Escalonamento Estático de Tarefas em Ambientes Computacionais Heterogêneos sob o Modelo *LogP*

Deolinda Fontes Cardoso

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Orientador: Eugene Francis Vinod Rebello

Niterói, 20 de dezembro de 2004.

Escalonamento Estático de Tarefas em Ambientes Computacionais Heterogêneos sob o Modelo *LogP*

Deolinda Fontes Cardoso

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Processamento Paralelo e Distribuído.

Aprovada por:

Prof. Eugene Francis Vinod Rebello - IC/UFF (Presidente)

Prof. Luiz Satoru Ochi - IC/UFF

Prof. Bruno Richard Schulze - LNCC

Niterói, 20 de Dezembro de 2004.

"*Para o meu Pai,*

que me presenteou com o meu lindo nome."

Agradecimentos

Ao CASNAV e a Marinha do Brasil por terem proporcionado a oportunidade e todo o incentivo necessário à conclusão desta pesquisa desenvolvida.

À UFF, aos professores e amigos, em especial a Jacques Alves da Silva, que de alguma forma contribuíram para o aprimoramento do meu conhecimento.

Aos membros da Banca, Professor Luiz Satoru Ochi e Professor Bruno Richard Schulze pela paciência, cordialidade e valorosas contribuições para esta Dissertação.

À minha família, em especial aos meus filhos, por todo Amor e compreensão sempre presentes. Mamãe ama voces!

Por fim, o maior agradecimento ao meu Orientador, Professor Vinod Rebello, por todo o seu apoio, incentivo e preocupação para a realização deste trabalho. Muito obrigada por partilhar comigo o seu conhecimento, pelos bons conselhos, pela sábia orientação, por confiar em mim e por me fazer acreditar que eu seria capaz. Foi realmente muito importante a sua presença para que eu realizasse, com sucesso, esta etapa da árdua tripla jornada da minha vida: Família-Trabalho-Estudo. Muito Obrigada por tudo, sempre!

Resumo da Tese apresentada ao Programa de Pós-Graduação em Computação do Instituto de Computação da UFF como requisito parcial para a obtenção do grau de Mestre em Computação (MSc).

Escalonamento Estático de Tarefas em Ambientes Computacionais Heterogêneos sob o Modelo *LogP*

Deolinda Fontes Cardoso

20/Dezembro/2004

Orientador: Eugene Francis Vinod Rebello

Programa de Pós-Graduação em Computação - IC/UFF

Atualmente diversas aplicações necessitam de poder computacional superior ao que um computador seqüencial pode fornecer para serem executadas em tempos aceitáveis. Face aos altos custos dos supercomputadores, alternativas como clusters e, mais recentemente, os grids computacionais estabeleceram-se como formas de agregar poder computacional a custos acessíveis. O objetivo de um grid é agregar equipamentos, distribuídos, heterogêneos e compartilhados formando uma "constelação"de recursos interconectados por redes de alta velocidade. Nessa "via lactea"de recursos é esperado um comportamento tipicamente dinâmico, uma vez que a disponibilidade dos mesmos nem sempre pode ser garantida. Sob esses aspectos, é imperativo que a execução das aplicações paralelas seja eficiente, ou seja, o paralelismo nelas existentes seja convenientemente explorado para que o potencial latente da grade possa ser totalmente aproveitado. O problema é tratado em termos do escalonamento de tarefas, alvo de muitos estudos e foco desta pesquisa. Devido ao comportamento instável do ambiente é necessário um escalonador dinâmico para realizar a alocação das tarefas aos recursos durante a execução da aplicação. Uma estratégia para aliviar a carga de trabalho do escalonador dinâmico é incorporar um pré-escalonamento de tarefas, através de um escalonador estático, assim, ao dinâ-

mico restará apenas os ajustes finais momentâneos. O objetivo do trabalho é propôr um escalonador estático para ambientes com características semelhantes às grades computacionais. A abordagem escolhida para o problema, apresenta uma heurística da classe de list scheduling para um número limitado de processadores heterogêneos. Durante a pesquisa foi possível observar que existem inúmeras heurísticas formuladas para ambientes homogêneos, e bem menos para ambientes heterogêneos. Entretanto, quase todas são formuladas em modelos de comunicação inapropriados para plataformas de clusters e grades. Por essa razão, resolveu-se adotar um modelo mais realístico como o LogP que considera parâmetros que permitem o cálculo preciso do custo das comunicações. A nova heurística é uma extensão da metodologia proposta por Kalinowski, Kort e Trystram adaptada para ambientes heterogêneos e implementada em quatro versões. A investigação consiste em avaliar políticas alternativas para minimizar os efeitos adversos que as sobrecargas decorrentes das comunicações causam ao *makespan* das aplicações paralelas.

Palavras-chave

1. Processamento Paralelo
2. O Problema do Escalonamento de Tarefas (PET)
3. Processadores Heterogêneos
4. Tempo de Execução Paralelo ou *Makespan*
5. Heurísticas
6. Modelo *LogP*
7. Clusters e Grades Computacionais

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Deolinda Fontes Cardoso

20/December/2004

Advisor: Eugene Francis Vinod Rebello
Department: Computer Science

Numerous applications require more performance than even state-of-the-art sequencial computers can provide in order to be executed in acceptable time frames. With high costs for the acquisition and maintenance of supercomputers, cheaper parallel computing alternatives such as Computing Clusters, and more recently Computational Grids, are now becoming the computing systems of choice within research centers, companies and universities. On these platforms, the efficient scheduling of the tasks of a parallel application is crucial to obtaining good performance. This work studies the problem of scheduling tasks in systems of distributed heterogeneous resources which communicate via message passing. The processing costs to send and to receive messages (traditional ignored by scheduling algorithms) can dramatically influence the execution time of parallel applications. In this Dissertation three new strategies are proposed to enable list scheduling heuristics to handle these overhead costs appropriately in order to generate efficient schedules for environments such as Clusters and Computational Grids. Based on the LogP model, results show that two of the proposed strategies provide significant improvements over the only existing approach known in the literature.

Keywords

1. Parallel Processing
2. The Task Scheduling Problem
3. Heterogeneous Processors
4. *Makespan*
5. Heuristics
6. *LogP* Model
7. Clusters and Grids Computing

Glossário

GAD	:	Grafo Acíclico Direcionado;
KPSG	:	Kwok Peer Set Graphs;
AET	:	Algoritmo de Escalonamento de Tarefas;
PET	:	Problema de Escalonamento de Tarefas;

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente devido a crescente utilização do processamento de informações nas mais diversas áreas, um número cada vez maior de aplicações necessita de capacidade computacional superior ao que o estado da arte de um computador seqüencial pode fornecer. Com isso, a demanda por máquinas que apresentem elevado desempenho tem-se constituído um fator crucial para os investimentos em pesquisas para melhoria da tecnologia e surgimento de novas arquiteturas. Nota-se, porém, a existência de uma crescente dificuldade em se obter mais desempenho numa arquitetura seqüencial, fatores como a velocidade da luz, as leis termodinâmicas, os processos de resfriamento do processador e seus custos de fabricação representam consideráveis restrições a esses avanços [53].

Durante vários anos acreditou-se que a quantidade de paralelismo que poderia ser explorado não justificaria os esforços para prover recursos para suportar a execução simultânea de diversas instruções. Entretanto, estudos apresentados mostraram que a utilização de certas técnicas tanto em *software* como em *hardware* podiam aumentar a quantidade de paralelismo a ser explorado, incentivando novas pesquisas [53]. Assim sendo, foram projetadas máquinas capazes de executar tarefas em paralelo com múltiplos processadores conectados, conhecidas como computadores paralelos ou, as de grande porte, como supercomputadores. Porém, tais máquinas

apresentam desenvolvimento e manutenção elevados e por isso sua utilização, para a maioria dos países subdesenvolvidos, é ainda muito restrita.

Nos últimos anos, principalmente, devido as melhorias nas tecnologias de redes de comunicação e aumento da largura de banda de transmissão [31], alternativas com custos bastantes acessíveis, facilitou o estabelecimento de coleções de computadores tradicionais, como forma de agregar poder computacional [10]. A este conceito de processamento paralelo denominamos de *Cluster Computing* [12, 64]. Trata-se de um agrupamento de máquinas PC tradicionais conectadas por uma rede de comunicação de alta velocidade, dedicadas para compartilhar recursos e armazenamento de dados [28]. Recentemente, é possível interconectar diversos *clusters* localizados em domínios geograficamente distribuídos, interconectados por redes *Internet*, constituindo, assim, "constelações"de recursos. A essa nova tecnologia chamamos de *Grid Computing* [12]. A terminologia *Grid* é uma metáfora à rede de energia elétrica uma vez que se espera que o poder computacional requerido esteja disponível, na *Internet*, tal como a eletricidade está na rede elética [32]. Grades Computacionais consistem em ambientes compostos por recursos heterogêneos, compartilhados e distribuídos. Além disso, a disponibilidade dos mesmos não é sempre garantida o que atribui a plataforma de Grade um comportamento instável e dinâmico. Sob esse aspecto, é imperativo que a execução das aplicações seja eficiente [10], ou seja, que o paralelismo existente nessas aplicações seja convenientemente explorado para que o potencial latente de uma grade possa ser completamente aproveitado. Esta determinação é tratada em termos do Problema de Escalonamento de Tarefas (PET), alvo de muitos estudos [47] e foco desta pesquisa. O problema consiste em designar as tarefas, que compõem a aplicação, aos diversos processadores do sistema de tal forma que o tempo total da execução, ou *Makespan*, seja mínimo.

Esse problema não é trivial e sim, considerado *NP-Completo* em sua forma mais geral [7, 33, 59] assim, heurísticas ou métodos aproximados são usados para se atingir resultados satisfatórios próximos do ótimo. Geralmente, os métodos de escalonamento propostos são formulados em modelos computacionais, da aplicação e da arquitetura, que permitem certas simplificações do problema [28]. Por exemplo, arquiteturas computacionais com um número infinito de processadores homogêneos;

eventos de comunicação cujo único parâmetro relevante é a latência, assim, sobrecargas decorrentes das transmissões e recepções de dados são desconsideradas. Com essas simplificações, os resultados podem convergir para valores práticos pouco eficientes, uma vez que a maioria dos ambientes computacionais são compostos por máquinas heterogêneas com diferentes capacidades de processamento que impõem sobrecargas quando se comunicam por troca de mensagens.

Nos últimos anos têm sido propostas, diversas heurísticas para ambientes heterogêneos [6, 73]. A maior parte delas formuladas pela técnica de escalonamento de *List Scheduling* [23, 56, 73]. A razão se deve principalmente, a baixa complexidade dessa técnica em comparação às outras existentes: Aglomeração e Meta-Heurísticas. Ainda mais, a técnica de *List Scheduling* é facilmente adaptável a ambientes computacionais com um número limitado de processadores, sejam eles homogêneos ou heterogêneos. Em contrapartida, a grande maioria dessas heurísticas adota o modelo de comunicação de Latência, no qual apenas a latência da comunicação é considerada relevante. Trata-se de um modelo bastante popular na área de escalonamento e tem sido muito explorado em comparação a modelos mais relísticos como o modelo *LogP*. Este estudo, em concordância com pesquisas publicadas em [22] adota esse último modelo por ser um dos mais realísticos para modelar a comunicação em máquinas paralelas reais. Sua principal metodologia consiste em enfatizar que, em adição a latência, existem outros parâmetros relevantes, como as sobrecargas, que não devem ser desprezadas sob pena de degradação dos resultados teóricos. Sob essa consideração, o problema de escalonamento torna-se ainda mais complexo.

A abordagem, deste trabalho, realiza uma investigação sobre o escalonamento estático de tarefas, baseado em heurísticas, sob o modelo *LogP*, para ambientes como as grades computacionais. Neste ambiente devido a distribuição geográfica dos recursos é esperada elevada latência pela transmissão das mensagens. Além disso, a oferta de recursos é dinâmica, limitada e de natureza heterogênea. Isso torna, ainda, mais complexa a atribuição eficiente de tarefas as unidades de processamento diponíveis. Esta atribuição deve ser realizada por um escalonador dinâmico, para que as decisões de escalonamento sejam feitas durante a execução da aplicação. Entretanto, o procedimento adotado no Projeto *EasyGrid* [63] consiste em utilizar um

escalonador híbrido [9], composto por um escalonador estático e outro dinâmico. Ao escalonador estático compete realizar um pré-escalonamento de tarefas para aliviar a carga de trabalho do escalonador dinâmico. Com isso, restará ao dinâmico apenas os ajustes finais momentâneos. Esta dissertação foca na parte estática do escalonador híbrido do *Framework Easygrid* [62]. Condições fundamentais são que o escalonador proposto apresente desempenho satisfatório, seja econômico em termos de utilização de processadores e apresente baixa complexidade. Por essas razões foi escolhida a técnica de *List Scheduling*.

A pesquisa realizada é baseada numa extensão ao estudo de Kalinowski, Kort e Trystram [40], que descreve uma metodologia para adaptar uma heurística *List Scheduling* aos parâmetros do modelo *LogP*. Baseada nessa metodologia são propostas novas versões, com variações da técnica dos autores, com o objetivo de avaliar qual a melhor estratégia para tratar as sobrecargas da comunicação. O objetivo principal é minimizar seus efeitos adversos no tempo de execução da aplicação escalonada ou *makespan*. Aliás, o *makespan* do escalonamento é a métrica considerada na avaliação dos resultados. Quanto menor for esse valor, para uma dada arquitetura composta de um número limitado de processadores, melhor será considerado o desempenho da heurística. Outra métrica utilizada é a qualidade do *makespan*; quanto mais próximo do valor unitário menor é a degradação em relação ao melhor *makespan* atingido na instância de testes, e portanto, melhor é a qualidade do resultado.

1.1 Contribuições

Com esta dissertação espera-se identificar estratégias apropriadas para o tratamento das sobrecargas de comunicação em heurísticas de escalonamento. Realçar a importância da utilização de modelos de comunicação realísticos, como o *LogP*, para se garantir a fidelidade dos resultados entre a teoria e a prática. Descrever a importância de considerar as características heterogêneas reais do ambiente computacional. Contribuir para o sucesso da inovação que promete ser a tecnologia de

Grade Computacional, através do estudo do problema de escalonamento de tarefas. Por fim, motivar novas pesquisas, uma vez que é através delas que se constrói e adquire o conhecimento.

1.2 Organização

O presente trabalho está dividido em sete capítulos e um apêndice. No Capítulo 2 é descrito o problema de escalonamento de tarefas concentrando-se no estudo do escalonamento estático, objeto deste estudo. O Capítulo 3 apresenta a técnica de *list scheduling* e algumas terminologias utilizadas na área de escalonamento. O Capítulo 4 descreve as três novas estratégias para o escalonamento sob o modelo *LogP*. A análise dos resultados obtidos está dividida em dois capítulos, no Capítulo 5 são apresentados os resultados dos testes em ambiente heterogêneo sob o modelo de Latência, e no Capítulo 6 os resultados dos testes em ambiente heterogêneo sob o modelo *LogP*. No Capítulo 7 são descritas as conclusões. Além disso, são identificados os estudos que podem ser desenvolvidos a partir desta dissertação. O Apêndice apresenta algumas estruturas de grafos acíclicos direcionados, representativos das aplicações paralelas, utilizadas nos testes.

Capítulo 2

O Problema de Escalonamento de Tarefas

Neste capítulo é definido o Problema de Escalonamento de Tarefas que consiste, basicamente, em designar as tarefas que compõem uma aplicação aos diversos processadores do sistema paralelo ordenando suas execuções. O procedimento da alocação das tarefas depende principalmente da estrutura da aplicação paralela, da arquitetura onde a aplicação será executada e das características relevantes ao desempenho do problema [47]. Por isso, inicialmente são descritos os modelos adotados para a representação da aplicação paralela e da arquitetura computacional onde essa aplicação será executada. Em seguida, o problema de escalonamento é apresentado dentro de uma possível classificação da literatura utilizada na área.

2.1 O Modelo Computacional de Escalonamento

Os modelos são muito úteis na área de computação por permitirem e facilitarem a análise e a previsão de desempenho dos algoritmos contribuindo assim para projetá-los eficientemente. Além disso, são capazes de promover a consistência e organização durante a fase do projeto de programação. O principal objetivo de mo-

delar o problema de escalonamento de tarefas é avaliar como uma aplicação será executada na máquina alvo, possibilitando assim, uma tomada de decisões que resultem na obtenção de melhor desempenho. Por isso, o modelo proposto é muitas vezes composto por outros modelos de forma a retratar todo o ambiente envolvido no processo de escalonamento. O ambiente a ser considerado diz respeito a dois universos fundamentais e distintos. O primeiro diz respeito às características relevantes ao desempenho da aplicação paralela em questão e o segundo ressalta às da máquina destino onde essa aplicação será executada. O modelo resultante é, então, composto por dois modelos, o da aplicação e o da arquitetura da máquina [7], conforme a Figura 2.1.

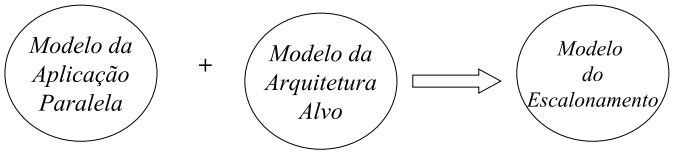

Figura 2.1: O Modelo de Escalonamento

2.1.1 Modelo da Aplicação Paralela

Neste estudo é considerada uma classe de aplicações paralelas que podem ser representadas por um Grafo Acíclico Direcionado (GAD) [47, 65]. No entanto, existem algumas desvantagens, principalmente a dificuldade de modelar ciclos. Isto porque, uma tarefa somente pode ser executada quando receber todos os dados de que precisa para iniciar, e em um ciclo não se observa esta condição. Na Figura 2.2 é apresentado um exemplo de um ciclo com três tarefas, é possível verificar que para v_0 começar a executar precisa receber os dados transmitidos por v_2 que depende dos dados transmitidos por v_1, e assim, não se consegue iniciar nenhuma das tarefas. Existem representações que tentam resolver o problema podem de grafos com ciclos e que podem ser encontradas em [69], mas não serão descritas por não fazerem parte deste estudo.

O GAD é denotado por G=(V,E,ϵ,w) onde V é o conjunto de n tarefas do grafo e E é o conjunto de arcos . A cada tarefa $v \in V$ é associado um peso de computação

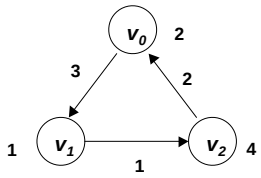

Figura 2.2: Grafo Cíclico com 3 tarefas

$\epsilon(v)$ representando a quantidade de trabalho da tarefa, esse valor multiplicado pelo fator de heterogeneidade do processador resulta no tempo de execução da tarefa; Cada arco $(v_i, v_j) \in E$ representa a dependência de dados entre essas duas tarefas. Assim, a tarefa v_i deve completar sua execução antes que a tarefa v_j comece, e mais, os dados transmitidos de v_i para v_j precisam estar disponíveis no processador destinado a executar v_j antes do início da sua execução. Um peso $w(v_i, v_j)$ pode ser associado ao arco representando a quantidade de dados a ser transmitida entre as tarefas v_i e v_j. Neste trabalho, uma tarefa é uma unidade de computação indivisível que pode ser uma instrução, uma subrotina ou um programa, que quando alocada a um processador executa até terminar sem interrupção.

O conjunto de predecessores imediatos de uma tarefa $v_j \in V$ é representado por:

$$pred(v_j) = \{v_i | (v_i, v_j) \in E\}$$

Da mesma forma, o conjunto dos sucessores imediatos de $v_i \in V$ é representado por:

$$succ(v_i) = \{v_j | (v_i, v_j) \in E\}$$

2.1.2 Modelo da Arquitetura Alvo

Este modelo define as características da arquitetura paralela onde é considerado que a memória é distribuída e assim, cada elemento de processamento é composto por um processador associado a sua própria memória local. A tais sistemas denominamos de multicomputadores ou sistemas fracamente acoplados nos quais as trocas de informações, entre os processadores, são por transmissão/recepção de mensagens,

através da rede de interconexão. Atualmente muitos pesquisadores concordam que quando o número de processadores, que compõem a máquina paralela, tende a ser alto, o desempenho do sistema pode ser melhor aproveitado, se a memória for distribuída ao invés de compartilhada [7]. A arquitetura com memória distribuída pode ser representada por um grafo não direcionado H =(P,C,h,l), onde P é o conjunto de k processadores e C é o conjunto de r interconexões heterogêneas entre os processadores [55]. Em um processo de escalonamento, o custo de computação ou tempo de execução de uma tarefa v em um processador p é dado pela multiplicação do peso de computação $\epsilon(v)$ da tarefa pelo fator de heterogeneidade h de p: $\epsilon(v) \times h(p)$. Convém ressaltar, que no modelo adotado por este trabalho cada processador do sistema heterogêneo têm fatores de heterogeneidade que representam a lentidão do processador em executar determinada tarefa. Como é abordado o escalonamento estático, inicialmente não se sabe a que processador será designada a tarefa, ou seja, aquele que lhe permitirá o menor tempo de término. Por essa razão, para calcular os atributos do grafo, como: nível, conível, que envolvem somatórios dos custos de execução das tarefas, e não dos pesos das tarefas, utiliza-se o fator médio de heterogeneidade do sistema para se determinar o custo de cada uma delas. O custo associado com a comunicação de $w(v_i, v_j)$ unidades de dados entre processadores que alojam tarefas adjacentes é definido conforme o modelo de comunicação adotado que se incorpora ao modelo da arquitetura.

2.1.3 Modelo de Comunicação

Com a crescente evolução das máquinas paralelas com memória distribuída, tornou-se necessário o desenvolvimento de novos modelos que representem as características dessas máquinas de forma mais precisa e realística [4, 5, 22, 72]. O ideal é serem suficientemente abstratos, para que detalhes da máquina sejam ignorados, mas ao mesmo tempo versáteis para permitir a portabilidade das aplicações desenvolvidas.

Na computação seqüêncial, após o desenvolvimento do modelo RAM - *Random Access Machine* surgiram diversos modelos, tais como: O modelo PRAM *Parallel*

Random Access Machine [30] muito utilizado para a análise da complexidade de algoritmos paralelos. Neste modelo é assumido um número de processadores ilimitado, trabalhando no modo síncrono que se comunicam através de uma memória compartilhada. Devido a esta última característica, custos de comunicação são desconsiderados [3]. É um modelo abstrato, e portanto algoritmos nele formulados apresentam na maioria dos casos desempenhos insatisfatórios quando executados em máquinas paralelas reais [22].

Posteriormente, na tentativa de desenvolver um modelo mais prático surgiu o Modelo *Bulk Synchrnous Parallel Model* (BSP) proposto por Valiant em 1990 [75], cujo objetivo principal é servir de modelo ponte entre as necessidades de *hardware* e *software* na computação paralela. O algoritmo BSP consiste de uma seqüência de superpassos, separados por barreiras de sincronização onde todas as comunicações são modeladas em conjunto, devido a essa característica é difícil projetar algoritmos paralelos para esse modelo [11]. Na mesma época surgiu o modelo de latência [59] e alguns anos depois, o modelo *LogP*, descritos adiante. Esses dois modelos foram escolhidos pora abordar o tema deste trabalho. O primeiro, por ser o mais utilizado nas pesquisas sobre escalonamento; o segundo por apresentar parâmetros considerados necessários e suficientes para representar *clusters* de computadores de forma mais realística.

Modelo de Latência

Atualmente é o modelo padrão de comunicação utilizado na área de escalonamento de tarefas. O único parâmetro arquitetural associado ao custo de comunicação é o atraso no tempo de transmissão, ou latência, para enviar uma unidade de dado no canal de comunicação entre dois processadores [59], denotada por L. Assim, um valor de latência $L(p, q)$ está associado ao segmento (p, q) representando a taxa de transmissão de dados entre os processadores p e q. Neste modelo, assume-se que um processador não gasta tempo preparando o envio ou recebimento de mensagens. Assim, comunicação e computação de tarefas podem se sobrepôr totalmente. Além disso um processador pode enviar várias mensagens distintas, simultaneamente, para destinos diferentes, o que se chama *multicast*. A Figura 2.3 apresenta um exemplo

do mapa de escalonamento de tarefas obtido a partir de um grafo GAD G sob o modelo latência, em uma arquitetura alvo composta por três processadores homogêneos, $h(p) = 1$. Observe que o custo de execução de cada tarefa unitária, $\epsilon(v) \times h(p)$, é igual a 1. O custo de comunicação entre v_1 e v_2 é dado por $w(v_1, v_2) \times L(P_1, P_2)$, também, igual a 1.

