pesquisa em simulação de NoCs enfatiza a fidelidade em nível de hardware, a precisão temporal detalhada e a integração com RTL. A maioria dos simuladores adotados é implementada em SystemC ou C++ para garantir desempenho e precisão em nível de ciclo, conforme sumarizado na Tabela 1. No entanto, essa abordagem introduz uma sobrecarga de configuração não trivial, curvas de aprendizado mais acentuadas e acessibilidade limitada para experimentação rápida.

Consequentemente, ainda existe uma lacuna metodológica para frameworks de simulação que priorizem clareza conceitual, facilidade de modificação e acessibilidade em vez de detalhes microarquiteturais— especialmente em contextos nos quais o objetivo é a avaliação de algoritmos, análise de mapeamento de tarefas ou exploração educacional, em vez da validação de hardware.

Tabela 1 – Resumo comparativo de simuladores de NoC categorizados por nível de modelagem

Categoria	**Simulador**	**Linguagem**	**Topologias**	**Roteamento**	**Métricas**
Geradores de RTL	Proteus	C++ / HLS	Mesh	Customizado	Latência, Throughput, RTL
Ciclo-a-ciclo	Noxim	SystemC	Mesh	XY, NF, WF, NL, OE, LT, Adapt.	Latência, Throughput, Energia
	(JOSEPH; PIONTECK, 2014)	C++	Mesh	Determinístico	Latência, Throughput
	BookSim	C++	Mesh, Torus, Butterfly	Determinístico, Adapt.	Latência, Throughput
	NoCTweak	SystemC	Mesh, Torus, Ring	XY, NF, WF, NL, OE, LT	Latência, Throughput, Energia
	gem5 / Garnet	C++ / Python	Mesh	Configurável	Latência, Throughput, Sistema
	NIRGAM	SystemC	Mesh, Torus, Irregular	XY, OE, Source-based	Latência, Throughput
	MuchiSim	C++	Mesh	Configurável	Latência, Throughput, Energia, Área, Custo
	DarSim	C++	Mesh	Table-based, Custom.	Latência, Throughput
Alto nível	Naxim	SystemC	Mesh	Determinístico	Latência, Throughput
	HNOCS	C++	Heterogêneos	XY, Adapt.	Latência, Throughput
	Simulador proposto (SimpyNoC)	**Python**	**Mesh, Custom.**	**XY, NF, WF, NL, Custom.**	**Latência, Throughput, Custom.**

NF: Negative First (NF)
WF: West First (WF)
NL: North Least (NL)
OE: Odd Even (OE)
LT: Left-Turn (LT)

Capítulo 3

Metodologia

Este capítulo apresenta a metodologia e os detalhes de implementação da *framework* proposta neste trabalho. Inicialmente, a Seção 3.1 fornece uma visão geral da arquitetura da *framework* e de seus principais componentes. Em seguida, a Seção 3.2 descreve os objetos de entrada utilizados pela ferramenta, incluindo os grafos de aplicações e os parâmetros gerais de configuração. A Seção 3.3 apresenta a implementação do módulo de otimização multiobjetivo em MATLAB, responsável pela geração dos mapeamentos de tarefas. Na Seção 3.4 são descritos os mecanismos de visualização dos mapas gerados e das estatísticas associadas às soluções obtidas. Por fim, a Seção 3.5 apresenta a implementação do simulador de NoC em Python, utilizado para análise do comportamento da rede e avaliação das métricas de desempenho associadas aos diferentes mapeamentos.

3.1 Visão Geral

De maneira geral, a ferramenta, implementada em MATLAB e Python, consiste em quatro módulos, conforme o diagrama na Figura 1: módulo de entrada dos parâmetros e grafo de aplicações; procedimento multiobjetivo de mapeamento das tarefas; visualização dos mapeamentos obtidos e métricas; simulação de mapeamentos em NoC para obtenção de métricas de rede.

Para a implementação do *Framework* proposto neste projeto de pesquisa, foi utilizada a plataforma PlatEMO como base para realizar o processamento dos algoritmos evolutivos multiobjetivos pretendidos, uma vez que sua flexibilidade permite a customização das funções de custo, geração da população inicial, mecanismos de operações de cruzamento e mutação, entre outros, bem como a representação do problema de otimização de mape-

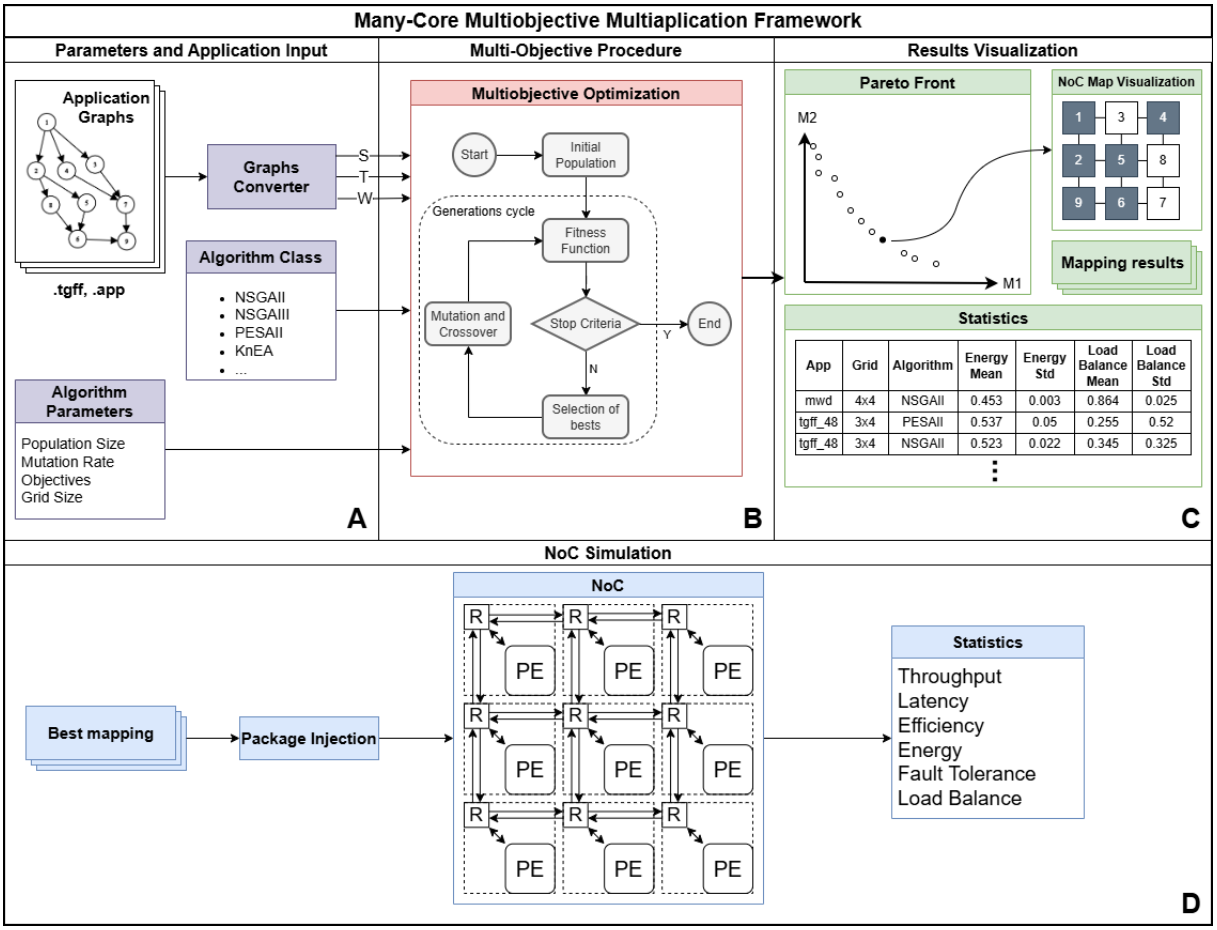


Figura 1 – Visão geral da ferramenta proposta

amento de tarefas de aplicações, representadas por grafos, em uma abstração de malha de processadores em um NoC.

O módulo de entrada (Fig. 1-A) converte grafos de aplicações *stream* sintéticos, por meio do conversor TGFF (DICK; RHODES; WOLF, 1998) integrado, e outros provenientes de *benchmarks* (TRAN; BAAS, 2012), que possuem interesse nas áreas de processamento digital de sinais e imagens, em um formato específico, a ser visto mais à frente na Seção 3.2.1, implementado no módulo de procedimento multiobjetivo. Nesta etapa, o usuário deve especificar os parâmetros da arquitetura a ser gerada, tais como: tamanho do *grid* do NoC, tipo do algoritmo multiobjetivo a ser utilizado e seus parâmetros internos, seleção dos grafos desejados e métricas de interesse (energia, tolerância a falhas, balanceamento de carga e latência, por exemplo).

O módulo de procedimento multiobjetivo (Fig. 1-B), implementado por meio de várias adaptações na ferramenta PlatEMO (TIAN et al., 2017) conforme explicado na Seção 3.3, gera uma abstração de arquitetura Many-Core e seu referido NoC, com roteamento X-Y, sendo responsável por gerar os mapeamentos otimizados dessas referidas aplicações de entrada na arquitetura proposta.

O módulo de resultados (Fig. 1-C) apresenta, caso a aplicação seja única, uma curva de Pareto, que permite ao usuário escolher a melhor solução de acordo com as métricas de interesse, ou uma relação de compromisso entre essas. Logo, um mapa para a referida aplicação, com a alocação de suas respectivas tarefas será gerado para fins de inspeção visual, além de uma tabela contendo resultados estatísticos em relação às métricas utilizadas. A ferramenta também permite avaliar várias aplicações simultaneamente. Neste último caso, é utilizado um ponto otimizado da curva de Pareto, que é calculado automaticamente, para cada uma dessas aplicações. Na tela de resultados, também é possível visualizar todas as métricas encontradas em cada cenário de estudo e suas respectivas estatísticas, conforme mostrado na Seção 3.4.

Por fim, o módulo de simulação de NoC (Fig. 1-D) em Python tem por objetivo validar e extrair métricas de rede, como largura de banda, latência média, perda de pacotes, entre outros, para o mapeamento de tarefas obtido nas etapas anteriores, conforme detalhado na Seção 3.5. A utilização desse simulador propõe ilustrar, de maneira flexível e didática, o comportamento na rede gerado pela troca constante de informações entre os nós dos processadores onde as tarefas da aplicação foram mapeadas, permitindo assim que a ferramenta de maneira geral forneça recursos para melhor entendimento e estudo de mapeamentos inteligentes em arquitetura Many-Core quando utilizado uma infraestrutura NoC.

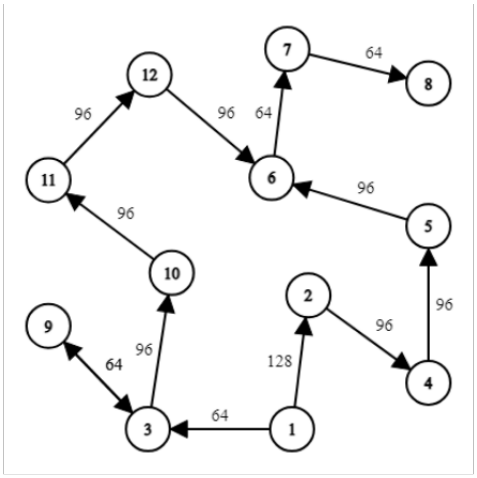
3.2 Módulo de Entrada

3.2.1 Representação das Aplicações

As aplicações, nesta versão de implementação, são tratadas como aplicações genéricas, ou seja, após o mapeamento das tarefas dessas referidas aplicações em seus respectivos nós processadores da arquitetura abstrata proposta de NoC, não há qualquer tratamento específico sobre tais dados. Essas aplicações genéricas, denominadas APG, são representadas por grafos unidirecionais de n vértices e m arestas. Cada tarefa pertencente à aplicação é representada por um vértice V_1, V_2, ... V_n e cada aresta P_1, P_2, ... P_m que interliga duas tarefas, possui um peso associado W_m, que representa o custo de comunicação entre as tarefas dentro da malha NoC.

Portanto, uma APG pode ser representada por três vetores inteiros de comprimento m, de modo que cada posição define uma aresta do grafo. Ou seja, para cada aresta i, S_i e T_i contêm os identificadores das tarefas de origem e destino, respectivamente, enquanto W_i representa o canal de comunicação e seu custo associado entre as tarefas S_i e T_i. No exemplo mostrado na Figura 2, a geração de dados está associada à Tarefa 1, enquanto as saídas finais da aplicação são produzidas pelas Tarefas 8 e 9. Entretanto, os mecanismos pelos quais os dados entram ou saem da NoC não são explicitamente modelados neste

trabalho, uma vez que o foco está restrito à comunicação entre as tarefas mapeadas dentro da rede.



S	1	1	2	4	5	6	7	3	9	3	10	11	12
T	2	3	4	5	6	7	8	9	3	10	11	12	6
W	128	64	96	96	96	64	64	64	64	96	96	96	96

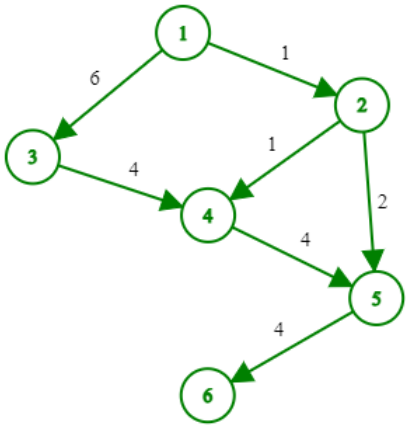
Figura 2 – Exemplo de representação de uma aplicação no formato APG

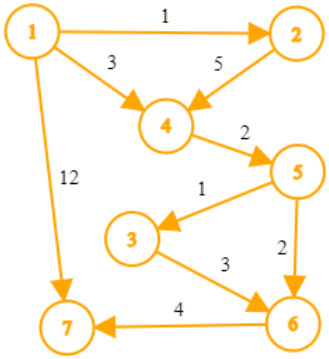
Na implementação proposta, também é possível realizar o mapeamento multiaplicação. Ou seja, é possível mapear as tarefas de mais de uma aplicação ao mesmo tempo na arquitetura NoC. Nesse modo, as tarefas de todas as aplicações são dispostas nos arranjos S, T e W sequencialmente, de maneira que o algoritmo genético interprete-os como uma única aplicação contendo sub-grafos não conectados, conforme o exemplo da Figura 3. A partir da segunda aplicação, os identificadores das tarefas são ajustados, recebendo identificadores virtuais, para que a sequência lógica de tarefas seja mantida. Após o processo evolutivo de mapeamento, os identificadores reais são recuperados, a fim de identificar os mapeamentos para cada APG.

3.3 Procedimento Multiobjetivo

3.3.1 PlatEMO

A plataforma PlatEMO é uma implementação *open-source* totalmente desenvolvida em MATLAB pelo BIMK (Institute of Bioinspired Intelligence and Mining Knowledge), da Universidade de Anhui, China, para otimização evolutiva de problemas (TIAN et al., 2017). Em outras palavras, essa plataforma contém mais de 200 algoritmos evolutivos implementados, dentre eles, os mais populares são: Algoritmos Genéticos, Otimização de Enxame de Partículas, MOEAs, entre outros. Também, conta com mais de 400 problemas de benchmark implementados e disponíveis para uso, permitindo ao usuário explorar





S	1	1	2	2	3	4	5	7 (1)	7 (1)	7 (1)	8 (2)	9 (3)	11 (5)	10 (4)	11 (5)	12 (6)
T	1	3	4	5	4	5	6	8 (2)	10 (4)	13 (7)	10 (4)	12 (6)	9 (3)	11 (5)	12 (6)	13 (7)
W	6	1	1	2	4	4	4	1	3	12	5	3	1	2	2	4

Figura 3 – Exemplo de representação de duas aplicações no formato APG multiaplicação

diversas técnicas, algoritmos e parametrizações para otimização de problemas, tanto de benchmarks quanto de problemas modelados de forma customizada.

Também possui uma interface gráfica amigável e de fácil entendimento, permitindo ao usuário fazer diversas parametrizações e estudos de técnicas e algoritmos para otimização de problemas de objetivos múltiplos, bem como extrair estatísticas e obter análises comparativas entre as técnicas utilizadas.

De maneira geral, o PlatEMO possui diversos algoritmos de otimização implementados e permite a utilização e customização dos mesmos por meio de interfaces de código. O diagrama de blocos da Figura 4 fornece uma visão geral de como as interfaces se comunicam. Inicialmente, o problema de otimização é parametrizado pelo tamanho da população, número de objetivos, número de variáveis de decisão, tipo de algoritmo de otimização (por exemplo, NSGAII, NSGAIII, PESAII, KnEA), tipo de operador para mutação e crossover, número máximo de gerações, entre outros.

Uma vez que os parâmetros são carregados para o contexto de execução, a população inicial é gerada por meio de uma interface que deve ser implementada pelo usuário. No contexto de Algoritmos Genéticos, os indivíduos da população, também chamados de cromossomos, devem então receber avaliação dos objetivos através da interface implementada pelo usuário. Após isso, o algoritmo de otimização se encarrega de aplicar estratégias de evolução ao longo das gerações, como a seleção dos melhores indivíduos e aplicação de operadores de mutação e crossover. Quando a população atinge o critério de parada, definido pelo número máximo de gerações, a evolução encerra retornando a população final. Nesse caso, cada indivíduo representa um mapeamento das tarefas para o NoC.

