UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALOCAÇÃO DE TAREFAS PARA APLICAÇÕES DE TEMPO REAL EM ARQUITETURAS *MULTI-CORE*

TRABALHO DE GRADUAÇÃO

Iaê Santos Bonilha

Santa Maria, RS, Brasil
2011

ALOCAÇÃO DE TAREFAS PARA APLICAÇÕES DE TEMPO REAL EM ARQUITETURAS *MULTI-CORE*

por

Iaê Santos Bonilha

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Osmar Marchi dos Santos

Trabalho de Graduação N. 331
Santa Maria, RS, Brasil
2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o
Trabalho de Graduação

ALOCAÇÃO DE TAREFAS PARA APLICAÇÕES DE TEMPO REAL EM ARQUITETURAS *MULTI-CORE*

elaborado por
Iaê Santos Bonilha

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Osmar Marchi dos Santos (UFSM)
(Presidente/Orientador)

Prof. Dr. Raul Ceretta Nunes (UFSM)

Prof. Dr. Renato Machado (UFSM)

Santa Maria, 20 de Dezembro de 2011.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

ALOCAÇÃO DE TAREFAS PARA APLICAÇÕES DE TEMPO REAL EM ARQUITETURAS *MULTI-CORE*

Autor: Iaê Santos Bonilha
Orientador: Osmar Marchi dos Santos
Local e data da defesa preliminar: Santa Maria, 20 de dezembro de 2011.

Com a popularização de arquiteturas *multi-core*, e a complexidade adicionada por essa arquitetura no desenvolvimento de sistemas de tempo real, a maioria das soluções desenvolvidas para o mapeamento de tarefas que são encontradas na literatura têm focado somente na distribuição do processamento das tarefas entre *cores* (núcleos de processamento). Em certas arquiteturas *multi-core*, como a arquitetura NoC (*Network-on-Chip*), a comunicação entre as tarefas, um fator de grande impacto no comportamento temporal de um sistema, que deveria ser levada em consideração durante o mapeamento de tarefas entre *cores*, porém não tem recebido a devida atenção na literatura atual. Com o objetivo de desenvolver mapeamentos de tarefas que foquem nestes dois fatores-chave (processamento e comunicação), neste trabalho utilizam-se algoritmos de busca de soluções direcionados para resolução do problema de mapeamento de tarefas. Para tanto, heurísticas foram definidas junto com indicativos importantes do comportamento temporal de um sistema advindos de um estudo de sistemas de tempo real. A arquitetura de sistema utilizada é a NoC. Através da utilização deste método para o escalonamento de um conjunto de tarefas projetado para impor diversas dificuldades à atividade de mapeamento, foram obtidos mapeamentos muito próximos do escalonável demonstrando a viabilidade desta técnica.

Palavras-chave: sistemas de tempo real, escalonamento de tarefas, sistemas multi-core, mapeamento de tarefas

Abstract

Graduation Paper

Science of Computation Course

Federal University of Santa Maria

TASK ALOCATION FOR REAL-TIME APPLICATIONS IN MULTI-CORE ARCHITECTURES

Author: Iaê Santos Bonilha

Advisor: Osmar Marchi dos Santos

With the popularization of the multi-core architectures and the complexity added by this archtecture on the development of real-time systems, most of the developed solutions for the task mapping issue in the current literacture have been focusing on the task processing distribution. In certain multi-core architectures, like NoCs (network on chip), the inter task communication, a factor that exerces great influence on the temporal behavior of a system, is not being take into account. In this paper solution search algorithms will be used with the objective of developing task mapping techniques that focus on both processing and communication. For that matter, a group of heuristics were developed, based on key factors on the temporal behavior of a real-time system. This paper will focus on the NoC architecture. Throught the utilization of this method on a taskset designed to impose dificulties on the task mapping process, mappings near to completly scheduable were obtained proving this altertative to be viable to adress the problem.

Key words: real-time system, task scheduling, multi-core systems, task mapping

AGRADECIMENTOS

Aos meus pais por todo o apoio e carinho no decorrer, não só desta empreitada, mas de toda minha vida, por serem modelos tão fortes de honestidade, empenho e altruísmo sem os quais eu não seria quem sou hoje. Finalmente, por nunca terem deixado de acreditar em mim, mesmo quando eu próprio duvidei de minhas capacidades.

Aos meus familiares pelo apoio e por sempre terem acreditado no meu potencial. Em especial a minha avó, Olga, por sempre me estimular a dar o melhor de mim em tudo que eu faço; a minha tia, Alice, e minha madrinha, Alexandra, pela fé inabalável que sempre tiveram em mim e por torcerem tanto pelo meu sucesso; as minhas tias, Moara e Nene, por sempre prezarem pelo meu bem-estar.

Ao meu orientador, Osmar, por ter me proporcionado um objetivo no qual me focar, por sempre estar aberto as minhas ideias, me dando a liberdade para sempre perseguir o curso de ação que eu considerava mais adequado e por ter me ajudado em muitos obstáculos que se apresentaram no decorrer desse ultimo semestre.

Aos professores com quem tive aula no decorrer desses anos por me mostrarem o universo de possibilidades dentro da área de Ciências da Computação. Em especial, ao professor Guilherme Dhein por ter me apresentado a área que se tornou minha grande paixão, Inteligência Artificial.

Aos meus amigos e colegas por sempre me darem uma perspectiva diferente quando eu me deparava com barreiras que pareciam intransponíveis.

Finalmente, dedico esse trabalho ao meu avô, Adaí Bonilha, que faleceu durante a sua produção. Esteja aonde estiver, que ele saiba o quanto foi amado e o quanto sentiremos sua falta.

“Experiência é o nome que damos aos nossos erros.”

-Oscar Wilde

Sumário

Índice de Figuras

1 Introdução

Sistemas em tempo real são sistemas que possuem restrições críticas de tempo para execução de suas tarefas, fazendo com que não apenas o resultado lógico seja importante mas também o tempo em que esse resultado é produzido (Stankovic,1988). O objetivo da computação em tempo real é criar formas de se proporcionar um comportamento temporal previsível em um sistema.

Em um sistema em tempo real, a aplicação é composta por um conjunto de tarefas. Uma tarefa é um conjunto de passos computacionais a serem executados pelo processador, ou seja, uma unidade de execução (*e.g.* um processo). Em um sistema em tempo real cada tarefa possui um prazo (*D – Deadline*) no qual ela deverá ser executada e um tempo de execução (*C – Computation Time*) para o pior caso de execução sem interferência de outras tarefas (*WCET - Worst Case Execution Time*) que deve ser conhecido de antemão. Para a obtenção do pior caso (*WCET*), podem ser utilizadas técnicas de medição ou modelagem, como especificado em (Wilhelm *et al*, 2008) .

As tarefas em um sistema em tempo real são classificadas de acordo com sua periodicidade como mencionado em (Davis e Burns, 2010):

- **Tarefas periódicas:** possuem um período (*T – period*), ou seja, um intervalo fixo de tempo em que executam sua computação;
- **Tarefas aperiódicas:** são tarefas não periódicas. Não possuem um comportamento temporal definido e podem executar a qualquer momento;
- **Tarefas esporádicas:** não seguem um período bem definido mas elas possuem a garantia que, a partir do inicio de sua execução, elas não serão reintroduzidas para escalonamento por pelo menos um tempo mínimo de intervalo (MIT – Minimum Inter-Arrival Time).

Cada tarefa também possui uma prioridade (*P - Priority*) que determina a precedência de sua execução em relação as outras tarefas que compõe o sistema.

Quando o conjunto de tarefas do sistema é executado e, mesmo com a ocorrência dos piores casos de execução de todas suas tarefas, nenhuma tarefa perde

seu prazo, dizemos que o sistema é escalonável. Para determinar a escalonabilidade de um sistema, frequentemente são utilizadas as chamadas análises de escalonamento que consistem em um modelo de sistema (conjunto de condições referentes ao sistema a serem respeitadas para validade da análise) e uma análise matemática que resultará em uma predição do tempo de término do conjunto de tarefas do sistema. Em certas análises de escalonamento (ver Seção 2.1), calcula-se a interferência específica que uma tarefa sofrerá das outras tarefas do conjunto e, portanto, possibilita a predição de possíveis perdas de prazos por parte de cada tarefa.

Para a determinação da interferência exercida no término da execução de uma tarefa pelas outras tarefas do sistema, é necessário conhecer a ordem na qual as tarefas serão executadas, pois cada tarefa influencia o tempo de término das tarefas que seguirão sua execução. Para agregar previsibilidade na ordem de execução das tarefas, são determinadas prioridades para cada uma das tarefas, representadas por um número inteiro que indicará a precedência da execução de uma tarefa em relação as demais tarefas do sistema. As prioridades são determinadas de acordo com modelos de prioridades, que consistem em um conjunto de regras para a atribuição de prioridades. Os modelos de prioridade se dividem em duas categorias principais:

- **Prioridade Fixa:** a princípio, as prioridades das tarefas não mudam em tempo de execução. Porém, dependendo da situação (*e.g.* possível compartilhamento de recursos), elas podem mudar dentro de um pequeno conjunto fixo de prioridades, que variam de acordo com o modelo de definição de prioridades adotado.
- **Prioridade Dinâmica:** as prioridades das tarefas variam em tempo de execução, normalmente, variando de acordo com o passar do tempo.

Os modelos de prioridade fixa agregam uma maior previsibilidade ao sistema, pois mesmo que haja casos em que as prioridades mudem, essa mudança é realizada seguindo regras estritas. Os modelos de prioridade dinâmica, tal como o EDF (*Earliest Deadline First*) proposto em (Liu e Layland, 1973), possuem uma certa imprevisibilidade que pode afetar as análises de escalonamento pois o estado de

escalonabilidade do sistema varia de acordo com o tempo. Porém, modelos de prioridade dinâmica agregam uma maior adaptabilidade do sistema pois conseguem um valor de utilização do sistema maior. Neste trabalho será utilizado um modelo de sistema com prioridades fixas pelo fato desta abordagem agregar uma maior previsibilidade ao sistema e ser o modelo padrão utilizado na indústria.

Independente do modelo utilizado em um sistema de tempo real, existem certas necessidades a serem atendidas para a sua utilização no escalonamento de conjuntos de tarefas. Primeiramente, o comportamento temporal das tarefas deve ser previsível e seu tempo de execução para o pior caso deve ser estudado e conhecido previamente. Além disso, fatores como distribuição de recursos mutuamente exclusivos, relação de precedência de tarefas, possíveis sobrecargas e periodicidade ou aperiodicidade das tarefas devem ser conhecidos e levados em consideração pois impactarão diretamente no comportamento temporal do sistema. Devendo-se ressaltar que, com a introdução de sistemas *multi-core,* o mapeamento de recursos se tornou um fator crítico para a escalonabilidade de um sistema pois a disposição das tarefas nos múltiplos *cores* afeta os conjuntos de interferência (uma tarefa sofrerá apenas interferência de tarefas que compartilham seu *core*) e rotas de comunicação, afetando diretamente o comportamento temporal do sistema.

1.1 Hipótese e Objetivos

Este trabalho se foca na premissa de que, a partir de um estudo de sistemas de tempo real e de modelos de análise de escalonamento, podem ser isoladas informações importantes para a escalonabilidade de um sistema e, a partir das mesmas, podem ser derivadas heurísticas para guiar algoritmos de busca através do problema de mapeamento de tarefas.

Os objetivos deste trabalho são:

(i) Desenvolver heurísticas para o mapeamento automático de tarefas utilizando algoritmos de busca.

(ii) O estudo de algoritmos de busca que possam ser utilizados para mapear tarefas em sistemas *multi-core* baseados em NoC, utilizando as heurísticas desenvolvidas.

Dados os objetivos estabelecidos, as principais etapas no desenvolvimento deste trabalho incluem:

1) Revisão sobre Sistemas de Tempo Real: Entender quais as principais características e como funcionam os sistemas de tempo real mono processados e multi-core;

2) Revisão sobre Algoritmos de Busca: algoritmos de busca tem como objetivo buscar um conjunto eficiente de soluções para problemas onde a complexidade envolvida na procura de todas as possibilidades introduz um custo alto e restritivo. Logo, heurísticas (fatores indicadores) podem ser utilizadas na obtenção de soluções satisfatórias em tempo hábil.