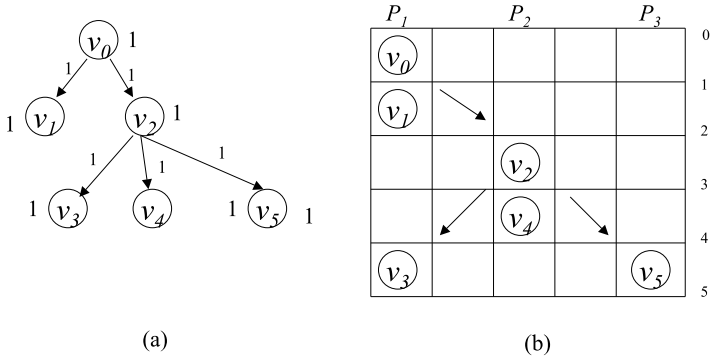

Figura 2.3: (a) GAD G unitário. (b) Mapa de Escalonamento no Modelo Latência em um sistema paralelo com três processadores homogêneos

Modelo *LogP*

Proposto por Culler *at al.* em 1993 [22], é um modelo assíncrono mais direcionado a rede de comunicação. Considera que as características relevantes que afetam o desempenho da aplicação podem ser resumidas em quatro parâmetros. Esses parâmetros permitem o cálculo preciso do custo de cada comunicação realizada entre dois processadores, são eles:

L - Latência: Custo de transmissão de uma unidade de dados entre dois processadores interconectados;

o - *overhead* ou sobrecarga: tempo que um processador gasta se preparando para receber ou enviar uma mensagem;

g - *gap*: intervalo mínimo entre duas sobrecargas de envio subseqüentes ou duas sobrecargas de recebimento consecutivas que ocorrem em um mesmo processador. Neste trabalho é assumido que $g \leq o$, ou seja, ao témino de uma sobrecarga (envio ou recebimento) já se pode iniciar a próxima sobrecarga;

P - Representa o número de processadores disponíveis.

Afim de se atingir maior precisão nos tempos de execução paralelos produzidos, neste trabalho são especificadas separadamente as sobrecargas de envio $o = O_s$ e de recebimento $o = O_r$. Esta suposição é adotada do modelo CLAUD (*Cost and Latency Augmented DAG*) [14]. A Figura 2.4 mostra um mapa de escalonamento S de um GAD sob o modelo *LogP*, onde as sobrecargas O_r e O_s são consideradas unitárias, mas isto não é uma condição.

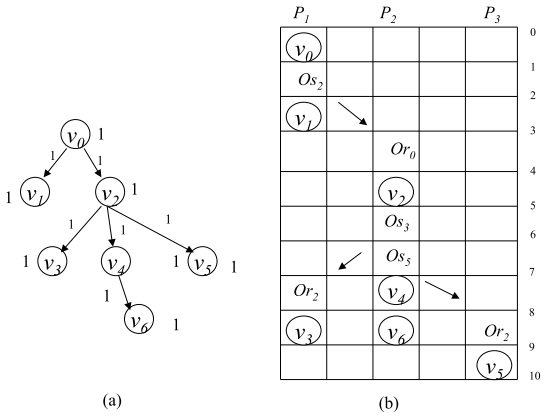

Figura 2.4: (a) GAD G unitário. (b) Mapa de Escalonamento no Modelo *LogP* em três processadores homogêneos

Outro modelo que surgiu, posterior ao *LogP* foi o *LogGP* que considera mais um parâmetro em adição aos já definidos no modelo *LogP*. Esse parâmetro é a largura de banda, representado por G (*Gap per byte*). Isto decorreu do fato de que em muitas máquinas paralelas atuais existe suporte para capturar a largura de banda do canal de comunicação, entre os processadores. Assim, através desse parâmetro é possível tratar a transmissão/recepção de mensagens longas, ultrapassando a limitação do modelo *LogP* em garantir o desempenho da comunicação apenas quando pequenas mensagens de tamanho fixo são enviadas [22]. Mais detalhes podem ser encontrados em [4].

2.2 O Problema do Escalonamento

O desempenho de uma aplicação paralela na máquina alvo depende diretamente do escalonamento *apropriado* das tarefas aos processadores. Isto significa conseguir uma eficiente alocação das tarefas, de forma que as relações de precedência entre

as tarefas sejam respeitadas e que o tempo de execução paralelo, também chamado de *makespan*, seja mínimo. Aliado a isso, o ideal é que seja utilizado um número de processadores razoavelmente baixo [45]. Diversos estudos propostos na literatura atestam que o problema do escalonamento é NP-Completo [74] em sua forma mais geral, e apenas para alguns poucos casos restritos são conhecidas soluções ótimas em tempo polinomial [18, 27], limitando com isso o uso exclusivo de métodos exatos. Por esta razão, métodos aproximados ou heurísticos são comumente utilizados para tentar achar escalonamentos com tempos de conclusão quase ótimos.

2.2.1 Escalonamento de Tarefas

O escalonamento de tarefas inicialmente pode ser classificado de acordo com o número de processadores disponíveis na palataforma distribuída. Quando a atribuição das tarefas de um programa paralelo é feita a um único processador, o escalonamento é dito local. No caso onde as tarefas são designadas à vários processadores, o escalonamento é denominado global. A Figura 2.5 apresenta uma visão simplificada da taxomomia proposta por Casavant e Kuhl [13].

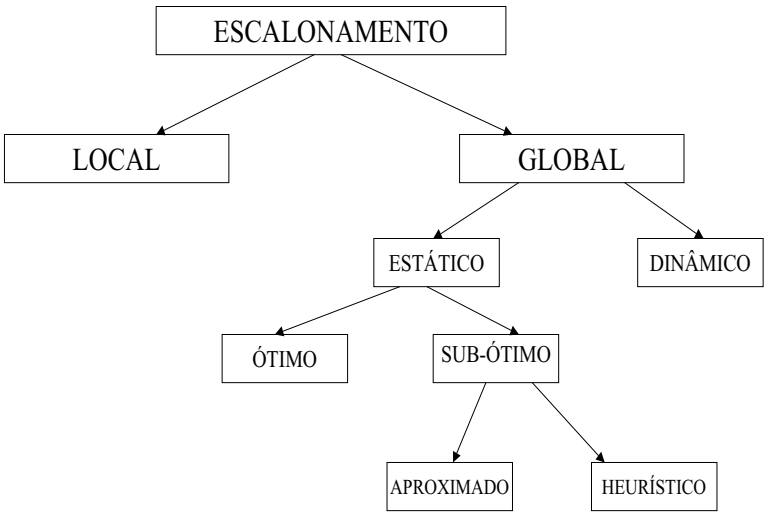

Figura 2.5: Taxonomia de Casavant e Kuhl

Nessa classificação o escalonamento global, pode ser dividido em dinâmico e estático, de acordo com o momento em que as decisões para a alocação das tarefas é feita. No escalonamento dinâmico as decisões de alocação se realizam durante a

execução do programa, sendo tipicamente usado quando a topologia do grafo, os custos de comunicação e de computação não podem ser totalmente conhecidos em tempo de compilação do programa. Este estudo se concentra no escalonamento estático das tarefas. Assim, as informações relativas ao tempo de execução de cada tarefa, ou ao menos, uma boa aproximação, bem como as suas relações de precedência são previamente especificadas. O grafo de tarefas, ou GAD, que representa a aplicação ou programa é analisado pelo escalonador e cada tarefa é designada a um processador antes da execução de qualquer tarefa da respectiva aplicação.

Um escalonamento S é um conjunto finito de tuplas (v_i,p_j,t_l), cada um especificando que uma tarefa $v_i \in V$ é executada no processador $p_j \in P$ no tempo t_l. Assim sendo, cada tarefa $v_i \in V$ deve ser escalonada em um processador $p_j \in P$ e essa tarefa só pode ser executada depois que todos os seus predecessores imediatos tenham completado a sua execução e que os respectivos dados necessários para o início da sua execução estejam disponíveis no processador p_j. Com isso, uma tarefa de computação não pode enviar qualquer resultado de seus dados, antes de ter completado. Além disso, uma mensagem de transferência de dados não pode ser recebida a menos que tenha sido enviada. Existe, portanto uma demora de pelo menos L, que é a latência, entre uma sobrecarga de envio e a respectiva sobrecarga de recebimento.

Uma vez definidos os modelos, da aplicação e da arquitetura, e estabelecidos os critérios sob o qual se realizará o escalonamento, é possível determinar, através da heurística desenvolvida, o tempo de execução paralelo final da aplicação, o *Schedule Length* ou *Makespan* que é dado pela seguinte expressão:

$$SL = max\,\{T_{inicial}(v_i, p_j) + \epsilon(v_i) \times h(p_j)\}$$

Sendo $T_{inicial}$ (v_i,p_j) o tempo no qual uma tarefa v_i inicia a sua execução em um processador p_j. O termo $\epsilon(v_i)$ é o peso de execução dessa tarefa em p_j e $h(p_j)$ representa o fator de heterogeneidade do processador p_j. O termo $\epsilon(v_i) \times h(p_j)$ representa o custo de execução da tarefa v_i em p_j.

2.2.2 Escalonamento Estático de Tarefas

O escalonamento estático, pode ser classificado como ótimo ou subótimo. Praticamente, em quase todos os problemas de escalonamento de tarefas, não existem soluções capazes de gerar escalonamentos em tempo polinomial que sejam proporcionais ao tamanho do problema. Apenas para três casos especiais são conhecidas soluções, ditas ótimas, nos quais os custos de comunicação entre as tarefas são considerados nulos:

1. Escalonamneto de grafos de tarefas estruturados em árvores com custos de computação uniforme em número arbitrário de processadores [17];
2. Escalonamento de grafos de tarefas com custos de computação uniforme em dois processadores [18]; e
3. Escalonamento de grafos de tarefas *interval-ordered* [29] com custos de computação uniforme em um número de arbitrário de processadores [58].

Com isso, devido a complexidade do problema, soluções subótimas são consideradas para se obter respostas em tempos inferiories ao exponencial. Dentro da categoria de subótimos, o escalonamento pode ser classificado em aproximados e heurísticos. O escalonamento aproximado ocorre quando o algoritmo possui limite de pior caso conhecido. O escalonamento heurístico, por sua vez, utiliza critérios identificados empiricamente ou intuitivamente, como responsáveis por algum fator de desempenho no escalonamento [70].

A principal vantagem do escalonamento estático é a ausência total de sobrecarga para determinar o escalonamento em tempo de execução. Em contra partida, as desvantagens são por conta da falta de adequabilidade às plataformas multiusuárias, devido a exigência de um alto conhecimento prévio do comportamento da aplicação em relação a plataforma usada [70]. Ainda mais, devido a ineficiência dos mecanismos que estimam os custos de computação e comunicação, o que acarreta sérias degradações no desempenho esperado [67].

2.2.3 Escalonamento Dinâmico de Tarefas

Realizado em tempo de execução da aplicação, uma vez que poucas concepções podem ser feitas em tempo de compilação. Dessa forma, as decisões a cerca do escalonamento precisam ser tomadas durante os passos de alocação das tarefas, ditas decisões *on the fly*. A meta principal, além de procurar minimizar o tempo de conclusão da aplicação, consiste em minimizar as sobrecargas causadas por essas decisões inesperadas necessárias. Tais sobrecargas representam uma das desvatagens mais significativas, devido ao aumento que acarretam no custo computacional da heurística. As vantagens estão por conta da possibilidade de alterações dinâmicas na estrutura computacional.

2.3 Técnicas de Escalonamento Estático

Os algoritmos de escalonamento estático podem ser classificados em duas grandes categorias [73]. Na primeira encontram-se as *Heurísticas de Construção*, foco deste trabalho, que utilizando um modelo para a aplicação, geralmente um *GAD* e outro modelo para a arquitetura, a partir desses modelos constrõem a cada passo um único escalonamento como resposta para uma determinada entrada. Na outra categoria encontram-se as *Heurísticas de Construção e Busca*, nas quais vários escalonamentos são criados durante a execução do algoritmo na busca da melhor solução final.

2.3.1 Heurísticas de Construção

As heurísticas de construção, definem somente um único escalonamento. Heurísticas desta categoria, geralmente, são utilizadas quando o tempo de construção de um escalonamento é considerado um fator crucial e de grande importância ao objetivo final. Normalmente apresentam complexidade mais baixa que as heurísticas de construção e busca. As heurísticas de construção odem ser divididas em quatro grupos: heurísticas *List Scheduling*, heurísticas de aglomeração de tarefas, heurísticas de análise de caminho crítico do grafo e heurísticas de particionamento

do grafo.

2.3.2 Heurísticas *List Scheduling*

Nas heurísticas *List Scheduling* a idéia básica é construir duas listas, uma de tarefas livres ordenadas de acordo com alguma prioridade pré-estabelecida; e outra de processadores disponíveis. A cada passo da heurística é selecionada a tarefa da lista de tarefas livres de mais alta prioridade e o processador favorito para executar a tarefa candidata. Uma tarefa é dita livre se todos os seus predecessores imediatos já foram escalonados. Um processador é dito disponível se não está executando nenhuma tarefa e nem está bloqueado preparando o envio ou o recebimento de uma mensagem. O processador favorito é aquele que permite minimizar uma função pré-determinada de custo. Por exemplo, em um ambiente homogêneo é comum minimizar a função de custo dada pelo tempo de início de uma tarefa. Em contra partida, em um ambiente heterogêneo é aconselhável minimizar a função de custo dada pelo tempo de término de uma tarefa. Isto porque, em tal ambiente nem sempre o processador que permite a uma tarefa começar o mais cedo possível lhe permitirá, também, terminar o mais breve possível (pode ser o mais lento no sistema). Exemplos baseados na técnica de *List Scheduling* podem ser encontrados em [39, 50, 56, 73, 61]. A principal diferença dos algoritmos pertencentes a este grupo é referente ao cálculo das prioridades para a ordenação das tarefas na lista de tarefas livres. Se o cálculo das prioridades é feito antes do escalonamento da primeira tarefa do grafo, é dito cálculo estático e o algoritmo denominado de *List Scheduling Static Priorities*, ou seja, com prioridade estática. Caso seja possível reordenar a lista de tarefas livres durante o escalonamento do grafo, recalculando as prioridades, o algoritmo é denominado de *List Scheduling Dynamic Priorities* por apresentar as prioridades calculadas de forma dinâmica.

2.3.3 Heurísticas de Aglomeração de Tarefas

Nas heurísticas baseadas em aglomeração são criados conjuntos ou coleções de tarefas que devem ser executadas em um mesmo processador. O objetivo é minimizar o *makespan* da aplicação pela eliminação das comunicações existentes entre elas, uma vez que tarefas adjacentes designadas ao mesmo processador têm custo de comunicação nulo. A cada iteração uma tarefa pode ser adicionada a um determinado conjunto, que normalmente contém predecessores da respectiva tarefa. Algoritmos deste grupo tendem a ter maior complexidade e trabalham em dois estágios distintos; no primeiro consideram um número infinito de processadores homogêneos e são formados os conjuntos de tarefas sendo que cada um deles é alocado a um processador virtual. A seguir, no segundo estágio, as coleçoes de tarefas existentes em cada processador virtual são mapeadas aos processadores reais disponíveis na arquitetura utilizada. Exemplos deste grupo podem ser encontrados em [15, 53, 57, 71].

2.3.4 Heurísticas de Análise de Caminho Crítico

As heurísticas de análise de caminho crítico utilizam a abordagem de tentar diminuir esse caminho, que é aquele com maior custo que se pode percorrer em um grafo, tentando agrupar tarefas a ele pertencentes a um mesmo processador. Entretanto, nem sempre isso é possível. A idéia consiste em procurar anular os custos de comunicação entre as tarefas adjacentes quando escalonadas no mesmo processador. Exemplos baseados nessa metodologia podem ser encontrados em [34, 42, 45, 76].

2.3.5 Heurísticas de Particionamento de Grafos

A estratégia que utiliza o particionamento de grafos [24, 41, 51], como o próprio nome indica, divide o grafo em várias partes, denominadas clãs, com o objetivo, também, de minimizar arestas que conectam vértices de diferentes partições.

2.3.6 Heurísticas de Construção e Busca

Um aspecto importante de algoritmos pertencentes a esta categoria é que apesar de consumirem um tempo mais significativo para fornecer a solução final, os escalonamentos resultantes podem apresentar melhor qualidade que os produzidos pelas heurísticas de construção, acima descritas. Isto porque as heurísticas de construção e busca podem gerar diversas soluções durante a sua execução com a finalidade de, a cada passo, melhorar cada vez mais a solução resultante da iteração anterior. Dentro desta categoria podemos citar os métodos de busca de propósito geral que objetivam encontrar uma boa solução através da aplicação de heurísticas modeladas para um determinado problema, conhecidas como metaheurísticas. Por serem de caráter geral as metaheurísticas possuem mecanismos que permitem percorrer o espaço de busca do problema escapando dos mínimos locais. São exemplos clássicos de metaheurísticas: Algoritmos Genéticos [20, 38], *Simulated Annealing* [43], Busca Tabu [35, 36], GRASP [26] e *Variable Neighborhood Search* (VNS) [37].

Existem, ainda, outras técnicas fora das cateorias apresentadas que podem ser utilizadas em conjunto com as heurísticas descritas, como por exemplo: Replicação de Tarefas, e *Lookahead*.

2.4 Heurísticas de Replicação

Na Replicação de tarefas o objetivo é reduzir o custo de comunicação entre os processadores, através da colocação de cópias da mesma tarefa em processadores distintos. Com isso, torna-se bastante vantajosa quando existem custos elevados de comunicação e tarefas com alto número de sucessores imediatos. A desvantagem fica por conta do aumento da complexidade observado nas heurísticas formuladas nesta técnica. A Figura 2.6 ilustra o escalonamento de um grafo de 5 tarefas em uma arquitetura com 3 processadores homogêneos. Observe como é possível reduzir o *makespan* da aplicação com o uso desta técnica. Exemplos de heurísticas utilizando esta técnica são: *Duplication Scheduling Heuristics* (DSH) [44]; *Bottom-up Top-down Duplication Heuristcs* (BTDH) [16]; *Economic Critical Path Fast Duplication*

(ECPFD) [2]; e *Critical Path Fast Duplication* (CPFD) [2].

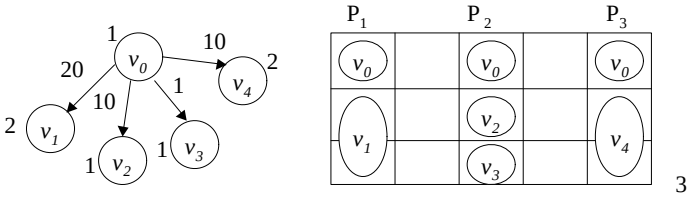

Figura 2.6: Exemplo de GAD escalonado em 3 processadores homogêneos com replicação de tarefas

2.5 Heurísticas com *Lookahead*

A técnica de *Lookahead* baseia-se na observação de que nem sempre para minimizar a função de custo do escalonamento, o processador favorito é aquele que oferece disponibilidade no menor tempo possível. Assim, essa estratégia busca escalonar tarefas considerando tarefas ainda não livres (com dependências não satisfeitas) [45]; ou busca atrasar o início de uma tarefa, alocando-a a um processador que já possui algum de seus predecessores [25]. A Figura 2.7 ilustra um exemplo, retirado dessa última referência. Observe que os processadores são homogêneos, a esquerda está o escalonamento onde não se utilizou *lookahead* e, assim, devido ao custo de comunicação o *makespan* é igual a 14. A direita está o escalonamento ótimo de 3 unidades, obtido por ter atrasado o início da tarefa v_1, colocando-a no processador P_1. Nesse processador a tarefa pode começar no tempo 1, ao invés de começar no tempo 0 conforme oferecia o processador P_2. O critério de ordenação da lista de tarefas livres é o conível da tarefa, ou seja, o tempo mais breve que pode começar a ser executada.

As técnicas de replicação e *lookahead* podem e devem ser usadas em conjunto com as outras técnicas contribuindo, assim, para alcançar melhores resultados no *makespan* do escalonamento. Um exemplo de *list scheduling* com *lookahead* pode ser encontrado em [25] que apresentou a heurística *List Scheduling para Grids* (LSG). Um exemplo da técnica de *list scheduling* com replicação pode ser visto em [23] denominado *Levelized Duplication Based Scheduling* (LDBS). Heurísticas baseadas em

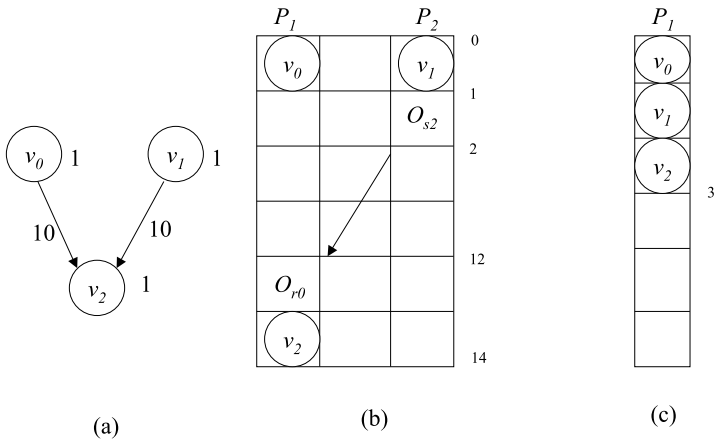

Figura 2.7: Exemplo da técnica de *lookahead* (a) GAD G (b) Escalonamento sem *lookahead* (c) Escalonamento com *lookahead*

caminho crítico com *lookahead* uma das mais conhecidas é a *Dynamic Critical Path* (DCP) [45]. Heurísticas de aglomeração de tarefas que incorporam a possibilidade de replicar tarefas consideradas críticas podem ser encontrados em: PLW [57], HAL [53].

2.6 A Técnica Escolhida

Este estudo inicial possibilitou perceber a variedade de caminhos que podem ser adotados para se especificar um algoritmo de escalonamento estático. Dentre as diversas técnicas apresentadas uma delas foi escolhida para o desenvolvimento da abordagem proposta por esta dissertação. O objetivo principal do presente estudo é formular uma heurística de escalonamento estático, baseada no modelo *LogP*, para ambientes com características semelhantes às grades computacionais. Nessa plataforma de alta latência, a comunicação para a transmissão de dados (sempre que tarefas adjacentes forem designadas a processadores distintos) acarreta sobrecargas ao sistema. Portanto, o objetivo principal da nova heurística é avaliar, através de versões, políticas alternativas para minimizar os efeitos adversos causados por essas sobrecargas no sistema.

As grades computacionais são ambientes de recursos em número limitado, tipicamente dinâmicos e heterogêneos. Assim, é importante visar considerações relevantes

para o desempenho da estratégia, entre elas: utilizar uma técnica prática que promova resultados de *makespan* aceitáveis com complexidade mais atrativa; prever um número limitado de processadores, em geral, totalmente concectados; e ser facilmente adaptável a ambientes duais compostos por processadores homogêneos e heterogêneos [40, 73]. Essas características são favoráveis a função de um escalonador estático numa grade computacional. Essa função refere-se a realização de um pré-escalonamento de tarefas, afim de amenizar a carga de trabalho do escalonador dinâmico. Um escalonador dinâmico é necessário devido à natureza instável e dinâmica dos recursos da plataforma. Assim, é necessário que as decisões de escalonamento sejam feitas durante a execução da aplicação. Porém, incorporando-se ao processo uma prévia etapa, através do escalonador estático, restará ao dinâmico, apenas, os ajustes finais momentâneos.

