Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Luneque Del Rio de Souza e Silva Junior

Roteamento em redes embutidas utilizando otimização por colônia de formigas

Rio de Janeiro
2011

Luneque Del Rio de Souza e Silva Junior

Roteamento em redes embutidas utilizando otimização por colônia de formigas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof.ª Dr.ª Nadia Nedjah
Coorientadora: Prof.ª Dr.ª Luiza de Macedo Mourelle

Rio de Janeiro
2011

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC / B

S586 Silva Jr., Luneque Del Rio de Souza e
Roteamento em redes embutidas utilizando otimização por colônia de formigas/Luneque Del Rio de Souza e Silva Junior. – 2011.
162 f.

Orientadora: Nadia Nedjah.
Coorientadora: Luiza de Macedo Mourelle.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Eletrônica - Teses. 2. Rede de Computador - Teses. I. Nedjah, Nadia. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 621.38:004.7

Assinatura Data

Luneque Del Rio de Souza e Silva Junior

Roteamento em redes embutidas utilizando otimização por colônia de formigas

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em: 19 de dezembro de 2011.

Banca Examinadora:

Prof.ª Dr.ª Nadia Nedjah (Orientadora)
Faculdade de Engenharia - UERJ

Prof.ª Dr.ª Luiza de Macedo Mourelle (Coorientadora)
Faculdade de Engenharia - UERJ

Prof. Dr. Leandro Nunes de Castro
Universidade Presbiteriana Mackenzie

Prof. Dr. Cláudio Luis de Amorim
Universidade Federal do Rio de Janeiro - COPPE/UFRJ

Rio de Janeiro
2011

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela possibilidade de ter nascido nesta época de grandes avanços. Ele é tão incrível, que nos criou com o intelecto necessário para entendermos tanto a nós mesmos quanto a este universo onde vivemos. Fantástico.

Agradeço à CAPES pelo auxílio financeiro recebido durante este curso de mestrado, sem a qual não poderia ter desenvolvido este trabalho em tempo integral.

Agradeço a todos os professores do PEL-UERJ pelas incríveis lições aprendidas neste dois anos de curso. Em especial, agradeço às professoras Nadia Nedjah e Luiza de Macedo Mourelle, não apenas pela orientação neste trabalho e pela oportunidade de desenvolvê-lo no LSAC, mas pela compreensão em lidar com um aluno tão preguiçoso. Obrigado por tudo, professoras!

Agradeço aos colegas de mestrado, Richard, Fábio, Rogério e Rafael. Obrigado pelas conversas intermináveis, pelas discussões político-filosóficas, ou simplesmente pela companhia no almoço. Agradeço ainda aos amigos Bruno, Sérgio e Victor que, cada um com seu motivo, não foram capazes de concluir este curso. E a todos, da graduação ou mestrado, que pude conhecer durante este período, muito obrigado.

Como não poderia deixar de ser, agradeço a minha família. Apesar de todo convívio problemático dos últimos tempos, o lar é o único lugar para onde realmente podemos voltar, não é mesmo? Se para tudo há um significado, então o destes anos está sendo tornar-me mentalmente forte. Coincidentemente, esta é uma característica essencial para suportar a pressão psicológica de algo como um mestrado.

Vai ter com a formiga, ó preguiçoso, considera os seus caminhos, e sê sábio; a qual, não tendo chefe, nem superintendente, nem governador, no verão faz a provisão do seu mantimento, e ajunta o seu alimento no tempo da ceifa.

Provérbios 6:6-8

RESUMO

SILVA JR, Luneque Del Rio de Souza e. *Roteamento em redes embutidas utilizando otimização por colônia de formigas.* 2011. 114f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2011.

Redes embutidas (NoC, *Network-on-Chip*) vêm sendo adotadas como uma solução interessante para o projeto de infraestruturas de comunicação em sistemas embutidos (SoC, *System-on-Chip*). Estas redes são em geral parametrizadas, podendo assim ser utilizadas em vários projetos de SoCs, cada qual com diferentes quantidades de núcleos. NoCs permitem uma escalabilidade dos sistemas, ao mesmo tempo que balanceiam a comunicação entre núcleos. Projetos baseados em NoC visam a implementação de uma aplicação específica. Neste contexto, ferramentas de auxílio de projeto são essenciais. Estas ferramentas são projetadas para, a partir de uma descrição simples da aplicação, realizar sucessivos processos de otimização que irão modelar as várias características do sistema. Estes algoritmos de otimização são necessários para que a rede atenda a um conjunto de restrições, como área, consumo de energia e tempo de execução. Dentre estas etapas, pode ser incluído o roteamento estático. As rotas através da rede por onde os núcleos irão se comunicar são otimizadas, de forma a minimizar o tempo de comunicação e os atrasos na transmissão de pacotes ocasionados por congestionamentos nas chaves que compõem a NoC. Nesta dissertação, foi utilizada a otimização por colônia de formigas no cálculo dos percursos. Esta é uma meta-heurística interessante para a solução de problemas de busca em grafos, inspirada no comportamento de formigas reais. Para os algoritmos propostos, múltiplas colônias são utilizadas, cada uma encarregada pela otimização do percurso de uma mensagem. Os diferentes testes realizados mostram o roteamento baseado no *Elitist Ant System* obtendo resultados superiores a outros algoritmos de roteamento.

Palavras-chave: Redes Embutidas. Algoritmos de roteamento. Otimização por Colônia de Formigas.

ABSTRACT

Network-on-Chip (NoC) have been adopted as an interesting option in design of communication infrastructures for embedded systems or SoCs (System-on-Chip). These networks are structures that can be used in multiple SoC projects, each with different numbers of cores. NoCs provide a scalable system and balances the communication between cores. NoC-based projects aimed at implementing a specific application. In this context, design aid tools are essential. These tools are designed to, from a simple description of the application, perform successive optimization processes that will shape the various features of the system. These optimization algorithms are needed to meet a set of constraints, such as area, power consumption and execution time. Among these steps, can be included static routing. The routes through the network are optimized by minimizing the communication time and the packet transmission delays caused by congestion on the switches. In this dissertation, the ant colony optimization was used in the search of routes. This is a meta-heuristic inspired by ant behavior and is widely used in solving combinatorial optimization problems, such as searching in graphs . Multiple colonies was used in the proposed algorithms, each one responsible for the optimization of the path of one message. The simulations show the EAS-based routing achieving superior results in compare to other routing algorithms.

Keywords: Network-on-Chip. Routing algorithms. Ant Colony Optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

INTRODUÇÃO

NO decorrer da história da indústria de componentes eletrônicos, sempre houve uma busca natural por dispositivos cada vez melhores. Esta melhoria pode ser descrita por diversos fatores, tais como o aumento do poder computacional, a diminuição do consumo, a redução do espaço e a redução do custo total de projeto. Por muito tempo, esta melhoria foi obtida graças à capacidade do aumento da integração. Em outras palavras, não apenas investiu-se no projeto de dispositivos com mais recursos, mas também em mudanças no processo de fabricação de semicondutores, de forma a conseguir quantidades cada vez maiores de transistores em áreas cada vez menores.

Os microprocessadores podem ser tomados como exemplo característico do aumento da qualidade dos dispositivos. A disputa pelo mercado por parte dos fabricantes culminou, na segunda metade do século XX, na popularização mundial do uso de microcomputadores. Nota-se desde então a produção destes dispositivos com capacidade computacional cada vez maior a um custo cada vez menor. Esta grande capacidade possibilitou que computadores passassem a ser utilizados para diversos fins, como jogos, a execução de vídeo e áudio digitais, comunicações pela internet, etc. Isto vai muito além dos cálculos científicos e da manipulação de bancos de dados, primeiras aplicações de sistemas computacionais.

A cultura emergente da utilização de computadores obviamente veio a influenciar a indústria de dispositivos eletrônicos. Um efeito marcante é a ideia de múltiplas aplicações, presente em microcomputadores, migrando para dispositivos móveis. Outrora, aparelhos portáteis (como reprodutores de áudio, telefones celulares, câmeras digitais e consoles de jogos) apresentavam uma única utilidade. Contudo, cada vez mais são fabricados dispositivos com diversas aplicações, tais como os *smartphones* e os *tablets*. Pode-se dizer que esta capacidade de aglutinação de diferentes operações é decorrente, dentre outros fatores, da adoção de projetos baseados em sistemas embutidos, ou SoC (*System-on-Chip*).

Um sistema embutido é um circuito integrado que, mesmo sendo projetado para um dispositivo específico, é composto internamente por um sistema computacional completo. Em geral um SoC contém, dentro de um mesmo encapsulamento, processadores, memórias, um

sistema de entrada e saída de dados e dispositivos de aplicação específica. Esta estrutura em blocos segue uma metodologia de projeto baseada em núcleos de propriedade intelectual, ou blocos IP (*Intellectual Property*). Componentes projetados para um projeto específico podem ser reutilizados em outros SoCs, reduzindo assim o tempo de projeto. Assim, sob um ponto de vista extremamente simplificado, para aumentar a quantidade de tarefas executadas por um SoC, bastaria acrescentar mais núcleos IPs com diferentes funcionalidades.

O crescimento da escala de SoCs levanta novos desafios de projeto. Dentre eles, está a comunicação entre IPs. Os blocos de um SoC são interligados por uma infraestrutura de comunicação, como barramentos ou canais ponto-a-ponto. Contudo, cada um destes modelos apresentam suas limitações. Barramentos compartilhados resultam em um elevado atraso na transmissão de sinais, quanto maior a quantidade de blocos a ele conectados. Isto não ocorre com canais ponto-a-ponto; estes contudo, necessitam ser projetados novamente para cada nova versão do SoC de interesse. Faz-se então necessário a criação de uma estrutura escalável como os barramentos e rápidos como os canais ponto-a-ponto. Um modelo que engloba estas duas características são as redes embutidas (BENINI; MICHELI, 2002) ou NoCs (*Network-on-Chip*).

Na arquitetura de uma NoC, chaves ou *switches* são interligados entre si por canais ponto-a-ponto, descrevendo assim uma *topologia de rede*. As chaves estão conectadas também aos núcleos IP que constituem o sistema, chamados então de *recursos*. Chaves trocam informações entre si sob a forma de mensagens e pacotes. Assim um sinal, originalmente gerado por um recurso, é dividido em pequenas partes e enviado pela rede; estes pacotes são organizados na chave de destino e então transmitido para outro recurso. Este é um funcionamento semelhante aos de redes de computadores.

Sistemas baseados em NoCs possuem sua modelagem em parte simplificada, pois toda a infra-estrutura de comunicação pode ser importada ao projeto como um único bloco IP parametrizável. Contudo, diversas serão as formas de se conectar recursos à rede, de modo que a aplicação de interesse seja realizada. Para auxiliar o projetista, são utilizadas ferramentas computacionais de auxílio a projeto, ou EDAs (*Electronic Design Automation*). O objetivo das EDAs é otimizar etapas intermediárias do projeto de SoCs e NoCs, de forma a obter uma implementação final mais eficiente (EDWARDS *et al.*, 1997).

Em geral, NoCs são soluções adotadas para o desenvolvimento de dispositivos responsáveis pela execução de aplicações específicas. Esta aplicação pode ser definida como um conjunto de tarefas que possuem uma interdependência entre si. A forma como estas tarefas estão organizadas pode ser ilustrada por um *grafo de tarefas*. Um grafo é uma estrutura composta

por *vértices* e *arcos*. Em um grafo característico de aplicação, vértices representam tarefas, enquanto que arcos descrevem a dependência de dados entre as tarefas.

A ferramenta de EDA deve ser capaz de usar as informações fornecidas pelo grafo da aplicação (em um alto nível de abstração) e, através de sucessivas etapas de otimização, alcançar uma solução para implementação que atenda as especificações de projeto. Dentre as otimizações que podem ser realizadas entre a especificação do sistema e a implementação final, podem ser citadas a alocação de tarefas e o mapeamento de IPs. A alocação diz respeito a seleção de quais núcleos IP irão realizar cada tarefa (dentro de um conjunto com um grande número de IPs, cada um com diferentes características). Já o mapeamento define onde cada IP, após a etapa de alocação, estará espacialmente localizado.

O presente trabalho está baseado em outra etapa de otimização: o roteamento estático de pacotes. Diferentes opções de mapeamento podem impactar de formas diferentes na comunicação entre recursos da NoC. É de se desejar que tarefas com uma comunicação mais intensa sejam mapeadas em recursos fisicamente mais próximos. A otimização das distâncias visivelmente melhora a comunicação entre núcleos. Ainda assim, atrasos na comunicação podem ocorrer em situações de congestionamento, isto é, vários pacotes precisando ser transmitidos por uma mesma chave. Se o roteamento adotado no projeto da NoC for determinístico, o percurso de um pacote (desde a chave de origem até a de destino) não irá considerar a utilização das chaves intermediárias. Se estas chaves intermediárias estiverem sob um intenso tráfego, então o pacote só poderá ser transmitido após o fim do congestionamento. Isto ocorre mesmo que outras chaves, não selecionadas pelo roteamento, estejam livres para transmissão.

A fim de superar o problema de congestionamento, este trabalho propõe uma etapa de otimização de rotas, ou um roteamento adaptativo e estático. Neste tipo de roteamento, um modelo de rede fornece os padrões de comunicação necessários para a execução da aplicação. O cálculo das rotas é realizado por um algoritmo de otimização de forma a minimizar o tempo de comunicação. A busca é sempre por um caminho mínimo entre origem e destino; se as chaves intermediárias deste caminho estiverem em uso, o algoritmo deve ser capaz de encontrar um outro percurso, de forma que os efeitos de contenção não afetem a transmissão. Como a NoC em questão é projetada para uma única aplicação, então as rotas calculadas (de todos os pacotes que serão transmitidos) poderão ser codificadas em uma tabela em memória, ou *lookup table*, que fará parte da solução para implementação (DUATO; YALAMANCHILI; LIONEL, 2002).

Diversos algoritmos de otimização podem ser empregados para a busca de percursos. O método selecionado para este trabalho é a *otimização por colônia de formigas* (DORIGO;

MANIEZZO; COLORNI, 1996). Este é um exemplo de inteligência de enxame, onde um grupo de indivíduos operam em conjunto para a busca de uma solução de um determinado problema. Formigas naturais descrevem um complexo comportamento social. Uma característica interessante é o fato de uma formiga, isoladamente, ser extremamente limitada. Contudo, uma colônia formada por uma grande quantidade de formigas é capaz de tarefas difíceis, como por exemplo a busca de um caminho mínimo entre o formigueiro e uma fonte de alimento. Este comportamento inspirou o desenvolvimento de modelos computacionais para a busca de caminhos mínimos em problemas descritos por grafos.

O objetivo desta dissertação é desenvolver um algoritmo baseado na otimização por colônia de formigas para a otimização de rotas em NoCs. Este é um processo que, bem como a alocação de tarefas e o mapeamento de IPs, otimiza uma determinada característica da rede para uma posterior etapa de implementação. A otimização tem por objetivo minimizar a latência, isto é, o tempo total de transmissão de pacotes das chaves de origem para suas respectivas chaves de destino. É considerado que a aplicação (e a rede como um todo) já passou pelas etapas de alocação e mapeamento. Assim, o modelo proposto seleciona as informações recebidas (do grafo de tarefas) as utiliza para o roteamento de pacotes. A principal contribuição deste projeto é a redução do atraso na transmissão ocasionado pelo congestionamento da rede. O roteamento inspirado em ACO é capaz de identificar as regiões onde existe congestionamento e busca percursos onde seus efeitos são reduzidos. Com a redução dos atrasos de comunicação, o tempo total de execução da aplicação também é reduzido. Esta dissertação está organizada em sete capítulos, cujos conteúdos são resumidamente descritos a seguir.

O Capítulo 1 apresenta a evolução dos sistemas de alta escala de integração, que resultou no surgimento de soluções baseadas em SoCs. Os conceitos gerais sobre redes embutidas são abordados, incluindo a metodologia de projeto baseada em núcleos IP. Diferentes infra-estruturas de comunicação empregadas por SoCs são citadas. Neste contexto, as redes embutidas aparecem como opções às arquiteturas baseadas em barramentos. As principais características das NoCs, como topologias, técnicas de chaveamento e roteamento, são descritas detalhadamente.

No Capítulo 2 é feito um levantamento bibliográfico sobre os principais métodos de roteamento empregados em NoCs e em redes de comunicação como um todo. É dada uma maior ênfase em dois destes métodos: o algoritmo XY e o modelo *Odd-Even*. O primeiro trata-se de um roteamento determinístico, enquanto o segundo é um parcialmente adaptativo. Por este fato, estes dois métodos são utilizados na comparação de resultados.

O Capítulo 3 apresenta uma introdução teórica sobre a otimização por colônia de formigas. É abordado o comportamento explorador de formigas reais. Este eventualmente serviu de inspiração para o desenvolvimento de algoritmos de otimização, enriquecendo assim a lista de métodos baseados em inteligência de enxame. É descrito o algoritmo mais conhecido, *Ant System*, desenvolvido originalmente para a solução do problema do caixeiro viajante e posteriormente adotado na solução de outros diversos problemas de otimização combinatória. Também são detalhadas as principais características que tornam o ACO uma meta-heurística.

O Capítulo 4 apresenta o modelo de roteamento para NoCs inspirado na otimização por colônia de formigas. É exposto o modo geral como a rede é simulada, sendo este modo similar para diferentes métodos de roteamento. As principais características da rede são especificadas, como a topologia e métodos de chaveamento adotadas. Também é descrito o modelo de latência adotado. A partir do modelo de rede, é apresentado o algoritmo REAS, um roteamento baseado no *elitist ant system*. Este algoritmo otimiza as rotas com base nos valores de latência: percursos que implicam em menores tempos te comunicação são os melhores. Um outro algoritmo também é proposto, chamado RACS. Este segundo é, por sua vez, um roteamento baseado no *ant colony system*.

No Capítulo 5 é feita uma avaliação de desempenho dos algoritmos de roteamento propostos. São apresentados os resultados de um grande número simulações, onde são comparados os valores de latência obtidos pelos modelos propostos, REAS e RACS, com a latência em redes utilizando roteamentos baseados no algoritmo XY e no modelo *Odd-Even*. Para a realização destes testes, são usados padrões sintéticos de geração de tráfego. Cada um dos seis padrões de tráfego são descritos. Com os resultados obtidos, é feita uma análise do comportamento observado para cada algoritmo sobre o efeito dos diferentes padrões de geração de tráfego.

O Capítulo 6 apresenta mais testes realizados como os roteamentos baseados em ACO. Desta vez, contudo, os algoritmos de roteamento são empregados em modelos que simulam o funcionamento da rede com aplicações. Assim, este capítulo mostra como aplicações são descritas como *grafos de tarefas* e utilizadas em sistemas multiprocessados. Também é apresentada a forma com que as informações da aplicação e do modelo de rede são utilizados, de forma que o roteamento possa ser realizado.

O Capítulo 7 conclui esta dissertação, apresentando as principais conclusões obtidas com o desenvolvimento do presente trabalho. São apresentadas também possíveis melhorias para os algoritmos propostos e as direções para futuros estudos envolvendo o roteamento em NoCs.

Capítulo 1

REDES EMBUTIDAS

A CRIAÇÃO do *circuito integrado* (CI) na década de 1950 representou um marco na indústria de dispositivos eletrônicos: a possibilidade do projeto de vários componentes em um mesmo encapsulamento. Desde então, viu-se uma grande evolução na tecnologia de fabricação de CIs, com cada nova geração contendo um número maior de transistores integrados. Assim, temos as primeiras gerações de CIs, implementando portas lógicas, fabricados com tecnologia de SSI (*Small Scale Integration*); na etapa seguinte, componentes mais complexos, como *chips* de calculadoras e memórias, são fabricados com tecnologia LSI (*Large Scale Integration*); posteriormente, a mudança para tecnologia VLSI (*Very Large Scale Integration*) proporcionou o surgimento dos microprocessadores de alta velocidade. Avanços tecnológicos têm permitido a fabricação de CIs cada vez mais complexos; o termo ULSI (*Ultra Large Scale Integration*) é então proposto para caracterizar chips com mais de 1 milhão de transistores integrados. Em LSI, um CI é um componente de um módulo de sistema; em VLSI, é um componente no nível de sistema; já em ULSI, é um sistema completo (BJERREGAARD; MAHADEVAN, 2006). Este tipo de componente, com todo um sistema em um mesmo encapsulamento, é conhecido como *sistema embutido* ou SoC (*System-on-Chip*).

A crescente utilização de SoCs é notória, dado o grande número de dispositivos de multimídia portáteis, como telefones celulares, reprodutores de som e vídeo, câmeras fotográficas digitais e consoles de jogos. Em geral, SoCs são encontrados em dispositivos eletrônicos que possuem uso específico e que necessitam de algum tipo de processamento. Há um contraste, pois, com os microcomputadores, que possuem uma arquitetura baseada em microprocessadores de propósito geral.

Internamente, um SoC é composto de um conjunto de módulos de *hardware* interconectados por uma infraestrutura de comunicação, como conexões ponto-a-ponto e barramentos. Contudo, estes dois tipos de comunicação podem vir a ser ineficientes em sistemas de maior

escala (MORENO, 2010). Uma solução que tem se mostrado promissora é o uso de *redes embutidas* ou NoCs (*Network-on-Chip*). Trata-se de um modelo de comunicação inspirado em redes de computadores. As principais vantagens de projetos baseados em NoCs são o paralelismo na comunicação, a escalabilidade e a reusabilidade.

Este capítulo apresenta uma visão geral sobre NoCs. A Seção 1.1 mostra o conceito de sistemas embutidos. Na Seção 1.2, são apresentadas as principais infraestruturas de SoCs. Na Seção 1.3, é feito um detalhamento sobre redes embutidas, apresentando suas principais características. Na Seção 1.4 são abordadas as principais etapas do projeto de NoCs.

1.1 Sistemas Embutidos

Um sistema embutido é um sistema computacional, geralmente de uso específico, construído todo no mesmo encapsulamento de um circuito integrado. Um SoC é composto tipicamente por um microprocessador, memória e dispositivos de entrada e saída. Pode conter ainda um ou mais componentes dedicados a uma dada função. Todos estes componentes trocam informações entre si através de uma estrutura de interconexão, de forma que o SoC seja capaz de executar uma aplicação. A organização de um SoC genérico pode ser vista na Figura 1.

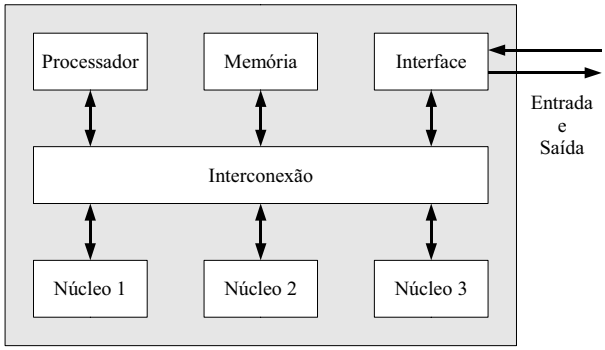


Figura 1: Arquitetura genérica de um SoC.

O projeto de SoC normalmente é baseado na utilização de blocos de propriedade intelectual, ou IPs (*Intellectual Property*). Um IP é todo componente que pode ser utilizado em um projeto de circuito integrado, como processadores, memórias, unidades lógicas e dispositivos de uso específico. A ideia central no uso de IPs é a reutilização: um bloco desenvolvido para um projeto pode ser utilizado em projetos futuros com um mínimo de adaptação. Desta forma, há uma diminuição no tempo total de desenvolvimento e no custo do projeto do sistema.

Desenvolvedores de SoC podem projetar seus próprios IPs ou adquiri-los de terceiros, sob a forma de *núcleos de software* ou *núcleos de hardware*. Em projetos baseados em uma

plataforma de prototipagem, núcleos de *software* são componentes sintetizáveis, disponibilizados sob a forma de código-fonte em uma *linguagem de descrição de hardware* ou de uma *netlist* no nível de portas lógicas. Os IPs definidos por *netlists* também são conhecidos por *núcleos de firmware*. Se o IP em questão for um projeto em linguagem de descrição de *hardware*, há a possibilidade de edição do código antes da etapa de síntese, possibilitando a adaptação do IP para o projeto de interesse. Por outro lado, IPs sob a forma de *netlists* não podem ser editados, cabendo ao projetista adaptar o sistema às especificações dos IPs. É possível ainda o uso de núcleos de *hardware*: IPs descritos diretamente em sua estrutura física (*layout* de transistores). Este formato é interessante por permitir ao projetista ter uma estimativa próxima a realidade do desempenho em relação ao tempo e à área ocupada pelo componente. Contudo, é uma metodologia extremamente específica no que diz respeito à fabricação de CIs: os IPs devem obrigatoriamente obedecer as regras de fabricação de uma dada indústria de semicondutores. Se duas fábricas possuem processos de produção diferentes, então núcleos de *hardware* projetados para uma não serão compatíveis com a outra.

O comércio de IPs disponibilizados como núcleos de *firmware* e *hardware* é interessante, uma vez que componentes complexos podem ser distribuídos sem que haja a possibilidade de engenharia reversa. É uma metodologia semelhante, de certa forma, ao uso de CIs em placas de circuito impresso: pode-se escolher como os CIs estarão conectados e como estes funcionarão, mas não se pode alterar seu funcionamento interno.

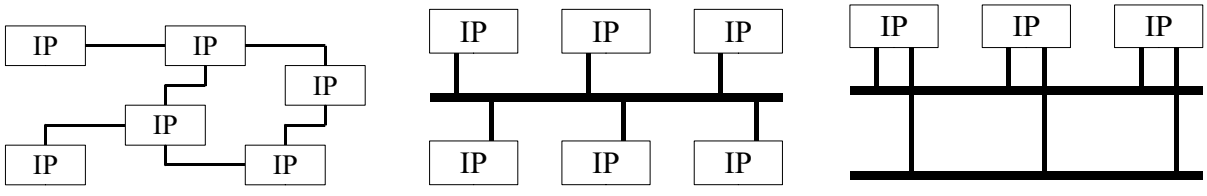
O SoC apresentado na Figura 1 possui um único elemento processador que se comunica com um conjunto de IPs (núcleos 1, 2 e 3). Contudo, dada a necessidade de sistemas cada vez mais rápidos, é comum o uso de inúmeros elementos processadores quando há a possibilidade de execução de partes de uma aplicação em paralelo. Este tipo de sistema, constituído por mais de um processador, é chamado de sistema embutido multiprocessado ou MPSoC (*Multi-Processor System-on-Chip*).

1.2 Comunicação intra-chip

Os componentes de um SoC precisam transmitir informações entre si durante a execução da aplicação. É responsabilidade da infraestrutura de comunicação garantir que os diversos conjuntos de dados sejam corretamente entregues do componente de origem para o alvo de destino. Além disso, a comunicação intra-chip precisa atender os requisitos de latência (tempo de transmissão da dados) e de largura de banda (quantidade de dados por unidade de tempo) de forma a alcançar um desempenho satisfatório para uma dada aplicação (PASRICHA; DUTT, 2008).

A forma mais simples de se conceber uma comunicação entre blocos de um sistema é através de comunicação ponto-a-ponto. Dada a aplicação a ser executada pelo SoC, um sistema de canais dedicados são construídos interligando os blocos de interesse. Por se tratar de canais dedicados, isto é, que só transmite informações do ponto de origem para o de destino, há um grande ganho de desempenho. Como a comunicação em um canal ocorre de forma independente dos demais, há a possibilidade da transmissão paralela de informação e com baixa latência. É uma estrutura que possui um baixo consumo de energia e que permite altas frequências de operação (ZEFERINO; SUSIN, 2003). Contudo, é um modelo de comunicação que possui um maior custo de projeto, devido a sua natureza específica. Por este mesmo motivo, perde-se a capacidade de reutilização, sempre desejável no desenvolvimento de sistemas embutidos.

Uma estrutura que permite reutilização, e consequentemente maior facilidade de uso em projetos, são os *barramentos compartilhados*. Um barramento consiste de um conjunto de fios paralelos onde vários componentes são conectados. Apenas um componente possui o controle dos fios para transmissão de informação em um determinado instante de tempo. Isto limita o paralelismo e o desempenho total do sistema, tornando o uso de barramentos compartilhados simples inviável em MPSoCs, onde se espera a comunicação entre um grande número de componentes. Ainda assim, devido ao baixo custo, arquiteturas baseadas em barramentos são bastante usadas em projetos de SoCs. Para contornar suas limitações, vários SoCs utilizam não apenas um, mas um conjunto de barramentos. Um exemplo simples são as arquiteturas com *barramentos múltiplos*, com cada núcleo podendo acessar mais de um barramento (CHEN; SHEU, 1991) . Desta forma, há uma maior disponibilidade de canais de comunicação. Exemplos de comunicação ponto-a-ponto, por barramento simples e por múltiplos barramentos são exibidos na Figura 2.



(a) Ponto-a-ponto. (b) Barramento compartilhado. (c) Múltiplos barramentos.

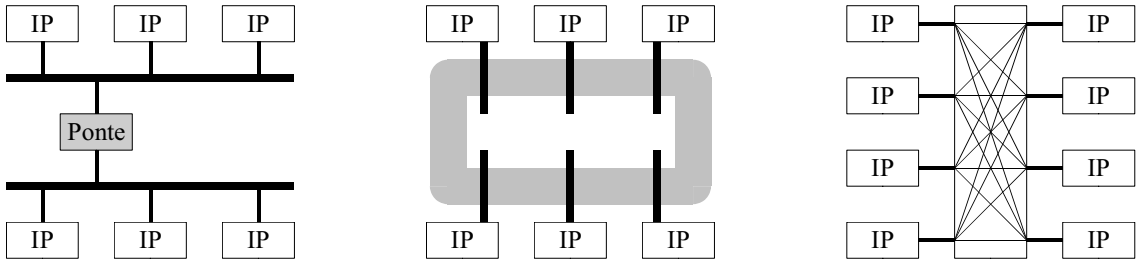
Figura 2: Comunicação intra-chip.

Outras arquiteturas de comunicação usadas em SoCs são apresentadas na figura 3. O uso de *barramentos em hierarquia* permite separar a comunicação entre conjuntos de núcleos. Constitui-se assim um SoC com vários subsistemas, cada qual com seu próprio barramento

- Figura 3(a). A comunicação interna entre componentes de um subsistema pode ocorrer de forma independente dos demais. Quando há necessidade de comunicação entre barramentos, esta é intermediada por um circuito ponte (*bridge*). Esta arquitetura é interessante por permitir que barramentos possuam frequências de operação diferentes - componentes mais rápidos são conectados em barramento com maior nível hierárquico, enquanto que componentes mais lentos podem usar a comunicação mais lenta de um barramento de nível inferior.

A Figura 3(b) ilustra um modelo de *barramento em anel*, semelhante ao utilizado pelo processador *Cell* (KAHLE *et al.*, 2005). O barramento de interconexão consiste de um conjunto de canais unidirecionais operando em *pipeline*. Permite uma alta frequência de operação e alta taxa de transferência de dados entre os componentes interligados.

Soluções baseadas em uma *matriz de barramentos* permitem conectar dois grupos de componentes, como um conjunto de processadores a um conjunto de memórias. Cada elemento da esquerda pode se comunicar com qualquer elemento da direita. A estrutura responsável pela comunicação (Figura 3(c)) também é conhecida como *chave crossbar*, e é composta por fios que conectam cada entrada a esquerda a todas as saídas a direita. Este tipo de arquitetura é uma combinação de barramentos compartilhados e interconexões ponto-a-ponto (PASRICHA; DUTT, 2008) .



(a) Barramento hierárquico. (b) Barramento em anel. (c) Matriz de barramentos.

Figura 3: Comunicação intra-chip.

1.3 Redes Embutidas

Projetos de SoC de baixa complexidade, ou que necessitam de um número reduzido de IPs, podem perfeitamente utilizar um sistema de comunicação intra-chip baseado nas arquiteturas apresentadas na Seção 1.2. Contudo, em sistemas de maior escala, tais arquiteturas não se mostram tão eficientes. Comunicações ponto-a-ponto, embora maximizem o desempenho do SoC como um todo, elevam o custo do projeto devido a sua complexidade. É uma solução que

apresenta pouca escalabilidade, reusabilidade e flexibilidade por necessitar um número excessivo de fios na interligação de um SoC com elevada quantidade de núcleos. A alternativa de comunicação baseada em barramentos falha por restringir comunicações paralelas. Há ainda problemas elétricos decorrentes das capacitâncias parasitas inerentes ao uso de fios longos. Em barramentos com hierarquia, há uma perda de flexibilidade devido a particularidades de cada projeto. Também o uso de pontes interligando subsistemas de diferentes níveis hierárquicos pode resultar em um atraso de comunicação, sem contar a necessidade de um *hardware* adicional. A solução por barramento em anel pode não ser interessante para SoCs de baixo desempenho, devido a sua alta complexidade e custo de implementação. Em matrizes de barramentos, há a limitação quanto a flexibilidade. Há de se considerar também o fato do excessivo número de fios possuir um elevado consumo de energia e necessitar de uma grande área de *hardware*.

As limitações existentes em cada estrutura motivam a pesquisa por outras formas de comunicação em SoCs, de forma a atender requisitos como desempenho, escalabilidade e consumo de energia. Um modelo que tem despertado grande interesse entre desenvolvedores de SoC são as redes embutidas, redes de interconexão ou NoCs (*Network-on-Chip*) (BENINI; MICHELI, 2002). Este é um paradigma de comunicação inspirado nos conceitos de redes de comunicação. Da mesma forma que computadores em uma rede podem constituir um *sistema distribuído* para executarem uma tarefa comum em conjunto, pode-se imaginar também os processadores de um MPSoC organizados como uma rede, executando assim uma dada aplicação. Embora a analogia entre uma NoC e um sistema distribuído seja de fácil compreensão, SoC em questão pode não ser necessariamente um MPSoC - qualquer sistema composto por inúmeros núcleos pode possuir uma rede de interconexão.

1.3.1 Elementos de uma NoC

Em uma rede embutida, tal como em outros tipos de SoC, um conjunto de componentes compartilham dados através de uma infraestrutura de comunicação. Contudo, uma NoC apresenta várias características que a diferencia das estruturas apresentadas na Seção 1.2. A primeira delas é o fato da rede de interconexão ser constituída por um conjunto de *chaves*, interligadas entre si e ao conjunto de núcleos IP por meio de canais de comunicação ponto-a-ponto. Outra diferença está na forma como a informação é transportada por entre as chaves. Em uma NoC, dados são transmitidos sob a forma de *mensagens*, as quais podem ser divididas em unidades menores chamadas *pacotes*. Os elementos básicos de uma NoC são expostos na Figura 4.

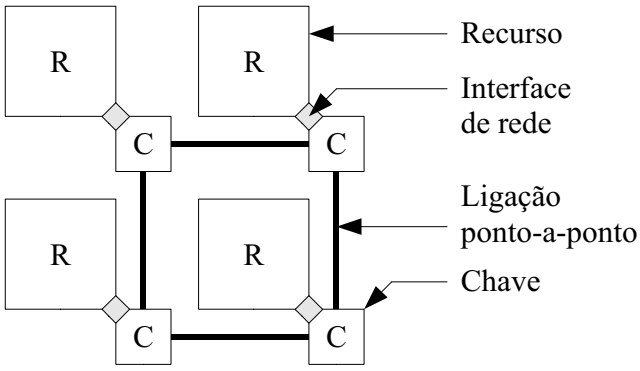


Figura 4: Componentes fundamentais de uma NoC em uma rede com topologia malha 2x2.

Em redes embutidas, os elementos de processamento são comumente chamados de *recurso*. Um recurso é qualquer componente capaz de manipular informação, seja ele um processador, uma memória, uma interface de comunicação ou um dispositivo de uso específico. É um meio físico onde IPs podem ser implementados, gerando assim um núcleo da rede (*core*).

A chave, ou *switch*, é o elemento da rede responsável por encaminhar a mensagem pela rede. Uma chave possui um conjunto de portas de entrada e saída para a comunicação com outras chaves; possui ainda uma porta de comunicação local com um recurso, também de entrada e saída. A Figura 5 ilustra a arquitetura interna de uma chave. Em geral, uma chave é constituída basicamente de uma estrutura *crossbar* e uma lógica de controle de roteamento e arbitragem. A estrutura *crossbar* é composta de um conjunto de ligações que permitem a comunicação das portas de entrada com as de saída, incluindo a porta de acesso local do recurso. A lógica de roteamento define por qual porta de saída o pacote será transmitido, baseando-se em um algoritmo de roteamento. Na situação de dois pacotes disputando a mesma porta de saída de uma chave, o controle de arbitragem julgará qual terá prioridade de envio. A chave pode conter uma memória de armazenamento temporário de dados (*buffer*) associada a cada porta. Estes *buffers* são organizados em forma de fila, e funcionam em uma estrutura FIFO (*First In - First Out*). Cada porta possui ainda um controlador de enlace (*link controller*) responsável pela implementação do protocolo físico de comunicação. Estes controladores regulam o tráfego da informação que entra e sai da chave (ZEFERINO, 2003).

A conexão entre duas chaves ou entre uma chave e um recurso é feita por meio de canais de comunicação ponto-a-ponto ou enlaces (*links*). Estes enlaces são geralmente *full-duplex*, compostos por dois canais unidirecionais opostos. Isto permite que uma porta transmita e receba informações simultaneamente. Como o enlace é composto apenas por um conjunto de fios que ligam duas chaves, o desempenho local da rede não é afetado quando se aumenta a escala da rede - o que não ocorre com barramentos, que sofrem com problemas de natureza

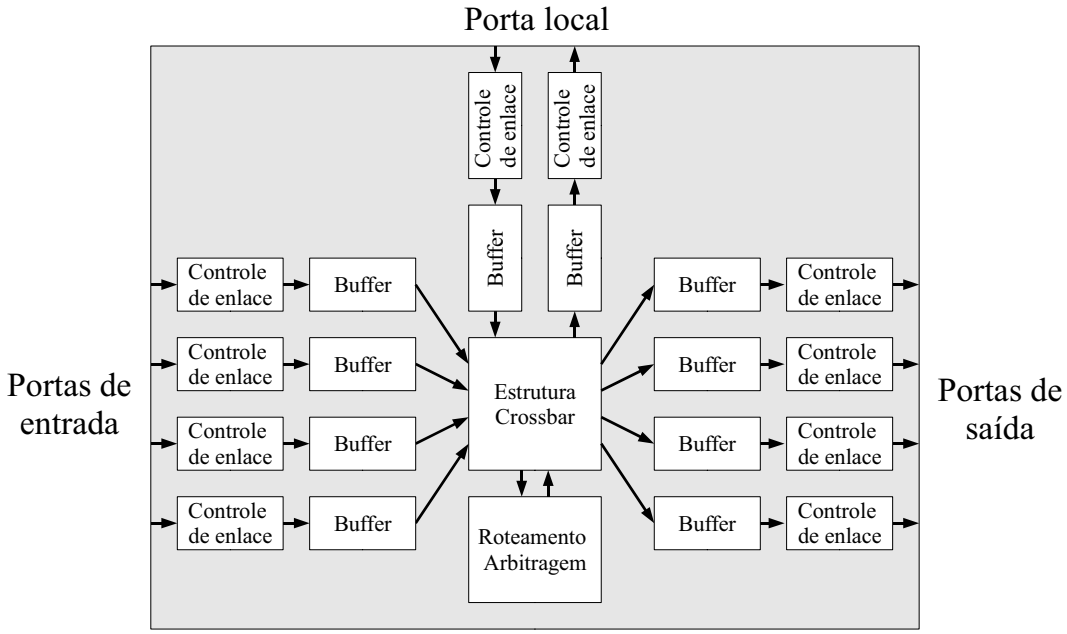


Figura 5: Arquitetura interna genérica de uma chave.

elétrica ao se usar fios de maior extensão.