3) Especificação do Modelo do Sistema: Especificar o conjunto de tarefas e fluxos de comunicação que definem o sistema a ser analisado e quais suas relações;

4) Definição de Heurísticas e da Implementação dos Algoritmos de Busca para o Mapeamento: A especificação de fatores indicadores para o problema de mapeamento de tarefas e fluxos em NoCs possibilitando a criação de Heurísticas de Mapeamento as quais conduziram os algoritmos de busca na geração de soluções eficientes de mapeamento.

5) Implementação dos Algoritmos de Busca: definição dos algoritmos de busca utilizados e a especificação da modelagem de seus procedimentos com o objetivo de tornar viável a resolução de problemas de mapeamento de tarefas através dos mesmos.

6) Definição de casos de teste: definição dos conjuntos de tarefas e arquitetura utilizados para realização de testes envolvendo as heurísticas desenvolvidas.

7) Análise de Resultados: análise dos resultados obtidos nos testes individuais das heurísticas e nos testes envolvendo a combinação das mesmas.

1.2 Estrutura do Texto

Este trabalho é organizado da seguinte forma. No próximo capítulo é realizada uma revisão da literatura na área de sistemas de tempo real, focando na análise de escalonamento, sistemas multi-core e fluxos em NoC (*Network-on-Chip*), e na área de heurísticas e algoritmos de busca de solução. O Capítulo 3 mostra o modelo de sistema, com o objetivo de abordar tempo de execução e comunicação, no qual este trabalho irá se focar. O Capítulo 4 trata do mapeamento de tarefas, começando pela definição das heurísticas desenvolvidas. No Capítulo 5, tem-se as implementações de dois algoritmos de busca os quais utilizarão as heurísticas desenvolvidas até o momento. O Capítulo 6 trata da demonstração e interpretação dos resultados obtidos através dos testes realizados. Finalmente, o Capítulo 7 tratará de considerações finais a respeito do trabalho, resultados obtidos até o momento da escrita e trabalhos futuros.

2 Revisão de Literatura

Neste capítulo é realizada uma revisão da literatura utilizada para fundamentar este trabalho, começando por uma conceitualização no escalonamento de tarefas e nos desafios introduzidos na migração dos mesmos para sistemas *multi-core*, passando pela descrição dos dois modelos de análise de escalonamento usados para criar o modelo de sistema deste trabalho. A última seção descreve os dois algoritmos de busca utilizados neste trabalho.

2.1 Escalonamento em Sistemas Monoprocessados

Escalonamento é o processo de determinar qual será a próxima tarefa, no conjunto de tarefas do sistema, a receber a posse de um determinado recurso, onde o recurso pode se tratar desde tempo de processamento até a utilização de um periférico. Este processo será guiado pela prioridade de cada uma das tarefas que é determinada por uma política de definição de prioridade. Logo, a prioridade é representada por um número inteiro, um menor valor representará uma maior prioridade. O escalonador cederá o recurso para a tarefa de maior prioridade dentro da lista de tarefas prontas para execução.

Atualmente, a grande maioria das técnicas utilizadas no escalonamento de sistemas multiprocessados foram criadas para sistemas mono processados e, então, adaptadas para serem utilizadas por múltiplos processadores. Nesse trabalho, nos fixaremos na política para atribuição de prioridades conhecida como taxa monotônica, fixar-se-á um modelo de definição de prioridades apresentado juntamente com o primeiro modelo proposto para a análise de escalonabilidade para sistemas em tempo real (Liu e Layland, 1973) .

Para entendermos como é feito o teste da escalonabilidade de um sistema, primeiramente, é necessário conhecer o conceito de instante crítico, definido por Liu e Layland juntamente com o modelo de taxa monotônica. Dado um conjunto

de tarefas com prioridades arbitrárias, o instante crítico de uma tarefa ocorre quando a mesma chegar ao escalonador simultaneamente com todas as tarefas que possuem prioridade igual ou maior que a sua. O instante crítico é considerado a pior situação possível para o escalonamento e, se existe uma ordem para as tarefas serem executadas na qual nenhuma das tarefas perderá seu prazo, o sistema é considerado escalonável.

Ainda no artigo de 1973 (Liu e Layland, 1973), Liu e Layland definiram o primeiro modelo para a análise (ou teste) de escalonamento de um sistema, em que um conjunto de tarefas, chegando simultaneamente no escalonador, com períodos definidos e priorizadas de acordo com o tamanho de seu período (definição de prioridades por taxa monotônica) é praticável, sem que nenhuma delas perca seu prazo, desde que a utilização de CPU não passe do limite dado pela Equação (1).

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \qquad (1)$$

Em que: *Ci* é o tempo de computação da tarefa *i*; *Ti* é o período da tarefa *i*; e *n* representa o número total de tarefas no sistema.

Para que essa condição se confirmasse, as seguintes restrições deviam ser obedecidas:

- As tarefas devem ser independentes umas das outras.
- Todas as tarefas devem ser periódicas.
- Nenhuma tarefa pode sofrer bloqueio devido a eventos externos.
- Todas as tarefas devem chegar simultaneamente ao escalonador.
- Os prazos das tarefas devem ser iguais aos seus períodos.

O teste de escalonamento dado pela Equação (1) é dito suficiente (se o sistema passar no teste, o sistema é escalonável), porém, não necessário (se o sistema falhar no teste, ele pode ou não ser escalonável) .

Cerca de 10 anos após a proposição do primeiro modelo de teste de escalonamento, foi proposto um novo teste para verificar se um conjunto de tarefas era escalonável através de um modelo de atribuição de prioridades arbitrário, como

descrito em (Audsley *et al*, 1995), conhecido como *response time*. Esse teste produz como resultado o pior tempo possível de respostas (*r* - *response time*) para uma tarefa no seu instante crítico e é calculado, de forma iterativa, pela seguinte equação.

$$r_i = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \qquad (2)$$

em que B_i é o tempo máximo de bloqueio da tarefa *i*; e *hp(i)* representa o conjunto de tarefas de prioridade mais alta que a tarefa *i*. O algoritmo a seguir demonstra o processo iterativo de cálculo do *response time*.

```
Calcular_response_time(i)
{
    ri=1;
    novo_ri=0;
    tarefa-alvo=conjunto_tarefas.elemento(i)
    hp(i)=tarefa-alvo.informar_tarefas_com_prioridade_maior();
    Equanto (novo_ri != ri) OU (novo_ri < Di) Faça
    {
        ri = novo_ri;
        Ci = tarefa-alvo.informar_tempo_computação();
        Bi = tarefa-alvo.informar_tempo_bloqueio();
        somatório = 0;
        j = 0;
        Enquanto (j <  tamanho_hp(i)) Faça
        {
            tarefa = hp(i).elemento(j);
            Cj = tarefa.informar_tempo_computação();
            Tj= tarefa.informar_período();
            somatório = somatório +  teto((ri*Cj)/Tj);
            j = j + 1;
```

}

novo_ri = *Ci* + *B*i + *somatório*;

}

Como pode ser visto, ver o instante crítico de uma tarefa é influenciado por todas as tarefas de prioridade mais alta que a sua, se o tempo de resposta de pior caso de todas as tarefas do sistema for menor ou igual aos seus respectivos prazos, o conjunto de tarefas é considerado escalonável. A análise de escalonamento por *response time* assume que todas as tarefas de prioridade mais alta que a tarefa que está sendo analisada apresentam o pior tempo de execução possível e a tarefa-alvo permanecerá bloqueada pelo maior tempo possível. Este teste é dito suficiente e necessário, pois o sucesso nesse teste indica que nenhuma tarefa perderá seu prazo na execução do conjunto de tarefas e a falha nesse teste indicará o contrário considerando-se o pior caso possível.

2.2 Sistemas Multiprocessados

Com a introdução de múltiplos *cores* em um sistema, é injetada complexidade no seu sistema de escalonamento pois a disposição das tarefas nos múltiplos *cores* afeta diretamente o comportamento temporal do sistema. Isso se dá ao fato de uma tarefa sofrer interferência apenas de tarefas que compartilhem o mesmo *core* em um dado momento, tornando a disposição das tarefas nos *cores* um ponto chave para a escalonabilidade de um conjunto de tarefas.

2.2.1 Escalonamento em Sistemas Multiprocessados

Para o escalonamento de tarefas em sistemas multiprocessados normalmente são utilizadas técnicas de escalonamento para sistemas monoprocessados adaptadas para servir aos propósitos de sistemas multiprocessados. Tais adaptações são regidas por estratégias de escalonamento adotadas pelo sistema.

Segundo (Davis e Burns, 2010), o escalonamento em sistemas multiprocessados pode ser visto como a tentativa de resolver dois problemas essenciais:

- **Problema de Alocação:** em qual processador uma tarefa deverá executar.
- **Problema de Prioridade:** em qual ordem as atividades que compõem uma tarefa deverão ser executadas em relação as atividades que compõem as demais tarefas.

Dessa forma, podemos classificar as estratégias de escalonamento utilizadas para a adaptação de modelos de escalonamento de sistemas monoprocessados para sistemas multiprocessados de acordo com o problema ao qual elas tratam.

Estratégias que tratam o problema de alocação são referidas como estratégias baseadas em migração como classificado em (Carpenter *et al*, 2004), dentre as estratégias baseadas em migração pode-se mencionar:

- **Migração no nível de atividades:** cada tarefa é divida em um conjunto de atividades (*jobs)*, cada uma das atividades pertencente a uma tarefa pode ser executada em *cores* distintos.
- **Migração em nível de tarefa:** uma tarefa pode ser transferida para um *core* diferente do atual em um dado momento, porém todas atividades relativas a essa tarefa devem ser executadas no novo *core* ao qual a tarefa foi atribuída
- **Não-Migração:** as tarefas, após serem atribuídas a um *core*, deverão permanecer nesse *core* até o fim de sua execução.

Sistemas sem migração são referidos como sistemas de escalonamento particionado pois o conjunto de tarefas é previamente distribuído entre os *core*, ou seja, as tarefas competem apenas com tarefas da sua partição (*core*). Sistemas com migração em nível de tarefa ou de atividades são referidos como sistemas de escalonamento global pois as tarefas competem de forma global por todos os processadores.

Dentre as estratégias que atacam o problema de prioridade, referidas como estratégias baseadas em prioridade (Carpenter *et al*, 2004), de acordo com (Davis e Burns, 2010) destacam-se:

- **Prioridade fixa para tarefas:** a prioridade é fixa para a tarefa e todas as atividades que a compõe.
- **Prioridade fixa para atividades:** uma tarefa é composta por várias atividades, podendo elas possuir prioridades distintas porém fixas.
- **Prioridade dinâmica:** as prioridades das atividades que compõe uma tarefa podem variar de acordo com o tempo.

2.3 Comunicação entre tarefas

A comunicação entre tarefas além de adicionar complexidade ao modelo do sistema, pode impactar diretamente na eficiência do mesmo. Esse fenômeno se dá devido ao fato de que tarefas podem depender de dados provenientes de outras tarefas para iniciar sua execução e o atraso no envio (perda de prazo por parte da tarefa geradora dos dados) ou na propagação desses dados através de sua rota pode resultar em perdas de prazos por parte da tarefa receptora. Em sistemas multiprocessados o problema de roteamento dos dados é acrescentado, pois a distância entre o *core* da tarefa geradora e da tarefa receptora afeta o tempo de propagação final dos dados.

Neste trabalho será utilizada a arquitetura de NoCs. Que utiliza comunicação entre tarefas. Porém, para comunicação entre tarefas, irá ser usado o modelo de comunicação baseado em NoCs proposto em (Ni e McKinley, 1993) conhecido como *Wormhole Switching*. O modelo de *Wormhole Switching* consiste em uma arquitetura quatro-conectada, conforme a Ilustração 1, com um roteador no ponto intermediário de cada barramento. Esses roteadores possuem um *buffer* capaz de guardar *n* pacotes. O envio dos pacotes é orientado pela prioridade associada a cada fluxo e é feito de acordo com o algoritmo XY, que trata da propagação do fluxo primeiramente no eixo *x* até atingir a coordenada desejada do destino e, então, a propagação final até o destino pelo eixo *y*. O envio de cada pacote só é realizado caso haja espaço no buffer

do próximo roteador na trajetória do pacote. Para o manejamento de confronto de rotas, as rotas do sistema são representadas através de canais virtuais sendo, esses um conjunto ordenado de roteadores por onde um fluxo de dados deve passar até seu destino, uma tarefa estará alocada a um canal virtual caso sua rota (respeitando ordenação) esteja contida no mesmo. Um canal virtual poderá comportar múltiplos fluxos de comunicação.