Por essas razões, este estudo adotou a técnica de *list scheduling* que em comparação a todas as demais, metaheurísticas e heurísticas, é a de mais baixa complexidade, suporta ambientes homogêneos e heterogêneos e em número limitado. As concorrentes do grupo, aglomeração, caminho crítico, particionamento e replicação são, geralmente, formuladas para um número ilimitado de processadores homogêneos e geralmente apresentam valores de complexidade mais altos. Ainda mais, devido aos poucos trabalhos disponibilizados na literatura torna-se difícil prever como seriam os resultados da adaptação dessas técnicas a ambientes heterogêneos.

2.7 Tabelas de Classificação de Heurísticas de Construção

Nesta seção são apresentadas tabelas de classificação de heurísticas de Construção, objeto desta pesquisa, com alguns algoritmos propostos na literatura. A finalidade é catalogar os algoritmos com as referências onde podem ser encontrados e apresentar a ordem de complexidade de cada um deles. Nas expressões da complexidade o parâmetro P indica o número de processadores da arquitetura disponibilizada, o parâmetro E indica o conjunto dos arcos e o parâmetro V o número

de tarefas do grafo utilizado como entrada. A Tabela 2.1 mostra algoritmos *List Scheduling* agrupados de acordo como é feito o cálculo da prioridade. Realizada de forma estática *List Scheduling Statics Priorities* (LSSP) e realizada de forma dinâmica *List Scheduling Dynamic Priorities* (LSDP).

LSSP - prioridade estática, P Limitado, sem replicação	**Complexidade**
HLFET (*High Levels First with Estimated Times*) de Adam [1]	$O(Vlog(V)+(E+V)P)$
CPND (*Critical Path Node Dominant*) de Kwok *et al.* [48]	$O(Vlog(V)+(E+V)P)$
DPS (*Decisive Path Scheduling*) de Park *et al.* [60]	$O(Vlog(V)+(E+V)P)$
MCP (*Modified Critical Path*) de Wu e Gajski [76]	$O(V^2(log(V)+P))$
CPM (*Critical Path Method*) de Park *et al.* [60]	$O(V(log(V)+P))$
WL (*Weighted Length Scheduling*) de Yang e Fu [77]	$O(V(log(V)+P))$
LSDP - prioridade dinâmica, P Limitado, sem replicação	**Complexidade**
ETF (*Earliest Task First*) de Hwang *et al.* [39]	$O(V(E+V)P)$
ERT (*Earliest Ready Task*) de Lee *et al.* [49]	$O(V(E+V)P)$
DLS (*Dynamic Level Scheduling*) de Sih *et al.* [68]	$O(V(E+V)P)$
LSDP - prioridade dinâmica, P Ilimitado, sem replicação	**Complexidade**
DCP (*Dynamic Critical Path*) de Kwok e Ahmad [45]	$O(V(E+V)P)$

Tabela 2.1: Alguns Algoritmos de *List Scheduling*

Exemplos de algoritmos baseados na técnica de Aglomeração são apresentados na Tabela 2.2 a seguir:

Aglomeração, P Ilimitado, sem replicação	**Complexidade**
DSC (*Dynamic Sequence Clustering*) de Yang e Gerasoulis [71]	$O((E+V)log(v))$
EZ (*Edge Zeroing*) de Sarkar [65]	$O(E(E+V)log(V))$
TCS (*Task Clustering and Scheduling*) de Palis *et al.* [57]	$O(V(Vlog(V)+E))$
LC (*Linear Clustering*) de Kim e Browne [42]	$O(V^2log(V))$

Tabela 2.2: Alguns Algoritmos de *Aglomeração*.

Exemplos de algoritmos baseados na técnica de Replicação são apresentados na Tabela 2.3 a seguir:

Replicação, P Limitado	**Complexidade**
DSH (*Duplication Scheduling Heuristics*) de Kruatrachue [44]	$O(V^4)$
BTDH (*Bottom-up Top-down Duplication Heuristcs*) de Chung [16]	$O(V^4)$
ECPFD (*Economic Critical Path Fast Duplication*) de Ahmad [2]	$O(V \times E \times P)$
Replicação, P Ilimitado	**Complexidade**
CPFD (*Critical Path Fast Duplication*) de Ahmad [2]	$O(E \times V^2)$

Tabela 2.3: Heurísticas de *Replicação*.

A Figura 2.4 apresenta mais alguns algoritmos.

Número Limitado de Processadores, sem replicação	**Complexidade**
ISH (*Insertion Scheduling Heuristic*) de Kruatrachue [44]	$O(Vlog(V)+(E+V)P)$
GD (*Graph Decomposition*) de Khan *et al.* [41]	$O(V^3)$
P **Limitado Heterogêneos, com replicação**	**Complexidade**
LDBS (*Levelized Duplication Based Scheduling*) de Dogan [23]	$O(\|V\|^3\|E\|\|M\|^2)$
P **Limitado Heterogêneos**	**Complexidade**
BIL (*Best Imaginary Level Scheduling*) de Oh e Soohoi [56]	$O(V^2 \times P \times logP)$
HEFT *(Heterogeneous Earliest Task First)* de Topcuoglu [73]	$O(V^2 \times P)$

Tabela 2.4: Alguns algoritmos para número limitado de processadores, outros.

2.8 Resumo

O objetivo deste capítulo foi apresentar o problema de escalonamento de tarefas baseado em heurísticas. Inicialmente foram descritos os modelos da aplicação e da arquitetura utilizados para a formulação do problema. Em seguida, foi apresentada uma possível classificação de heurísticas de escalonamento estático ressaltando algumas de suas características principais. O próximo capítulo apresenta a técnica de *List Scheduling* escolhida para abordar o tema proposto e descreve algumas terminologias utilizadas neste trabalho.

Capítulo 3

A Técnica de *List Scheduling*

Este capítulo é composto por duas seções, na primeira é apresentada a técnica *List Scheduling* e na segunda algumas terminologias da área que fazem parte deste trabalho.

3.1 A Estratégia *List Scheduling*

Pode ser considerada como uma técnica base de escalonamento estático, foi proposta por Hwang *et al.* em 1989 [39] e é uma das técnicas mais populares segundo a literatura atual. A razão principal é devida ao fator *baixo custo versus bons resultados* que apresenta. A idéia básica consiste em atribuir prioridades às tarefas do grafo criando uma lista ordenada de tarefas cujos predecessores imediatos já foram escalonados, ditas livres. Além disso, também, é construída uma lista de processadores disponíveis. Como apresentado no capítulo 2, as heurísticas de *List Scheduling* são chamadas de construção por construirem, sucessivamente a cada iteração, um único escalonamento final [1, 17, 68]. A metodologia empregada consiste em a cada passo do algoritmo determinar o conjunto das tarefas livres e o conjunto dos processadores disponíveis. Assim, duas fases distintas são observadas durante a construção do escalonamento. Na primeira fase, chamada de *Seleção-Tarefa*, é selecionada a tarefa

livre de mais alta prioridade. Na segunda fase, *Seleção-Processador* é feita a escolha do processador, mais adequado para minimizar uma função de custo pré-definida. Assim, a escolha para o escalonamento baseia-se no par *tarefa-processador*.

A maior diferença entre os algoritmos formulados nessa estratégia, é a definição da prioridade a ser estabelecida para escolher a próxima tarefa livre a ser executada [8]. Existem diversas possibilidades para a definição das prioridades a serem atribuídas as tarefas, e devido a essa diversidade podemos encontrar diferentes algoritmos *List Scheduling* na literatura, mas a estrutura base parte do mesmo princípio listada no algoritmo da Figura 3.1. Nesse exemplo, a função de custo que se deseja minimizar é o tempo de início da tarefa candidata ao escalonamento. Isto porque as heurísticas de *List Scheduling* foram inicialmente formuladas para ambientes homogêneos. A adpatação para ambientes heterogêneos é bastante simples, consite em alterar a linha 5, da referida figura, para selecionar o processador que permite a tarefa candidata terminar o mais cedo possível.

Algoritmo 1 :*List Scheduling*

```
1  Definição da prioridade;
2  Criação de uma lista de tarefas livres (lista_de_livres);
3  enquanto lista_de_livres ≠ vazia faça
4     Selecione a tarefa livre de maior prioridade;
5     Selecione um processador disponível onde a tarefa pode começar
      mais cedo;
6     Escalone a tarefa nesse processador;
7     Determine as novas tarefas livres e insira na lista_de_livres;
8  fim enquanto;
```

Figura 3.1: Algoritmo *List Scheduling*.

As maioria das heurísticas de *List Scheduling* são classificadas, por alguns autores, como heurísticas voltadas para ambientes computacionais com um número limitado de processadores, homogêneos, totalmente conectados. De acordo com a pesquisa publicada em [61] as heurísticas de *list scheduling* utilizadas em sistemas de memória distribuída, geralmente são mais práticas e simples. Dessa forma, conseguem produzir resultados de *makespan* mais atrativos em termos de complexidade

que as heurísticas concorrentes de outros grupos.

3.1.1 Classificações da Técnica

A técnica de *List Scheduling* pode ser classificada em duas categorias, dependendo de como é computada a prioridade das tarefas [61]: Algoritmos *List Scheduling Static Priorities* (*LSSP*), com prioridades estáticas, nos quais as prioridades estabelecidas para as tarefas são calculadas antes do início do escalonamento da primeira tarefa e uma vez construída a lista de prioridades não poderá mais ser alterada; e algoritmos *List Scheduling Dynamic Priorities LSDP*, com prioridades dinâmicas, nos quais a lista de tarefas é recalculada a cada passo do algoritmo, sendo possível capturar mais precisamente alterações nas prioridades das tarefas e com isso reordená-las na lista de tarefas livres. A reordenação permite ressaltar a real importância das tarefas no passo do escalonamento.

São exemplos desta técnica: a ETF (*Earliest Task First*) proposta por Hwang *et al.* em 1989 [39] de complexidade $O(V(E+V)P)$ é considerado um dos melhores algoritmos baseado na técnica de *List Scheduling* formulado para o modelo latência e para um número limitado de processadores homogêneos. A metodologia empregada consiste em computar, tentativamente, a cada passo do algoritmo, o tempo mais breve de início de cada tarefa livre em cada processador disponível; a HLFET (*Highest Level First with Estimated Times*) [1], de complexidade $O(Vlog(V)+(E+V)P)$, formulada para o modelo de Latência e para ambientes de processadores homogêneos. A prioridade para a escolha da tarefa é o atributo de nível estático (onde não se considera custos de comunicação), e para a seleção do processador é escolhido aquele que permite minimizar o tempo de início da tarefa selecionada; MCP (*Modified Critical Path*) [76], de complexidade $O(V^2(log(V)+P))$, utiliza o atributo $ALAP$ *As Late As Possible* para priorizar as tarefas ordenando a lista de prioridades em modo ascendente. Desempates são feitos preferindo a tarefa sucessora imediata de menor $ALAP$. Quanto ao processador é escolhido o que permitir menor tempo de início para a tarefa. Incorpora a possibilidade de encaixar tarefas entre espaços existentes entre duas tarefas já escalonadas. A essa técnica de otimi-

zação denominamos de Inserção; Outros exemplos de algoritmos incluem: a DPS (*Decisive Path Scheduling*) [60], de complexidade $O(Vlog(V)+(E+V)P)$; a CPND (*Critical Path Node Dominant*) [48] de complexidade $O(Vlog(V)+(E+V)P)$; a CPM (*Critical Path Method*) [60] de complexidade $O(V(log(V)+P))$; a WL (*Weighted Length*) [77] de complexidade $O(V(log(V)+P))$; e a ERT (*Earliest Ready task*) [49] de complexidade $O(V(E+V)P)$.

3.2 Terminologias da Área de Escalonamento

Neste trabalho são estudados algoritmos construídos sob a técnica *list scheduling* onde as tarefas são escalonadas em ordem determinada por uma prioridade pré-estabelecida. Por esta razão, existem diversos algoritmos diferindo apenas pela prioridade escolhida para ordenar a lista. As mudanças de prioridades podem resultar em diferentes escalonamentos já que as tarefas serão escalonadas em ordens diferentes. Dois problemas são verificados nas heurísticas baseadas nessa técnica. O primeiro é devido as prioridades das tarefas ordenadas na lista, nem sempre representarem a real importância da tarefa naquele determinado momento da decisão para o escalonamento [45]. Isto porque, uma vez que a lista de tarefas livres é construída antes do início do escalonamento não poderá mais ser reformulada durante as iterações do escalonamento. O segundo problema observado é que existem casos com mais de uma tarefa candidata ao escalonamento por apresentarem a mesma prioridade. Neste trabalho, a estratégia adotada para contornar o primeiro problema consiste em, além de calcular as prioridades de forma estática, ou seja, antes de escalonar a primeira do grafo, também, recalculá-las a cada passo do algoritmo, caracterizando uma abordagem dinâmica. A forma dinâmica objetiva uma maior precisão, uma vez que os valores recalculados capturam melhor as variações ocorridas. O segundo problema pode ser amenizado incorporando-se ao processo critérios de desempate que garantam a melhor decisão para o escalonamento. Assim, estão previstas duas etapas para o desempate das tarefas. Na primeira etapa de desempate escolhe-se, por exemplo, a que tiver maior atributo de conível, caso persista o empate, novo critério é empregado: escolher a de maior nível. Os atributos escolhi-

dos, neste trabalho e descitos adiante, para priorizar as tarefas são *nível, conível, caminho crítico e ALAP*, calculados de duas formas estática e dinâmica. No total são oito possibilidades de escolha e dois critérios de desempate para a heurística proposta. Outras definições importantes, neste estudo, são as de granulosidade ou granularidade e função de custo.

3.2.1 Granulosidade ou Granularidade

Aplicações paralelas, usualmente, podem ser classificadas como aplicações de *computação intensiva* ou de *comunicação intensiva* dependendo da quantidade relativa de computação ou comunicação presente. Tradicionalmente o termo granulosidade ou granularidade (como muitos pesquisadores preferem utilizar) é utilizado para capturar o grau da quantidade relativa de computação ou comunicação existente, e é calculado como a razão entre os custos de computação e os custos de comunicação do grafo [46, 69, 71]. Quando uma aplicação tem custo de computação superior ao de comunicação, diz-se que apresenta granulosidade grossa. As aplicações dessa categoria podem efetivamente utilizar processadores interconectados por redes de longa distância, sem sofrerem grande perda de desempenho. Em contrapartida, quando os custos de comunicação são elevados em relação ao custos de computação, classifica-se a aplicação como de granulosidade fina. Tais aplicações, geralmente, demandam plataformas computacionais dedicadas, como os supercomputadores massivamente paralelos, para que as sobrecargas impostas pelas comunicações não anule os ganhos decorrentes da paralelização. O conceito de granulosidade é muito importante pois estabelece até que ponto é vantajoso ou não paralelizar uma aplicação [8]. Existem diversas formulações para a granulosidade de uma aplicação paralela. Entretanto, a maioria é proposta para ambientes de processadores homogêneos. Em ambientes heterogêneos, o fator de heterogeneidade $h(p)$ de cada processador influi no escalonamento do grafo, uma vez que o tempo de execução da tarefa é função do processador ao qual será designada. No processo de escalonamento estático, os custos das tarefas são calculados com base na média do tempo de execução em cada processador do sistema. Essa média é calculada através das seguintes fórmulas, que também

serão utilizadas nas próximas subseções quando da definição de nível, conível, entre outras:

(a) Fator de Heterogeneidade médio:

$$\overline{h(p)} = \frac{\sum_{\forall p_j \in P} h(p_j)}{|p|}$$

(b) Latência média:

$$\overline{L} = \frac{\sum_{\forall p_i, p_j \in P} L(p_i, p_j)}{|p|^2}$$

A definição de granulosidade adaptada para ambientes heterogêneos é dada por:

$$granulosidade(G) = \frac{\sum_{\forall v_i \in V} \epsilon(v_i) \times \overline{h(p)}}{\sum_{\forall (v_i, v_j) \in E} (w(v_i, v_j) \times \overline{L})}$$

3.2.2 Nível

O Nível de uma tarefa, também, denominado de *b-level* ou *bottom-level* é o comprimento mais longo, ou caminho de maior custo, começando na tarefa em questão até uma tarefa de saída ou finalizadora do grafo. Uma tarefa de saída é aquela que não possui sucessores. O cálculo é feito somando-se os custos de computação das tarefas e os custos de comunicação dos arcos pertencentes a esse caminho. Durante o processo de escalonamento o nível é usualmente constante até que todos as tarefas tenham sido escalonadas. O nível de uma tarefa é dado pela seguinte expressão, adaptada para ambientes heterogêneos:

$$nivel(v_i) = \epsilon(v_i) \times \overline{h(p)} + \max_{v_q \in Succ(v_i)} \{w(v_i, v_q) \times \overline{L} + nivel(v_q)\}$$

Onde $\epsilon(v_i)$ é o peso da tarefa v_i que multiplicado ao fator de heterogeneidade médio dos processadores disponíveis fornece o custo da tarefa v_i; $w(v_i, v_q)$ é o peso do arco entre as tarefas adjacentes v_i e v_q que multiplicado ao valor da latência média do canal $\overline{L}$, que interliga os processadores distintos que alojaram as tarefas v_i e v_q, fornece o custo da comunicação. Observe que no cálculo do nível em um processo

de escalonamento envolve cálculos referentes aos custos de execução das tarefas e não apenas dos pesos das tarefas dados no grafo. Em ambientes homogêneos onde os fatores de heterogeneidade são todos iguais a 1, o custo de execução de cada tarefa é igual ao seu peso. Por outro lado, em ambientes heterogêneos existem processadores com diferentes fatores de heterogeneidade, por isso é utilizado o fator de heterogeneidade médio $\overline{h(p)}$. Isto porque, o atributo nível, como qualquer outro atributo descrito nas próximas seções, é utilizado para a ordenação das tarefas na lista de tarefas livres candidatas ao escalonamento. Assim, durante a elaboração da lista de tarefas livres ainda não se sabe a que processador serão designadas (aquele que lhe permitirá terminar mais cedo) por isso utiliza-se o fator médio de heterogeneidade. Devido a essa forma de calcular o atributo prioritário para ordenar a lista de tarefas livres, considerando o fator de heterogeneidade médio, inúmeros empates de tarefas com mesma prioridade são observados.

O atributo nível é muito utilizado para priorizar tarefas do grafo em um processo de escalonamento. Tarefas de maior nível são aquelas seguidas pelas maiores cadeias de tarefas no grafo, e portanto, indicadas para serem as mais prioritárias. A lista de tarefas é ordenada de acôrdo com valores decrescentes do nível; assim, quanto maior o valor do nível maior será a prioridade da tarefa.

3.2.3 Conível ou ASAP (*As Soon As Possible*)

O Conivel, também chamado de *t-level* ou *top-level*, é o comprimento do caminho mais longo desde uma tarefa origem (tarefa entrada do grafo ou tarefa sem predecessores) até a tarefa em questão, excluindo o seu próprio custo de computação. O cálculo é feito somando-se os custos de computação das tarefas e de comunicação dos arcos pertencentes a esse caminho. Significa o tempo mais breve possível que uma tarefa pode ser escalonada respeitando as condições de precedência existentes. Utilizando a prioridade de menor conível, as tarefas são escalonadas na ordem topológica do grafo. Durante o processo de escalonamento o conível pode variar, até a tarefa ser escalonada, porque o custo de comunicação associado a um arco pode ser zerado quando duas tarefas adjacentes são escalonados no mesmo processador.

Além disso, no caso de sistemas heterogêneos os custos de computação podem ser recalculados quando o processador onde v_i é alocada é conhecido. O conível de uma tarefa é dado pela expressão:

$$conivel(v_i) = \max_{v_j \in Pred(v_i)} \{conivel(v_j) + \epsilon(v_i) \times \overline{h(p)} + w(v_j, v_i) \times \overline{L}\}$$

Onde $\epsilon(v_i)$ é o peso da tarefa v_i que multiplicado ao fator de heterogeneidade médio dos processadores disponíveis fornece o custo da tarefa v_i; $w(v_j, v_i)$ é o peso do arco entre as tarefas adjacentes v_j e v_i que multiplicado ao valor da latência média do canal $\overline{L}$, que interliga os processadores distintos que alojaram as tarefas v_j e v_i, fornece o custo da comunicação. O termo $\overline{h(p)}$ é o fator médio de heterogeneidade do sistema composto por processaores heterogêneos.

A prioridade conível é, também, calculada neste trabalho das mesmas duas formas descritas na prioridade nível. Ao ser calculada a cada passo da heurística, esta prioridade tenta ser mais precisa, uma vez que o valor do conível é reajustado quando o processador para alojar a tarefa tenha coincidido com algum que já aloja predecessores imediatos da tarefa.

3.2.4 Caminho Crítico de um Grafo CC(G)

Denominação atribuída ao caminho mais longo que se pode percorrer dentro de um grafo, desde uma tarefa de entrada até uma tarefa finalizadora do grafo. O caminho crítico de uma tarefa v_i é o caminho de maior custo que passa por essa tarefa, isto é, o conjunto de tarefas do grafo ordenadas respeitando a precedência entre elas, iniciando por uma tarefa de entrada do grafo até uma tarefa de saída no qual o somatório dos custos médios de execução e dos custos médios de comunicação é máximo. Uma observação importante é que o comprimento do caminho crítico de uma tarefa v_i é dado pela soma dos seus valores de nível e conível, dado pela expressão abaixo:

$$CC(G) = \max_{\forall t_i \in V} \{conivel(t_i) + nivel(t_i)\}$$

Por exemplo, dado o valor do caminho crítico de um grafo como sendo de 10

unidades e composto pelas tarefas v_1, v_2 e v_3. O $CC(v_1) = CC(v_2) = CC(v_3) = 10$, o que significa, que as tarefas pertencentes ao caminho crítico de um grafo apresentam todas o mesmo valor na expressão do caminho crítico que passa por elas.

No estudo do escalonamento, as tarefas do caminho crítico de um grafo GAD, representam a porção de contribuição mais significativa para o tempo de conclusão final da aplicação ou *makespan*. Isto porque, o somatório dos tempos de execução das tarefas do CC determinam o limite inferior do *makespan*. Neste estudo, a prioridade caminho crítico é utilizada para priorizar as tarefas que pertencem ao CC. Caso não haja tarefas CC na lista, são preferidas as tarefas predecessoras imediatas das CC. Ressalta-se que em um grafo pode existir mais de um caminho crítico. O cálculo desta prioridade, também, pode ser feito de forma estática e dinâmica. Assim, as tarefas do CC são ordenadas de acôrdo com as relações de precedência entre elas. Após estas, são inseridas na lista de prioridades as outras tarefas do grafo.

3.2.5 ALAP (*As Late As Possible*)

O termo traduzido significa "Tão tarde quanto possível", e representa a medida de quanto tempo se pode atrasar o início da execução de uma tarefa sem que se aumente o tempo total de escalonamento. A expressão abaixo apresenta o ALAP, do valor do caminho crítico do grafo subtrai-se o valor do nível da tarefa em questão:

$ALAP(v_i) = CC(G) - nivel(v_i)$.

Se a tarefa v_i pertence ao caminho crítico:

$ALAP(t_i) = ASAP(t_i)$, ou seja:

$ALAP(v_i) = (nivel(v_i) + conivel(v_i)) - nivel(v_i) = conivel(v_i) = ASAP(v_i)$

Significando que as tarefa do caminho crítico (CC) não podem ter seu tempo de início atrasado, se isso ocorrer resultará em um aumento no *makespan*. Por isso, tarefas do CC são ditas tarefas sem folga nos seus tempos de início.

A prioridade menor ALAP prioriza as tarefas mais críticas, ou seja, aquelas cujos

tempos de início mais tarde possível seja menor. O que torna uma tarefa mais crítica é o quanto o valor do ALAP se aproxima do seu tempo de início mais cedo possível ASAP. É, também, utilizada como prioridade e pode ser calculada de forma estática e dinâmica. Nesses dois casos, a lista é construída priorizando tarefas (de menor ALAP), ou seja, as que tem menor tempo para iniciar o mais tarde possível.

Ordem ou Nível Topológico

Termo utilizado para se qualificar um grafo em diversos degraus ou níveis topológicos de tarefas. Afim se não gerar confusão com o atributo nível, anteriormente descrito, será utilizado o termo *ordem topológica*. A ordem topológica de uma tarefa v_i é a seguinte:

$$ordem_{top}(v_i) = \max_{v_j \in Pred(v_i)} \{ordem_{top}(v_j)\} + 1$$

As tarefas origem ou de entrada v de um grafo apresentam $ordem_{top}(v) = 1$.