A informação em uma NoC é transmitida entre as chaves sob a forma de uma mensagem. Uma mensagem é uma estrutura de dados composta pela informação processada pelos núcleos acrescida de informações referentes à comunicação. O recurso que processa a informação que será transmitida pode não conhecer o protocolo de comunicação usado na rede. Isto é de se esperar, considerando que o projeto de NoCs é baseado na reutilização de IPs. É necessário, pois, que a informação fornecida pelo recurso de origem seja montada em mensagens para a transmissão na rede. Posteriormente, estas mensagens precisarão ser reorganizadas em seu formato original para que sejam compreendidas pelo recurso de destino. Cada recurso deve então possuir uma interface de rede ou RNI (*Resource Network Interface*). A interface de rede é um componente que intermediará a comunicação entre os recursos e a rede de interconexão.

A mensagem disponibilizada pela interface de rede é composta de três partes partes principais, sendo elas:

- Cabeçalho (header): Informação sobre o endereço de origem e destino da mensagem na rede. São dados de roteamento e controle utilizados pelas chaves para propagar a mensagem até o núcleo de destino.

- Carga útil (payload): É a informação original processada pelos núcleos IP da rede. A interface de rede adiciona o cabeçalho e o terminador a mensagem, formando um envelope ao redor daq carga útil.

- Terminador (trailler): Inclui informações usadas para a detecção de erros e para a indicação do fim da mensagem.

Uma mensagem, em geral, é transmitida em unidades menores, chamadas pacotes. Cada pacote possui estrutura semelhante a da mensagem - cabeçalho, carga útil e terminador. Um pacote é constituído por uma sequência de palavras, denominadas *flits* (*flow control unit*, ou unidade de controle de fluxo). A largura em *bits* de um *flit* é igual ou múltipla da largura física do enlace. A quantidade física de fios que compõem o enlace define quantos *bits* podem ser transmitidos (ou recebidos) simultaneamente por uma porta. Esta característica física do canal de comunicação é chamada de *phit* (*phisical unit*, ou unidade física).

1.3.2 O modelo OSI de abstração de rede

O estudo de redes embutidas é bastante abrangente, indo desde a modelagem da aplicação de interesse até a implementação física do sistema. Um dos motivos da grande aceitação deste paradigma está na possibilidade de uso de modelos de abstração de redes de comunicação, já amplamente aceitos.

O Modelo de Referência OSI (*Open System Interconnection*, ou Interconexão de Sistemas Aberto) é um padrão que formalmente organiza sistemas de comunicação (ZIMMERMANN, 1980). É um modelo que divide o sistema em grupos menores, chamados *camadas*. Estas camadas são hierárquicas, isto é, cada camada usa suas próprias funções e de camadas inferiores para esconder sua complexidade e transparecer operações para uma camada superior. Uma vantagem do uso de um modelo em camadas é a possibilidade de modificar a implementação de uma camada sem modificar as demais, uma vez que cada camada manipula dados de forma diferente. O modelo OSI é composto de uma pilha de sete camadas de abstração, sendo elas:

- Camada física: Define o meio físico por onde serão transmitidos os *bits*. Abrange características mecânicas e elétricas, como a composição de fios e conexões.

- Camada de enlace: Permite a transferência confiável de dados em *frames* (quadros, ou grupos de *bits*) através da camada física. Estabelece um controle de fluxo e sincronização dos *frames* transmitidos. É responsável por detectar, e possivelmente corrigir erros que possam ocorrer no meio físico.

- Camada de rede: Transmite pacotes desde a chave de origem até a de destino através de um ou mais enlaces. Nesta camada são definidas informações como os endereços lógicos das chaves na rede e o processo de roteamento dos pacotes.

- Camada de transporte: Estabelece uma conexão fim-a-fim entre origem e destino. É responsável pelo controle de fluxo, segmentação e remontagem de pacotes.

- Camada de sessão: Controla a comunicação entre aplicações. Estabelece, gerencia e termina conexões entre aplicações. A unidade de dados é o PDU (*Protocol Data Unit*, ou unidade de protocolo de dados).

- Camada de apresentação: Traduz ou converte os dados entre o formato de aplicação e o formato de rede. Nesta camada, dados podem ser compactados, codificados ou criptografados.

- Camada de aplicação: Corresponde a aplicação (*software*) que está sendo executado e que transmite as informações pela rede.

Uma rede de interconexão implementa, geralmente, apenas as camadas física, enlace de dados e de rede. As camadas de transporte e sessão estão relacionadas com a interface de rede, enquanto que as camadas de apresentação e aplicação são definidas pelos recursos que efetivamente executam a aplicação. A relação entre os componentes de uma NoC e as camadas do modelo OSI é ilustrada pela Figura 6.

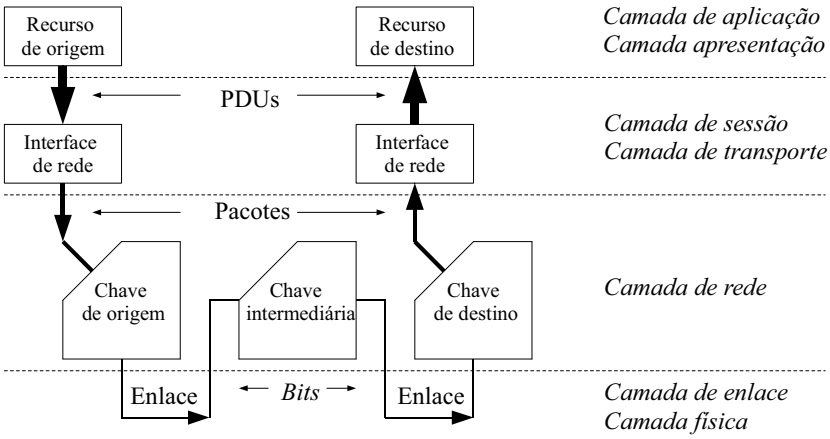


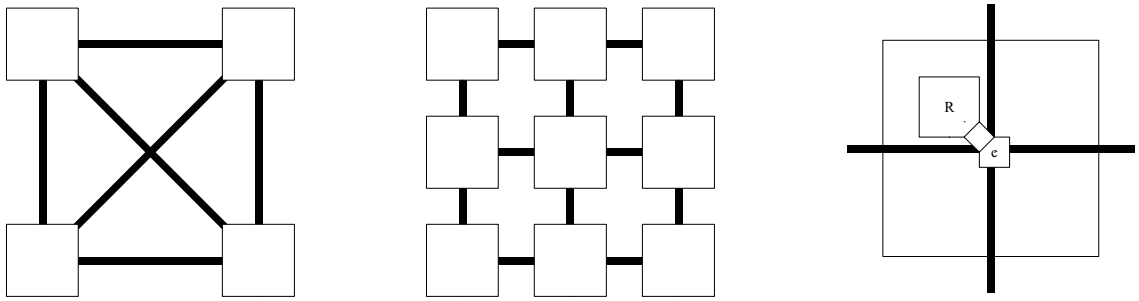
Figura 6: Componentes de uma NoC e as camadas OSI associadas.

1.3.3 Topologias

Redes podem ser caracterizadas por sua *topologia*, a forma que suas chaves estão interconectadas. Uma topologia é representada pelo grafo $G(C, E)$, com C representando o conjunto de chaves e E representando o conjunto de canais de comunicação ou enlaces. São geralmente divididas em duas classes de topologias: redes indiretas e redes diretas (ZEFERINO, 2003).

Uma rede direta é caracterizada por cada chave possuir um recurso conectado a sua porta local. Este par chave-recurso é visto como um único elemento na rede, conhecido pelo termo *nó*. Cada nó está conectado a um conjunto de nós vizinhos através de canais ponto-a-ponto. Se um nó necessitar transmitir uma mensagem para um nó não vizinho, esta precisará percorrer um ou mais nós intermediários. Nesta situação, apenas a chave é usada, não havendo nenhuma ação por parte do recurso do nó intermediário.

Uma rede direta ideal é representada por um grafo completamente conectado. Desta forma, cada nó é vizinho de todos os demais. Contudo, uma rede deste tipo apresenta uma grande restrição em sua escalabilidade. Na prática, muitos dos projetos de redes diretas possuem uma implementação ortogonal. Uma rede é dita ortogonal quando seus nós estão dispostos em um espaço n-dimensional e cada enlace produz um deslocamento em uma única dimensão. Um exemplos de topologia de rede direta ortogonal é a malha (*mesh*). A Figura 7 ilustra uma rede direta ideal e uma malha de duas dimensões, bem como a estrutura interna de um nó.



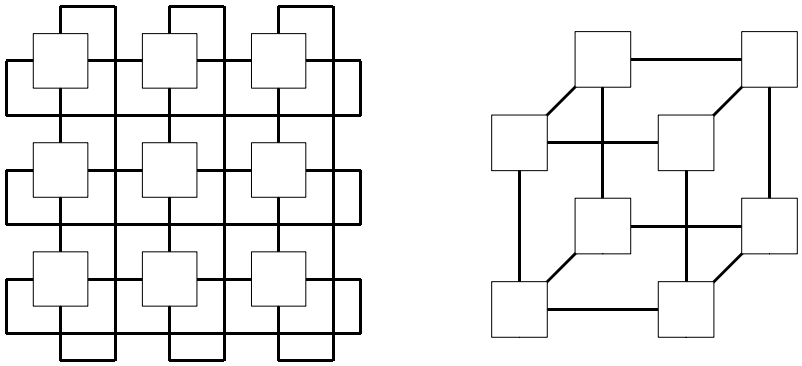
(a) Rede totalmente conectada. (b) Malha 2D. (c) Nó de rede.

Figura 7: Redes diretas.

Outras topologias para redes diretas incluem o toroide e o hipercubo. Estes dois modelos de topologia são ilustrados pela Figura 8.

Enquanto em uma rede direta todas as chaves estão conectadas a um recurso, em uma rede indireta isto não ocorre. Cada chave está conectada às demais, mas apenas algumas possuem conexão com recursos. A topologia da rede é então definida pela forma com que as chaves estão conectadas.

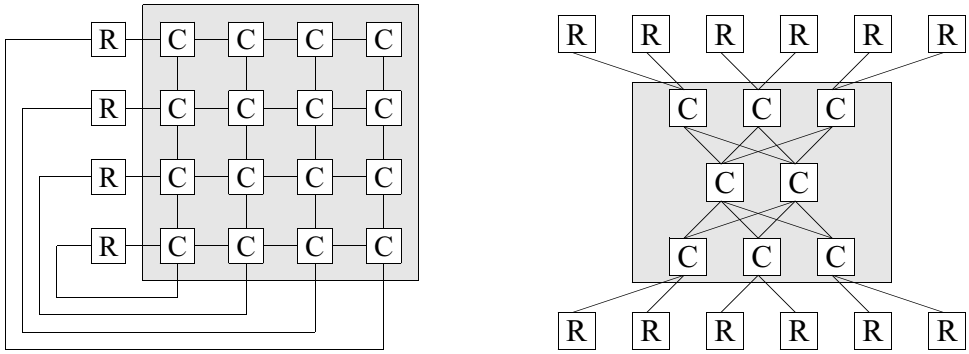
As duas topologias de redes indiretas mais usadas são a rede *crossbar* e a rede multi-estágio. Em uma topologia *crossbar*, um conjunto de chaves é organizado de forma que a rede funcione como uma única chave com N portas de entrada e N portas de saída. Em uma topologia multiestágio, uma mensagem que sai de um recurso atravessa vários estágios de



(a) Toroide. (b) Hipercubo.

Figura 8: Redes diretas.

chaves até alcançar o recurso de destino. Exemplos de redes do tipo *crossbar* e multi-estágio do tipo borboleta são vistos na Figura 9.



(a) *Crossbar*. (b) Borboleta.

Figura 9: Redes indiretas.

As topologias de NoCs ainda podem receber outro tipo de classificação: redes regulares e irregulares. Em uma rede regular, as chaves são todas iguais, e sua quantidade e organização obedece, de certa forma, a um padrão. Em uma rede irregular, não há este padrão, ou apenas parte da rede o segue. Uma rede irregular é, em geral, uma versão personalizada (*custom*) como o intuito de obter um desempenho melhor que o obtido pela solução regular. Tanto redes diretas quanto indiretas podem ser regulares ou irregulares.

1.3.4 Técnicas de Chaveamento

Como visto na Seção 1.3.1, um núcleo produz uma mensagem composta por uma sequência de pacotes. Cada pacote é composto por unidades menores chamadas de *flits*, que são em geral iguais ou múltiplos de um *phit*, a largura em *bits* do canal de comunicação. Um *phit* é a menor

quantidade de informação capaz de ser transferida de uma chave para outra vizinha em um ciclo de *clock*. A forma como os *phits* são transmitidos por entre as chaves da rede é defina pela técnica de chaveamento adotada. As duas formas de chaveamento mais utilizadas são o chaveamento por circuito e o chaveamento por pacote.

No chaveamento por circuito, um caminho é estabelecido entre o nó de origem até o nó de destino atravessando um ou mais nós intermediários. Este caminho ou circuito é estabelecido para a transferência da mensagem e é mantido enquanto esta é transmitida. Enquanto o caminho estiver em uso, qualquer comunicação no canal alocado será recusada. A transmissão é realizada em etapas. Um cabeçalho é enviado desde o nó de origem até o de destino, reservando os canais físicos que serão usados na transmissão. Ao atingir o destino, este envia uma informação de volta até a fonte, indicando o estabelecimento do circuito. Após este instante, a mensagem passa a ser enviada. Quando o último trecho da mensagem avança pela rede, a informação de controle existente no terminador libera a chave para o uso de outros circuitos. A vantagem desta técnica é que, uma vez estabelecido o percurso, a mensagem passa a possuir o controle sobre o caminho. Não há, desta forma, atrasos extras de comunicação decorrentes da espera da liberação de chaves. Uma desvantagem está no fato dos canais de comunicação permanecerem ocupados desde a transmissão do cabeçalho, o que reduz a eficiência da rede como um todo.

No chaveamento por pacotes, a mensagem é dividida em partes menores, com cada pacote contendo informações em um cabeçalho referentes a sua transmissão. Assim, o caminho permanece ocupado apenas enquanto um pacote está sendo transmitido. Contudo, o uso do chaveamento por pacotes necessita, em geral, do uso de *buffers* nas portas de comunicação de cada chave. Isto acarreta em um grande custo de área de *hardware*. Os três métodos mais comuns de chaveamento por pacotes são: *store-and-foward*, *virtual cut-through* e *wormhole*.

No método *store-and-foward*, uma chave deve armazenar completamente um pacote para só então iniciar a transmiti-lo. Isto significa que cada porta da chave deve possuir *buffers* cuja capacidade de armazenamento deve ser de pelo menos de um pacote, semelhante ao apresentado pela Figura 5. Após o armazenamento, o pacote precisa ser transferido para a porta de saída definida pelo algoritmo de roteamento. O pacote só será enviado após a autorização de uma árbitro, que verifica a requisição por saída de todos os pacotes que chegam a chave.

No método *virtual cut through*, cada pacote recebido por uma porta só será armazenado quando a chave de destino não estiver liberada. O cabeçalho do pacote identifica se o canal de saída está livre para uso. Se estiver, o restante do pacote é diretamente transmitido, sem a

necessidade de armazenamento. Há então uma redução da latência da chave. Na pior situação, o pacote inteiro é armazenado nos *buffers*, da mesma forma que no *virtual cut-through*.

O método *wormhole* reduz a unidade de transmissão de dados para uma sequência de *flits*. Esta técnica é uma variação do *virtual cut-through* e tem por objetivo reduzir o tamanho dos *buffers* nas chaves. O *flit* contendo o cabeçalho é transmitido pela rede; os demais *flits* seguem o cabeçalho pelas mesmas chaves, de modo *pipeline*. No método *wormhole*, os *buffers* contém espaço para armazenamento de poucos *flits* apenas. Quando o *flit* contendo o cabeçalho fica impedido de ser transmitido, os *flits* subsequentes são armazenados no *buffer* da chave. Se a espera pela liberação demorar, os *flits* restantes serão armazenados nos *buffers* das chaves atravessadas anteriormente pelo cabeçalho.

1.3.5 Controle de Fluxo

Em uma NoC, dados que chegam em uma chave podem vir a requisitar como saída um canal que já esteja em uso. Nesta situação, é dito que há uma *contenção* ou *congestionamento* da rede. A rede deve possuir um controle que define o gerenciamento de canais de comunicação e *buffers* de forma a minimizar os efeitos do congestionamento. Um pacote bloqueado pode ser armazenado na chave de destino, bloqueado na chave anterior, desviado para uma outra chave ou até mesmo descartado, necessitando de um re-envio. Este gerenciamento é conhecido por controle de fluxo.

Em um controle de fluxo sem *buffers*, por não haver a possibilidade de armazenamento, pacotes ou *flits* são imediatamente enviados de uma chave para outra. No caso de congestionamento, duas políticas podem ser adotadas: o descarte ou a deflexão do pacote. No descarte, quando o cabeçalho de um pacote encontra um congestionamento, o pacote inteiro é inutilizado, necessitando que seja re-injetado na rede pelo nó de origem (GÓMEZ *et al.*, 2008). Na deflexão, quando o pacote encontra congestionamento, é redirecionado para outra porta livre. Esta processo, também conhecido como *hot-potato routing* (BARAN, 1964), se repete até que o pacote chegue ao nó de destino. Em geral, soluções baseadas em chaves sem *buffers* são interessantes por proporcionar uma grande economia no consumo de energia e na área de *hardware* necessária (MOSCIBRODA; MUTLU, 2009). Porém, os atrasos envolvidos pode tornar a rede pouco eficiente para uma dada aplicação.

Os métodos de controle de fluxo mais utilizados envolvem o uso de *buffers* para armazenar mensagens na situação de contenção. Algumas destas técnicas são enumeradas a seguir:

- Créditos: No controle de fluxo baseado em créditos, uma chave só recebe um pacote quando possui espaço em seus *buffers* para armazená-lo. O espaço livre para armazenamento em um *buffer* é chamado de *crédito*. A chave de destino envia a informação dos créditos para a chave que pretende enviar um pacote. A chave de origem só estará apta a transferir quando possuir crédito suficiente. O crédito na chave de origem é reduzido com o envio de dados e aumentado com a sinalização da chave de destino (KUNG; MORRIS, 1995).

- *Handshake*: No controle handshake (ZEFERINO; SUSIN, 2003), a chave de origem primeiramente ativa um sinal de *validação*, indicando sua intenção de enviar um pacote. Ao receber este sinal, a chave de destino verifica se há espaço em seus *buffers*. Se houver, a chave receptora ativa um sinal de *aceitação*. Somente após o recebimento deste sinal de aceitação a chave emissora iniciará a transmissão do pacote.

- *ACK/NACK*: Erros podem ocorrer na transmissão de um pacote. O controle de fluxo *ACK/NACK* opera com o intuito de corrigir estas possíveis falhas. Ao enviar um pacote, uma chave mantém uma cópia do dado em seus *buffers*. Em caso de acerto, a chave receptora envia um sinal de *reconhecimento* ou *ACK* (*acknowledgement*) de volta para a emissora, que então descarta os dados armazenados. Se ocorre um erro de transmissão, a chave de destino envia um sinal de não reconhecimento, ou *NACK* (*not acknowledgement*), e o pacote é re-enviado (JALABERT *et al.*, 2004).

- *Stall/Go*: O controle de fluxo *Stall/Go* opera usando dois sinais de controle, um da chave emissora para a receptora e outro no sentido inverso. Enquanto o *buffer* da chave de destino está com sua capacidade abaixo de um limiar superior, a chave de origem estará liberada para envio de *flits* de um pacote. Quando o armazenamento do *buffer* alcança este limiar, a chave de destino envia a informação *Stall*, indicando que a chave de origem deve interromper o envio do pacote. A transmissão só será retomada quando o espaço de armazenamento do *buffer* alcançar um limiar inferior, e a chave de destino enviar a sinalização *Go* para a chave de origem. Este método permite que o *buffer* trabalhe com a capacidade entre os dois limiares. O *Stall/Go* é um controle de fluxo estilo *ON/OFF* (ligado/desligado) (PULLINI *et al.*, 2005), também conhecido por controle de fluxo *slack buffer* (ZEFERINO, 2003).

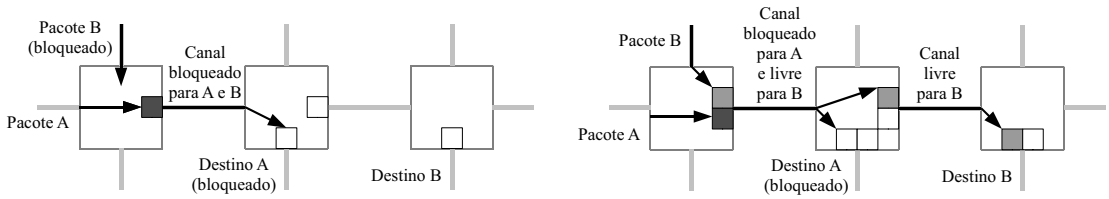
- *T-Error*: O controle *T-Error* (*timing error*, ou erro de temporização) lida de forma agressiva com o envio de *flits* através de enlaces. Ou o comprimento físico dos canais

de comunicação é reduzido, ou a frequência da transmissão de informação é elevada (em relação a outros projetos de NoC), a fim de elevar o desempenho da rede (TAMHANKAR *et al.*, 2007). Como resultado, podem ocorrer erros de transmissão decorrentes de violações na temporização no canal. Falhas são detectadas com o uso de uma re-amostragem dos dados. Um *flit* é amostrado em dois instantes, com o uso de dois sinais de *clock* defasados. Não havendo inconsistência dos dados, um sinal de validade é ativado pela chave emissora, indicando que dados devem ser transmitidos. Caso a chave receptora não possua espaço de armazenamento suficiente, um sinal de *Stall* é transmitido de volta para a chave emissora, que interrompe o envio.

Cada um dos métodos de controle de fluxo baseados em *buffers* citados possuem, de certa forma, uma característica em comum: o uso de sinais de controle entre chaves vizinhas, conforme ilustrado pela Figura 10.

Figura 10: Sinais de controle em um controle de fluxo baseado em *buffers*.

Um outro importante método de controle de fluxo é o uso de *canais virtuais* (DALLY; SEITZ, 1987). Em uma chave com *buffers*, estes não estão organizados em uma simples fila, mas em um conjunto de filas logicamente independentes. Estas filas são chamadas de canais virtuais, e compartilham todas um mesmo canal físico de comunicação. Quando o cabeçalho de um pacote encontra um congestionamento em uma rede com chaveamento *wormhole*, os *flits* seguintes também bloqueiam as portas das chaves intermediárias anteriores. Desta forma, *flits* de outros pacotes ficam impedidos de usar as portas destas chaves, que estão sendo ocupadas por *flits* que não estão sendo transmitidos devido ao bloqueio do cabeçalho. É justamente na exploração desta característica que se empregam os canais virtuais. Não havendo transmissão, o canal teoricamente estaria livre para uso na transmissão de *flits* de outros pacotes, armazenados em uma outra fila, conforme visto na Figura 11. É uma técnica que consome uma considerável área em *hadware*, visto a necessidade do aumento da quantidade de *buffers*. Contudo, o uso dos canais virtuais resulta um impacto substancial no desempenho da comunicação no sistema (VAIDYA; SIVASUBRAMANIAM; DAS, 2001).



(a) O bloqueio do pacote A impede também a transmissão do pacote B.

(b) Chaves usando canais virtuais possibilitam que o pacote B seja enviado mesmo com o bloqueio de A.

Figura 11: Uso dos canais virtuais.

1.3.6 Roteamento

Técnicas de chaveamento e controle de fluxo definem como fisicamente será o uso de chaves e canais de comunicação. Especificam como pacotes são transmitidos desde a chave de origem até a de destino. Já o algoritmo de roteamento não lida com a forma como que um pacote é transmitido, mas sim com a escolha do percurso ou rota necessária para que um determinado pacote seja transmitido. O objetivo de um algoritmo de roteamento é, dado um nó de origem e um de destino, descobrir por quais nós intermediários o pacote pode ser enviado. Se existir mais de um par origem/destino, o roteamento deve ser capaz de encontrar um caminhos para cada pacote de forma que não ocorra - ou que pelo menos minimalize - a ocorrência colisão dos dados.

Diversos são os algoritmos de roteamento existentes para redes embutidas. Esta variedade é decorrente da grande experiência prévia existente na área de redes de comunicação, como redes de telefonia ou de computadores. Essas redes possuem diferentes tipos de topologias, técnicas de chaveamento, controle de fluxo; consequentemente também os algoritmos de roteamento tiveram de ser adaptados a fim de atender as características próprias de cada rede. Bem como estas características, também o conhecimento existente sobre roteamento em redes de comunicação foi herdado para o uso em NoCs. Os principais algoritmos de roteamento existentes são enumerados e descritos no Capítulo 2 desta dissertação.

Considerando sua diversidade, os algoritmos de roteamento podem ser classificados sob vários aspectos diferentes.

O local onde o algoritmo é executado define o roteamento como centralizado, na fonte (ou origem), ou distribuído. No roteamento centralizado, um controlador central da rede estabelece todos os caminhos de forma global. Em um roteamento na origem, a chave emissora decide o percurso do pacote antes de seu envio. Já o roteamento distribuído é realizado em

todas as chaves enquanto um pacote atravessa a rede. Tanto o roteamento na origem quanto o distribuído necessita que a chave possua um controlador interno que realize o roteamento - diferentemente do centralizado, que as chaves fazem uso de um controlador externo. Por esse motivo, muitas vezes a chave também é chamada de *roteador*. É possível ainda um roteamento híbrido entre o distribuído e o ocorrido na fonte, chamado de roteamento multi-fase.

O instante em que o roteamento ocorre caracteriza o roteamento como estático ou dinâmico. O roteamento estático ocorre antes da injeção de dos dados na rede. Uma vez estabelecido, um caminho obtido por um roteamento estático não mais poderá ser alterado. Já o roteamento dinâmico ocorre durante o envio do pacote pela rede, quando este chega em cada uma das chaves.

Considerando a quantidade de destinatários de um pacote, o roteamento pode ser *unicast* ou *multicast*. Em um roteamento *unicast*, um pacote é enviado de um nó de origem para um único nó de destino na rede. O roteamento *multicast* é caracterizado por um nó distribuindo o mesmo pacote para vários destinatários simultaneamente. A solução *unicast* torna-se mais prática em NoCs, tendo em vista a existência de canais ponto-a-ponto conectando duas chaves.

Quanto a implementação, o roteamento pode ser baseado em tabela, se o caminho de cada pacote de uma mensagem for decidido a partir da consulta a uma tabela armazenada em memória (*lookup table*). Ou então, a implementação pode ser baseada em uma máquina de estados, ou FSM (*Finite State Machine*), se o caminho for decidido a partir da execução de um algoritmo implementado em *hardware* ou *software*.

A forma com que o algoritmo realiza a busca por caminhos classifica como determinístico ou adaptativo. Na abordagem determinística, dados os nós de origem e de destino, o algoritmo retorna sempre o mesmo caminho. No modelo adaptativo, diferentes caminhos podem ser encontrados, dependendo de certas informações da rede como tráfego e uso de canais. Assim, um pacote pode desviar de uma região congestionada da rede, obtendo uma menor latência de transmissão. Algoritmos de roteamento adaptativos podem ainda ser classificados quanto a progressividade, minimalidade e número de caminhos.

O roteamento é dito progressivo se o cabeçalho de um pacote sempre avança pela rede, reservando assim um novo canal de comunicação a cada passo de roteamento. O algoritmo será regressivo se, em caso de algum tipo de falha o cabeçalho puder retornar pelo caminho traçado (*backtracking*), liberando assim os canais previamente utilizados.

Quanto a minimalidade, o roteamento será mínimo quando um pacote se deslocar sem-

pre para uma chave mais próxima do destino, fazendo com que o percurso descrito seja sempre o mais curto possível ou que utilize a menor quantidade possível de chaves intermediárias. O roteamento será considerado não mínimo se o caminho encontrado utilizar uma quantidade de nós maior que a mínima. É uma situação que pode ocorrer em um algoritmo adaptativo ao desviar pacotes de chaves congestionadas.

O roteamento é considerado completo ou total quando o algoritmo puder utilizar todos os nós da rede ou todas as combinações de caminhos possíveis. Por outro lado, o roteamento será considerado parcial se apenas um subconjunto dos caminhos possíveis puder ser utilizado.

1.3.7 Condições não desejadas

Como já foi dito, um algoritmo de roteamento deve ser capaz de encontrar um caminho através da rede que conecte os nós de origem e de destino. Contudo, três situações - *deadlock*, *livelock* e *starvation* - podem vir a ocorrer na construção de percursos, de forma que a transmissão de pacotes fica impossibilitada.

O *deadlock* é um problema que ocorre quando um pacote fica bloqueado em uma chave aguardando a liberação de um canal de comunicação que nunca estará livre. Um exemplo é o caso da dependência cíclica, como pode ser vista na Figura 12. O primeiro pacote espera a liberação da porta pelo segundo pacote. O segundo espera pelo terceiro, o terceiro pelo quarto, e o quarto pelo primeiro. Como o primeiro já está bloqueado, os quatro ficam impossibilitados de transmitir.

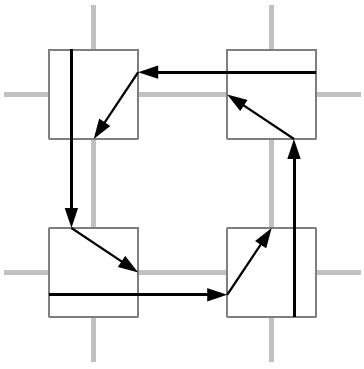


Figura 12: Quatro pacotes em dependência cíclica, ocasionando uma situação de *deadlock*.

Em um roteamento adaptativo, o pacote transmitido pode ser desviado de chaves em congestionamento, gerando assim um percurso não mínimo. O *livelock* é a condição em que um pacote, dadas as condições de tráfego da rede, sempre é redirecionado, nunca alcançando a chave de destino. É um problema que pode ser evitado restringindo a quantidade de desvios que o pacote pode realizar.

Uma chave necessita de um controle de arbitragem (ou simplesmente, árbitro), que pode ser distribuído ou centralizado. Dois pacotes que chegam simultaneamente em uma chave podem, devido ao algoritmo de roteamento empregado, requisitar ambos a mesma porta de saída. Nesta situação, o controlador irá arbitrar qual dos dois pacotes terá prioridade de envio. Se um dos pacotes possuir uma baixa prioridade, seu envio poderá sempre ser postergado caso novos pacotes continuam a chegar à chave escolhendo a mesma porta como saída. Este problema é conhecido como *starvation*, e pode ser evitado com o uso de uma política de arbitragem adequada.

1.4 Metodologia de projeto de NoCs

Em geral, deseja-se que um sistema embutido realize uma dada aplicação; ao mesmo tempo, este sistema deve ser implementado de forma a atender requisitos, tais como custo de *hardware*, consumo de energia e tempo de execução. A Figura 13 apresenta de forma simplificada o fluxograma de um projeto de SoC baseado em redes embutidas.

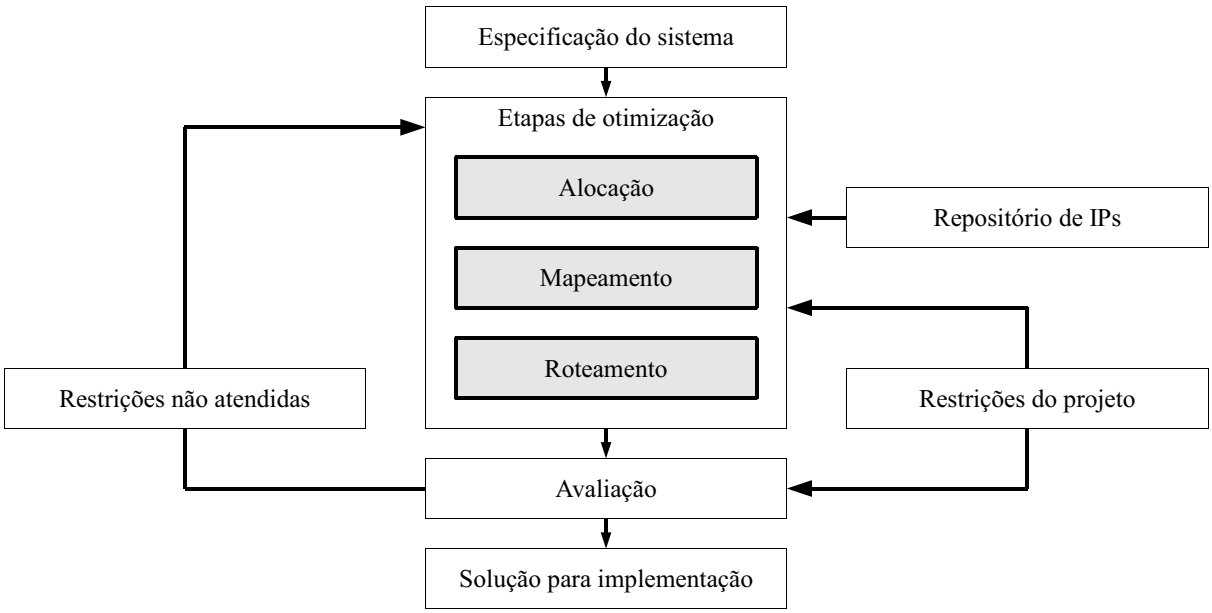


Figura 13: Fluxo típico de projeto de sistemas embutidos para plataforma NoC

O primeira passo do projeto consiste na especificação do sistema. Nesta etapa são introduzidas as restrições do projeto, como tempo máximo de execução ou área máxima de *hardware*. Também nesta etapa é definida a aplicação a ser executada pelo sistema. A aplicação é descrita por um *grafo de tarefas*[1], onde cada tarefa é um algoritmo específico.

[1]A definição de grafo de tarefas e seu uso em SoCs será apresentada no Capítulo 6 desta dissertação.

A etapa seguinte pode ser definida como uma etapa de otimização: a partir das especificações do sistema, certas características da rede são otimizadas por uma ferramenta de auxílio de projeto de forma a atender as restrições de projeto. Dentre os processos que necessitam de otimização, podem ser citados a alocação de IPs, o mapeamento de tarefas e o roteamento de pacotes. O processo de alocação de IPs consiste em associar cada tarefa (ou conjunto de tarefas) a um bloco IP adequado, dentro de um conjunto ou repositório de IPs capazes de executar tal tarefa. O mapeamento de uma aplicação consiste em associar o conjunto de IPs resultante da alocação a cada um dos canais de acesso da infraestrutura de comunicação - neste caso, a NoC. Em outras palavras, é definido onde espacialmente cada recurso será implementado, ou ainda, em que nós da rede cada núcleo IP estará conectado.

Considerando que várias alocações são possíveis - bem como vários mapeamentos - é comum o uso de ferramentas de auxílio baseadas em inteligência computacional. Desta forma, tais algoritmos são capazes de realizar escolhas baseados na avaliação do *trade-off* entre custo de implementação e desempenho do sistema.

1.5 Considerações Finais do Capítulo

Neste capítulo foi apresentado o conceito de SoCs, como sistemas compostos por diversos blocos em um mesmo CI. As principais arquiteturas de comunicação foram abordadas, como enlaces ponto-a-ponto e barramentos compartilhados (BJERREGAARD; MAHADEVAN, 2006). Redes embutidas foram descritas como uma solução para comunicação em SoCs de grande escala. As principais características das NoC foram detalhadas. No capítulo seguinte são abordados os principais algoritmos de roteamento empregados em redes embutidas e de comunicação.

Capítulo 2

TRABALHOS RELACIONADOS

ESTE capítulo apresenta alguns dos algoritmos de roteamento mais utilizados em redes embutidas. Em geral, são algoritmos desenvolvidos para sistemas distribuídos ou multiprocessados, tendo sido adaptados para o uso em NoCs. Deu-se preferência a métodos aplicáveis a redes com chaveamento *wormhole*, por ser este o chaveamento empregado no roteamento proposto nesta dissertação. Entre os roteamentos citados, estão o algoritmo XY e o modelo *Odd-Even*, usados nos Capítulos 5 e 6 para comparação de resultados.

2.1 Algoritmo XY

Uma malha n-dimensional é uma topologia de redes que possui $K_0 \times K_1 \times \ldots \times K_{n-1}$ nós e onde n é o número de dimensões da rede e K_i é a raiz da dimensão i. Cada nó é identificado por um vetor de n-coordenadas $(x_0, x_1, \ldots, x_n)$, onde $0 \leq x_i \leq K_i - 1$. Um esquema de numeração de canais amplamente utilizado em malhas n-dimensionais é baseado na dimensão dos enlaces. Em um roteamento ordenado por dimensão (DOR, *Dimension Ordered Routing*), cada pacote é transportado através de uma dimensão por vez, necessitando alcançar a coordenada correta em uma dimensão antes de iniciar a transmissão pela dimensão seguinte. Em uma malha 2D, uma vez que cada nó é representado por sua posição (x, y), a utilização de um DOR resulta no algoritmo de roteamento XY (INTEL, 1991). Em hipercubos, o algoritmo de roteamento obtido é o *e-cube* (SULLIVAN; BASHKOW, 1977).