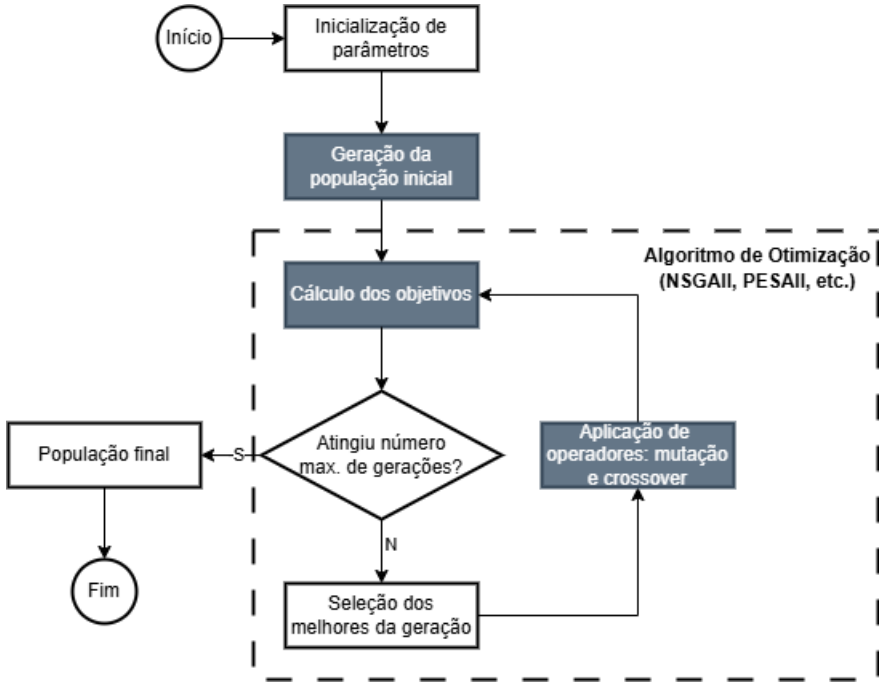


Figura 4 – Visão geral do funcionamento do PlatEMO

3.3.2 Representação dos indivíduos da população

Uma vez compreendido o funcionamento básico geral da plataforma PlatEMO, é necessário modelar o problema proposto de otimização de mapeamentos de tarefas provenientes de aplicações de *stream* em arquiteturas Many-Core para um problema customizado que possa ser interpretado por essa plataforma contendo algoritmos de otimização multiobjetivo. Logo, o problema consiste em, dado uma aplicação qualquer representada por grafos, gerar um mapeamento de tarefas para a malha de nós-processadores no NoC abstrato, com parâmetros determinados pelo usuário, conforme já supracitado, de forma a se obter o melhor equilíbrio entre as métricas pretendidas de um dado cenário hipotético de estudo, que também devem ser escolhidas de antemão pelo usuário. Assim, toda essa abstração deverá ser convertida em uma representação de cromossomo que possa ser interpretada pela plataforma base, conforme será discutido mais à frente.

Considere um cenário de alocação de tarefas de uma dada aplicação, em que necessita-se obter um mapa otimizado em relação ao gasto energético para a execução dessas tarefas e que ao mesmo tempo seja tolerante a falhas, e/ou possua o melhor balanceamento de cargas. Em outras palavras, espera-se encontrar um mapeamento otimizado de forma que as tarefas interdependentes fiquem em nós próximos, para que o caminho a ser percorrido para troca de dados através do NoC seja menor e, consequentemente, o seu custo de comunicação também seja menor. Porém deve-se haver a possibilidade de existência de nós ociosos, para que a referida configuração de mapeamento seja resiliente a falhas. Se houver algum erro de comunicação de links ou falha em algum nó no NoC, o sistema deverá ter a capacidade de realocar a tarefa desse nó falho para o nó ocioso mais próximo

sem grandes perdas de desempenho, como no exemplo da Figura 5 em que devido a uma falha no nó 8, a tarefa 3 foi realocada para o nó 12. Ao mesmo tempo, deve-se atentar para que não haja concentração de tarefas em uma única região do chip, a fim de evitar o superaquecimento da região e reduzir o tempo de vida útil do mesmo em um cenário de aplicação real.

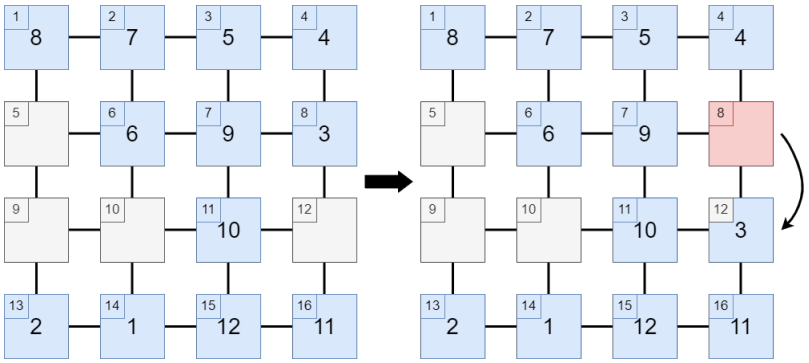
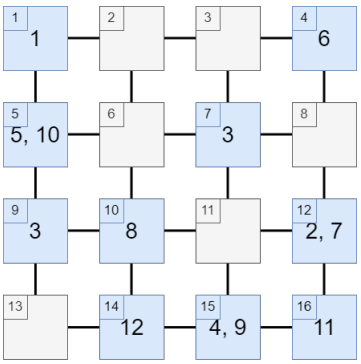


Figura 5 – Exemplo de realocação de uma tarefa no NoC

Em outras palavras, dada uma malha X-Y de nós processadores em um NoC, com r linhas e c colunas, sendo p a quantidade desses processadores, ou seja: $p = r \times c$, e uma APG com n tarefas, tal indivíduo corresponderá a um arranjo de comprimento n que contém valores $t_1, t_2, \ldots, t_n$, em que a cada t_i será atribuído um nó processador. Para um melhor entendimento, na Figura 6 pode-se observar a representação de um mapeamento hipotético para uma malha de processadores de dimensão 4×4. Observa-se que nesse mapeamento os processadores 2, 3, 6, 8, 11, e 13 não contêm tarefas alocadas, permanecendo, cada um, em estado ocioso. Esses nós ociosos permitem que caso algum processador ativo em execução falhe, o sistema possa realocar a referida tarefa para o processador mais próximo sem grandes perdas de desempenho.



t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
1	12	9	15	5	4	12	10	15	5	16	14

Figura 6 – Exemplo de cromossomo e sua representação

3.3.3 População Inicial

Na presente proposta, a população inicial pode ser gerada totalmente aleatória ou pelo uso de algoritmos determinísticos, como descrito por (BONNEY et al., 2016). No modo totalmente aleatório, os cromossomos são gerados pela permutação das tarefas em nós aleatórios, conforme o padrão descrito na Seção 3.3.2. Para a utilização de algoritmos determinísticos, também chamados de *Enginnered Mappings*, após gerado uma sequência aleatória de tarefas, as mesmas são mapeadas de maneira determinística para a malha de processadores do NoC. Os *Enginnered Mappings* são: Horizontal Raster (Varredura Horizontal) (HR), Horizontal Snake ("Cobra"Horizontal) (HS), Diagonal Raster (Varredura Diagonal) (DR) e Diagonal Snake ("Cobra"diagonal) (DS) e suas respectivas sequências determinísticas são exemplificadas na Figura 7 para a sequência de tarefas de 1 a 12 mapeadas em um *grid* 4×4. Por padrão, o mapeamento determinístico se inicia sempre com o nó superior mais à esquerda. Caso o número de tarefas seja maior que a quantidade de nós, os algoritmos determinísticos continuam a distribuição das tarefas restantes a partir do primeiro nó. A geração de uma sequência aleatória de tarefas antes do processamento permite que apesar do mapeamento seguir a mesma lógica, haverá variabilidade de possíveis soluções dentro de uma população de cromossomos.

3.3.4 Cálculo de Objetivos

Na ferramenta, é possível otimizar até 3 objetivos: energia (comunicação), balanço de carga e tolerância a falhas.

O custo de comunicação entre as tarefas alocadas é obtido pelo somatório das distâncias Manhattan de cada arco dos grafos da APG multiplicado pelo peso do arco, levando-se em consideração o roteamento X-Y para tráfego de dados no arranjo idealizado. Ou seja, para cada arco i que conecta duas tarefas s e t, no grafo de uma dada aplicação G, por exemplo, com n arestas, calcula-se para cada cromossomo C a distância Manhattan entre essas tarefas no mapeamento do *grid* (Eq. 1) multiplicado pelo peso associado w. O custo de comunicação também é multiplicado por uma constante k, onde k está relacionada à largura de banda do link em questão. Para a implementação desse projeto, foi considerado nos testes $k = 1$.

$$energy\_cost(C, G) = k * \sum_{i=1}^{n} manhattan\_dist(s_i, t_i) * w_i \quad (1)$$

Na Figura 8 é possível observar que dado a APG em 8.a, o peso da aresta que liga as tarefas 1 e 2 é 4 e no mapeamento 8.b as tarefas distam 3 blocos. Logo, o custo de comunicação entre tarefas é $3 * 4 = 12$. O mesmo se aplica para as demais arestas da APG, conforme ilustrado e portanto o custo total de comunicação do mapa apresentado é $(3 * 4) + (1 * 3) + (2 * 5) + (2 * 6) + (4 * 3) + (4 * 3) + (2 * 3) + (1 * 4) + (1 * 4) = 75$.

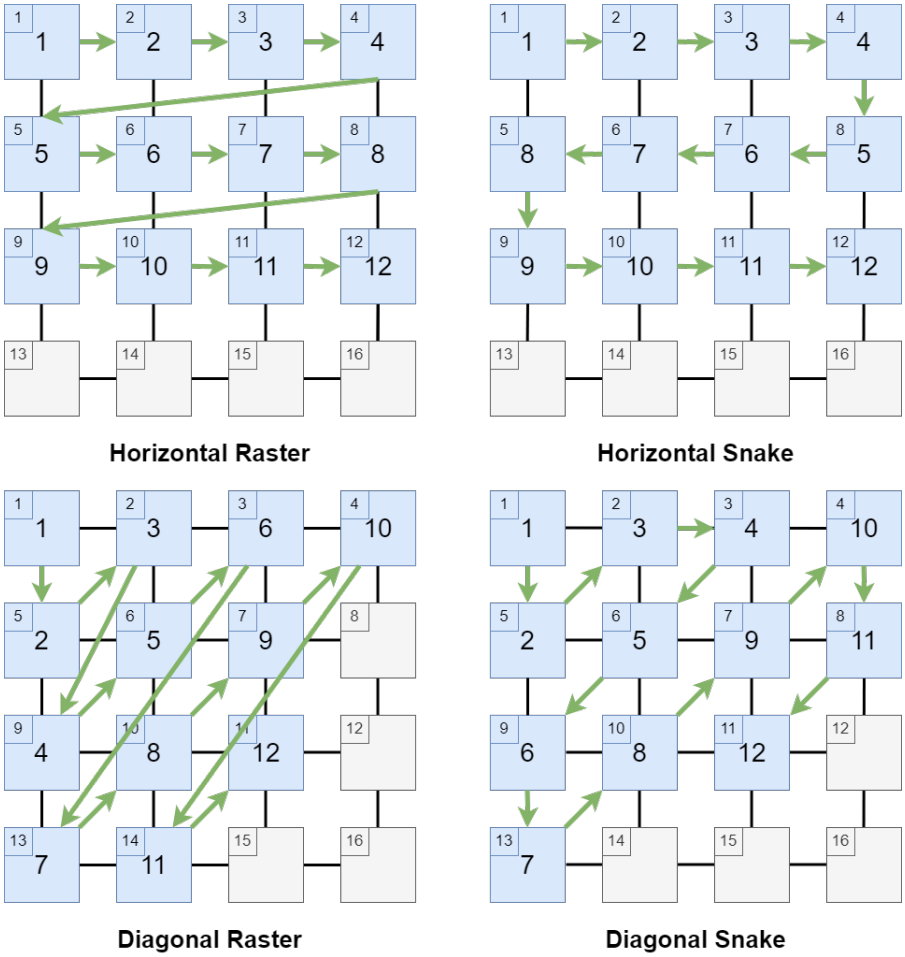


Figura 7 – Os quatro tipos de Enginnered Mappings implementados

Em otimizações multiobjetivo, o algoritmo de otimização é em função de métricas conjuntas, por exemplo, otimização de energia e balanceamento de carga, ou energia e tolerância a falhas, ou até mesmo as três simultaneamente. Para tal, é necessário que as funções de custo sejam implementadas como métricas opostas, conforme descrito em (DEB et al., 2002). Isto quer dizer que na função de balanceamento de carga, obtido pela Equação 2, obtêm menores valores mapeamentos em que há mais pontos de concentração de tarefas. Ou seja, mapas onde há nós processadores que concentram mais tarefas possuem um valor associado maior do que mapas onde as tarefas estão esparsas. Para cada cromossomo C, calcula-se o desvio padrão da quantidade de tarefas alocadas em cada processador de 1 a p.

Para o mapeamento exemplificado na Figura 8.b, tem-se para cada o custo de balanceamento de carga: $1 - \sigma(1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 2, 0, 1, 1, 0) = 0.38$. Se no nó 10 estivessem concentradas 3 tarefas, por exemplo, a função custo de balanceamento de carga seria: $1 - \sigma(1, 0, 0, 1, 1, 0, 0, 0, 0, 3, 0, 2, 0, 1, 1, 0) = 0.115$.

$$load\_balance(C) = 1 - \sigma\left[\sum_{i=1}^{p} num\_tasks(i))\right] \tag{2}$$

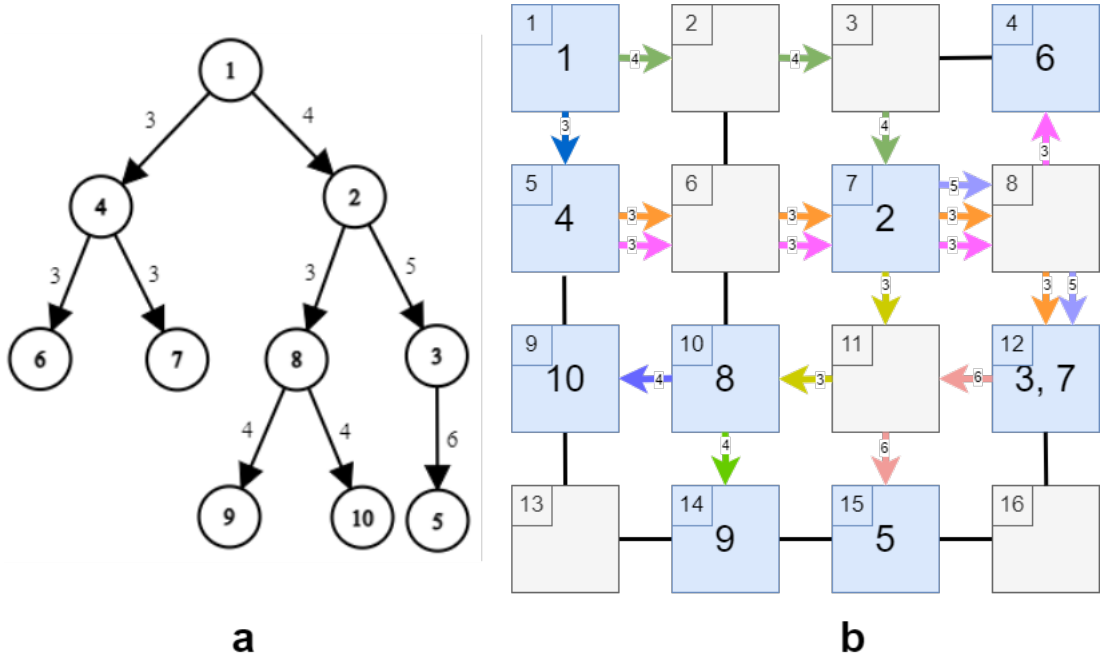


Figura 8 – Exemplo de cálculo do custo de comunicação

A tolerância a falhas mensura a capacidade do mapeamento em conseguir realocar uma ou mais tarefas em processadores vizinhos, em caso de falha do nó atual da respectiva tarefa, com o mínimo de degradação. Na Equação 4, tem-se para cada cromossomo C, para cada processador com tarefa alocada p_t a somatória de distâncias entre os demais processadores ociosos p_o no mapeamento corrente, em que i é a tarefa e j um processador ocioso do arranjo.

$$\forall i \in P_{\text{ocupados}},\ d_{\text{local}}(i) = \max_{j \in P_{\text{ociosos}}} d_{\text{Manhattan}}(i,j) \tag{3}$$

$$fault\_tolerance(C) = \min_{i \in P_{\text{ocupados}}} d_{\text{local}}(i) \tag{4}$$

Tomando como exemplo o processador 1 do mapeamento da Figura 8.b, tem-se que as distâncias Manhattan são: $\max[(1,2) = 1, (1,3) = 2, (1,6) = 2, (1,8) = 4, (1,11) = 4, (1,13) = 3, (1,16) = 6] = 4$. Logo, o custo de tolerância a falhas apenas para o processador 1 é igual a 4.

3.3.5 Operadores de Mutação e Crossover

A mutação de um dado indivíduo é realizada de maneira direta, selecionando-se um intervalo aleatório entre 1 e n, sendo n o comprimento do referido cromossomo, em que realiza-se uma permutação dos valores contidos nesse intervalo de acordo com uma taxa de mutação determinada pelo usuário, conforme Algoritmo 1.

Algoritmo 1 Procedimento Mutação

Require: $population, n, mutation\_rate$
Ensure: $init, end, pop\_size, chosen, chosen\_indexes$
1: $init \leftarrow \varnothing$
2: $end \leftarrow \varnothing$
3: $pop\_size = \text{SIZE}(population)$
4: $chosen\_indexes \leftarrow \text{RANDOMSELECT}(population, mutation\_rate * pop\_size)$
5: **for** $idx \in chosen\_indexes$ **do**
6: $\quad chosen = population[idx]$
7: $\quad a = \text{RANDOM}(1, n)$
8: $\quad b = \text{RANDOM}(1, n)$
9: $\quad init = \text{MIN}(a, b)$
10: $\quad end = \text{MAX}(a, b)$
11: $\quad population[idx] = \text{RANDOMPERMUT}(chosen, init, end)$
12: **end for**
13: **return** population

O crossover ocorre pela composição das partes dos indivíduos pais. Para cada par de cromossomos, obtém-se um ponto de cruzamento k entre 1 e n, sendo n o tamanho do referido cromossomo que será utilizado nesse processo. A partir disso, um novo indivíduo é gerado a partir da sequência de 1 a $k-1$ do pai 1 e k até n do pai 2, conforme Algoritmo 2.

Algoritmo 2 Procedimento Crossover

Require: $parent\_1, parent\_2, n$
Ensure: $k, offspring\_1, offspring\_2$
1: $offspring\_1 \leftarrow \varnothing$
2: $offspring\_2 \leftarrow \varnothing$
3: $k \leftarrow \text{RANDOM}(1, n)$
4: $offspring\_1 = parent\_1[1 : k-1] + parent\_2[k : n]$
5: $offspring\_2 = parent\_2[1 : k-1] + parent\_1[k : n]$
6: **return** offspring_1, offspring_2

3.4 Módulo de visualização dos resultados

A interface gráfica permite a execução por meio de dois modos distintos: *single* ou *batch*, e os resultados são exibidos diferentemente em cada caso.