Função de Custo

Métrica utilizada no processo de escalonamento para determinar que atributo do grafo a heurística tenta minimizar. Neste trabalho, onde se considera um ambiente paralelo composto por processadores heterogêneos, a função de custo adotada é a que calcula o tempo mais breve de término ou (*Earliest Finih Time*), $EFT(v_i, p_j)$ de uma tarefa v_i no processador p_j. Ressalta-se que para computar esse valor, todos os predecessores imediatos de v_i já devem ter sido escalonados.

$$EFT(v_i, p_j) = min\{\epsilon(v_i) \times h(p_j) + EST(v_i, p_j)\}$$

Onde $EST(v_i, p_j)$ é o tempo mais breve de início (*Earliest Start Time*) que v_i pode iniciar sua execução em p_j. Assim, o processador "favorito"para executar uma tarefa não é, necessariamente, aquele que lhe permite o tempo de início mais breve mas sim, aquele que lhe permite terminar no tempo mais breve possível. Na Figura 3.2 é apresentado um exemplo para ilustrar a expressão do EFT de uma tarefa v_i em um processador p_j. O processador P_0 tem fator de heterogeneidade (lentidão) $h(P_0) = 1$ e o processador P_1 tem fator $h(P_1)$=2. Observe que o termo $\epsilon(v_i) \times h(p_j)$ não utiliza fator de heterogeneidade médio uma vez que já se sabe o processador

para escalonar a tarefa.

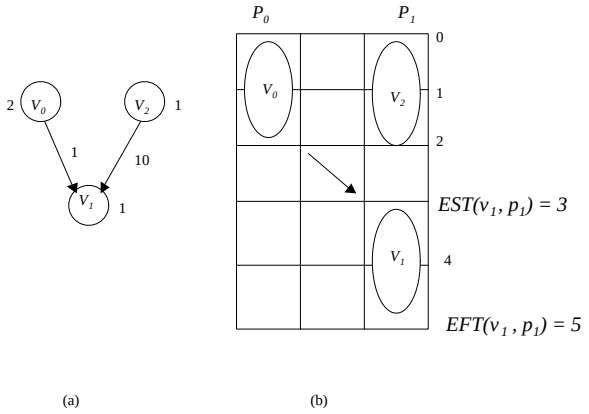

Figura 3.2: (a) GAD (b) Ambiente Heterogêneo com 2 Processadores

3.3 Resumo

O presente capítulo apresentou a técnica de *List Scheduling* escolhida para a abordagem do tema e algumas definições de termos utilizados neste documento. O próximo capítulo apresenta o algoritmo proposto baseado nessa técnica adaptada ao modelo *LogP* para ambientes com um número de processadores limitados homogêneos ou heterogêneos.

Capítulo 4

A Heurística Proposta

Este capítulo apresenta as etapas de desenvolvimento do algoritmo de escalonamento estático LSRGC (*List Scheduling Reservation Garbage Collection*) formulado para o modelo *LogP* incorporando os fatores de heterogeneidade dos processadores disponíveis. O algoritmo usa variações da técnica proposta por Kalinowski, Kort e Trystram no algoritmo ETFRGC [40]. Conforme descrito no capitulo 2, o modelo *LogP* considera as sobrecargas decorrentes das comunicações. Assim, o objetivo do novo algoritmo é minimizar o impacto dessas sobrecargas ao *makespan* da aplicação. São apresentadas quatro versões: LSRGCV0, LSRGCV1, LSRGCV2, LSRGCV3 que implementam políticas distintas para tratamento das sobrecargas. O capítulo inicia descrevendo a técnica do algoritmo ETFRGC [40]. A seguir, apresenta as versões propostas. Ao descrever cada versão, figuras com exemplos são apresentadas nas quais as sobrecargas apresentam valores unitários o que não é uma condição, uma vez que podem assumir valores arbitrários. A técnica base escolhida para a formulação da heurística é a de *list scheduling*. O objetivo primordial de uma heurística de escalonamento é minimizar o *makespan* da aplicação através de uma alocação cuidadosa das tarefas aos processadores disponíveis do sistema paralelo.

4.1 Considerações Iniciais

O objetivo final deste estudo é desenvolver um escalonador estático para uma plataforma de grade computacional. Entretanto, devido às características instáveis dos recursos da grade é necessário, um escalonador dinâmico, para que as decisões de escalonamento sejam realizadas durante a execução da aplicação. Uma tática utilizada no Projeto *Easygrid* [63] consiste em utilizar um escalonador híbrido [9], composto por um escalonador estático e outro dinâmico. Ao estático compete amenizar a carga de trabalho do dinâmico realizando um pré-escalonamento das tarefas. Dessa forma, ao dinâmico restará apenas os ajustes finais momentâneos. Esta dissertação enfoca a parte estática do escalonador híbrido do *Framework Easygrid* [62]. Para não acarretar mais complexidade ao escalonador híbrido, a proposta inicial do projeto foi desenvolver um algoritmo estático de baixa complexidade. Tal aspecto foi o ponto de partida para estabelecer que a técnica do algoritmo, dentre as apresentadas neste documento, deveria ser a de mais baixa complexidade mesmo que isso conduzisse a resultados de *Makespan* menos favoráveis. Por exemplo, algoritmos formulados na técnica de *Aglomeração* com *Replicação* apresentam resultados de desempenho superior aos formulados na técnica de *list scheduling* para ambientes homogêneos sobre o modelo *LogP*, mas requerem maior tempo computacional. Além disso, um ponto não menos relevante é que os algoritmos de aglomeração utilizam um número elevado de processadores, que às vezes pode não ser condizente com a arquitetura alvo disponível. Ainda mais, essa técnica não é facilmente adaptável para ambientes heterogêneos onde existe um número limitado de processadores, interconectados com *links* de velocidades diferentes.

Sendo assim, optou-se por escolher a técnica de *List Scheduling*, considerada segundo [40, 61] como uma técnica simples, muito utilizada na literatura para ambientes de processadores limitados, de complexidade relativamente baixa e de resultados aceitáveis em termos de *makespan* quando comparada as outras duas técnicas.

4.2 *List Scheduling* adaptada ao modelo *LogP*

Uma metodologia para adaptar uma heurística de *list scheduling* do modelo de latência para o modelo de *LogP* foi desenvolvida por Kalinowski *et al.* [40]. Os autores adaptaram a heurística ETF (*Earliest Task First*) [39], formulada originalmente para o modelo de latência em ambientes homogêneos, para suportar os parâmetros do modelo $LogP$, também em ambientes homogêneos. A heurística resultante originou a ETFRGC *Earliest Task First Reservation Garbage Collection* [40]. As duas heurísticas ETF e EFTRGC apresentam elevada complexidade, em relação à outras do grupo de *list scheduling*. Uma vez que, todas as tarefas livres, candidatas ao escalonamento, são testadas em todos os processadores disponíveis. Aquele que permitir menor tempo de início para determinada tarefa é escolhido. As etapas principais da metodologia podem ser resumidas da seguinte forma:

1) Existem duas variáveis principais no processo, o *Próximo Momento de Envio (PME)* e o *Tempo de Chegada da Mensagem (TCM)*. O PME calcula, a cada passo do escalonamento, o próximo instante em que pode ser enviada a mensagem de dados (após a respectiva sobrecarga de envio O_s) e é incrementado a cada O_s escalonada. O TCM marca o tempo de chegada da mensagem e serve para determinar o início de cada sobrecarga de recebimento O_r;

2) Conforme ilustra a Figura 4.1, após o escalonamento de cada tarefa v_i em um processador P_j, uma área de reserva R é prevista em P_j. Essa área é grande o suficiente para acomodar todas as sobrecargas de envio, caso seja necessário enviar mensagens para todos os sucessores de v_i alocados em processadores diferentes de P_j. Sendo a ETFRGC formulada para ambientes homogêneos, os processadores do exemplo são os três homogêneos;

3) Para cada tarefa sucessora de v_i designada a processador diferente de P_j, uma sobrecarga de envio O_s é escalonada dentro da reserva R. Ao término dessa sobrecarga, instante marcado pela variável PME começa a transmissão da mensagem de dados. Depois que a mensagem chega ao processador destino (instante marcado pelo TCM), a respectiva sobrecarga de recebimento O_r é escalonada. Na Figura 4.2

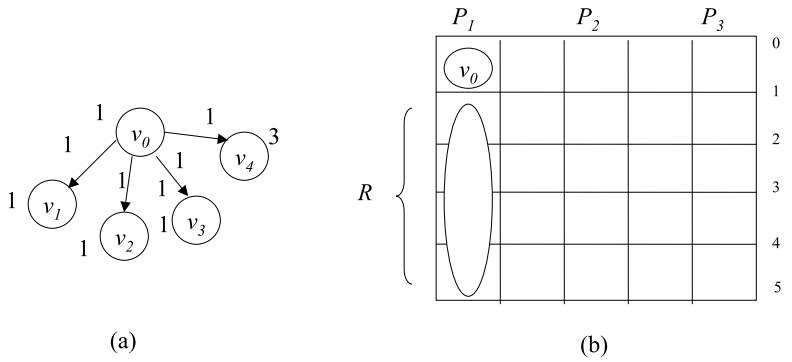

Figura 4.1: (a) GAD. (b) Área de Reserva R de tamanho 4

é apresentado o escalonamento do grafo de cinco tarefas ilustrado na Figura 4.1(a) em uma arquitetura paralela de três processadores homogêneos. A heurística ETFRGC somente suporta ambientes homogêneos. Observe que na Figura 4.2 (a) após v_0 é reservada a área R. As tarefas sucessoras v_1 e v_2 são escalonadas no mesmo processador de v_0, as reservas marcadas com asteriscos tornam-se inúteis e serão removidas ao final do escalonamento. Em 4.2 (b) é ilustrado o escalonamento das tarefas sucessoras de v_0: v_3 e v_4, que são designadas pela heurística a processadores diferentes de P_1 (os processadores são homogêneos, ou seja, o fator de heterogeneidade dos três é igual a 1) e das respectivas sobrecargas O_s e O_r. Em 4.2(c) são ilustrados os envios das mensagens após o término de cada sobrecarga de envio;

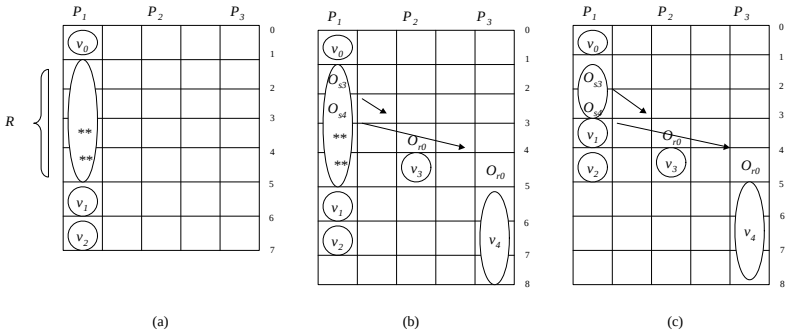

Figura 4.2: Exemplo da heurística ETFRGC (a) Reserva R (b) Sobrecargas de envio alocadas em R (c) Escalonamento Final com Makespan = 8

4) Os *Tempos de Chegada das Mensagens* TCM no processador destino, estabelece o início de cada sobrecarga de recebimento O_r. Somente após o recebimento de todas as mensagens, é que a tarefa recebedora dos dados, sucessora de v_i, é escalonada. Por exemplo, na 4.2(c) o TCM da mensagem enviada pela sobrecarga O_{s3}

ocorre no processador P_2 no tempo 3. Nesse tempo é escalonada a sobrecarga de recebimento O_{r0} e ao seu término é escalonada a tarefa recebedora dos dados v_3;

5) Considerando que tarefas sucessoras de v_i podem ser designadas ao mesmo processador P_j, o espaço destinado em R a essas sobrecargas de envio, que não ocorreram, tornam-se inúteis; isto porque não podem acomodar sobrecargas de outra tarefa. Por isso, ao fim do escalonamento de todo o grafo é aplicada uma rotina de *Garbage Collection* (coleta de lixo) que ajusta os tempos de início das tarefas removendo o espaços não utilizados dentro de todas as áreas de reserva, como ilustra a Figura 4.2 (c).

Nesta seção foi descrita a metodologia para a daptação da heurística ETF aos parâmetros do modelo *LogP* originando a ETFRGC [40], ambas suportando apenas arquiteturas com processadores homogêneos. Uma vez que a função de custo que buscam minimizar é o tempo mais breve que uma tarefa pode começar a ser executada em um determinado processador. Até a presente pesquisa, desenvolvida nesta dissertação, essa metodologia apresentada por por Kalinowski *et al.* [40] era a única existente na literatura abordando heurísticas de *List Scheduling* adaptadas ao modelo *LogP*. Sendo esta dissertação voltada para ambientes heterogêneos as versões propostas, com novas políticas de tratamento para as sobrecargas, foram desenvolvidas para suportar ambientes homogêneos e heterogêneos e são descritas na próxima seção.

4.3 As Versões Alternativas Propostas

Nesta seção são descritas as três versões propostas para a LSRGCV (*List Scheduling Reservation Garbage Collection*), chamadas de: LSRGCV1, LSRGCV2 e LSRGCV3. Tratam-se de variações da metodologia empregada na heurística ETFRGC. A partir dessa estratégia foram desenvolvidas novas versões, que suportam ambientes homogêneos e heterogêneo. As versões LSRGCV1, LSRGCV2, LSRGCV3 apresentam políticas alternativas com o objetivo de avaliar novas formas de tratamento para minimizar os efeitos que as sobrecargas decorrentes das comunicações

entre os processadores, causam ao *makespan* da applicação. A versão LSRGCV0 implementa a mesma técnica proposta por Kalinowski *et al.* utilizada na ETFRGC, porém, em um algoritmo de *list scheduling* padrão.

4.3.1 A Versão LSRGCV0

Esta versão foi criada com o objetivo de avaliar a metodologia proposta por Kalinowski, Kort e Trystram em um algoritmo básico da técnica de *list scheduling*. O algoritmo de ETF [39] que os autores se basearam, apesar de ser considerado do grupo de *list scheduling*, utiliza momentos de decisão para o escalonamento, guiados por um relógio de tempo. Assim, determinados parâmetros são testados e dependendo de seus valores o escalonamento de uma tarefa pode ser adiado. Além disso, a decisão para o escalonamento é obtido por tentativas testando todas as tarefas candidatas, tarefas da lista de livres, em todos os processadores disponíveis. Essa busca exaustiva pelo melhor tempo de início de uma tarefa, confere a heurística superioridade de resultados. Porém, acarreta complexidade mais alta em comparação a uma heurística básica da técnica de *list scheduling*. Sob essas diferenças não ficaria justo comparar a heurística dos autores com as variações propostas por esta dissertação: LSRGCV1, LSRGCV2 e LSRGCV3. Assim, optou-se por implementar a mesma estratégia da ETFRGC, originando a LSRGCV0. Além disso, foco deste trabalho é avaliar qual é a melhor estratégia de tratamento das sobrecargas em algoritmos de escalonamento sob o modelo *LogP*, e não avaliar resultados de escalonamentos produzidos por algoritmos de *list scheduling*.

4.3.2 A Versão LSRGCV1

Esta versão pode ser descrita através das seguintes etapas:

1) Uma área de reserva R continua a ser prevista após o escalonamento de cada tarefa v_i. Conforme as sobrecargas de envio são escalonadas, dentro da reserva R, uma restrição imposta somente permite que as mensagens sejam enviadas ao fim de todo o espaço reservado R. Com isso, inicialmente, as sobrecargas ainda não são

associadas às mensagens transmitidas e, portanto, são consideradas anônimas;

2) As sobrecargas de recebimento são escalonadas, no processador destino, ordenadas pelo TCM antes do escalonamento das tarefas recebedoras dos dados;

3) Ao fim do escalonamento é aplicada a rotina de coleta de lixo;

4) Uma etapa final do algoritmo realiza a nomeação e ordenação das sobrecargas de envio com base no conceito do *caminho crítico do grafo escalonado*, que é o maior caminho que se pode percorrer dentro do grafo escalonado. O valor desse caminho é calculado usando uma função que soma os custos de execução das tarefas e os custos de comunicação (incluindo os custos das sobrecargas). Esse valor determina o *makespan* da aplicação. Assim, a ordenação para ocupar as primeiras posições dentro das reservas R, prioriza as sobrecargas que enviam dados para tarefas pertencentes a esse caminho. Com isso, é possível antecipar o recebimento dos dados e conseqüentemente abreviar o início das execuções dessas tarefas diminuindo o *makespan* do escalonamento, que é a métrica a ser minimizada. Quanto menor é o *makespan* melhor é o escalonamento gerado. A Figura 4.3 ilustra o escalonamento do grafo apresentado em (a) pela heurística LSRGCV1 em uma arquitetura composta por três processadores heterogêneos, os processadores P_0 e P_1 possuem fator de heterogeneidade (lentidão) de 1, sendo os mais rápidos e o processador P_2, é o mais lento, com fator de atraso igual a 2. A mesma arquitetura será utilizada nos exemplos ilustrativos das versões LSRGCV2 e LSRGCV3, descritas adiante.

Observe em 4.3(b) que a área R, de tamanho de 6 unidades, reservada após v_0, com o escalonamento das tarefas sucessoras v_1 e v_2 em P_1 apresenta espaços marcados com asteriscos 4.3(c). Os asteriscos indicam que os espaços destinados as respectivas sobrecargas de envio, para essas duas tarefas, não foram utilizados. Esses espaços serão removidos após o escalonamento de todo o grafo. Além disso, é possível observar que os envidas mensagens somente ocorrem ao fim de toda a reserva R original de 6 unidades e que as sobrecargas são anônimas. Em (d) é ilustrado que já ocorreu a coleta de lixo removendo os espaços inativos existentes em R e que as sobrecargas foram ordenadas e associadas as mensagens que transmitem.

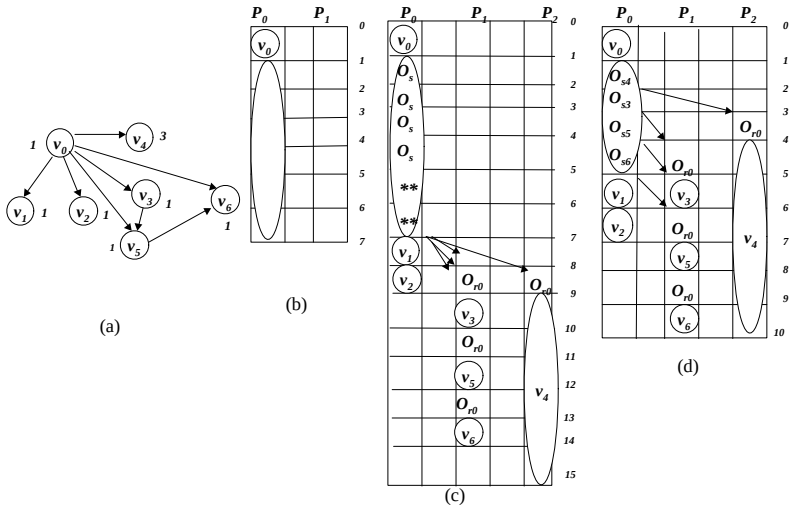

Figura 4.3: Exemplo da heurística LSRGCV1 (a) GAD (b) Reserva R (c) Sobrecargas anônimas (d) Escalonamento Final com *makespan*=10

Após a coleta de lixo as sobrecargas são ordenadas priorizando para ocupar a primeira posição em R, a sobrecarga que envia mensagem para v_4. Uma vez que a tarefa v_4 é a que determina o tempo final do escalonamento, que ocorre no tempo 15. Por isso, a tarefa v_4 é priorizada para receber a mensagem de dados e assim, seu início é antecipado para o tempo 4, conforme pode ser observado na Figura 4.3(d). Adiante, na seção sobre critérios para ordenação das sobrecargas mais detalhes serão apresentados.

4.3.3 A versão LSRGCV2

A política desta versão é praticamente a mesma da LSRGCV0. Porém, a inovação consiste em aplicar a coleta dos espaços não utilizados, dentro das reservas R, a cada passo do escalonamento. Dessa forma, não existem espaços inativos durante o escalonamento. O funcionamento da versão LSRGCV2 pode ser descrita através das seguintes etapas:

1) Após o escalonamento de cada tarefa de computação são reservadas as áreas R. No exemplo apresentado na Figura 4.4(b);

2) A cada passo do algoritmo, caso uma tarefa sucessora seja designada ao mesmo processador que já aloja um de seus predecessores imediatos, é aplicada a rotina de

coleta de lixo. Como nas outras versões a coleta remove o espaço, dentro da reserva R, destinado à sobrecarga de envio que não ocorre. A coleta de lixo instantânea permite reajustar o tamanho de R durante o escalonamento. Com isso, não existirão espaços inúteis, dentro das reservas R (durante o escalonamento). Conforme ilustra a Figura 4.4 (c). A área R inicialmente era de tamanho 6, devido ao escalonamento de v_3, v_5 e v_6 no mesmo processador de v_0, a área R é reajustada para um tamanho de 3 unidades;

3) As sobrecargas de envio são associadas as mensagens durante o escalonamento e ao fim de cada uma delas as respectivas mensagens são enviadas. As sobrecargas de recebimento são escalonadas, ordenadas pelo TCM, antes do escalonamento da tarefa de computação recebedora dos dados; tal como ocorria na versão LSRGCV0. A Figura 4.4(c) ilustra o escalonamento final gerado por esta versão; observe que o *makespan* é igual a 9 unidades.

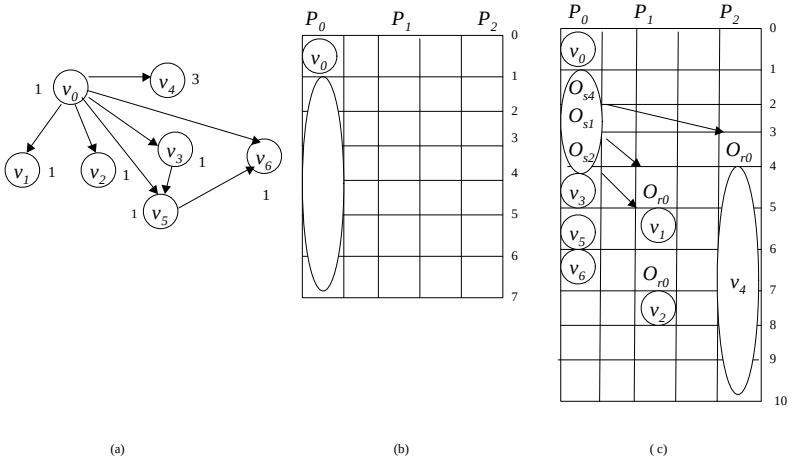

Figura 4.4: Exemplo da heurística LSRGCV2 (a) GAD G (b) Reserva R (c) Coleta de lixo instantânea e sobrecargas ordenadas resultando num escalonamento final com *makespan*=9

4.3.4 A Versão LSRGCV3

Esta versão é, relativamente, semelhante a LSRGCV1. Porém, ao invés de realizar a coleta de lixo no final do escalonamento, é feita a cada passo da heurística, reajustando o tamanho da reserva R. A Figura 4.5 ilustra as etapas desta versão,

que são exemplificadas através de figuras, a fim de complementar o que já foi exposto na versão LSRGCV1. O grafo utilizado no exemplo é aprsentado na Figura 4.5(a). Na mesma Figura 4.5(b)é possível verificar que o tamanho da reserva R é reajustado para 5 unidades após o escalonamento de v_1 em P_0. Com isso, não existem espaços inúteis dentro da reserva R. No passo seguinte, com o escalonamento de v_4, também, no processador P_0, o tamanho de 5 unidades é reajustado, com a coleta de lixo instantânea, para 4 unidades. Na Figura 4.5(c) é apresentado o escalonamento (não final) de todas as tarefas. Porém, as sobrecargas são anônimas e as mensagens circulam ao fim da reserva R. Numa próxima etapa do algoritmo, as sobrecargas serão nomeadas e ordenadas.