No roteamento XY, dados são enviados primeiro através da dimensão X e depois pela dimensão Y. Em outras palavras, apenas uma mudança de direção é permitida neste modelo (NI; MCKINLEY, 1993). Considerando (S_x, S_y) o endereço do nó de origem e (D_x, D_y) o endereço do nó do destinatário, o roteamento XY pode ser realizado inserindo ambos os endereços no cabeçalho do pacote. Um outro endereço, (C_x, C_y), indica a posição atual do cabeçalho na rede. Primeiramente, o valor de C_x é incrementado ou decrementado de forma a variar desde

S_x até D_x. Enquanto o valor de C_x for diferente de D_x, o pacote continuará a ser transmitido na mesma direção. Quando C_x e D_x se igualam, a transmissão em X se encerra. O processo se repete, desta vez com C_y, indicando a transmissão em Y. Esta descrição de roteamento XY pode ser visto no Algoritmo 1.

Algoritmo 1 Roteamento XY

Entrada (S_x, S_y), (D_x, D_y)
1: $C_x \leftarrow S_x$; $C_y \leftarrow S_y$;
2: **Enquanto** $C_x \neq D_x$ **E** $C_y \neq D_y$ **Faça**
3: **Se** $C_x < D_x$ **Então** {Deslocamento para o Leste}
4: $C_x \leftarrow C_x + 1$;
5: Transmite o pacote para o nó (C_x, C_y);
6: **Senão Se** $C_x > D_x$ **Então** {Deslocamento para o Oeste}
7: $C_x \leftarrow C_x - 1$;
8: Transmite o pacote para o nó (C_x, C_y);
9: **Senão Se** $C_x = D_x$ **Então**
10: **Se** $C_y < D_y$ **Então** {Deslocamento para o Norte}
11: $C_y \leftarrow C_y + 1$;
12: Transmite o pacote para o nó (C_x, C_y);
13: **Senão Se** $C_y > D_y$ **Então** {Deslocamento para o Sul}
14: $C_y \leftarrow C_y - 1$;
15: Transmite o pacote para o nó (C_x, C_y);
16: **Senão Se** $C_y = D_y$ **Então**
17: Fornece o pacote ao recurso de destino;
18: **Fim Se**
19: **Fim Se**
20: **Fim Enquanto**

Malhas 2D com roteamento *wormhole* baseado no algoritmo XY foram empregadas em diversos computadores paralelos (*multicomputers*), tais como o Intel Touchstone DELTA (INTEL, 1991), o Intel Paragon (ESSER; KNECHT, 1993), o Symult 2010 (SEITZ *et al.*, 1988) e o Caltech MOASIC (SEITZ *et al.*, 1993).

No contexto de NoCs, roteamento XY mostra-se eficiente visto sua simplicidade de implementação e o fato de ser livre da ocorrência de *deadlocks*. Dentre os trabalhos que fizeram uso deste algoritmo, podem ser citados a rede HERMES (MORAES *et al.*, 2004) e a rede SoCIN (*System on Chip Interconnection Network*) (ZEFERINO; SUSIN, 2003). Em ambas as implementações, a rede possui topologia malha 2D, com chaves possuindo cinco portas bidirecionais (norte, sul, leste, oeste e a comunicação local com o recurso).

2.2 O1Turn

O *O1Turn* foi desenvolvido por Seo et al. para roteamento em redes de comunicação (SEO *et al.*, 2005). O termo vem de *orthogonal one-turn routing* (roteamento ortogonal de um giro), indicando que os pacotes enviados pela rede estão habilitados a realizar no máximo uma mudança de direção. Desta forma, todas as rotas encontradas pelo algoritmo são mínimas. Em uma malha 2D, apenas duas rotas são permitidas entre o nó de origem e o de destino: uma em que o pacote é transmitido primeiramente por X e depois por Y, e outra em que a transmissão é iniciada por Y e depois por X. Isto equivale a dizer que o *O1Turn* faz uso do dos algoritmos XY e YX, conforme ilustrado na Figura 14.

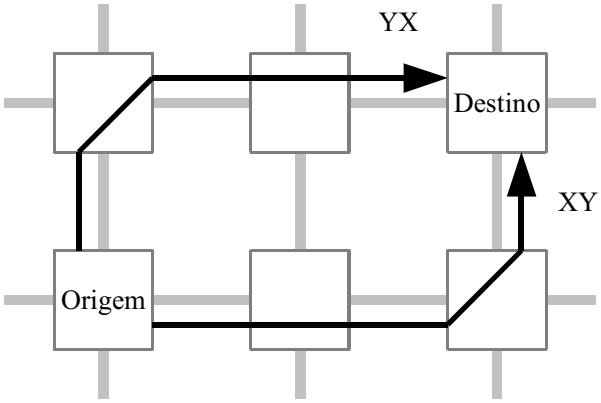


Figura 14: Caminhos possíveis entre nós de origem e destino em uma malha 2D usando roteamento *O1Turn*.

O *O1Turn* foi desenvolvido pensando no pior caso de uma rede usando o algoritmo XY (ou YX) (TOWLES; DALLY, 2002). Segundo os autores, nesta situação a carga de um canal de comunicação pode ser reduzida em 50% se metade dos pacotes usarem o roteamento XY e a outra metade, YX. No roteamento *O1Turn*, esta ideia é ampliada para toda a rede. A escolha pelo algoritmo é aleatória: todos os pacotes transmitidos têm 50% de chance de usarem cada um dos roteamentos, XY ou YX. Outro fator interessante está na implementação física de uma chave empregando o roteamento *O1Turn*, que possui uma complexidade comparável a de uma chave que use o algoritmo XY.

2.3 Algoritmo WOT

Um roteamento semelhante ao *O1Turn* foi desenvolvido de forma independente por Gindin et al. para o roteamento em NoCs. Em seu trabalho, vários algoritmos são apresentados, sendo o principal chamado de WOT (GINDIN; CIDON; KEIDAR, 2007). O TXY, ou XY alternado

(*Toggle XY*), divide o fluxo de pacotes pelos dois caminhos possíveis usando XY e YX. Esta solução, porém, não apresenta um bom desempenho com padrões de tráfego não uniformes. O roteamento WTXY (*Weighted Toggle XY*, ou XY alternado por peso) adiciona pesos a cada opção possível, visando balancear o tráfego na rede. Contudo, estes dois modelos de roteamento realizam a divisão do fluxo de pacotes, o que pode ocasionar a desorganização dos mesmos no nó de saída. O algoritmo STXY (*Source Toggle XY*, ou XY alternado na origem) é apresentado como uma versão do TXY sem a divisão do fluxo de pacotes. Unindo a ideia de pesos do WTXY e a não divisão dos dados do STXY, obteve-se o WOT (*Weighted Ordered Toggle*, ou alternância ordenada por peso).

Diferente do *O1Turn*, que seleciona XY ou YX de forma puramente aleatória, o WOT faz uso de informação heurística da rede. É um algoritmo que inicialmente utiliza o STXY como solução, e posteriormente otimiza os caminhos de forma iterativa. O WOT é um roteamento estático, com os percursos sendo calculados pelo algoritmo para cada aplicação antes da etapa de implementação física da rede.

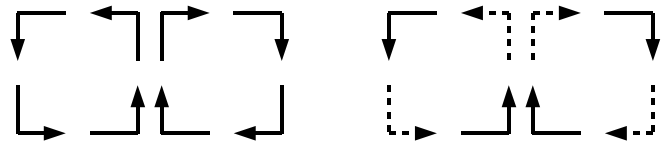
2.4 Turn Model

Glass e Ni propuseram o chamado *Turn Model* (modelo de giro) para a criação de algoritmos de roteamento adaptativos livres de *deadlocks* e *livelocks* (GLASS; NI, 1992). Um giro é uma mudança de 90° na direção do envio de um pacote. A ideia principal deste modelo é restringir a quantidade de giros que um pacote pode realizar, de forma a evitar a formação de ciclos que ocasionam *deadlocks*. As etapas que descrevem o *turn model* são apresentadas a seguir:

1. Classificar os canais de acordo com a direção em que os pacotes são roteados.
2. Identificar os giros que podem ocorrer entre duas direções.
3. Identificar os ciclos que os giros podem formar.
4. Proibir um número mínimo de giros, de forma a evitar a ocorrência de ciclos. Pelo menos um giro de cada ciclo deve ser proibido.

Em uma malha 2D, quatro direções são identificadas: Norte, Sul, Leste e Oeste (NSLO). Oito giros são possíveis considerando estas quatro direções, sendo eles: LS, LN, OS, ON, SL, SO, NL e NO. Estes giros formam 2 ciclos, conforme visto na Figura 15(a). No roteamento XY, quatro destes giros são proibidos, conforme ilustrado pela Figura 15(b). Esta proibição impede a ocorrência de *deadlocks*, uma vez que os quatro giros restantes são incapazes de

formar um ciclo. Contudo, os giros restantes também não permitem nenhuma adaptatividade ao roteamento.

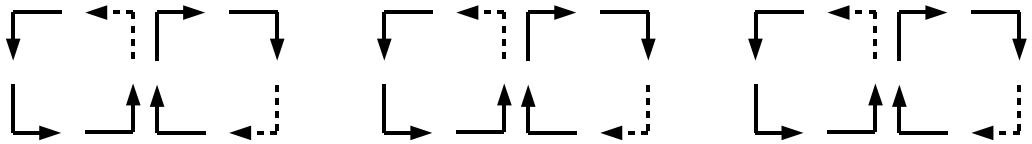


(a) Ciclos formados pelos giros possíveis em uma malha 2D.

(b) Giros proibidos no roteamento XY (linhas tracejadas).

Figura 15: Giros ou mudanças de direção na transmissão de um pacote em malhas.

Baseado nos conceitos introduzidos pelo *turn model*, Glass e Ni propuseram três algoritmos de roteamento: o *West-First* (primeiro pelo oeste), o *North-Last* (último pelo norte) e o *Negative-First* (negativos primeiro). As mudanças de direção para cada um destes algoritmos são apresentadas na Figura 16. No *West-First*, os dois giros para Oeste, NO e SO, são proibidos. Por este motivo, o pacote deve iniciar a transmissão por esta direção se necessitar do envio para o Oeste. O roteamento *Nort-Last* proíbe os giros NL e NO. Como ambos os giros a partir da direção Norte estão impedidos, então esta deve ser a última direção a ser utilizada. Por fim, o roteamento *Negative-First* proíbe ambos os giros NO e LS. Estes são considerados giros negativos, indo das direções Norte e Leste para Oeste e Sul. Neste roteamento, o pacote deve ser primeiramente transmitido para as direções chamadas negativas (Oeste e Sul), para só então ser roteado pelas direções positivas (Norte e Leste).



(a) *West-First.* (b) *North-Last.* (c) *Negative-First.*

Figura 16: Giros proibidos nos algoritmos do *Turn Model*.

Nestes três modelos o objetivo é obter um roteamento restringindo apenas o menor número de giros, de forma a obter um algoritmo livre de *deadlocks*. De fato, nas três situações, apenas dois giros (um em cada ciclo) são proibidos. Mais giros poderiam ser restritos, contudo também a adaptatividade do algoritmo seria reduzida.

2.5 Odd-Even Turn Model

Chiu propôs o chamado *Odd-Even Turn Model* (modelo de giro par-ímpar) para algoritmos de roteamento parcialmente adaptativos em malhas (CHIU, 2000). Este modelo possui inspiração no método proposto por Glass e Ni; contudo, diferente do *turn model*, o *Odd-Even* (OE) é livre de *deadlocks* não por restringir a ocorrência de certos tipos de mudança de direção, mas sim por restringir o local onde certos giros podem ocorrer.

No modelo *Odd-Even*, uma malha 2D é definida como sendo composta por linhas (nós com mesmo valor na coordenada de dimensão 1) e colunas (nós com mesmo valor na coordenada de dimensão 0). Uma coluna é dita *par* se for identificada por um valor par, ou *ímpar* se o valor da coordenada for ímpar. Exemplificando: em uma malha $K_0 \times K_1$, a coluna $(2, j)$, com $0 < j < K_1 - 1$, é uma coluna par.

O fundamento básico do *Odd-Even* consiste em impedir que certos giros ocorram em uma dada posição da rede, tais como LN e NO (ou LS e SO)em uma mesma coluna. Mais precisamente, este modelo é ditado pelas duas regras a seguir:

Regra 1

Não é permitido a nenhum pacote realizar um giro LN em nós localizados em colunas pares, nem giros NO em nós localizados em colunas ímpares.

Regra 2

Não é permitido a nenhum pacote realizar um giro LS em nós localizados em colunas pares, nem giros SO em nós localizados em colunas ímpares.

Nenhuma restrição a mais é necessária. As colunas pares ou ímpares podem ainda ser invertidas em ambas as regras. A ideia deste modelo é impedir que um pacote proveniente de um nó mais a Oeste na rede (um pacote que realizou um deslocamento para o Leste) retorne para o Oeste após ser transmitido para o Norte ou Sul, conforme visto na Figura 17.

Baseado no modelo *Odd-Even*, vários algoritmos podem ser feitos. Desde que as Regras 1 e 2 sejam obedecidas, o roteamento será parcialmente adaptativo livre de *deadlocks*. O autor apresenta o algoritmo *ROUTE* (CHIU, 2000) para a seleção de nós intermediários. Durante o roteamento entre origem e destino, um nó intermediário utiliza o *ROUTE* para construir um conjunto de direções possíveis, obedecendo as Regras 1 e 2. O nó intermediário seguinte é então selecionado entre as direções disponíveis. O Algoritmo 2 ilustra o *ROUTE* proposto por Chiu.

Algoritmo 2 Algoritmo *ROUTE* para Roteamento OE

Entrada (S_0, S_1), (D_0, D_1)
1: $C_0 \leftarrow S_0$;
2: $C_1 \leftarrow S_1$;
3: **Enquanto** $C_x \neq D_x$ **E** $C_y \neq D_y$ **Faça**
4: $E_0 \leftarrow D_0 - C_0$;
5: $E_1 \leftarrow D_1 - C_1$;
6: **Se** $E_0 = 0$ **E** $E_1 = 1$ **Então** {É o nó de destino.}
7: Entrega o pacote para o nó de destino e sai;
8: **Fim Se**
9: **Se** $E_0 = 0$ **Então** {Está na mesma coluna que o nó de destino.}
10: **Se** $E_1 > 0$ **Então**
11: Adiciona Norte ao conjunto de direções;
12: **Senão**
13: Adiciona Sul ao conjunto de direções;
14: **Fim Se**
15: **Senão** {Está a oeste da coluna do nó de destino.}
16: **Se** $E_0 > 0$ **Então**
17: **Se** $E_1 = 0$ **Então**
18: Adiciona Leste ao conjunto de direções;
19: **Senão**
20: **Se** C_0 é par **OU** $C_0 = S_0$ **Então**
21: **Se** $E_1 > 0$ **Então**
22: Adiciona Norte ao conjunto de direções;
23: **Senão**
24: Adiciona Sul ao conjunto de direções;
25: **Fim Se**
26: **Fim Se**
27: **Se** D_0 é ímpar **OU** $E_0 \neq 1$ **Então**
28: Adiciona Leste ao conjunto de direções;
29: **Fim Se**
30: **Fim Se**
31: **Fim Se**
32: **Senão** {Está a leste da coluna do nó de destino.}
33: Adiciona Oeste ao conjunto de direções;
34: **Se** C_0 é par **Então**
35: **Se** $E_1 > 0$ **Então**
36: Adiciona Norte ao conjunto de direções;
37: **Senão**
38: Adiciona Sul ao conjunto de direções;
39: **Fim Se**
40: **Fim Se**
41: **Fim Se**
42: Seleciona uma direção do conjunto de direções possíveis;
43: $(C_0, C_1) \leftarrow$ nó na direção selecionada;
44: **Fim Enquanto**

(a) Leste-Norte e Norte-Oeste em uma mesma coluna.

(b) Leste-Sul e Sul-Oeste em uma mesma coluna.

Figura 17: Sequência de giros proibidos no modelo *Odd-Even*.

2.6 Roteamento DyAD

Hu e Marculescu desenvolveram uma infraestrutura que combina roteamento determinístico e adaptativo (HU; MARCULESCU, 2004). O método, chamado DyAD (*Dynamic Adaptive Deterministic*, ou chaveamento determinístico adaptativo dinâmico), alterna entre os dois tipos de roteamento conforme a condição da rede.

A motivação do DyAD vem da análise do desempenho de uma rede com o uso de roteamento determinístico e adaptativo. Os autores observaram que sob uma situação de baixa utilização, a rede apresenta uma menor latência com o uso de um roteamento determinístico em comparação ao adaptativo. Por outro lado, sob uma intensa utilização da rede, roteamentos determinísticos tornam-se mais susceptíveis a ocasionar congestionamento. Nesta situação, roteamentos adaptativos apresentam uma menor latência. O DyAD é capaz de se valer das vantagens de ambos os roteamentos determinístico e adaptativo, selecionando criteriosamente qual roteamento usar sob diferentes condições de utilização da rede.

O DyAD utiliza o algoritmo XY quando opta por um roteamento determinístico, e o *Odd-Even* para o caso adaptativo. Por ambos serem roteamentos livres de *deadlocks* e *livelocks*, a rede com DyAD também será livre destas duas situações.

2.7 Modelos baseados em aleatoriedade

Vários modelos de roteamento em redes baseados em aleatoriedade foram propostos. Esta seção apresenta brevemente alguns destes trabalhos.

Valiant e Brebner desenvolveram um método que tornou-se conhecido como *Roteamento Valiant* (VALIANT; BREBNER, 1981). Neste método, o caminho desde o nó de origem até o nó de destino atravessa obrigatoriamente um nó intermediário escolhido aleatoriamente. O caminho

total é composto, pois, por dois percursos: um do nó de origem até o nó intermediário aleatório, e outro do nó intermediário até o de destino. Ambos os caminhos são selecionados utilizando-se um DOR. Desta forma, o roteamento é livre de *deadlocks* e *livelocks*. Como qualquer nó da rede pode ser selecionado, incluindo os mais distantes da origem e do destino, trata-se de om roteamento não mínimo. É um roteamento que apresenta como problema o fato de dobrar o caminho médio descrito pelas mensagens. Contudo, apresenta um melhor balanceamento na utilização da rede, devido a adição do fator aleatório.

Nesson e Johnson propuseram o roteamento ROMM (*Randomized, Oblivious, Multi-phase, Minimal*) (NESSON; JOHNSSON, 1994). É um roteamento semelhante ao Valiant; contudo, não apenas um, mas vários nós aleatórios podem ser selecionados. Para garantir a minimalidade do roteamento, apenas nós localizados espacialmente entre a origem e o destino podem ser selecionados. Também a quantidade total de nós intermediários não ultrapassar a quantidade de nós em um caminho mínimo entre origem e destino.

O *Chaos Router* de Konstantinidou e Snyder faz uso do chamado *roteamento caótico*, sendo adaptativo, não mínimo e aleatório (KONSTANTINIDOU; SNYDER, 1991). O *Chaos Router* seleciona aleatoriamente um caminho mínimo entre origem e destino. Em caso de congestionamento, a mensagem é redirecionada por um outro caminho, o que pode ocasionar caminhos não mínimos. O roteador não necessita de proteção contra *livelocks*. A completa aleatoriedade na escolha por caminhos leva a rede a um estado caótico. Nesta situação, padrões repetitivos de roteamento que causam *livelocks* tendem a ocorrer em menor quantidade. Assim, os autores assumem que este roteamento é probabilisticamente livre de *livelocks*. Uma implementação física do *Chaos Router* foi feita por (BOLDING *et al.*, 1994).

2.8 AntNet

Di Cado e Dorigo propuseram o AntNet, um algoritmo de roteamento inspirado na otimização por colônia de formigas[1] (DI CARO; DORIGO, 1998b) (DI CARO; DORIGO, 1998a). Trata-se de um roteamento adaptativo, dinâmico e distribuído, desenvolvido originalmente para comunicação em redes de computadores.

No AntNet, cada nó possui uma tabela de roteamento contendo, além das informações referentes ao roteamento, informações estatísticas sobre a utilização da rede. Esta tabela mantém uma medida de qualidade para cada possível nó a ser usado na transmissão de uma

[1]Uma descrição detalhada sobre otimização por colônia de formigas é apresentada no Capítulo 3 desta dissertação.

mensagem. Estes valores de qualidade, chamados variáveis feromônio, são usados por uma política de roteamento estocástica. AntNet utiliza dois conjuntos de agentes móveis, chamados *forward ants* (formigas que avançam pela rede) e *backward ants* (formigas que retornam pela rede). Formigas são pacotes de controle que periodicamente são injetados na rede. As *forward ants* utilizam a informação heurística das tabelas de roteamento para selecionar qual caminho escolher, desde a chave de origem até a de destino. Quando o destinatário é alcançado, uma *backward ant* é injetada na rede, retornando pelo mesmo caminho descrito pela *forward ant* (porém, em sentido inverso), até o nó de origem. Para cada nó visitado pela *backward ant*, a tabela de roteamento é atualizada.

Daneshtalab adaptou o roteamento AntNet para o uso em redes embutidas (DANESHTALAB *et al.*, 2006). O autor compara o algoritmo com outros modelos de roteamento, como XY, *Odd-Even* e DyAD. Os resultados de simulação mostram o AntNet possuindo um melhor desempenho que os demais algoritmos em uma situação de alto tráfego na rede.

2.9 Considerações Finais do Capítulo

Neste capítulo foram apresentados resumidamente diversos trabalhos sobre roteamento em redes de comunicação, incluindo métodos que serviram de inspiração para o roteamento em NoCs. Dois destes métodos, XY e *Odd-Even* servirão de base para os algoritmos para comparação no Capítulo 5. Também é apresentado como a otimização por colônia de formigas é empregada em trabalhos envolvendo roteamento. O AntNet, contudo, utiliza esta otimização na implementação de um roteamento dinâmico.

O capítulo seguinte apresenta uma introdução teórica sobre a Otimização por Colônia de Formigas, o método computacional empregado na construção de algoritmos de roteamento adaptativo propostos nesta dissertação.

Capítulo 3

OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

INTELIGÊNCIA de enxame (*swarm intelligence*) é o campo de pesquisa que estuda a computação inspirada no comportamento de grupos de indivíduos, como insetos e pássaros. Um algoritmo baseado em enxame é composto por um grupo de agentes simples que cooperam em si para a solução de um problema. Cada membro do enxame interage com os demais indivíduos e com o ambiente, não havendo um controle central sobre o sistema.

Várias técnicas foram propostas tendo como base a inteligência coletiva. Dentre as mais difundidas estão a otimização por enxame de partículas ou PSO (*Partitcle Swarm Optimization*), inspirada no voo de um bando de pássaros, e a otimização por colônia de formigas ou ACO (*Ant Colony Optimization*). Há ainda modelos que se baseiam no comportamento social de bactérias, aranhas abelhas e tubarões (ENGELBRECHT, 2006).

Certas espécies de formigas apresentam um comportamento interessante: a capacidade de realizar tarefas que os cientistas de computação chamam de *busca por um caminho mínimo* (DORIGO; STÜTZLE, 2004). Biologistas provaram que o trabalho coletivo de formigas é possível graças ao *feromônio* - substância química depositadas por uma formiga para que as demais possam segui-la. Este padrão de comunicação indireta levantou o interesse de pesquisadores, possibilitando o desenvolvimento de algoritmos de otimização combinatória baseados em formigas. O primeiro algoritmo, chamado *Ant System*, propôs o uso do comportamento das formigas na solução do problema do caixeiro viajante. Técnicas derivadas do *Ant System* mostraram-se eficientes na solução de uma grande variedade de problemas de otimização. Atualmente, as bases do *Ant System* definem formalmente a meta-heurística ACO.

A organização deste capítulo é descrita a seguir. A Seção 3.1 apresenta o comportamento de formigas reais. A Seção 3.2 introduz a ideia de formigas artificias, e como este conceito pode ser usado em computação. Na Seção 3.3, são apresentados os principais algoritmos de

formigas. O *Ant System* é descrito detalhadamente, seguido de uma enumeração de suas principais variações. A Seção 3.4 apresenta a estrutura geral da meta-heurística ACO.

3.1 A inspiração na Natureza

Formigas, assim como abelhas, cupins e outros insetos que vivem em colônias, apresentam uma complexa organização social. Individualmente, cada formiga é capaz de realizar apenas tarefas simples. Porém, trabalhando em conjunto, um grupo destes insetos é capaz de executar tarefas complexas, tais como:

- Construção manutenção de seu ninho: estrutura formada muitas vezes por um intricado sistema de túneis subterrâneos.
- Busca e transporte de comida: mesmo localizada a uma grande distância do ninho, ao encontrarem comida, formigas conseguem guiar um grande numero de indivíduos através de longas filas.

3.1.1 Stigmergia

O entomologista francês Pierre-Paul Grassè foi um dos primeiros a estudar o comportamento social de insetos (GRASSÉ, 1946). Observando certas espécies de cupins, chegou a constatação os insetos reagiam ao que ele chamou de *significant stimuli*. Esta reação se dá através da produção de mais estímulos para o próprio inseto e para os demais membros da colônia. Grassé usou o termo *stigmergia*, do grego *stigma* (marca) e *ergon* (trabalho) para descrever "trabalhadores que são estimulados pelo desempenho previamente obtido" (GRASSÉ, 1959).

Duas características que diferenciam a stigmergia de outras formas de comunicação (DORIGO; BIRATTARI; STÜTZLE, 2006) são listadas a seguir:

- Ocorre de forma indireta. Trata-se de uma comunicação não-simbólica, onde a natureza da informação transmitida pelos insetos corresponde a uma modificação física do ambiente visitado.
- A informação é de natureza local, isto é, apenas pode ser acessada se o inseto visitar o local ou vizinhanças de onde a informação foi liberada.

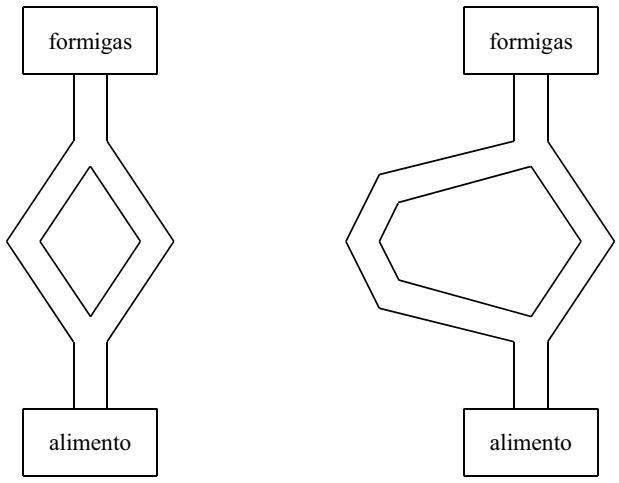
A stigmergia consiste, portanto, em um mecanismo que conduz à emergência de um comportamento coletivo descrito por todos os membros da colônia. Este comportamento não resulta de nenhuma intencionalidade nem organização prévia. Cada inseto de um grupo que

apresenta stigmergia, em verdade, não faz mais que agir sobre uma pequena e limitada vizinhança local, sem muitas das vezes se darem conta do resultado final da tarefa realizada (MOURA; PEREIRA, 2003).

Um exemplo de stigmergia é visto em uma colônia de formigas. Grande parte das espécies são constituídas por indivíduos com visão bastante rudimentar ou ainda por insetos completamente cegos. Porém, formigas possuem uma capacidade extremamente desenvolvida para identificar substâncias químicas chamadas feromônios, que são produzidos por outras formigas. Ao caminhar do ninho até uma fonte de comida, uma formiga deixa um rastro de feromônio; as formigas que posteriormente passarem pelo mesmo caminho, ao sentirem a presença do feromônio, poderão ser influenciadas a segui-lo. Após um período finito de tempo, um grande número de formigas estará compartilhando um mesmo caminho entre a fonte de comida e o ninho, em um percurso desenhado pelo feromônio resultante das diversas formigas que por ele passaram.

3.1.2 O experimento da ponte dupla

Visando compreender melhor o uso de feromônios por formigas, Deneubourg et al. desenvolveu o chamado *experimento da ponte dupla* (DENEUBOURG *et al.*, 1990). Neste experimento, uma colônia de formigas da espécie *Iridomyrmex humilis* foi colocada próxima a uma fonte de alimento, estando apenas separadas por duas pontes de mesmo tamanho, como ilustrado na Figura 18.



(a) Pontes com comprimentos iguais.

(b) Pontes com comprimentos diferentes.

Figura 18: Experimento da ponte dupla.

Ao andarem pelo ambiente, algumas formigas eventualmente atravessam a ponte e encontram o local com comida. Após a inicial movimentação aleatória várias formigas conseguem realizar o trajeto utilizando ambas as pontes. Em algum tempo, um comportamento interessante passa a ser observado: a tendência de mais formigas seguirem por um dos caminhos. Esta tendencia aumenta cada vez mais, até que todas as formigas passam a seguir o caminho pela mesma ponte. Deneubourg repetiu este experimento inúmeras vezes e observou que cada uma das pontes foi utilizada em 50% dos casos. Não há, a principio, preferência por um dos caminhos. A partir do momento em que um maior número de formigas passa por uma das pontes, devido principalmente a irregularidades aleatórias do caminho, maior é a concentração do feromônio depositado. Isto torna o caminho mais atraente para as formigas seguintes, possibilitando a convergência da maioria das formigas para uma das pontes.

Uma variação do experimento da ponte dupla original foi realizado por Goss et al., usando desta vez com pontes de tamanhos diferentes (GOSS *et al.*, 1989). Com esta configuração, mostrada na Figura 18, as formigas escolhem o caminho mais curto em grande parte das execuções do experimento. Inicialmente, formigas escolhem igualmente ambos os caminhos. Contudo, as que optam pelo mais curto são capazes de ir e voltar até a fonte de alimento antes que as formigas que seguem pelo caminho mais longo. Assim, também a concentração de feromônio no caminho mais curto será maior a partir do instante em que as formigas concluem o trajeto de ida e volta.

O estudo dos experimentos de ponte dupla resultou em um modelo matemático para a probabilidade de uma formiga selecionar um dos caminhos, determinado pela Equação 1.

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} = p_2 - 1 \tag{1}$$

Em um determinado momento, o número de formigas que atravessam o caminho 1 e o caminho 2 é representado por m_1 e m_2, respectivamente. Desta forma a probabilidade de uma formiga escolher o caminho 1, em um instante de tempo imediatamente após, será o valor p_1 da Equação 1. O parâmetro k define a atração que uma área inexplorada exerce sobre as formigas, enquanto que h representa o quanto a decisão será baseada no feromônio sentido pelas formigas. Simulações de Monte Carlo mostraram que valores de $k \approx 20$ e $h \approx 2$ são os mais compatíveis com o comportamento real das formigas observado experimentalmente (PASTEELS; DENEUBOURG; GOSS, 1987). O modelo descrito pela Equação 1, contudo, não representa o processo de evaporação do feromônio, considerando-o apenas proporcional a quantidade de formigas que previamente passam por um dos caminhos.

3.2 Computação com formigas artificiais

O conceito básico de como formigas conseguem encontrar um menor caminho baseando-se em feromônio, somado ao modelo matemático formulado por Goss et al., tem servido de inspiração para algoritmos computacionais de otimização em grafos.

Um grafo é uma abstração usada na modelagem de diversos tipos de problemas. É uma representação composta por um conjunto de elementos, onde pares destes elementos podem estar conectados. Cada elemento é abstração matemática chamada de *vértice*; as conexões entre vértices são chamadas de *arcos*, e representam alguma característica que relaciona dois elementos. Assim, um grafo é descrito pelo par ordenado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arcos.

O ambiente do experimento da ponte dupla pode ser modelado através de grafos. As Figuras 19(a) e 19(b) mostram duas representações para a situação em que as pontes possuem mesmo tamanho. Em (a), as duas pontes são ilustradas como os vértices V_5 e V_6. Em (b), as pontes são representadas pelos dois arcos entre os vértices V_2 e V_3. Por se tratarem de pontes iguais, o valor associado aos dois arcos são iguais (representado na figura pelo valor numérico 2). Já a representação do caso com pontes de tamanhos diferentes é ilustrada pelas Figuras 19(c) e 19(d). No primeiro grafo, a ponte mais extensa é representada por dois vértices, $V6$ e $V7$. No segundo, cada arco possui um valor diferente associado, de forma a poderem ser diferenciados.

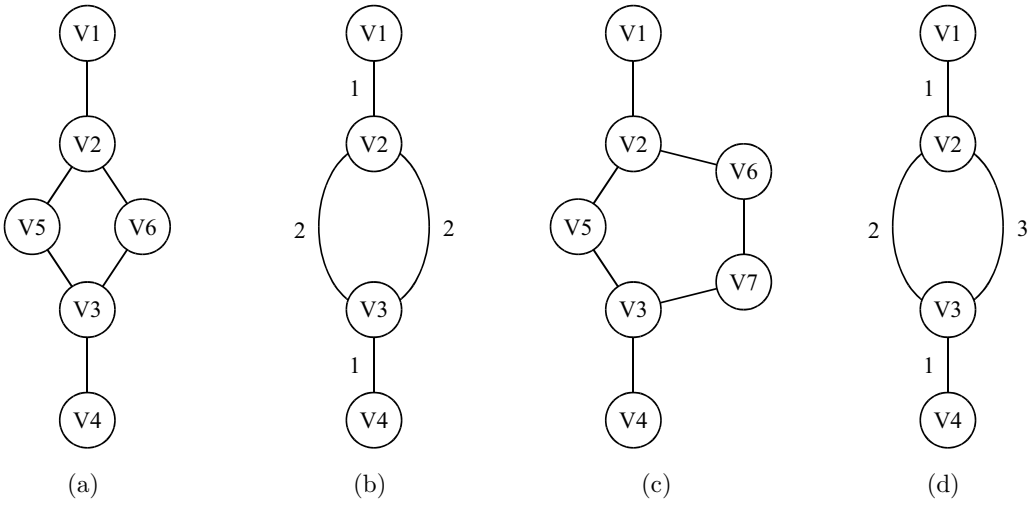


Figura 19: Grafos representando o experimento da ponte dupla.

Análogo ao modelo natural, um algoritmo de colônia é composto por inúmeras *formigas artificiais*. Estas formigas carregam as características de formigas reais que são interessantes

na solução de problemas em grafos, como a capacidade de sentir e alterar o feromônio do ambiente, e de decidir qual entre vários caminhos seguir. Contudo, existem algumas importantes diferenças entre formigas reais e artificiais, tais como:

- Formigas artificiais existem em um ambiente de discreto. Estas se deslocam sequencialmente através de um conjunto finito de estados do problema.
- O equivalente artificial do feromônio pode funcionar de forma diferente do feromônio natural. O depósito de feromônio pode ser realizado apenas por um grupo restrito de formigas, por exemplo. Este feromônio pode estar relacionado não apenas a quantidade de formigas, mas ser também função da qualidade da solução encontrada.
- Formigas artificiais possuem alguma memória, como o histórico de todo o caminho percorrido; também não são completamente cegas, podendo saber, quando em um vértice, qual o valor associado aos arcos para vértices adjacentes.

Algoritmos baseados em formigas artificiais são iterativos. Uma certa quantidade de formigas é espalhada no grafo de interesse, como o ilustrado pela Figura 19. A cada unidade de tempo, elas se movimentam de um vértice para outro. Na primeira iteração, este movimento é puramente aleatório, devido a ausência de feromônio no ambiente. Após todas as formigas construírem um caminho entre os vértices desejados, uma nova iteração é iniciada, com outro grupo sendo inserido no grafo. Desta vez porém, cada arco do grafo possui uma certa quantidade de feromônio associada, o que passará a influenciar na escolha das formigas. Quando chegam ao vértice com uma bifurcação (pontes), as formigas precisam decidir por qual caminho seguir. A lógica de tomada de decisão entre dois caminhos, apresentada pelo Algoritmo 3 utiliza o modelo probabilístico da Equação 1. Em um número finito de iterações, o melhor caminho será encontrado, e todas as formigas irão segui-lo.

3.3 Principais Algoritmos

Com o modelo que agrega as características do problema (grafo) e os agentes que irão resolvê-lo (formigas artificiais), inúmeros algoritmos de otimização puderam ser desenvolvidos. Aqueles que obtiveram maior aceitação estão listados nas Seções 3.3.1 e 3.3.2.

3.3.1 Ant System

Durante a década de 1990, Marco Dorigo desenvolveu uma série de estudos envolvendo o uso computacional do comportamento de formigas. O primeiro algoritmo com grande repercussão

Algoritmo 3 Lógica de seleção entre caminhos 1 e 2

```
1: r ← aleatório entre 0 e 1;
2: Calcula P_A usando a Equação 1;
3: Se r ≤ P_A Então
4:    Escolhe o caminho 1;
5: Senão
6:    Escolhe o caminho 2;
7: Fim Se
```

foi chamado de *Ant System*, proposto para a solução do *problema do caixeiro viajante* (DORIGO; COLORNI; MANIEZZO, 1991), (DORIGO; MANIEZZO; COLORNI, 1996). Este foi escolhido por se tratar de problema NP-difícil amplamente estudado e comumente usado como *benchmark* para algoritmos de otimização combinatória.

3.3.1.1 O problema do caixeiro viajante

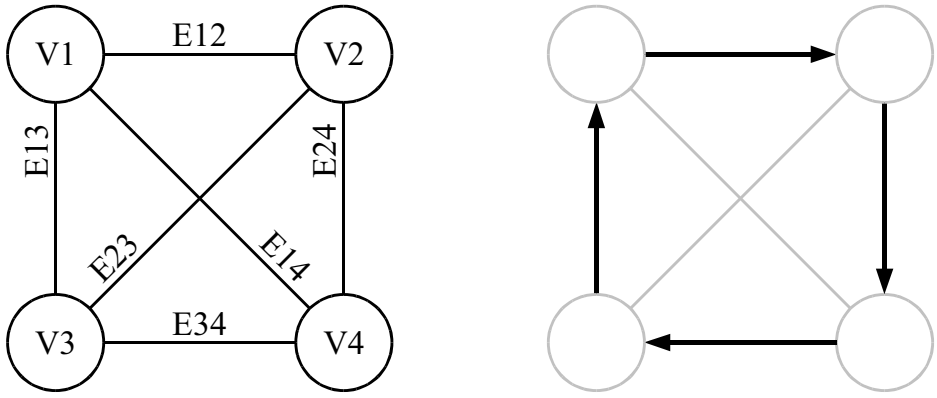
O problema do caixeiro viajante, também conhecido por TSP (do inglês *Travelling Salesman Problem*), considera um vendedor que deve visitar um conjunto de N cidades, sabendo-de a a distancia entre todas elas. Visando economizar o tempo em sua viagem, deve planejar um percurso tal que a soma da distância entre pares de cidades seja mínimo. Em termos mais formais, o objetivo é encontrar um circuito Hamiltoniano de menor extensão em um grafo totalmente conectado. Um circuito Hamiltoniano é percurso fechado em que se visita todos os vértices de um grafo exatamente uma vez. Pode ser representado por um grafo completo $G(V, E)$, onde o conjunto de vértices E representa o conjunto de cidades, enquanto que o conjunto de arcos V é o conjunto com as distâncias entre todas as cidades. A distância d_{ij} é a distância Euclidiana entre as cidades, definida pela Equação 2.