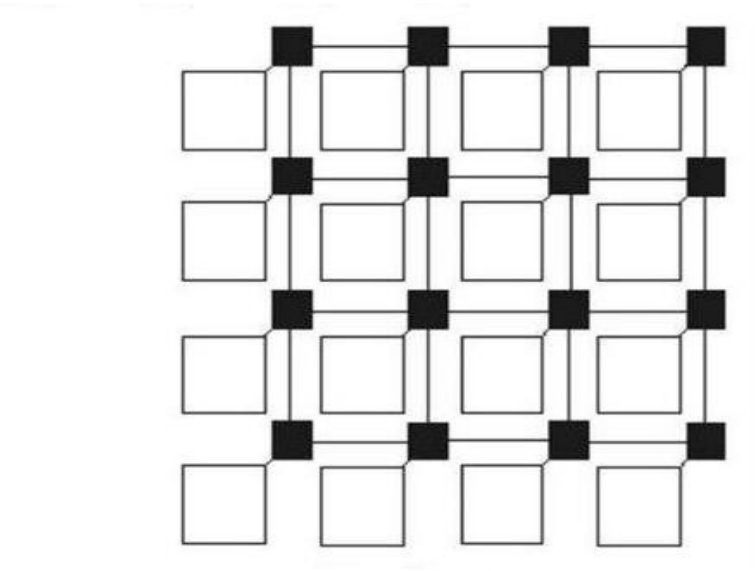

Figura 1: NoC quatro-conectada.

2.3.1 Análise de escalonamento orientado a comunicação

Em (Shi e Burns, 2010), foi proposto um modelo para o escalonamento de comunicações baseado no modelo de análise de escalonamento proposto em (Audsley *et al*, 1995) e utilizando o *Wormhole Switching* como modelo de comunicação. Este modelo utiliza equações similares ao *response time analysis* e utiliza um sistema de prioridades para determinar a precedência dos fluxos de comunicação em um determinado barramento. Neste modelo de análise de escalonamento, cada fluxo de comunicação, constituído por um conjunto de dados que devem ser transmitidos de um *core* a outro, é tratado da mesma forma que uma tarefa é tratada no modelo de *response time*.

2.3.2 Anatomia de um fluxo

De acordo com o modelo proposto em (Shi e Burns, 2010), cada fluxo pode ser representado por uma série de atributos, possuindo uma trajetória ou rota (*t* - *route*) que se trata de um conjunto ordenado de barramentos ao qual os dados tem que passar, um prazo (D_c - *deadline*) que representa o tempo limite para a chegada de seus dados no destino, uma latência (*C* - *latency*) que define o pior tempo que o conjunto de dados que compõe o fluxo levará para passar por um barramento, um período (T_c - period) que representa o mínimo intervalo entre duas liberações consecutivas do conjunto de pacotes pertencentes ao fluxo e uma prioridade (P_c - priority) que define a precedência dos pacotes pertencentes ao fluxo em relações a pacotes pertencentes a outros fluxos. A prioridade pode ser compartilhada por múltiplos fluxos pois ela não é atribuída diretamente ao fluxo, ela é atribuída a canais virtuais que consistem em uma rota pré-determinada. Todo fluxo ao qual sua rota estiver contida no canal virtual, fará parte do mesmo e, consequentemente, receberá a prioridade designada para este canal virtual. Finalmente, os fluxos de dados também possuem um atraso de liberação (J_r – *release jitter*) que indica o possível atraso no inicio do envio de dados de um fluxo após sua liberação e o atraso devido a interferência (J_i - *interference jitter*) que indica o desvio máximo entre o envio de conjuntos de pacotes consecutivos pertencentes ao fluxo em relação ao período do mesmo.

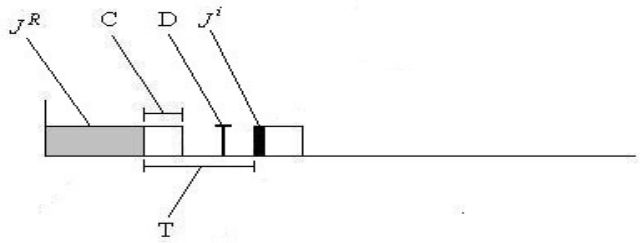

Figura 2: Anatomia de um fluxo de comunicação.

2.3.3 Análise de escalonabilidade de fluxos

O modelo de análise de Shi e Burns (Shi e Burns, 2010) utiliza a noção de janela de prioridade (*priority window*) para definir a escalonabilidade dos fluxos. O conceito de janela de prioridade baseia-se no conceito de instante crítico, onde a janela de prioridade se inicia com a liberação de todos os fluxos de comunicação de prioridade mais alta que o fluxo em análise e termina quando a atividade de todos os barramentos que compõe a rota do fluxo cessam. O fim da janela de prioridade é determinado pelo cessamento de todas atividades de barramentos pertencentes a rota, uma vez que na determinação do pior tempo de chegada da comunicação, assume-se que os pacotes pertencentes ao canal virtual serão sempre os últimos a atravessarem cada barramento. A janela de prioridade de uma tarefa é calculada, de forma iterativa, através da formula:

$$W(i) = \sum_{\forall \tau_n \in S(i)} \left\lceil \frac{W(i) + J_n^R}{T_n} \right\rceil C_n + \sum_{\forall \tau_j \in hp(i)} \left\lceil \frac{W(i) + J_j^R + R_j - C_j}{T_j} \right\rceil C_j \qquad (3)$$

em que: *S(i)* é o conjunto de fluxos que compartilham o mesmo canal virtual, ou seja,

que possuem a mesma prioridade; *hp(i)* representa o conjunto de fluxos de mais alta prioridade que o fluxo em questão; C_n e C_i representam a latência do fluxo de dados *n* e *i* respectivamente; J^R indica o atraso de liberação do fluxo; *T* indica o período máximo entre a liberação de dois pacotes distintos pertencentes ao fluxo; e *R* representa o tempo de pior caso de latência de um fluxo. O algoritmo para o cálculo do valor do *priority window* é similar ao algoritmo apresentado na Seção 2.1. As diferenças são a inclusão de mais um somatório para todas os fluxos de mesma prioridade (pertencentes ao mesmo canal virtual) e a desconsideração do tempo de bloqueio presente na Equação (2).

Tendo sido calculada a janela de prioridade, é feita uma análise comparando o valor da janela de prioridade com o período do fluxo. Essa comparação pode resultar em três diferentes situações:

- $W(i) \leq min(T_b - J^R_n)$, indicando que a *priority window* do fluxo acaba antes de liberações repetidas de fluxos em qualquer um dos canais virtuais envolvidos na análise.

- $min(T_b - J^R_n) < W(i) \leq T_i - J^R_i$, representando o fato de que liberações repetidas de fluxos pertencentes a outros canais virtuais ocorreram. Porém, nenhuma liberação repetida de fluxos pertencentes ao canal ao qual está se analisando no momento ocorre.

- $W(i) > T_i - J^R_i$, situação na qual ocorreram liberações repetidas em outros canais e no canal que está sendo analisado.

Terminada a análise, será calculado o pior tempo de recepção do fluxo de dados. Na primeira situação, que é pertinente a este trabalho, o pior tempo de recepção do fluxo será dado pela equação:

$$R_i = W(i) + J_i^R \tag{4}$$

Na segunda e terceira situações são necessários ajustes na equação, porém como não são utilizadas neste trabalho, não serão explicitadas.

2.4 Mapeamento de tarefas

Com o isolamento da interferência em cada *core* e a comunicação entre tarefas sendo realizada entre *cores* distintos, a disposição das tarefas nos *cores* (ou mapeamento de tarefas) se torna um fator chave para se alcançar a escalonabilidade de um conjunto de tarefas. Em sistemas particionados, existe um conjunto de trabalhos que focam na alocação de tarefas com o intuito de garantir escalonabilidade do sistema focada no processamento. Na sua maioria estes trabalhos são derivados do trabalho realizado em (Burchard, 1995). Porém, as relações entre tarefas (comunicação e dependência) tornam a alocação de tarefas um problema mais complexo que não é tratado pela literatura atual.

Para realização do mapeamento automático de tarefas uma série de técnicas podem ser utilizadas. Algoritmos de busca da área de inteligência artificial são uma das possibilidades mais promissoras para tal tarefa, visto que o problema de alocação é de complexidade NP-completo (Ullman, 1975).

Os algoritmos de busca exploram o conjunto de todas as soluções possíveis para um problema (espaço de busca) sendo guiados por informações pertinentes ao mesmo que determinam o quão boa ou o quão próxima de uma solução boa, a solução atual se encontra. Tais informações são conhecidas como heurísticas.

2.4.1 Força Bruta

O método de força bruta é a algoritmo mais simples para busca de soluções, ele explora todas as soluções possíveis para o problema, uma por uma, e avalia cada solução dado um determinado critério. Apesar da garantia de se achar a melhor solução

possível, esse algoritmo não utiliza-se de heurísticas para se guiar pelo espaço de busca o que o torna extremamente lento e ineficiente.

Suponhamos que está sendo tratado um problema simples, a ordenação de *n* números. Em tal problema nosso espaço de busca consistirá de todas as disposições possíveis para *n* números, ou seja, *n!* combinações diferentes as quais, no pior caso, serão exploradas pelo algoritmo de força bruta. Esse fato torna o algoritmo de força bruta inviável para mapeamento de grandes conjuntos de tarefa pois sua complexidade cresce fatorialmente, sendo classificado como um problema NP-completo (Ullman, 1975).

2.4.2 Subida de Colina

O algoritmo de subida de colina (*hill climbing*) é um algoritmo de busca que se baseia no conceito de vizinhança para se mover pelo espaço de busca, dessa forma utilizando o conhecimento local (vizinhança) para encontrar máximos ou mínimos locais (Nilsson,1982). A vizinhança de uma solução é definida por um movimento, que consiste em uma alteração padronizada que ao ser infligida na solução, resulta em um vizinho da mesma. O movimento é definido pelo programador e deve ser ajustado conforme o problema para obter uma melhor eficiência do algoritmo.

Após definido o movimento, o método explorará a vizinhança da solução corrente e avaliará seus vizinhos. Ao ser achado um vizinho da solução com uma pontuação melhor (de acordo com critérios de avaliação estabelecidos de acordo com o problema), esse vizinho substituirá a solução corrente e, então, sua vizinhança começará a ser explorada. O algoritmo continuará a explorar até que ele deixe de achar melhorias após um número a ser definido de tentativas.

Como o *hill climbing* se move através do espaço do movimento definido, não se pode garantir que ele achará a solução ótima pois ele não aceita pioras e, ao se deslocar pelo espaço de busca através do movimento, ele ficará preso em mínimos locais (depressões no gráfico de avaliação do espaço de busca). Para aumentar sua eficiência, é utilizado o método de reinícios aleatórios onde, após achar um mínimo, o

algoritmo guardará a solução e criará uma nova solução aleatória para ser explorada, sempre guardando a melhor solução geral.

2.4.3 Algoritmo Genético

Segundo (Mitchell,1998), a ideia por trás do algoritmo genético é evoluir uma população de soluções para um problema usando operadores inspirados pela variedade genética e seleção natural. Algoritmo genético se utiliza de uma analogia a biologia evolutiva para deslocar-se no espaço de busca, partindo do princípio que cada solução do espaço de busca possui trechos do ótimo e, o quão melhor for a solução, maiores serão esses trechos. Considerando cada solução como um indivíduo, possuindo um cromossomo (disposição da solução na forma vetorial) e um fator de avaliação que representa o quão boa a solução é, conhecido como *fitness*.

O algoritmo inicia com a criação de uma população aleatória, um conjunto de n indivíduos representando n soluções distintas. Cada indivíduo dessa população passará por uma avaliação para determinar seu *fitness*. Após a determinação do *fitness* dos indivíduos, começa o processo de reprodução da população orientado ao *fitness*, quanto melhor o *fitness* de um indivíduo, maior será a chance de ele se reproduzir.