3.4.1 Execução *single*

Na execução *single*, o processo evolutivo é realizado apenas uma vez e os resultados obtidos são exibidos em forma de gráfico, onde é possível visualizar a curva de Pareto encontrada para uma dada solução (Fig. 9).

No painel à esquerda, o usuário pode inserir a(s) APG(s) pelo botão *Load from file*, que permite o carregamento de um arquivo de extensão *.app* ou $.tgff$ e é convertido para o formato descrito na Seção 3.3.2. Além disso, é possível visualizar graficamente a aplicação no botão *V*. Também é possível configurar o tamanho do *Grid* do NoC em termos de linhas e colunas, o tamanho da população, a taxa de mutação, o número de

objetivos (somente energia, energia + tolerância a falhas, energia + balanço de carga) e se a execução será single ou multiaplicação.

Após a execução do algoritmo de otimização, o painel à direita disponibiliza o gráfico as coordenadas em termos dos objetivos otimizados e destaca em vermelho o ponto *best* da curva de Pareto, referindo-se ao indivíduo solução com a distância euclidiana mais curta em relação à origem do plano cartesiano (0,0). Ou seja, é a solução com o menor custo de energia e balanceamento de carga simultaneamente, por exemplo. Os pontos do gráfico são iterativos e o usuário pode selecionar a solução que deseja visualizar no mapeamento de tarefas pelo NoC.

Ao selecionar uma combinação na curva de Pareto, o usuário pode visualizar o mapeamento obtido na rede NoC. O visualizador indica, na rede de processadores, onde as tarefas do(s) APG(s) de input estão alocadas, tanto para single quanto multiaplicações (Fig. 10).

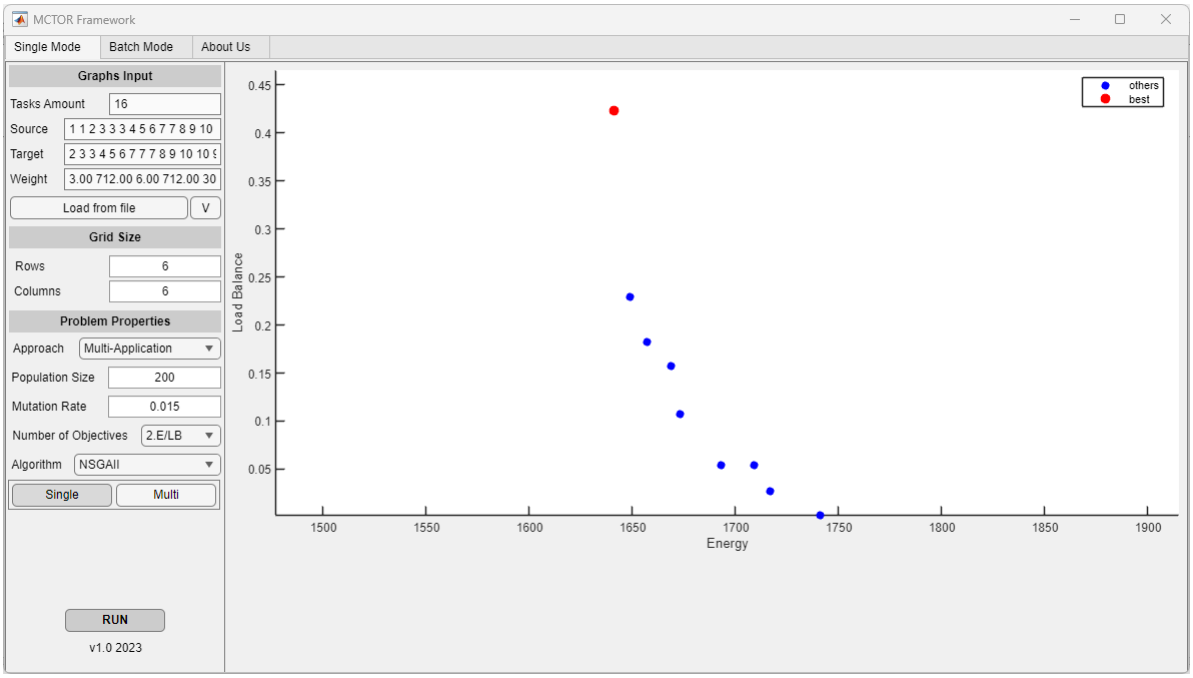


Figura 9 – Exemplo de execução no modo *single*, com a exibição da curva de Pareto e destaque em vermelho para o melhor resultado.

3.4.2 Execução *batch*

No modo de execução *batch*, é possível inserir como entrada mais de uma aplicação para execução. Também, é possível configurar múltiplos parâmetros, como diferentes tamanhos de *grid*, tamanhos de população, taxas de mutação e classes de algoritmos simultaneamente. A ferramenta se encarrega de executar todas as aplicações para todas as combinações dos múltiplos parâmetros dados pelo usuário (Fig. 11). Na execução *batch*, o intuito é extrair estatísticas das execuções, como média e desvio padrão de cada objetivo

Figura 10 – Visualização de um mapeamento solução para multiaplicações. Nesse exemplo, o mapa foi gerado para duas APGs: a primeira destacada na cor azul escuro e a segunda na cor azul claro.

otimizado. Sendo assim, é possível realizar repetições de execuções para cada combinação de parâmetros. O campo *Number of executions* define quantas vezes o procedimento evolutivo será executado para cada combinação de parâmetros. Para cada término da execução, o indivíduo *best* é selecionado através da menor distância euclidiana entre os objetivos otimizados e o ponto de origem $(0, 0)$ da curva de Pareto. Por exemplo, o usuário carregou as aplicações de benchmark *mms* e *mpeg*4, configurou o *Grid Size* para os tamanhos 4×4 e 6×6 e as classes de Algoritmos para $NSGAII$ e $PESAII$ e o número de execuções igual à 50, logo são $2 * 2 * 2 = 8$ combinações executadas 50 vezes cada. Ao final de cada conjunto de parâmetros, tem-se os 50 melhores indivíduos. Desses 50 melhores, extraem-se a média e o desvio padrão para cada objetivo, por exemplo, Energia e Balanceamento de Carga (Alg. 3).

Ao final da execução de todas as combinações, o *Framework* armazena os resultados em uma pasta dentro do projeto, chamada "batch", com o dia e horário da execução. Dentro dessa pasta, encontram-se arquivos com as estatísticas obtidas e também com os melhores indivíduos obtidos para cada combinação utilizada em formato CSV. Também, é possível, no painel de exibição de resultados, selecionar uma combinação e visualizar o melhor indivíduo ao clicar no botão "Visualize Solution".

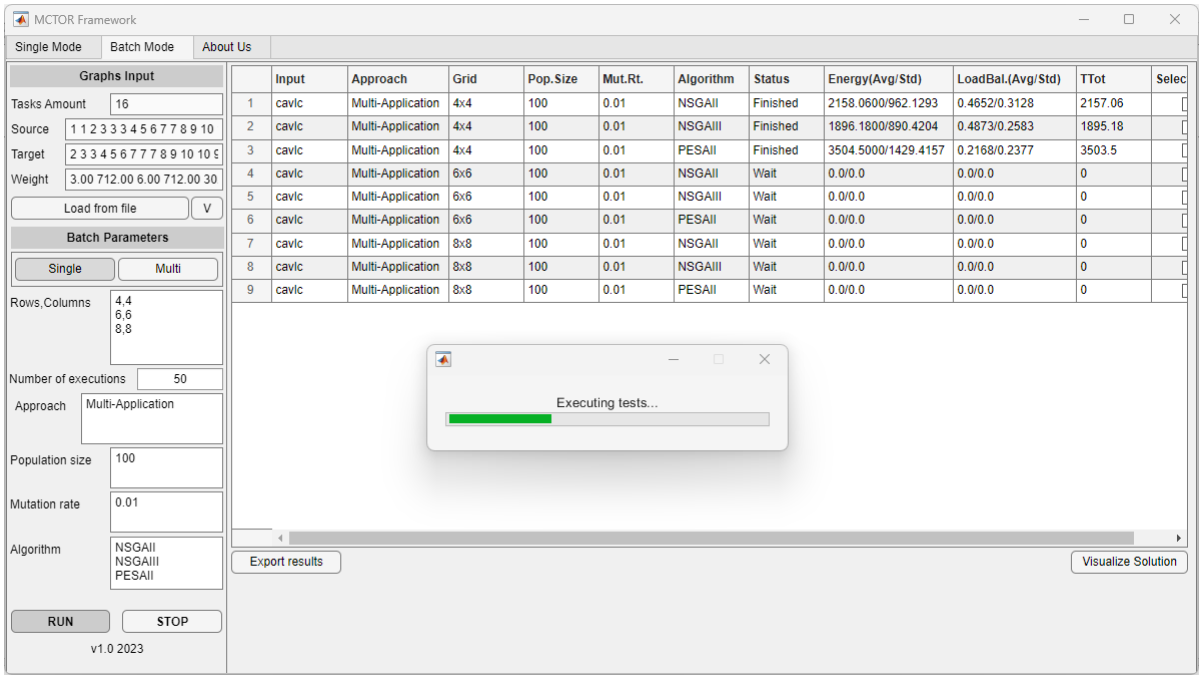


Figura 11 – Exemplo de execução no modo *batch*, a tela à direita mostra as combinações de parâmetros e dados estatísticos para cada configuração em execução.

Algoritmo 3 Procedimento Geração de Estatísticas em modo *batch*

Require: $app\_list, grid\_list, alg\_list, mut\_rate\_list, n\_exec$
Ensure: $Statistics, BestChromosomes$
$CombParams \leftarrow \text{CartesianProduct}(app\_list, grid\_list, alg\_list, mut\_rate\_list)$
$Statistics \leftarrow \varnothing$
$BestChromosomes \leftarrow \varnothing$
for $p \in CombParams$ **do**
  $i \leftarrow 0$
  $b \leftarrow \varnothing$
  while $i < n\_exec$ **do**
    $r \leftarrow \text{PlatEMO}(p)$
    $b(i) \leftarrow \text{GetBestChromosome}(r)$
    $i \leftarrow i + 1$
    $r \leftarrow \varnothing$
  end while
  $BestChromosomes(p) \leftarrow b$
  $Statistics(p) \leftarrow [\text{Mean}(b), \text{StdDev}(b), \text{Min}(b), \text{Max}(b), \text{Q1}(b), \text{Q3}(b)]$
end for
return Statistics, BestChromosomes

3.5 Simulador NoC

A metodologia adotada para o desenvolvimento do simulador de NoC baseia-se em um programa implementado em Python (Fig. 12). O simulador fornece uma interface gráfica de usuário que permite configurar os principais parâmetros experimentais, incluindo o tamanho da topologia Mesh, o(s) algoritmo(s) de roteamento, as taxas de injeção de pacotes, a duração da simulação, o arquivo contendo o grafo da aplicação, um arquivo contendo o mapeamento das tarefas para a NoC, entre outros.

Uma vez definidos os parâmetros, uma topologia de rede Mesh 2D convencional é instanciada, com a criação e interconexão de roteadores e PE. A simulação segue um

modelo de tempo discreto, no qual cada passo de tempo corresponde a uma iteração da simulação. Durante a execução, pacotes são injetados de acordo com as taxas e modos de injeção configurados, permitindo a análise do comportamento dinâmico da rede sob diferentes cargas de tráfego e estratégias de roteamento.

Ao final da simulação, a ferramenta coleta e processa métricas de desempenho relevantes, que são apresentadas diretamente na interface gráfica. Entre as estatísticas reportadas estão o *throughput* da rede e a latência média de pacotes, exibidos por meio de gráficos.

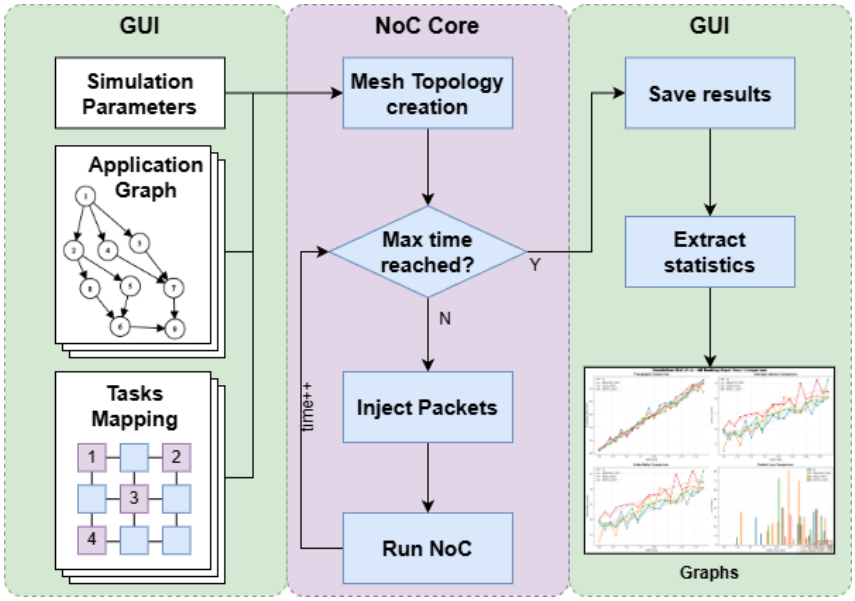


Figura 12 – Visão Geral do módulo Simulador NoC

3.5.1 Criação da topologia Mesh

Embora diversas topologias alternativas de NoC tenham sido propostas na literatura — como Mesh, Torus, Tree, Butterfly, Polygon, Star, Ring, entre outras (RAMAKRISHNAN; SANJU; FARHANAAZ, 2023) — a modelagem arquitetural da NoC no simulador proposto adota a tradicional topologia Mesh bidimensional, também conhecida na literatura como topologia CLICHÉ (FARMAN; DURRANI, 2023), composta por r linhas e c colunas, formando uma matriz regular de roteadores $R_{i,j}$ (Fig. 13).

Cada roteador é diretamente conectado a um PE, que representa um nó computacional responsável pela execução das tarefas da aplicação. Os roteadores são interconectados por meio de barramentos bidirecionais com seus roteadores vizinhos. Dependendo de sua posição na topologia 2D, os roteadores podem se conectar a até quatro roteadores adjacentes nas direções norte, sul, leste e oeste.

A rede virtual é construída dentro do simulador por meio do procedimento descrito no Algoritmo 4, que gera sistematicamente a matriz de roteadores, associa cada roteador ao

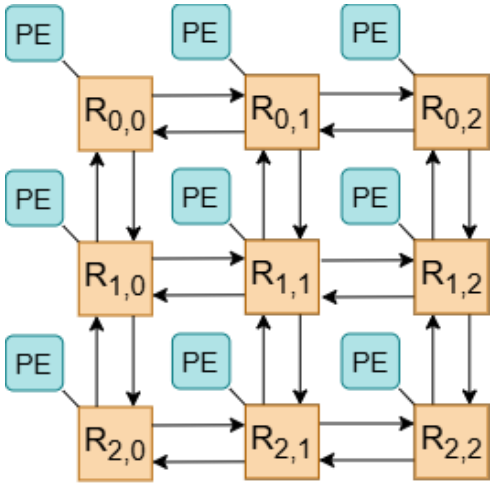


Figura 13 – Exemplo de uma topologia Mesh 2D com 3 linhas e 3 colunas e as ligações entre os roteadores e os PEs

seu respectivo PE, estabelece os enlaces entre roteadores de acordo com a topologia Mesh 2D e configura as interfaces PE–roteador que permitem a injeção e a entrega de pacotes.

Algoritmo 4 Procedimento de criação da topologia Mesh 2D

Require: $rows, columns$
1: $routers \leftarrow \varnothing$
2: $pes \leftarrow \varnothing$
3: **for** $i \in rows$ **do**
4: **for** $j \in columns$ **do**
5: $routers[i][j] = newRouter(i, j)$
6: $pes[i][j] = newProcessingElement(i, j)$
7: $pes[i][j].out\_buffer \leftarrow routers[i][j].in\_buffer[\text{"local"}]$
8: **end for**
9: **end for**
10: **for** $r \in routers$ **do**
11: $neightbors \leftarrow \varnothing$
12: **if** $\exists routers[r.x][r.y-1]$ **then**
13: $neighbors[WEST] = routers[r.x][r.y-1].in\_buffer[EAST]$
14: **end if**
15: **if** $\exists routers[r.x][r.y+1]$ **then**
16: $neighbors[EAST] = routers[r.x][r.y+1].in\_buffer[WEST]$
17: **end if**
18: **if** $\exists routers[r.x+1][y]$ **then**
19: $neighbors[SOUTH] = routers[r.x+1][r.y].in\_buffer[NORTH]$
20: **end if**
21: **if** $\exists routers[r.x-1][r.y]$ **then**
22: $neighbors[NORTH] = routers[r.x-1][r.y].in\_buffer[SOUTH]$
23: **end if**
24: $r.out\_buffer \leftarrow neighbors$
25: **end for**
26: **return** routers, pes

Cada componente de roteador é modelado com um buffer de entrada dedicado para cada direção de entrada, com capacidade configurável. Além disso, uma interface local conecta o roteador ao seu respectivo PE. Do ponto de vista da rede, a injeção de pacotes ocorre quando o PE insere um pacote no buffer de entrada local do roteador, tornando-o elegível para roteamento nas iterações subsequentes da simulação. Os enlaces entre roteadores são modelados como referências aos buffers de entrada dos roteadores vizinhos.

Dessa forma, a transmissão de um pacote por um enlace corresponde à sua inserção no buffer de entrada apropriado do roteador adjacente.

Do ponto de vista do sistema, os dados entram na NoC quando um PE gera um pacote e o injeta na rede ao colocá-lo no *buffer* de entrada local de seu roteador associado. De forma análoga, os dados deixam a NoC quando um pacote alcança seu roteador de destino e é transferido para a fila de recepção do PE correspondente para processamento local (Fig. 14). Dessa forma, os PEs atuam tanto como fontes quanto como destinos de tráfego, definindo o limite lógico do sistema NoC.