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2}$$

Um exemplo simples de grafo representando o TSP para quatro cidades é visto na Figura 20.

3.3.1.2 Construção dos caminhos

No *Ant System*, m formigas são espalhadas pelo conjunto de vértices do grafo que caracteriza o TSP. A cada arco está a associado à variável $\tau_{ij}(t)$, feromônio relativo ao caminho entre as cidades i e j. Cada formiga se movimenta de um vértice para o outro; a cada transição, o arco que conduz ao vértice de destino é acrescentado a solução. O processo se repete até que cada formiga tenha construído uma solução, iniciando assim uma nova iteração ou ciclo do algoritmo.



(a) Representação das cidades (b) Possível ciclo hamiltoniano.

Figura 20: Grafo representando o TSP para quatro cidades.

Para forçar a construção de apenas percursos válidos do TSP, cada formiga possui consigo uma estrutura de dados chamada de *lista tabu*. Trata-se de uma lista com todos os vértices já visitados pela formiga durante a iteração atual. Desta forma, estes vértices estarão proibidos de serem usados novamente pela mesma formiga na mesma solução - o que é uma restrição do próprio enunciado do TSP. Com o fim de uma iteração, a lista de cada formiga é reiniciada.

3.3.1.3 A atualização do feromônio

Quando todas as formigas terminam a construção de seu respectivo ciclo, o feromônio de cada arco é atualizado de acordo com a Equação 3.

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij} \tag{3}$$

Esta atualização é composta de dois termos: um associado a realimentação negativa e outro associado a realimentação positiva. A realimentação negativa modela o processo de evaporação do feromônio. O parâmetro ρ é chamado de *coeficiente de evaporação* e indica qual a quantidade do feromônio de uma iteração será descartada para a iteração seguinte. Já a realimentação positiva descreve o depósito de feromônio de todas as formigas que utilizaram cada arco. Como pode ser visto na Equação 4, a parcela $\Delta\tau_{ij}$ é composto pela soma do feromônio de todas as formigas que atravessaram o arco (i, j).

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{4}$$

A contribuição individual de cada formiga é definida pela Equação 5. O parâmetro Q é uma constante, enquanto que L_k é o custo total do percurso encontrado pela formiga k. No caso do TSP, L_k será a soma do comprimento de todos os arcos percorridos pela formiga. Assim, a quantidade Q/L_k está diretamente relacionada com a qualidade do resultado: quanto maior o percurso encontrado por uma formiga, menor será a sua contribuição de feromônio.

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{se a formiga k usa o arco (i,j)} \\ 0 & \text{caso contrário} \end{cases} \tag{5}$$

Durante o desenvolvimento do *Ant System*, três formatos de depósito de feromônio formam propostos *ant-cycle*, *ant-density* e *ant-quantity*. O algoritmo *ant-cycle* é versão do *Ant System* que utiliza a Equação 5. Já os algoritmos *ant-density* e *ant-quantity* possuem , respectivamente, as Equações 6 e 7 como reforço de feromônio.

$$\Delta\tau_{ij}^{k} = \begin{cases} Q & \text{se a formiga k usa o arco (i,j)} \\ 0 & \text{caso contrário} \end{cases} \tag{6}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/d_{ij} & \text{se a formiga k usa o arco (i,j)} \\ 0 & \text{caso contrário} \end{cases} \tag{7}$$

Experimentos mostraram que tanto o *ant-density* quanto o *ant-quantity* apresentaram resultados inferiores aos obtidos com o *ant-cycle* (DORIGO; MANIEZZO; COLORNI, 1996). No algoritmo *ant-density*, o rastro de feromônio de todas as formigas é igual a quantidade Q. Desta forma, não é possível fazer uma distinção entre os resultados melhores e os piores. No *ant-quantity*, a quantidade depositada é função de d_{ij}, a distância entre as cidades i e j. Trata-se de uma informação local, logo também este método não será capaz de diferenciar a qualidade das soluções. No *ant-cycle*, por sua vez, o depósito de cada formiga é função de uma informação global do problema: a solução encontrada por cada formiga. De fato, formigas que encontram uma solução com menor custo depositam uma maior quantidade de feromônio do que as que descreveram um percurso de maior custo. Os modelos *ant-density* e *ant-quantity* não mais foram alvo de estudo, sendo a versão definitiva do *Ant System* baseada no *ant-cycle*.

O modelo de atualização do feromônio do ACO difere do modelo do experimento da ponte dupla. Para o experimento, o feromônio está relacionado apenas com a quantidade de formigas que passam por cada ponte. No modelo probabilístico da Equação 1, por exemplo, não há nenhuma referência ao processo de evaporação do feromônio. Segundo (GOSS *et al.*, 1989), o tempo para a evaporação de uma parcela significativa do feromônio é da mesma ordem do tempo de busca pelo caminho , não sendo portanto relevante no modelo probabilístico.

Contudo, no contexto do ACO, a evaporação é um conceito importante. Se apenas a realimentação positiva for considerada, o algoritmo pode chegar em um estado de *estagnação* em um mínimo local. Em outras palavras, quanto mais feromônio for depositado em uma região, mais esta será interessante para futuras formigas, que também reforçarão o feromônio desta região. Eventualmente o feromônio de uma das soluções pode alcançar valores tão altos a ponto desta solução ser sempre selecionada. Nesta situação, o algoritmo perde a capacidade de encontrar outras soluções, sejam elas melhores ou piores. Visando evitar uma rápida convergência para um resultado sub-ótimo, é utilizada a realimentação negativa. Com a evaporação do feromônio, parte da experiência adquirida pelo algoritmo se perde, possibilitando a busca por novos resultados.

3.3.1.4 Probabilidade de transição

Durante a construção de uma solução, formigas selecionam a próxima cidade a ser visitada através de um mecanismo estocástico. Em um movimento uniformemente aleatório, todos as cidades adjacentes à cidade atual possuem exatamente a mesma probabilidade de serem escolhidos. Em um movimento estocástico, a decisão de qual cidade será a próxima do percurso está sujeita a uma distribuição de probabilidade. A Equação 8 mostra a probabilidade de uma formiga k localizada na cidade i selecionar a cidade j como parte do percurso.

$$p_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}(t)^{\alpha} \cdot {\eta_{ij}}^{\beta}}{\sum_{k \in permitido_k} \tau_{ik}(t)^{\alpha} \cdot {\eta_{ik}}^{\beta}} & \text{se } j \in permitido_k \\ 0 & \text{senão} \end{cases} \tag{8}$$

O conjunto $permitido_k$ é constituído por todos os vértices adjacentes a i que permite a construção de um percurso válido. Os vértices que estiverem na lista tabu não fazem parte deste conjunto.

O termo η_{ij} é uma informação heurística do problema. É chamado de *visibilidade*, e como pode ser visto na Equação 9, é o inverso da distância entre duas cidades.

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{9}$$

Uma formiga localizada na cidade i pode se deslocar para as cidades contidas no conjunto $permitido_k$; a cidade que estiver mais próxima será a com maior visibilidade, sendo intuitivamente a escolha mais provável. Contudo, também o feromônio dos arcos que ligam a cidade atual às adjacentes influi nesta seleção. Os parâmetros que controlam a importância relativa do feromônio e da visibilidade são α e β, respectivamente. Se $\alpha > \beta$, o rastro de feromônio terá uma maior influência na tomada de decisão do que a visibilidade.

3.3.1.5 O critério de parada

Após todas as formigas realizarem a busca, um novo ciclo é iniciado. O processo, como pode ser visto no Algoritmo 4, é repetido até que se alcance o critério de parada estabelecido. Dentre os possíveis critérios de parada para o *Ant System*, podem ser citados:

- Limitar o algoritmo a um valor máximo de ciclos. Tem-se controle do tempo total de execução, mas não é garantido que se alcance a melhor solução possível.
- Interromper a execução quando o algoritmo encontrar uma solução melhor que um limiar pré-estabelecido. Porém, nesta situação, não se sabe quantos ciclos serão necessários até se obter o resultado esperado.
- Interromper a execução quando todas as formigas estiverem seguindo o mesmo caminho. Neste caso, é dito que o algoritmo convergiu para uma mesma solução.

O *Ant System* possui inúmeras implementações na solução de problemas de otimização combinatória. A primeira e mais conhecida é a solução para o problema do caixeiro viajante, apresentada no Algoritmo 4. Outras implementações podem ser listadas para solução de diferentes tipos de problemas, tais como:

- AS-QAP (MANIEZZO; COLORNI, 1999) *Ant System* para a solução do problema de atribuição quadrática (*Ant System Quadratic Assignment Problem*) .
- AS-JSP (COLORNI *et al.*, 1994) *Ant System* para a solução do problema de agendamento de processos (*Ant System Job-shop Scheduling Problem*) .
- AS-VRP (BULLNHEIMER; HARTL; STRAUSS, 1999) *Ant System* para a solução do problema de roteamento de veículos (*Ant System Vehicles Routing Problem*) .

3.3.2 Variações do Ant System

O *Ant System* foi comparado a outras heurísticas de uso geral, como algoritmos genéticos e *simulated annealing* (BONABEAU; DORIGO; THERAULAZ, 1999). Foi obtido desempenho semelhante na solução de versões do TSP com até 30 cidades. Contudo, em problemas com maior dimensão (50 a 75 cidades), o *Ant System* não conseguiu encontrar a melhor solução conhecida em um tempo de busca limite de 3000 ciclos, sendo este um desempenho inferior a outros algoritmos específicos para o TSP. Contudo, o *Ant System* serviu de inspiração para outros diversos algoritmos. De fato, vários destes são extensões do *Ant System* contendo alterações que permitiram melhorias em seu desempenho.

Algoritmo 4 O Ant System para solução do TSP

```
1: Para todo arco (i,j) Faça
2:    Inicializa o feromônio do arco (i,j) com um valor aleatório baixo;
3: Fim Para;
4: Para k = 1 → m Faça
5:    Escolhe aleatoriamente a cidade inicial da formiga k;
6:    Adiciona a cidade inicial à lista tabu;
7: Fim Para;
8: Enquanto Condição de parada não alcançada Faça
9:    Para k = 1 → m Faça
10:      Escolhe a próxima cidade com base na Equação 8;
11:      Adiciona a próxima cidade à lista tabu;
12:   Fim Para;
13:   Para k = 1 → m Faça
14:      Calcula o rastro de feromônio da formiga k usando a Equação 5;
15:      Se Solução da formiga k < Melhor resultado Então
16:         Melhor resultado ← Solução da formiga k;
17:      Fim Se;
18:      Apaga a lista tabu da formiga k;
19:   Fim Para;
20:   Para todo arco(i, j) Faça
21:      Atualiza o feromônio do arco (i,j) usando as Equações 4 e 3;
22:   Fim Para;
23: Fim Enquanto;
24: Retorna Melhor resultado;
```

3.3.2.1 Elitist Ant System

A primeira modificação do algoritmo *Ant System* foi o uso de elitismo no processo de atualização do feromônio. Este algoritmo ficou conhecido como *Elitist Ant System* (EAS) (DORIGO; MANIEZZO; COLORNI, 1996). A chamada *estratégia elitista* consiste em reforçar a cada ciclo o feromônio associado à melhor solução encontrada desde o início da execução do algoritmo. A atualização do feromônio é definida pela Equação 10.

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij} + \Delta\tau^{*}_{ij} \tag{10}$$

A parcela $\Delta\tau^{*}_{ij}$ é o depósito de feromônio referente às *formigas elitistas*. Seu valor é calculado pela Equação 11.

$$\Delta\tau^{*}_{ij} = \begin{cases} e\cdot Q/L^{*} & \text{se o arco (i,j) pertence a melhor solução} \\ 0 & \text{caso contrário} \end{cases} \tag{11}$$

O parâmetro e indica a quantidade de formigas elitistas que serão usadas, enquanto que o termo L^{*} é a melhor solução encontrada. Se nenhuma formiga elitista for usada ($e = 0$), a Equação 10 se torna igual a Equação 3. Sem elitismo, o processo de atualização do feromônio é igual ao do *Ant System*.

Resultados obtidos por Dorigo et al. indicam que o uso da estratégia elitista com um valor apropriado do parâmetro e permite encontrar melhores soluções e em uma menor quantidade de iterações (DORIGO; MANIEZZO; COLORNI, 1996).

3.3.2.2 Rank-Based Ant System

O algoritmo proposto por Bullnheimer et al., *Rank-Based Ant System* (AS_{rank}), adiciona a ideia de classificação ou *ranking* às formigas (BULLNHEIMER; HARTL; STRAUSS, 1997). Após a construção das soluções, todas as formigas são classificadas de acordo com a qualidade do resultado obtido. A quantidade de feromônio depositado estará ponderado pelo *rank* μ obtido por cada formiga. Além disso, apenas as ω melhores formigas são consideradas. O depósito de feromônio do AS_{rank} é definido pela Equação 12.

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij} + \Delta\tau^{*}_{ij} \tag{12}$$

O termo $\Delta\tau_{ij}$ é o feromônio associado às ω formigas mais bem classificadas durante a iteração (Equações 13 e 14). Já $\Delta\tau^{*}_{ij}$ é o feromônio da formiga com melhor resultado desde o início do algoritmo (Equação 15).

$$\Delta\tau_{ij} = \sum_{\mu=1}^{\sigma-1} \Delta\tau^{\mu}_{ij} \tag{13}$$

O parâmetro σ é o peso do rastro de feromônio da melhor solução encontrada. O peso associado às demais formigas mais bem classificadas é $(\sigma - \mu)$; já a quantidade de formigas consideradas na atualização do feromônio é definida como sendo $\omega = \sigma - 1$. Assim, os pesos variam desde o valor σ para a formiga com a melhor solução de todas até o valor "1" para a formiga com pior classificação.

$$\Delta\tau^{\mu}_{ij} = \begin{cases} (\sigma-\mu)\cdot Q/L_{\mu} & \text{se a formiga com rank } \mu \text{ usa o arco (i,j)} \\ 0 & \text{caso contrário} \end{cases} \tag{14}$$

$$\Delta\tau^{*}_{ij} = \begin{cases} \sigma\cdot Q/L^{*} & \text{se o arco (i,j) pertence a melhor solução} \\ 0 & \text{caso contrário} \end{cases} \tag{15}$$

O AS_{rank} obteve resultados um pouco melhores que o *Ant System* e o *Elitist Ant System*. De fato, a restrição quanto ao depósito apenas pelas formigas mais bem classificadas evita o espalhamento de feromônio por formigas usando caminhos sub-ótimos (BULLNHEIMER; HARTL; STRAUSS, 1997).

3.3.2.3 MAX-MIN Ant System

O algoritmo *MAX-MIN Ant System* (MMAS) acrescenta várias melhorias ao *Ant System* original (STÜTZLE, 1997). No MMAS, apenas a formiga que obteve a melhor solução desde o início da execução do algoritmo (ou a que obteve a melhor solução da iteração) está apta a depositar feromônio durante a atualização. Contudo, esta estratégia pode levar rapidamente o algoritmo à estagnação, devido ao grande depósito de feromônio em um caminho sub-ótimo. Para evitar esta situação, o feromônio está restrito a um limite superior e inferior. Assim, este só pode assumir valores no intervalo $[\tau_{min}, \tau_{max}]$. Outra diferença está no fato do feromônio ser inicializado por seu limite superior. Esta característica, em conjunto com uma baixa taxa de evaporação, aumenta a exploração de caminhos ao início da busca. Por fim, no MMAS, os rastros de feromônio são reinicializados todas as vezes que o algoritmo se aproxima da estagnação ou passa por uma certa quantidade de ciclos sem encontrar uma solução melhor. A atualização do feromônio para o MMAS é definida pela Equação 16.

$$\tau_{ij}(t+1) = [(1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}^{best}]_{\tau_{min}}^{\tau_{max}} \tag{16}$$

A parcela τ_{ij}^{best} é o depósito de feromônio da formiga que obteve o melhor resultado em todas as iterações ou o melhor resultado da iteração atual, (Equação 17). Se o valor obtido pela Equação 16 ultrapassar limite superior, será arredondado para τ_{max}. Da mesma forma, se este valor estiver abaixo do limite inferior, será arredondado para τ_{min}.

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L^{best} & \text{se o arco (i,j) pertence a melhor solução} \\ 0 & \text{caso contrário} \end{cases} \tag{17}$$

Em geral, implementações do MMAS usam tanto o melhor valor global quanto o melhor valor da iteração são usados; quanto maior for a utilização do melhor valor global, mais direcionada será a busca (DORIGO; STÜTZLE, 2004).

3.3.2.4 Ant Colony System

As principais contribuições do *Ant Colony System* (ACS) estão na diferente regra de tomada de decisão para a construção das soluções, e no uso de uma atualização local do feromônio em acréscimo à atualização ao fim da construção de todas as soluções (DORIGO; GAMBARDELLA, 1997).

A regra para transição de estado do ACS é vista na Equação 18, chamada de *regra pseudo-aleatória proporcional*.

$$j = \begin{cases} argmax_{l \in J_k(i)} \left\{ \tau_{il} \cdot \eta_{il}^{\beta} \right\} & \text{se } q \leq q_0, \\ S & \text{caso contrário} \end{cases} \tag{18}$$

Seja uma formiga localizada em um vértice i. Deseja-se encontrar o vértice seguinte j pertencente ao conjunto de possíveis vértices adjacentes $J_k(i)$. A seleção pode ser feita de duas formas diferentes. Na primeira, escolhe-se como próximo vértice k aquele que, entre os vértices possíveis, retorna o maior valor do produto $(\tau_{ik} \cdot \eta_{ik}^{\beta})$. Na segunda forma de seleção, o vértice é escolhido de forma aleatória seguindo uma distribuição de probabilidade. A probabilidade de cada um dos vértices possíveis ser escolhido é calculada usando a Equação 8, considerando o parâmetro $\alpha = 1$.

A escolha por qual das duas opções de seleção depende da variável q e do parâmetro q_0. O valor de q é escolhido aleatoriamente no intervalo [0,1], seguindo uma distribuição uniforme de probabilidades. O parâmetro q_0 define a importância relativa de cada tipo de seleção.

Outra característica do ACS é o uso de duas atualizações do feromônio. A atualização global é semelhante à usada pelo MMAS, exceto pela não limitação do feromônio. Ao fim da construção de soluções por todas as formigas, o feromônio evapora com taxa $(1 - \rho_{global})$ e recebe o reforço apenas da formiga com melhor solução desde o início do algoritmo. Este processo de atualização global pode ser visto na Equação 19.

$$\tau_{ij}(t+1) = (1 - \rho_{global}) \cdot \tau_{ij}(t) + \rho_{global} \cdot \Delta\tau_{ij}^{best} \tag{19}$$

A atualização local é realizada de forma independente por cada uma das formigas. Em cada passo realizado (deslocamento de uma formiga do vértice i para o vértice j) a atualização descrita pela Equação 20 é aplicada ao último arco (i, j) visitado. O parâmetro ρ_{local} é o coeficiente de evaporação local, e valor τ_0 é o feromônio inicial de cada arco.

$$\tau_{ij}(t+1) = (1 - \rho_{local}) \cdot \tau_{ij}(t) + \rho_{local} \cdot \tau_0 \tag{20}$$

A principal ideia ao usar a atualização local é diversificar a busca realizada pelas formigas seguintes. Ao reduzir a concentração do feromônio, a formiga da iteração atual força as subsequentes a escolherem outros caminhos, uma vez que os demais arcos não sofrem atualização.

3.4 A meta-heurística ACO

A Otimização por Colônia de Formigas foi formalizada como uma heta-heurística por Dorigo et al. (DORIGO; DI CARO, 1999). Meta-heurísticas são métodos computacionais que otimizam um determinado problema de forma iterativa, tentando melhorar uma solução candidata através da avaliação de sua qualidade. Geralmente são aplicadas a problemas em que não se conhece

nenhum algoritmo eficiente para sua solução. Utilizam escolhas aleatórias e o conhecimento histórico adquirido de resultados anteriores na construção de soluções dentro de um espaço de busca. Outros exemplos de meta-heurísticas incluem *simulated annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) , busca tabu (GLOVER, 1986), algoritmos genéticos (HOLLAND, 1975) e a otimização por enxame de partículas (KENNEDY; EBERHART, 1995).

Em geral, algoritmos ACO são usados na solução de problemas de otimização combinatória. Tais problemas são caracterizados por:

- um espaço de busca discreto;
- um conjunto finito de componentes;
- um conjunto finito de transições entre componentes;
- um conjunto finito de restrições;
- um conjunto finito de sequências de componentes, definindo assim o espaço de busca; uma sequência é possível se não for contra o conjunto de restrições;
- uma solução, representada por uma sequência ordenada de componentes;
- uma função custo, que associa um custo a cada solução;

Dadas estas características do problema, o propósito dos algoritmos de otimização é encontrar uma solução possível com custo mínimo (para o caso de um problema de minimização).

Como pôde ser visto na Seção 3.3, os diversos algoritmos de formigas compartilham todos características semelhantes. Este conjunto de conceitos algorítmicos é definido como a meta-heurística ACO, podendo ser usado na construção de novos métodos heurísticos aplicáveis a outros problemas. Em outras palavras, é um arcabouço algorítmico de uso geral, que pode ser usado em diferentes problemas de otimização necessitando apenas de uma pequena quantidade de modificações (DORIGO; BIRATTARI; STÜTZLE, 2006).

A meta-heurística ACO é apresentada no Algoritmo 5. Três ações são realizadas enquanto o critério de parada não é alcançado. A *construção de soluções* inicializa as formigas e escolhe a sequência de arcos que formará uma solução seguindo uma regra de transição de estados. A *busca local* realiza uma comparação entre soluções a fim de encontrar as melhores, como é o caso do algoritmo AS_{rank}. Por ser um processo útil, mas que não é usado em todos os algoritmos de formigas, é apresentado na meta-heurística como sendo uma etapa opcional.

Algoritmo 5 A meta-heurística ACO

```
1: Inicializa os parâmetros e o feromônio;
2: Enquanto Condição de parada não alcançada Faça
3:    Constrói as soluções de todas as formigas do ciclo;
4:    Aplica a busca local (opcional);
5:    Atualiza o feromônio;
6: Fim Enquanto;
```

Por fim, a *atualização* define como o feromônio irá variar no decorrer da execução do algoritmo. É interessante ressaltar que a meta-heurística não define como estas três etapas devem ser realizadas. Assim, o projetista tem liberdade para alterar instâncias - tais como as regras de transição, a atualização de feromônio ou ainda o critério de parada - a fim de obter um algoritmo que solucione de forma mais eficiente o problema de interesse.

3.5 Considerações Finais do Capítulo

Neste capítulo foi apresentada a otimização por colônia de formigas (ACO) como um método computacional de inteligência coletiva. Foi brevemente abordado o comportamento natural de formigas que serviu de inspiração para o *Ant System* e, posteriormente, algoritmos como o ACS e o MMAS. Cada um destes algoritmos foi abordado. Por fim, o ACO foi formalizado como uma meta-heurística de otimização. O capítulo seguinte apresenta dois algoritmos inspirados em ACO sendo usados para o roteamento em redes embutidas.

Capítulo 4

ROTEAMENTO EM NOCS BASEADO EM ACO

A COMUNICAÇÃO entre blocos de um SoC é um fator de grande importância no projeto do sistemas. Um modelo de comunicação mal planejado pode acarretar em uma perda de eficiência do sistema como um todo, devido a atrasos na execução da aplicação. Como pôde ser visto no Capítulo 1, NoCs passaram a ser usadas com o objetivo de desviar destes problemas de comunicação, reduzindo assim o tempo de execução de aplicações embutidas. Ainda assim, NoCs também podem apresentar atrasos, em situações de congestionamento da rede. Uma forma de evitar, ou pelo menos reduzir o congestionamento, é utilizando um roteamento de rede adaptativo. Desta forma, o caminho a ser percorrido por uma mensagem é definido levando-se em consideração a carga de mensagens em toda rede.

Algoritmos baseados na meta-heurística ACO têm se mostrado eficientes na otimização de problemas envolvendo grafos, conforme visto no Capítulo 3. Esta ideia pode ser utilizada também em problemas de busca de rotas entre vértices de grafos.

Este capítulo apresenta o modelo proposto para um roteamento centralizado em NoCs inspirados em ACO. A Seção 4.1 mostra as especificações e modelagem da rede para utilização do método proposto. A Seção 4.2 ilustra o problema de roteamento, em que se deseja minimizar o tempo de transmissão de pacotes em uma rede. Os dois algoritmos implementados, baseados no EAS e no ACS, são detalhados na Seção 4.3.

4.1 Simulação da rede embutida

Em projetos de redes embutidas, simulações e testes são necessários para a avaliação de desempenho. Quando se deseja conhecer todo o comportamento do sistema, é necessário o desenvolvimento completo da rede, preferencialmente em uma HDL (*hardware description language*, ou linguagem de descrição de *hardware*). De posse de um modelo sintetizável, pode-se ter

um protótipo em FPGA (*Field Program Grid Array*), que deverá apresentar o comportamento esperado do sistema de interesse. Embora seja uma metodologia que permite uma análise extremamente acurada, a simulação em modelos em HDL - e posteriormente, em modelos sintetizados - pode não ser uma boa solução em projetos que visam o estudo do impacto de uma característica específica da rede. Isto se deve à complexidade de um projeto completo de um sistema baseado em NoCs, que eventualmente necessita de um elevado tempo de modelagem e implementação. Em uma situação como esta, é mais interessante uma modelagem em um nível de abstração mais elevado. Assim, pode-se empregar uma linguagem de programação de alto nível, restringindo a simulação a apenas parâmetros que efetivamente influem no comportamento da característica de interesse.

Nesta dissertação foi desenvolvido um modelo de simulação usando o software Matlab (MATHWORKS, 2008). A base do código fonte - responsável pelas principais características da rede - foi empregada em testes envolvendo quatro métodos de roteamento: os algoritmos inspirados em EAS e ACS propostos neste trabalho, e os algoritmos XY e *Odd-Even*, amplamente usados na literatura como base de comparação.

4.1.1 Especificações da rede

A chave idealizada para a rede é composta por cinco portas, sendo quatro para comunicação com outras chaves (Norte, Sul Oeste e Leste) e uma para comunicação local com o recurso. Cada porta é constituída por dois canais de comunicação unidirecionais em sentidos inversos - um canal transmitindo dados para a chave e outro recebendo dados vindos da chave - conforme visto no diagrama da Figura 21. Por haver cinco canais de entrada, a chave é capaz de manipular *flits* de até cinco pacotes simultaneamente. Assume-se que cada canal possui uma largura em *bits* de um *flit*, isto é, um *phit* equivale a um *flit*.

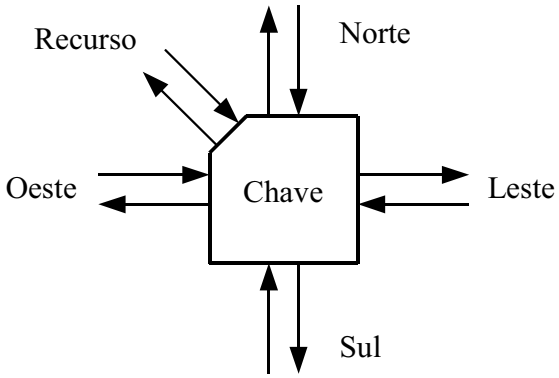


Figura 21: Modelo de chave para a rede simulada.

A chave não faz uso de canais virtuais ou *buffers*. Ao invés disso, utiliza-se um único elemento armazenador por canal de entrada, registrando um *flit* de cada pacote por ciclo de *clock*. Vale ressaltar que estes elementos atuam apenas como repetidores de *flits* entre nós de rede, não servindo de armazenamento para o caso de bloqueio da rede. Esta abordagem assemelha-se à chave sem *buffers* utilizada para o BPS (*Blind Packet Switching*) ou chaveamento cego de pacotes (GÓMEZ *et al.*, 2008).

A topologia de rede empregada nas simulações foi a malha 2D. A escolha por uma topologia de rede direta ortogonal acarreta em diversos parâmetros sendo caracterizados por matrizes. Este é um fator interessante para a modelagem, uma vez que o ambiente de desenvolvimento Matlab (do inglês *Matrix Laboratory*, ou laboratório de matrizes) é um software destinado justamente para cálculos envolvendo o uso de matrizes.

A técnica de chaveamento adotada foi o *wormhole*. Como apresentado na Seção 1.3.4, neste método uma mensagem é partida em grupos menores, chamados pacotes. Os pacotes são, por sua vez, divididos em unidades menores, chamadas *flits*. A transmissão dos *flits* pelos nós da rede é feita em *pipeline*, conforme visto na Figura 22.

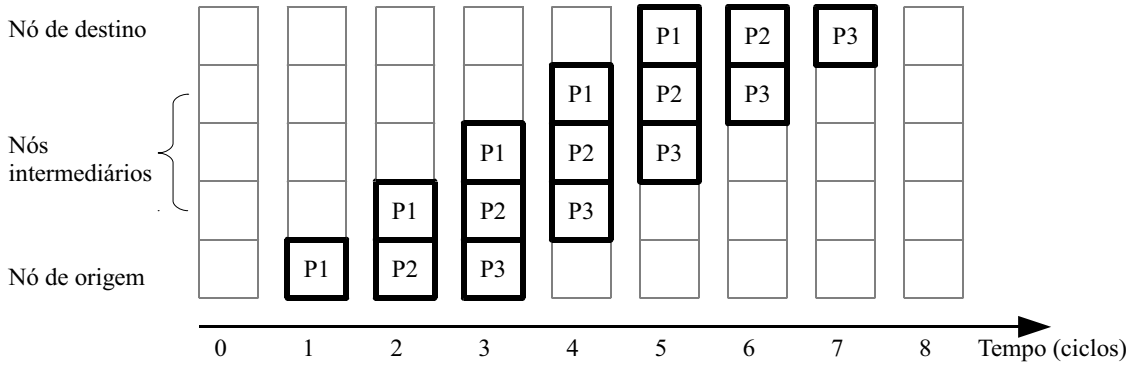


Figura 22: Transmissão de três pacotes com chaveamento *wormhole*.

Assume-se que a rede possui um controle de fluxo regido pelo controlador central. Quando o cabeçalho de um pacote é direcionado para um canal que já se encontra em uso (congestionamento), ele é imediatamente interrompido, bem como os demais *flits* nas chaves intermediárias. Se o recurso ainda estiver injetando dados na chave de origem, este também será cessado. A transmissão só é reiniciada quando o canal de saída na chave onde o cabeçalho está bloqueado for liberada.

Na situação de mais de uma porta de entrada da chave requisitando uma mesma porta de saída, é utilizada a política de arbitragem FCFS (*First-come, first-served*, o primeiro a

chegar é o primeiro atendido). Esta escolha de acesso permite que a ocorrência de *starvation* em uma porta particular seja evitada.

4.1.2 Latência da rede

A principal métrica de desempenho adotada neste trabalho é a latência. Define-se por latência de um pacote o tempo decorrido desde a injeção do cabeçalho na chave de origem até a chegada do último *flit* na chave de destino.

Em geral, a latência individual de cada pacote é de pouca importância, principalmente quando são utilizadas cargas sintéticas. Neste caso, o valor médio das latências é mais significativo para a avaliação da rede. Porém, se alguns pacotes experimentam latências muito maiores que os demais, isto pode ter um impacto importante no desempenho de algumas aplicações. Logo, além da latência média, o desvio padrão pode ser importante para ajudar a identificar estas situações.

A latência é medida em unidades de tempo. Porém, como muitas comparações de alternativas de projeto são realizadas utilizando-se simuladores de rede, a latência pode ser medida em ciclos de *clock* do simulador.

4.1.2.1 Modelo de latência sem carga

Um modelo de latência é apresentado em (DUATO; YALAMANCHILI; LIONEL, 2002) para redes ortogonais usando chaveamento *wormhole*. Esse modelo, visto na Equação 21, considera uma rede sem carga. Desta forma, tem-se um pacote apenas sendo transmitido de um nó de origem até um nó de destino.

$$T_{sem\ carga} = D \cdot (t_r + t_s + t_w) + max(t_s, t_w) \cdot \left\lceil \frac{L}{W} \right\rceil \tag{21}$$

A primeira parcela do lado direito da Equação 21 indica o tempo de envio através dos nó da rede, com D representando o número de chaves utilizadas e t_r, t_s e t_w sendo, respectivamente, o tempo de decisão do roteamento, o tempo de transmissão através da estrutura interna da chave e o tempo de transmissão através de um canal de comunicação. A segunda parcela define a transmissão entre recurso e chave, com L sendo o comprimento total em *bits* de um pacote e W a largura em *bits* do canal de comunicação. Assim, a fração L/W indica a quantidade de *phits* de um pacote. Segundo Duato et al., o tempo para transmissão em *pipeline* de um *phit* através de uma chave com *buffers* é o valor máximo entre t_s e t_w. Para uma chave sem *buffers* - ou com armazenamento de um *phit* apenas - este tempo pode ser substituído por $(t_s + t_w)$.

O modelo de rede adotado nesta dissertação considera o roteamento sendo feito de forma estática e centralizada: um controlador dedicado realiza os cálculos de roteamento uma única vez, antes da execução da aplicação. O tempo de roteamento nas chaves é, pois, igual a zero. Considerando também que a chave emprega armazenamento de um *phit* por canal de entrada, tem-se o modelo de latência descrito pela Equação 22.

$$T_{sem\ carga} = (t_s + t_w) \cdot \left(D + \left\lceil \frac{L}{W} \right\rceil \right) \tag{22}$$

Uma vez que o tempo de transmissão através da chave e o tempo de comunicação pelos dos canais são constantes, e considerando que a comunicação entre chaves é compatível com a comunicação entre chaves e recursos, o modelo de latência pode ser simplificado para a Equação 23.

$$T_{sem\ carga} = t_L \cdot \left(D + \left\lceil \frac{L}{W} \right\rceil \right) \tag{23}$$

O termo t_L é o tempo de *link* ou enlace, que engloba todo o período desde a injeção de um *phit* em um canal de comunicação até seu armazenamento na chave. Assume-se nesta dissertação que t_L é equivalente a um ciclo de *clock* da rede.

A Figura 22 pode ser tomada como exemplo, onde três pacotes estão sendo transmitidos através de cinco chaves da rede. Nesta situação, a latência é de 8 ciclos de *clock*.

4.1.2.2 Modelo de latência com carga

Zeferino define a latência de uma rede com carga como um somatório de quatro parcelas (ZEFERINO, 2003), conforme visto pela Equação 24.

$$T_{com\ carga} = S + O + A_{RC} + C \tag{24}$$

A sobrecarga S, ou *overhead*, refere-se aos tempos gastos pelos nós de origem e destino para injetar e retirar um pacote da rede. A ocupação do canal O é uma medida do tempo gasto para a transferência de um pacote através dos enlaces. O atraso de roteamento a chaveamento A_{RC} é o tempo gasto para a determinação da rota a ser utilizada para a propagação do pacote. E por fim, o atraso de contenção C diz respeito aos períodos de tempo durante os quais um pacote é impedido de avançar devido ao congestionamento da rede.

Comparando o modelo de Zeferino com o de Duato, observa-se que estes se diferenciam pela parcela C, o atraso de contenção da rede. Acrescentando à expressão da Equação 23 um termo equivalente, o modelo passa a representar a latência de uma rede com atrasos. A Equação 25 ilustra este modelo.

$$T_{com\ carga} = t_L \cdot \left(D + \left\lceil \frac{L}{W} \right\rceil + \left\lceil \frac{L_{atraso}}{W} \right\rceil \right) \tag{25}$$

O termo L_{atraso} indica a quantidade de *bits* de um pacote que ainda precisam atravessar uma chave para que esta esteja livre para transmissão de um outro pacote em espera. A fração L_{atraso}/W, então, indica a quantidade de *phits* que acarretam atraso em uma chave. Se um mesmo pacote experimentar atrasos em chaves diferentes da rede, L_{atraso} será composto pelos *bits* dos pacotes que acarretam em atraso em cada uma destas chaves.

4.1.3 Estrutura para simulação de roteamento

Esta seção apresenta uma estrutura básica para a simulação de algoritmos de roteamento em uma rede com as características previamente definidas.

Todos os algoritmos executados nesta dissertação apresentam uma estrutura semelhante a do Algoritmo 6. Um conjunto de pares origem-destino são considerados, onde cada par representa o nó de origem e o de destino de um pacote. O termo *Grupo* indica a quantidade total de pacotes injetados na rede durante a simulação. Pacotes podem ser inseridos todos em um mesmo tempo ou em tempos diferentes; contudo, é considerado que a partir de um certo tempo, todos os pacotes da simulação estarão sento transmitidos simultaneamente na rede. O objetivo da estrutura é retornar o percurso encontrado para cada pacote, bem como a sua latência.