A reprodução consiste na definição de k pontos de corte, esses pontos determinam os pontos aonde o cromossomo será separado resultando em $k+1$ trechos, esse processo é realizado para os dois indivíduos envolvidos na reprodução. A seguir, esses trechos são inseridos em um novo indivíduo alternando entre os dois indivíduos geradores, seguindo a ordem do trecho 0 para o trecho $k+1$. Após o término, o novo indivíduo gerado será inserido na nova população.

O processo de reprodução seguirá até que uma nova população esteja completa com o mesmo número de indivíduos que a população originadora. Essa nova população será avaliada, e um novo processo de reprodução se iniciará. Esse ciclo continuará até que um critério determinado pelo programador seja atingido.

Para melhorar a eficiência do algoritmo genético são utilizadas algumas noções adicionais, tais como:

- **Elitismo:** a noção de elitismo trata de preservar as melhores solução passando-as para a próximo população inalteradas. Muito embora o objetivo dessa prática seja manter bons genes na população, esta prática causa efeitos colaterais indesejados. O fato de manter indivíduos idênticos a população anterior com alto *fitness* provoca com que os indivíduos das próximas populações começam a se assemelhar cada vez mais com os indivíduos passados por elitismo, até que a variedade genética (a pedra fundamental da melhoria no algoritmo) acabe completamente. Para restringir esses efeitos, normalmente passa-se apenas um indivíduo através de elitismo para a próxima população ou não utiliza-se o elitismo.

- **Mutação:** consiste em causar variações aleatórias nos cromossomos de certos indivíduos da população de acordo com uma certa chance (medida através de uma porcentagem). A mutação tem como objetivo injetar variedade genética na população, retardando o processo de convergência para uma população uniforme. A chance de mutação deve ser calibrada com cuidado. Uma possibilidade muito alta de mutação pode prejudicar a população, resultando na ineficácia do algoritmo na busca de boas soluções. Uma baixa possibilidade de mutação pode causar uma convergência muito rápida para uniformidade genética da população.

3 Modelo do Sistema

Neste trabalho será utilizado o modelo de escalonamento por prioridade fixa (*fixed priority scheduling*) para sistemas *multi-core*, a estratégia de escalonamento utilizada será o escalonamento particionado, ou seja, sem migração de tarefas.

As tarefas serão representadas pela tupla $t_i=(C_i,T_i,D_i,c_i,P_i)$, onde, T_i representa o período da tarefa i, C_i é o tempo computacional da tarefa i, D_i indica o prazo da tarefa i, P_i é a prioridade da tarefa i e c_i constitui as atividades de comunicação geradas pela tarefa i. Após o termino da execução de uma tarefa i, ocorro o início das atividades de comunicação descritas em c_i.

O modelo de comunicação utilizado nesse trabalho é o modelo de *wormhole switching* mencionado na Seção 2.3, com a exceção de que, ao invés de vários fluxos de comunicação (comunicação realizada por uma única tarefa) compartilharem um mesmo canal virtual, cada fluxo de comunicação irá dispor de seu próprio canal virtual. Esta simplificação no modelo original foi realizada pois a criação de canais virtuais (no modelo original) é feita a partir de um estudo de todos as rotas de comunicação do sistema e, como este projeto lida com a produção de vários mapeamentos com o objetivo de obter uma melhora progressiva, as rotas do sistema mudam constantemente tornando tal estudo proibitivo.

As atividades de comunicação, ou fluxo de comunicação, c_i serão compostas por um conjunto de propriedades representados pela tupla $c_i=(L_i,Tc_i,Dc_i,t_i,Pc_i,Jr_i,J_{ii})$, onde, L_i representa a latência da comunicação i, Tc_i representa o mínimo período entre duas liberações consecutivas de pacotes pertencentes ao fluxo i, Dc_i é o prazo para a comunicação i, t_i é a rota do fluxo de comunicação e Pc_i demonstra a prioridade do fluxo que, dadas a alteração realizada no modelo de comunicação, é determinado pela prioridade da tarefa geradora do fluxo ($P_i=Pc_i$) . Nesse modelo, Jr_i demonstra o atraso de liberação para o fluxo de comunicação e J_{ii} representa o atraso de interferência.

3.1 Calculando o pior tempo de término das tarefas e da comunicação

A análise de escalonamento será realizada em dois passos distintos. Primeiramente, será realizada uma análise de *response time*, calculando o pior tempo de execução de uma tarefa *i* (*ri*) através da Equação (2), em que *ri* = Resultado (Equação (2)).

Após realizada a análise de *response time*, o próximo passo é realizar uma análise de *priority window* para os fluxos de comunicação que seguirão o término das tarefas. A conexão entre estas duas análises de escalonamento será realizada através do atraso de liberação do fluxo (*Jri*) ao qual será atribuído o valor resultante da análise de *response time* (*ri*) da tarefa que o produziu, sendo que o tempo de liberação de ambos o fluxo e a tarefa tem o mesmo valor. Ao calcular o pior tempo de término da comunicação, o valor resultante da *priority window* será adicionado ao atraso de liberação do fluxo. Ou seja, nesse caso é o pior tempo de execução da tarefa que gerou o fluxo, resultando no pior tempo para ambos comunicação e execução. Esse valor poderá futuramente ser utilizado para a implementação da noção completa de dependência entre tarefas. Ao término da análise de *priority window* será testada a escalonabilidade do fluxo. Para calcular o valor da *priority window*, será utilizada a Equação (3) com os termos ajustados para o modelo de sistema descrito:

$$J_{ri} = r_i$$

$$W(i) = \frac{W(i) + J_{ri}}{T_{ci}} L_i + \sum \frac{W(i) + J_{rj} + R_j - L_j}{T_{cj}} L_j \qquad (5)$$

Ressaltando que a interação entre tarefas de mesmo canal virtual foi removida da equação original do *priority window* devido a modificação realizada no modelo de comunicação pois haverá apenas uma tarefa por canal virtual.

Tendo sido calculado o valor da *priority window*, o pior tempo de comunicação e execução (*Ri*), o qual será usado para análise de escalonabilidade do fluxo, é dado por:

$$R_i = W(i) + J_{ri} \qquad (6)$$

4 Mapeamento de tarefas

Para se direcionar os algoritmos de busca são necessárias informações pertinentes ao problema (ou heurísticas), podendo dar um indicativo de progresso e um patamar de avaliação para o mesmo. No problema de mapeamento de tarefas em sistemas *multi-core* existem uma série de informações com a possibilidade de uso como heurísticas.

4.1 Fatores chave para o mapeamento

Após um estudo dos modelos de escalonamento utilizados nesse trabalho e um estudo comportamental de sistemas de tempo real realizado através de conjuntos de tarefa sintéticos executando em um ambiente simulado, foram evidenciados uma série de fatores, a serem discutidos nas seções a seguir, que se ressaltaram como indicadores promissores de progresso na exploração de soluções de mapeamento.

4.1.1 Balanceamento da utilização dos *cores*

Liu e Layland provaram, em 1973, que o modelo de taxa monotônica garante o escalonamento para qualquer conjunto de tarefas até um limite de utilização do processador dado pela Equação (1). Sendo assim, qualquer balanceamento de utilização de *cores* que respeite esse limite para cada um dos *cores* do sistema, caso o conjunto de tarefas permita tal evento, resultará em um mapeamento escalonável. Mesmo no caso que o conjunto de tarefas não permita atingir o limite de utilização em todos os *cores*, a distribuição de carga evitará sobrecargas resultando em uma menor possibilidade de perdas de prazos.

4.1.2 Número de perdas de prazos

Através de uma análise de *response time* e *priority window*, pode-se obter o possível número de perdas de prazos de tarefas e fluxos de uma determinada solução, gerando um grande indicador do estado instantâneo da solução, pois a redução das perdas de prazos (com o objetivo de alcançar um mapeamento completamente escalonável) é o maior foco de um algoritmo de mapeamento. Apesar de ser um grande indicativo da qualidade de uma solução, este fator não fornece um critério específico de progresso da busca pois apenas mede alterações que resultam em menos erros e não o progresso da busca entre tais estados.

4.1.3 Estresse gerado nos barramentos

O estresse infligido nos barramentos pela comunicação entre tarefas, representado pela suas utilizações pode representar um indicativo de progresso quanto a redução de erros temporais de fluxo. A utilização de barramentos comporta-se de forma diferente da utilização de *cores*. A soma da utilização de todos os *cores* é regida pelo conjunto de tarefas e permanecerá a mesma independente da disposição das tarefas. Já a soma da utilização de todos os barramentos pode variar drasticamente de acordo com o mapeamento das tarefas, quanto maior a distância entre a tarefa originadora e a tarefa receptora maior será a quantidade de barramentos ocupados com a utilização gerada pelo fluxo (constante em todos os barramentos). Além de causar erros temporais de fluxos que podem causar erros temporais de tarefas, no caso de dependência, a alta utilização de um barramento também causa um maior consumo de energia e a redução da vida útil do mesmo. Este fator possui grande utilidade não só para prevenção de erros temporais de fluxo mas para outros quesitos. Porém, ele desconsidera a ocupação dos *cores* e por isso deve ser utilizado em conjunto com algum fator com esse foco.

4.1.4 Harmonização dos períodos de tarefas

Em 1995, Burchard provou que pode ser obtido um limite de utilização (*utilization bound*) acima do previsto por Liu e Layland se o período das tarefas do processador forem harmônicos (o período de cada uma delas é um múltiplo direto das outras) ou estiverem próximos disso. Portanto, a harmonização dos períodos das tarefas em cada um dos *cores* pode resultar em uma menor ocorrência de erros temporais de tarefas. Apesar de ser um bom indicativo para prevenção de perdas de prazos de tarefas, esse fator não pode ser levado em consideração sozinho pois ele ignora as comunicações geradas pelas tarefas capazes de gerar erros temporais de fluxos.

4.1.5 Folga na execução de tarefas de baixa prioridade

A folga produzida pela execução da tarefa mais baixa prioridade de um core, chamada de gap neste trabalho, pode ser um bom indicativo da disponibilidade de espaço para execução de outras tarefas ou a falta do mesmo. Esse fator pode ser um bom indicativo para algoritmos de busca mais específicos orientados a migração de tarefas.

4.2 Heurísticas utilizadas

Para utilizar os fatores-chave mencionados na seção anterior para orientar os algoritmos de busca, necessita-se criar uma forma de transformá-los em valores numéricos passíveis de avaliação. Para tal, serão utilizados, além dos valores resultantes das análises, o desvio padrão. Muito embora heurísticas sejam normalmente consideradas fatores não-determinísticos, neste trabalho consideramos heurísticas como qualquer fator importante para a avaliação de uma determinada disposição de tarefas entre os múltiplos *cores* do sistema.

4.2.1 Desvio padrão de utilização de processadores

Para medir o balanceamento na utilização dos *cores*, a medida estatística de desvio padrão será utilizada. Essa medida garante que 96,5% dos valores do conjunto, ou seja, os valores de utilização de todos os *cores*, estejam dentro um intervalo de duas vezes o desvio padrão para menos e para mais em relação a média. Sendo assim, quanto menor for o valor do desvio padrão de utilização dos *cores* (Du), maior será o balanceamento de utilização dos *cores* do sistemas. O desvio padrão da utilização dos *cores* é calculado através da formula:

$$Du = \frac{\sum (U - Au)^2}{n} \tag{7}$$

em que: U representa a utilização de um core, n o número de *cores* do sistema e Au representa a média de utilização entre todos os *cores*.

4.2.2 Número de Erros Temporais

O número de erros temporais do sistema é obtido através das duas análises de escalonabilidade do sistema, a análise de *response time* prove o número de erros de tarefas e a análise de *priority window* determina o número de erros de fluxos. Somando o resultado de ambas as análises, o número total de erros temporais do sistema será obtido.

4.2.3 Estresse de rede

O comportamento da utilização de barramentos difere do comportamento da utilização de processadores pois, enquanto a soma da utilização em todos os *cores* permanece a mesma para um mesmo conjunto de tarefas, a soma da utilização de todos

os barramentos varia de acordo com o mapeamento das tarefas. Tendo em vista essa peculiaridade, a utilização exclusiva do desvio padrão da utilização de rede (Dn) não será efetiva para medir o estresse imposto na rede e a utilização máxima de barramento ($MAXr$) deve ser incluída na análise. Dessa forma, o estresse de rede (S) é calculado através da fórmula:

$$S = MAX\ r + MAX\ r * D\ n \tag{8}$$

O foco na redução dessa medida implica na redução da utilização máxima de barramento junto com o melhoramento do balanceamento da utilização de todos os barramentos.