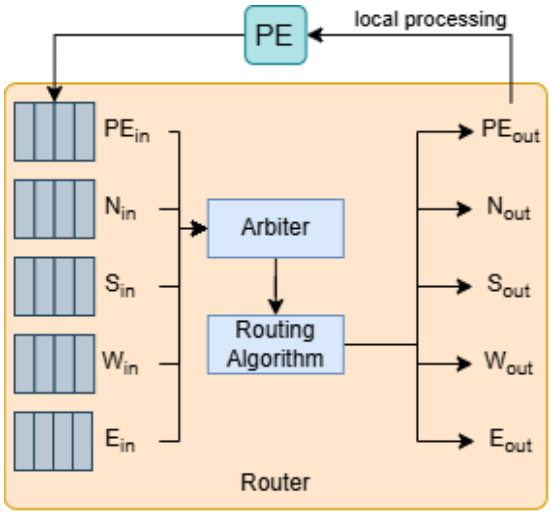


Figura 14 – Visão interna do componente de Roteador, destacando as conexões com roteadores vizinhos (em uma topologia Mesh 2D) e os buffers internos de entrada de pacotes, bem como a conexão com o PE, o árbitro e o algoritmo de roteamento

A cada iteração da simulação, um mecanismo de arbitragem seleciona um dos buffers de entrada não vazios para atendimento. O árbitro decide qual buffer receberá acesso na iteração corrente, determinando de qual buffer o pacote na cabeça da fila será processado e encaminhado de acordo com o algoritmo de roteamento ativo.

Uma vez que um buffer de entrada é selecionado, o pacote é analisado. Se o roteador atual corresponder ao nó de destino do pacote, o pacote é removido da rede e colocado na fila de recepção do PE associado para processamento local. Caso contrário, o algoritmo de roteamento determina a direção de saída apropriada, e o pacote é encaminhado ao ser inserido no buffer de entrada correspondente do próximo roteador ao longo do caminho selecionado.

3.5.2 Algoritmos de roteamento

Este trabalho avalia estratégias de roteamento determinísticas e não determinísticas para arquiteturas de NoC. Os algoritmos de roteamento influenciam diretamente a la-

tência, a distribuição de congestionamento e o desempenho geral da rede (R; H.; V., 2019).

A abordagem determinística considerada é (MYUNG; QI; CHENG, 2019):

- **XY** – Um algoritmo de roteamento por ordem de dimensões, no qual os pacotes são roteados primeiro ao longo da dimensão X e, em seguida, ao longo da dimensão Y.

Os algoritmos não determinísticos (parcialmente adaptativos) avaliados são:

- **Negative-First (NF)**: Restringe determinados desvios após o percurso em direções negativas, de modo a garantir a ausência de *deadlock*. Todos os movimentos nas direções negativas ($X-$ e $Y-$) devem ser realizados antes de qualquer movimento nas direções positivas ($X+$ e $Y+$).

- **West-First (WF)**: Exige que os pacotes completem os movimentos na direção oeste antes de utilizarem outras direções.

- **North-Last (NL)**: Adia movimentos na direção norte até que as demais direções tenham sido exploradas minimamente.

3.5.3 Execução da simulação

Para simplificar o modelo de simulação, o simulador considera pacotes compostos exclusivamente por um cabeçalho, sem *tail flits* ou *payload flits*, ou seja, 1 pacote = 1 *flit*.

A simulação é conduzida utilizando um modelo de tempo discreto baseado em iterações de simulação globalmente sincronizadas. A cada iteração, *flits* podem ser injetados na rede por meio dos PEs, que colocam os *flits* gerados no buffer de entrada local de seus roteadores associados.

A ferramenta proposta suporta dois modos distintos de injeção de pacotes. Em ambos os modos, o processo de injeção é limitado por uma taxa de injeção configurável, que restringe o número de *flits* que podem ser gerados por PE e por iteração de simulação.

No primeiro modo (Algoritmo 5), denominado *injeção aleatória*, cada PE injeta um pacote em direção a um PE de destino selecionado aleatoriamente de acordo com uma taxa de injeção predefinida. Nesse esquema, todos os PEs avaliam independentemente a condição de injeção a cada iteração da simulação e, quando satisfeita, geram um pacote destinado a um nó escolhido aleatoriamente na rede.

No segundo modo (Algoritmo 6), denominado *injeção orientada pelo mapeamento de tarefas*, a geração de pacotes é guiada pelos canais de comunicação definidos no grafo da aplicação. Quando um mapeamento de tarefas é fornecido, os pacotes são injetados de acordo com os canais de comunicação entre as tarefas e seus custos de comunicação associados, conforme instanciados pelo mapeamento na arquitetura NoC (Seção 3.3.3).

Algoritmo 5 Injeção aleatória de pacotes

Require: *max_simulation_iterations*, *injection_rate*, *pes*
for $i \leftarrow 1$ to *max_simulation_iterations* **do**
  if RANDOM$(0,1) <$ *injection_rate* **then**
    for all $p_s \in pes$ **do**
      $p_d \leftarrow$ RANDOM(pes) ▷ Select a random destination
      if $p_d \neq p_s$ **then**
        $packet \leftarrow$ PACKET(p_s, p_d)
        ENQUEUE$(p_s.input_buffer, packet)$
      else
        $p_d \leftarrow$ RANDOM(pes) ▷ Another destination should be selected
      end if
    end for
  end if
end for
return

Nesse caso, a geração de tráfego reflete a estrutura da aplicação mapeada, em vez de um comportamento aleatório uniforme.

Algoritmo 6 Injeção de pacotes orientada pelo mapeamento de tarefas

Require: *max_simulation_iterations*, *pes*, *mapping*, *app_graph*
for $i \leftarrow 1$ to *max_simulation_iterations* **do**
  for all $g \in app\_graph$ **do**
    $p_s \leftarrow pes(mapping(g.source\_id))$
    $p_d \leftarrow pes(mapping(g.destination\_id))$
    $packet \leftarrow$ PACKET(p_s, p_d)
    ENQUEUE$(p_s.input_buffer, packet)$
  end for
end for
return

Após a fase de injeção de pacotes, todos os roteadores executam exatamente uma iteração de forma globalmente sincronizada, durante a qual são realizadas a arbitragem dos buffers de entrada e as decisões de roteamento. Cada roteador seleciona independentemente um de seus buffers de entrada por meio do mecanismo de arbitragem e determina a direção de saída apropriada de acordo com o algoritmo de roteamento selecionado.

Uma vez definida a direção de saída, o roteador tenta encaminhar o *flit* para o roteador vizinho correspondente. Essa transferência somente é concluída se o roteador de destino possuir espaço disponível em seu buffer de entrada correspondente. Caso contrário, quando o buffer de destino está ocupado, o *flit* permanece em seu buffer de entrada original e a transmissão é adiada para uma iteração futura da simulação. Esse mecanismo modela a contenção na rede e resulta no aumento do atraso na entrega de pacotes à medida que a carga de tráfego cresce e os buffers se tornam saturados. Após todos os roteadores concluírem suas tentativas de encaminhamento, a simulação avança de forma sincronizada para a próxima iteração, e a fase de injeção de pacotes é repetida até que o número total de iterações da simulação atinja um valor máximo configurável, conforme ilustrado na Figura 12.

3.5.4 Métricas e estatísticas de rede

Ao longo da simulação, métricas relacionadas ao desempenho são continuamente coletadas tanto dos PEs quanto dos roteadores. Essas métricas incluem o número de pacotes injetados, eventos de roteamento, o número de saltos percorridos por cada pacote e se um pacote é entregue com sucesso ao seu destino ou descartado durante a transmissão. Para cada pacote, o simulador também registra a iteração discreta da simulação em que ele é gerado e a iteração em que é entregue. É importante enfatizar que essas iterações não correspondem a ciclos de *clock* físicos de hardware. Em vez disso, elas representam passos de tempo abstratos, globalmente sincronizados, utilizados para avançar o estado do sistema no modelo de simulação em tempo discreto adotado.

Como o simulador proposto opera em um nível de abstração mais alto do que modelos ciclo-a-ciclo, certos detalhes de hardware de baixo nível — como estágios de pipeline, microarquitetura precisa dos roteadores e efeitos temporais em nível de ciclo — não são explicitamente modelados. Consequentemente, o simulador não tem como objetivo reproduzir com exatidão o comportamento temporal do hardware ou estimativas altamente detalhadas de consumo de energia. Entretanto, essa abstração permite execuções de simulação mais rápidas e uma configuração mais simples, tornando o framework adequado para exploração rápida do espaço de projeto, avaliação de algoritmos de roteamento e estudos de mapeamento de tarefas.

Neste trabalho, o ***throughput*** da rede é definido como a taxa efetiva de comunicação da NoC e é medido em *flits* por iteração de simulação. Ele é calculado como o número total de *flits* entregues com sucesso aos nós de destino dividido pelo número total de iterações de simulação necessárias para uma determinada execução, sendo representado pela Equação (5).

$$Throughput = \frac{F_{\text{delivered}}}{T_{\text{sim}}} \tag{5}$$

A **latência média** é definida como o tempo médio de simulação necessário para que um pacote atravesse a NoC desde sua injeção no PE de origem até sua entrega ao PE de destino. Para cada pacote injetado, o simulador registra o tempo discreto em que o pacote é criado e o tempo discreto em que ele é entregue com sucesso.

Seja $N_{\text{delivered}}$ o número de *flits* entregues, e sejam $T^{(i)}_{\text{inject}}$ e $T^{(i)}_{\text{deliver}}$ os instantes de criação e de entrega do *flit* i, respectivamente. A latência média da rede L_{avg} é dada por (6).

$$L_{\text{avg}} = \frac{1}{N_{\text{delivered}}} \sum_{i=1}^{N_{\text{delivered}}} \left(T^{(i)}_{\text{deliver}} - T^{(i)}_{\text{inject}} \right) \tag{6}$$

Um pacote é considerado perdido (***package loss***) quando o tempo máximo de simulação, definido em termos do número total de passos de tempo discretos, é atingido e o pacote ainda não foi entregue ao seu nó de destino. Nessa situação, assume-se que o pacote permaneceu retido na rede devido a condições de congestionamento, contenção de

buffers ou atrasos acumulados ao longo do roteamento, não sendo possível completar sua transmissão dentro do limite temporal estabelecido para a execução da simulação.

Capítulo 4

Resultados

Esta seção apresenta os resultados experimentais obtidos com as ferramentas desenvolvidas neste trabalho. Inicialmente, na Seção 4.1, são apresentados os experimentos realizados no módulo de otimização multiobjetivo, implementado em MATLAB, com foco na geração e análise de diferentes estratégias de mapeamento de tarefas em arquiteturas NoC. Em seguida, na Seção 4.2, são descritos os experimentos conduzidos utilizando o simulador de NoC implementado em Python, no qual são avaliadas métricas de desempenho da rede sob diferentes configurações de tráfego, algoritmos de roteamento e cenários de mapeamento de aplicações.

4.1 Experimentos no módulo Multi-objetivo

Para verificar e validar a nova abordagem de mapeamento Multiaplicação, é utilizado um teste similar ao proposto em (GE et al., 2021) como referência (Multifase). São utilizados gráficos de *benchmark* com 12, 16, 20 e 25 tarefas ($mpeg4$, $vopd$, vce, $wifirx$) executadas simultaneamente para uma plataforma NoC de 9×9 processadores, onde esse processo é repetido 100 vezes.

A proposta é avaliada considerando o conjunto de todas as aplicações para métricas de latência (energia) e tolerância a falhas (FT). Na Figura 15, é demonstrado que a proposta Multiaplicação tem uma latência total menor do que a proposta Multifase. Na Figura 16, a tolerância a falhas total é muito mais baixa do que na Multifase.

Nas Figuras 18 (melhor latência - proposta atual), 19 (melhor tolerância a falhas - proposta atual), 20 (balanceado - proposta atual) e 21 (mapa comparado - Multifase) e na Tabela 2, é possível visualizar os resultados dos mapas de aplicação gerados em cada

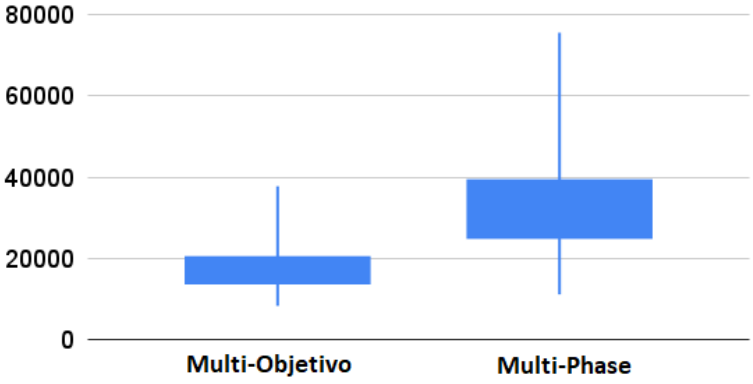


Figura 15 – Comparativo de energia

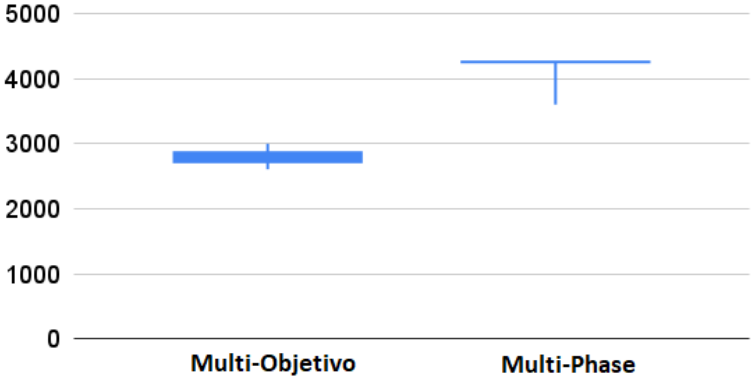


Figura 16 – Comparativo de tolerância a falhas

caso, para cada abordagem, de acordo com a curva de Pareto apresentada na Figura 17, gerada pela proposta atual.

Tabela 2 – Comparativo de energia e tolerância a falhas entre a abordagem proposta Multiaplicação e a referência Multifase (GE et al., 2021)

	Min	Q1	Q3	Max
Energia Multiaplicação	8400	14000	20290	37800
Energia Multifase	11200	25200	39200	75600
FT Multiaplicação	2611	2724	2866	3002
FT Multifase	3604	4260	4260	4260

Os experimentos da Tabela 3 fornecem uma ilustração do aprimoramento quantitativo da qualidade dos mapeamentos que podem ser obtidos ao aplicar a cadeia de ferramentas evolucionária. A análise é realizada na métrica de Energia, que pode ser considerada a "pior"métrica neste cenário, pois a natureza acíclica dos gráficos de tarefas implica uma forte compatibilidade com algoritmos determinísticos de alocação baseados em fluxo.

As colunas (HR, HS, DR, DS) exibem o *fitness* energético dos mapeamentos obtidos aplicando um conjunto de estratégias de *Engineered Mappings* (BONNEY et al., 2016). A

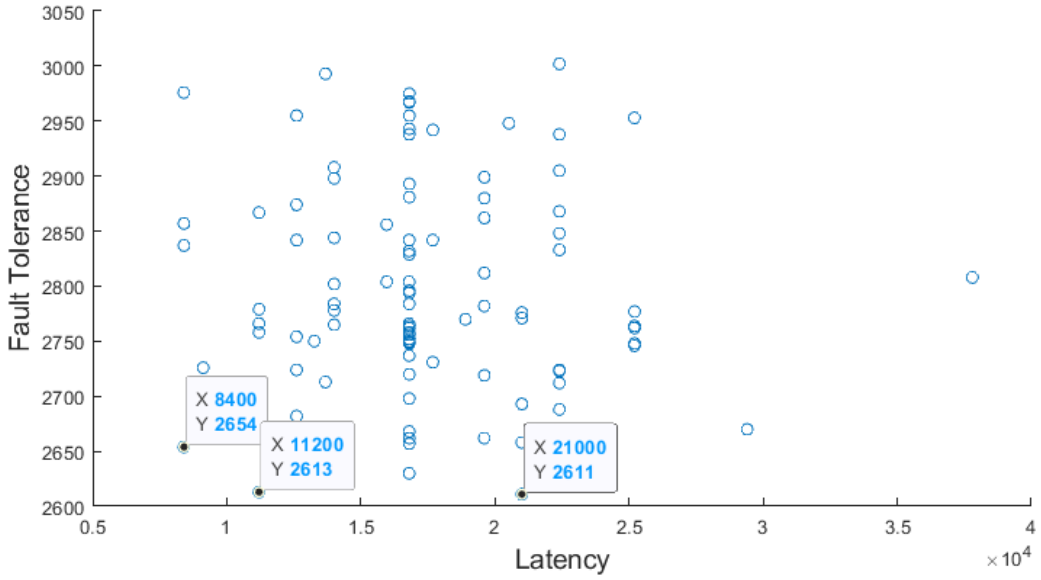


Figura 17 – Curva de Pareto para teste Multiaplicação - energia × tolerância a falhas

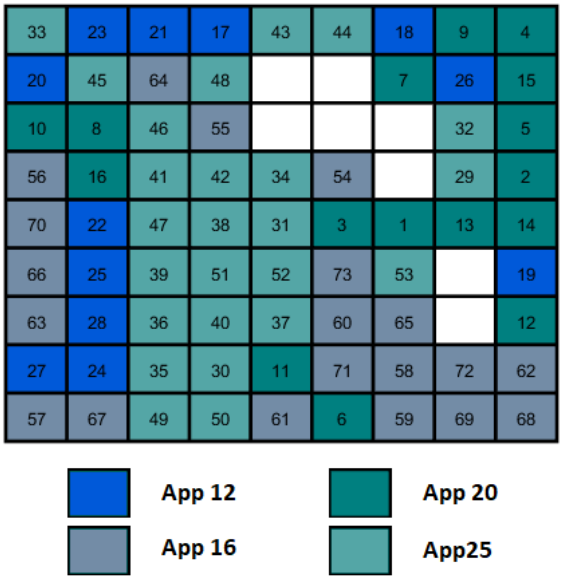


Figura 18 – Melhor energia - Multiaplicação

coluna `EV` mostra a aptidão do melhor mapeamento encontrado pela cadeia de ferramentas (algoritmo PESAII com uma população inicial de 200 indivíduos aleatórios e taxa de mutação de 0.01). A coluna `Gain` exibe, em porcentagem, o ganho (em termos da métrica de energia) obtido pela aplicação da cadeia de ferramentas, conforme Equação 7.