Algoritmo 6 Estrutura de simulação da rede

1: *Grupo* $\leftarrow$ *nº de mensagens ou pacotes*;
2: *origem* $\leftarrow$ *coordenadas da chave de origem*;
3: *destino* $\leftarrow$ *coordenadas da chave de destino*;
4: **Para** $g = 1 \rightarrow Grupo$ **Faça**
5: **Enquanto** *chave de destino não alcançada* **Faça**
6: Escolhe a chave seguinte no $percurso_g$;
7: Calcula o custo do $percurso_g$ escolhido;
8: **Fim Enquanto**
9: **Fim Para**
10: **Retorna** Percurso encontrado;
11: **Retorna** Latência de cada percurso;

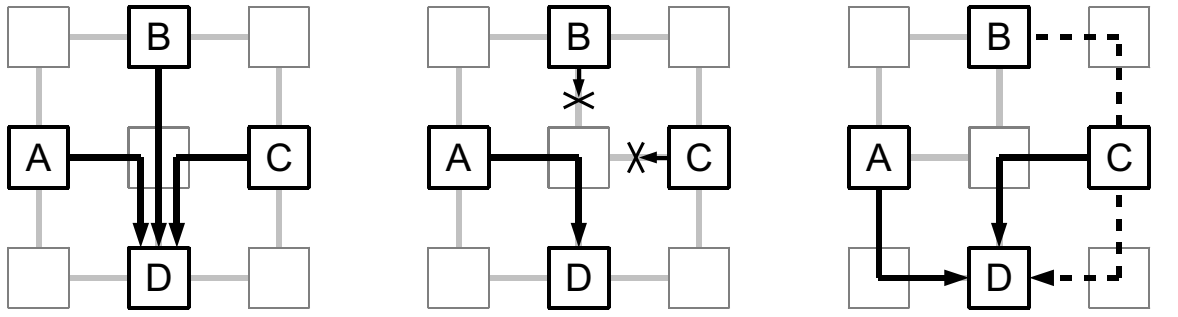
A base do Algoritmo 6 é a repetição da busca de um nó seguinte em um percurso até que se encontre o nó de destino. A escolha irá depender do método de roteamento empregado. Para algoritmos de roteamento determinísticos, a escolha depende unicamente dos dados do próprio percurso, como a posição dos nós de origem, destino, e do nó atual. Para um roteamento adaptativo, o Algoritmo 6 estará interno a uma estrutura iterativa, onde os percursos encontrados são otimizados a cada iteração considerando os demais caminhos encontrados.

4.2 O problema de roteamento de pacotes

A otimização do roteamento em NoCs visa a redução da latência de comunicação, permitindo assim um menor tempo de execução da aplicação embutida. Para um conjunto de pacotes sendo enviados simultaneamente por uma rede, dois objetivos principais podem ser definidos:

- O percurso de um pacote deve acarretar o menor atraso possível na transmissão dos demais pacotes.
- O percurso deve atravessar a menor quantidade possível de chaves entre origem e destino.

Estes são objetivos que podem eventualmente não ser alcançados por um algoritmo de roteamento. Em um roteamento determinístico, como o algoritmo XY, o percurso de cada pacote é definido pela localização dos nós de origem e destino. A Figura 23(a) ilustra uma malha 3x3 onde três pacotes, cujos nós de origem são A, B e C, devem ser enviados para o nó localizado em D. As setas indicam os percursos encontrados com o algoritmo XY. Os três pacotes precisam atravessar um mesmo canal de comunicação para alcançar o nó D. Uma vez que cada canal de comunicação suporta a transmissão de um único *flit* por ciclo de *clock*, apenas um pacote será transmitido (escolhido pela política de arbitragem), enquanto os demais ficam bloqueados, conforme visto na Figura 23(b). O pacote originado de B só chega em D após a transmissão de todos os *flits* enviados a partir de A; por sua vez, o pacote vindo de C só alcançará D após a transmissão dos *fits* de B. Embora os três percursos sejam mínimos, o algoritmo XY não pode evitar a ocorrência do congestionamento.



(a) Situação hipotética ideal.

(b) Pacotes bloqueados no algoritmo XY.

(c) Caminho não mínimo (tracejado) em um roteamento adaptativo.

Figura 23: Roteamento em uma malha 3x3.

Por outro lado, algoritmos de roteamento adaptativo podem ser empregados para evitar situações de congestionamento da rede. Tais algoritmos utilizam não apenas a posição dos nós

de origem e destino, mas também a condição de carga atual da rede no cálculo dos percursos. Ao encontrar uma região da rede em uso, o roteamento pode definir que o pacote siga por um outro trajeto. Este trajeto livre de congestionamento pode, contudo, não ser mínimo, como o exemplo da Figura 23(c).

4.2.1 Formulação do problema

A busca pelo roteamento ótimo em uma NoC pertence a classe dos *problemas de roteamento*. Mais precisamente, trata-se do caso particular de um *problema de roteamento de pacotes* (PEIS; SKUTELLA; WIESE, 2010). Dentre os problemas de otimização combinatória, problemas de roteamento são de complexidade NP-completo. Esta classe inclui ainda o *problema do caixeiro viajante*, o *problema de roteamento de veículos* e o *problema do caminho mais curto*.

Seja $G = (V, E)$ um grafo não direcionado e com pesos, onde V representa o conjunto de nós da rede e E o conjunto de enlaces. Este grafo, que representa a topologia da rede, também é conhecido por grafo de caracterização de arquitetura ou ARCG (*Architecture Characterization Graph*). Seja $M = \{M_1, M_2, M_3, \ldots, M_M\}$ o conjunto de pacotes injetados na rede. Um pacote $M_i = (S_i, D_i, flits_i)$ consiste de um nó de origem $S_i \in V$, um nó de destino $D_i \in V$ e a quantidade de *flits* do pacote. Então, (G, M) é uma instância do problema de roteamento de pacotes com percursos variáveis. Para cada pacote é necessário encontrar o caminho $P_i = (v_0, v_1, \ldots, v_{l-1}, v_l)$, onde $v_0 = S$ e $v_l = D$. Assume-se que a transmissão de um *flit* através de um canal de comunicação consome uma unidade de tempo (ou um ciclo de *clock*) e que cada canal pode ser usado por no máximo um *flit*. Considera-se ainda que o tempo é discreto - eventos ocorrem apenas em momentos específicos, como transições de *clock* na rede - e que todos os pacotes realizam um passo (envio do cabeçalho de uma chave para outra) simultaneamente. Inicialmente - antes da injeção de pacotes na rede - o peso associado aos canais de comunicação são todos iguais. Cada caminho P_i construído altera o valor do peso dos enlaces por onde o pacote M_i é transmitido. Assim, não só o percurso de um pacote afeta o tempo de transmissão dos demais pacotes: o percurso de todos os pacotes pode influir em todos os demais percursos. O objetivo do problema é minimizar o tempo de chegada do último pacote a seu respectivo nó de destino. Dessa forma, o desempenho da rede pode ser maximizado.

4.3 Roteamento baseado em ACO

Considerando a eficácia da meta-heurística ACO na solução de problemas de otimização com grafos, principalmente no que diz respeito a busca por caminhos, foram desenvolvidos dois

algoritmos para o roteamento em NoCs.

4.3.1 Roteamento EAS

Em algoritmos baseados no *Ant System*, a cada iteração um conjunto de formigas artificiais percorre os vértices do grafo de caracterização do problema. Após várias iterações, ocorrendo a convergência do feromônio, todas as formigas passam a traçar o mesmo percurso. Este percurso é uma solução mínima (local ou global) do problema em questão. Para o roteamento proposto, são consideradas várias instâncias do *Ant System* sobrepostas em um mesmo espaço de busca, sendo uma para cada cada pacote que se deseja transmitir na rede.

A estrutura do REAS - Roteamento baseado no *Elitist Ant System* - é apresentada em pseudocódigo no Algoritmo 7. Os principais parâmetros deste algoritmo são enumerados na Tabela 1. Uma NoC com topologia malha 2D é representada por um ARCG com $size \times size$ vértices. Para uma quantidade de pacotes injetada na rede - definida pela constante $Grupo$ - deseja-se que o algoritmo encontre um percurso para cada pacote, de forma que o tempo de transmissão total seja mínimo.

O algoritmo completo é descrito como uma repetição de um processo iterativo chamado *ciclo*. Por sua vez, o ciclo iterativo é composto pela execução sequencial de um conjunto de m formigas para cada percurso desejado. Dessa forma a quantidade total de formigas que contribuem para o algoritmo é $m \times Grupo \times Max$, conforme visto na Figura 24.

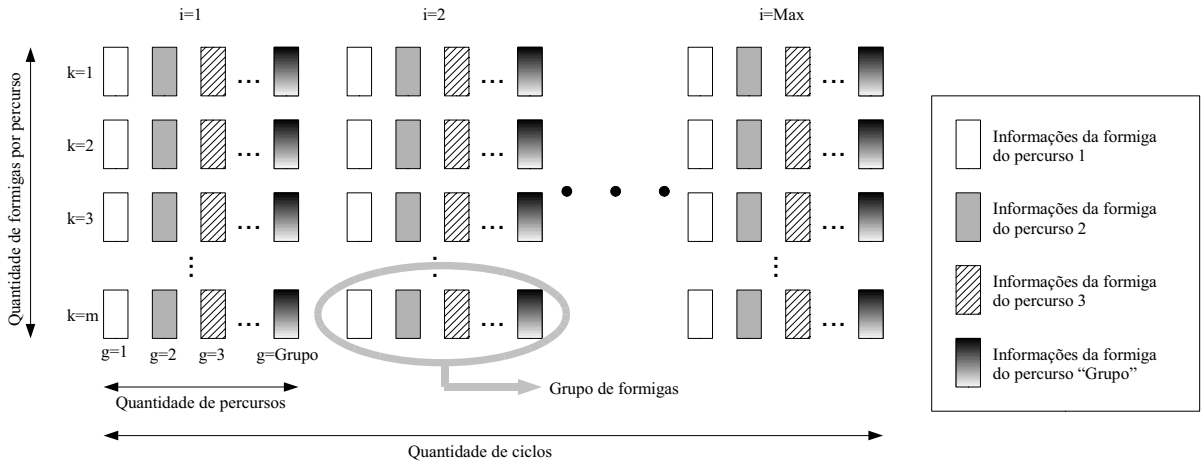


Figura 24: Formigas geradas na execução do Roteamento EAS.

A formiga artificial do algoritmo proposto é constituída por uma sequência de instruções, onde os dados do pacote - nós de origem e destino e o tempo de início da transmissão - são usados na busca por uma rota. O passo de uma formiga é a transição de um nó para outro dentro da rede; a cada passo, um novo nó é adicionado ao percurso e informações como custo,

Algoritmo 7 Roteamento EAS

Define size, tL;
Define α, β, ρ, Q, E, m;
Define Grupo;
Para $g = 1 \rightarrow Grupo$ **Faça**
  Define (xi(g), yi(g));
  Define (xf(g), yf(g));
  Define t_INICIO(g), num_phits(g);
Fim Para

Inicializa τ, carga, F;
Inicializa Fb, i;

Enquanto $i < max$ **Faça**
  Inicializa τ_{atual}, dead;
  Para $k = 1 \rightarrow m$ **Faça**
    Inicializa τ_{local}, percurso;
    Para $g = 1 \rightarrow Grupo$ **Faça**
      $posição\ atual \leftarrow (x_i(g), y_i(g))$;
      $posição\ final \leftarrow (x_f(g), y_f(g))$;
      Inicializa tabu;
      Inicializa F(formiga atual), percurso(formiga atual);
      Enquanto $posição\ atual \neq posição\ final$ **Faça**
        $posição\ atual, F_{out} \leftarrow proximo\_eas\ (\ )$;
        Se $atual = (0, 0)$ **Então**
          $dead \leftarrow dead + 1$;
          Reinicializa formiga atual;
        Senão
          $F \leftarrow F + F_{out}$;
          $carga(formiga\ atual, posicao\ atual) \leftarrow F_{out}$
          $percurso(formiga\ atual, tempo\ atual) \leftarrow posição\ atual$;
          $tabu(posição\ atual) \leftarrow 1$;
        Fim Se
      Fim Enquanto
      $\tau_{local}(formiga\ atual) \leftarrow \frac{Q}{F} \cdot tabu$;
    Fim Para
    Se $F < Fb$ **Então**
      $Fb \leftarrow F$;
      $\tau_{elite} \leftarrow E \cdot \tau_{local}$;
    Fim Se
    $\tau_{atual} \leftarrow \tau_{atual} + \tau_{local} + \tau_{elite}$;
  Fim Para
  $\tau \leftarrow (1 - \rho) \cdot \tau + \tau_{atual}$;
  $i \leftarrow i + 1$;
Fim Enquanto
Retorna Fb, percurso;

Tabela 1: Parâmetros do Roteamento EAS

Parâmetro	Definição
α	Importância relativa do rastro de feromônio
β	Importância relativa da carga da rede
ρ	Constante de evaporação do feromônio
E	Constante de elitismo
Q	Quantidade de feromônio
m	Quantidade de formigas por iteração
$Grupo$	Quantidade de pacotes simultâneos
$(xi,\ yi)$	Coordenadas do nó de origem de um pacote
$(xf,\ yf)$	Coordenadas do nó de destino
t_INICIO	Tempo de início da injeção de um pacote
num_phits	Quantidade de *phits* de um pacote
$size$	Tamanho da rede
t_L	Tempo de enlace
$carga$	Matriz de carga da rede
τ	Matriz de feromônio
F	Custo do percurso encontrado por uma formiga
Fb	Custo do melhor percurso encontrado
$tabu$	Matriz tabu para uma formiga
τ_{local}	Feromônio de uma formiga por caminho
τ_{elite}	Feromônio da formigas elitistas
τ_{atual}	Feromônio de todas as formigas de um ciclo iterativo

feromônio e carga são atualizadas. Um *grupo de formigas* é composto por cada formiga da iteração k responsável por um pacote diferente na rede. Logo, cada ciclo iterativo é composto por m grupos de formigas.

4.3.1.1 Matriz Tabu

O REAS faz uso de uma *matriz tabu*, inspirada na *lista tabu* do *Ant System* (DORIGO; MANIEZZO; COLORNI, 1996). Cada nó da rede é representado por um elemento de uma matriz $size \times size$. Esta matriz é inicializada com todos os seus elementos possuindo valor 0. Quando um nó é visitado por uma formiga, o elemento associado na matriz passa a ter valor 1. Durante o processo de busca pelo percurso, uma formiga fica impedida de percorrer um nó marcado na matriz tabu.

O objetivo principal do uso de uma matriz tabu é permitir que haja uma maior exploração do espaço de busca pela formiga, impedindo que esta visite várias vezes o mesmo nó da rede. Dessa forma, evita-se também que a formiga transite indefinidamente pela rede, sem nunca alcançar o nó de destino.

O uso do tabu pode, contudo, levar a uma situação em que todos os nós adjacentes -

nós que podem ser adicionados ao percurso - já estão sinalizados como visitados. Neste caso, como a formiga não pode seguir em seu percurso, ela é considerada uma *formiga morta*. Por se tratar de uma solução que não alcançou seu objetivo, esta é considerada uma solução ruim. Assim, todos os valores referentes à formiga morta são reinicializados, e uma nova formiga fica encarregada de encontrar uma solução.

O fato de existirem formigas mortas no algoritmo constitui, a primeira vista, uma situação não desejada. Se muitas formigas morrem, então há um elevado esforço computacional que não contribui para a solução final. Uma alternativa a esta situação é a retirada da matriz tabu, permitindo que as formigas circulem livremente por toda rede. Uma versão do REAS sem o uso da matriz tabu foi implementada. Graças ao comportamento do ACO, este outro algoritmo também foi capaz de encontrar a solução para o problema de roteamento; contudo este se deu em um número maior de ciclos. Tem-se então duas situações possíveis:

- Usar a matriz tabu. A restrição de nós da rede pode ocasionar formigas mortas, que devem ser descartadas e reiniciadas. Várias formigas mortas representam um esforço computacional não utilizado na solução final do problema.

- Não usar a matriz tabu. Cada formiga pode circular sem restrições pela rede, podendo inclusive passar várias vezes pelos mesmos nós. Todos os percursos encontrados são válidos. Podem ser encontrados resultados muito ruins por iteração, degradando o desempenho do algoritmo.

Ambas as versões - com e sem tabu - apresentam características interessantes. A escolha por qual destas será empregada no decorrer deste trabalho foi feita com base na análise empírica do desempenho dos dois modelos. Um conjunto de 20 simulações foi realizado para cada versão, variando a quantidade de pacotes simultâneos de 1 até 20 pacotes. Todos os demais parâmetros de simulação foram mantidos os mesmos para os dois casos. Para cada simulação, dois valores retornados foram analisados: a latência e o tempo de execução. A Figura 25 mostra os resultados obtidos. Observa-se que tanto para o tempo de execução da simulação quanto para a latência média, a versão com tabu mostrou um desempenho melhor que a versão sem tabu. Por este motivo, optou-se pela implementação que faz uso da matriz tabu.

4.3.1.2 Matriz de Feromônio

O feromônio empregado no REAS, assim como o tabu, é uma matriz onde cada nó de rede é representado por um elemento. Contudo, enquanto o tabu existe apenas durante a construção

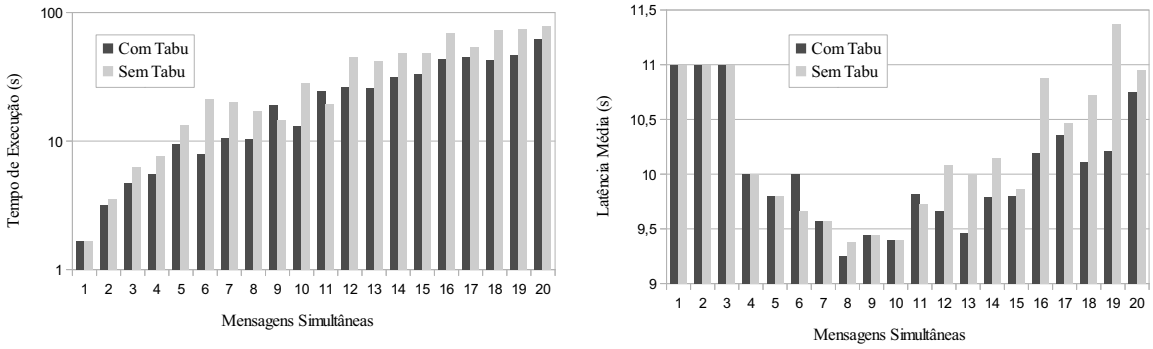


Figura 25: Impacto do uso da matriz tabu.

de uma solução por uma formiga, o feromônio persiste por toda a execução do algoritmo. Outra diferença está no fato de cada conjunto de m formigas responsável pela busca do percurso de um pacote possuir seu próprio feromônio. A matriz feromônio τ é, pois, uma matriz $size \times size \times Grupo$, onde o valor da terceira dimensão indica a qual pacote está associado o feromônio da matriz formada pelas duas primeiras dimensões. Em outras palavras, o algoritmo faz uso de múltiplas colônias (ENGELBRECHT, 2006), onde cada colônia é responsável pela busca do percurso para um pacote. O elementos da matriz de feromônio são todos inicializados com o valor da constante Q.

Uma formiga k usa, ao final da construção do percurso, a matriz tabu na criação de seu depósito de feromônio τ^k_{local}, seguindo a Equação 26. A formiga k associada a cada pacote irá usar a Equação 26 para construir um τ^k_{local} de tamanho $size \times size \times Grupo$, tal como a matriz feromônio τ.

$$\tau^k_{local} = \frac{Q}{L_k} \cdot tabu \tag{26}$$

A contribuição de cada grupo k de formigas é acumulada de forma sequencial pela matriz τ_{atual}. Se o grupo de formigas encontra uma solução que é melhor que todas as anteriores, sua contribuição de feromônio é feita de forma diferenciada. Além da contribuição normal τ^k_{local}, é acrescentada uma parcela de feromônio de *formigas elitistas*, descrito pela Equação 27.

$$\tau_{elite} = E \cdot \tau^k_{local} \tag{27}$$

Dessa forma, ao fim de um ciclo iterativo, a matriz de feromônio τ é atualizada conforme a Equação 28. Parte do feromônio evapora com taxa $(1 - \rho)$, enquanto que outra é reforçada pela contribuição do ciclo atual τ_{atual}

$$\tau = (1 - \rho) \cdot \tau + \tau_{atual} \tag{28}$$

4.3.1.3 Carga da rede

A carga da rede é representada por uma matriz $size \times size \times 4 \times Grupo$. Assim como o feromônio, é composta de duas dimensões representando as dimensões da rede e uma representando a quantidade de pacotes transmitidos simultaneamente na rede. Há ainda mais uma dimensão, de tamanho 4, que indica as quatro possíveis portas de saída de uma chave. Assim, a carga em uma chave específica representa, na realidade, o custo da utilização de cada um de seus canais de saída.

Seja a rede ao início da execução do REAS. A matriz carga possui todos seus elementos inicializados com o valor 0. Estes valores são atualizados a cada percurso encontrado por uma formiga, sendo então reiniciado a cada novo ciclo iterativo do algoritmo. Uma formiga, ao construir um percurso, armazena na matriz carga o valor atual da função custo F, que segue o modelo apresentado na Seção 4.1.2. A carga resultante de uma formiga será vista pelas demais formigas do grupo. Como o algoritmo é executado de forma sequencial (para o grupo de formigas de uma iteração k são geradas formigas de $g = 1$ até $g = Grupo$) a atuação da carga ocorre de forma cíclica. A primeira formiga gerada afeta as demais formigas do grupo ($g = 2$ até $g = Grupo$) da mesma iteração k. A partir da segunda formiga, a carga afeta não só as demais formigas da iteração atual k, mas também as primeiras formigas da iteração seguinte $k + 1$.

4.3.1.4 Função de seleção

Durante a construção do percurso, uma formiga seleciona uma sequência de nós da rede, desde a chave de origem até a de destino. A escolha por qual nó será o próximo do percurso é feita pela função *proximo_eas*(), cuja estrutura é apresentada pelo Algoritmo 8. As etapas principais da função de seleção são: definir a carga vista por um pacote; calcular a probabilidade de escolha das quatro direções; selecionar uma direção; retornar o próximo nó e o custo associado.

Dado o nó atual na construção do percurso, a função *proximo_eas*() define inicialmente quais são os quatro nós adjacentes - Norte, Sul, Leste e Oeste. A análise da matriz carga indica qual será o atraso sofrido pelo nó atual em cada uma das possíveis opções de escolha. Três condições de atraso podem ser caracterizadas, listadas a seguir:

- Condição 1 - Atraso na origem. Um mesmo nó é a origem de mais de um pacote, onde cada pacote possui uma origem diferente. Se estes pacotes selecionam o mesmo canal de saída da chave, então o segundo pacote só iniciará sua transmissão após todos os *flits* do primeiro pacote serem enviados.

Algoritmo 8 Função de seleção *proximo_eas*()

Entrada Parâmetros da rede e do EAS

{Cálculo da carga vista no nó atual}
Define os quatro nós vizinhos;
Para $p = 1 \rightarrow 4$ **Faça** {Os quatro nós vizinhos}
  Para $g = 1 \rightarrow Grupo$ **Faça**
    Se $g = percurso\ atual$ **Então**
      $carga\_vista(1, g) \leftarrow F + tL$;
      $carga\_vista(2, g) \leftarrow F + tL$;
      $carga\_vista(3, g) \leftarrow F + tL$;
      $carga\_vista(4, g) \leftarrow F + tL$;
    Senão
      Se *Condição 1* ***Então***
        $carga\_vista(p, g) \leftarrow num\_phits(g)$;
      Senão Se *Condição 2* ***Então***
        $carga\_vista(p, g) \leftarrow num\_phits(g) - (flits\ do\ pacote\ g\ enviados)$;
      Senão Se *Condição 3* ***Então***
        $carga\_vista(p, g) \leftarrow 0$;
      Fim Se
    Fim Se
  Fim Para
Fim Para

$carga\_efetiva \leftarrow (soma\ de\ todas\ as\ colunas\ de\ carga\_vista)$;

{Cálculo das probabilidades dos nós vizinhos}
Para $i = 1 \rightarrow 4$ **Faça**
  Se tabu(atual)=1 **Então**
    $atratividade(i) \leftarrow 0$;
  Senão
    $N(i) \leftarrow \frac{1}{carga\_efetiva}(i)$;
    $atratividade(i) \leftarrow \tau(i)^{\alpha} \cdot N(i)^{\beta}$;
  Fim Se
Fim Para
Para $i = 1 \rightarrow 4$ **Faça**
  $Prob(i) \leftarrow \frac{atratividade(i)}{atratividade(1)+atratividade(2)+atratividade(3)+atratividade(4)}$;
Fim Para

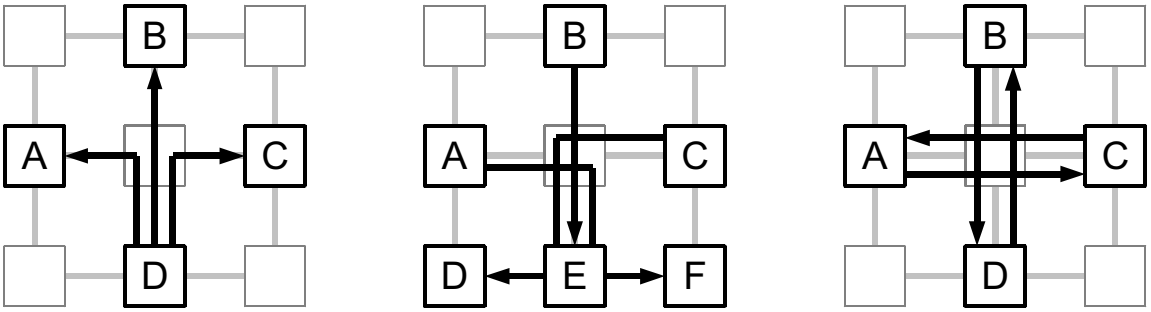
{Escolha do próximo nó}
$proximo \leftarrow roleta(Prob)$;
Se $Prob(proximo) = 0$ **Então**
  $proximo \leftarrow (0, 0)$;
Fim Se
$F\_out \leftarrow carga\_efetiva(proximo)$;

Retorna proximo, F_out;

- Condição 2 - Atraso em um nó intermediário. Um pacote A em uma chave seleciona como saída um canal que já se encontra em uso por outro pacote B. O canal só estará livre para uso por A após B encerrar sua transmissão. O atraso sofrido por A ocasionado por B é definido pela quantidade total de *flits* de B menos a quantidade de *flits* já transmitidos pelo canal no instante que A chega a chave.

- Condição 3 - Sem atraso. O canal de saída selecionado não foi usado por nenhum outro pacote, ou foi usado por um pacote que já encerrou sua transmissão. Nesta situação, o atraso é nulo.

Estas três situações podem ser descritas pela Figura 26. Em (a), o nó D transmite três pacotes distintos para os nós A, B e C. O nó B só receberá seu pacote após o envio para A; já o nó deverá aguardar a transmissão para os nós A e B. Em (b), três nós A, B e C transmitem pacotes distintos para os nós D, E e F. Contudo, todos os três necessitam atravessar um mesmo canal de comunicação. Por último, (c) mostra a situação em que quatro pacotes atravessam uma mesma chave, mas nenhum sofre atraso, por estarem todos usando canais de comunicação diferentes.



(a) Conflito na chave de origem.

(b) Conflito em uma chave intermediária.

(c) Transmissão sem conflito.

Figura 26: Condições de conflito possíveis em uma chave.

Dada as condições de atraso que o pacote observa no nó atual, a função *proximo_eas*() cria internamente a matriz *carga_vista*, de tamanho $4 \times Grupo$. Cada linha representa o atraso em uma direção, enquanto que cada coluna indica a qual pacote este atraso está referenciado. O atraso total será aquele ocasionado por todos os pacotes; desta forma, a matriz $carga\_efetiva$ será a soma de todas as colunas de *carga_vista*. Este será um vetor de quatro elementos, cada um indicando o atraso associado a cada direção. Como o valor do atraso será utilizado na

composição de F, o valor atual do custo é somado a cada elemento de $carga\_efetiva$, bem como o custo de um passo, tL.

De posse de $carga\_efetiva$, é necessário em seguida calcular a probabilidade de escolha de cada direção. Como mostrado na Seção 3.3.1.4, a probabilidade de seleção de uma dada direção (dentro de um conjunto de direções possíveis) é função da visibilidade e do feromônio em cada direção, e dos parâmetros α e β. Para o TSP, a visibilidade foi definida como o inverso da distância entre duas cidades. No problema de roteamento, a visibilidade η_i será o inverso da carga na direção i (Equação 29). Assim, quanto maior é a carga em uma direção, menos vantajosa esta será para a formiga atual.

$$\eta_i = \frac{1}{carga\_efetiva_i} \tag{29}$$

Embora a visibilidade tenha influência na seleção do nó seguinte, a *atratividade* de uma direção é função também do feromônio do nó nesta mesma direção. A variável *atratividade* representa o quanto cada caminho é interessante de ser seguido. Seu valor é definido pela Equação 30. O parâmetro α define o quanto o feromônio em uma dada direção i é importante. Já β representa a importância da visibilidade nesta mesma direção.

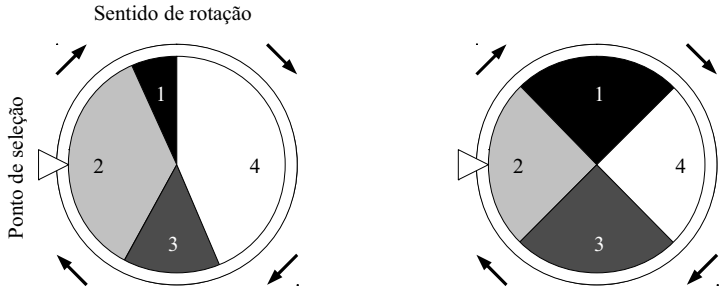
$$Atratividade_i = \tau_i^{\alpha} \cdot \eta_i^{\beta} \tag{30}$$

Com o valor da *atratividade* de cada direção, é possível calcular a probabilidade de seleção de uma direção específica, conforme a Equação 31.

$$Prob_i = \frac{Atratividade_i}{\sum_{j=1}^{4} Atratividade_j} \tag{31}$$

A seleção de uma entre as quatro direções possíveis é feita usando o método da *roleta viciada*. Este é um método amplamente usado em algoritmos genéticos na seleção de indivíduos com base em sua aptidão (GOLDBERG, 1989). Seu funcionamento é inspirado no funcionamento da roleta de jogos de azar. Neste método, cada direção é representada em uma roleta conforme sua probabilidade. A roleta é dividida em quatro regiões, com a área de cada região sendo proporcional à probabilidade de escolha de uma direção, como visto na Figura 27. Ao girar a roleta, a região de maior área terá mais chances de ser escolhida. Contudo, as demais regiões ainda podem eventualmente ser escolhidas.

A seleção por roleta é baseada na geração de um número aleatório. A probabilidade das quatro direções é somada, obtendo-se o valor S. Um valor aleatório R é gerado no intervalo $[0, S]$, seguindo uma distribuição uniforme. Os valores das probabilidades são somados individualmente até que se obtenha um valor maior que R. A direção corrente é então selecionada.



(a) Quatro direções com probabilidades de seleção diferentes.

(b) Com probabilidades iguais.

Figura 27: Seleção por roleta.

4.3.1.5 Exemplo de execução do REAS

Esta seção apresenta um exemplo de simulação do algoritmo proposto. O Roteamento EAS foi modelado e executado no ambiente Matlab. Em uma malha 5×5, deseja-se transmitir 6 pacotes de 4 *flits* cada. Cada pacote possui nós de origem e destino diferentes, gerados aleatoriamente para a simulação. Cada *flit* gasta 1 ciclo de *clock* da rede para se deslocar de um nó para um vizinho; todos os pacotes começam a ser transmitidos após os 2 primeiros ciclos de *clock* da rede.

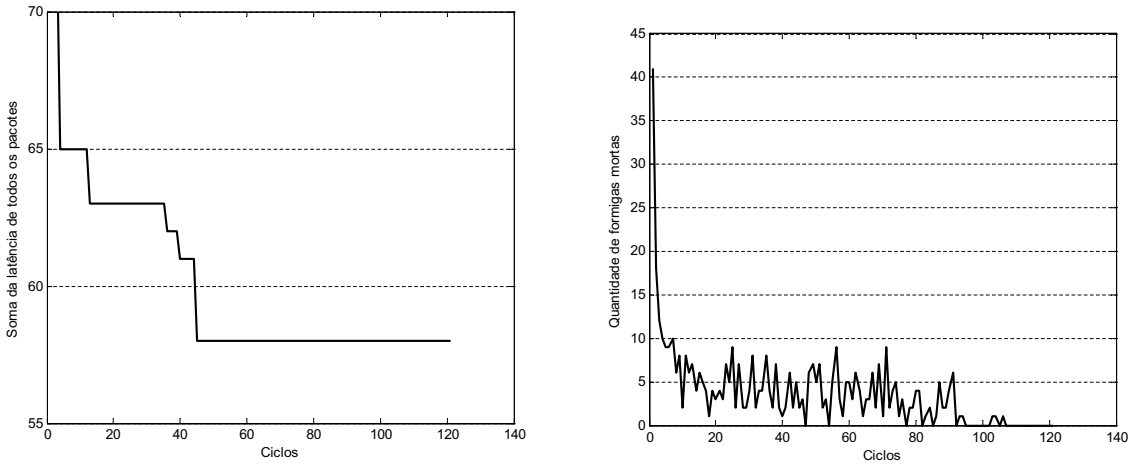
Os principais parâmetros do EAS são apresentados na Tabela 2. A escolha do valor destes parâmetros foi baseada em exemplos encontrados na literatura (DORIGO; GAMBARDELLA, 1997)(DORIGO; STÜTZLE, 2004). Os valores de m e E (quantidade de formigas e constante de elitismo) foram selecionadas de forma empírica[1].

Tabela 2: Parâmetros para simulação do REAS

Parâmetro	α	β	ρ	Q	m	E
Valor	1	2	0,5	1	6	6

O resultado retornado pelo REAS é o melhor percurso encontrado pelo algoritmo. O melhor resultado é aquele que possui o menor valor de F para cada percurso de um grupo de formigas. Como o valor de F indica a latência individual de um percurso, a soma do F de cada formiga de um grupo é a latência total. Em outras palavras, trata-se do valor que se deseja minimizar. A Figura 28(a) mostra como ocorre, para este exemplo, a variação do melhor valor de F no decorrer dos ciclos iterativos do EAS.

[1]Em testes com valores de m e E variando de 5 até 10, observou-se que o valor 6 retorna os melhores resultados



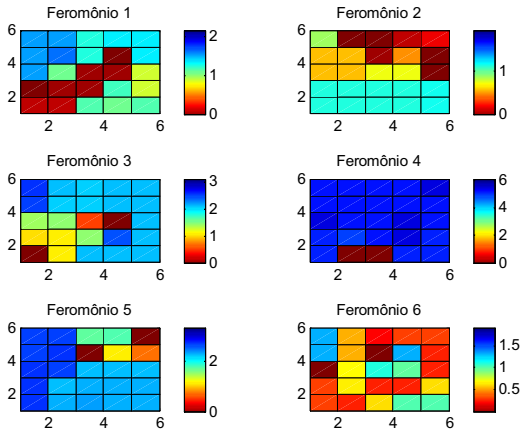
(a) Melhor valor de F.

(b) Quantidade de formigas mortas.

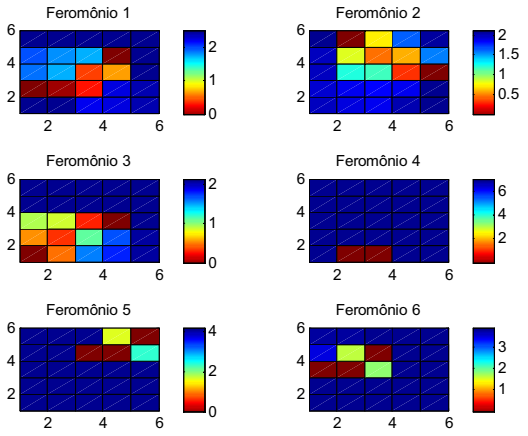
Figura 28: Otimização no decorrer dos ciclos iterativos do REAS.

Um valor importante observado na execução do algoritmo é a quantidade de formigas mortas por ciclo iterativo. Para a simulação realizada, este valor daria em torno de cinco formigas por ciclo, como pode ser visto na Figura 28(b). Contudo, a partir do ciclo 100, este valor cai bruscamente, até que mantém-se em zero até o fim da execução da simulação. Esta diminuição está relacionada com a convergência do feromônio. A Figura 29 mostra a concentração do feromônio associado a cada percurso em diferentes ciclos. A rede é representada por um gráfico com 25 subdivisões, com cada subdivisão representando um nó. A cor de cada subdivisão indica a concentração de feromônio. Nos primeiros ciclos, o feromônio apresenta-se bastante distribuído pelos nós da rede. A cada ciclo, o feromônio se concentra cada vez mais em uma região em torno do melhor percurso. É possível observar que vários ciclos antes do termino do algoritmo, alguns percursos apresentam todo o feromônio concentrado em uma única região, formada pelos nós que constituem o caminho ótimo. Nesta situação, é dito que houve a convergência do feromônio. A concentração de feromônio eleva-se de tal forma que as formigas dos ciclos seguintes passam a seguir todas o mesmo percurso. Como consequência, não há mais a ocorrência formigas mortas para este caminho.