4.2.4 Fator de multiplicidade de tarefas

O fator de multiplicidade entre duas tarefas é calculado através da divisão do período da tarefa de maior período pelo período da tarefa de menor período, esse valor então é subtraído do valor inteiro mais próximo, obtendo-se o fator de multiplicidade entre das duas tarefas. Para obter o fator de multiplicidade de uma tarefa em relação a um conjunto de tarefa, é calculada a média entre os fatores de multiplicidade entre a tarefa em questão e todas as outras tarefas do conjunto.

4.2.5 Gap

Para calcular o *gap* de um *core*, deve-se calcular o *response time* de todas suas tarefas por ordem de prioridade até que se chegue na última tarefa ou até que uma tarefa perca seu prazo de acordo com a análise. Caso todas tarefas tenham cumprido seu prazo, o *gap* do *cores* (G) será dado pelo período da sua tarefa de menor prioridade subtraído do *response time* da mesma, resultando na equação:

$$G = T - r \tag{8}$$

No segundo caso, o *gap* resultará em um número negativo dado pelo período da primeira tarefa a perder o prazo, subtraído de seu *response time* que é somado ao somatório do período de todas tarefas residentes no *core*, que possuam uma prioridade mais baixa que a primeira tarefa a perder o prazo. Assim, é obtido a equação:

$$G = T - (r + \sum T) \tag{10}$$

5 Implementação dos algoritmos de busca

Os algoritmos selecionados para serem implementados neste trabalho são o algoritmo de subida de colina (Nilsson, 1982) e o algoritmo genético (Mitchell, 1998). Apesar desses algoritmos consistirem em uma série de passos fixa, os dados utilizados e certos procedimentos envolvidos em tais algoritmos devem ser adaptados ao problema a ser resolvido, nesta seção serão discutidas tais adaptações.

5.1 Subida de colina

Nessa seção é discutida a modelagem e implementação do algoritmo de subida de colina e, com o objetivo de demonstrar o processo completo de modelagem, foi utilizada na descrição de sua implementação a combinação de heurísticas que apresentou os melhores resultados na fase de testes. Para a utilização do algoritmo de subida de colina no mapeamento de tarefas, se faz necessária a definição de certos procedimentos envolvidos no funcionamento do mesmo, sendo eles:

- **Movimento:** o movimento selecionado para a exploração das soluções do algoritmo de subida de colina foi a troca de uma tarefa aleatória do seu *core* corrente para um novo *core*, caso a inserção desta tarefa sobrecarregue o *core*, uma tarefa será retirada do mesmo e enviada para um *core* aleatório (sem restrições de sobrecarga desta vez). Porém, para se melhorar o direcionamento da busca, será realizada uma filtragem nos novos estados gerados pelo movimento. Esta filtragem será realizada utilizando-se duas heurísticas de avaliação, a primeira sendo o desvio padrão de utilização dos *cores* e a segunda sendo o estresse de rede. No momento, utiliza-se a condição de que haja uma melhora em um dos dois fatores em relação ao mapeamento anterior, visando limitar a restrição no espaço de busca imposta pela filtragem de estados.
- **Avaliação:** a avaliação será realizada utilizando-se dois valores heurísticos distintos, o número de erros temporais e o desvio padrão da folga periódica dos

cores. Enquanto a solução corrente não atingir o marco de zero erros temporais (ou seja, um mapeamento escalonável), as soluções serão avaliadas pelo desvio padrão da folga periódica dos *cores*, sendo o número de erros temporais critério de desempate. Após o marco de zero erros temporais ser atingido, apenas soluções com zero erros serão avaliadas, sendo o critério de avaliação a folga periódica.

- **Critério de parada:** o critério de parada será um limite de *n* iterações sem melhora na avaliação das soluções, sendo *n* um número arbitrário.

5.2 Algoritmo genético

Da mesma forma que foi utilizada a heurística que obteve melhores resultados para a descrição da implementação do algoritmo de subida de colina, utilizaremos a heurística que obteve os melhores resultados nos testes do algoritmo genética na descrição de sua implementação, com o objetivo de prover uma noção completa do processo de modelagem do mesmo. Para utilizar-se o algoritmo genético para o mapeamento de tarefas, inicialmente, deve-se realizar a modelagem do cromossomo. Para este trabalho, o cromossomo foi definido como um mapeamento entre tarefa e *core*. Tal relação utiliza-se de uma tarefa como chave retornando um *core* que seria o *core* ao qual a tarefa está alocada.

Após a modelagem do cromossomo, foi feita a definição dos seguintes procedimentos pertinentes ao algoritmo genético:

- **Reprodução:** como o mapeamento entre tarefa e *core* não implícita uma ordem certa, adaptações foram feitas para se simular o corte nos cromossomos. Para tal, é utilizada uma lista contendo todas as tarefas do sistema, esta lista é segmentada simulando os cortes realizados na reprodução tradicional do algoritmo genético. Cada segmento, então, é mapeado de acordo com um dos cromossomos, sendo os mesmos alternados entre segmentos. Foi definido que a lista sofrerá três cortes resultando em quatro segmentos, a localização de tais

cortes é definida aleatoriamente a cada reprodução.

- **Elitismo:** foi optada pela utilização do elitismo, portanto, o cromossomo representante da melhor solução encontrada até o dado momento será introduzida na próxima população a ser gerada.

- **Mutação:** para a mutação foi selecionado um procedimento similar ao movimento do algoritmo de subida de colina do item anterior porém sem a tentativa de evitar sobrecarga dos *cores* na primeira troca. Como está se utilizando uma modificação direcionada para o melhoramento da solução, foi atribuída uma taxa de ocorrência de 20% para mutação.

- **Avaliação:** para avaliação dos indivíduos (cromossomos) da população, são utilizados o número de erros temporais (E), o desvio padrão de utilização dos *cores* (Du) e o estresse de rede (S) de acordo com a seguinte equação:

$$Fitness = E + Du + S \tag{11}$$

  Como o desvio padrão de utilização e o estresse de rede resultam em números fracionários, pelo fato de serem calculados utilizando-se percentuais, o número de erros temporais será o fator dominante da avaliação e a interação entre o balanceamento dos cores e o estresse infligido na rede se tornará o critério de desempate da avaliação, de forma a criar um avaliação de progresso entre mapeamentos que resultam no mesmo número de erros temporais.

- **Critério de parada:** no momento, o critério de parada utilizado é apenas um número arbitrário de populações geradas. No futuro, poderá ser utilizada uma medida que indique a convergência de uma população para uma mesma solução como, por exemplo, o desvio padrão do *fitness* dos indivíduos de cada população.

- **Tamanho da população:** ainda não foram definidos tamanhos ideais para populações de acordo com o conjunto que está sendo avaliado. No momento, está sendo utilizado o número de tarefas no sistema somado ao número de fluxos gerados pelas mesmas.

6 Testes e Resultados

Para a realização dos testes, foi utilizado um conjunto sintético de tarefas, tal conjunto consiste em 78 tarefas que foram executadas em uma NoC 4X3 (12 *cores*). Desse conjunto de tarefas, 39 delas são tarefas com tempo de computação que desencadearão fluxos de dados, as outras 39 tarefas são unicamente receptoras de comunicação, não possuindo tempo de computação. Este conjunto de tarefas possui algumas características que tem a finalidade de aumentar a dificuldade do mapeamento, sendo elas:

- **Tarefas Grandes (*chuncky tasks*):** esse conjunto possui tarefas com alto grau de utilização (75% de utilização do processador), o que força os algoritmos a isolá-las em *cores* pois estas tarefas, muito provavelmente, provocarão perdas de prazos, caso estejam compartilhando o *cores* com outras tarefas.
- **Alto Grau de Utilização:** este conjunto de tarefas impõe um alto grau de utilização de processadores causando uma média de ocupação de processadores de cerca de 74.58%, o que torna difícil alcançar o limite de utilização proposto em (Liu e Layland, 1973) em grande parte dos *cores*.

Os testes foram divididos por algoritmo. Em cada um dos dois algoritmos de busca utilizados nesse trabalho foram testados, primeiramente, os efeitos de cada uma das heurísticas isoladas e, posteriormente, foram testadas as combinações de heurísticas selecionadas para cada um dos algoritmos. Cada teste consistirá em 10 repetições da execução do algoritmo até seu critério de parada que será 150 iterações sem melhora para o algoritmo de subida de colina e 350 gerações para o algoritmo genético, números decididos arbitrariamente tendo em vista o tempo de processamento dos algoritmos. Para se isolar o efeito de cada heurística no algoritmo de subida de colina, foram removidos os filtros da movimentação sugeridos na Seção 5.1 pois eles utilizam duas das heurísticas testadas e isso introduziria interação entre heurísticas.

Todos os testes foram realizados na máquina descrita a seguir:

- **Processador:** Phenom X4 3.8 Ghz com 8 MB de memória cache
- **Memória RAM:** 3.5 GB DDR3 devido a limitações do sistema operacional
- **Sistema Operacional:** Windows XP Professional SP3
- **HD:** 1 TB 7200 rpm
- **Ambiente de execução:** Eclipse Indigo 3.7.1

Os tempos de execução dos algoritmos foram em média de 40 minutos para uma execução completo (até o critério de parada definido) para o algoritmo genético e de cerca de 30 minutos para execução completa do algoritmo de subida de colina. Os tempos de execução variaram de acordo com as heurísticas (heurísticas que produziram resultados piores demoraram mais para terminar suas execuções).

Finalmente, os resultados dos testes foram traduzidos em gráficos de barras, tendo as barras de número de perdas de prazos de tarefas e número de perdas de prazos de fluxos empilhadas para possibilitar tanto a análise de balanceamento em termo desses dois parâmetros, quanto do desempenho geral da heurística, a escala do número total de erros possíveis de serem obtidos (39 erros de fluxo e 39 erros de tarefa totalizando 78 erros possíveis) é mantida em todos os gráficos para se facilitar a comparação de desempenho entre heurísticas.

6.1 Subida de Colina

Nesta seção serão discutidos os resultados obtidos nos testes realizados para cada uma das cinco heurísticas propostas nesse trabalho, sendo cada uma delas utilizada isoladamente. Após a exposição dos resultados obtidos com a utilização das heurísticas isoladas, serão mostrados os resultados obtidos com a combinação de heurísticas proposta na Seção 5.1.

6.1.1 Desvio Padrão de Utilização

Como era esperado, a utilização do desvio padrão de utilização dos *cores*

resultou em uma das taxas mais altas de tarefas escalonáveis das heurísticas isoladas (perdendo apenas para a heurística de número de perdas de prazos). O balanceamento do número de erros de tarefas em relação ao número de erros de fluxos tendeu para um maior número de erros de fluxos pois esta heurística não leva em conta os fluxos de comunicação, porém, a taxa de erros de fluxos foi, em média, apenas 15% maior que a taxa de erros de fluxos. O fato da utilização dessa heurística não apresentar um número maior de perdas de comunicação se dá ao fato de, com uma menor sobrecarga dos *cores*, as tarefas tendem a sofrer uma menor interferência no contexto geral o que resulta em um menor atraso na liberação dos fluxos que possuem um intervalo de tempo maior para chegar ao seu destino. Essa heurística utilizada junto com o algoritmo de subida de colina obteve como melhor resultado uma percentual de 89,75% das tarefas escalonáveis, sendo este valor mais baixo do que o melhor resultado obtido pelo algoritmo genético (92%), os resultados obtidos durante os testes da heurística de desvio padrão de utilização são mostrados no gráfico presente na Ilustração 3.