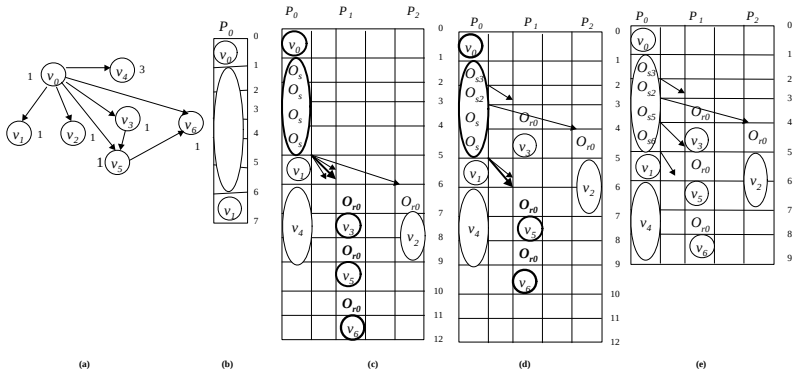

Figura 4.5: Exemplo da heurística LSRGCV3 (a) GAD G (b) Reserva R com a coleta de lixo instantânea (c) Escalonamento intermediário com as sobrecargas ainda anônimas (d) Ordenação das sobrecargas de envio (e) Escalonamneto final com *makespan*=9

A Figura 4.5(c) ilustra, em negrito, o caminho crítico do grafo escalonado. O caminho ilustrado na figura citada é composto pelas tarefas de computação v_0, v_3, v_5 e v_6, além das sobrecargas e arestas de comunicação. Dessa forma, a sobrecarga que envia dados para a tarefa v_3 é nomeada e priorizada para ocupar a primeira posição em R. Entretanto, uma vez posicionada a sobrecargas ordenada, dentro da reserva R, não poderá mais ser reordenadas afim de contemplar novos caminhos críticos que apareçam durante o processo de ordenação das sobrecargas de envio e antecipação de tarefas. Após a ordenação da sobrecarga O_{s3} a tarefa v_3 é antecipada para iniciar no tempo 4. A seguir a heurística verifica que é mais vantajoso priorizar a O_{s2} para ocupar a segunda posição dentro da reserva R, ao invés de priorizar a O_{s5}. Isto porque, é verificada a *janela máxima de subida* existente acima de cada sobrecarga

de recebimento que se deseja antecipar. Neste exemplo, na Figura 4.5(d) pode ser observado que a tarefa v_5 e sua respectiva sobrecarga de recebimento O_{r0} somente podem subir uma unidade. Em contra partida v_2 e sua sobrecarga de recebimento O_{r0} podem subir duas unidades. Na Figura 4.5(e) é apresentado o escalonamento final, com todas as sobrecargas ordenadas e com *makespan* de 9 unidades.

4.4 Critérios de ordenação das sobrecargas de envio

Quando uma tarefa v_i é escalonada em um processador p_j, três valores são a ela atribuídos: O tempo de início de sua execução $ST(v_i, p_j)$; O tempo de término de sua execução $FT(v_i, p_j)$; e o processador p_j em que vai ser executada. Dessa forma, após o escalonamento de todas as tarefas do GAD é possível construir um *grafo de escalonamento* a partir do mapa de escalonamento gerado, conforme ilustrado na Figura 4.5(c), anteriormente apresentada. Dessa forma, pela observação do grafo escalonado, é possível definir os conceitos de *Nível Escalonado*, *Conível Escalonado* e *Caminho Crítico Escalonado*. Tais conceitos, definidos adiante, serão utilizados para a ordenação e nomeação das sobrecargas de envio, dentro dos espaços de reserva R de cada tarefa escalonada. O conceito de cada um desses três atributos, é praticamente o mesmo dos conceitos de *nível*, *conível* e *caminho crítico* apresentados no capítulo 3. Porém, ao invés de serem calculados sobre um grafo de tarefas GAD são calculados sobre o grafo escalonado, ou seja, a partir do mapa de escalonamento gerado.

Nível Escalonado - N_E

Denominação que se atribui ao maior caminho que se pode percorrer em um *grafo escalonado*, desde a tarefa em questão v_i, até a tarefa de saída (finalizadora) do escalonamento. Essa tarefa v_z é aquela que apresenta o maior tempo de término (no mapa de escalonamento). Para calcular esse valor, primeiro é preciso verificar o(s) processador(es) que apresenta(m) o tempo final do escalonamento, ou seja:

$$Makespan = \max_{\forall v_z \in V} \{FT(v_z, p_j)\}$$

O processador *Makespan* é aquele (p_j) onde v_z foi escalonada. Na Figura 4.5(c)

é possível observar que P_1 é o processador $P_{makespan}$ por apresentar a tarefa terminal v_6 finalizadora do caminho descrito em negrito. Em seguida, a partir de v_6, subindo no grafo, é preciso visitar todas as tarefas escalonadas, de acordo com os valores decrescentes de seus tempos de término. Por exemplo: na mesma figura para calcular o *nível escalonado* de v_0, a partir de uma tarefa de saída (v_6) do escalonamento, calcula-se o somatório dos *tempos de execução* (TE) das tarefas e das sobrecargas, de envio e de recebimento, adicionado ao somatório dos tempos de comunicação (TC) entre as tarefas pertencentes ao maior caminho desde v_0 até v_6, assim, o *nível escalonado* $N_E(v_6) = 12$. A expressão abaixo apresenta o cálculo do nível escalonado considerando o escalonamento definido pela heurística LSRGCV1 ou LSRGCV3, onde a tarefa v_j pertence ao conjunto de sucessores de v_i.

$$Nivel_E(v_i) = TE(v_i, p_j) + max\{TC(v_i, v_j) + nivel_E(v_j) + \sum O_s(v_i) + \sum O_r(v_y)\}$$

Onde $TE(v_i, p_j) = \epsilon(v_i) \times h(p_j)$, é o tempo de execução da tarefa v_i no processador p_j que apresenta fator de heterogeneidade $h(p)$. Observe que aqui não se usa fator de heterogeneidade médio, pois sabe-se o processador, e portanto o seu respectivo fator de heterogeneidade (lentidão), no qual a tarefa v_i está escalonada. O termo $TC(v_i, p_j, v_j, p_q) = w(v_i, v_j) \times L(p_j, p_q)$, se $p_j \neq p_q$; caso seja no mesmo processador $TC(v_i, v_j) = 0$. Os processadores p_j e p_q são, respectivamente, os processadores que a heurística designou para alojar as referidas tarefas. O termo L é a latência do canal de comunicação entre p_j e p_q, para transmitir uma unidade de dados. Uma vez conhecidos os dois processadores não é preciso utilizar a latência média, e sim a própria latência do canal existente entre os dois processadores.

Da mesma forma, define-se o conceito de *Conível Escalonado*. O cálculo é bastante simples uma vez que coincide com o tempo de início de execução da tarefa v_i em p_j, $ST(v_i, p_j)$, ou seja:

$$Conivel_E(v_i) = ST(v_i, p_j)$$

A partir dos dois conceitos acima é possível definir o *Caminho Crítico do Grafo Escalonado* (CCE):

$$\text{Valor } CCE = \max_{\forall v_i \in V}\{Nivel_E(v_i) + Conivel_E(v_i)\} = Makespan$$

Uma tarefa pertence ao CCE se a soma do valor de seu *nível escalonado* com o valor do seu *conível escalonado* for igual ao valor do *makespan* do escalonamento. No exemplo da Figura 4.5(c) o *makespan* é de 12 unidades. É ainda, um valor não final, uma vez que serão ordenadas as sobrecargas de envio, com o objetivo de minimizar esse valor. A estratégia para minimizá-lo consiste em ordenar as sobrecargas de envio O_s, dentro dos espaços de reserva R, com o intuito de antecipar o envio das mensagens de dados para as tarefas pertencentes ao CCE. Em conseqüencia, recebendo os dados mais cedo as tarefas do CCE podem iniciar suas execuções, também, mais cedo contribuindo para a diminuição do *makespan*.

Após calcular as tarefas pertencentes ao CCE é necessário verificar se existem espaços livres acima da sobrecarga que irá receber os dados (que serão antecipados pela priorização da sobrecarga de envio). Caso, essa sobrecarga de recebimento seja a primeira do processador, é calculado o tamanho da chamada *janela máxima de subida*, J_M. Assim, a sobrecarga de recebimento poderá subir no máximo de (R - O_s). A medida que os O_r vão subindo no mapa de escalonamento é possível que o CCE se altere, por isso, é preciso efetuar constantes recálculos para computar novas tarefas CCE.

A Figura 4.6(a) ilustra em negrito o *caminho crítico escalonado*, do mesmo grafo apresentado na Figura 4.5(a). A ilustração 4.6(b) apresenta as *janelas máximas de subida* J_M representadas pelas áreas hachuradas acima da sobrecarga de recebimento O_{r0} existente em P_1 e acima da sobrecarga de recebimento O_{r0} existente em P_2. Após a ordenação da sobrecarga de envio O_{s3}, com base nas *janelas máximas de subida* a sobrecarga O_{s2} é preferida para ocupar a segunda posição dentro da reserva R, por apresentar uma janela de subida mais vantajosa para aproveitar a antecipação do recebimento da mesnsagem. A *janela máxima de subida* da sobrecarga de recebimento da tarefa v_5 é limitada pelo término da tarefa v_3, apresentando tamanho de uma unidade. O que resultará em ordenar a O_{s5} dentro do espaço R na posição permitida pela janela J_M, ou seja, subindo no máximo apenas 1 unidade. Na mesma figura letra (c) é ilustrada a ordenação final que resultou no *makespan* de 9 unidades.

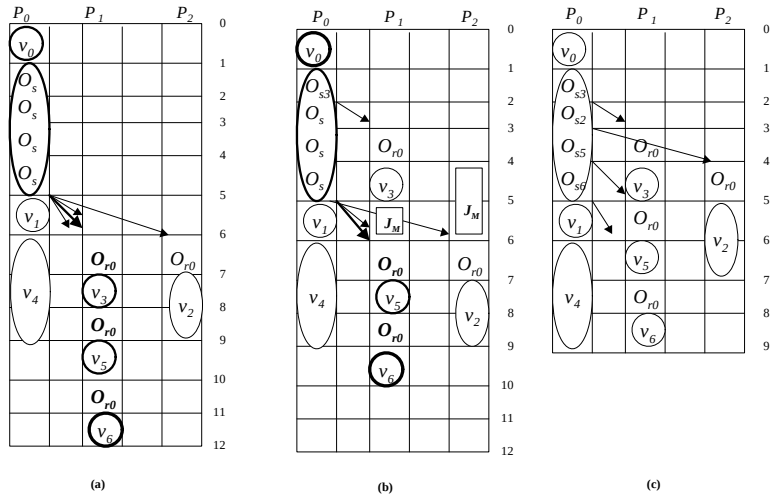

Figura 4.6: (a)Caminho Crítico do Grafo Escalonado (b) Janela máxima de subida J_M (c) Ordenação das sobrecargas de envio

4.5 Resumo

Este capítulo iniciou descrevendo uma técnica existente na literatura, para adaptar o algoritmo ETF [39], formulado no modelo de latência aos parâmetros do modelo de comunicação *LogP*, resultando na heurística ETFRGC [40], ambas somente suportando ambientes homogêneos. Baseada nessa metodologia, foi proposta a heurística LSRGC *List Scheduling Reservation Garbage Collection*, apresentada sob quatro versões: LSRGCV0, LSRGCV1, LSRGCV2 e LSRGCV3. Todas as quatro versões estão adaptadas para suportar ambientes homogêneos e heterogênos e para os modelos de Latência e *LogP*. Além disso, permitem utilizar oito escolhas diferentes para priorizar tarefas livres, e dois critérios sucessivos de desempate. O primeiro critério decide qual tarefa escolher em caso de empate na prioridade principal. O segundo critério é utilizado para decidir qual tarefa escolher caso persista o empate. As prioridades escolhidas, e que podem ser calculadas de forma dinâmica e de forma estática, são as seguintes: *Nível*, *Conível*, *Caminho Crítico* e *ALAP*. Os dois próximos capítulos apresentam os testes realizados. O Capítulo 5 apresenta os testes efetuados em ambientes heterogêneos sob o modelo de Latência, o objetivo é verificar se algum trio de prioridades (prioridade principal; prioridade de primeiro critério de desempate; e prioridade de segundo critério de desempate sucessivo) propostas se destaca sobre as outras prioridades. O Capítulo 6 apresenta os testes realizados em

ambiente heterogêneo sob o modelo *LogP* que objetivam determinar qual é a melhor estratégia (considerando as quatro versões apresentadas) para minimizar os efeitos adversos que as sobrecargas, decorrentes das comunicações, causam ao *makespan* das aplicações paralelas.

Capítulo 5

Experimentos em Ambiente Heterogêneo sob o Modelo de Latência

Este capítulo destina-se a avaliar prioridades e duplas de critérios de desempate em uma heurística básica formulada na técnica de *List Scheduling* sob o modelo de latência. No total foram realizados 8320 experimentos, utilizando quatro arquiteturas heterogêneas e agrupados pelas seis classes dos grafos (descritas no Apêndice A). O objetivo é verificar qual prioridade e critério de desempate mais se destacam em cada classe de topologias, sob o modelo da Latência. Ao final do capítulo é eleita a prioridade e dupla de critério de desempate, que permitiram a heurística *List Scheduling* atingir os melhores resultados de *makespan* considerando todos os grafos e as quatro arquiteturas alvo disponibilizadas.

5.1 Considerações Iniciais

Nos experimentos foram utilizadas quatro arquiteturas compostas por processadores e por canais de comunicação (que interligam processadores) heterogêneos.

Os grafos GADs foram divididos por classe de topologia e agrupadas dentro de dois conjuntos principais, grafos unitários, nos quais as tarefas e arcos têm pesos unitários:

1. Diamantes;
2. *Intree*;
3. *Outtree*;e
4. Randômicos.

e grafos não unitários, nos quais as tarefas apresentam tarefas e arcos com pesos arbitrários:

1. Irregulares; e
2. KPSG *Kwok Peer Set Graphs*).

As prioridades avaliadas (descritas no capítulo 3) são:

1. *Nível*;
2. *Conível*;
3. *Caminho Crítico*; e
4. *ALAP*.

Calculadas de forma estática e recalculadas de forma dinâmica. Constituindo, assim, 8 possibilidades de prioridades. Além disso, são adotadas duas possibilidades: sem considerar nenhum critério de desempate e considerando dois critérios sucessivos de desempate. Os dois critérios sucessivos são aplicados da seguinte forma: o primeiro critério é empregado caso haja empate na prioridade principal de tarefas canditadas ao escalonamento; caso novo empate aconteça é utilizado o segundo critério. Os dois critérios formam, assim, um par, onde os quatro pares adotados são:

1. sem adotar critérios de desempate ou critério (A);

2. Nível Dinâmico e Caminho Crítico Dinâmico ou critério de desempate (B);

3. Conível Dinâmico e Caminho Crítico Dinâmico ou critério de desempate (C); e

4. Conível e ALAP ou critério de desempate(D).

Tais pares ou critérios (B), (C) e (D) foram escolhidos de acôrdo com os experimentos realizados onde, por observação dos valores de melhor percentual de *makespan* alcançado pelas heurísticas os pares com percentuais menos relevantes foram desconsiderados. Os experimentos foram agrupados em forma de baterias, realizadas para cada uma das quatro arquiteturas utilizadas e por cada classe de grafos, totalizando 24 baterias de testes. Cada uma das baterias era composta por 32 versões de heurísticas de *list scheduling*. Isto porque, uma vez modificada a escolha da prioridade principal ou os pares de critério de desempate nova versão é considerada:

1. Em 8 versões não se adotou nenhum critério de desempate (A);

2. Em 8 versões adotou-se o par de desempate *Nível Dinâmico* e *Caminho Crítico Dinâmico* (B);

3. Em 8 versões adotou-se a dupla de desempate *Conível Dinâmico* e *Caminho Crítico Dinâmico* (C);

4. Em 8 versões adotou-se a dupla de desempate *Conível* e *ALAP* (D).

Por exemplo: *list scheduling* com prioridade *Nível*, primeiro desempate *Nível Dinâmico* e segundo desempate *Caminho Crítico Dinâmico*, também pode ser referenciado como *list scheduling* com prioridade *Nível* e critério *A* ; *list scheduling* com prioridade *ALAP*, primeiro desempate *Conível Dinâmico* e segundo desempate *Caminho Crítico Dinâmico*, ou *list scheduling* com prioridade *ALAP* e critério *C*.

Em cada uma das 24 baterias (6 classes de grafos e 4 arquiteturas) de testes, as 32 versões das heurísticas de *List Scheduling* eram executadas ao mesmo tempo. Uma vez gerados os 32 valores de *makespan* com base na observação do melhor valor, ou seja, do menor *makespan* apresentado, calculava-se o percentual de melhor *makespan* atingido, por cada versão. Por exemplo, a Figura 5.1 ilustra uma parte da tabela que foi montada para a confrontação dos resultados. A tabela, para a classe de grafos diamantes, é composta por 11 linhas, uma para cada grafo e 32 colunas uma para cada versão de *List Scheduling*. Considerando uma respectiva arquitetura de processadores. Na tabela exemplo os valores indicam o *makespan* atingido pela versão para o grafo em questão, e entre parenteses está o número de processadores utilizados. Para cada grafo (em cada linha) encontra-se o melhor *makespan*, depois observa-se na coluna o percentual de melhores ou iguais *makespan* atingidos pela referida versão de *List Scheduling*. Por exemplo: $1/11 = 9\%$ somente em um grafo do total de 11 grafos as versões): Niv. + A; Con. + A; e ConD. + B atingiram o melhor ou igual *makespan*; e $4/11 = 36\%$, em quatro grafos do total de 11 grafos as versões, no exemplo, CC+A e Cam. Crí. + B atingiram o melhor ou igual *makespan*.

	...	Nív + A	Convel + A	CC + A	...	Cam. Crí. + B	Convel D. + B	...
Di9		9(2)	9(2)	9(2)		9(2)	9(2)	
Di16		13(3)	13(3)	13(3)		13(3)	13(2)	
Di25		17(4)	17(4)	17(4)		17(4)	17(2)	
Di36		23(4)	23(4)	23(4)		23(4)	22(3)	
Di64		32(4)	32(4)	32(4)		32(4)	33(4)	
Di100		46(4)	46(4)	46(4)		46(4)	46(4)	
Di144		59(5)	59(5)	59(5)		59(5)	60(6)	
Di225		83(7)	87(7)	83(7)		83(7)	82(7)	
Di256		90(7)	90(7)	90(7)		90(7)	91(8)	
Di400		126(8)	126(8)	126(8)		126(8)	126(9)	
Di1024		276(9)	276(9)	276(9)		276(9)	276(11)	
		9%	9%	36%		36%	9%	

Figura 5.1: Tabela de exemplo (parte) da confrontação dos 32 *makespan* para calcular o percentual de melhor ou igual *makespan*

Além disso, foram calculadas as degradações em relação ao melhor *makespan* apresentadas por cada uma das 32 versões. A degradação em relação ao melhor valor do *makespan*, também, é conhecida como qualidade do escalonamento. Quanto mais próximo do valor um (1) melhor é a qualidade do escalonamento produzido. Os resultados são apresentados em forma de tabelas (Figuras) onde existem colunas com os percentuais e colunas com os resultados da qualidade do *makespan* produzido. Ao final de cada classe são apresentadas iguais estruturas de tabelas compostas com os percentuais, atingidos na média, pelas quatro arquiteturas utilizadas.

5.2 Arquiteturas Utilizadas

Os testes foram realizados utilizando-se quatro arquiteturas distintas. Cada uma delas é compostas, respectivamente por 8, 12, 32 e 64 processadores, com diferentes fatores de heterogeneidade $h(p)$, que representam a lentidão ou atraso do processador em executar uma determinada tarefa. A tabela da Figura 5.2 apresenta, em cada arquitetura utilizada, o número de processadores com o respectivo fator de heterogeneidade que apresentam. Fato a ser notado na distribuição apresentada é que na arquitetura de 64 processadores, praticamente, todas as máquinas são as mais rápidas. Isto porque, em uma plataforma de grade computacional espera-se que sejam colocadas máquinas mais rápidas e não mais lentas ou ultrapassadas. Além disso, existe uma certa conformidade em as máquinas de um determinado domínio apresentarem o mesmo fator de heterogeneidade ($h(p) = 1$). O que lhes atribui a característica de serem homogêneas em relação ao domínio de localização.

	P = 8	P = 12	P = 32	P = 64
h(p)=1	1	2	7	59
h(p)=2	1	4	7	2
h(p)=4	2	3	7	1
h(p)=8	3	2	8	1
h(p)=16	1	1	3	1

Figura 5.2: Fator de Heterogeneidade (lentidão) dos Processadores por Arquitetura

A Figura 5.3 ilustra o arquivo de entrada da arquitetura de 12 processadores heterogêneos. A primeira linha identifica o número total de processadores, as 12 linhas seguintes referem-se a cada um deles e iniciam com o número representativo do seu fator de heterogeneidade (representa o grau de lentidão do processador, quando multiplicado pelo peso da tarefa permite obter o tempo de execução da tarefa naquele processador); o nome da máquina; e os valores das sobrecargas de envio (O_s) e de recebimento (O_r) que são iguais a zero no modelo de Latência. Assim, a máquina *abacate.ic.uff.br* e a máquina *morango.ic.uff.br* são as mais rápidas e a máquina mais lenta com $h = 16$ é a *melancia.ic.uff.br*. O arquivo termina com uma Matriz 12 x 12 representativa dos fatores de atraso ou Latência do canal de comunicação entre os processadores. A matriz é triangular superior, indicando que existe o mesmo fator entre o canal de comunicação $p_{i,j}$ e $p_{j,i}$ mas esses valores poderiam não ser os

mesmos. As arquiteturas de 8, 32 e 64 processadores não estão apresentadas uma vez que são semelhantes ao padrão da arquitetura ilustrada na Figura 5.3. Idem na confecção da matriz (8X8), (32X32) e (64X64) para cada arquitetura.

```
12
8 goiaba. ic.uff.br 0 0
4 pequi. ic.uff.br 0 0
1 abacate. ic.uff.br 0 0
2 pitanga. ic.uff.br 0 0
2 acerola. ic.uff.br 0 0
4 carambola. ic.uff.br 0 0
2 pinha. ic.uff.br 0 0
1 seriguela .ic.uff.br 0 0
8 banana. ic.uff.br 0 0
2 morango. ic.uff.br 0 0
4 caju. ic.uff.br 0 0
16 melancia. ic.uff.br 0 0
0 1 2 1 1 4 1 1 1 2 1 1
1 0 1 1 1 1 1 1 1 1 1 1
2 1 0 1 1 1 1 1 1 1 1 2
1 1 1 0 1 1 1 1 2 1 1 2
1 1 1 1 0 1 1 1 1 1 1 1
4 1 1 1 1 0 1 1 1 1 1 4
1 1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 1 2 1 1 0 1 1 1
2 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 1 1 1 1 1 1 0 1
1 1 2 1 1 4 1 1 4 1 1 0
```

Figura 5.3: Arquitetura de 12 Processadores Heterogêneos no Modelo de Latência

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
h(p)	8	4	1	2	2	4	2	1	8	2	4	16

Figura 5.4: Tabela Processadores X Lentidão, na arquitetura de 12 Processadores

Todos os experimentos foram realizados em duas etapas. Foi necessário incorporar duas etapas face a diversidade de escolhas que poderiam ser utilizadas para se analisar os escalonamentos gerados. Na primeira etapa as 32 versões da heurística *List Scheduling* eram executadas, em cada uma das quatro arquiteturas e dentro de cada classe de grafos. O objetivo era verificar qual das versões apresenta a prioridade e par de critério de desempate que resultava no melhor *makespan* e na melhor qualidade. Com base na prioridade e no critério que se destacasse era iniciada a segunda etapa de testes, utilizando essas prioridades, com o objetivo de analisar os escalonamentos gerados. Assim, alguns exemplos são ilustrados procurando-se confrontar o escalonamento da versão de melhor *makespan* com outros escalonamento gerados

pelas concorrentes. Nos escalonamentos ilustrados as representações das comunicações foram omitidas e apenas são mostradas as distribuições das tarefas. O objetivo é permitir uma melhor visualização das alterações resultantes do escalonamento.