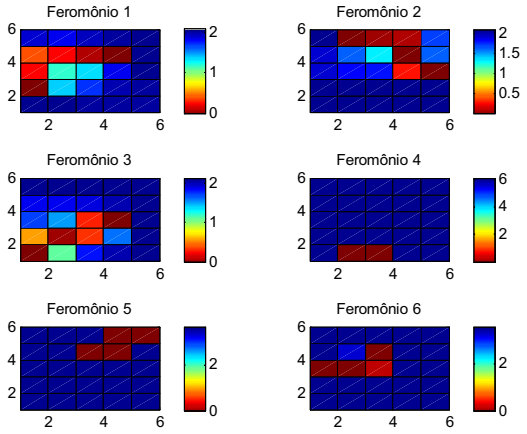
A quantidade de formigas mortas se iguala a zero por uma grande sequência de ciclos apenas quando o algoritmo converge. Dessa forma, a situação de convergência pode ser identificada se o valor do vetor *dead* for zero por uma quantidade considerável de ciclos. Assim, este foi usado como critério de parada nesta simulação: a convergência do feromônio impede a busca por soluções diferentes, logo o algoritmo não precisa continuar sendo executado. Foi



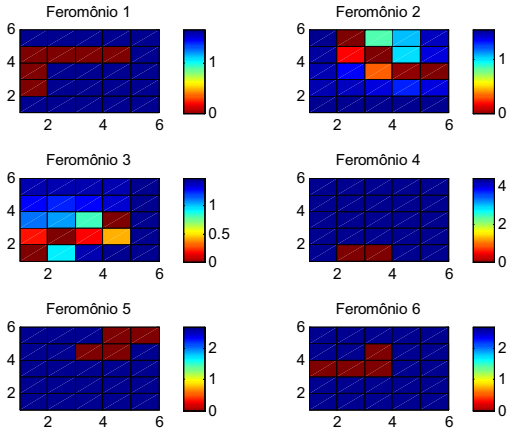
(a) Ciclo 1



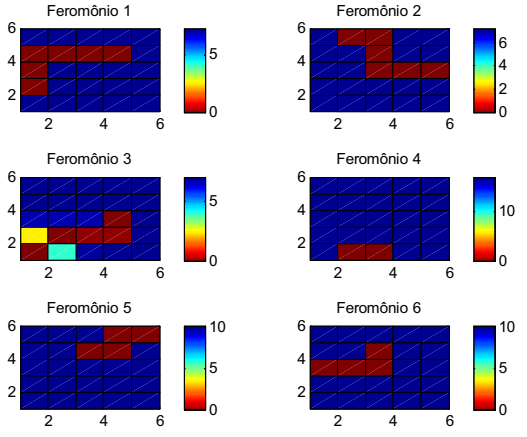
(b) Ciclo 15



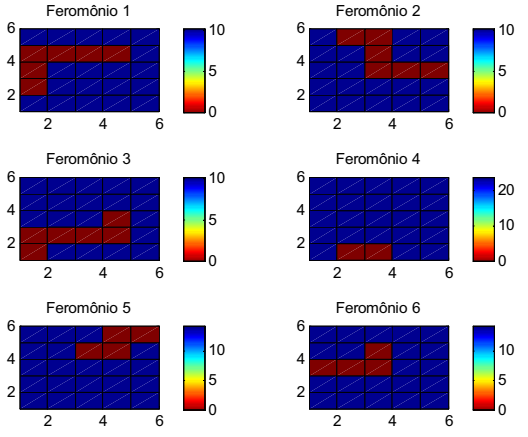
(c) Ciclo 45



(d) Ciclo 75



(e) Ciclo 105



(f) Ciclo 135

Figura 29: Concentração do feromônio de 6 percursos em diferentes ciclos do REAS.

considerado o número de 30 ciclos em que o valor de *dead* se manteve menor que um limiar de 5 formigas mortas para que o algoritmo fosse encerrado. Caso contrário, este seria finalizado ao completar 500 ciclos iterativos.

A latência total retornada pelo REAS - a soma da latência individual de cada pacote na rede - foi de 58 ciclos de *clock*. Como foram transmitidos 6 pacotes, a latência média obtida foi de 9,6667 ciclos de *clock*.

4.3.2 Roteamento ACS

O segundo algoritmo de roteamento modelado com base na meta-heurística ACO é apresentado nesta seção. O RACS é um Roteamento baseado em *Ant Colony System* (DORIGO; GAMBARDELLA, 1997) . Como visto na Seção 3.3.2.4, o ACS é caracterizado pelo método de seleção do estado seguinte, que obedece uma regra pseudo-aleatória proporcional, e pela atualização do feromônio, composta por uma atualização local e por uma global.

Embora o *Ant Colony System* apresente diferenças substanciais em relação ao *Ant System*, o algoritmo RACS proposto possui uma estrutura extremamente semelhante a do REAS. De fato, o pseudo-código apresentado no Algoritmo 9 é baseado no REAS, apenas modificado para atender as características do ACS. Os parâmetros α e Q do EAS não são usados. Por sua vez, o ACS adiciona ao roteamento outros dois parâmetros: q_0, usado pela regra de seleção, e τ_0, o valor inicial da matriz de feromônio.

4.3.2.1 Atualização local e global

A atualização do feromônio ocorre em dois instantes distintos. Na construção do percurso, uma formiga altera o feromônio de cada nó por onde passa seguindo a Equação 32. Assim, o caminho descrito por uma formiga tem seu feromônio reduzido pela taxa de evaporação $(1-\rho)$, e reforçado por $(\rho \cdot \tau_0)$, onde τ_0 é o valor inicial do feromônio. Este processo é chamado de atualização local do feromônio.

$$\tau_{local} \leftarrow (1-\rho) \cdot \tau_{local} + \rho \cdot \tau_0 \tag{32}$$

Ao término de um ciclo iterativo, o feromônio sofre mais uma atualização, chamada atualização global. O melhor percurso encontrado até então - aquele que obteve o menor valor de F - gera um feromônio diferenciado, τ_{best}. A atualização global do RACS ocorre apenas sobre os nós pertencentes ao melhor percurso, seguindo a Equação 33.

$$\tau \leftarrow (1-\rho) \cdot \tau + \rho \cdot \tau_{best} \tag{33}$$

Algoritmo 9 Roteamento ACS

Define size, tL;
Define β, ρ, q_0, τ_0, m;
Define Grupo;
Para $g = 1 \rightarrow Grupo$ **Faça**
  Define (xi(g), yi(g));
  Define (xf(g), yf(g));
  Define t_INICIO(g), num_phits(g);
Fim Para
Inicializa τ, carga, F;
Inicializa Fb, i;
Enquanto $i < max$ **Faça**
  Inicializa dead;
  Para $k = 1 \rightarrow m$ **Faça**
    Inicializa τ_{local}, percurso;
    Para $g = 1 \rightarrow Grupo$ **Faça**
      $posição\ atual \leftarrow (x_i(g), y_i(g))$;
      $posição\ final \leftarrow (x_f(g), y_f(g))$;
      Inicializa tabu;
      Inicializa F(formiga atual), percurso(formiga atual);
      Enquanto $posição\ atual \neq posição\ final$ **Faça**
        $posição\ atual, F_{out} \leftarrow proximo\_acs\ (\ )$;
        Se $posição\ atual = (0, 0)$ **Então**
          $dead \leftarrow dead + 1$;
          Reinicializa formiga atual;
        Senão
          $F \leftarrow F + F_{out}$;
          $carga(formiga\ atual, posicao\ atual) \leftarrow F_{out}$;
          $percurso(formiga\ atual, tempo\ atual) \leftarrow posição\ atual$;
          {Atualização local}
          $\tau_{local} \leftarrow (1 - \rho) \cdot \tau_{local} + \rho \cdot \tau_0$;
          $tabu(posição\ atual) \leftarrow 1$;
        Fim Se
      Fim Enquanto
      $\tau_{temp}(formiga\ atual) \leftarrow \frac{1}{F} \cdot tabu$;
    Fim Para
    Se $F < Fb$ **Então**
      $Fb \leftarrow F$;
      $\tau_{best} \leftarrow \tau_{temp}$;
    Fim Se
    $\tau \leftarrow \tau_{local}$;
  Fim Para
  {Atualização global}
  $\tau \leftarrow (1 - \rho) \cdot \tau + \rho \cdot \tau_{best}$;
  $i \leftarrow i + 1$;
Fim Enquanto
Retorna Fb, percurso;

4.3.2.2 Função de seleção

A função de seleção *proximo_acs*() vista no Algoritmo 10 realiza a escolha do nó seguinte de um percurso seguindo a regra pseudo-aleatória proporcional. Um valor q é escolhido aleatoriamente no intervalo $[0, 1]$. Se q for maior que o parâmetro q_0, então a seleção será feita de forma semelhante a realizada pelo EAS. Calcula-se a probabilidade de seleção em cada direção; em seguida, uma das direções é escolhida através de uma seleção por roleta. Caso contrário, a direção escolhida será a que possuir o maior valor de *atratividade*. Assim, o parâmetro q_0 indica o quanto a seleção será baseada em cada um destes dois métodos.

4.3.2.3 Exemplo de execução do RACS

Assim como o caso do REAS, é apresentado um exemplo de simulação do Roteamento ACS. A configuração da rede e dos pacotes é a mesma empregada no exemplo anterior. Os valores dos parâmetros do RACS dão mostrados na Tabela 3. Os valores de β, ρ e q_0 foram escolhidos com base em exemplos da literatura (DORIGO; GAMBARDELLA, 1997) (DORIGO; STÜTZLE, 2004). Já o melhor valor para a quantidade de formigas m e o feromônio inicial τ_0 foram obtidos de forma experimental.

Tabela 3: Parâmetros para simulação do RACS

Parâmetro	β	ρ	q_0	m	τ_0
Valor	2	0,1	0,9	7	0,04

As Figuras 30(a) e 30(b) mostram, respectivamente, a variação do valor de Fb e a quantidade de formigas mortas no decorrer da execução do RACS. Nota-se que o RACS é bem mais agressivo que o REAS: em poucos ciclos iterativos o valor mínimo de Fb é encontrado. Analisando a concentração de feromônio na Figura 31, observa-se que em poucos ciclos cada percurso forma um rastro bem destacado. Este comportamento difere do feromônio no REAS, em que uma região com alta concentração é inicialmente formada em torno de um caminho ótimo, sendo refinada a cada ciclo pelo processo de evaporação. Contudo, devido ao feromônio no RACS estar limitado pelo valor mínimo τ_0, existirá sempre regiões com uma certa quantidade de feromônio fora do rastro principal. No REAS, o feromônio em regiões fora do rastro principal evapora até tender a zero. Assim, embora forme-se um rastro de feromônio para cada percurso, no RACS não é possível afirmar que há uma convergência do feromônio. Desta forma, ocorrem formigas mortas em diferentes taxas em toda a execução do algoritmo. Ainda assim, o critério de parada é semelhante ao adotado para o REAS: encerrar a execução se o número de formigas

Algoritmo 10 Função de seleção *proximo_acs*()

Entrada Parâmetros da rede e do ACS

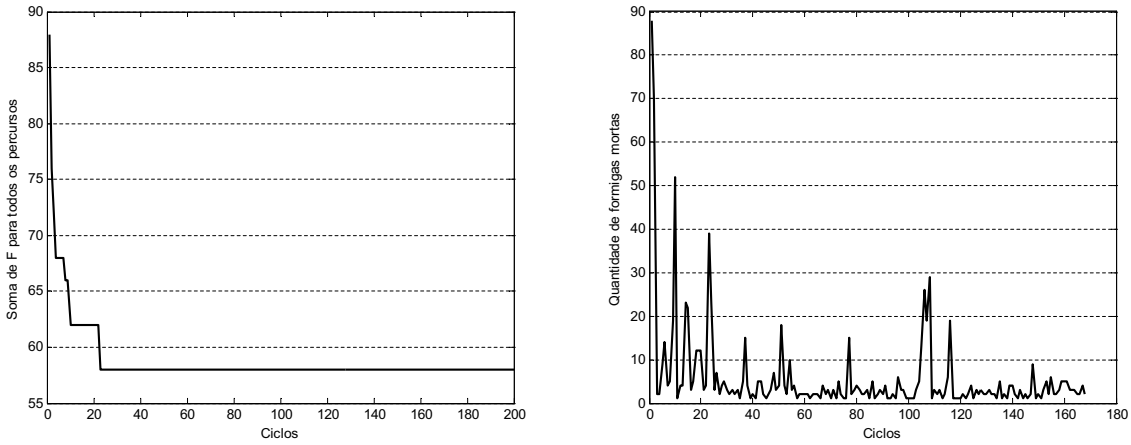
Define os quatro nós vizinhos;
Para $p = 1 \rightarrow 4$ **Faça** {Os quatro nós vizinhos}
  Para $g = 1 \rightarrow Grupo$ **Faça**
    Se $g = percurso\ atual$ **Então**
      $carga\_vista(1 \rightarrow 4, g) \leftarrow F + tL$;
    Senão
      Se *Condição 1* ***Então***
        $carga\_vista(p, g) \leftarrow num\_phits(g)$;
      Senão Se *Condição 2* ***Então***
        $carga\_vista(p, g) \leftarrow num\_phits(g) - (flits\ do\ pacote\ g\ enviados)$;
      Senão Se *Condição 3* ***Então***
        $carga\_vista(p, g) \leftarrow 0$;
      Fim Se
    Fim Se
  Fim Para
Fim Para
$carga\_efetiva \leftarrow (soma\ de\ todas\ as\ colunas\ de\ carga\_vista)$;

Para $i = 1 \rightarrow 4$ **Faça**
  Se tabu(atual)=1 **Então**
    $atratividade(i) \leftarrow 0$;
  Senão
    $N(i) \leftarrow \frac{1}{carga\_efetiva}(i)$;
    $atratividade(i) \leftarrow \tau(i) \times N(i)^{\beta}$;
  Fim Se
Fim Para

$q \leftarrow$ aleatório entre 0 e 1;
Se $q > q_0$ **Então**
  Para $i = 1 \rightarrow 4$ **Faça**
    $Prob(i) \leftarrow \frac{atratividade(i)}{atratividade(1)+atratividade(2)+atratividade(3)+atratividade(4)}$;
  Fim Para
  $proximo \leftarrow roleta(Prob)$;
  Se $Prob(proximo) = 0$ **Então**
    $proximo \leftarrow (0, 0)$
  Fim Se
Senão
  $proximo \leftarrow argmax(atratividade)$;
  Se $atratividade(proximo) = 0$ **Então**
    $proximo \leftarrow (0, 0)$;
  Fim Se
Fim Se

$F\_out \leftarrow carga\_efeteiva(proximo)$;
Retorna proximo, F_out;

mortas for inferior a 5 formigas por ciclo por mais de 30 ciclos consecutivos, ou se o limite de 200 ciclos for alcançado.



(a) Melhor valor de F.

(b) Quantidade de formigas mortas.

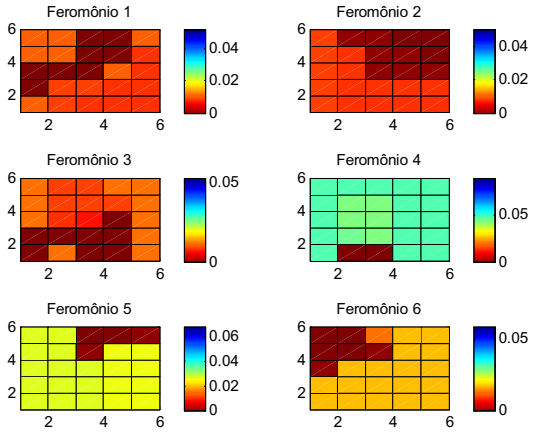
Figura 30: Otimização no decorrer dos ciclos iterativos do RACS.

O RACS foi capaz de encontrar o mesmo resultado retornado pelo REAS de 58 ciclos de *clock* da rede de latência total.

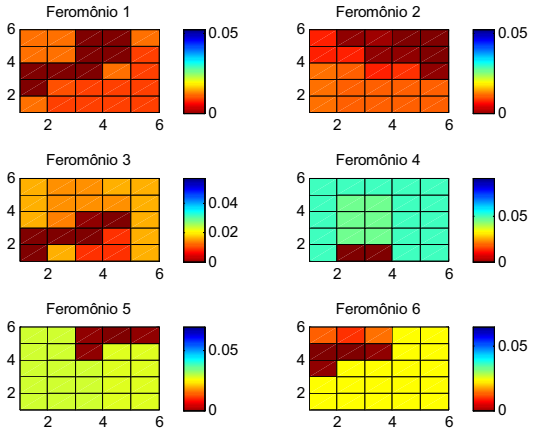
4.4 Considerações Finais do Capítulo

A Otimização por Colônia de Formigas, com sua capacidade de busca por percursos, desponta como uma opção eficiente para a solução de problemas de roteamento. Desta forma, o presente trabalho apresenta o uso da meta-heurística ACO na construção de algoritmos de roteamento. Mais precisamente, dois modelos de roteamento estático para redes embutidas são propostos.

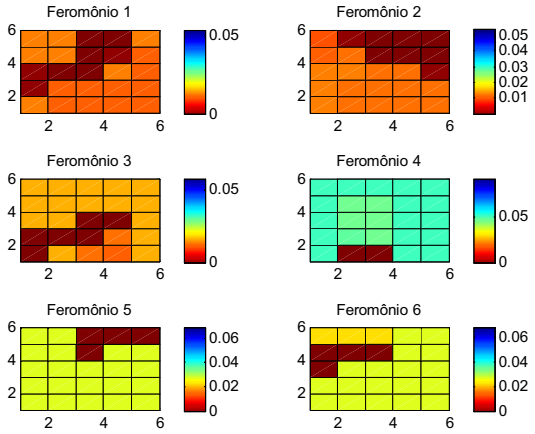
Neste capítulo, foram apresentadas as especificações das redes onde os algoritmos de roteamento serão empregados. O problema de roteamento foi formulado, onde o objetivo principal é a transmissão de um conjunto de pacotes com a menor latência possível. Os dois modelos propostos são detalhados. O primeiro, REAS, é baseado no *Elitist Ant System* (DORIGO; MANIEZZO; COLORNI, 1996). Suas principais características são apresentadas, incluindo o uso de formigas elitistas. O segundo, chamado de RACS, é inspirado no *Ant Colony System* (DORIGO; GAMBARDELLA, 1997), possuindo como principal característica suas duas atualizações de feromônio e uma regra de seleção pseudo aleatória proporcional. O capítulo seguinte apresenta simulações com os dois algoritmos propostos, encontrando soluções de roteamento para redes operando sobre padrões sintéticos de geração de pacotes.



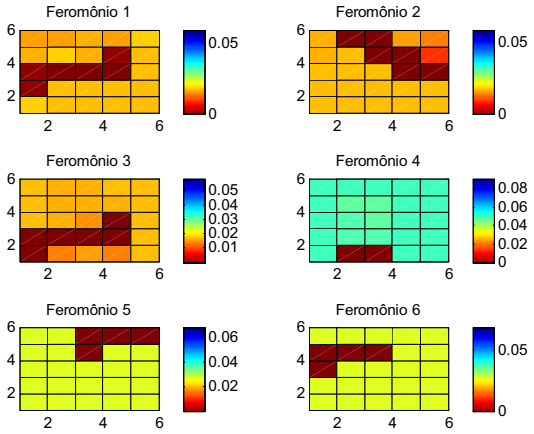
(a) Ciclo 5



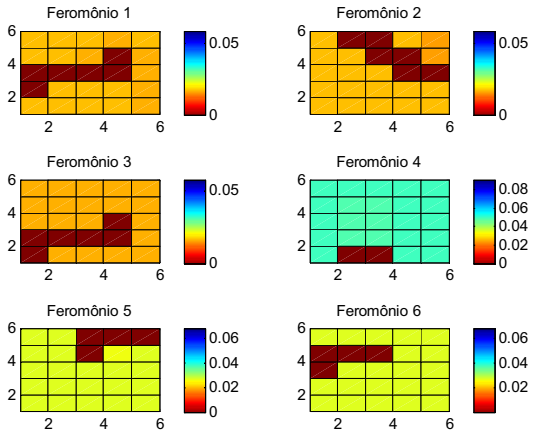
(b) Ciclo 10



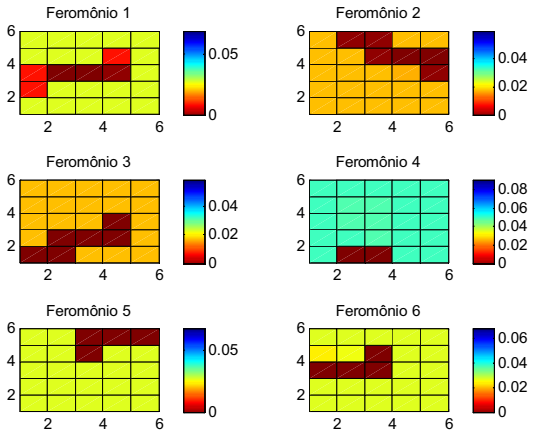
(c) Ciclo 20



(d) Ciclo 50



(e) Ciclo 100



(f) Ciclo 200

Figura 31: Concentração de feromônio durante a execução do RACS.

Capítulo 5

AVALIAÇÃO DE DESEMPENHO

NESTE capítulo são analisados os resultados de uma série de simulações, com o objetivo de estimar o desempenho dos algoritmos propostos em relação a outros métodos de roteamento. Para cada simulação, é observado o valor de latência total e média da rede sob diferentes condições de carga.

Segundo Duato, a carga da rede exerce uma extrema influência sobre seu desempenho. Em uma rede com chaveamento *wormhole*, para uma dada distribuição espacial de pacotes, a latência média é mais afetada pela carga da rede do que por qualquer outro parâmetro (DUATO; YALAMANCHILI; LIONEL, 2002). Assim, uma correta modelagem da carga da rede é essencial para a avaliação de seu desempenho (TEDESCO *et al.*, 2005).

Diferentes padrões de tráfego são capazes de reproduzir o comportamento de diferentes tipos de aplicações. Simulações com padrões sintéticos auxiliam a análise do comportamento da rede de forma mais abrangente, forçando a rede a operar sob padrões de comunicação possíveis, mas que eventualmente não serão encontrados nas aplicações de teste.

Na Seção 5.1 são abordados padrões sintéticos comumente empregados na modelagem do comportamento da carga de aplicações reais. Na Seção 5.2 são mostrados dois métodos de roteamento usados para comparação com os modelos propostos. A Seção 5.3 apresenta os resultados de simulação empregando tais padrões.

5.1 Padrões de geração de tráfego

A definição de modelos representativos para carga é necessária para a avaliação de uma rede de interconexão. Esta é uma tarefa difícil, pois o comportamento da rede pode variar consideravelmente com a arquitetura e com a aplicação. Ainda assim, o desempenho costuma ser mais sensível a carga que aos demais parâmetros do projeto da rede. Em geral, não há um consenso sobre um conjunto padrão de modelos de carga que poderiam ser empregados para

a análise da rede. Assim, a avaliação da rede é feita com o uso de padrões de tráfego com diferentes características. Estes modelos podem ser usados na falta de mais informações sobre o comportamento das aplicações.

Um modelo de tráfego define a estrutura da transmissão de dados desde os nós de origem até os de destino. Em geral, a modelagem da carga é definida por três parâmetros: a *distribuição espacial dos pacotes*, a *taxa de injeção dos pacotes* e o *tamanho dos pacotes*.

5.1.1 Distribuição de pacotes

A distribuição espacial dos pacotes indica a relação entre os endereços dos nós de origem e dos nós de destino. O modelo mais frequentemente usada é a *distribuição uniforme*, onde a probabilidade de um nó i enviar um pacote para um nó j é a mesma para todo i e j. Esta distribuição não faz suposição alguma sobre o tipo de computação que gera os pacotes.

Devido às similaridades entre MPSoCs e sistemas distribuídos, padrões encontrados em aplicações paralelas podem ser usados para definir distribuições espaciais de pacotes em NoCs. Padrões atualmente usados incluem *bit reversal*, *perfect shuffle*, *butterfly*, *matrix transpose* e *complement*. É usado, para estes padrões, um endereçamento dos nós por um sistema de coordenadas binário, conforme apresentado pela Figura 32. Os nós de destino de todos os pacotes gerados por um nó será sempre o mesmo para um determinado padrão, sendo assim escolhidos em função do endereço do nó de origem. Pelo fato dos pares de nós serem gerados seguindo uma regra, a utilização dos enlaces da rede não é uniforme. A descrição destes padrões é descrita a seguir:

- *Bit Reversal*. O nó com coordenadas binárias $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ envia pacotes para o nó $a_0, a_1, \ldots, a_{n-2}, a_{n-1}$. A ordem dos *bits* é invertida do menos significativo para o mais significativo.

- *Perfect Shuffle*. O nó com coordenadas binárias $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ envia pacotes para o nó $a_{n-2}, a_{n-3}, \ldots, a_0, a_{n-1}$. É feita a rotação de 1 *bit*.

- *Butterfly*. O nó com coordenadas binárias $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ envia pacotes para o nó $a_0, a_{n-2}, \ldots, a_1, a_{n-1}$. O *bit* mais significativo troca de posição com o menos significativo.

- *Matrix Transpose*. O nó com coordenadas binárias $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ envia pacotes para o nó $a_{\frac{n}{2}-1}, a_{\frac{n}{2}-2}, \ldots, a_0, a_{n-1}, \ldots, a_{\frac{n}{2}}$.

- *Complement*. O nó com coordenadas binárias $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ envia pacotes para o nó $\bar{a}_{n-1}, \bar{a}_{n-2}, \ldots, \bar{a}_1, \bar{a}_0$. É feita a inversão de todos os *bits*.

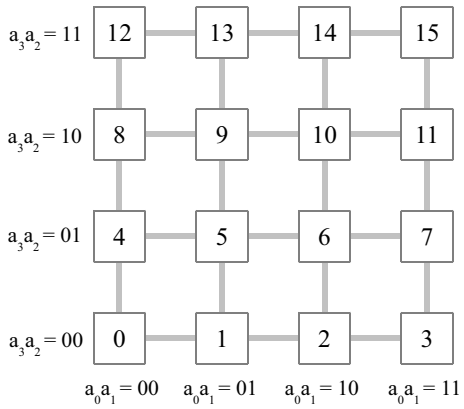


Figura 32: Endereçamento binário. O valor em decimal de cada posição em uma malha 4×4 é definido pelo endereço binário das coordenadas $a_3a_2a_1a_0$.

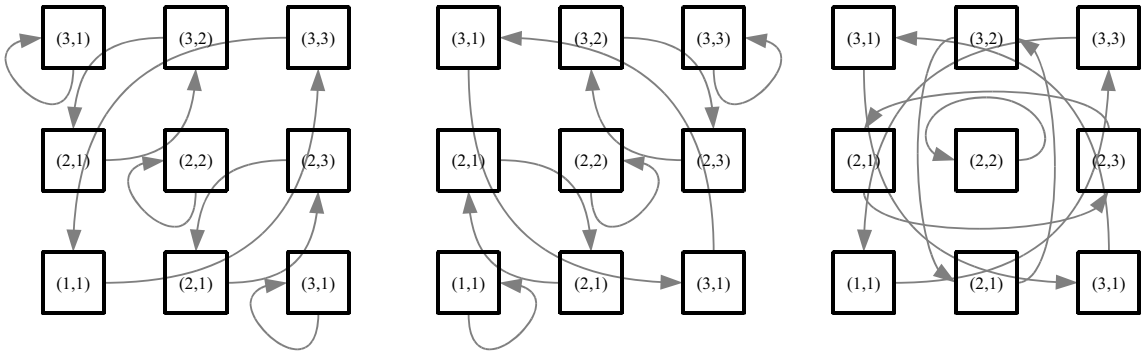
Seis padrões de tráfego foram adotados para a realização das simulações: uniforme, complemento, matriz transposta 1, matriz transposta 2, local e *hotspot*. Exceto pelo padrão local, todos os demais padrões selecionam o nó de origem, (x_i, y_i), de forma aleatória seguindo uma distribuição uniforme. Os padrões uniforme, local e *hotspot* realizam a seleção do nó de destino de forma aleatória, seguindo cada um uma regra específica, como será descrito a seguir. Já os padrões complemento e transposta 1 e 2 fazem esta escolha de forma determinística a partir do endereço do nó de origem.

No padrão uniforme, tanto os nós de origem quanto os nós de destino são escolhidos de forma aleatória. Todos os nós possuem a mesma probabilidade de serem escolhidos, isto é, a escolha é baseada em uma distribuição uniforme de probabilidades. Assim, na distribuição uniforme, qualquer tipo de combinação entre origem e destino é possível. É um modelo de tráfego considerado padrão para testes em redes, por ser capaz de melhor balancear a carga por todos os nós da rede.

Entretanto, em aplicações reais, é grande a possibilidade de alguns nós se comunicarem mais com alguns do que com outros. Nesse sentido, é necessário o uso de padrões não uniformes para simular um comportamento similar ao de tais aplicações. Dois padrões com esta característica de não-uniformidade são utilizados: o padrão *hotspot* (PFISTER; NORTON, 1985) e o padrão local. Para o padrão *hotspot*, 10% de todos os pacotes possuem o mesmo nó de destino. Este é um nó sobrecarregado, ou *hotspot* (do inglês, ponto quente). Os demais 90% são escolhidos seguindo o padrão uniforme. O padrão local escolhe as coordenadas do nó de origem de um pacote aleatoriamente dentro do intervalo $[1, size - 1]$. O nó de destino é escolhido aleatoriamente apenas no entorno do nó de origem (x_i, y_i). Assim, a coordenada x do nó de destino pode ser escolhida no intervalo $[x_i - 1, x_i + 1]$, enquanto que a coordenada y

é escolhida no intervalo $[y_i - 1, y_i + 1]$.

No padrão complemento, o nó de destino do pacote é função do endereço do nó de origem, possuindo coordenadas $(size - x_i + 1, size - y_i + 1)$. No padrão matriz transposta 1, o nó de destino é definido pelas coordenadas $(size - y_i + 1, size - x_i + 1)$. No padrão matriz transposta 2, a coordenada do nó de destino é (y_i, x_i). Os padrões matriz transposta 1 e 2 representam o reflexo dos nós de origem sobre as retas $y = -x$ e $y = x$, respectivamente. Para estes três padrões, cada nó de origem se comunica com um único nó de destino, como visto na Figura 33.



(a) *Matriz transposta 1.* (b) *Matriz transposta 2.* (c) *Complemento.*

Figura 33: Possíveis nós de origem e destino em uma malha 3×3 usando os padrões de tráfego.

5.1.2 Taxa de injeção de pacotes

A taxa de injeção de pacotes relaciona o tempo decorrido com a transmissão dos *flits* de um IP para uma chave e o tempo inativo entre o fim da transmissão de um pacote e o início da transmissão do pacote seguinte. A taxa de injeção é uma fração da largura de banda total de um canal da rede.

Sejam, por exemplo, os *pacotes 1 e 2* ilustrados pela Figura 34(a), compostos por 5 *flits* cada. O período inativo entre a transmissão do último *flit* do pacote 1 e o primeiro *flit* do pacote 2 é de 5 ciclos de *clock* da rede. Como este é o mesmo período gasto para a transmissão de um pacote inteiro (cada *flit* é transmitido em um ciclo), então é dito que a taxa de injeção é de 50% - 5 ciclos gastos com a transmissão do pacote e 5 ciclos de rede ociosa. Apenas metade da capacidade total de transmissão é utilizada. Já a Figura 34(b) exemplifica a injeção com taxa de 100%: nenhum ciclo inativo é usado entre a transmissão de dois pacotes.

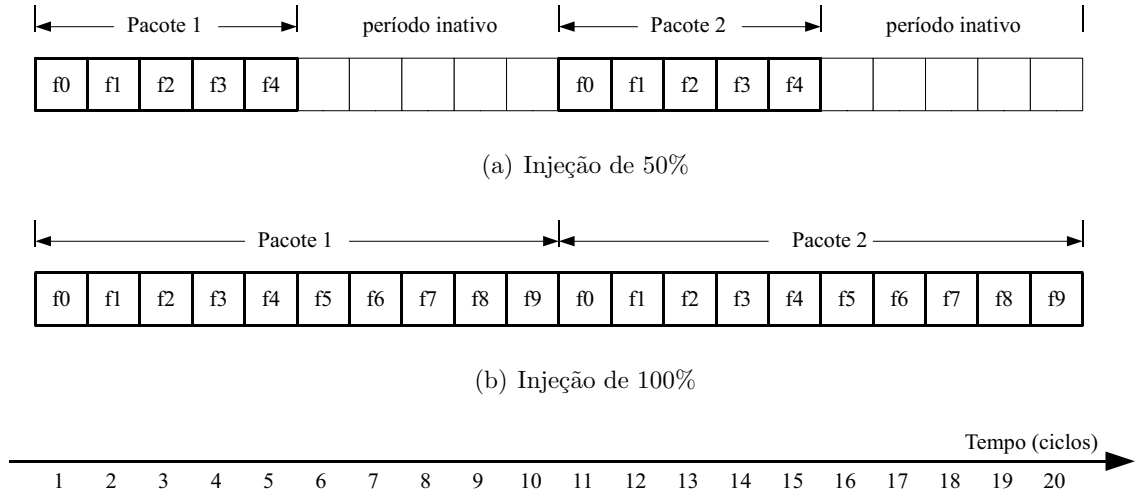


Figura 34: Distribuição de *flits* e pacotes em diferentes taxas de injeção.

A utilização de taxas inferiores a 100% é interessante em situações de congestionamento da rede. Na Figura 35, o congestionamento da rede acarreta em um período de espera de 3 ciclos para o pacote 1 e 2 ciclos para o pacote 2 (representado pelas regiões cinzas). Embora exista um aumento de 5 ciclos na transmissão dos pacotes, este período é compensado pelo tempo ocioso entre cada pacote. No presente trabalho, para uma dada taxa de injeção, é considerado fixo o período entre o início da transmissão de dois pacotes consecutivos. Porém, o período ocioso entre pacotes se adapta para diferentes condições de atraso na rede.

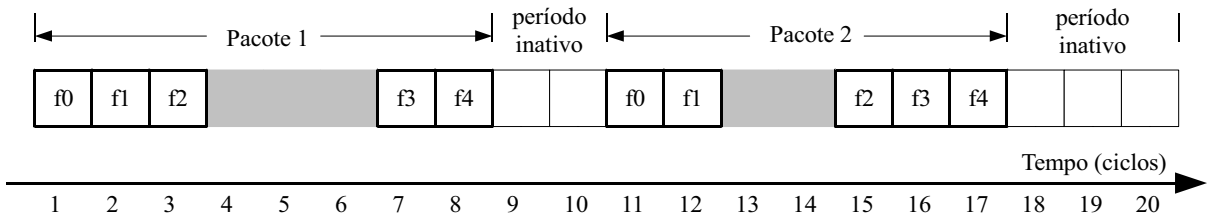


Figura 35: Taxa de 50%: o período inativo é usado para a transmissão de de *flits* atrasados devido a congestão da rede.

No exemplo ilustrado pela Figura 35, os atrasos não afetam a latência da rede. Por outro lado, se o atraso ocasionado for superior ao período inativo da rede, então haverá um aumento da latência. Nesta situação, o período compreendido entre o primeiro *flit* de dois pacotes consecutivos não é fixo: o pacote seguinte só iniciará sua transmissão após a injeção do último *flit* do pacote atual. Este caso é bem visível com o uso de uma taxa de injeção de 100%, onde qualquer atraso resulta em um aumento da latência total de transmissão. A Figura 36 ilustra esta situação. Durante 3 ciclos, a transmissão é interrompida devido a congestionamentos;

por não haver tempo ocioso entre pacotes, o resultado é o atraso no nicio da transmissão dos pacotes seguintes.

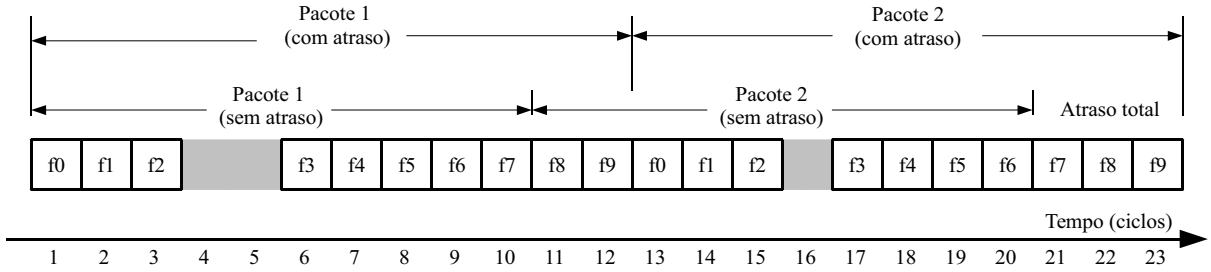


Figura 36: Taxa de 100%: os atrasos de congestionamento acarretam em um aumento real da latência.

5.1.3 Tamanho dos pacotes

O tamanho dos pacotes também pode ser modelado de diversas formas. Dois valores devem ser diferenciados: o tamanho de um pacote e sua quantidade de *flits*. O tamanho L de um pacote é o valor de seu comprimento total em *bits*. Já a quantidade de *flits*, definida pela Equação 34, é o maior valor inteiro resultante da divisão do tamanho do pacote pelo tamanho de um *phit*[1].

$$quantidade\ de\ flits = \left\lceil \frac{L}{W} \right\rceil \tag{34}$$

Geralmente, o comprimento dos pacotes é definido como constante em simulações (DUATO; YALAMANCHILI; LIONEL, 2002). Ou então, o comprimento pode ser feito variável de uma simulação para outra, quando se deseja estudar os efeitos de pacotes de diferentes tamanhos sobre a rede. Nesta situação, o tamanho pode ser escolhido de forma aleatória seguindo uma distribuição de probabilidades específica, como é o caso da distribuição espacial de pacotes.

Neste trabalho, a quantidade de *flits* dos pacotes usados nas simulações está associada com a taxa de injeção. A quantidade de ciclos entre o início da transmissão de um pacote e o início da transmissão do seguinte é definida como o valor fixo de 20 ciclos. Assim, a quantidade de *flits* varia conforme a taxa de injeção desejada, como visto na Tabela 4.

Tabela 4: Quantidade de *flits* para cada taxa de injeção.

Taxa de injeção	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Quantidade de *flits*	02	04	06	08	10	12	14	16	18	20
Ciclos inativos	18	16	14	12	10	08	06	04	02	00

[1]Considerando que $1flit = 1phit$.

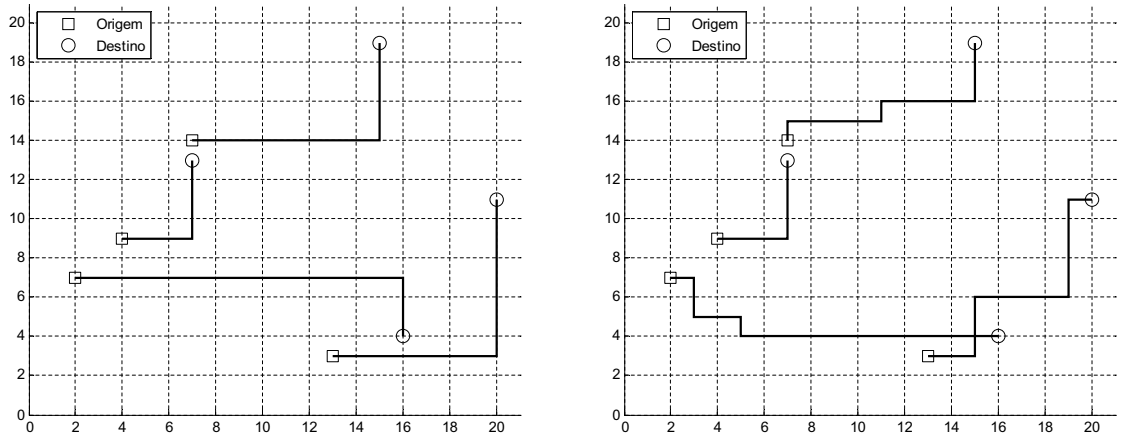
5.2 Algoritmos para comparação

As simulações apresentadas neste capítulo tem por fim ilustrar as características de uma rede usando os algoritmos de roteamento propostos, REAS e RACS. Para auxiliar a análise de desempenho, é feita uma comparação da NoC empregando dois outros modelos de roteamento, baseados no algoritmo XY e no *Odd-Even* (OE). Em ambos os casos, foi utilizada uma modelagem de rede embutida semelhante a apresentada na Seção 4.1. Desta forma, os roteamentos XY e OE foram utilizados no ambiente Matlab com uma estrutura de rede idêntica a empregada pelo REAS e o RACS.