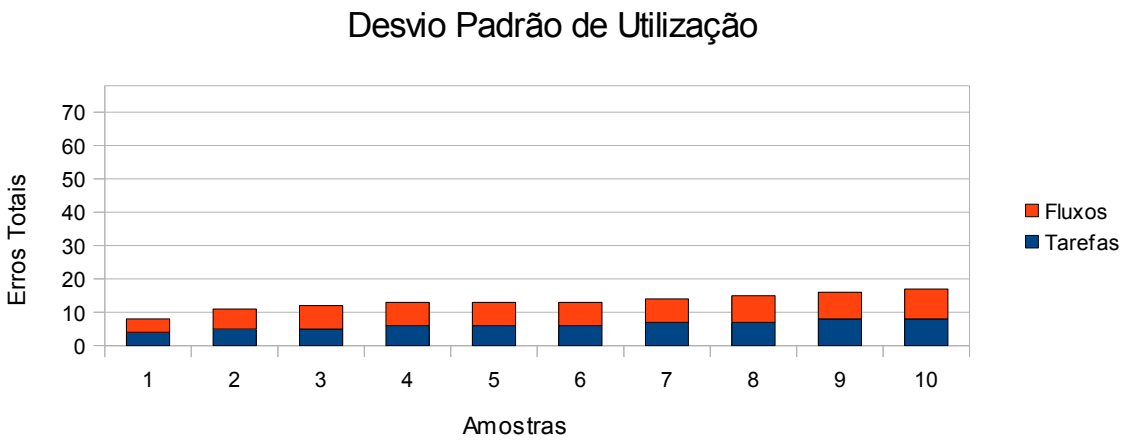

Figura 3: Resultados obtidos nos testes da utilização de desvio padrão de utilização como heurística para o mapeamento de tarefas com o algoritmo de subida de colina.

6.1.2 Número de Perdas de Prazos

Como esta heurística ataca diretamente o quesito utilizado para a avaliação

desses resultados, foi a heurística que obteve melhores resultados em relação a todas as outras heurísticas testadas isoladamente. No seu melhor resultado, esta heurística obteve uma taxa de 94,8% de escalonabilidade, tanto de tarefas, quanto de fluxos. Apesar de obter melhores resultados em relação a outras heurísticas isoladas, esta heurística falha em perceber mudanças que resultem em um estado com um mesmo número de perdas de prazos mas que possibilite uma posterior redução de perdas de prazos. A percepção de melhorias fora do escopo do número de perdas de prazos é essencial para o algoritmo de subida de colina pois este só aceita melhorias nas suas transições de estados, portanto, as mais sutis melhorias devem ser detectadas para um melhor aproveitamento deste algoritmo. Os resultados obtidos nos testes da heurística de números de perdas de prazos são demonstrados no gráfico da Ilustração 4.

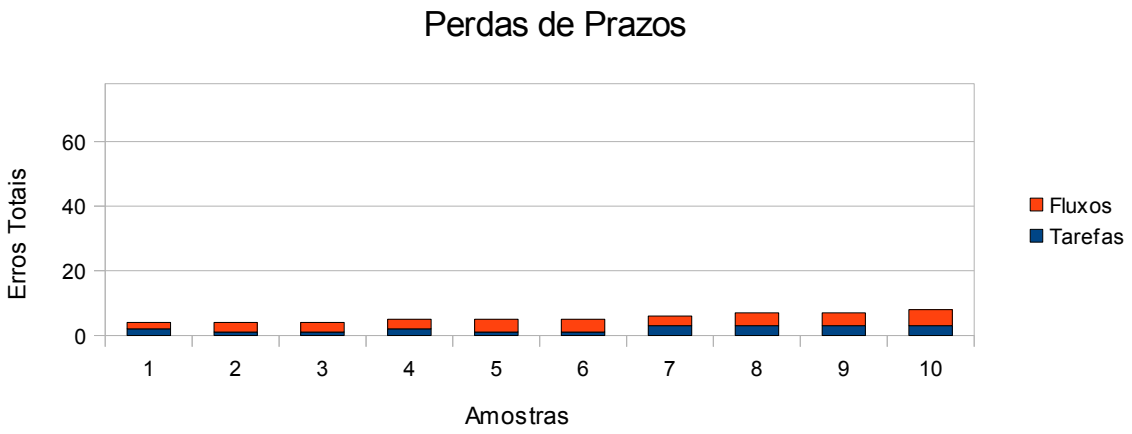

Figura 4: Resultados obtidos nos testes da utilização do número de perdas de prazos como heurística para o mapeamento de tarefas com o algoritmo de subida de colina.

6.1.3 Estresse de Rede

A heurística do estresse de rede apresentou resultados inesperados pois era esperado que o balanceamento das perdas de prazos tendesse para um maior número de perdas de prazos de tarefas em relação as perdas de prazos de fluxos. Porém, o resultado dos testes foi, em muitas ocasiões, o número de perdas de prazos de fluxos

ultrapassando o número de perdas de prazos de tarefas. Esse fenômeno pode ser explicado pela relação de dependência imposta pelo modelo de sistema pois o término da tarefa dita o inicio das atividades de comunicação e, consequentemente, o tamanho do intervalo de tempo que o fluxo de comunicação possuí para chegar ao seu destino. Como o algoritmo de subida de colina possui restrições leves quanto a sobrecarga de *core* (mesmo não utilizando uma heurística que se foque nisso), ele apresentou resultados melhores que o algoritmo genético para esta heurística pois a utilização, mesmo sendo extrapolada em alguns *cores*, estava mais balanceada do que nos resultados obtidos com o algoritmo genético. No seu melhor resultado, a heurística do estresse de rede utilizada para o algoritmo de subida de colina obteve uma taxa de 78,2% de escalonabilidade tanto para tarefas quanto para fluxos de comunicação. Deve-se ressaltar que os resultados obtidos por essa heurística, mesmo no quesito de perdas de prazos de fluxo, perderam para as duas heurísticas citadas anteriormente pelo motivo da dependência imposta pelo modelo, portanto, essa heurística deve ser utilizada junto com uma heurística de balanceamento de *cores* para obter melhores resultados. Resultados obtidos com os testes para esta heurística estão expostos no gráfico da Ilustração 5.

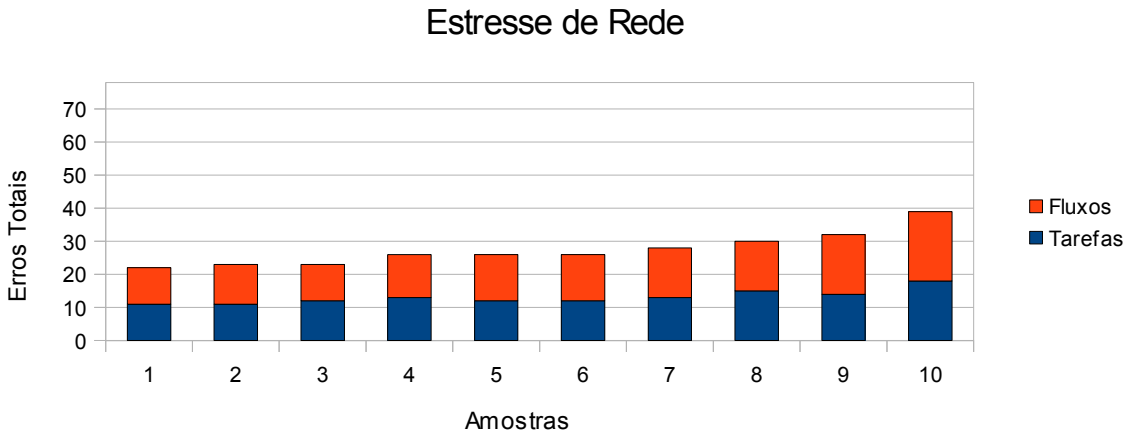

Figura 5: Resultados obtidos nos testes da utilização do estresse de rede como heurística para o mapeamento de tarefas com o algoritmo de subida de colina.

6.1.4 Harmonização do Período das Tarefas

Para o algoritmo de subida de colina, a heurística de harmonização do período das tarefas foi uma das duas heurísticas que apresentou grande variação entre a pior e a melhor solução, tendo a pior solução um número total de erros igual a 200% do número total de erros da melhor solução. Essa variação brusca pode ser explicada pela condição leve de balanceamento pois o algoritmo tenta evitar a extrapolação da utilização do *core* na primeira troca, isso cria uma maior probabilidade de existirem diversos *cores* contendo tarefas com períodos harmônicos. Porém, como o algoritmo evita o extrapolamento apenas na primeira troca, ele não impede completamente que todas as tarefas que possuam períodos relativamente harmônicos sejam acumuladas em apenas um core, resultando em um core altamente sobrecarregado ou mesmo um grande número de *cores* sobrecarregados. Para evitar-se a acumulação de todas as tarefas com períodos relativamente harmônicos em apenas um core, essa heurística deve ser utilizada em conjunto com uma heurística de balanceamento de utilização de *cores*. No melhor resultado, esta heurística obteve cerca de 77% de escalonabilidade de tarefas e 69,2% de escalonabilidade de fluxos de comunicação, um resultado melhor que os obtidos pelo algoritmo genético (pois o algoritmo genético não possui restrição de ocupação nos *cores* em nenhum nível). Os resultados obtidos nos testes realizados para a harmonização de períodos são mostrados no gráfico presente na Ilustração 6.

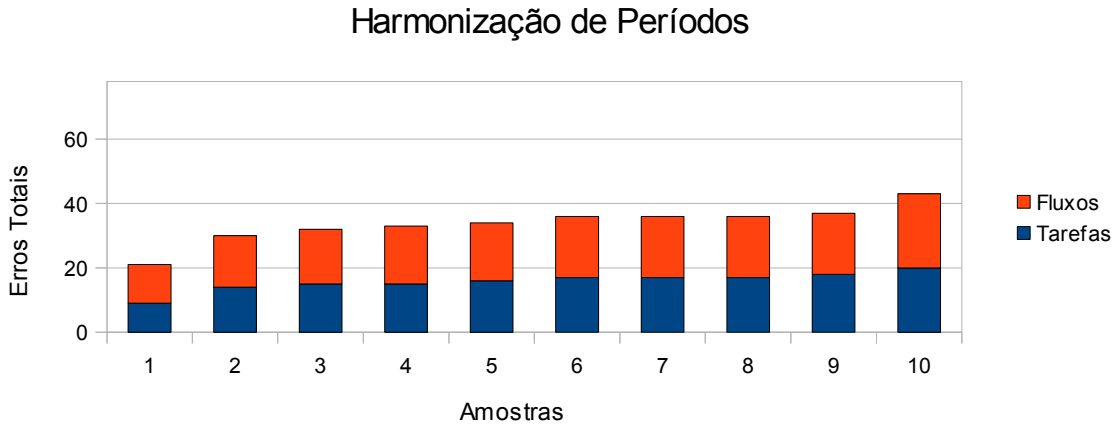

Figura 6: Resultados obtidos nos testes da utilização da harmonização de períodos como heurística para o mapeamento de tarefas com o algoritmo de subida de colina.

6.1.5 Gap

A utilização do gap como heurística para o algoritmo de subida de colina também apresentou grande variação entre o melhor e pior resultados, tendo uma variação de 260% da melhor para a pior solução. Esta variação tem motivos similares a variação ocorrida na harmonização do período de tarefas, ela ocorre pelo fato de a heurística do gap em si não apresentar um foco no balanceamento de cores mas é afetada pela restrição leve de sobrecarga do algoritmo de subida de colina resultando em resultados com a utilização relativamente balanceada e resultados com a utilização desbalanceada. No seu melhor resultado, a heurística do gap apresentou 87% de escalonabilidade de tarefas e 84,61% de escalonabilidade de fluxos, um resultado melhor que os obtidos na utilização da mesma heurística para algoritmo genético (causada pela não restrição da sobrecarga dos *cores* no algoritmo genético). Resultados obtidos nos testes desta heurística para o algoritmo de subida de colina estão expostos no gráfico da Ilustração 7.

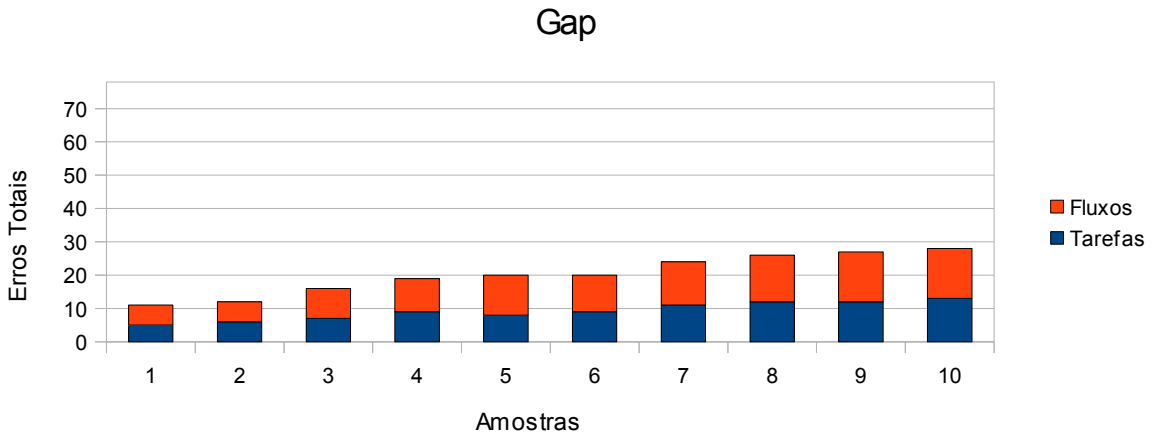

Figura 7: Resultados obtidos nos testes da utilização do gap como heurística para o mapeamento de tarefas com o algoritmo de subida de colina.