$$gain = 1 - \frac{EV}{BestOf(HR, HS, DR, DS)} \tag{7}$$

Os experimentos foram realizados em uma série de tamanhos de *grid* (4×4, 6×6 e

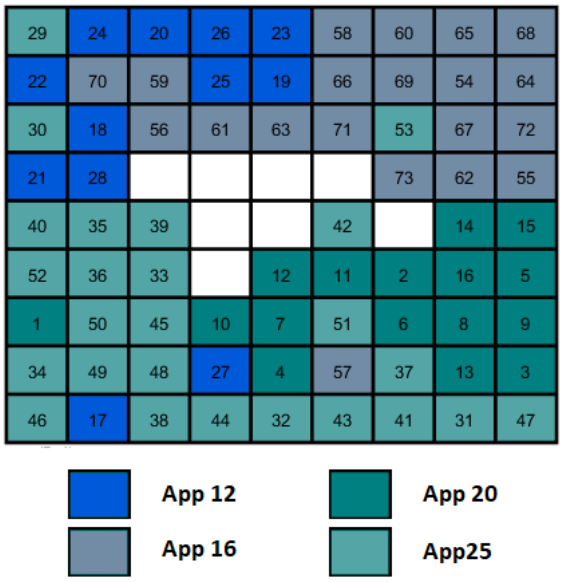


Figura 19 – Melhor tolerância a falhas - Multiaplicação

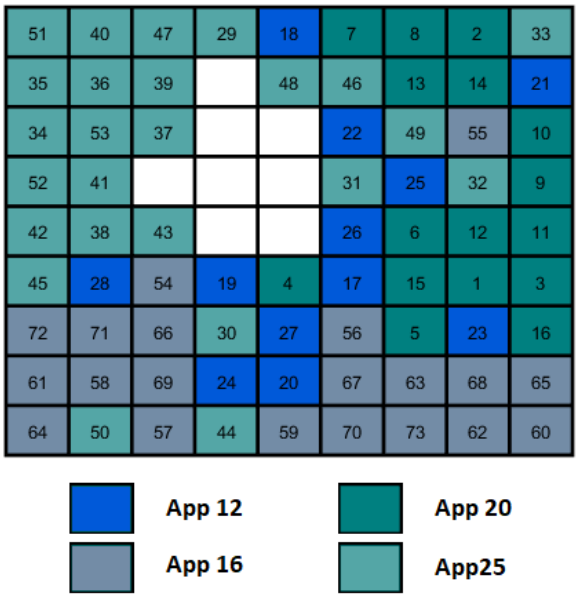


Figura 20 – Balanceado entre energia e tolerância a falhas - Multiaplicação

8×8) para 4 APGs reais, onde 3 são filtros morfológicos desenhados para detecção de padrões em imagens ($CIRCLE$, $SQUARE$ e $TRACK$) e outro é um filtro morfológico para detecção de tanques de combustível em plantas de produção de petróleo via imagens de satélite ($SATELLITE$). Para tamanhos de grade menores (4×4), onde o consumo de energia é mais baixo, mas mais dependente do posicionamento das tarefas, os ganhos obtidos aplicando a cadeia de ferramentas evolucionária variam de 65% a 74% (Fig. 22). Para os maiores tamanhos de grade, que são mais compatíveis com o fluxo de dados do gráfico acíclico considerando a abundância de nós computacionais, melhorias de 39% a

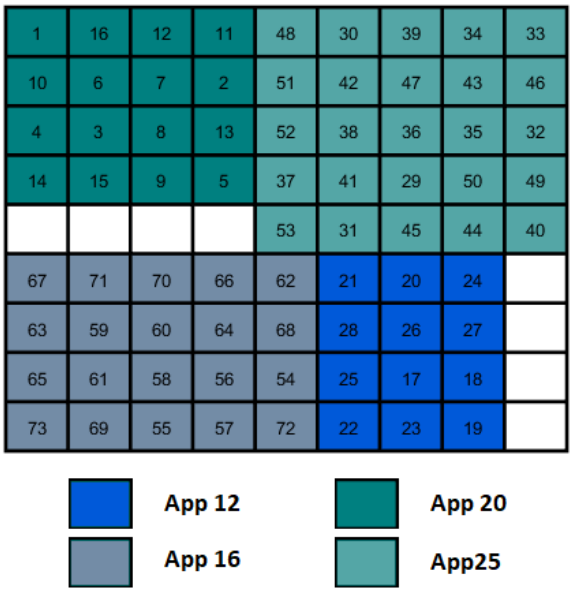


Figura 21 – Melhor indivíduo - Multifase

Tabela 3 – Comparação de ganhos em termos de energia entre abordagens determinísticas (*Engineered Mappings*) e abordagem evolutiva proposta. População = 200, taxa de mutação = 0.01

	grid	HR	HS	DR	DS	EV	Gain (%)
CIRCLE	4x4	52,62	52,76	52,48	52,26	18,2	65,17
	6x6	73,32	72,74	74,74	74,28	31,56	56,61
	8x8	78,82	78,62	77,98	77,82	46,3	40,50
SQUARE	4x4	52,98	52,28	51,64	52,5	13,54	73,78
	6x6	72,8	72,32	75,06	75,22	23,74	67,17
	8x8	78,76	78,88	77,16	78,54	39,54	48,76
TRACK	4x4	58,76	59,16	58,74	58,8	18,96	67,72
	6x6	81,76	80,88	84,2	83,26	35,52	56,08
	8x8	87,96	88,08	86,72	86,08	52,22	39,34
SATELLITE	4x4	52,92	52,5	52	52,34	14,44	72,23
	6x6	72,86	72,36	74,5	74,2	27,7	61,72
	8x8	77,96	78,86	77,46	77,18	39,34	49,03

49% podem ser alcançadas, ainda assim.

4.2 Experimentos no Simulador NoC

Nesta seção, os experimentos são projetados para comparar o comportamento do simulador proposto com resultados reportados para outros simuladores de NoC na literatura sob condições experimentais equivalentes. O objetivo não é reproduzir ou validar valores absolutos de desempenho da NoC (por exemplo, valores exatos de latência ou *throughput*), mas sim avaliar se o simulador é capaz de capturar tendências de desempenho consistentes

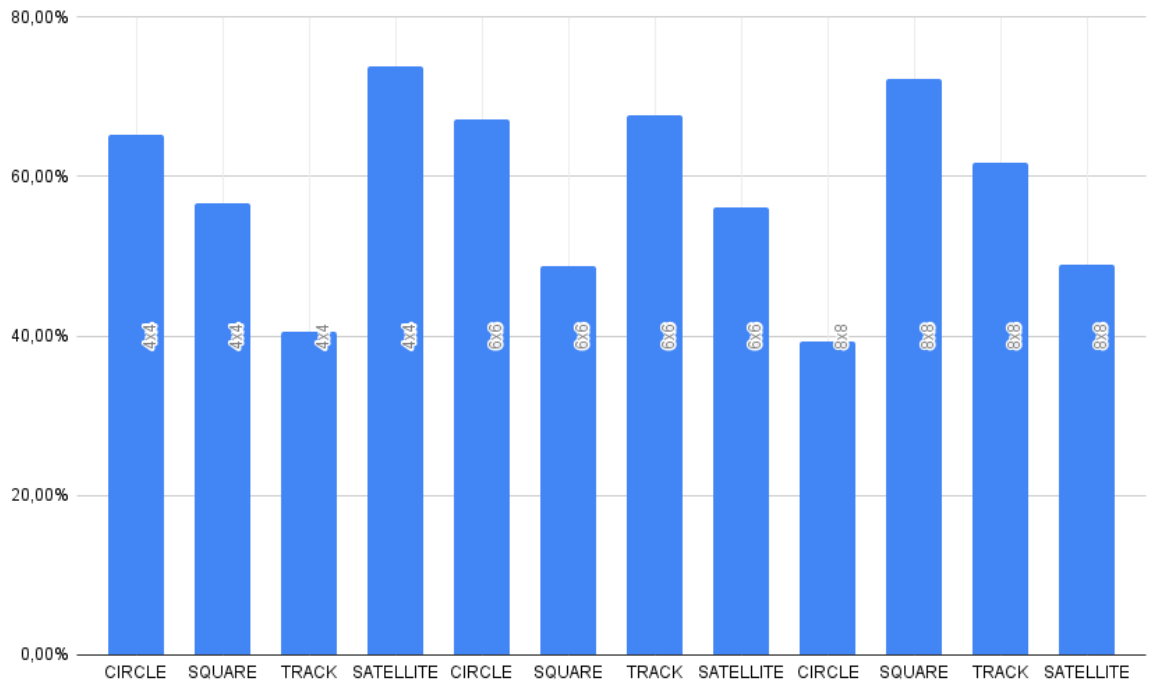


Figura 22 – Ganhos em termos de energia da abordagem evolutiva comparado aos mapeamentos determinísticos

quando comparado a ferramentas já estabelecidas.

Para validar a eficácia do simulador proposto, três grupos de experimentos foram conduzidos. O primeiro grupo (Seção 4.2.1) analisa o impacto da variação de parâmetros de simulação — como diferentes algoritmos de roteamento — em métricas de desempenho da rede, comparando as tendências observadas com aquelas reportadas em estudos anteriores. O segundo grupo (Seção 4.2.2) concentra-se na avaliação de mapeamentos de tarefas predefinidos derivados de aplicações de *benchmark*. Nesse caso, os resultados obtidos são comparados com aqueles reportados utilizando simuladores de NoC amplamente conhecidos, como Noxim, NoCTweak e *frameworks* relacionados. Por fim, o terceiro grupo de experimentos (Seção 4.2.3) tem como objetivo validar as métricas de rede para os mapeamentos obtidos na Seção 4.1, analisando a consistência dos resultados produzidos pelo simulador quando aplicado a soluções geradas por abordagens de otimização multiobjetivo.

Ao longo desses experimentos, parâmetros de arquitetura do NoC, padrões de tráfego e algoritmos de roteamento são sistematicamente variados com o objetivo de avaliar seu impacto no comportamento da rede dentro do modelo de simulação em tempo discreto baseado em iterações adotado neste trabalho.

4.2.1 Experimento I

No primeiro experimento, o objetivo principal foi reproduzir os parâmetros de simulação utilizados em (CHENG; XIANG; HU, 2025), de modo a obter um comportamento semelhante em termos de latência média e *throughput*. Especificamente, o simulador proposto foi configurado com a mesma topologia de rede e tamanho de malha, taxas de injeção de pacotes e padrão de tráfego descritos no trabalho de referência. A simulação foi executada considerando diferentes algoritmos de roteamento: XY, NEGATIVE_FIRST, WEST_FIRST e NORTH_LEAST. Os parâmetros avaliados e seus respectivos intervalos são apresentados na Tabela 4.

Tabela 4 – Parâmetros de simulação utilizados no Experimento 4.2.1

Parâmetro	**Valor**
Tamanho da Mesh	6×6
Tamanho do buffer	4
Estratégia do Árbitro	Round Robin
Algoritmo de Roteamento	XY Negative First West First North Least
Tempo de simulação	10000 passos
Modo de injeção de pacotes	Aleatório
Taxa de injeção de pacotes	0.01 to 0.2

As Figuras 23 e 24 ilustram, respectivamente, a comparação entre os diferentes algoritmos de roteamento para as métricas de latência média e taxa de transferência (*throughput*). À medida que a taxa de injeção de pacotes aumenta progressivamente, a latência média tende a crescer de forma exponencial, enquanto o *throughput* aumenta aproximadamente de forma linear. Em relação à latência média, o algoritmo de roteamento XY apresentou o melhor desempenho, enquanto NEGATIVE_FIRST e NORTH_LEAST resultaram nas maiores latências, comportamento consistente com os resultados apresentados em (CHENG; XIANG; HU, 2025).

4.2.2 Experimento II

No segundo experimento, reproduzimos múltiplas estratégias de mapeamento de tarefas para a aplicação de *benchmark* VOPD (*Video Object Plan Decoder*) (Fig. 25), seguindo a metodologia experimental descrita em (AMIN et al., 2020). Nesse estudo, uma versão modificada do simulador NoCTweak, denominada ENoCTweak, foi utilizada como plataforma de avaliação. No entanto, o conjunto completo de parâmetros de simulação adotado nesse ambiente não é explicitamente reportado. Dessa forma, para garantir uma configuração experimental consistente e reprodutível, configuramos nosso simulador de acordo com os parâmetros resumidos na Tabela 5.

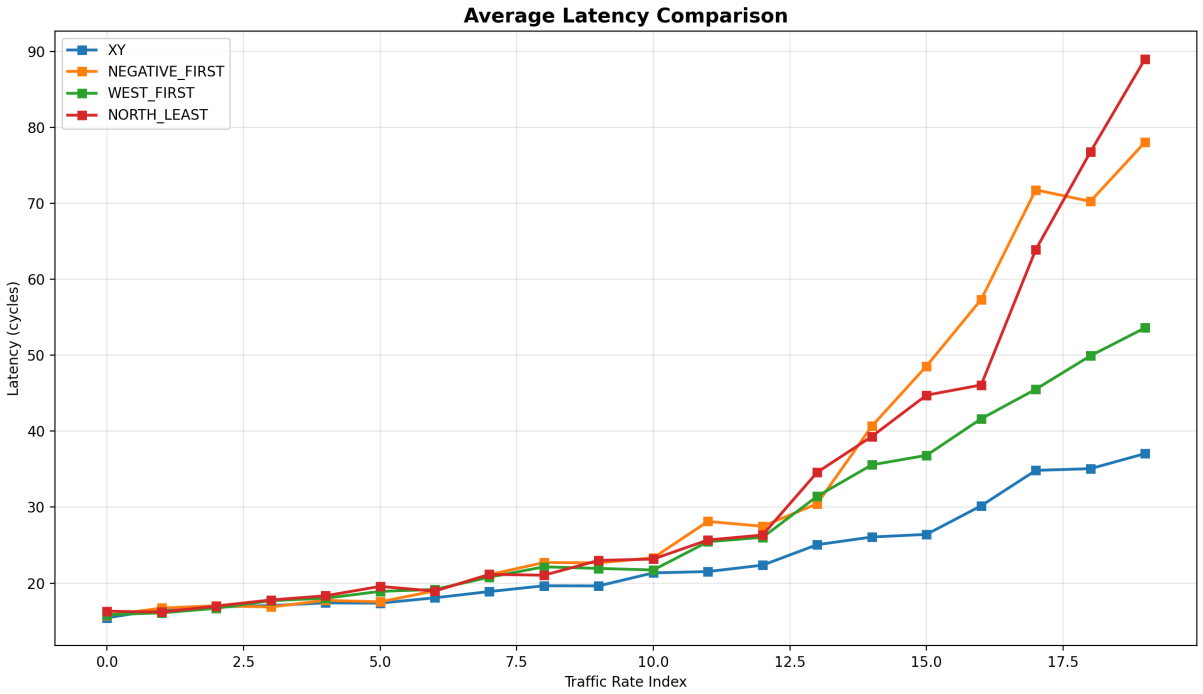


Figura 23 – Experimento 4.2.1. Comportamento da latência média para diferentes taxas de injeção de pacotes, em comparação com o cenário descrito em (CHENG; XIANG; HU, 2025)

Tabela 5 – Parâmetros de simulação utilizados no Experimento 4.2.2

Parâmetro	**Valor**
Tamanho da Mesh	4×4
Tamanho do buffer	4
Estratégia do Árbitro	Round Robin
Algoritmo de Roteamento	XY
Tempo de simulação	10000 passos
Modo de injeção de pacotes	Orientado a mapeamento
Taxa de injeção de pacotes	0.01 to 0.5

Os mapeamentos de tarefas apresentados na Figura 26 foram posteriormente implementados no simulador de NoC proposto, possibilitando uma comparação qualitativa direta com os resultados apresentados em (AMIN et al., 2020). Esses mapeamentos foram originalmente coletados de diversos trabalhos da literatura e organizados em categorias distintas com base em suas estratégias de otimização subjacentes. Especificamente, incluem: *heurísticas construtivas com melhoria iterativa* (NMAP (MURALI; MICHELI, 2004), LMAP (SAHU et al., 2010), Map_Graph (SAHU et al., 2015), XY-ADB (ALIKHAH-ASL; RESHADI, 2016) e SA (LU; XIA; JANTSCH, 2008)); *heurísticas construtivas sem refinamento iterativo* (RMAP (PATOOGHY; TABKHI; MIREMADI, 2010), CastNet (TOSUN, 2011) e MapGtoM (SHARMA; BISWAS; MITRA, 2019)); *técnicas exatas*, representadas pela Programação Linear Inteira (ILP) (OSTLER; CHATHA,

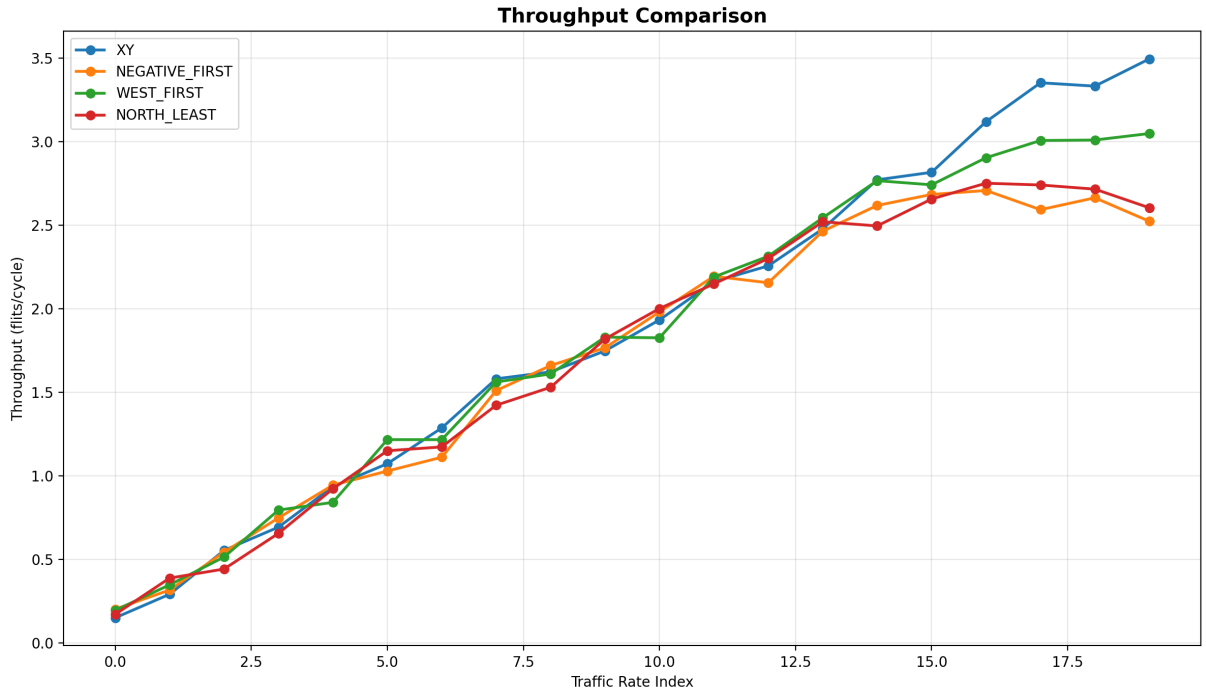


Figura 24 – Experimento 4.2.1. Comportamento da taxa de transferência (*throughput*) para diferentes taxas de injeção de pacotes, em comparação com o cenário descrito em (CHENG; XIANG; HU, 2025)

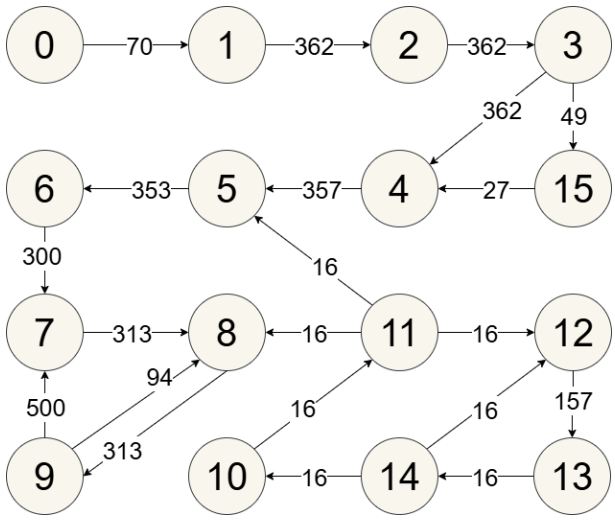


Figura 25 – Aplicação benchmark VOPD usada no Experimento 4.2.2

2007); *heurísticas transformativas*, como Algoritmos Genéticos (GA); e *abordagens híbridas*, exemplificadas pelo ONMAP (KHAN et al., 2018).