O funcionamento destes dois algoritmos foi bem debatida no Capítulo 2 desta dissertação. O roteamento XY segue o Algoritmo 1 apresentado na Seção 2.1. Já o roteamento baseado no modelo *Odd-Even* utiliza a função *ROUTE* descrita pelo Algoritmo 2 da Seção 2.5. A função *ROUTE* constrói, a cada passo, um conjunto de possíveis nós seguintes a partir das regras do modelo *Odd-Even*. Para o roteamento OE usado, a seleção do nó seguinte dentro do conjunto de nós é feita de forma aleatória: se dois nós podem ser selecionados, a probabilidade de cada um ser escolhido é de 50%.

A escolha destes dois algoritmos se baseia no fato de se tratar de um roteamento determinístico (XY) e um parcialmente adaptativo (OE). Também foi considerado importante a comparação dos algoritmos propostos com modelos de rotamento mínimos e livres de *deadlocks*, como é o caso do XY e do OE.

A Figura 37 exemplifica a execução simples dos roteamentos XY e OE na construção de rotas para quatro pacotes em uma malha 20×20. Nota-se, em ambos os casos, a natureza mínima dos roteamentos.



(a) Roteamento XY (b) Roteamento OE

Figura 37: Exemplo de trajetórias calculadas.

5.3 Simulações com padrões sintéticos

Nesta seção são apresentados os resultados para as simulações de roteamento empregando padrões sintéticos de tráfego. Todos os algoritmos foram executados com o *software* Matlab Version 7.7.0.471 (R008b). As simulações foram realizadas em PCs (*Personal Computers*) com processador Intel Core i7 950 operando em 3Ghz, memória *ram* de 8Gb e sistema operacional Microsoft Windows 7 Home Premium.

Cada execução foi realizada utilizando um parâmetro diferente, conforme apresentado pela Tabela 5. Foram realizadas testes com quatro algoritmos de roteamento, seis padrões de tráfego, dez valores de taxa de injeção e dez distintas quantidades de pacotes[2], totalizando assim 2400 simulações. Para todos os testes, a rede adotada foi uma malha 5×5.

Tabela 5: Parâmetros das simulações.

Roteamento	REAS, RACS, XY e OE
Padrões de Tráfego	Uniforme, *Hotspot*, Local, Complemento, Trans. 1 e Trans. 2
Taxa de injeção	10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% e 100%
Quantidade de pacotes	10, 20, 30, 40, 50, 60, 70, 80, 90 e 100

5.3.1 Resultados obtidos

Para cada simulação, obteve-se o valor de latência total e o valor de latência média por pacote. A latência total é a soma da latência individual de todos os pacotes que circulam pela rede, onde a latência individual é o número de ciclos do simulador decorridos desde a injeção do primeiro *flit* de um pacote até o início da injeção do pacote seguinte. Já a latência média é o valor obtido de latência total dividido pela quantidade de pacotes. O objetivo geral destes testes é verificar a variação da latência sob diferentes taxas de injeção e diferentes quantidades de pacotes. Para um mesmo padrão de tráfego, esta variação é comparada para cada um dos quatro algoritmos de roteamento. Visando reduzir a quantidade de dados e facilitar a análise, foi feita a média dos valores de latência para diferentes quantidades de pacotes e diferentes taxas de injeção . Desta forma, são obtidos gráficos de *latência por pacote* $\times$ *taxa de injeção* e *latência por pacote* $\times$ *quantidade de pacotes*. Estes gráficos são vistos nas Figuras 38, 39, 40, 41, 42 e 43, com cada figura representando um dos seis padrões de tráfego adotados. Os valores numéricos obtidos nestas simulações podem ser consultados no Apêndice A desta dissertação, com as Tabelas 7, 9, 11, 13, 15 e 17 sendo as médias para diferentes quantidades de pacotes e as Tabelas 8, 10, 12, 14, 16 e 18 as médias para diferentes quantidades de pacotes.

[2]Inseridos simultaneamente na rede a partir de diferentes nós de origem.

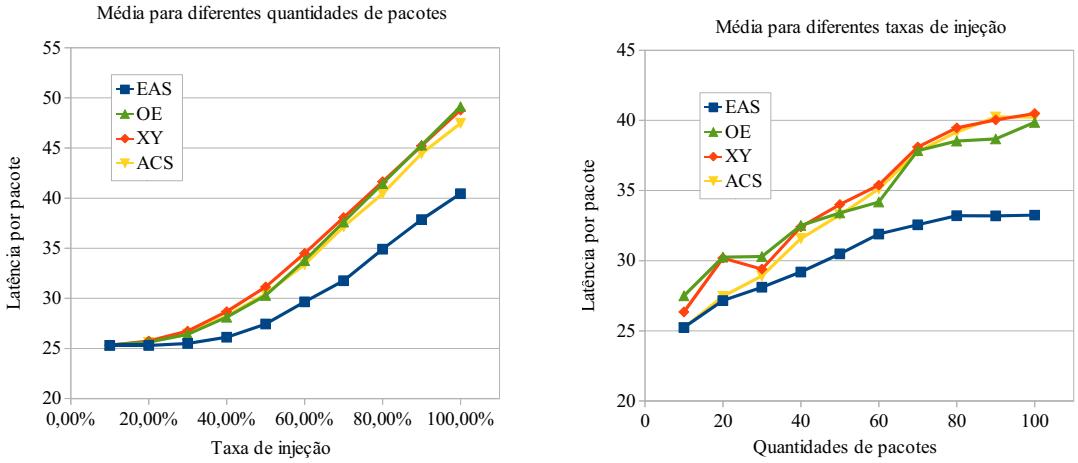


Figura 38: Resultados para Padrão Uniforme.

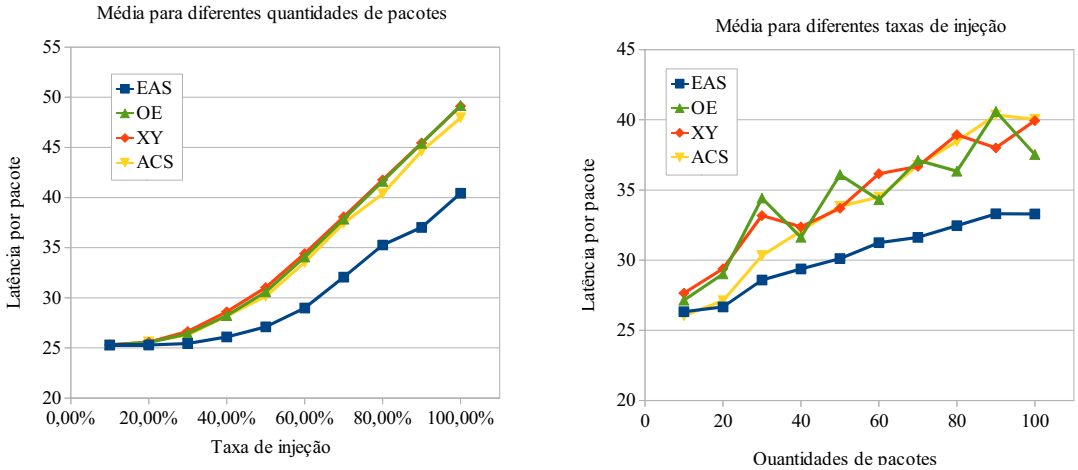


Figura 39: Resultados para Padrão *Hotspot*.

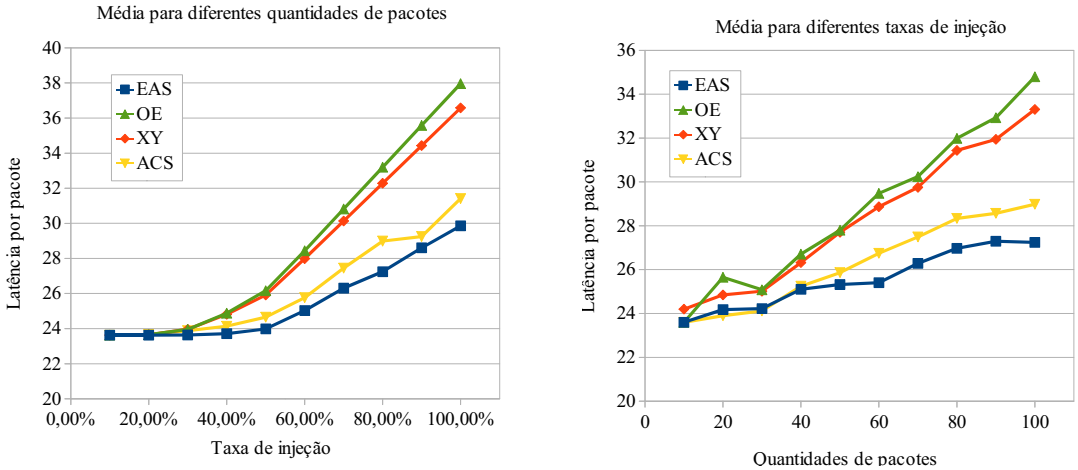


Figura 40: Resultados para Padrão Local.

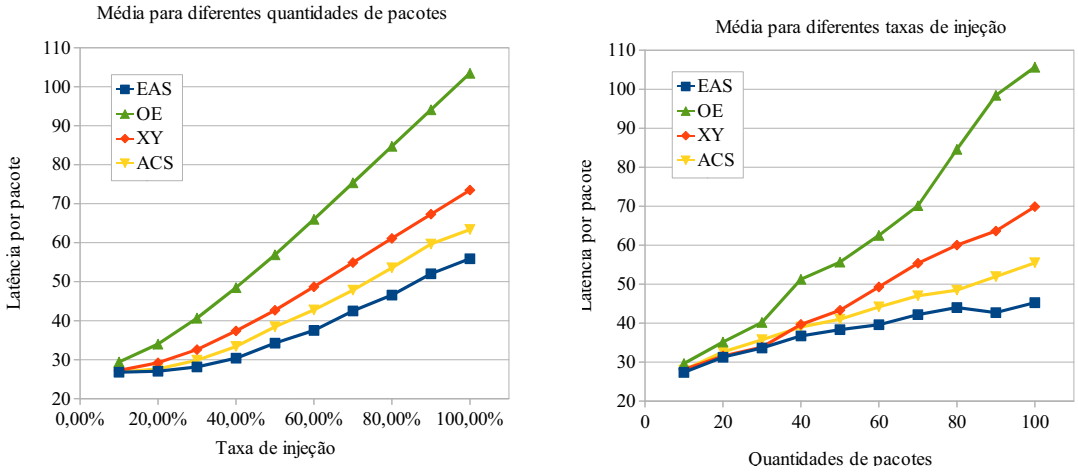


Figura 41: Resultados para Padrão Complemento.

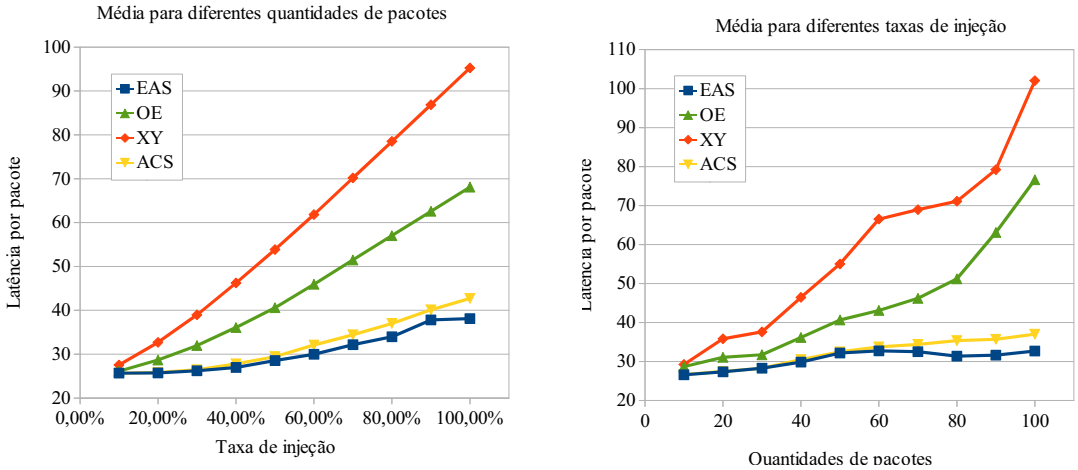


Figura 42: Resultados para Padrão Transposta 1.

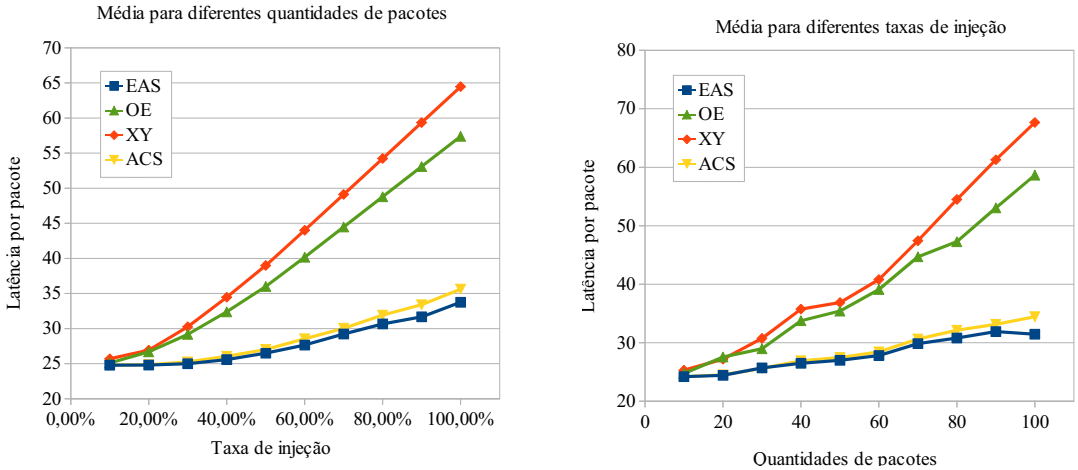


Figura 43: Resultados para Padrão Transposta 2.

5.3.2 Análise dos resultados

Em todos os gráficos, observa-se a superioridade do método proposto sobre os demais roteamentos. Para todos os padrões de tráfego, a curva de latência do REAS apresenta-se abaixo das curvas dos demais métodos, indicando sua capacidade de busca por rotas que proporcionam um menor tempo de transmissão.

No padrão uniforme, o REAS apresenta um desempenho pouco superior ao obtido pelos demais métodos em baixas taxas de injeção (e em baixa quantidade de pacotes). A diferença entre o REAS e os demais aumenta, mantendo-se aproximadamente constante a partir de uma taxa de 60% e de 70 pacotes injetados. Os demais métodos de roteamento apresentam, para o tráfego uniforme, todos um mesmo padrão.

O comportamento da rede sob o padrão de tráfego *hotspot* mostrou-se semelhante ao obtido com o padrão uniforme. Como este padrão de tráfego é, em parte, formado por pacotes seguindo o padrão uniforme (excluindo os 10% de pacotes destinados para ao *hotspot*), possivelmente esta característica tenha sido significativa durante os testes.

No padrão local, a diferença de desempenho foi acentuada. O REAS e o RACS apresentaram um comportamento semelhante entre si, mostrando-se melhores em relação aos roteamentos XY e OE.

Para o padrão complemento, cada método apresentou uma característica diferente. O REAS obteve um menor valor de latência, seguido pelo RACS, XY e OE. Para este padrão, o roteamento baseado no modelo *Odd-Even* obteve um desempenho muito inferior aos demais, incluindo o roteamento XY. O padrão complemento é caracterizado por impor que os pacotes atravessem a bisseção da rede (RAHMAN; HORIGUCHI, 2004), o que eventualmente resulta em um alto tráfego próximo aos nós centrais da malha. O roteamento OE, como visto na Figura 37, transporta pacotes por percursos mínimos realizando inúmeras mudanças de direção (em oposição ao XY, em que apenas uma mudança de direção ocorre). Este elevado número de giros tende a direcionar o pacote por nós diagonalmente entre origem e destino. Este fato, combinado com a característica do padrão complemento, explica o elevado valor de latência encontrado para os testes realizados.

Ambos os padrões de matriz transposta obtiveram resultados semelhantes: o REAS sendo pouco melhor que o RACS e ambos sendo melhores que os roteamentos XY e OE. Para o padrão transposta 1, o roteamento OE obteve um desempenho superior ao XY; já no padrão transposta 2, tanto XY quanto OE obtiveram desempenhos similares.

5.4 Considerações Finais do Capítulo

A comunicação entre núcleos IP de uma NoC pode ocasionar padrões característicos de transmissão de pacotes, dependendo da aplicação a ser executada pela rede. Assim, nós podem se comunicar mais com nós distantes na rede, ou apenas com nós vizinhos. Ou ainda, vários nós podem se comunicar com um mesmo nó de destino. Este comportamento motivou a criação de padrões sintéticos de comunicação: grupos de pacotes sem nenhum significado computacional, criados exclusivamente para testes, cuja geração segue regras de forma a mimetizar os padrões de comunicação de aplicações reais. Testes com parões sintéticos auxiliam na avaliação de desempenho de uma dada característica da rede (no caso do presente trabalho, a latência).

Neste capítulo, foram apresentados diferentes padrões de geração de tráfego comumente usados em simulações de redes de comunicação. Desta forma, seis padrões foram adotados para a avaliação dos algoritmos de roteamento propostos. Testes com diferentes configurações foram realizadas, de forma a destacar a variação da latência em função da taxa de injeção e da quantidade de pacotes. Foram comparados os resultados da rede utilizando os algoritmos de roteamento propostos (REAS e RACS) e roteamentos baseados em algoritmos amplamente adotados na literatura (XY e *Odd-Even*). Como esperado, os roteamentos adaptativos baseados na meta-heurística ACO obtiveram um desempenho superior aos demais. O capítulo seguinte introduz a ideia de roteamento de pacotes para aplicações, apresentando resultados para aplicações sintéticas aleatórias e para aplicações reais.

Capítulo 6

TESTES COM APLICAÇÕES

QUANDO se pensa no funcionamento de uma NoC, em geral se tem em mente a aplicação para qual a rede foi projetada. De fato, uma rede embutida é modelada visando uma aplicação de destino: características tanto da NoC quanto da aplicação são balanceadas de forma que a execução se dê de forma mais eficiente.

No Capítulo 5, testes envolvendo padrões de tráfego foram exaustivamente abordados. Ainda assim, o tráfego do mundo real pode eventualmente exibir padrões diferentes daqueles apresentados por padrões sintéticos (VARATKAR; MARCULESCU, 2004). Dessa forma, a fim de validar os algoritmos de roteamento propostos em testes mais realísticos, são necessárias simulações envolvendo o uso de aplicações.

A Seção 6.1 apresenta uma introdução teórica sobre a modelagem de aplicações e como este modelo é empregado em sistemas computacionais. Na Seção 6.2 o é descrita a forma com que uma dada aplicação é associada ao modelo de rede exposto no Capítulo 4. Na Seção 6.3 são apresentados os resultados de um conjunto de simulações. É realizado o roteamento para redes empregando aplicações sintéticas geradas aleatoriamente.

6.1 Modelos de aplicações

De uma forma geral, NoCs são desenvolvidas para a execução de uma aplicação específica. Esta aplicação pode ser descrita inicialmente como um *software* que deve ser embarcado em um *hardware*. Contudo, o paradigma de redes embutidas tem como principal característica o paralelismo que estas proporcionam. Dessa forma, o software em questão deve também possuir como característica a capacidade de, pelo menos em algum momento de sua execução, ser realizado de forma independente e simultânea em diversas partes da rede. Trata-se de uma metodologia sob dois pontos de vista: modela-se o *software* pensando-se nas limitações de *hardware* e modela-se o *hardware* tendo em vista alcançar a execução do software.

Toda aplicação, seja ela para qualquer tipo de sistema computacional, pode ser descrita por um *grafo de tarefas*. Esta é uma estrutura de dados em que a aplicação como um todo é dividida em blocos responsáveis por tarefas especificas. Estes blocos, por sua vez trocam informações entre si a fim de completar a execução da aplicação. A quantidade de blocos poderá ser maior ou menor, dependendo do nível de abstração adotado. Assim, o grafo de tarefas é denotado por $GT = G(T, D)$, um grafo direcionado, com pesos e acíclico. Cada vértice de T representa uma tarefa, ou um módulo de processamento da aplicação. Em geral, uma tarefa é uma operação bem definida, como um cálculo matemático ou uma codificação de dados. Cada arco do conjunto D caracteriza a dependência de dados entre duas tarefas. Por exemplo, em uma tarefa representando a operação de soma, os arcos de entrada fornecem os dados dos operandos, enquanto que o arco de saída retorna o resultado obtido.

A Figura 44 ilustra um exemplo de grafo composto por cinco tarefas. Cada elemento do grafo possui uma identificação do tipo x/y, onde x é o número do vértice ou arco e y é o seu tipo. Vértices do mesmo tipo, como os vértices 0 e 4, possui as mesmas características. O mesmo vale para os arcos de numeração 0 e 4. As tarefas 0, 1, 3 e 4 (ou 0, 2, 3 e 4) são sequenciais: a tarefa seguinte apenas inicia após o término das tarefas antecedentes. Todas estas tarefas podem ser executadas por um mesmo elemento computacional, desde que este seja capaz de realizar diferentes tipos de tarefas. Em outras palavras, esta sequência de tarefas pode ser executada por um mesmo núcleo, ou cada tarefa por um núcleo diferente, não havendo uma diferença no tempo de execução[1]. Já as tarefas 1 e 2 não possuem dependência entre si. Desta forma, podem ser executadas em paralelo, isto é, cada tarefa pode ser executada por um elemento processador diferente. Assume-se que os vértices sem predecessores recebem dados do ambiente externo. Do mesmo modo, assume-se que aqueles sem sucessores geram dados de saída para o ambiente externo.

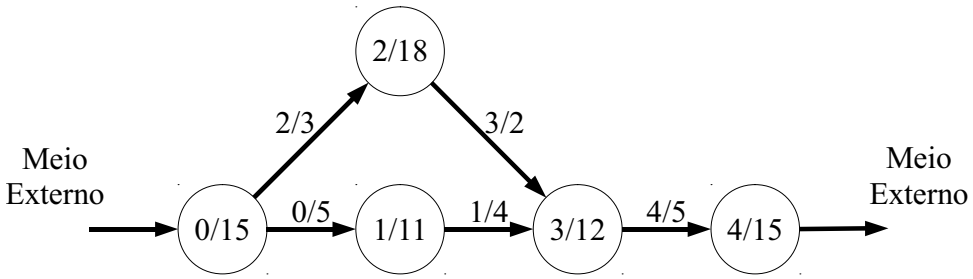


Figura 44: Exemplo de grafo de tarefas.

[1]Desconsiderando o tempo de comunicação entre tarefas.

6.2 Redes com aplicações

Na presente dissertação, deseja-se realizar o roteamento de pacotes transmitidos entre nós de uma NoC executando uma dada aplicação. Assim, é necessário que as informações da aplicação sejam transmitidas para o modelo de rede exposto nos Capítulos 4 e 5.

6.2.1 Codificação do grafo de tarefas

Duas informações acerca de um grafo de tarefas são essenciais para o problema de roteamento: o tempo em ciclos de *clock* de execução de uma tarefa e o fluxo de *bits* entre duas tarefas. Na simulação de rede usada até então, o tempo de execução pode ser associado à t_INICIO (apresentado na Tabela 1), sendo uma variável que indica em quantos ciclos a transmissão de *flits* é iniciada. Já o fluxo de *bits* pode ser empregado como o parâmetro L da Equação 34, definindo assim a quantidade de *flits* do pacote.

O formato com que os dados de um grafo de tarefas estão organizados neste trabalho é ilustrado pelo Algoritmo 11. O grafo é composto por duas estruturas: uma com informações de cada vértice e outra com informações dos arcos. As variáveis de cada estrutura são vetores, onde cada posição indica que o valor está associado a um vértice (ou arco) diferente.

Algoritmo 11 Função representativa de um grafo de tarefas, $GT()$

1: **Estrutura** $Vertice$ {
2: $ID \leftarrow v_1,\ v_2,\ v_3,\ \ldots$
3: $nvel \leftarrow n_1,\ n_2,\ n_3,\ \ldots$
4: $x \leftarrow x_1,\ x_2,\ x_3,\ \ldots$
5: $y \leftarrow y_1,\ y_2,\ y_3,\ \ldots$
6: $t\_COMP \leftarrow tc_1,\ tc_2,\ tc_3,\ \ldots$
7: $t\_INICIO \leftarrow ti_1,\ ti_2,\ ti_3,\ \ldots$
8: } **Fim Estrutura**
9: **Estrutura** $Arco$ {
10: $origem \leftarrow origem\_arco_1,\ origem\_arco_2,\ origem\_arco_3,\ \ldots$
11: $destino \leftarrow destino\_arco_1,\ destino\_arco_2,\ destino\_arco_3,\ \ldots$
12: $fluxo \leftarrow fluxo\_arco_1,\ fluxo\_arco_2,\ fluxo\_arco_3,\ \ldots$
13: } **Fim Estrutura**
14: **Retorna** Vertice, Arco;

A variável ID é a identificação (ou índice) de cada vértice. Já a variável *nvel* organiza sequencialmente os vértices: tarefas com valor de nível menor são executadas antes daquelas com valores maiores; se dois vértices possuem mesmo valor de nível, estes podem ser executados de forma simultânea. Esta é uma variável que auxilia na identificação das tarefas sequenciais e das paralelas.

Os vetores *origem* e *destino* armazenam, para cada arco, o valor de ID de seus respectivos vértices de de origem e destino.

6.2.2 Mapeamento aleatório

A estrutura de vértices apresentada no Algoritmo 11 possui mais vetores de dados não descritos na Seção 6.2.1. Estas não são informações específicas do grafo de tarefas, mas sim informações transmitidas para o grafo pelo algoritmo de simulação da rede.

O vetor t_COMP identifica o tempo, em ciclos de *clock*, que uma dada tarefa necessita para ser executada pelo núcleo IP da NoC para o qual foi alocada. Já os valores de x e y mostram as coordenadas da rede (considerando ser empregada a topologia malha 2D) onde cada tarefa foi mapeada. Assim, t_COPM, x e y podem ter seus valores alterados dependendo das características de *hardware* da NoC, sem contudo, alterar as características próprias da aplicação.

Como foi visto na Seção 1.4, os processos de alocação e mapeamento são duas etapas de projeto localizadas entre a especificação inicial e a implementação final da rede. A Figura 13 ilustra esta sequência de etapas. Embora apenas a alocação e o mapeamento estejam indicados, diversos outros métodos podem ser acrescentados com o intuito de otimizar o funcionamento conjunto da aplicação e do *hardware* da rede. Assim, até mesmo o roteamento estático descrito nesta dissertação pode ser incluído entre a especificação e a implementação como um processo de melhoria do sistema.

Tanto a alocação de IPs quanto o mapeamento de tarefas são assuntos com grande interesse de estudo. Em geral, no desenvolvimento de NoCs, utiliza-se métodos que otimizem estes dois processos, tal como apresentado em (SILVA, 2009). Contudo, no presente trabalho não se deu muita ênfase nestes processos, uma vez que o interesse maior está no estudo do roteamento da rede. De fato, por ser o roteamento a estar sendo otimizado, este deve ser capaz de obter bons resultados independente do mapeamento adotado. Desta forma, utilizou-se nesta dissertação um *mapeamento aleatório de tarefas*. Quanto a alocação, é considerado que os grafos utilizados já possuem previamente a informação sobre os IPs.

No mapeamento aleatório, cada vértice do *grafo de caracterização de aplicação* é associado a um *vértice do grafo de caracterização da arquitetura*. A forma com que esta associação é feita é aleatória: um vértice do grafo da aplicação seleciona de uma lista um vértice do grafo da arquitetura; a posição escolhida é removida da lista e o processo se repete, até que todos os vértices do grafo da aplicação tenham uma posição definida no grafo da arquitetura.

O mapeamento adotado também define a quantidade de nós que a NoC possui. Assume-se que a quantidade n de nós da malha deve ser o suficientes para mapear uma aplicação com p tarefas. Dessa forma, por estar sendo usada uma malha 2D, a relação entre n e p é definida pela Equação 35

$$n = \lceil \sqrt{p} \rceil^2 \tag{35}$$

Um exemplo de mapeamento aleatório é ilustrado pela Figura 45. Para um grafo composto por cinco tarefas, a arquitetura de destino possuirá nove núcleos, conforme definido pela Equação 35. Cada tarefa é mapeada em um nó diferente da rede; as hachuras denotam os nós onde nenhuma tarefa está mapeada.

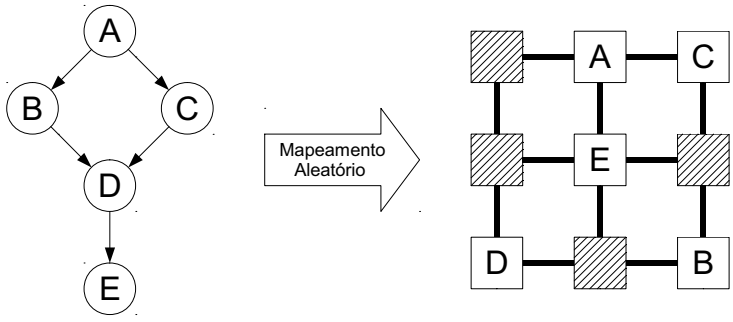


Figura 45: Exemplo de mapeamento aleatório para uma aplicação com cinco tarefas.

6.2.3 Simulação da rede incluindo mapeamento

Nesta seção é descrito o modelo para o cálculo do tempo total de execução de uma aplicação em uma NoC. O tempo de execução total consiste da soma do tempo de execução de todas as tarefas pertencentes ao *caminho crítico* de um grafo com o tempo de comunicação de cada arco conectando estas tarefas. O caminho crítico é aquele, dentre todos os caminhos entre os vértices inicial e final, que possui maior tempo de execução e comunicação. Seja por exemplo o grafo ilustrativo da Figura 44. Se o valor de y da representação x/y for considerado como sendo os tempos, em ciclos, de execução (para os vértices) e de comunicação (para os arcos), então o percurso composto pelos vértices 0, 2, 3 e 4 será o caminho crítico. Isto pois o tempo total de execução será $15+3+18+2+12+5+15 = 70$ ciclos, contra $15+5+11+4+12+5+15 = 67$ ciclos do caminho 0, 1, 3 e 4.

O modelo proposto consiste de um algoritmo iterativo, onde em cada etapa são selecionados arcos que podem transmitir pacotes paralelamente pela rede. Arcos com esta característica são todos aqueles originados de vértices com mesmo valor de nível. Tomando novamente como exemplo a Figura 44, os dois arcos originados do vértice 0 realizam, em paralelo, comunicação com os nós 1 e 2. A tarefa 0 é dita de nível 1. Também os arcos provenientes dos

vértices 1 e 2 poderão transmitir, em paralelo, pacotes para o vértice 3. Assim, 1 e 2 serão tarefas de nível 2.

O Algoritmo 12 mostra a versão em pseudocódigo do modelo feito no ambiente Matlab. A principal característica deste algoritmo está no fato dos quatro métodos de roteamento apresentados no Capítulo 5 serem chamados como funções. Informações dos arcos que precisam ser roteados, como nós de origem e destino e o tamanho do pacote, são passadas para estas funções como parâmetros. A função de roteamento, por sua vez, fornece ao algoritmo informações do percurso calculado, como a sequência de chaves e o tempo necessário para comunicação.

Algoritmo 12 Modelo para mapeamento e roteamento

Entrada $vertice, arco \leftarrow GT()$
1: Define o tamanho da rede a partir da quantidade de vértices;
2: Mapeia aleatoriamente cada vértice em um nó da rede;
3: Extrai o nível de cada vértice;
4: $max\_niv \leftarrow$ valor máximo de nível;
5: $num\_arc \leftarrow$ quantidade de arcos;
6: **Para** $i = 1 \rightarrow max\_niv - 1$ **Faça**
7: $lista \leftarrow$ arcos cujo vértice de origem possui nível i;
8: $num\_lista \leftarrow$ Quantidade de arcos de $lista$;
9: **Se** $num\_lista > 1$ **Então**
10: Extrai o fluxo de cada arco de $lista$;
11: Organiza $lista$ de acordo com os fluxos;
12: **Para** $j = 1 \rightarrow num\_lista$ **Faça**
13: Extrai $x_i, y_i, t\_INICIO, t\_COMP$ do vértice de origem de $lista(j)$;
14: Extrai x_f, y_f do vértice de destino de $lista(j)$;
15: **Fim Para**
16: $Fb \leftarrow ROTEAMENTO(\ )$;
17: Organiza Fb com a ordenação original de $lista$;
18: Atualiza t_INICIO dos vértices de destino dos arcos de $lista$ com Fb;
19: **Senão**
20: Extrai $x_i, y_i, t\_INICIO, t\_COMP$ do vértice de origem de $lista$;
21: Extrai x_f, y_f do vértice de destino de $lista$;
22: $Fb \leftarrow ROTEAMENTO(\ )$;
23: Atualiza t_INICIO do vértice de destino de $lista$ com Fb;
24: **Fim Se**
25: **Fim Para**
26: **Para** o último vértice de GT **Faça**
27: $tempo\_exec \leftarrow t\_COMP + t\_INICIO$;
28: **Fim Para**
29: **Retorna** $tempo\_exec$;

Outro detalhe importante presente no Algoritmo 12 é a ordenação dos pacotes que precisarão ser roteados. Os quatro modelos de roteamento considerados - EAS, ACS, XY e OE - utilizam como arbitragem uma política FCFS (*First-Come, First-Served*), isto é, a ordem

dos pacotes define sua prioridade. Contudo, esta não é, necessariamente, a melhor opção. Sejam dois pacotes cujos cabeçalhos disputam a mesma saída de uma chave; o tempo total de transmissão destes dois pacotes através da chave será a soma do tempo de transmissão de cada pacote mais o atraso imposto pelo pacote com maior prioridade no de menor prioridade. Desta forma, é interessante que pacotes com menor quantidade de *flits* possuam maior prioridade, pois acarretarão em um menor atraso. Para o Algoritmo 12, o conjunto de arcos que representa os pacotes que serão roteados é ordenado conforme seus respectivos valores da variável $fluxo$. Os resultados obtidos com o roteamento são então, reorganizados em sua ordem original.

6.3 Testes com aplicações sintéticas

Para as simulações apresentadas nesta seção, foi criado um conjunto de *aplicações sintéticas*. Uma aplicação sintética é aquela cujas características do grafo de tarefas, como a quantidade de vértices, são definidas aleatoriamente dentro de um intervalo.

Na criação do conjunto de aplicações para teste foi utilizada a ferramenta *Task Graph For Free* - TGFF (DICK; RHODES; WOLF, 1998). O TGFF é um gerador de grafos pseudo-aleatório, de uso geral e controlável pelo usuário. Os principais parâmetros fornecidos ao TGFF para a criação dos grafos de teste são a quantidade de níveis e a quantidade de vértices por nível. Também é especificada uma faixa para o tempo de execução o tamanho que cada pacote. Assim, cada vértice pode assumir, aleatoriamente, um valor de tempo de execução de 1000, 1100, 1200, 1300 ou 1400 ciclos. Estes mesmos valores são usados para o tamanho, em *bits*, do pacote transmitido por cada arco.

Cinco conjuntos de grafos foram criados com o auxílio do TGFF, chamados de $Ex1$, $Ex2$, $Ex3$, $Ex4$ e $Ex5$. Um conjunto é composto por dez grafos de tarefas distintos. Cada grafo é simulado, para cada um dos quatro algoritmos de roteamento, com dez mapeamentos aleatórios diferentes. Assim, tem-se um total de 2000 simulações. Os grafos das 50 aplicações geradas são apresentados no Apêndice C desta dissertação.

Para os conjuntos $Ex2$, $Ex3$, $Ex4$ e $Ex5$, o algarismo indica a quantidade de tarefas pertencentes a um mesmo nível. Cada grafo de um conjunto possui uma quantidade diferente de níveis, indo de 1 até 10 níveis. O conjunto de grafos $Ex1$ difere um pouco dos demais: neste, cada grafo possui uma quantidade diferente de tarefas. Esta quantidade varia de 10 até 100 tarefas. Sendo definida apenas a quantidade de tarefas, os grafos do conjunto $Ex1$ são gerados com uma diferente quantidade de níveis e de vértices por nível.

6.3.1 Resultados obtidos

Esta seção apresenta os resultados obtidos para as simulações de roteamento para os cinco conjuntos de aplicações. Cada simulação retorna, além dos percursos para roteamento, o tempo total de execução da aplicação (em ciclos). O valor adotado nos resultados, contudo, é o atraso de cada aplicação em relação ao tempo de execução e comunicação sem atrasos. Assim, quando o roteamento obter um percurso mínimo e sem efeitos de contenção, o valor de atraso é zero. As Figuras 46, 47, 48, 49 e 50, por sua vez, ilustram comparativamente os atrasos para os quatro diferentes algoritmos de roteamento. Estes valores representam o valor médio (para dez diferentes mapeamentos) do atraso para cada grafo de tarefas dos conjuntos $Ex1$, $Ex2$, $Ex3$, $Ex4$ e $Ex5$. No Apêndice B, as Tabelas 19, 20, 21, 22 e 23 apresentam os valores numéricos obtidos nestas simulações.