5.3 Grafos Diamantes

A classe de grafos com topologia isoforma a uma grade planar conhecida como diamante, é composta por grafos com $n \times n$ tarefas. Cada vértice ou tarefa v_i coincide com as coordenadas cartesianas (i, j) onde $i = 1, ...n$ e $j = 1, ...n$. Nos experimentos foram utilizadas 11 instâncias de grafos diamantes, sendo que cada grafo apresenta um número de tarefas variando entre 9 a 1024. Utilizando esta classe de grafos, no total foram realizados 1408 experimentos para analisar a qualidade e o *makespan* produzido. A Figura 5.5 ilustra a topologia de um grafo diamante $5X5$ utilizado para ilustrar os escalonamentos gerados na segunda etapa de testes.

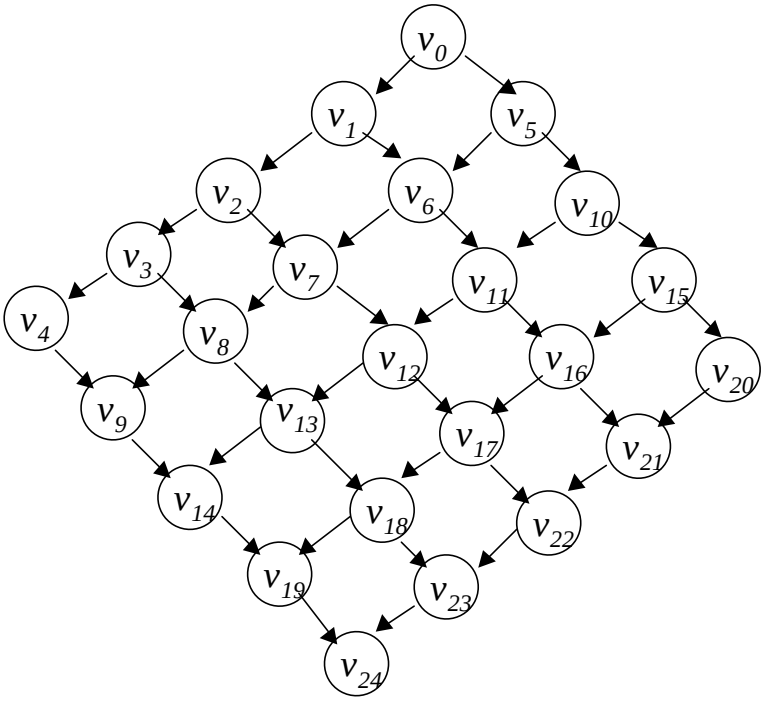

Figura 5.5: Grafo Diamante de 25 tarefas

Primeira Etapa de Experimentos

Na arquitetura de 8 processadores, se destacaram nos testes seis trios de priorida-

des que atingiram o maior percentual de casos com melhor *makespan*. O critério de desempate utilizado foi, *Nível Dinâmico e Caminho Crítico Dinâmico* (critério (B)), conforme pode ser observado na Figura 5.6. Avaliando a qualidade dos resultados gerados, a melhor foi de 1,005, observada para esses trios vencedores. A pior qualidade foi de 1,056, observada com a prioridade *Conível Dinâmico* sem desempates, o que indicou que priorizar tarefas pelo tempo mais cedo que podem começar não surtiu benefícios para o escalonamento nessa arquitetura. Aumentando-se o número de processadores para 12, o trio de prioridades de melhores *makespan* foi *Nível, Nível Dinâmico e Caminho Crítico Dinâmico*, conforme ilustra a Figura 5.7. Quando de disponibilizou mais processadores, 32 o comportamento guloso do *list scheduling* em procurar o melhor processador se acentuou. Com isso, ao se disponibilizar mais processadores mais eram utilizados. Além disso, outros trios de prioridades se tornaram evidentes ao atingir maior número de casos de melhor *makespan*, como pode ser observado na Figura 5.8, a prioridade *Caminho Crítico* se destacou ao atingir o maior percentual de casos quando não se adotaram critérios de desempate. Na última arquitetura com 64, composta por praticamente todos os processadores rápidos, se enfatizou mais ainda, o desempenho da prioridade *Caminho Crítico* sem adotar critérios de desempate. A propósito, a adoção de critérios de desempate, nesta arquitetura, não contribuiu para a diminuição do *makespan* indicando que as decisões de desempate não optaram por tarefas importantes para o escalonamento.

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	9,09%	1,046	81,81%	1,005	9,09%	1,048	9,09%	1,046
Conível	9,09%	1,046	81,81%	1,005	9,09%	1,048	9,09%	1,046
Caminho Crítico	36,36%	1,028	81,81%	1,005	18,18%	1,031	9,09%	1,046
ALAP	9,09%	1,046	81,81%	1,005	9,09%	1,048	9,09%	1,046
Nível Dinâmico	9,09%	1,046	81,81%	1,005	9,09%	1,048	9,09%	1,046
Conível Dinâmico	9,09%	1,056	18,18%	1,031	18,18%	1,031	18,18%	1,031
Caminho Crítico Dinâmico	36,36%	1,036	54,54%	1,023	54,54%	1,023	54,54%	1,023
ALAP Dinâmico	9,09%	1,046	81,81%	1,005	9,09%	1,048	9,09%	1,046

Figura 5.6: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos Diamantes

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	9,09%	1,084	54,54%	1,028	9,09%	1,084	9,09%	1,101
Conível	9,09%	1,084	9,09%	1,084	9,09%	1,084	18,18%	1,098
Caminho Crítico	36,36%	1,088	36,36%	1,088	36,36%	1,088	18,18%	1,098
ALAP	0,00%	1,084	0,00%	1,09	0,00%	1,084	18,18%	1,098
Nível Dinâmico	9,09%	1,084	9,09%	1,084	0,00%	1,091	9,09%	1,084
Conível Dinâmico	9,09%	1,084	9,09%	1,084	0,00%	1,09	9,09%	1,084
Caminho Crítico Dinâmico	36,36%	1,088	36,36%	1,088	0,00%	1,09	36,36%	1,088
ALAP Dinâmico	0,00%	1,084	0,00%	1,084	0,00%	1,09	0,00%	1,084

Figura 5.7: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos Diamantes

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	18,18%	1,206	9,09%	1,252	9,09%	1,207	18,18%	1,206
Conível	18,18%	1,206	9,09%	1,252	9,09%	1,207	18,18%	1,206
Caminho Crítico	81,81%	1,009	9,09%	1,252	0,00%	1,234	18,18%	1,206
ALAP	18,18%	1,206	9,09%	1,252	9,09%	1,207	18,18%	1,206
Nível Dinâmico	18,18%	1,206	9,09%	1,252	9,09%	1,207	18,18%	1,206
Conível Dinâmico	0,00%	1,235	0,00%	1,234	0,00%	1,234	0,00%	1,234
Caminho Crítico Dinâmico	0,00%	1,235	9,09%	1,252	9,09%	1,252	9,09%	1,252
ALAP Dinâmico	18,18%	1,206	9,09%	1,252	9,09%	1,207	18,18%	1,206

Figura 5.8: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos Diamantes

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	0,00%	1,264	0,00%	1,316	0,00%	1,264	0,00%	1,264
Conível	0,00%	1,264	0,00%	1,316	0,00%	1,264	0,00%	1,264
Caminho Crítico	100,00%	1	0,00%	1,316	0,00%	1,264	0,00%	1,264
ALAP	0,00%	1,264	0,00%	1,316	0,00%	1,264	0,00%	1,264
Nível Dinâmico	0,00%	1,264	0,00%	1,316	0,00%	1,264	0,00%	1,264
Conível Dinâmico	0,00%	1,264	0,00%	1,264	0,00%	1,264	0,00%	1,264
Caminho Crítico Dinâmico	0,00%	1,316	0,00%	1,316	0,00%	1,316	0,00%	1,316
ALAP Dinâmico	0,00%	1,264	0,00%	1,316	0,00%	1,264	0,00%	1,264

Figura 5.9: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos Diamantes

Na Figura 5.10 estão apresentados os percentuais médios obtidos pelas quatro arquiteturas analisadas. A prioridade em destaque foi *caminho crítico*, sem adotar critério nenhum para o desempate. O percentual apresentado foi destoante em relação aos outros 31 casos analisados: 63,63%. A Figura 5.11 ilustra os valores, tornando mais perceptível as diferenças observadas. A qualidade do resultado apresentado foi de 1,031 o que representa uma qualidade quase 5 vezes superior à apresentada pelas outras heurísticas concorrentes.

	S/Desempate	*Qualidade*	**ND/CCD**	*Qualidade*	**ConD/CCD**	*Qualidade*	**Conível/ALAP**	*Qualidade*
Nível	**9,09%**	*1,150*	**36,36%**	*1,150*	**6,82%**	*1,151*	**9,09%**	*1,154*
Conível	**9,09%**	*1,150*	**25,00%**	*1,164*	**6,82%**	*1,151*	**11,36%**	*1,154*
Caminho Crítico	**63,63%**	*1,031*	**31,82%**	*1,165*	**13,64%**	*1,154*	**11,36%**	*1,154*
ALAP	**6,82%**	*1,150*	**22,73%**	*1,166*	**4,55%**	*1,151*	**11,36%**	*1,154*
Nível Dinâmico	**9,09%**	*1,150*	**25,00%**	*1,164*	**4,55%**	*1,153*	**9,09%**	*1,150*
Conível Dinâmico	**4,55%**	*1,160*	**6,82%**	*1,153*	**4,55%**	*1,155*	**6,82%**	*1,153*
Caminho Crítico Dinâmico	**18,18%**	*1,169*	**25,00%**	*1,170*	**15,91%**	*1,170*	**25,00%**	*1,170*
ALAP Dinâmico	**6,82%**	*1,150*	**22,73%**	*1,164*	**4,55%**	*1,152*	**6,82%**	*1,150*

Figura 5.10: Médias considerando as quatro arquiteturas - Grafos Diamantes

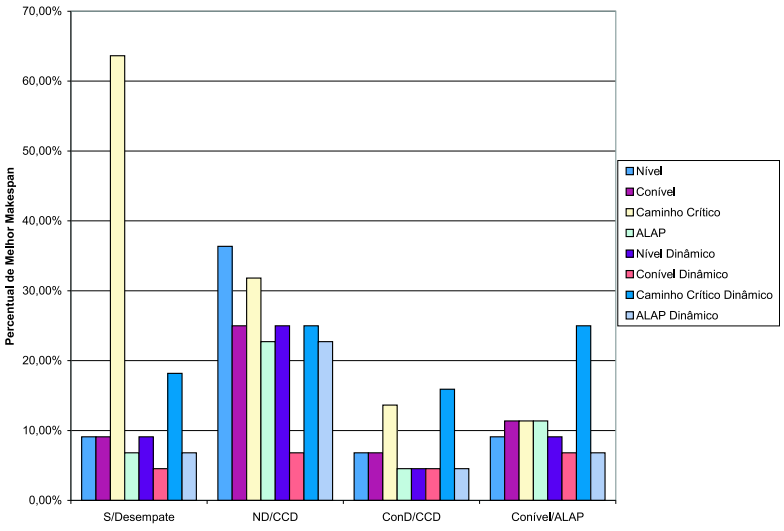

Figura 5.11: Médias por Critério de Desempate - Grafos Diamantes

Segunda Etapa de Testes

Na segunda etapa de testes foi analisado o escalonamento do grafo Diamante com 25 tarefas (Di25), na arquitetura de 12 processadores. Como dito, anteriormente, os testes foram direcionados uma vez que se utilizou algumas poucas versões com as prioridades e critérios de desempate mais promissores decorrentes das conclusões da primeira etapa de testes. Conforme apresenta a Figura 5.12(a) a versão da heurística de *list Scheduling* com prioridade: *Conível Dinâmico* sem adotar critérios de desempate gerou um escalonamento utilizando 3 processadores, sendo os dois mais rápidos P_2 e P_7 e o próximo mais rápido P_3; o *makespan* foi de 17 unidades. Na Figura 5.12(b) está ilustrado o escalonamento gerado pelas versões que utilizaram *Nível Dinâmico* e *Nível* sem adotar critérios de desempate (A), *caminho crítico* com critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico* (B) e *Nível Dinâmico* e critérios de desempate *Conível Dinâmico/Caminho Crítico Dinâmico* (C), todas as quatro versões construíram o mesmo escalonamento. Nos quatro casos os processadores P_2 e P_7, os mais rápidos, foram os únicos utilizados resultando num *makespan* de 17 unidades. Muitas outras combinações conduziram ao mesmo resultado usando os mesmos dois processadores, porém, não serão mencionadas. O que se pode afirmar é que se observou um número relativamente elevado de empates entre as versões, com a mesma distribuição das tarefas nos mesmos processadores. Na Figura 5.12(c) está ilustrado o escalonamento gerado pela versão LS com *Nível Dinâmico* e critério *Nível Dinâmico/Caminho Crítico Dinâmico*, a propósito é a versão de melhores resultados indicada pela média final dos experimentos realizados na primeira etapa de testes. Contudo, na classe diamante não foi econômica em termos da utilização de processadores. Para o mesmo *makespan* observado nas versões com que foi confrontada, nesta etapa de experimentos, utilizou 4 processadores.

Face aos inúmeros empates observados seria bastante difícil chegar a prioridade que mais contribuiu para atingir o melhor *makespan*. Realmente, foi de grande importância as baterias de testes realizadas na etapa 1, que permitiram concluir que a versão com prioridade *Caminho Crítico* sem critérios de desempate (A) foi a de maior sucesso, ou seja, a que atingiu o maior percentual de casos com menor *makespan* e com a melhor qualidade. O escalonamento produzido por essa versão está

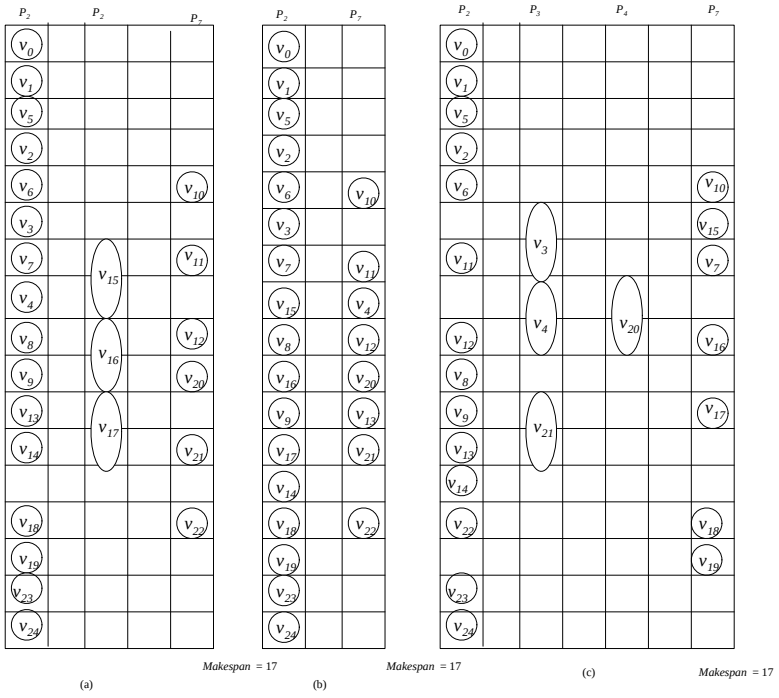

Figura 5.12: Exemplo do escalonamento do grafo Di25 (a) *LS* com *Conível Din.* e critério A (b) *LS* com *Nível, Nível Din.* e critério A; com *Nível Din.* e critério C; e *Cam. Crít.* com critério B (c) *LS* com *Nív. Din.* com critério B

ilustrado na Figura 5.13, observe o fato interessante que ocorre na distribuição das tarefas nos dois processadores, P_2 e P_7, mais rápidos da arquitetura disponibilizada: O grafo diamante é dividido em linhas de tarefas lineares. Assim, as cinco linhas que se podem observar no grafo, por exemplo, (v_0, v_1, v_2, v_2, v_4); (v_5, v_6, v_7, v_8, v_9), etc... têm todas as tarefas escalonadas seqüencialmente no mesmo processador. Essa distribuição é a considerada ótima, verificada com o algoritmo *Earliest Task First - ETF* que atinge escalonamento ótimos na classe diamante para processadores homogêneos. O *Makespan* é igual a 15 unidades e é o melhor observado nos testes para o grafo Di25.

5.4 Grafos Intree

Nesta classe de grafos, conhecidas como árvores binárias reversas completas, cada vértice ou tarefa tem dois predecessores imediatos, com exceção das tarefas de entrada, e apenas um sucessor imediato, com exceção da tarefa de saída do grafo.

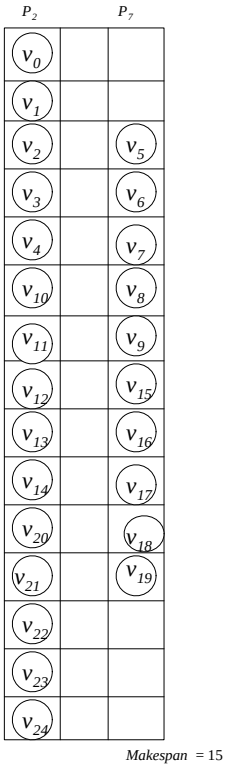

Figura 5.13: Melhor *makespan* gerado pela *LS* com prioridade *Caminho Crítico* sem critério de desempate (A) para o grafo Di25

A classe dos grafos utilizada, neste trabalho, está composta por 7 grafos, cada grafo composto por um número de tarefas que variam de 3 a 511 tarefas. No total, com esta classe, foram realizados 896 experimentos cujo objetivo é observar o percentual de casos onde a respectiva versão atingiu o melhor *makespan* e a degradação em relação ao melhor *makespan* produzido na respectiva instância de testes. A Figura 5.14 ilustra a topologia do grafo *intree* In31 composto por 31 tarefas distribuídas em 5 níveis topológicos utilizado, posteriormente, na segunda etapa de testes para ilustrar escalonamentos obtidos na arquitetura de 12 processadores.

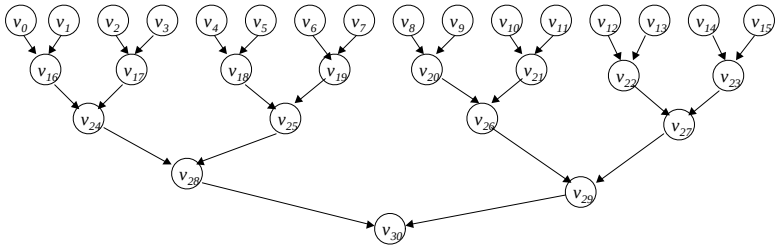

Figura 5.14: Grafo Intree com 31 tarefas

Primeira Etapa de Testes

Nos experimentos realizados na arquitetura de 8 processadores dois critérios de desempate se destacaram devido aos iguais percentuais de melhores *makespans* atingidos, conforme pode ser observado na Figura 5.15. Na mesma figura, também, é possível observar que as versões que utilizaram o critério de desempate *Conível Dinâmico/Caminho Crítico Dinâmico* foram as que apresentaram, na média, a menor degradação no *makespan* em relação ao melhor valor verificado. Em relação ao maior percentual de casos onde se atingiu o melhor *makespan*, as versões que utilizaram o critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico* (B) foram, em média, as de maiores percentuais. Entretanto, através da qualidade dos escalonamentos gerados é possível verificar que as prioridades que utilizaram o critério de desempate de *Conível Dinâmico e Caminho Crítico Dinâmico* (C) foram as de menor degradação em relação ao melhor *makespan*. Na Figura 5.16 estão apresentados os percentuais atingidos nos 32 testes realizados, na arquitetura de 12 processadores. Nesses testes observou-se que as quatro prioridades calculadas de forma dinâmica em conjunto com a dupla de desempate *Conível Dinâmico e Caminho Crítico Dinâmico* se destacaram sobre as demais atingindo 100% de casos onde apresentaram o melhor *makespam*. O trio *ALAP e Nível Dinâmico/Caminho Crítico Dinâmico* também atingiu o percentual de 100%. A Figura 5.17 apresenta os percentuais atingidos pelas heurísticas utilizando uma arquitetura com 32 processadores. Pode ser observado que um maior número de prioridades utilizando o critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* atingiram o melhor *makespan*. A degradação máxima apresentada foi no máximo de 1,4%. Na arquitetura de 64 processadores cujos testes estão apresentados na Figura 5.18 pode ser observado que a maior parte das versões atingiram percentual máximo. O que indica, que nos escalonamentos gerados, as heurísticas fizeram uma melhor utilização dos processadores disponíveis. Na média, pelas quatro arquiteturas, a versão que utilizou a prioridade *ALAP* com critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* é a de maior percentual de casos onde foi atingido o melhor *makespan* 96,43%. Conforme está apresentado na Figura 5.19 e ilustrado no gráfico da Figura 5.20.