Com base nessa configuração, foram executadas as simulações e coletadas as métricas de desempenho, a saber, latência média e *throughput*, conforme apresentado nas Figuras 27 e 28, respectivamente. Na Tabela 6, a latência média e o *throughput* são resumidos para cada técnica de mapeamento.

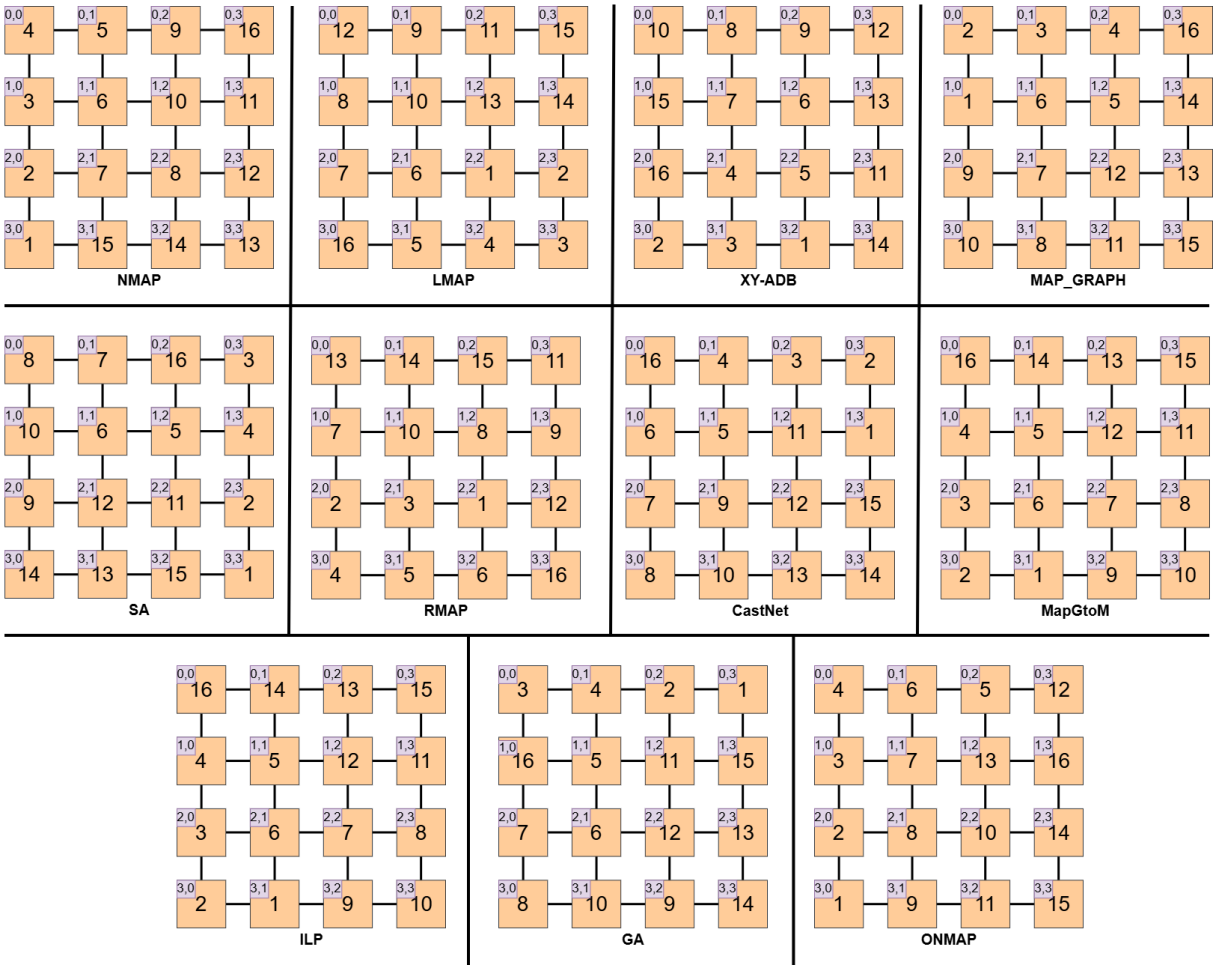


Figura 26 – Experimento 4.2.2. Mapeamento de tarefas da aplicação de benchmark VOPD em diferentes abordagens comparadas em (AMIN et al., 2020)

Os autores em (AMIN et al., 2020) relataram que, para o benchmark VOPD, as estratégias de mapeamento baseadas em ILP, Map_Graph e MapGtoM alcançaram os menores valores de latência, enquanto a abordagem RMAP apresentou a maior latência. Essas observações são consistentes com os resultados obtidos utilizando o simulador proposto.

Como ilustrado na Figura 29, que apresenta a distribuição da latência média para diferentes taxas de injeção de pacotes em cada estratégia de mapeamento, a abordagem RMAP apresenta consistentemente os maiores valores de latência entre os mapeamentos avaliados. Em contraste, os mapeamentos baseados em ILP, Map_Graph e MapGtoM apresentam menores valores médios de latência e menor variabilidade. Esses resultados reforçam a consistência entre o comportamento capturado pelo simulador proposto e as tendências de desempenho relatadas na literatura.

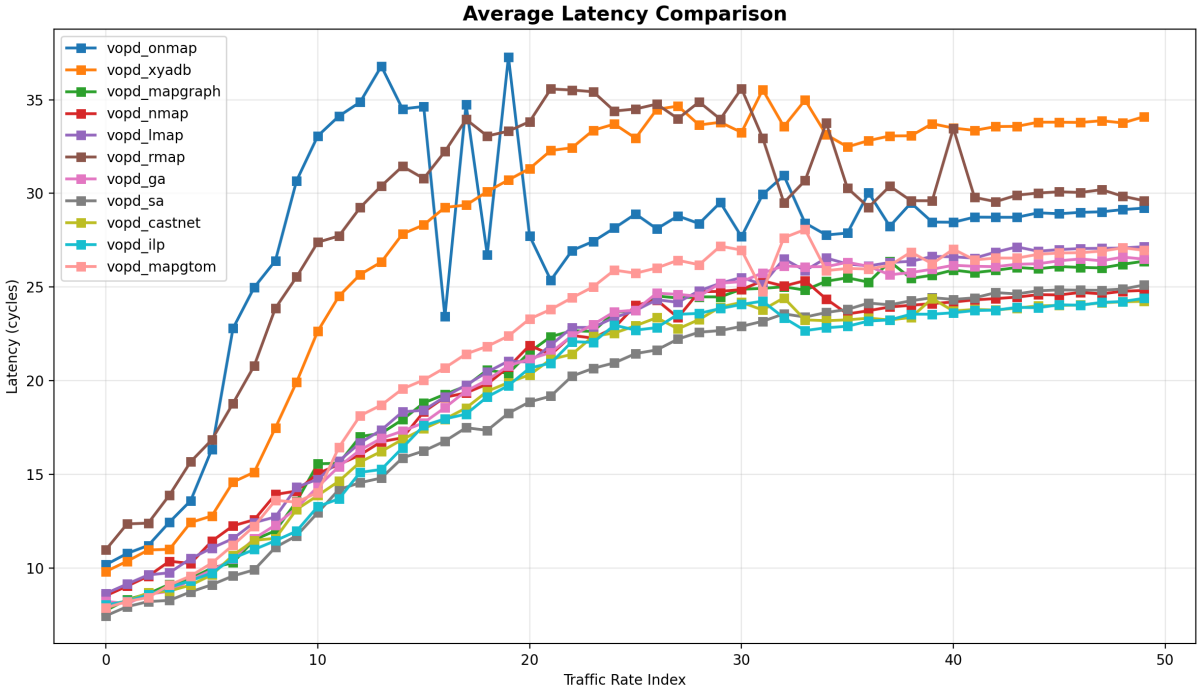


Figura 27 – Experimento 4.2.2. Comportamento da latência média para diferentes taxas de injeção de pacotes, aplicadas para as técnicas de mapeamento descritas em (AMIN et al., 2020)

Tabela 6 – Resultados resumidos obtidos no Experimento 4.2.2

Técnica de Mapeamento	**Latência média (cycles)**	**Throughput (flits/cycle)**
SA	18.91	2.5039
ILP	19.26	2.4969
CastNet	19.40	2.3427
NMAP	20.23	2.3433
Map_Graph	20.61	2.3636
GA	20.68	2.3813
LMAP	21.13	2.3641
MapGtoM	21.67	2.2256
ONMAP	27.33	1.7747
XY_ADB	28.29	2.1048
RMAP	28.83	1.9217

4.2.3 Experimento III

Nesta subseção, é apresentado o Experimento III, cujo objetivo é verificar se o simulador proposto é capaz de refletir, por meio de métricas de rede, as diferenças estruturais entre mapeamentos gerados pelo módulo de otimização multiobjetivo. Para isso, foram considerados cinco mapeamentos distintos da aplicação *CIRCLE* em uma rede 4×4, previamente obtidos na Seção 4.1: um gerado por um algoritmo evolutivo (PESA-II com população inicial aleatória — EV) e quatro derivados de estratégias determinísticas uti-

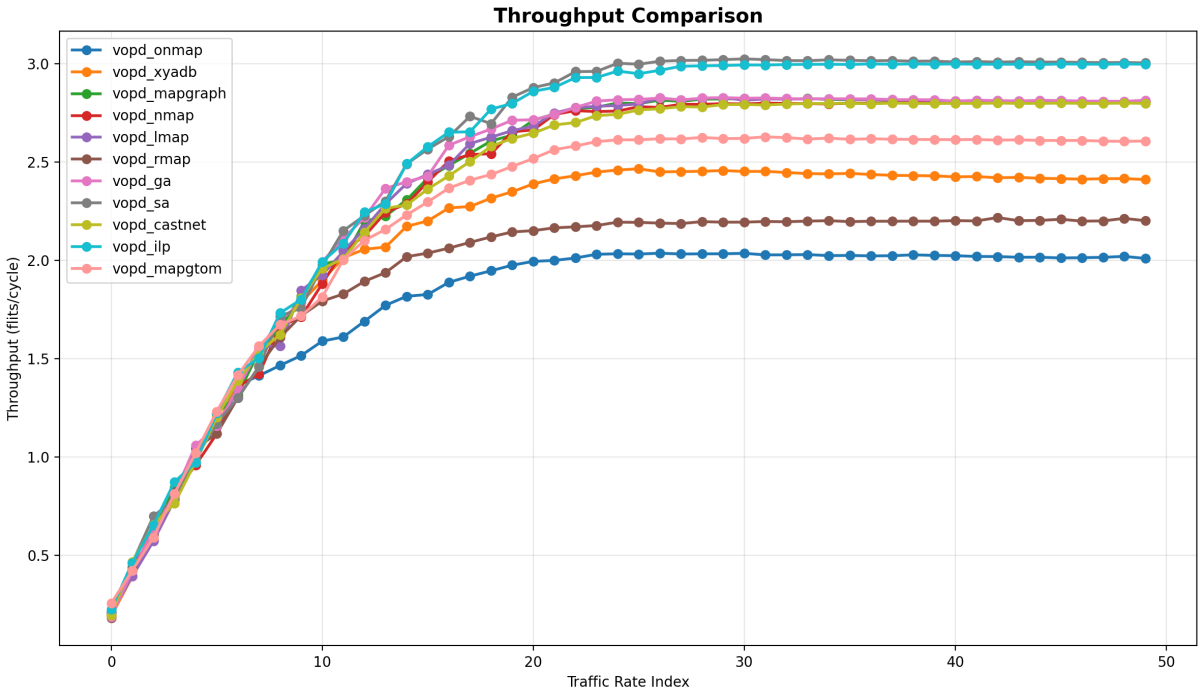


Figura 28 – Experimento 4.2.2. Comportamento da taxa de transferência (*throughput*) para diferentes taxas de injeção de pacotes, aplicadas para as técnicas de mapeamento descritas em (AMIN et al., 2020)

lizadas na geração da população inicial, conforme descrito na Seção 3.3.3 (HR, HS, DR e DS).

Inicialmente, esses mapeamentos foram avaliados quanto ao consumo de energia no módulo de otimização, no qual o mapeamento EV apresentou ganhos expressivos — da ordem de 60% — em relação às abordagens determinísticas. A partir disso, os mesmos mapeamentos foram inseridos no simulador NoC proposto, mantendo-se fixos os parâmetros arquiteturais e de tráfego, especificamente aqueles definidos no Experimento 4.2.2 (Tabela 5), de modo a isolar o impacto do mapeamento no comportamento da rede.

Durante a simulação, foram coletadas métricas como latência média e *throughput*, permitindo a análise comparativa entre os diferentes mapeamentos sob condições equivalentes. Os resultados são apresentados nas Figuras 30 e 31, que ilustram, respectivamente, a latência média e o *throughput* obtidos para cada mapeamento. Conforme observado nessas figuras, o mapeamento EV apresenta os menores valores de latência média e melhores resultados de *throughput* em comparação aos demais, indicando que a otimização energética também se reflete em melhorias no desempenho da rede.

Adicionalmente, observa-se que o simulador, mesmo adotando um modelo de simulação em tempo discreto e uma implementação simplificada, é capaz de capturar de forma consistente as tendências esperadas. Esse resultado reforça a eficácia do simulador em reproduzir o comportamento da rede frente a diferentes estratégias de mapeamento, con-

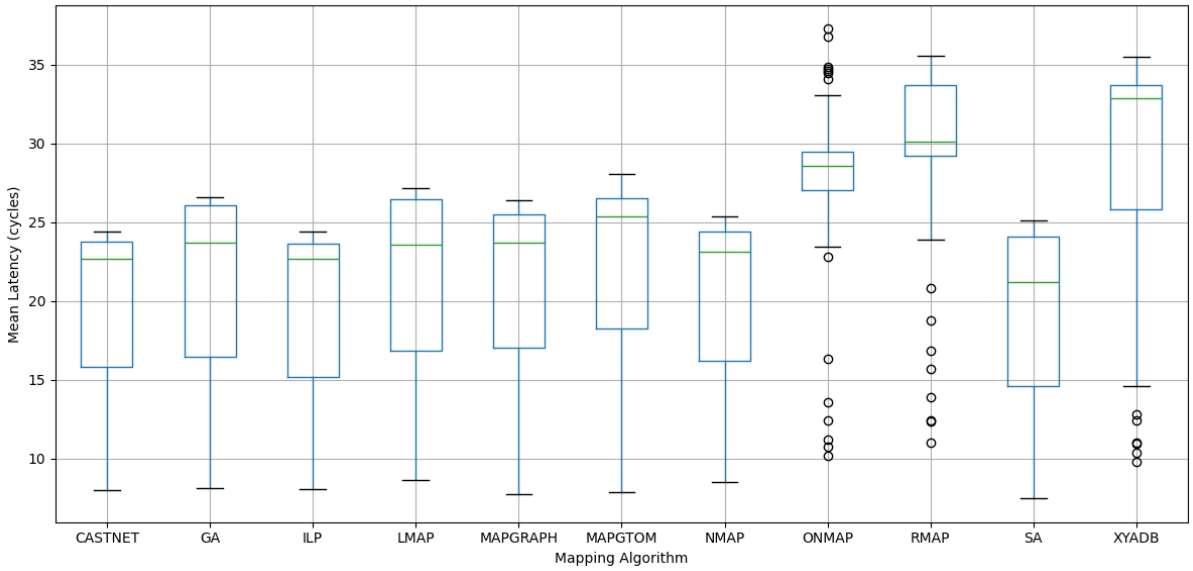


Figura 29 – Experimento 4.2.2. Latência média em ciclos para cada técnica de mapeamento

tribuindo para a validação da abordagem proposta.