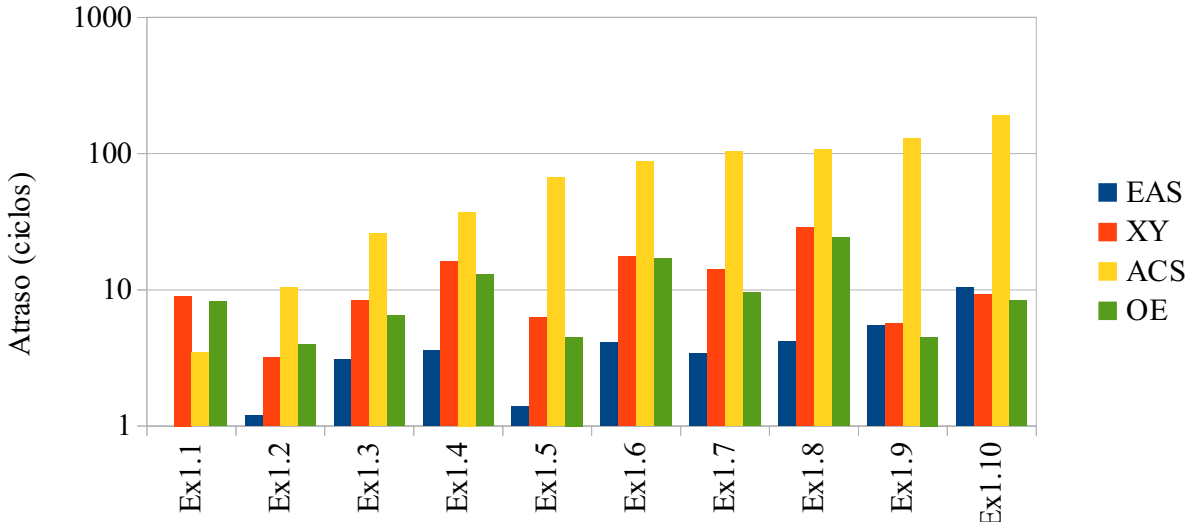


Figura 46: Resultados para o conjunto $Ex1$.

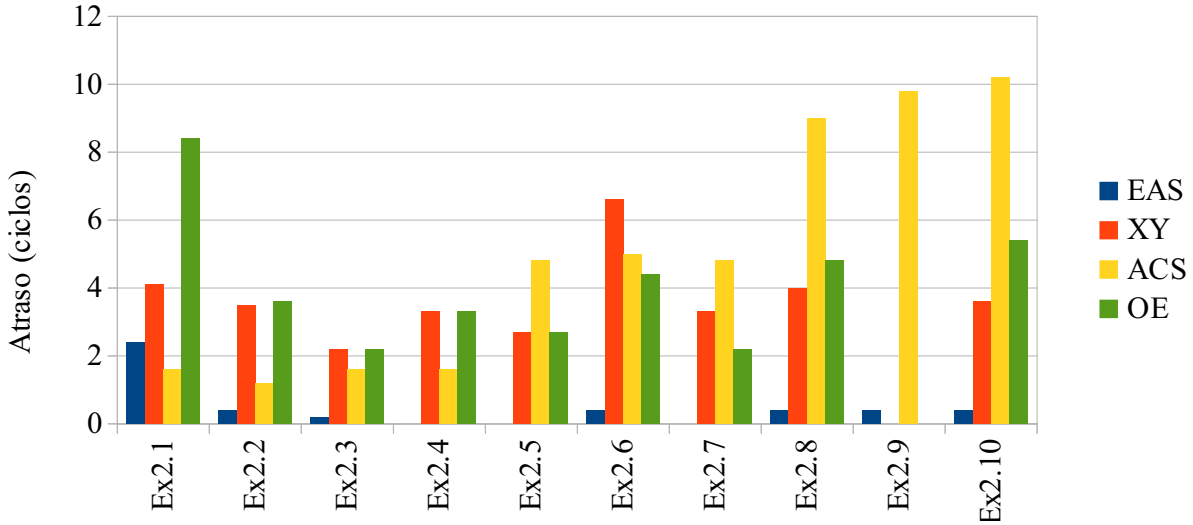


Figura 47: Resultados para o conjunto $Ex2$.

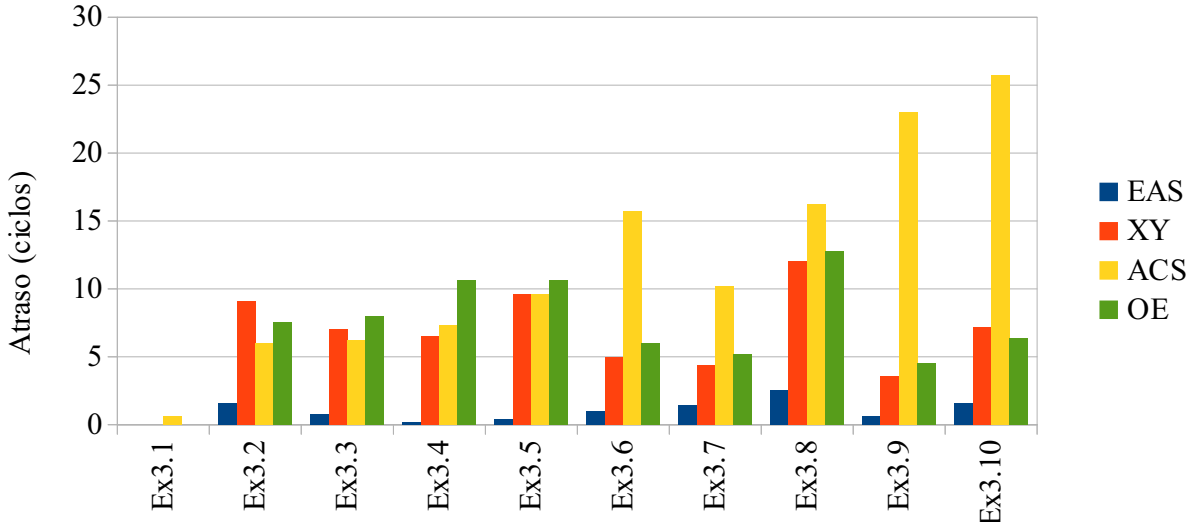


Figura 48: Resultados para o conjunto $Ex3$.

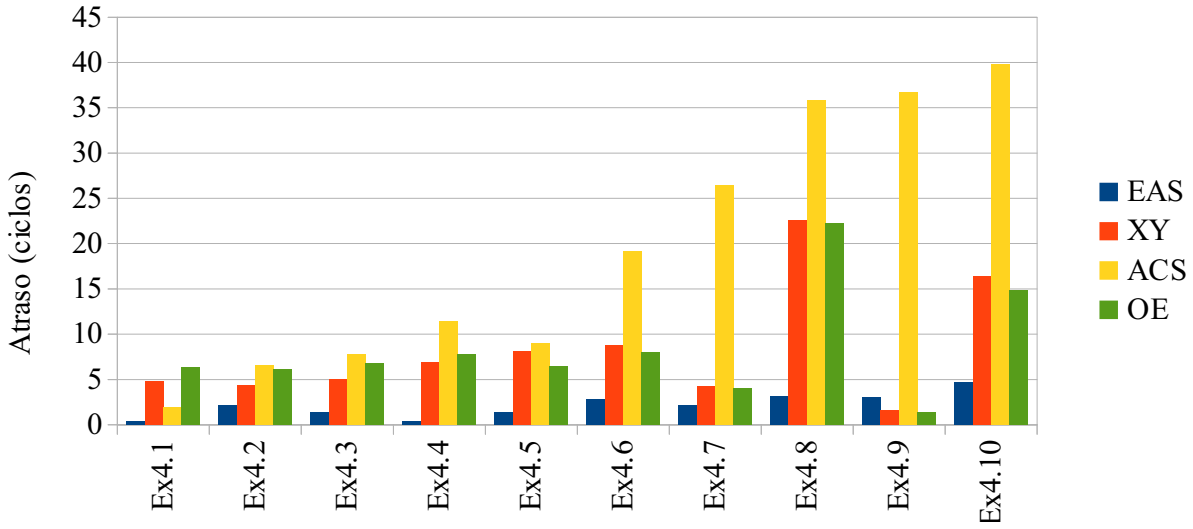


Figura 49: Resultados para o conjunto $Ex4$.

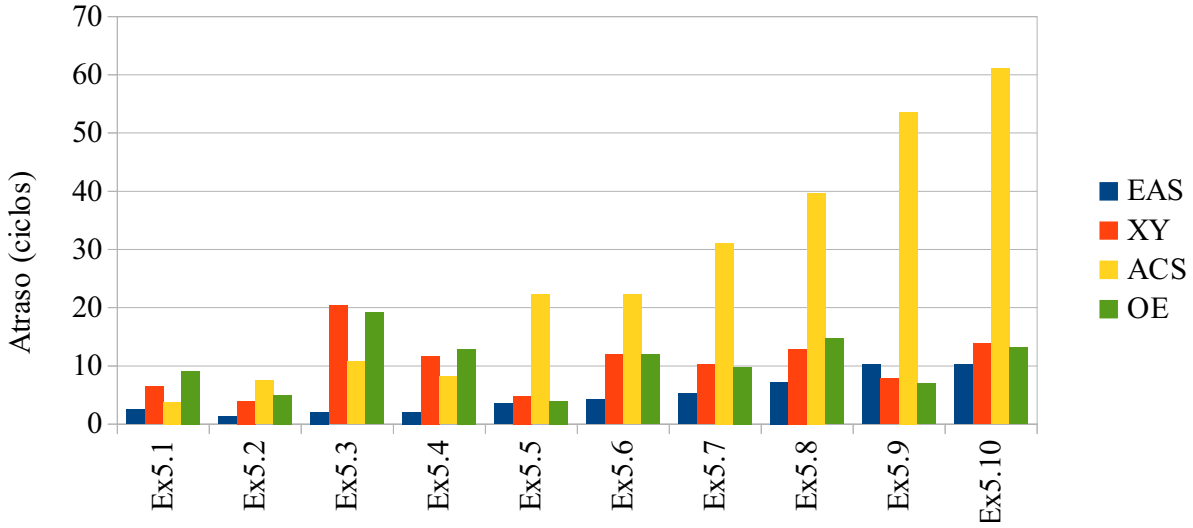


Figura 50: Resultados para o conjunto $Ex5$.

6.3.2 Análise dos resultados

Mais uma vez observou-se o roteamento baseado em EAS alcançando resultados melhores que os demais, salvo algumas exceções. Verifica-se que nas simulações utilizando o REAS, o atraso médio obtido varia extremamente pouco quando comparado aos demais métodos. De fato o REAS, aparentemente, consegue obter resultados satisfatórios independente tanto do mapeamento adotado quanto da complexidade da aplicação testada.

Um detalhe interessante quanto ao REAS está nos parâmetros α e β que tiveram seus valores invertidos. Ainda no desenvolvimento do REAS, observou-se que atribuindo o valor de $\alpha = 2$ e $\beta = 1$, o roteamento convergia de forma rápida para um resultado. Porém, o resultado era geralmente um mínimo local. Assim, estes valores foram descartados e adotou-se, para os testes do Capítulo 5, valores presentes na literatura (DORIGO; STÜTZLE, 2004). Já para os testes com aplicações, a escolha de parâmetros $\alpha = 2$ e $\beta = 1$ resultou em resultados melhores (menor atraso ou mesmo atraso em um menor número de iterações) quando comparado ao REAS com os parâmetros $\alpha = 1$ e $\beta = 2$. Desta forma optou-se, para todos os testes do REAS com aplicações, pelos valores de $\alpha = 2$ e $\beta = 1$.

Para os roteamentos baseados no algoritmo XY e no modelo *Odd-Even* observa-se uma grande variação nos atrasos médios obtidos para um mesmo conjunto de grafos. Apesar da variação, estes atrasos ainda são, contudo, superiores aos obtidos com o REAS. Este grande desvio nos valores de atraso pode sugerir que XY e OE são muito sensíveis ao mapeamento adotado, mais até do que à complexidade da aplicação.

Por fim, os piores resultados foram obtidos pelo roteamento baseado em ACS. Embora este método tenha alcançado bons valores de latência para os testes como padrões sintéticos de tráfego, para os testes com aplicações o RACS apresentou um atraso crescente conforme o aumento da complexidade (quantidade de tarefas ou níveis) dos grafos de tarefas. Este aumento quase linear do atraso evidencia que o RACS não realiza eficientemente a o processo de otimização para diferentes tamanhos de rede. Para todos os testes apresentados no Capítulo 5, o tamanho da rede foi considerado constante (uma malha 5×5). Já para os testes com aplicações, o tamanho da rede varia com a quantidade de tarefas, conforme definido pela Equação 35. Assim, diferente do que ocorre com o REAS, os parâmetros que resultam em bons resultados do RACS para uma rede com poucos nós podem não ser, necessariamente, os melhores para uma rede com um grande número de nós.

Possivelmente o RACS seja capaz de encontrar resultados melhores se forem utilizados mais ciclos iterativos. Contudo, a fim de manter uma similaridade, as simulações empregando

tanto o RACS quanto o REAS utilizaram como critério de parada o número máximo de iterações de 200 ciclos iterativos.

6.3.3 Significância estatística dos resultados

Um determinado conjunto de resultados possui significância estatística se for improvável que estes tenham ocorrido por acaso. Os resultados apresentados na Seção 6.3.1 mostram o REAS sendo capaz de obter resultados melhores que os demais algoritmos de roteamento empregados para comparação. A fim de assegurar a significância estatística destes resultados obtidos, é necessária a realização de um teste de *aderência* ou *confiança estatística* de dados. Dentre os métodos teóricos mais utilizados, pode ser citado o teste χ^2, ou chi-quadrado (HEDGES *et al.*, 1985). Este teste é capaz de identificar se a diferença entre dois grupos de dados é significativa ou se ocorreu por coincidência.

O teste χ^2 leva em conta a diferença entre os valores observados e esperados para cada um dos quesitos considerados. Os valores observados são aqueles obtidos experimentalmente, enquanto que os valores esperados se referem à distribuição hipotética baseada nas proporções globais dentro dos contextos comparados. Foram realizados testes $REAS \times XY$, $REAS \times OE$ e $REAS \times RACS$ separadamente para cada conjunto de simulações ($Ex1$ até $Ex5$) utilizando os valores obtidos de latência. O cálculo do valor esperado é realizado com a Equação 36. O valor de χ^2 é calculado com a Equação 37, onde A e T são os conjuntos de algoritmos e simulações utilizados em cada teste. O valor O_{at} é a latência observada na simulação t pelo algoritmo de roteamento a.

$$E_{at} = \frac{\sum_{x \in A} O_{xt} \cdot \sum_{y \in T} O_{ay}}{\sum_{(x,y) \in A \times T} O_{x,y}} \tag{36}$$

$$\chi^2 = \sum_{(a,t) \in A \times T} \frac{(O_{at} - E_{at})^2}{E_{at}} \tag{37}$$

O teste para descartar a *hipótese nula*, isto é, a de que os dados foram obtidos ao acaso, é realizado comparando o valor calculado de χ^2 com um determinado valor crítico tabelado. Essa tabela organiza valores de χ^2 em função dos *graus de liberdade* e do *nível de significância*. Em geral, é adotado o nível de significância de 95% como valor crítico. Desta forma, se o valor calculado for superior ao valor critico tabelado, então a hipótese nula pode ser descartada. O número de graus de liberdade é função da formatação dos resultados empregados no teste. Assumindo que os resultados estão em uma estrutura matricial de l linhas e c colunas, o grau de liberdade é $(l-1) \times (c-1)$.

Os valores de χ^2 calculados são vistos na Tabela 6. Devido a limitações existentes neste tipo de teste, duas alterações no conjunto de resultados foram realizadas. Como o teste não aceita valores inferiores a 1, os valores de atraso foram convertidos para 0,1 ciclo. A segunda alteração diz respeito às simulações em que um dos algoritmos retorna atraso de 0 ciclos. Nesta situação, os valores da simulação não são utilizados nos cálculo do teste. Por consequência, a quantidade de graus de liberdade é reduzida. Como pode ser observado na Tabela 6, a hipótese nula pode ser descartada em todos os casos. Com base nesta análise estatística, o REAS pode ser considerado significativamente melhor que os algoritmos XY, OE e RACS

Tabela 6: Níveis de significância obtidos com o teste χ^2.

		Graus de liberdade	χ^2	χ^2 crítico	Nível de significância
Ex1	$REAS \times XY$	8	155,5	26,12	99,90%
	$REAS \times OE$	8	124,48	26,12	99,90%
	$REAS \times RACS$	8	68,38	26,12	99,90%
Ex2	$REAS \times XY$	5	32,94	20,52	99,90%
	$REAS \times OE$	5	13,25	12,83	99,90%
	$REAS \times RACS$	6	120,12	22,46	97,50%
Ex3	$REAS \times XY$	8	24,43	24,35	99,80%
	$REAS \times OE$	8	33,47	26,12	99,90%
	$REAS \times RACS$	8	105,25	26,12	99,90%
Ex4	$REAS \times XY$	9	90,93	27,88	99,90%
	$REAS \times OE$	9	99,36	27,88	99,90%
	$REAS \times RACS$	9	37,57	27,88	99,90%
Ex5	$REAS \times XY$	9	144,53	27,88	99,90%
	$REAS \times OE$	9	166,6	27,88	99,90%
	$REAS \times RACS$	9	31,21	27,88	99,90%

6.4 Considerações Finais do Capítulo

Neste capítulo foram apresentado os resultados de testes de roteamento de pacotes gerados e transmitidos durante a execução de uma aplicação. O modelo de aplicação utilizado, baseado em um grafo de tarefas, é descrito. Também é apresentada a forma com que os dados inerentes a aplicação são usados a fim de simular a comunicação entre tarefas. Nos resultados obtidos, verificou-se a superioridade do REAS sobre os demais algoritmos de roteamento.

O capítulo seguinte finaliza este trabalho, abordando suas principais conclusões, bem como os pontos mais relevantes da presente dissertação. Também serão tratadas direções para possíveis trabalhos futuros.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

A OTIMIZAÇÃO por colônia de formigas foi empregada nesta dissertação na etapa de otimização do roteamento no projeto de redes embutidas. Os percursos por onde pacotes são transmitidos foram otimizados em função da latência, auxiliando assim na diminuição do tempo de comunicação de uma determinada aplicação.

7.1 Conclusões

Esta dissertação abordou a otimização de um roteamento estático para redes embutidas. Este estudo foi motivado pela carência de ferramentas de EDA voltadas para projetos baseados em plataforma NoC (OGRAS; HU; MARCULESCU, 2005) e pela perspectiva de que em breve esta será a arquitetura de comunicação predominante em sistemas embutidos. Em projetos baseados em NoC, diversas etapas de otimização podem ser realizadas a fim de melhorar um modelo proposto, resultando assim em uma possível implementação final. Entre estas etapas, é comum citar a alocação de tarefas a o mapeamento de IPs. Este trabalho considera uma etapa de otimização a mais: o roteamento. Uma NoC idealizada para uma tarefa específica pode ter toda a informação referente à comunicação entre blocos previamente definida durante o projeto da rede. Como após a implementação a NoC não mais será alterada, então as rotas de comunicação podem ser considerados caminhos fixos, no chamado roteamento estático. Este difere do roteamento dinâmico (em que o percurso de um pacote é definido durante sua transmissão), permitindo que as chaves possuam uma estrutura mais simples.

Algoritmos baseados em ACO foram selecionados para a tarefa de busca por percursos por sua bem conhecida capacidade de otimização de rotas em problemas modelados por grafos. Um recurso que necessita enviar dados para outro através de uma rede pode ser descrito por um par de vértices em um grafo. O ACO encontra o melhor percurso de comunicação em

uma rede com vários pacotes sendo transmitidos simultaneamente, de origens diferentes para destinos diferentes. Os melhores caminhos serão aqueles com menor extensão ou com menor atraso em cada chave, considerando que um pacote que atravessa uma chave pode acarretar em atrasos para os demais pacotes que passam pela mesma chave.

Dois algoritmos de otimização foram modelados no ambiente Matlab: o REAS, um roteamento baseado no *Elitist Ant System*, e o RACS, roteamento baseado no *Ant Colony Sistem*. Nestes dois algoritmos, grupos de agentes, chamados formigas artificiais, buscam por percursos em um grafo de caracterização da arquitetura, uma estrutura que representa a topologia malha 2D da rede. Cada conjunto de formigas é responsável pela otimização de um único percurso. Para mais de uma pacote transmitido, foram usadas múltiplas colônias, onde o caminho encontrado por uma colônia resulta em uma restrição para às demais.

Os dois algoritmos propostos foram testados com a rede sob um tráfego sintético, com pacotes gerados seguindo diversos tipos de padrões. Os resultados foram comparados com a rede, sob os mesmos padrões de tráfego, utilizando roteamentos baseados em XY e no modelo *Odd-Even* (OE). Em todos os testes, o REAS obteve uma latência média por pacote inferior aos demais métodos. Para estes mesmos testes, o RACS apresentou resultados ora semelhantes ao REAS, ora semelhantes ao XY e OE.

Uma segunda seção de testes teve por fim verificar o uso dos métodos de roteamento em aplicações, caracterizadas por grafos de tarefas. O conjunto de grafos usados nestes testes foram gerados aleatoriamente com o auxílio do TGFF, uma ferramenta popular para geração de grafos para estudo de sistemas multiprocessados. O roteamento estático é realizado após uma etapa de mapeamento, considerando que a posição de cada IP que realiza uma determinada tarefa já está definida. Neste sentido, o REAS obteve resultados melhores que os demais métodos em grande parte dos testes. De fato, os roteamentos XY e OE mostraram-se muito susceptíveis a mudanças no mapeamento, eventualmente conseguindo resultados bons em alguns poucos testes. Já o REAS manteve uma regularidade quanto aos atrasos obtidos independente do mapeamento e das aplicações, mostrando assim uma robustez quanto ao tamanho do problema. A grande surpresa ficou a cargo do RACS. Este obteve resultados bastante inferiores aos encontrados pelos demais em testes com grafos com um grande número de camadas ou com muitas tarefas.

7.2 Trabalhos Futuros

Nesta seção, são citadas algumas possíveis mudanças no algoritmo proposto, como o intuito de melhorar seu desempenho. Também são levantadas propostas para futuros trabalhos na área

de otimização em NoCs.

Sem dúvida, uma primeira sugestão a ser feita é o estudo mais aprofundado dos parâmetros do *Ant Colony System*. O interesse é fazer com que o RACS aplicado ao problema de roteamento em uma rede com aplicações (grafo mapeado) alcance resultados semelhantes aos obtidos com a rede sob ação de um tráfego sintético, tal como ocorrido com o REAS. Para os testes com padrões sintéticos de geração de pacotes apresentados nesta dissertação, foi considerada uma rede com topologia malha de tamanho fixo. Nesta situação, o roteamento baseado no ACS obteve resultados semelhantes ao REAS em grande parte das simulações. Contudo, pelo fato do REAS ter conseguido resultados satisfatórios nos testes com aplicações, não foi feita uma análise do efeito do tamanho da rede sobre os algoritmos propostos. Para um possível uso mais eficiente do RACS, é necessário verificar a relação entre a variação da complexidade do problema com os valores adotados para os parâmetros, como por exemplo β e q_0.

Uma segunda modificação diz respeito não ao roteamento em si, mas como este interage com a rede e a aplicação de interesse. Este trabalho utilizou-se de um simples mapeamento aleatório para a posicionar tarefas na rede. Esta é uma opção que intencionalmente não visa otimizar a etapa de mapeamento de IPs, sendo útil para verificar o comportamento do roteamento para uma mesma aplicação sujeita a diversos mapeamentos diferentes. Desta forma, duas opções poderiam ser sugeridas:

- Utilizar o resultado do mapeamento de outros trabalhos, ou utilizar uma ferramenta de terceiros para fazer uma etapa prévia de mapeamento, e só então realizar o roteamento.
- Combinar a etapa de roteamento ao roteamento, resultando em uma única etapa de otimização.

Por fim, um possível acréscimo aos trabalhos abordando roteamento está no uso de grafos de tarefas representativos de aplicações reais. Embora com o TGFF seja possível gerar grafos com diferentes níveis e complexidade (essencial para o estudo dos algoritmos), tratam-se de aplicações que não necessariamente resultariam em uma implementação final. Assim, para uma aplicação real, o algoritmo de otimização de rotas poderia ser integrado a uma ferramenta EDA para complementar as demais etapas de um projeto baseado em NoCs.

REFERÊNCIAS

BARAN, P. On distributed communications networks. *Communications Systems, IEEE Transactions on*, IEEE, v. 12, n. 1, p. 1–9, 1964.

BENINI, L.; MICHELI, G. D. Networks on chips: A new soc paradigm. *COMPUTER,*, Published by the IEEE Computer Society, p. 70–78, 2002.

BJERREGAARD, T.; MAHADEVAN, S. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)*, ACM, v. 38, n. 1, p. 1–es, 2006.

BOLDING, K.; CHEUNG, S.-C.; CHOI, S.-E.; EBELING, C.; HASSOUN, S.; NGO, T. A.; WILLE, R. The chaos router chip: design and implementation of an adaptive router. In: *Proceedings of the IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration*. Amsterdam, The Netherlands: North-Holland Publishing Co., 1994. p. 311–320. ISBN 0-444-89911-1.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. *Swarm intelligence: from natural to artificial systems*. USA: Oxford University Press, 1999.

BULLNHEIMER, B.; HARTL, R.; STRAUSS, C. A new rank based version of the ant system. a computational study. SFB Adaptive Information Systems and Modelling in Economics and Management Science, WU Vienna University of Economics and Business, 1997.

BULLNHEIMER, B.; HARTL, R.; STRAUSS, C. Applying the ant system to the vehicle routing problem. *Meta-heuristics: Advances and trends in local search paradigms for optimization*, Citeseer, p. 285–296, 1999.

CHEN, W.; SHEU, J. Performance analysis of multiple bus interconnection networks with hierarchical requesting model. *Computers, IEEE Transactions on*, IEEE, v. 40, n. 7, p. 834–842, 1991.

CHIU, G. The odd-even turn model for adaptive routing. *Parallel and Distributed Systems, IEEE Transactions on*, IEEE, v. 11, n. 7, p. 729–738, 2000.

COLORNI, A.; DORIGO, M.; MANIEZZO, V.; TRUBIAN, M. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, v. 34, n. 1, p. 39–53, 1994.

DALLY, W.; SEITZ, C. Deadlock-free message routing in multiprocessor interconnection networks. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 5, p. 547–553, 1987.

DANESHTALAB, M.; KUSHA, A. A.; SOBHANI, A.; NAVABI, Z.; MOTTAGHI, M. D.; FATEMI, O. Ant colony based routing architecture for minimizing hot spots in nocs. In: *Proceedings of the 19th annual symposium on Integrated circuits and systems design*. New York, NY, USA: ACM, 2006. (SBCCI '06), p. 56–61. ISBN 1-59593-479-0.

DENEUBOURG, J. L.; ARON, S.; GOSS, S.; PASTEELS, J. M. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, Springer Netherlands, v. 3, p. 159–168, 1990. ISSN 0892-7553. 10.1007/BF01417909.

DI CARO, G.; DORIGO, M. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, v. 9, p. 317–365, 1998.

DI CARO, G.; DORIGO, M. Mobile agents for adaptive routing. In: *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on System Science*. Los Alamitos, CA, USA: IEEE Computer Society, 1998. v. 7, p. 74 –83 vol.7.

DICK, R. P.; RHODES, D. L.; WOLF, W. TGFF: task graphs for free. In: *Proceedings of the 6th international workshop on Hardware/software codesign*. Washington, DC, USA: IEEE Computer Society, 1998. (CODES/CASHE '98), p. 97–101. ISBN 0-8186-8442-9.

DORIGO, M.; BIRATTARI, M.; STÜTZLE, T. Ant colony optimization - artificial ants as a computational intelligence technique. *Computational Intelligence Magazine, IEEE*, IEEE, v. 1, n. 4, p. 28–39, 2006.

DORIGO, M.; COLORNI, A.; MANIEZZO, V. *Positive feedback as a search strategy*. Milan, Italy, 1991.

DORIGO, M.; DI CARO, G. The ant colony optimization meta-heuristic. In: *New ideas in optimization*. Maidenhead, UK, England: McGraw-Hill Ltd., UK, 1999. p. 11–32. ISBN 0-07-709506-5.

DORIGO, M.; GAMBARDELLA, L. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 1, n. 1, p. 53–66, 1997.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, v. 26, n. 1, p. 29 –41, feb 1996. ISSN 1083-4419.

DORIGO, M.; STÜTZLE, T. *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004. ISBN 0262042193.

DUATO, J.; YALAMANCHILI, S.; LIONEL, N. *Interconnection Networks: An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 1558608524.

EDWARDS, S.; LAVAGNO, L.; LEE, E.; SANGIOVANNI-VINCENTELLI, A. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE*, IEEE, v. 85, n. 3, p. 366–390, 1997.

ENGELBRECHT, A. P. *Fundamentals of Computational Swarm Intelligence*. Chichester, UK.: John Wiley & Sons, 2006. ISBN 0470091916.

ESSER, R.; KNECHT, R. Intel paragon xp/s - architecture and software environment. Springer-Verlag, Berlin, p. 121–141, 1993.

GINDIN, R.; CIDON, I.; KEIDAR, I. Noc-based fpga: Architecture and routing. IEEE Computer Society, Washington, DC, USA, p. 253–264, 2007.

GLASS, C. J.; NI, L. M. The turn model for adaptive routing. In: *Proceedings of the 19th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 1992. (ISCA '92), p. 278–287. ISBN 0-89791-509-7.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, Elsevier, v. 13, n. 5, p. 533–549, 1986.

GÓMEZ, C.; GÓMEZ, M.; LÓPEZ, P.; DUATO, J. Reducing packet dropping in a bufferless noc. *Euro-Par 2008–Parallel Processing*, Springer, p. 899–909, 2008.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

GOSS, S.; ARON, S.; DENEUBOURG, J.; PASTEELS, J. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, Springer Berlin / Heidelberg, v. 76, p. 579–581, 1989. ISSN 0028-1042. 10.1007/BF00462870.

GRASSÉ, P.-P. Les insectes dans leur univers. Ed. du Palais de la decouverte, France, 1946.

GRASSÉ, P.-P. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, Birkhäuser Basel, v. 6, p. 41–80, 1959. ISSN 0020-1812. 10.1007/BF02223791.

HEDGES, L.; OLKIN, I.; STATISTIKER, M.; OLKIN, I.; OLKIN, I. *Statistical methods for meta-analysis*. [S.l.]: Academic Press Orlando, FL, 1985.

HOLLAND, J. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor, MI: U Michigan Press, 1975.

HU, J.; MARCULESCU, R. Dyad: smart routing for networks-on-chip. In: *Proceedings of the 41st annual Design Automation Conference*. New York, NY, USA: ACM, 2004. (DAC '04), p. 260–263. ISBN 1-58113-828-8.

INTEL, A. Touchstone delta system description. *Supercomputer Systems Division, Intel Corporation, Beaverton, OR*, v. 97006, 1991.

JALABERT, A.; MURALI, S.; BENINI, L.; MICHELI, G. D. × pipescompiler: A tool for instantiating application specific networks on chip. In: *Proceedings of the conference on Design, automation and test in Europe - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2004. (DATE '04), p. 20884–. ISBN 0-7695-2085-5.

KAHLE, J.; DAY, M.; HOFSTEE, H.; JOHNS, C.; MAEURER, T.; SHIPPY, D. Introduction to the cell multiprocessor. *IBM journal of Research and Development*, IBM, v. 49, n. 4.5, p. 589–604, 2005.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE. *Proceedings of the 1995 IEEE International Conference on Neural Networks*. Perth, WA , Australia, 1995. v. 4, p. 1942 –1948 vol.4.

KIRKPATRICK, S.; GELATT, C.; VECCHI, M. Optimization by simulated annealing. *science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671, 1983.

KONSTANTINIDOU, S.; SNYDER, L. Chaos router: architecture and performance. In: *Proceedings of the 18th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 1991. (ISCA '91), p. 212–221. ISBN 0-89791-394-9.

KUNG, N.; MORRIS, R. Credit-based flow control for atm networks. *Network, IEEE*, v. 9, n. 2, p. 40 –48, mar/apr 1995. ISSN 0890-8044.

MANIEZZO, V.; COLORNI, A. The ant system applied to the quadratic assignment problem. *Knowledge and Data Engineering, IEEE Transactions on*, v. 11, n. 5, p. 769 –778, sep/oct 1999. ISSN 1041-4347.

MATHWORKS. *MATLAB Version 7.7.0 (R2008b)*. Natick, Massachusetts: The MathWorks Inc., 2008.

MORAES, F.; CALAZANS, N.; MELLO, A.; MOLLER, L.; OST, L. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, Elsevier, v. 38, n. 1, p. 69–93, 2004.

MORENO, E. *Mapeamento e Adaptação de Rotas de Comunicação em Redes em Chip*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2010.

MOSCIBRODA, T.; MUTLU, O. A case for bufferless routing in on-chip networks. In: *Proceedings of the 36th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2009. (ISCA '09), p. 196–207. ISBN 978-1-60558-526-0.

MOURA, L.; PEREIRA, H. G. Aprendendo com a stigmergia, a auto-organização e as redes de cooperação. In: *III Conferência Internacional sobre Tecnologias de Informação e Comunicação*. Braga, Portugal: CHALENGES 2003, 2003.

NESSON, T.; JOHNSSON, L. Romm routing: A class of efficient minimal routing algorithms. *Parallel Computer Routing and Communication*, Springer, p. 185–199, 1994.

NI, L.; MCKINLEY, P. A survey of wormhole routing techniques in direct networks. *Computer*, IEEE, v. 26, n. 2, p. 62–76, 1993.

OGRAS, U. Y.; HU, J.; MARCULESCU, R. Key research problems in noc design: a holistic perspective. In: *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2005. (CODES+ISSS '05), p. 69–74. ISBN 1-59593-161-9.

PASRICHA, S.; DUTT, N. *On-Chip Communication Architectures: System on Chip Interconnect*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN 012373892X, 9780123738929.

PASTEELS, J.; DENEUBOURG, J.; GOSS, S. Self-organization mechanisms in ant societies (i): trail recruitment to newly discovered food sources. *Experientia*, Birkhauser, v. 54, n. 1, p. 155–175, 1987.

PEIS, B.; SKUTELLA, M.; WIESE, A. Packet routing on the grid. *LATIN 2010: Theoretical Informatics*, Springer, p. 120–130, 2010.

PFISTER, G.; NORTON, V. Hot spot contention and combining in multistage interconnection networks. *IEEE TRANS. COMP.*, v. 34, n. 10, p. 943–948, 1985.

PULLINI, A.; ANGIOLINI, F.; BERTOZZI, D.; BENINI, L. Fault tolerance overhead in network-on-chip flow control schemes. In: *Proceedings of the 18th annual symposium on Integrated circuits and system design*. New York, NY, USA: ACM, 2005. (SBCCI '05), p. 224–229. ISBN 1-59593-174-0.

RAHMAN, M.; HORIGUCHI, S. Dynamic communication performance of a hierarchical torus network under non-uniform traffic patterns. *IEICE TRANSACTIONS on Information and Systems*, v. 87, n. 7, p. 1887–1896, 2004.

SEITZ, C. L.; ATHAS, W. C.; FLAIG, C. M.; MARTIN, A. J.; SEIZOVIC, J.; STEELE, C. S.; SU, W.-K. The architecture and programming of the ametek series 2010 multicomputer. In: *Proceedings of the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1*. New York, NY, USA: ACM, 1988. (C3P), p. 33–37. ISBN 0-89791-278-0.

SEITZ, C. L.; BODEN, N. J.; SEIZOVIC, J.; SU, W.-K. The design of the caltech mosaic c multicomputer. In: *Proceedings of the 1993 symposium on Research on integrated systems*. Cambridge, MA, USA: MIT Press, 1993. p. 1–22. ISBN 0-262-02357-1.

SEO, D.; ALI, A.; LIM, W.-T.; RAFIQUE, N.; THOTTETHODI, M. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In: *Proceedings of the 32nd annual international symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2005. (ISCA '05), p. 432–443. ISBN 0-7695-2270-X.

SILVA, M. V. d. C. *Alocação e mapeamento de IPs para redes embutidas utilizando algoritmos evolucionários multiobjetivos*. Dissertação (Mestrado) — Universidade do Estado do Rio de Janeiro, 2009.

STÜTZLE, T. The max-min ant system and local search for combinatorial optimization problems: Towards adaptive tools for global optimization. In: *Proc. of the 2nd Metaheuristics Int. Conf.(MIC-97)*. Sophia Antipolis, France: INRIA, 1997.

SULLIVAN, H.; BASHKOW, T. R. A large scale, homogeneous, fully distributed parallel machine, i. In: *Proceedings of the 4th annual symposium on Computer architecture*. New York, NY, USA: ACM, 1977. (ISCA '77), p. 105–117.

TAMHANKAR, R.; MURALI, S.; STERGIOU, S.; PULLINI, A.; ANGIOLINI, F.; BENINI, L.; MICHELI, G. D. Timing-error-tolerant network-on-chip design methodology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, IEEE, v. 26, n. 7, p. 1297–1310, 2007.

TEDESCO, L.; MELLO, A.; GARIBOTTI, D.; CALAZANS, N.; MORAES, F. Traffic generation and performance evaluation for mesh-based nocs. In: *Proceedings of the 18th annual symposium on Integrated circuits and system design*. New York, NY, USA: ACM, 2005. (SBCCI '05), p. 184–189. ISBN 1-59593-174-0.

TOWLES, B.; DALLY, W. J. Worst-case traffic for oblivious routing functions. In: *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 2002. (SPAA '02), p. 1–8. ISBN 1-58113-529-7.

VAIDYA, A.; SIVASUBRAMANIAM, A.; DAS, C. Impact of virtual channels and adaptive routing on application performance. *Parallel and Distributed Systems, IEEE Transactions on*, IEEE, v. 12, n. 2, p. 223–237, 2001.

VALIANT, L. G.; BREBNER, G. J. Universal schemes for parallel communication. In: *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1981. (STOC '81), p. 263–277.

VARATKAR, G.; MARCULESCU, R. On-chip traffic modeling and synthesis for mpeg-2 video applications. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, IEEE, v. 12, n. 1, p. 108–119, 2004.

ZEFERINO, C. *Redes-em-Chip: arquiteturas e modelos para avaliação de área e desempenho. 2003. 242 f.* Tese (Doutorado) — Universidade Federal do Rio Grande do Sul. Programa de Pós Graduação em Computação, Porto Alegre, 2003.

ZEFERINO, C. A.; SUSIN, A. A. Socin: A parametric and scalable network-on-chip. In: *Proceedings of the 16th symposium on Integrated circuits and systems design*. Washington, DC, USA: IEEE Computer Society, 2003. (SBCCI '03), p. 169–. ISBN 0-7695-2009-X.

ZIMMERMANN, H. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, IEEE, v. 28, n. 4, p. 425–432, 1980.