6.1.6 Combinação de Heurísticas

O algoritmo de subida de colina apresentou uma melhora mais significativa da combinação de heurísticas para o número de perdas de prazos em relação a melhora apresentada pelo algoritmo genético. Foi constatada uma melhora de 50% nos erros totais da combinação de heurísticas proposta na Seção 5.1, em relação a heurística do número de perdas de prazos. Essa diferença se dá pela característica do algoritmo de subida de colina de apenas aceitar melhoras. Como o algoritmo de subida de colina apenas aceita melhoras, ele não explora eficientemente variações de mapeamento resultantes no mesmo número de perdas de prazos, sendo que tais variações podem abrir caminho para vizinhanças com soluções ainda melhores (que resultem em um menor número de perdas de prazos. Na sua melhor solução está combinação de heurísticas resultou em 100% de escalonabilidade de tarefas e 94,9% de escalonabilidade de fluxos. Os resultados obtidos nos testes desta combinação de heurísticas estão explicitados no gráfico da Ilustração 8.

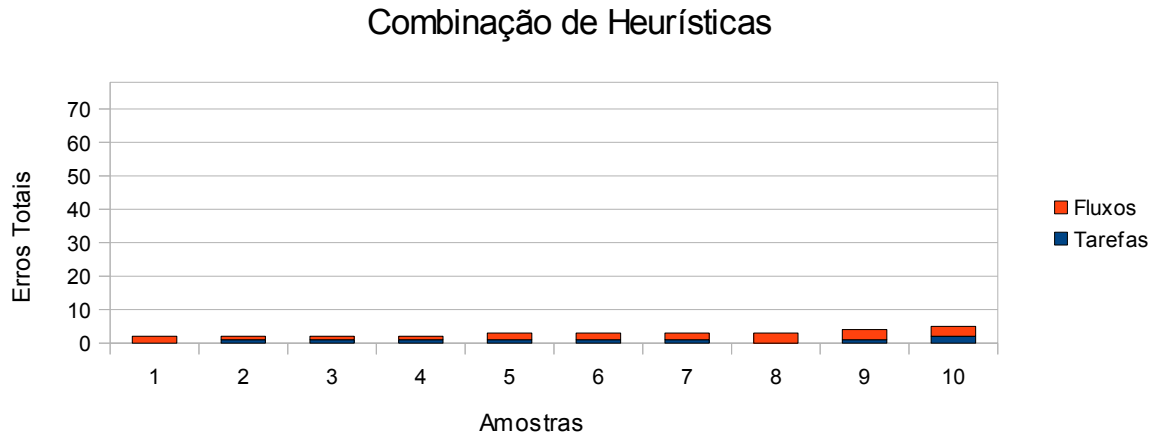

Figura 8: Resultados obtidos nos testes da utilização da combinação das heurísticas de número de perdas de prazos e gap para o mapeamento de tarefas com o algoritmo de subida de colina.

6.2 Algoritmo Genético

Nesta seção, discutiremos os resultados da utilização de cada uma das cinco heurísticas propostas nesse trabalho, isoladamente, junto com o algoritmo genético. Após a demonstração dos resultados das heurísticas isoladas, mostraremos os resultados obtidos com a combinação de heurísticas proposta na Seção 5.2.

6.2.1 Desvio Padrão de Utilização

No teste referente ao desvio padrão de utilização, como previsto na discussão sobre a heurística, os resultados tendem a apresentar um maior número de perdas de fluxos . Esta heurística apresentou um bom percentual de tarefas escalonáveis, considerando-se as dificuldades impostas pelo conjunto de tarefas, chegando a um percentual de 92% das tarefas do conjunto escalonáveis no seu melhor resultado. Os resultados dos testes para a heurística de desvio padrão de utilização são apresentados no gráfico presente na Ilustração 9.

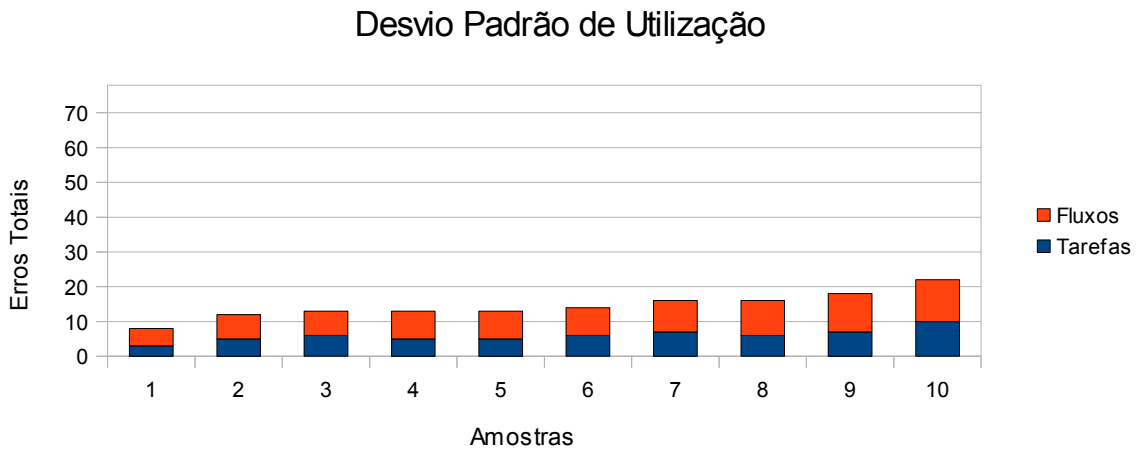

Figura 9: Resultados obtidos nos testes da utilização de desvio padrão de utilização como heurística para o mapeamento de tarefas com o algoritmo genético.

6.2.2 Número de Perdas de Prazos

Devido ao fato da heurística focada no número de perdas de prazos atacar o problema sendo abordado por estes testes diretamente, esta heurística apresentou o menor número de perdas de prazos de tarefas e fluxos, apresentando consistentemente baixos percentuais de ambos. No seu melhor resultado, esta heurística apresentou apenas cerca de 3% de perdas de prazos tanto para tarefas, quanto para fluxos. Embora essa heurística tenha apresentado resultados muito bons, ela falha em distinguir a melhoria entre dois mapeamentos que resultam na mesma quantidade de perdas de prazos. A consistência e os bons resultados produzidos por esta heurística indicam que a mesma pode ser utilizada como heurística principal para o problema de mapeamento juntamente com heurísticas secundárias para a distinção da qualidade de mapeamentos que resultem na mesma quantidade de perdas de prazos. Os resultados obtidos no teste de número de perdas de prazos são exibidos no gráfico da Ilustração 10.

Figura 10: Resultados obtidos na utilização do número de perdas de prazos como heurística para o mapeamento de tarefas com o algoritmo genético.

6.2.3 Estresse de Rede

Ao se testar a heurística de estresse de rede era esperado um maior número de ocorrências de perdas de prazos por parte de tarefas em relação ao número de perdas de prazos em relação a fluxos de comunicação, porém, os resultados apresentaram um comportamento contrário ao esperado, muitas vezes apresentando um número maior de erros de fluxos. Ao se analisar os resultados, foi descoberto que este fenômeno foi causado pela noção parcial de dependência imposta pelo modelo de sistema utilizado. O atraso de liberação imposto pelo término da tarefa no fluxo gerado pela mesma resulta em uma interferência direta na capacidade de um fluxo atender seu prazo. Portanto, *cores* sobrecarregados e uma grande quantidade de perdas de prazos de tarefas podem se traduzir em uma alta ocorrência de perdas de prazos por parte dos fluxos. Pode-se observar os resultados obtidos nos testes envolvendo estresse de rede na ilustração 11.

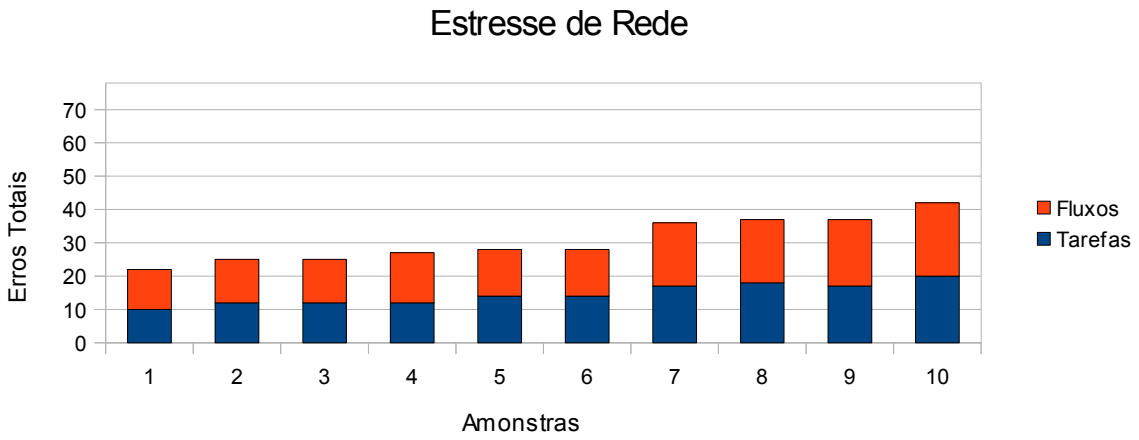

Figura 11: Resultados obtidos nos testes da utilização do estresse de rede como heurística para o mapeamento de tarefas com o algoritmo genético.

6.2.4 Harmonização do Período das Tarefas

De todas as heurísticas com o foco na redução da perda de prazos de tarefas utilizadas em conjunto com o algoritmo genético, a harmonização de períodos de tarefas foi a que apresentou a maior quantidade de perdas de prazos de tarefas, cerca de 31% de perdas de prazos no melhor resultado. Esse fato ocorreu devido a falta de uma condição de balanceamento da utilização dos *cores*, resultando em *cores* muito sobrecarregados pelo fato das tarefas alocadas a ele possuírem períodos altamente harmônicos. Portanto, para a utilização dessa heurística, deve-se incluir uma heurística de balanceamento como o desvio padrão de utilização.

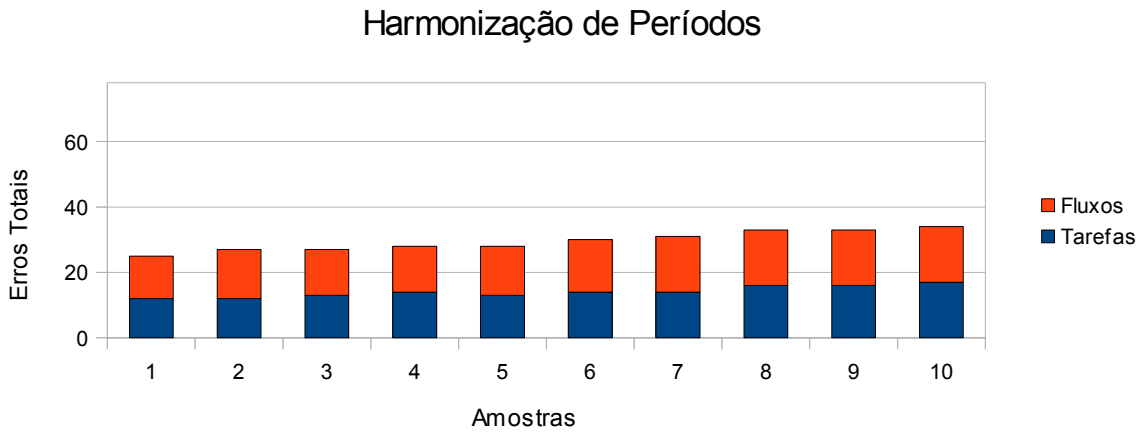

Figura 12: Resultados obtidos nos testes da utilização da harmonização de períodos como heurística para o mapeamento de tarefas com o algoritmo genético.