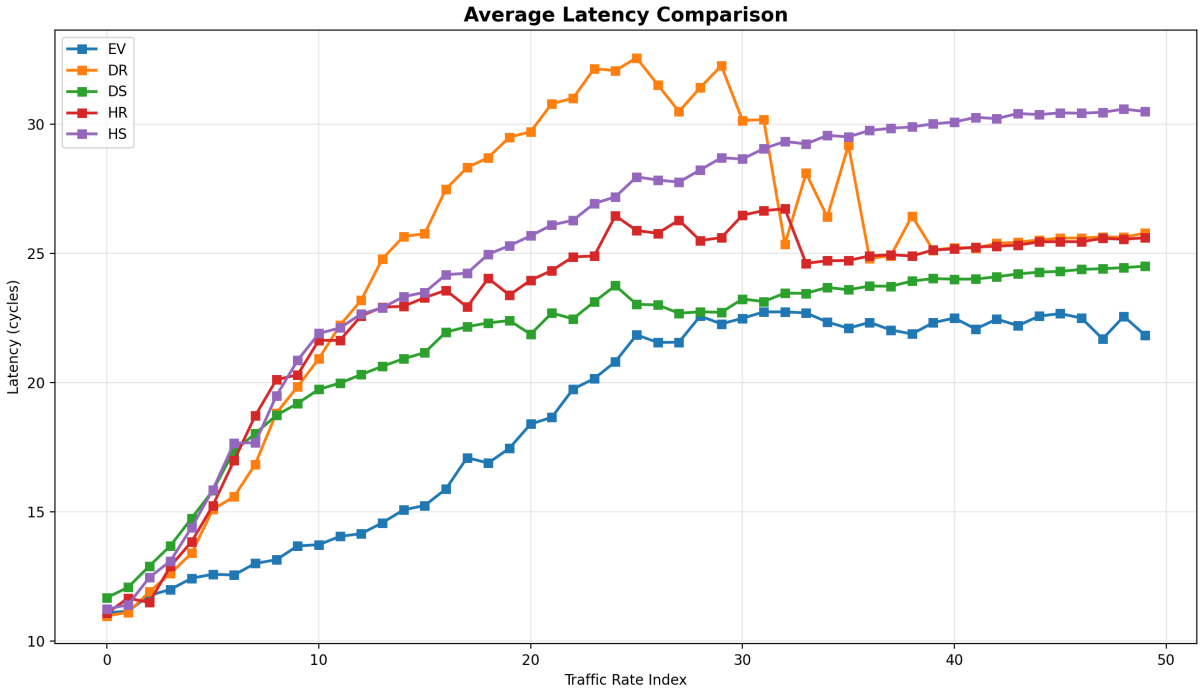


Figura 30 – Experimento 4.2.3. Comportamento da latência média para diferentes taxas de injeção de pacotes, aplicadas para as técnicas de mapeamento obtidas pelo módulo multi-objetivo

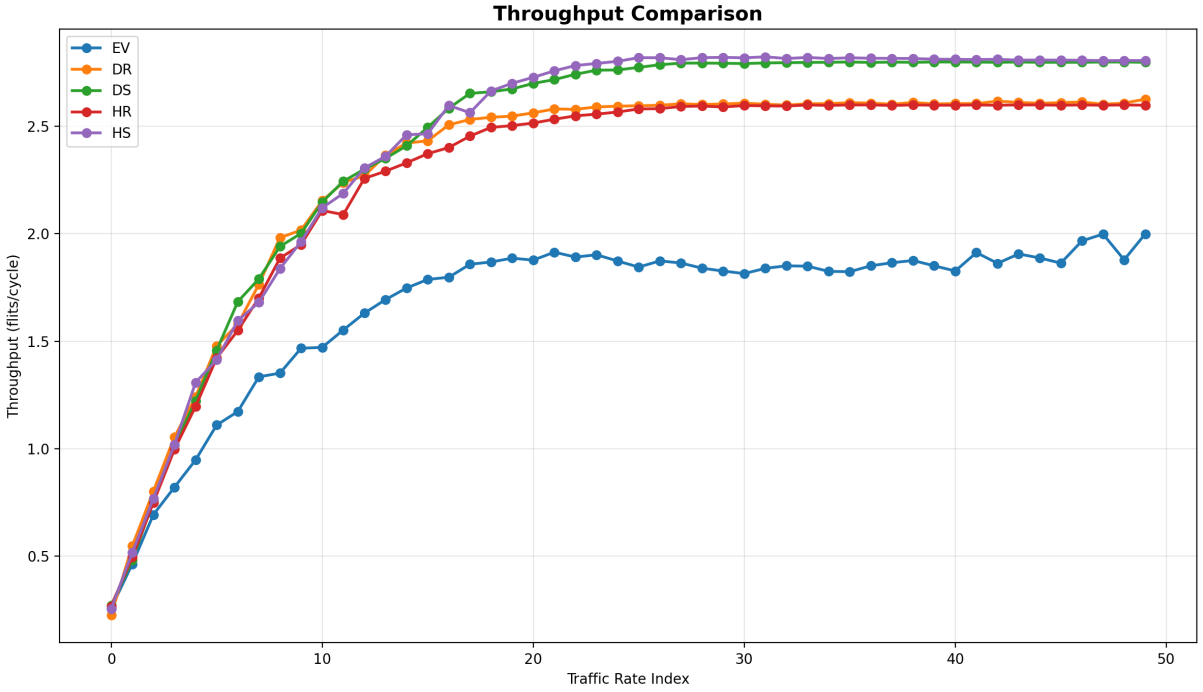


Figura 31 – Experimento 4.2.3. Comportamento da taxa de transferência (*throughput*) para diferentes taxas de injeção de pacotes, aplicadas para as técnicas de mapeamento obtidas pelo módulo multi-objetivo

Conclusões

No presente trabalho, foi proposto um **framework** para estudos de mapeamentos de multiaplicações em arquiteturas many-core, que integra um módulo de otimização multiobjetivo em *MATLAB* e um simulador de NoC implementado em *Python*, denominado **SimpyNoC**.

A abordagem multiobjetivo demonstrou eficiência em alcançar soluções equilibradas entre as métricas analisadas, promovendo um uso mais eficiente da NoC e melhor aproveitamento da área disponível. Foram observadas melhorias superiores a 30% em latência e 38% em tolerância a falhas em comparação com mapeamentos tradicionais, como o *Multifase* (GE et al., 2021). Embora abordagens específicas possam apresentar vantagens pontuais para aplicações isoladas, esses ganhos não se traduzem necessariamente em melhor desempenho global da rede quando múltiplas aplicações são executadas simultaneamente. Nesse contexto, as comparações com estratégias determinísticas, como os *Engineered Mappings* (BONNEY et al., 2016), evidenciam os benefícios quantitativos do uso de técnicas evolutivas multiobjetivo.

O simulador SimpyNoC mostrou-se uma plataforma eficiente e de alto nível para a modelagem e análise de NoCs, abstraindo detalhes de baixo nível e permitindo rápida prototipagem. Sua implementação em *Python*, aliada a um design modular e orientado a objetos, facilita a configuração de topologias, algoritmos de roteamento e estratégias de mapeamento, tornando-o adequado tanto para fins educacionais quanto para pesquisa exploratória. Os resultados experimentais indicam que o simulador é capaz de capturar tendências relevantes de desempenho, como latência média e *throughput*, sob diferentes cenários, corroborando sua utilidade na análise inicial de arquiteturas many-core.

Apesar dos resultados promissores, este trabalho apresenta algumas limitações que abrem espaço para investigações futuras. Nos experimentos realizados, foi considerado apenas um algoritmo de otimização (*PESAII*), tanto para comparação com abordagens determinísticas quanto com um método de objetivo único (*Multifase NSGAII*). Dessa forma, a avaliação pode ser ampliada com a incorporação de outros algoritmos evolutivos

e meta-heurísticas, bem como pela exploração de diferentes configurações de parâmetros, incluindo tamanho da população, taxas de mutação, dimensões da NoC, combinações de aplicações e volumes de tarefas.

Nesse sentido, como continuidade natural deste trabalho, propõe-se a expansão do simulador para suportar modelos de tráfego mais realistas, algoritmos de roteamento adaptativo e a incorporação de métricas energéticas diretamente no processo de simulação. Adicionalmente, a integração de ferramentas de visualização dinâmica pode contribuir para a análise em tempo real do comportamento da rede, enquanto o uso de técnicas de paralelização pode aumentar a escalabilidade do simulador, permitindo a avaliação de arquiteturas many-core de maior porte. Tais avanços têm potencial para consolidar o framework proposto como uma ferramenta robusta para apoio à pesquisa e ao ensino na área de sistemas baseados em NoC.

Referências

ACHBALLAH, A. B.; OTHMAN, S. B.; SAOUD, S. B. Problems and challenges of emerging technology networksonchip: A review. **Microprocessors and Microsystems**, v. 53, p. 1–20, 2017. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933116303593>.

AGARWAL, N. et al. Garnet: A detailed on-chip network model inside a full-system simulator. In: **2009 IEEE International Symposium on Performance Analysis of Systems and Software**. [S.l.: s.n.], 2009. p. 33–42.

ALIKHAH-ASL, E.; RESHADI, M. Xy-axis and distance based noc mapping (xy-adb). In: **2016 8th International Symposium on Telecommunications (IST)**. [S.l.: s.n.], 2016. p. 678–683.

AMIN, W. et al. Performance evaluation of application mapping approaches for network-on-chip designs. **IEEE Access**, v. 8, p. 63607–63631, 2020.

_____. Hydra: Hybrid task mapping application framework for noc-based mpsocs. **IEEE Access**, v. 11, p. 52309–52326, 2023.

AYARI, R. et al. Schedulability-guided exploration of multi-core systems. In: **Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype**. New York, NY, USA: Association for Computing Machinery, 2016. (RSP '16), p. 121–127. ISBN 9781450345354. Disponível em: <https://doi-org.ez31.periodicos.capes.gov.br/10.1145/2990299.2990319>.

BAMBHANIYA, A. R. et al. Proteus: Hls-based noc generator and simulator. In: **2023 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2023. p. 1–6.

BEN-ITZHAK, Y. et al. Hnocs: Modular open-source simulator for heterogeneous nocs. In: **2012 International Conference on Embedded Computer Systems (SAMOS)**. [S.l.: s.n.], 2012. p. 51–57.

BENINI, L.; MICHELI, G. D. Networks on chips: a new soc paradigm. **Computer**, v. 35, n. 1, p. 70–78, 2002.

BINKERT, N. et al. The gem5 simulator. **ACM SIGARCH Computer Architecture News**, v. 39, n. 2, p. 1–7, 2011.

BOLCHINI, C. et al. Run-time mapping for reliable many-cores based on energy/performance trade-offs. In: **2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)**. [S.l.: s.n.], 2013. p. 58–64.

BONNEY, C. et al. Fault tolerant task mapping on many-core arrays. In: **2016 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.: s.n.], 2016. p. 1–8.

BORKAR, S. Thousand core chips: a technology perspective. In: **Proceedings of the 44th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2007. (DAC '07), p. 746–749. ISBN 9781595936271. Disponível em: <https://doi.org/10.1145/1278480.1278667>.

CATANIA, V. et al. Cycle-accurate network on chip simulation with noxim. **ACM Trans. Model. Comput. Simul.**, Association for Computing Machinery, New York, NY, USA, v. 27, n. 1, ago. 2016. ISSN 1049-3301. Disponível em: <https://doi.org/10.1145/2953878>.

CENG, J. et al. Maps: An integrated framework for mpsoc application parallelization. In: **2008 45th ACM/IEEE Design Automation Conference**. [S.l.: s.n.], 2008. p. 754–759.

CHENG, Z.; XIANG, X.; HU, C. Teaching research and practice for computer composition and structure based on network-on-chip. In: **Proceedings of the 2025 International Conference on Big Data and Informatization Education**. New York, NY, USA: Association for Computing Machinery, 2025. (ICBDIE '25), p. 255–260. ISBN 9798400714405.

CHOU, C.-L.; MARCULESCU, R. Farm: Fault-aware resource management in noc-based multiprocessor platforms. In: **2011 Design, Automation & Test in Europe**. [S.l.: s.n.], 2011. p. 1–6.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **IEEE Transactions on Evolutionary Computation**, v. 6, n. 2, p. 182–197, 2002.

DICK, R.; RHODES, D.; WOLF, W. Tgff: task graphs for free. In: **Proceedings of the 6th international workshop on hardware/software codesign**. [S.l.]: IEEE Computer Society, 1998. p. 97–101. ISBN 9780818684425. ISSN 1092-6100.

EMERETLIS, A. et al. A multi-stage hybrid approach for mapping applications on heterogeneous multi-core platforms. In: **2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2022. p. 1–6.

FARMAN, A.; DURRANI, Y. A. Design, implementation, and power analysis for network-on-chip architectures. In: **2023 7th International Multi-Topic ICT Conference (IMTIC)**. [S.l.: s.n.], 2023. p. 1–7.

FARUQUE, M. A. A.; KRIST, R.; HENKEL, J. Adam: Run-time agent-based distributed application mapping for on-chip communication. In: **2008 45th ACM/IEEE Design Automation Conference**. [S.l.: s.n.], 2008. p. 760–765.

GAFFOUR, K. et al. Survey of network-on-chip simulators. In: . [S.l.: s.n.], 2017.

GE, F. et al. A multi-phase based multi-application mapping approach for many-core networks-on-chip. **Micromachines**, v. 12, n. 6, 2021. ISSN 2072-666X. Disponível em: <https://www.mdpi.com/2072-666X/12/6/613>.

GUPTA, M.; BHARGAVA, L.; INDU, S. Mapping techniques in multicore processors: Current and future trends. **The Journal of Supercomputing**, v. 77, n. 8, p. 9308–9363, ago. 2021. ISSN 1573-0484. Disponível em: <https://doi.org/10.1007/s11227-021-03650-6>.

HARADA, T.; ALBA, E. Parallel genetic algorithms: A useful survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, aug 2020. ISSN 0360-0300. Disponível em: <https://doi-org.ez31.periodicos.capes.gov.br/10.1145/3400031>.

HASSAN, A. S.; MORGAN, A. A.; EL-KHARASHI, M. W. Clustered networks-on-chip: Simulation and performance evaluation. **International Journal of Computing and Digital Systems**, v. 6, n. 02, p. 51–61, 2017.

HENKEL, J.; WOLF, W.; CHAKRADHAR, S. On-chip networks: a scalable, communication-centric embedded system design paradigm. In: **17th International Conference on VLSI Design. Proceedings.** [S.l.: s.n.], 2004. p. 845–851.

HILL, M. D.; MARTY, M. R. Amdahl's law in the multicore era. **Computer**, v. 41, n. 7, p. 33–38, 2008.

HONG, S. et al. Process variation aware thread mapping for chip multiprocessors. In: **2009 Design, Automation Test in Europe Conference Exhibition**. [S.l.: s.n.], 2009. p. 821–826.

JAIN, L. et al. Nirgam: a simulator for noc interconnect routing and application modeling. In: **Design, Automation and Test in Europe Conference (DATE)**. [S.l.]: IEEE, 2007. p. 1146–1151.

JANG, H. et al. Developing a multicore platform utilizing open risc-v cores. **IEEE Access**, v. 9, p. 120010–120023, 2021.

JERRAYA, A.; WOLF, W. **Multiprocessor Systems-on-Chips**. [S.l.]: Morgan Kaufmann Publishers Inc., 2005. ISBN 978-0-12-385251-9.

JIANG, N. et al. A detailed and flexible cycle-accurate network-on-chip simulator. In: **Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software**. [S.l.]: IEEE, 2013. p. 86–96.

JIANG, Q.; XU, J.; CHEN, Y. A genetic algorithm for scheduling in heterogeneous multicore system integrated with fpga. In: **2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)**. [S.l.: s.n.], 2021. p. 594–602.

JOSEPH, J. M.; PIONTECK, T. A cycle-accurate network-on-chip simulator with support for abstract task graph modeling. In: **2014 International Symposium on System-on-Chip (SoC)**. [S.l.: s.n.], 2014. p. 1–6.

KHALILI, F.; ZARANDI, H. R. A fault-tolerant low-energy multi-application mapping onto noc-based multiprocessors. In: **2012 IEEE 15th International Conference on Computational Science and Engineering**. [S.l.: s.n.], 2012. p. 421–428.

KHAN, S. et al. An optimized hybrid algorithm in term of energy and performance for mapping real time workloads on 2d based on-chip networks. **Applied Intelligence**, v. 48, n. 12, p. 4792–4804, Dec 2018. ISSN 1573-7497. Disponível em: <https://doi.org/10.1007/s10489-018-1246-7>.

KHASANOV, R.; CASTRILLON, J. Energy-efficient runtime resource management for adaptable multi-application mapping. In: **2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)**. [S.l.: s.n.], 2020. p. 909–914.

KHULLER, S.; YANG, S. Revisiting connected dominating sets: An optimal local algorithm? In: **2018 Information Theory and Applications Workshop (ITA)**. [S.l.: s.n.], 2018. p. 1–9.

KOBBE, S. et al. Distrm: Distributed resource management for on-chip many-core systems. In: **2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)**. [S.l.: s.n.], 2011. p. 119–128.

KUMAR, A.; RAO, T.; BEECHU, N. An efficient real-time embedded application mapping for noc based multiprocessor system on chip. 09 2021.

LE, Q. et al. Pareto optimal mapping for tile-based network-on-chip under reliability constraints. **International Journal of Computer Mathematics**, Taylor & Francis, v. 92, n. 1, p. 41–58, 2015. Disponível em: <https://doi.org/10.1080/00207160.2014.892073>.

LIS, M. et al. Darsim: a parallel cycle-level noc simulator. **MoBS 2010 - Sixth Annual Workshop on Modeling, Benchmarking and Simulation**, 06 2010.

LIU, N.; TEMPESTI, G. Predictive age-aware runtime remapping for many-core systems. In: **2024 8th International Conference on System Reliability and Safety (ICSRS)**. [S.l.: s.n.], 2024. p. 812–821.

LIU, Y.; KATO, S.; EDAHIRO, M. Analysis of memory system of tiled many-core processors. **IEEE Access**, v. 7, p. 18964–18977, 2019.

LU, Z.; XIA, L.; JANTSCH, A. Cluster-based simulated annealing for mapping cores onto 2d mesh networks on chip. In: **2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems**. [S.l.: s.n.], 2008. p. 1–6.