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	57,14%	1,013	85,71%	1,014	85,71%	1,004	57,14%	1,013
Conível	57,14%	1,013	85,71%	1,014	85,71%	1,004	57,14%	1,013
Caminho Crítico	57,14%	1,013	85,71%	1,014	28,57%	1,016	57,14%	1,013
ALAP	57,14%	1,013	85,71%	1,014	85,71%	1,004	57,14%	1,013
Nível Dinâmico	57,14%	1,013	85,71%	1,014	85,71%	1,004	57,14%	1,013
Conível Dinâmico	28,57%	1,016	28,57%	1,016	28,57%	1,016	28,57%	1,016
Caminho Crítico Dinâmico	28,57%	1,057	28,57%	1,057	28,57%	1,057	28,57%	1,057
ALAP Dinâmico	57,14%	1,013	85,71%	1,014	85,71%	1,004	57,14%	1,013

Figura 5.15: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos Intree

in12	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	71,42%	1,006	71,42%	1,006	71,42%	1,006	57,14%	1,024
Conível	71,42%	1,006	71,42%	1,006	71,42%	1,006	57,14%	1,024
Caminho Crítico	85,71%	1,004	85,71%	1,004	85,71%	1,004	57,14%	1,024
ALAP	85,71%	1,004	100,00%	1	85,71%	1,004	57,14%	1,024
Nível Dinâmico	71,42%	1,006	71,42%	1,006	100,00%	1	71,42%	1,006
Conível Dinâmico	71,42%	1,006	71,42%	1,006	100,00%	1	71,42%	1,006
Caminho Crítico Dinâmico	85,71%	1,004	85,71%	1,004	100,00%	1	85,71%	1,004
ALAP Dinâmico	85,71%	1,004	85,71%	1,004	100,00%	1	85,71%	1,004

Figura 5.16: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos Intree

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	71,42%	1,008	100,00%	1	85,71%	1,003	71,42%	1,008
Conível	71,42%	1,008	100,00%	1	85,71%	1,003	71,42%	1,008
Caminho Crítico	71,42%	1,008	100,00%	1	71,42%	1,008	71,42%	1,008
ALAP	71,42%	1,008	100,00%	1	85,71%	1,003	71,42%	1,008
Nível Dinâmico	71,42%	1,008	100,00%	1	85,71%	1,003	71,42%	1,008
Conível Dinâmico	71,42%	1,008	71,42%	1,008	71,42%	1,008	71,42%	1,008
Caminho Crítico Dinâmico	71,42%	1,014	71,42%	1,014	71,42%	1,014	71,42%	1,014
ALAP Dinâmico	71,42%	1,008	100,00%	1	85,71%	1,003	71,42%	1,008

Figura 5.17: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos Intree

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	85,71%	1,009	100,00%	1	100,00%	1	85,71%	1,009
Conível	85,71%	1,009	100,00%	1	100,00%	1	85,71%	1,009
Caminho Crítico	85,71%	1,009	100,00%	1	100,00%	1	85,71%	1,009
ALAP	85,71%	1,009	100,00%	1	100,00%	1	85,71%	1,009
Nível Dinâmico	85,71%	1,009	100,00%	1	100,00%	1	85,71%	1,009
Conível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP Dinâmico	85,71%	1	100,00%	1	100,00%	1	85,71%	1,009

Figura 5.18: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos Intree

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	71,42%	*1,009*	89,28%	*1,005*	85,71%	*1,003*	67,85%	*1,014*
Conível	71,42%	*1,009*	89,28%	*1,005*	85,71%	*1,003*	67,85%	*1,014*
Caminho Crítico	75,00%	*1,009*	92,86%	*1,005*	71,43%	*1,007*	67,85%	*1,014*
ALAP	75,00%	*1,009*	96,43%	*1,004*	89,28%	*1,003*	67,85%	*1,014*
Nível Dinâmico	71,42%	*1,009*	89,28%	*1,005*	92,86%	*1,002*	71,42%	*1,009*
Conível Dinâmico	67,85%	*1,008*	67,85%	*1,008*	75,00%	*1,006*	67,85%	*1,008*
Caminho Crítico Dinâmico	71,43%	*1,019*	71,43%	*1,019*	75,00%	*1,018*	71,43%	*1,019*
ALAP Dinâmico	75,00%	*1,006*	92,86%	*1,005*	92,86%	*1,002*	75,00%	*1,009*

Figura 5.19: Médias considerando as quatro arquiteturas - Grafos Intree

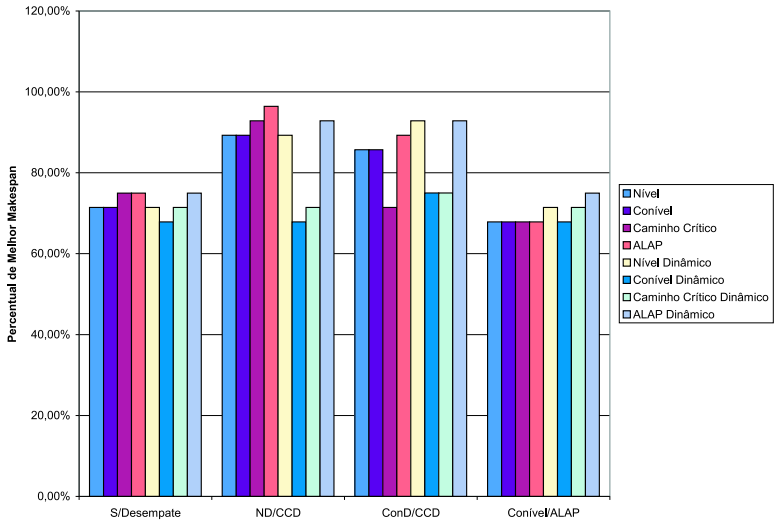

Figura 5.20: Médias por critério de desempate - Grafos Intree

Segunda Etapa de Testes

Na segunda etapa de testes foi analisado o escalonamento do grafo *Intree* com 31 tarefas (In31), na arquitetura de 12 processadores. Os testes foram, também, direcionados uma vez que se utilizou algumas poucas versões com as prioridades e critérios de desempate mais promissores decorrentes das conclusões da primeira etapa de testes. Fato interessante observado foi a ocorrência de inúmeras distribuições semelhantes de tarefas, principalmente, com as tarefas pertencentes aos níveis topológicos mais baixos do grafo. Isto foi devido a própria estrutura dos grafos *intree*. As tarefas pertencentes a níveis menores são em maior número, e todas (do mesmo nível) possuem a mesma prioridade estática, e essa prioridade é superior a prioridade das tarefas pertencentes a níveis topológicos maiores. Assim, as tarefas de mesmo nível topológico com mesmas prioridades eram escalonadas de acôrdo com a ordem de entrada na lista de tarefas livres. Quando as prioridades eram calculadas de forma dinâmica, ou seja, os valores das prioridades eram recalculadas a cada passo do escalonamento, ocorreram distribuições, ligeiramente diferentes, nas tarefas pertencentes aos níveis topológicos mais altos. Foram escolhidos alguns exemplos para ilustrar a distribuição das tarefas do grafo *intree31* na arquitetura de 12 processadores. Na Figura 5.21 está apresentado o escalonamento utilizando a versão de *List Scheduling* com prioridade *Nível Dinâmico* sem critério de desempate e, também, da versão de *List Scheduling* com critério de desempate *Conível Dinâmico/Caminho Crítico Dinâmico*. Ambas as versões utilizaram 8 dos 12 processadores disponibilizados, sendo que a cada passo o processador que permitia o menor tempo de término, para a tarefa candidata ao escalonamento, era o preferido. Para alocar a primeira tarefa v_0 foi determinado o processador P_2, a seguir a próxima a ser escalonada é a tarefa v_1, que é designada para o processador P_7. Assim, sucessivamente, vão sendo designadas as 15 tarefas pertencentes ao nível topológico de $ordem = 1$, de acordo com a ordem de chegada na lista de tarefas livres. O *makespan* é de 12 unidades.

Na Figura 5.22 está ilustrado o escalonamento produzido pela versão que utilizou a prioridade *Nível Dinâmico* e critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico* (indicada pelos testes da primeira etapa) e, também, produzido pela versão que utilizou a prioridade *ALAP* e critério de desempate *Nível Dinâ-*

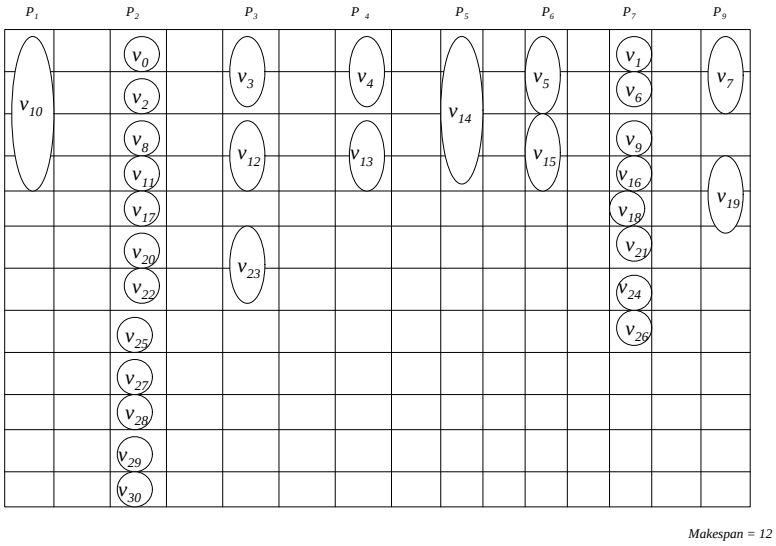

Figura 5.21: Escalonamento do grafo *Intree31* na arquitetura com 12 processadores heterogêneos utilizando a versão de *LS* com prioridade *Nív. Din.* sem critério de desempate e a versão com prioridade *Nív. Din.* com critério C

mico/Caminho Crítico Dinâmico que se destacou com 100% dos casos onde se atingiu o melhor desempenho, conforme apresentou a Figura 5.16. Pode ser observado que se atingiu o mesmo *makespan* de 12 unidades e que as alterações de designação das tarefas aos processadores ocorrem entre as tarefas pertencentes ao último nível topológico e pertencentes a sub árvore da direita, últimas a serem escalonadas. Essa distribuição permitiu, em relação anteriormente apresentada, um melhor balanceamento de carga pelos processadores.

5.5 Grafos Outtree

Nesta classe de grafos, conhecidas como árvores binárias completas, cada vértice ou tarefa tem apenas um predecessor imediato, com exceção da tarefa de entrada (chamada raiz da árvore), e ainda, possui dois sucessores imediatos, com exceção das tarefa de saída do grafo (chamadas de folhas do grafo). Uma árvore binária completa com n tarefas tem $(n+1) \div 2$ folhas, altura de $log(n+1)-1$ e $log(n+1)$ níveis. A classe utilizada, neste trabalho, está composta por 8 grafos, cada grafo composto por um número de tarefas que variam de 3 a 511 tarefas. No total, com esta classe, foram realizados 1024 experimentos cujo objetivo é observar o percentual

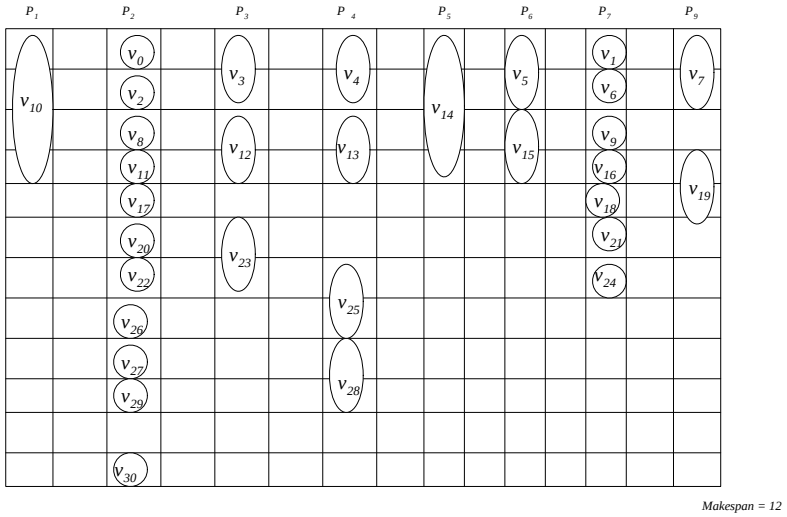

Figura 5.22: Escalonamento do grafo *Intree31* na arquitetura com 12 processadores heterogêneos gerado pela versão de *LS* com prioridade *Nív. Din.* e critério de desempate de *Nív. Din./ Cam. Crí. Din.* (B) e pela versão *LS* com prioridade *ALAP* e critério de desempate B

de casos onde a respectiva versão atingiu o melhor *makespan* e a degradação em relação ao melhor *makespan* produzido na respectiva instância de testes. Na primeira etapa de testes foram aplicadas diferentes combinações de prioridades e critérios de desempate. Essas combinações podem alterar a distribuição das tarefas em um ambiente paralelo, os resultados significativos são descritos a seguir.

Primeira Etapa de testes

Nos testes realizados na arquitetura com 8 processadores, os valores atingidos por seis prioridades quando utilizadas em conjunto com os critérios de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* foram os mais significativos. Conforme descreve a tabela da Figura 5.23. Disponibilizando mais processadores, com a utilização da arquitetura de 12 processadores, foi verificado que um maior número de trios de prioridades atingem melhores escalonamentos. Exceção ocorreu para o grupo das prioridades estáticas quando em conjunto com a dupla de critérios de desempate *Conível/ALAP* que apresentaram, cada uma 75%, e qualidade 1,005. Os valores estão apresentados na Figura 5.24. Na arquitetura de 32 processadores todos os valores de *makespan* foram coincidentes, em todos os testes realizados, 100%, conforme ilustra a Figura 5.25. Na arquitetura de 64 processadores, também, foi ve-

rificada essa totalidade de valores. Apesar da disponibilidade dos processadores nem todos os grafos fizeram uso deles devido ao número de tarefas que apresentavam. Somente os Grafos Out255 e Out511 utilizaram, respectivamente, 61 e 62 processadores. A Figuras 5.26 ilustra os percentuais atingidos utilizando a arquitetura de 64 processadores. Na média pelas arquiteturas, as prioridades *nível, conível, caminho crítico, ALAP, nível dinâmico, e ALAP dinâmico* em conjunto com o critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* se destacam atingindo 100%. Conforme pode ser observado na Figura 5.27 e no gráfico ilustrado na Figura 5.28

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	37,50%	1,021	100,00%	1	37,50%	1,021	37,50%	1,021
Conível	37,50%	1,021	100,00%	1	37,50%	1,021	37,50%	1,021
Caminho Crítico	37,50%	1,021	100,00%	1	25,00%	1,025	37,50%	1,021
ALAP	37,50%	1,021	100,00%	1	37,50%	1,021	37,50%	1,021
Nível Dinâmico	37,50%	1,021	100,00%	1	37,50%	1,021	37,50%	1,021
Conível Dinâmico	25,00%	1,025	25,00%	1,025	25,00%	1,025	25,00%	1,025
Caminho Crítico Dinâmico	75,00%	1,013	75,00%	1,013	75,00%	1,013	75,00%	1,013
ALAP Dinâmico	37,50%	1,01	100,00%	1	37,05%	1,021	37,50%	1,021

Figura 5.23: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos Outtree

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	100,00%	1	100,00%	1	100,00%	1	75,00%	1,005
Conível	100,00%	1	100,00%	1	100,00%	1	75,00%	1,005
Caminho Crítico	100,00%	1	100,00%	1	100,00%	1	75,00%	1,005
ALAP	100,00%	1	100,00%	1	100,00%	1	75,00%	1,005
Nível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Conível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1

Figura 5.24: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos Outtree

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Conível	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Nível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Conível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1

Figura 5.25: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos Outtree

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Conível	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Nível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Conível Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
Caminho Crítico Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1
ALAP Dinâmico	100,00%	1	100,00%	1	100,00%	1	100,00%	1

Figura 5.26: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos Outtree

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	84,38%	*1,005*	100,00%	*1,000*	84,38%	*1,005*	78,13%	*1,007*
Conível	84,38%	*1,005*	100,00%	*1,000*	84,38%	*1,005*	78,13%	*1,007*
Caminho Crítico	84,38%	*1,005*	100,00%	*1,000*	81,25%	*1,006*	78,13%	*1,007*
ALAP	84,38%	*1,005*	100,00%	*1,000*	84,38%	*1,005*	78,13%	*1,007*
Nível Dinâmico	84,38%	*1,005*	100,00%	*1,000*	84,38%	*1,005*	84,38%	*1,005*
Conível Dinâmico	81,25%	*1,006*	81,25%	*1,006*	81,25%	*1,006*	81,25%	*1,006*
Caminho Crítico Dinâmico	93,75%	*1,003*	93,75%	*1,003*	93,75%	*1,003*	93,75%	*1,003*
ALAP Dinâmico	84,38%	*1,003*	100,00%	*1,000*	84,26%	*1,005*	84,38%	*1,005*

Figura 5.27: Médias considerando as quatro Arquiteturas - Grafos Outtree

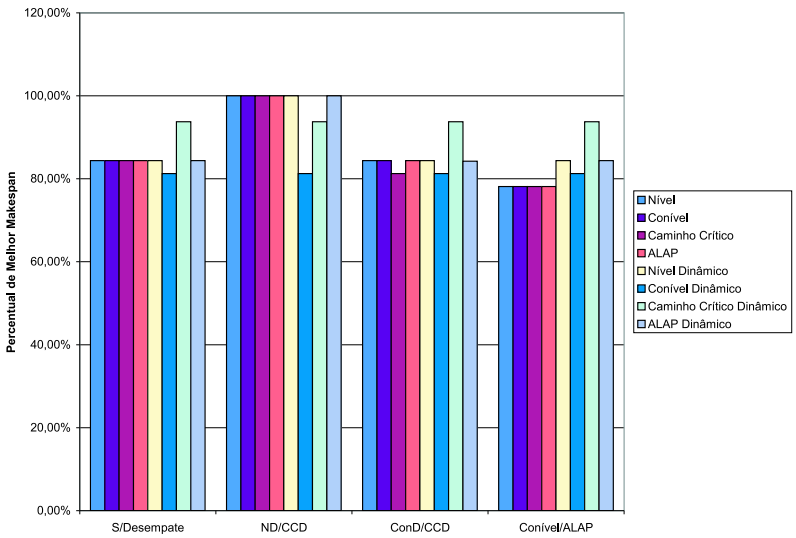

Figura 5.28: Médias por critérios de desempate - Grafos Outtree

Segunda Etapa de Testes

Na segunda etapa de testes foi analisado o escalonamento do grafo *Outtree* com 31 tarefas (Out31), na arquitetura de 12 processadores. A Figura 5.29 ilustra a topologia da *Outtree31* composto por 5 níveis topológicos. Em cada nível topológico existem 2^{k-1} tarefas, onde k é o (número do) nível topológico ou ordem topológica do grafo. O número total de tarefas dos grafos *outtree* podem ser encontrados através da seguinte fórmula: $2^k - 1$, ou seja neste grafo, onde $k = 5$ existem 31 tarefas.

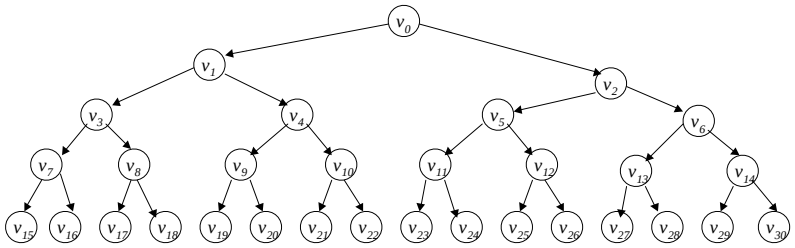

Figura 5.29: Grafo Outtree de 31 tarefas

Os testes foram, também, direcionados uma vez que se utilizou algumas poucas versões com as prioridades e critérios de desempate mais promissores decorrentes das conclusões da primeira etapa de testes. Na topologia do grafo *outtree* as tarefas pertencentes a níveis menores são em menor número, e todas (do mesmo nível) possuem a mesma prioridade estática. Assim, as prioridades das tarefas de níveis mais baixos é superior a prioridade das tarefas pertencentes a níveis topológicos maiores, onde existem maior número de tarefas. Assim, as tarefas de mesmo nível topológico (com mesmas prioridades) eram escalonadas de acôrdo com a ordem de entrada na lista de tarefas livres. Quando as prioridades eram calculadas de forma dinâmica, ou seja, os valores das prioridades eram recalculadas a cada passo do escalonamento, ocorreram distribuições, ligeiramente diferentes, nas tarefas pertencentes aos níveis topológicos mais altos. Foram escolhidos alguns exemplos para ilustrar a distribuição das tarefas do grafo *outtree31* na arquitetura de 12 processadores. Na Figura 5.30 está apresentado o escalonamento com *makespan* de 12 unidades, utilizando a versão de *List Scheduling* com prioridade *Nível Dinâmico* com critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico*. Pode ser verificado que foram utilizados 7 processadores sendo primeiro utilizados os mais rápidos, estando esses ocupados a heurística escolhia os próximos mais rápidos e que permitiam a

tarefa candidata ao escalonamento terminar o mais breve possível. A característica de paralelizar ao máximo tarefas independentes, própria das heurísticas de *List Scheduling*, ditas gulosas, pode ser observada. Por exemplo, as tarefas v_3 e v_4 de mesmo nível topológico são escalonadas no mesmo tempo, nos processadores mais rápidos P_2 e P_7, idem para as tarefas v_5 e v_6. Da mesma forma, para as tarefas v_{11} e v_{12}, que são paralelizadas para terminar no mesmo tempo. Na designação das tarefas aos processadores do sistema paralelo primeiro são utilizados os processadores mais rápidos, posteriormente, são utilizados os próximos mais rápidos e assim sucessivamente.

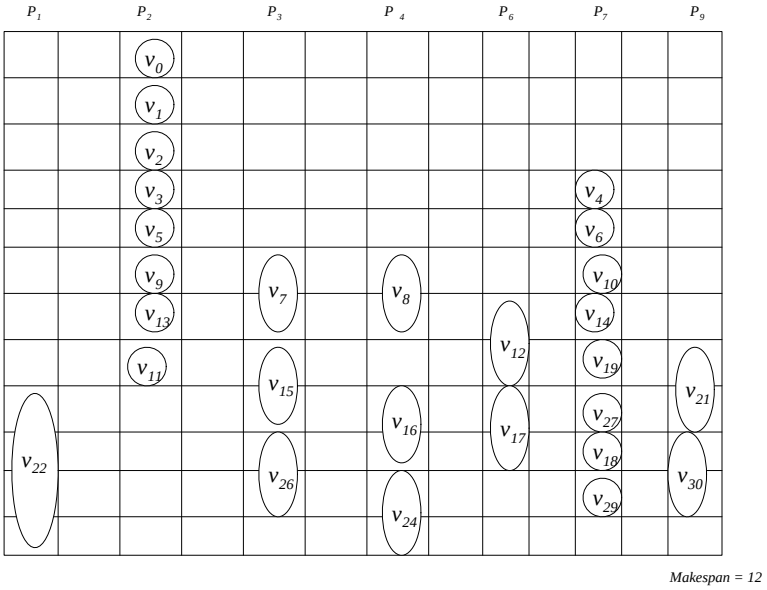

Figura 5.30: Escalonamento do *Outtree31* com LS prioridade *Nível Dinâmico* e critério de desempate *Nível Dinâmico/CaminhoCrítico Dinâmico*

Na Figura 5.31 está apresentado o escalonamento do grafo *outtree* de 31 tarefas realizado pela versão de *List Scheduling* que utilizou a prioridade *Nível Dinâmico* e critério de desempate *Conível Dinâmico/Caminho Crítico Dinâmico*. Nota-se que houve uma economia de um processador, quando comparado com o caso anterior. Com isso, essa versão contribuiu mais para enfatizar o principal objetivo de uma heurística de escalonamento que consiste em balancear a maximização da exploração ao grau de paralelismo com a minimização do custo de comunicação para que, assim, se verifique a minimização do *makespan*. Foram utilizados 6 processadores, os mais rápidos, da arquitetura de 12 processadores disponibilizada. Pode ser observado que

a maioria das tarefas (paralelas) pertencentes a um mesmo nível foram paralelizadas e terminam no mesmo tempo.

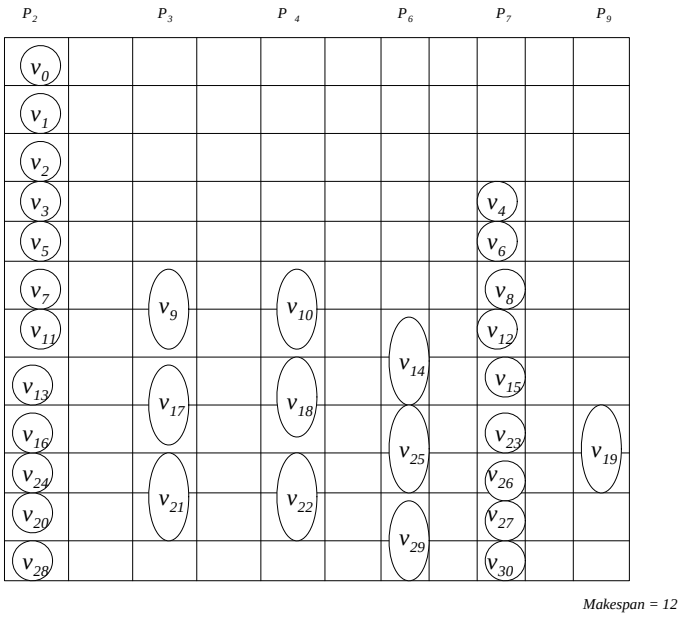

Figura 5.31: Escalonamento do *Outtree31* com LS prioridade *Nível Dinâmico* e critério de desempate *Conível Dinâmico/Caminho Crítico Dinâmico*

Mais um escalonamento gerado pela versão que utilizou a prioridade *Nível Dinâmico* sem critério de desempate e pela versão que utilizou a prioridade *Nível Dinâmico* com critério de desempate *Conível/ALAP* está apresentado na Figura 5.32. O *makespan* é de 12 unidades e poucas foram as alterações observadas em relação a distribuição das tarefas.

5.6 Grafos Randômicos

Nesta classe foram utilizados 21 grafos gerados aleatoriamente com a finalidade de representar qualquer estrutura irregular. Os grafos são compostos por um número de tarefas que variam desde 80 a 546 tarefas. Foram realizados 2688 experimentos com esta classe de grafos. Os resultados observados na segunda etapa de testes não serão ilustrados devido ao elevado número de tarefas do menor grafo desta classe o *Ran80* com 80 tarefas. Assim, apenas a descrição de alguns experimentos com o *Ran80* será apresentada.