6.2.5 Gap

Os resultados dos testes da utilização do gap como heurística se comportaram de forma previsível quanto ao balanceamento entre perdas de prazos de tarefas e fluxos, apresentando uma maior ocorrência de perda de prazos de fluxos, porém, eles apresentaram uma taxa de perdas de prazos de tarefas maior que a apresentada na utilização do desvio padrão de utilização, outra heurística que foca-se em balancear a interferência sofrida pelas tarefas em cada *core*, como se pode notar comparando os gráficos presentes nas Ilustrações 2 e 3. No melhor dos resultados, a utilização do gap para o algoritmo genético apresentou uma taxa de cerca de 18% de perdas de prazos de tarefas. Os resultados obtidos nos testes da utilização do gap para algoritmo genético são mostrados na Ilustração 13.

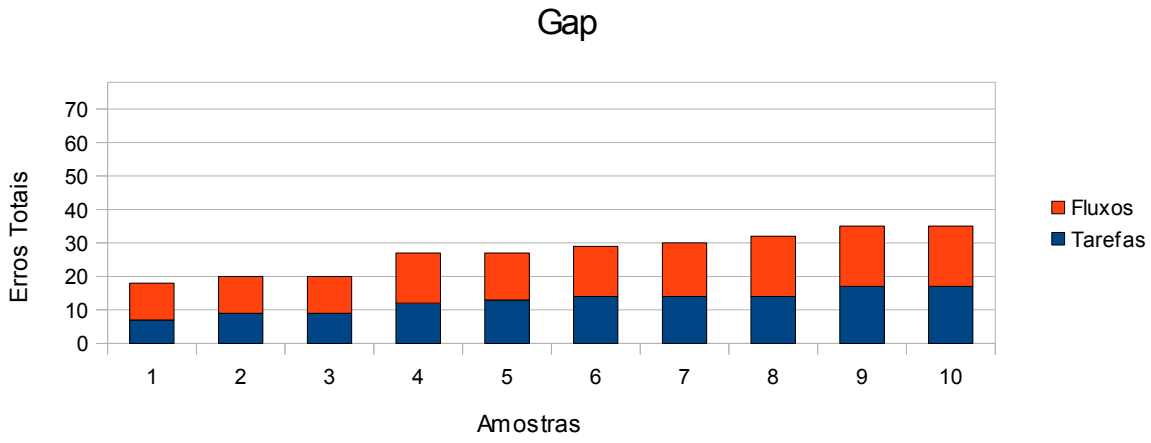

Figura 13: Resultados obtidos nos testes da utilização do gap como heurística para o mapeamento de tarefas com o algoritmo genético.

6.2.6 Combinação de Heurísticas

Como proposto na Seção 6.2 e posteriormente ressaltado na discussão sobre a utilização do número de perdas de prazos como heurística para o mapeamento de tarefas com o algoritmo genético, foram realizados testes combinando a heurística de perdas de prazos, juntamente com as heurísticas de estresse de rede e desvio padrão de utilização para a avaliação de estados que resultem em um mesmo número de erros. Estas heurísticas foram modeladas em um único valor numérico da forma que o número de erros de tarefas ocupará a parte inteira do valor e a soma do estresse de rede e do desvio padrão de utilização ocuparão a parte fracionária do valor. Esta combinação de heurísticas apresentou melhores resultados que qualquer heurística isolada, chegando a um estado com 100% de escalonabilidade de tarefas e 97% de escalonabilidade de fluxos. Os resultados obtidos no teste da combinação de heurísticas para algoritmo genético estão representados na Ilustração 14.

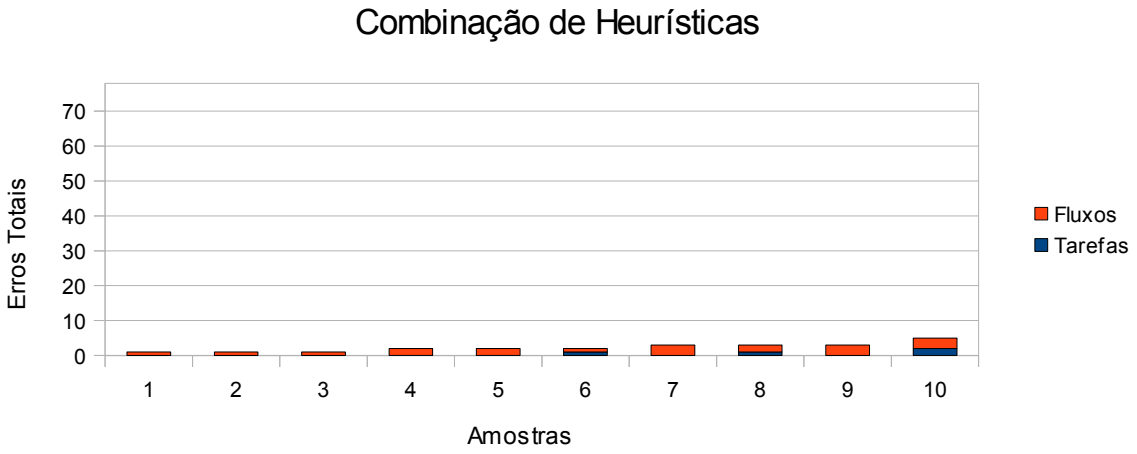

Figura 14: Resultados obtidos na utilização da combinação das heurísticas de número de perdas de prazos, desvio padrão de utilização e estresse de rede como heurística para o mapeamento de tarefas com o algoritmo genético.

7 Considerações Finais

A comunicação entre tarefas é um fator que impacta diretamente na escalonabilidade de um sistema de tempo real que vem sendo desconsiderado pela literatura atual referente ao mapeamento de tarefas, sendo a mesma normalmente focada apenas no tempo de execução das tarefas. Neste trabalho foi demonstrada a implementação de algoritmos de busca que, juntamente com heurísticas resultantes do estudo de modelos de análise de escalonamento para tarefas e fluxos de comunicação, abordam o tempo de execução e de comunicação simultaneamente. Tal implementação foi baseada em um novo modelo de sistema criado, com o objetivo de suprir a necessidade de se levar em consideração a comunicação e a execução em um único modelo, a partir da junção de dois modelos de análise de escalonamento, o *response time* que se trata de um modelo de análise de escalonamento de tarefas e o *priority window* que é um modelo de análise de escalonamento para fluxos de comunicação.

Os algoritmos de busca implementados juntamente com as heurísticas desenvolvidas apresentaram resultados favoráveis na fase de testes resultando em mapeamentos muito próximos da escalonabilidade total para um conjunto de tarefas que foi desenhado para impor uma série desafios para o seu mapeamento. Os resultados obtidos demonstram que a utilização de algoritmos de busca, juntamente com heurísticas voltadas para a escalonabilidade de fluxos de comunicação e tarefas, é uma alternativa viável para o mapeamento automático de tarefas em sistemas de tempo real.

O processo de implementação e testes evidenciou áreas nas quais poderiam ser realizadas melhoras, tanto para os algoritmos quanto o modelo de sistema utilizado. Atualmente, o algoritmo genético (o mais demorado dos dois algoritmos) termina sua execução (considerando-se o critério de para descrito nos testes) em aproximadamente 40 minutos, nas condições descritas no Capítulo 6, variando de acordo com a qualidade dos indivíduos em todas as populações. A variação de tempo de execução com as qualidade dos mapeamentos gerados é explicada pela análise de escalonamento, um dos elementos de maior peso computacional do algoritmo, se

tornar mais demorada a medida que os *cores* tornam-se mais sobrecarregados (pois aumenta o conjunto de interferência a ser calculado e cada *loop* da análise de escalonamento). Como este trabalho foi focado no mapeamento *off-line* (mapeamento é realizado antes do sistema ser executado e permanece o mesmo), o tempo de execução não é um fator crítico, porém, se este método for utilizado para mapeamentos *on-line*, o tempo de execução deverá ser melhorado.

O modelo de sistema utilizado nos testes possui uma noção parcial de dependência, sendo que o término da execução das tarefas exerce influência no inicio das atividades de comunicação, a possível dependência da tarefa que recebe a comunicação é simulada através do prazo atribuído ao fluxo que se trata de uma aproximação e não fornece um comportamento tão realista quanto uma simulação de dependência completa forneceria.

Finalmente, a convergência geral do método pode ser melhorada pois cada um dos dois algoritmos testados no decorrer deste trabalho possuem características distintas de convergência. Enquanto o algoritmo genético possui uma rápida convergência grossa (convergência de soluções consideradas ruins para soluções consideradas de média ou boa qualidade), apresentando uma lenta convergência fina (convergência de soluções consideradas boas ou médias para soluções próximas ao ótimo), o algoritmo de subida de colina apresenta uma lenta convergência grossa e uma convergência fina mais rápida.

7.1 Trabalhos Futuros

Tendo em vista as áreas nas quais foram discutidas possíveis melhoras, foram consideradas modificações a serem realizadas no modelo de sistema e na implementação com o intuito de obter tais melhoras.

Para se reduzir o tempo computacional do algoritmo, existem duas opções principais: a utilização de um algoritmo mais rápido (como árvores de decisão) ou utilizar-se de uma abordagem que não utilize a análise de escalonamento (substituindo-a por heurísticas com um menor peso computacional).

A noção de dependência do modelo pode ser mudada para uma noção de dependência completa, utilizando-se o resultado final da análise de escalonamento de tarefas e fluxos para determinar o atraso imposto ao início da execução da tarefa dependente. Munido do atraso imposto ao início da tarefa dependente, uma análise completa de escalonamento pode ser realizada, determinando se, mediante tal atraso, o sistema continuará escalonável. Deve-se ressaltar que a inclusão da noção total de dependência no modelo vai aumentar sua complexidade e, consequentemente, seu peso computacional.

Para melhorar-se a convergência geral, pode-se utilizar uma abordagem híbrida, utilizando o algoritmo genético para gerar soluções relativamente boas e as refinando através do algoritmo de subida de colina com heurísticas voltadas para um ajuste fino. Dessa forma, as melhores características de convergência de cada um dos algoritmos poderia ser aproveitada.

Referências Bibliográficas

R. I. Davis e A. Burns. *A Survey of Hard Real-Time Scheduling for Multiprocessor Systems,* ACM Computing Surveys, vol. 43, no. 4, 35-35, 2011.

A. Burchard, J. Liebeherr, Y. Oh e S.H. Son. *New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems,* IEEE Transactions on Computers, vol. 44, no. 12, 1429-1442, 1995.

Z. Shi e A. Burns. Schedulability Analysis and task mapping for real-time on-chip communication, Real-Time System, vol. 46, issue 3, 360-385, 2010.

L. M. Ni e P. K. McKinley. A survey of wormhole routing techniques in direct networks, IEE Computer, vol. 26, issue 2, 62-76, 1993.

C. L. Liu e J. W. Layland. Scheduling Algorithms for Multiprogramming in Hard-Real-Time Enviroment, Journal of the ACM, vol. 10, issue 1, 1973.

M. Mitchell. *An introduction to genetic algorithms*, Massachusetts Institute of Tecnology, MIT press, 1998.

N. J. Nilsson. *Principles of Artificial Intelligence*, Springer-Verlag, 1982.

R. Wilhelm e al. The worst-case execution-time problem - overview of methods and survey of tools, ACM Transactions on Embedded Computing Systems, vol. 7, issue 3, 1-53. 2008.

J. A. Stankovick. *Misconceptions About Real-Time Computing: A Serious Problem for Next- generation Systems*, Computer, vol. 21, issue 10, 10-19, 1988.

N. C. Audsley, A. Burns, R. I. Davis, K. Tindell e A. J. Wellings. *Fixed Priority Pre-emptive Scheduling: An Historical Perspective*, Real-Time Systems, vol. 8, issue 2-3, 173-198, 1995.

J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson e S. K. Bariah. *A categorization of real-time multiprocessor scheduling problems and algorithms*, em Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman and Hall, 2004.

J. D. Ullman. *NP-Complete Scheduling Problems*, Journal of Computer and System Sciences, vol. 10, issue 3, 384-393, 1975.