MARCULESCU, R. et al. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 28, n. 1, p. 3–21, 2009.

MARWEDEL, P. et al. Mapping of applications to mpsocs. In: **2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)**. [S.l.: s.n.], 2011. p. 109–118.

MIAO, Z. et al. Review of task-scheduling methods for heterogeneous chips. **Electronics**, v. 14, n. 6, p. 1191, 2025.

MITTAL, S. A survey of techniques for architecting and managing asymmetric multicore processors. Association for Computing Machinery, New York, NY, USA, v. 48, n. 3, fev. 2016. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/2856125>.

MURALI, S.; MICHELI, G. D. Bandwidth-constrained mapping of cores onto noc architectures. In: **Proceedings Design, Automation and Test in Europe Conference and Exhibition**. [S.l.: s.n.], 2004. v. 2, p. 896–901 Vol.2.

MURALIDHAR, R.; BOROVICA-GAJIC, R.; BUYYA, R. Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 11s, sep 2022. ISSN 0360-0300. Disponível em: <https://doi-org.ez31.periodicos.capes.gov.br/10.1145/3511094>.

MYUNG, W.; QI, Z.; CHENG, M. Performance analysis of routing algorithms in mesh based network on chip using booksim simulator. In: **2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE)**. [S.l.: s.n.], 2019. p. 297–300.

NAKAJIMA, K. et al. Naxim: A fast and retargetable network-on-chip simulator with qemu and systemc. **International Journal of Networking and Computing**, v. 3, p. 217–227, 01 2013.

ORENES-VERA, M. et al. Muchisim: A simulation framework for design exploration of multi-chip manycore systems. In: **2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)**. [S.l.: s.n.], 2024. p. 48–60.

ORSILA, H. et al. Automated memory-aware application distribution for multi-processor system-on-chips. **Journal of Systems Architecture**, v. 53, n. 11, p. 795–815, 2007. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762107000197>.

OSTLER, C.; CHATHA, K. S. An ilp formulation for system-level application mapping on network processor architectures. In: **2007 Design, Automation Test in Europe Conference Exhibition**. [S.l.: s.n.], 2007. p. 1–6.

PATOOGHY, A.; TABKHI, H.; MIREMADI, S. G. Rmap: A reliability-aware application mapping for network-on-chips. In: **2010 Third International Conference on Dependability**. [S.l.: s.n.], 2010. p. 112–117.

POURMOHSENI, B. et al. Hybrid application mapping for composable many-core systems: Overview and future perspective. **Journal of Low Power Electronics and Applications**, v. 10, n. 4, 2020. ISSN 2079-9268. Disponível em: <https://www.mdpi.com/2079-9268/10/4/38>.

R, U.; H., S.; V., S. Network-on-chip (noc) - routing techniques: A study and analysis. In: **2019 Global Conference for Advancement in Technology (GCAT)**. [S.l.: s.n.], 2019. p. 1–6.

RAMAKRISHNAN, N.; SANJU, V.; FARHANAAZ. Analysis of network on chip topologies. In: **2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC)**. [S.l.: s.n.], 2023. p. 1–6.

REDDY, B. N. K.; RAHMAN, M. Z. U.; LAY-EKUAKILLE, A. Enhancing reliability and energy efficiency in many-core processors through fault-tolerant network-on-chip. **IEEE Transactions on Network and Service Management**, v. 21, n. 5, p. 5049–5062, 2024.

ROCHA, H. M. G. de A. et al. A routing based genetic algorithm for task mapping on mpsoc. In: **2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.: s.n.], 2020. p. 1–8.

SAHU, P. K. et al. A constructive heuristic for application mapping onto mesh based network-on-chip. **Journal of Circuits, Systems and Computers**, v. 24, n. 08, p. 1550126, 2015.

_____. A new application mapping algorithm for mesh based network-on-chip design. In: **2010 Annual IEEE India Conference (INDICON)**. [S.l.: s.n.], 2010. p. 1–4.

SEPúLVEDA, J. et al. A multi-objective approach for multi-application noc mapping. In: **2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2011. p. 1–4.

SHARMA, P. K.; BISWAS, S.; MITRA, P. Energy efficient heuristic application mapping for 2-d mesh-based network-on-chip. **Microprocessors and Microsystems**, v. 64, p. 88–100, 2019. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933117304039>.

SINGH, A. K. et al. Mapping on multi/many-core systems: Survey of current and emerging trends. In: **2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2013. p. 1–10.

TENDULKAR, P. **Mapping and Scheduling on Multi-core Processors using SMT Solvers**. Tese (Doutorado), 10 2014.

THEOCHARIDES, T. et al. Towards embedded runtime system level optimization for mpsocs: on-chip task allocation. In: . New York, NY, USA: Association for Computing Machinery, 2009. (GLSVLSI '09), p. 121–124. ISBN 9781605585222. Disponível em: <https://doi.org/10.1145/1531542.1531573>.

THIELE, L. et al. Mapping applications to tiled multiprocessor embedded systems. In: **Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)**. [S.l.: s.n.], 2007. p. 29–40.

TIAN, Y. et al. Platemo: A matlab platform for evolutionary multi-objective optimization [educational forum]. **IEEE Computational Intelligence Magazine**, v. 12, n. 4, p. 73–87, 2017.

TONETTO, R. B. et al. A machine learning approach for reliability-aware application mapping for heterogeneous multicores. In: **2020 57th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2020. p. 1–6.

TOSUN, S. New heuristic algorithms for energy aware application mapping and routing on mesh-based nocs. **Journal of Systems Architecture**, v. 57, n. 1, p. 69–78, 2011. ISSN 1383-7621. Special Issue On-Chip Parallel And Network-Based Systems. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762110001256>.

TRAN, A. T.; BAAS, B. M. Noctweak: a highly parameterizable simulator for early exploration of performance and energy of networks on-chip. In: . [s.n.], 2012. Disponível em: <https://api.semanticscholar.org/CorpusID:18109445>.

XU, C.; LIU, Y.; YANG, Y. A multi-objective architecture optimization method for application-specific noc design. In: **2018 31st IEEE International System-on-Chip Conference (SOCC)**. [S.l.: s.n.], 2018. p. 130–135.

XU, T. C.; LEPPäNEN, V. An efficient dynamic energy-aware application mapping algorithm for multicore processors. In: **2016 Sixth International Conference on Digital Information Processing and Communications (ICDIPC)**. [S.l.: s.n.], 2016. p. 119–124.

YANG, B. et al. Multi-application multi-step mapping method for many-core network-on-chips. In: **NORCHIP 2010**. [S.l.: s.n.], 2010. p. 1–6.

ZHU, D. et al. Balancing on-chip network latency in multi-application mapping for chip-multiprocessors. In: **2014 IEEE 28th International Parallel and Distributed Processing Symposium**. [S.l.: s.n.], 2014. p. 872–881.

_____. Providing balanced mapping for multiple applications in many-core chip multiprocessors. **IEEE Transactions on Computers**, v. 65, n. 10, p. 3122–3135, 2016.

ZHU, K.; DING, Y. Research on low power scheduling of heterogeneous multi core mission based on genetic algorithm. In: **2017 9th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)**. [S.l.: s.n.], 2017. p. 219–223.

Apêndices

APÊNDICE A

Artigos produzidos

Artigo submetido ao periódico IEEE Access em 06/03/2026

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

SimpyNoC: A Configurable High-Level Network-on-Chip Simulator in Python

IGOR F. GALLON[1], GIANLUCA TEMPESTI[2], EMERSON C. PEDRINO[3]
[1]Federal University of São Carlos, São Carlos, SP 13565 905 BR (e-mail: igorgallon@estudante.ufscar.br)
[2]University of York, York, YO10 5DD UK (e-mail: gianluca.tempesti@york.ac.uk)
[3]Federal University of São Carlos, São Carlos, SP 13565 905 BR (e-mail: emerson@ufscar.br)

Corresponding author: Emerson C. Pedrino (e-mail: emerson@ufscar.br).

This work was supported by FAPESP (Grant 2017/26421-3 and 2023/00212-0) and CAPES

ABSTRACT Network-on-Chip (NoC) architectures have become a fundamental communication paradigm for many-core systems, requiring efficient evaluation methodologies to analyze routing strategies, task mapping policies, and architectural parameters. Although numerous NoC simulators have been proposed in the literature, most of them are implemented in low-level languages such as C++ or SystemC and are tightly coupled to hardware-specific assumptions. While these tools provide detailed and cycle-accurate modeling, they often require complex configuration, technology-dependent calibration, and substantial implementation effort, which may limit their suitability for rapid design-space exploration, high-level experimentation, and educational purposes. This paper presents SimpyNoC, a lightweight and configurable Python-based Network-on-Chip simulator designed to support fast prototyping and systematic evaluation of NoC architectures. The proposed framework adopts a modular and object-oriented design, enabling configurable Mesh topologies, pluggable routing algorithms, and flexible task mapping strategies under a discrete-time simulation model. By prioritizing simplicity, accessibility, and ease of configuration, SimpyNoC allows rapid experimentation without requiring detailed hardware-oriented modeling. Experimental evaluations demonstrate that the simulator effectively supports comparative analysis of deterministic routing algorithms and different mapping configurations, providing consistent performance metrics such as latency and packet delivery statistics. The results highlight the suitability of the proposed approach for rapid architectural exploration and academic use, offering a practical alternative to hardware-centric simulation tools while maintaining meaningful analytical capability.

INDEX TERMS Network-on-Chip (NoC), simulator, python, application mapping

I. INTRODUCTION

THE maximum operating frequency of single-core processors has reached practical limits due to power dissipation and thermal constraints, which have significantly reduced the effectiveness of traditional frequency scaling as a primary source of performance improvement. As the benefits of voltage and frequency scaling diminished in recent technology generations, the industry shifted toward exploiting task-level parallelism via multicore and many-core designs [1], [2]. As a result, integrated circuit (IC) manufacturers have increasingly adopted multicore architectures in which multiple processing cores operate at lower frequencies while improving overall throughput through concurrent execution [3]–[5].

Furthermore, the computational demands of modern applications—such as video processing, big data analytics, and machine learning—have grown substantially in recent years and can no longer be met solely by increasing the clock frequency of a single-core processor or by relying on highly customized architectures [6], [7]. Therefore, multicore and many-core platforms have emerged as a dominant paradigm within contemporary *Systems-on-Chip* (SoCs), requiring scalable and energy-efficient on-chip communication infrastructures [8].

However, traditional on-chip interconnect architectures, such as shared buses and crossbars, face major limitations in scalability and wiring complexity as the number of components increases. In this context, NoC has emerged as an efficient communication paradigm within SoC, providing a structured network of routers and links responsible for data exchange among processing elements. By adopting packet-based communication, NoCs improve scalability and better

Artigo publicado ao periódico IEEE Access em 24/03/2025

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 6 February 2025, accepted 21 March 2025, date of publication 24 March 2025, date of current version 2 April 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3554478

RESEARCH ARTICLE

An Evolutionary Toolchain for Morphological Filter Mapping on Many-Core Architectures

EMERSON C. PEDRINO[1], DENIS P. LIMA[1], IGOR F. GALLON[1], VALENTIN O. RODA[2], NAIJIA LIU[3], (Graduate Student Member, IEEE), AND GIANLUCA TEMPESTI[3], (Member, IEEE)

[1]Department of Computer Science, Federal University of São Carlos, São Carlos 13565-905, Brazil
[2]Electrical Engineering Department, Federal University of Rio Grande do Norte, Natal 59078-900, Brazil
[3]School of Physics, Engineering and Technology, University of York, YO10 5DD York, U.K.

Corresponding author: Emerson C. Pedrino (emerson@ufscar.br)

This work was supported in part by the U.K. Research and Innovation Engineering and Physical Sciences Research Council (UKRI EPSRC) Graceful Project under Grant EP/L000563/1. The work of Emerson C. Pedrino was supported in part by São Paulo Research Foundation (FAPESP) under Grant 2017/26421-3 and Grant 2023/00212-0, and in part by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES).

ABSTRACT Many-core systems are systolic architectures consisting of an arbitrarily large number of processing nodes connected by a point-to-point communication network. Their architecture makes them ideally suited for the implementation of data-flow algorithms, of which Mathematical Morphology (MM) filters are a typical example. However, the performance of data-flow applications on many-core systems is highly dependent on the quality of the mapping of the application tasks to the computational cores. Decomposing the structuring elements of morphological operations improves their performance, however, performing such decomposition on many-core systems leads to increased communication. The need to find a balance between performance and communication is a representative example of the general problem of mapping optimizations. The approach presented in this paper explores a two-phase design-time optimization toolchain based on evolutionary algorithms: a front-end single-objective algorithm decomposes a MM filter using smaller operators, while a back-end multi-objective (fault-tolerance, energy, and communication) algorithm searches for optimal mappings of the filter on a specific many-core system, taking into account the architectural parameters of the hardware. The output of the toolchain is a Pareto front of mapping solutions, allowing the designer to select an implementation that matches application-specific requirements. A set of standard benchmark applications was used to determine the optimal parameters for the algorithms, which were then validated on two real-world application examples involving the detection of features in high-resolution PCB images. Two application mapping experiments focusing on energy constraints were conducted, in which the proposed procedure was compared to deterministic mapping techniques. The evolutionary procedure was observed to offer a significant advantage over the deterministic approach, with percentage gains of up to 73.78% for smaller grids.

INDEX TERMS Image processing, many-core systems, mathematical morphology, multiobjective optimization, tasks mapping.

I. INTRODUCTION

Mathematical Morphology (MM) is a non-linear branch of computer vision, based on geometry and on the mathematical Theory of Order [1], [2], [3], [4]. It was developed by Matheron and Serra in the 1960s, focusing primarily on biomedical and geological image analysis problems [5]. Later, it proved to be a powerful tool for computer vision tasks involving binary and grey-level images for noise suppression, skeletonization, segmentation, pattern recognition, and for the design of morphological filters [6], [7]. In the 1980s, its theory was generalized for complete lattices, allowing it to be applied to color and hyperspectral images [8], [9].

The basic operators of MM are *dilation* and *erosion* (which can be combined algorithmically to form other

The associate editor coordinating the review of this manuscript and approving it for publication was Mehul S. Raval.

Artigo publicado ao periódico IEEE Latin America Transactions em 04/04/2025

A New Solution based on Multi-objective Algorithm for Multi-application Mappings for Many-Core Systems

M. A. Almeida, I. F. Gallon, and E. C. Pedrino

***Abstract*— Mapping multiple applications onto intra-chip communication structures, such as Network-on-Chip (NoC), commonly used in the context of manycore systems, is a problem classified as NP-hard. This is due to the need for the simultaneous optimization of performance, reliability, and energy efficiency metrics. For instance, it is essential to account for heat dissipation effects throughout the system, while also ensuring fault tolerance and proper load balancing. While there is still a limited number of works in the literature addressing the allocation of multi-applications in NoCs, many studies focus on the mapping of single applications. Therefore, this paper proposes a multi-objective mapping model that targets performance metrics and heat distribution for multi-application scenarios in manycores with NoCs, aiming to contribute to this area. The results obtained are compared with a state-of-the-art algorithm for this scenario, showing promising improvements, with more than 30% reduction in latency and 38% increase in fault tolerance when all applications are considered.**

Link to graphical and video abstracts, and to code: https://latamt.ieeer9.org/index.php/transactions/article/view/9346

***Index Terms*—many-core, network-on-chip, task mapping, metrics.**

I. Introduction

TECHNOLOGICAL advances in the manufacture of ICs (Integrated Circuits) have enabled the integration of a greater number of components onto a single chip. This progress has led to the construction of devices with an increasing number of processing cores, especially in embedded systems. For some years now, multicore and manycore devices have been utilizing lower operating frequencies [1]–[3] because the performance demands can no longer be met by increasing the frequency of single-core operation alone [4]–[7]. Another challenge in this scenario is the distribution of mono-application or multi-application tasks among these cores, which is not trivial to solve in practice. Depending on the arrangement chosen, latency and/or power distribution throughout the chip may be compromised. In manycore devices, the basic principle used to address this problem involves dividing applications into smaller tasks that can be efficiently allocated to different cores for parallel execution [8], [9]. This approach allows for better synchronization control, memory management, and reliability in executing parallel code [10], [11]. Therefore, mapping application tasks in these systems involves establishing a communication scheme between them, aiming to optimize hardware variables such as energy consumption, computational performance, voltage, frequency, and others, as exemplified in [12]. However, the optimization involving two or more objectives in multi-applications for NoC mapping can be further explored.

Furthermore, as stated in [13]–[15], to optimize communication delays and energy consumption, tasks should be mapped to the same core or close to each other. These optimizations are necessary to meet the constraints of each application. Thus, there is a need to develop more efficient mapping methodologies that can provide optimal mappings satisfying the demands of different applications. This problem is classified as NP-hard [13], so heuristics based on application domain knowledge must be employed to find an optimized solution.

Current strategies for task distribution in systems with many cores depend on the number of applications to be executed. In scenarios with fixed or variable workloads, improvements to task distribution in these architectures are made during the design or execution phase, respectively. These architectures can be uniform (with identical cores) or heterogeneous (with different cores). During runtime distribution, one core manages tasks such as scheduling, resource control, configuration, and migration. This distribution can be centralized, distributed, or a combination of both [8], [16], [17]. Distributions made during the design phase are best suited for static workloads, where a predefined set of applications with known computing and communication behaviors are considered. Dynamic changes to applications while running [18], [19] are not supported on these systems.

In [24] combine the benefits of evolution-based search with a learning-based local search to quickly determine the PE and communication link placement to optimize multiple objectives (e.g., latency, throughput, and energy) in 3D NoC-enabled manycore heterogeneous systems. It is a promising technique that can be benchmarked in future work.

[25] is a proposed technique that uses an ML-based model to extract relevant information from research data and incorporate it into the research process. This results in a more robust model with a higher convergence rate and solution quality.

The associate editor coordinating the review of this manuscript and approving it for publication was Ruth Aguilar (*Corresponding author: Manoel de Almeida*).

E. C. Pedrino is grateful to FAPESP (Grant 2017/26421-3 and 2023/00212-0) and CAPES.

Manoel de Almeida, I. F. Gallon, and E. C. Pedrino are with Federal University of Sao Carlos, Rodovia Washington Luis, São Carlos, Brazil (e-mails: manoel.aranda@ufscar.br, igorfelipegallon@gmail.com, and emerson@ufscar.br).