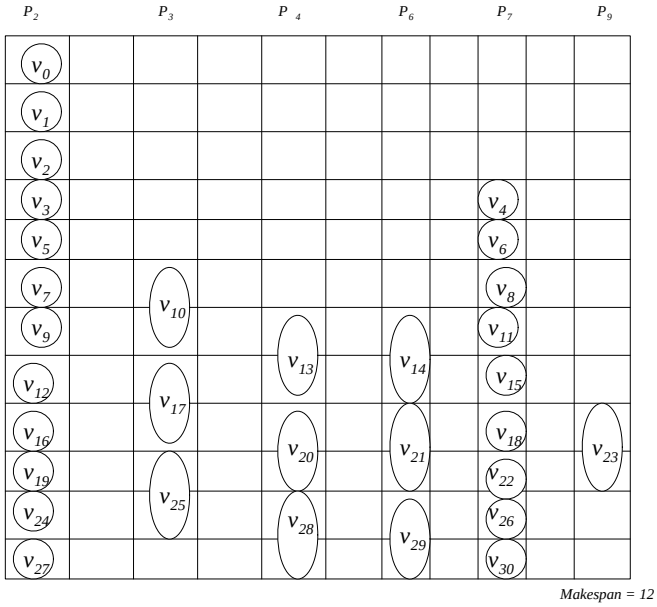

Figura 5.32: Escalonamento do grafo *Outtree31* em 12 processadores pela versão *List Scheduling* com prioridade de *Nível Dinâmico* sem critério de desempate e pela versão de *List Scheduling* com mesma prioridade e critério D

Primeira Etapa de Testes

Os grafos randômicos foram os que mais utilizaram, em comparação às outras classes, um maior número de processadores conforme estes iam sendo disponibilizados na arquitetura adotada na instância de testes. A Figura 5.33 apresenta os valores obtidos utilizando a arquitetura de 8 processadores. Pode ser observado que os trios de maior sucesso em atingir percentuais de melhor *makespan* ocorrem sem a necessidade de se incorporar critérios de desempate. Entretanto, é nessa mesma bateria de testes que, também, se verifica o pior sucesso e as maiores degradações em relação ao melhor *makespan*. Nos experimentos realizados na arquitetura de 12 processadores, cujos valores estão descritos na Figura 5.34 é possível perceber que que a prioridade *conível* calculada, na forma estática e na forma dinâmica, atingiu os melhores percentuais em distintos critérios com ou sem desempate incorporado, mantendo a qualidade de 1,002 em todos os casos. Na arquitetura de 32 processadores, observou-se o destaque de resultados produzidos, pelas prioridades *Nível* e *ALAP*, calculadas de forma estática e dinâmica, quando utilizadas sem a adoção de critérios de desempate e quando em conjunto com a dupla de desempate *Conível Dinâmico e Caminho Crítico Dinâmico*. A qualidade dos resultados foi a

mesma em todos os casos: 1,026. A Figura 5.35 mostra os percentuais atingidos. Na arquitetura de 64 processadores, as mesmas prioridades agora em conjunto com a dupla *Nível Dinâmico e Caminho Crítico Dinâmico* atingiram o melhor *makespan* no maior número de casos, com qualidade de 1,003. Conforme apresenta a Figura 5.36. Considerando a média pelas quatro arquiteturas utilizadas, como mostra a Figura 5.37 a prioridade *Nível Dinâmico* em conjunto com a dupla de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* foi o que atingiu o melhor *makespan* no maior número de casos: 70,23%, com qualidade de 1,013. Na média, as prioridades *Conível* e *Conível Dinâmico* foram em conjunto com distintos critérios as de resultados mais insatisfatórios, atingindo apenas em 3% dos casos o melhor *makespan*. A Figura 5.38 apresenta o gráfico com os percentuais, médios, de melhor *makespan*.

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	90,47%	1,001	52,38%	1,01	71,42%	1,005	71,42%	1,004
Conível	0,00%	1,062	0,00%	1,062	0,00%	1,073	0,00%	1,062
Caminho Crítico	0,00%	1,151	0,00%	1,135	0,00%	1,137	0,00%	1,133
ALAP	90,47%	1,001	52,38%	1,01	71,42%	1,005	71,42%	1,004
Nível Dinâmico	90,47%	1,001	52,38%	1,01	71,42%	1,005	71,42%	1,004
Conível Dinâmico	4,76%	1,066	4,76%	1,057	4,76%	1,057	9,52%	1,061
Caminho Crítico Dinâmico	0,00%	1,146	0,00%	1,13	0,00%	1,13	0,00%	1,129
ALAP Dinâmico	90,47%	1,001	52,38%	1,01	71,42%	1,005	71,42%	1,004

Figura 5.33: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos Randômicos

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	85,71%	1,004	0,00%	1,138	85,71%	1,004	0,00%	1,147
Conível	90,47%	1,002	0,00%	1,124	90,47%	1,002	0,00%	1,133
Caminho Crítico	19,04%	1,029	0,00%	1,126	19,04%	1,029	0,00%	1,132
ALAP	80,95%	1,004	0,00%	1,124	80,95%	1,004	0,00%	1,132
Nível Dinâmico	0,00%	1,115	85,71%	1,004	0,00%	1,125	85,71%	1,004
Conível Dinâmico	0,00%	1,105	90,47%	1,002	0,00%	1,114	90,47%	1,002
Caminho Crítico Dinâmico	0,00%	1,106	19,04%	1,029	0,00%	1,106	19,04%	1,029
ALAP Dinâmico	0,00%	1,127	80,95%	1,004	0,00%	1,106	80,95%	1,004

Figura 5.34: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos Randômicos

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	76,19%	1,026	47,61%	1,035	76,19%	1,26	66,66%	1,042
Conível	0,00%	1,271	0,00%	1,253	0,00%	1,259	0,00%	1,254
Caminho Crítico	9,52%	1,125	14,28%	1,112	9,52%	1,104	9,52%	1,102
ALAP	76,19%	1,026	47,61%	1,035	76,19%	1,026	66,66%	1,042
Nível Dinâmico	76,19%	1,026	47,61%	1,035	76,19%	1,026	66,66%	1,042
Conível Dinâmico	0,00%	1,273	0,00%	1,234	0,00%	1,234	0,00%	1,254
Caminho Crítico Dinâmico	19,04%	1,1105	19,04%	1,099	19,04%	1,099	19,04%	1,098
ALAP Dinâmico	76,19%	1,026	47,61%	1,035	76,19%	1,026	66,66%	1,042

Figura 5.35: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos Randômicos

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	9,52%	1,162	95,23%	1,003	9,52%	1,185	9,52%	1,197
Conível	0,00%	1,271	4,76%	1,306	4,76%	1,296	4,76%	1,305
Caminho Crítico	33,33%	1,074	28,57%	1,076	23,80%	1,083	23,80%	1,083
ALAP	9,52%	1,162	95,23%	1,003	9,52%	1,185	9,52%	1,197
Nível Dinâmico	9,52%	1,162	95,23%	1,003	9,52%	1,185	9,52%	1,197
Conível Dinâmico	0,00%	1,278	4,76%	1,244	4,76%	1,244	4,76%	1,266
Caminho Crítico Dinâmico	33,33%	1,077	33,33%	1,067	33,33%	1,067	38,09%	1,06
ALAP Dinâmico	9,52%	1,162	95,23%	1,003	9,52%	1,185	9,52%	1,197

Figura 5.36: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos Randômicos

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	65,47%	*1,048*	48,81%	*1,047*	60,71%	*1,114*	36,90%	*1,098*
Conível	22,62%	*1,152*	1,19%	*1,186*	23,81%	*1,158*	1,19%	*1,189*
Caminho Crítico	15,47%	*1,095*	10,71%	*1,112*	13,09%	*1,088*	8,33%	*1,113*
ALAP	64,28%	*1,048*	48,81%	*1,043*	59,52%	*1,055*	36,90%	*1,094*
Nível Dinâmico	44,05%	*1,076*	70,23%	*1,013*	39,28%	*1,085*	58,33%	*1,062*
Conível Dinâmico	1,19%	*1,181*	25,00%	*1,134*	2,38%	*1,162*	26,19%	*1,146*
Caminho Crítico Dinâmico	13,09%	*1,110*	17,85%	*1,081*	13,09%	*1,101*	19,04%	*1,079*
ALAP Dinâmico	44,05%	*1,079*	69,04%	*1,013*	39,28%	*1,081*	57,14%	*1,062*

Figura 5.37: Médias considerando as quatro arquiteturas - Grafos Randômicos

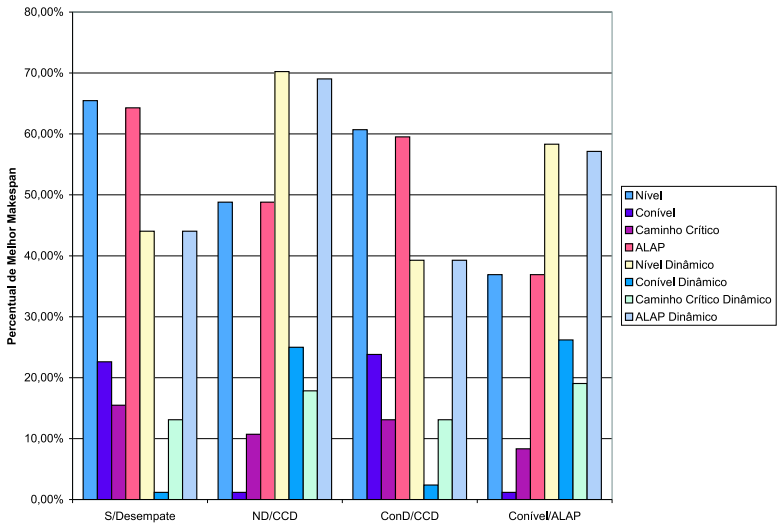

Figura 5.38: Médias por critérios de desempate - Grafos Randômicos

Segunda Etapa de Testes

A versão de *List Scheduling* que se destacou nos experimentos realizados na primeira etapa, considerando as quatro arquiteturas e os 21 grafos da classe, foi a de *Nível Dinâmico* com o critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico*. Entretanto, na arquitetura de 12 processadores em nenhum dos 21 grafos conseguiu atingir o melhor ou igual *makespan*. Nessa arquitetura quatro versões se destacaram: a que utilizou a prioridade *Cónível* com critério (A) e (C) e a versão que

utilizou a prioridade *Conível Dinâmico* com critério de desempate (B) e (D), todas apresentaram o mesmo percentual de 90,47% dos casos com o melhor *makespan* e para o mesmo valor de degradação em relação ao melhor *makespan* 1,002. Mesmo assim, para o grafo escolhido o *Ran80* os valores de *makespan* cada uma das quatro versões foi diferente. A versão que atingiu o melhor *makespan* de 23 unidades, utilizou 10 processadores deixando de utilizar os dois mais lentos P_8 ($h(p) = 8$) e P_{11} ($h(p) = 16$). As outras três versões, também, não utilizaram esses mesmos dois processadores, porém geraram diferentes e piores *makespan*. A versão que utilizou *Conível* com critério (C), apresentou *makespan* de 26 unidades. A versão com mesma prioridade e sem critério de desempate (A) gerou um escalonamento com *makespan* de 25 unidades. A versão que utilizou a prioridade *Conível Dinâmico* com critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* atingiu *makespan* de 24 unidades. Fato interessante a ser mencionado é que as prioridades, neste caso *conível*, quando calculada de forma dinâmica, realmente mais contribuiram para a minimização do *makespan*.

5.7 Grafos Irregulares

Esta classe é composta por grafos como o próprio nome indica, de estrutura irregular. A característica principal dos grafos desta topologia é apresentarem tarefas e arestas com pesos de grande valor. Nos experimentos foram utilizados 7 grafos nos quais as tarefas variam de 7 a 41. A Figura 5.39 apresenta um dos grafos irregulares utilizados no testes e escolhido como exemplo para ilustrar escalonamentos gerados na segunda etapa de testes. Considerando a granulosidade desta classe de grafos existe uma tendência a caracterizá-los como grafos de granulosidade fina. Isto porque o somatório dos custos de comunicação excedem os custos de computação. Para esta classe de grafos foram realizados 896 experimentos onde se observaram o percentual de casos onde a respectiva versão de *List Scheduling* atingiu o melhor *makespan* e o valor da qualidade do *makespan* gerado.

Primeira Etapa de Testes

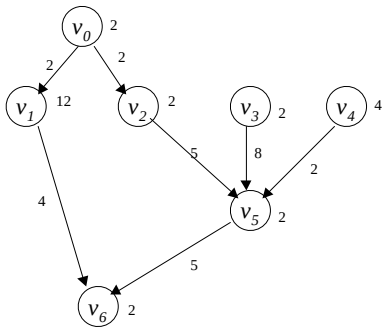

Figura 5.39: Grafo Irregular de 7 tarefas

Os experimentos realizados com a classe de grafos irregulares, na arquitetura de 8 processadores estão ilustrados na Figura 5.40. Quatro prioridades se destacam empatadas no mesmo percentual de casos onde atingiram o melhor *makespan*, 85,71%. As prioridades são *Nível* e *ALAP*, calculadas de forma estática e dinâmica, utilizadas em conjunto com a dupla de critérios de desempate *Nível Dinâmico e Caminho Crítico Dinâmico*. Com respeito a qualidade dos resultados produzidos pelas heurísticas com essas prioridades todas empataram em 1,015. Os experimentos realizados na arquitetura de 12 processadores são mostrados na Figura 5.41, distintos trios de prioridades atingem o valor mais significativo do percentual de casos onde atingiram o melhor *makespan*. As quatro prioridades calculadas de forma estática, quando em conjunto com o critério de desempate de *Conível e ALAP* foram as que apresentaram resultados de melhor qualidade 1,069, em relação às outras. Na arquitetura de 32 processadores, as prioridades *Nível* e *ALAP*, calculadas de forma estática e dinâmica, em conjunto com a dupla de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* foram as que atingiram o melhor *makespan* em 71,42% dos escalonamentos gerados. A qualidade dos resultados foi igual para todas as quatro prioridades. Os valores estão apresentados na Figura 5.42. Com a arquitetura de 64 processadores, apenas a prioridade *Caminho Crítico Dinâmico*, independente de usar ou não critérios de desempate, se destaca com 57,14% e qualidade de 1.068 nos resultados apresentados. Os valores estão mostrados na Figura 5.43. Na média final, conforme ilustra a Figura 5.44 o trio de prioridades que atinge melhor *makespan* na maioria dos casos é a prioridade *ALAP* com critérios de desempate *Nível Dinâmico e Caminho Crítico Dinâmico*. Com essa heurística a qualidade de *Makespan* gerados

é a melhor dentre todos os experimentos, 1,053. Os valores estão apresentados na Figura 5.44 e ilustrados no gráfico da Figura 5.45.

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	71,42%	1,025	85,71%	1,015	57,14%	1,031	57,14%	1,031
Conível	28,57%	1,098	28,57%	1,135	14,28%	1,145	14,28%	1,145
Caminho Crítico	28,57%	1,094	42,85%	1,084	28,75%	1,089	28,57%	1,094
ALAP	71,42%	1,025	85,71%	1,015	57,14%	1,031	57,14%	1,031
Nível Dinâmico	71,42%	1,025	85,71%	1,015	57,14%	1,031	57,14%	1,031
Conível Dinâmico	28,57%	1,099	14,28%	1,132	14,28%	1,132	14,28%	1,146
Caminho Crítico Dinâmico	42,85%	1,037	42,85%	1,037	42,85%	1,037	42,85%	1,037
ALAP Dinâmico	71,42%	1,025	85,71%	1,015	57,14%	1,031	57,14%	1,031

Figura 5.40: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos Irregulares

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	14,28%	1,117	14,28%	1,089	14,28%	1,117	42,85%	1,069
Conível	14,28%	1,11	14,28%	1,089	14,28%	1,11	42,85%	1,069
Caminho Crítico	28,57%	1,111	28,57%	1,082	28,57%	1,111	42,85%	1,069
ALAP	14,28%	1,11	28,57%	1,082	14,28%	1,11	42,85%	1,069
Nível Dinâmico	28,57%	1,126	14,28%	1,117	42,85%	1,12	14,28%	1,117
Conível Dinâmico	28,57%	1,106	14,28%	1,11	14,28%	1,131	14,28%	1,11
Caminho Crítico Dinâmico	42,85%	1,099	28,57%	1,111	28,57%	1,104	28,57%	1,111
ALAP Dinâmico	28,57%	1,106	14,28%	1,11	28,57%	1,104	14,28%	1,11

Figura 5.41: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos Irregulares

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	57,14%	1,049	71,42%	1,041	57,14%	1,049	57,14%	1,049
Conível	14,28%	1,104	57,14%	1,057	42,85%	1,066	42,85%	1,066
Caminho Crítico	42,85%	1,076	57,14%	1,068	42,85%	1,076	42,85%	1,076
ALAP	57,14%	1,049	71,42%	1,041	57,14%	1,049	57,14%	1,049
Nível Dinâmico	57,14%	1,049	71,42%	1,041	57,14%	1,049	57,14%	1,049
Conível Dinâmico	14,28%	1,112	14,28%	1,106	14,28%	1,106	14,28%	1,106
Caminho Crítico Dinâmico	57,14%	1,068	57,14%	1,068	57,14%	1,068	57,14%	1,068
ALAP Dinâmico	57,14%	1,049	71,42%	1,041	57,14%	1,049	57,14%	1,049

Figura 5.42: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos Irregulares

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	42,85%	1,073	42,85%	1,073	42,85%	1,073	42,85%	1,073
Conível	14,28%	1,104	28,57%	1,079	28,57%	1,079	28,57%	1,079
Caminho Crítico	42,85%	1,076	42,85%	1,076	42,85%	1,076	42,85%	1,076
ALAP	42,85%	1,073	42,85%	1,073	42,85%	1,073	42,85%	1,073
Nível Dinâmico	42,85%	1,073	42,85%	1,073	42,85%	1,073	42,85%	1,073
Conível Dinâmico	14,28%	1,112	28,57%	1,087	28,57%	1,087	28,57%	1,087
Caminho Crítico Dinâmico	57,14%	1,068	57,14%	1,068	57,14%	1,068	57,14%	1,068
ALAP Dinâmico	42,85%	1,073	42,85%	1,073	42,85%	1,073	42,85%	1,073

Figura 5.43: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos Irregulares

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	46,42%	*1,066*	53,57%	*1,055*	42,85%	*1,068*	50,00%	*1,056*
Conível	17,85%	*1,104*	32,14%	*1,090*	25,00%	*1,100*	32,14%	*1,090*
Caminho Crítico	35,71%	*1,089*	42,85%	*1,078*	35,76%	*1,088*	39,28%	*1,079*
ALAP	46,42%	*1,064*	57,14%	*1,053*	42,85%	*1,066*	50,00%	*1,056*
Nível Dinâmico	50,00%	*1,068*	53,57%	*1,062*	50,00%	*1,068*	42,85%	*1,068*
Conível Dinâmico	21,43%	*1,107*	17,85%	*1,109*	17,85%	*1,114*	17,85%	*1,112*
Caminho Crítico Dinâmico	50,00%	*1,068*	46,43%	*1,071*	46,43%	*1,069*	46,43%	*1,071*
ALAP Dinâmico	50,00%	*1,063*	53,57%	*1,060*	46,43%	*1,064*	42,85%	*1,066*

Figura 5.44: Médias considerando as quatro arquiteturas - Grafos Irregulares

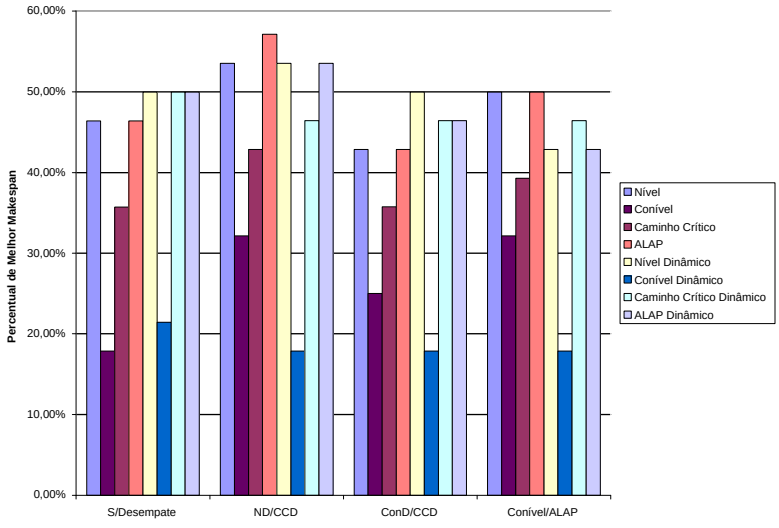

Figura 5.45: Médias por critérios de desempate - Grafos Irregulares

Segunda Etapa de Testes

A Figura 5.46 ilustra o escalonamento do grafo Irregular de 7 tarefas, anteriormente, apresentado. A mesma distribuição de tarefas, pelos mesmos processadores foi observadas nas 3 versões de *List Scheduling* que utilizaram: prioridade *ALAP* com critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico*; prioridade *Nível Dinâmico* com critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico*; e prioridade *Nível* com critério de desempate *Conível/ALAP*. Todas as três versões atingiram *makespan* de 20 unidades, e os processadores utilizados foram os mais rápidos P_2 e P_7, e a seguir P_3. É possivel observar a existência de intervalos de tempos entre as tarefas onde o processador permanece ocioso em virtude da comunicação entre as tarefas. Por exemplo, a tarefa v_5 somente pode começar em P_7 no tempo 12, uma vez que necessita esperar os dados vindos de v_3, escalonada em P_3. Da mesma forma, a t arefa v_6 somente pode iniciar a executar no tempo 18 depois de receber os dados, de 4 unidades, transmitidos por v_1, escalonada em P_2.

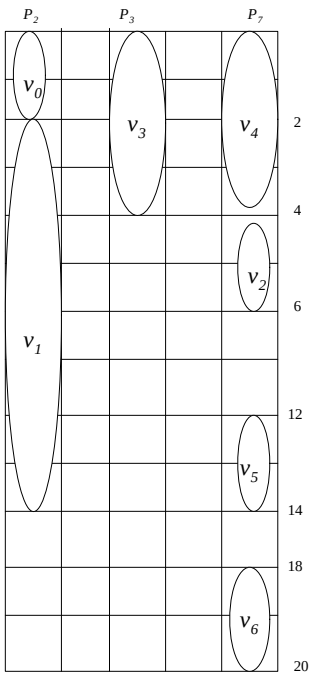

Figura 5.46: Escalonamento do Grafo *Irr7a* na arquitetura de 12 processadores com *makespan*= 20

Na Figura 5.47 está ilustrado o escalonamento do mesmo grafo irregular de 7 tarefas, na arquitetura de 12 processadores. A versão utilizada foi a *List Scheduling*

com prioridade *Conível Dinâmico* e critério de desempate *Nível Dinâmico/Caminho Crítico Dinâmico*. Em relação ao exemplo anterior, houve uma economia de um processador P_3. Entretanto, devido aos custos de comunicação entre as tarefas, é possível observar um intervalo de 8 unidades, no processador P_7. Isso ocorre, também, devido a topologia do grafo e do fato de que existindo processador disponível que permita minimizar o tempo de término das tarefas, as heurísticas de *List Scheduling* paralelizam as tarefas o máximo que podem, enfatizando assim a sua característica de *greedy*. A utilização de poucos processadores, neste caso, pode ser explicada pela tendência do *List Scheduling* em tentar agrupar as tarefas em poucos processadores a fim de minimizar os custos de comunicação, observados em grafos caracterizados como de granulosidade fina.

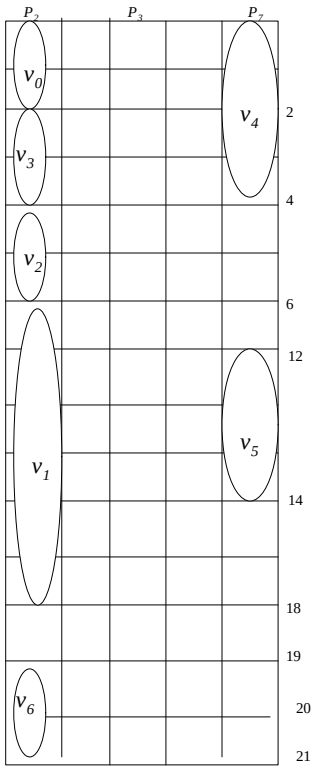

Figura 5.47: Escalonamento do Grafo *Irr7a* na arquitetura de 12 processadores com *makespan*=21

5.8 Grafos KPSG (*Kwok Peer Set Graphs*)

Esta classe de grafos é composta por 11 pequenos grafos de topologia irregular, com tarefas variando de 7 a 18. Esses grafos possuem, em média, custos de comunicação que excedem os custos de computação, por isso são classificados como grafos de granulosidade ou granularidade fina. Devido a pequena quantidade de tarefas que possuem, é possível acompanhar melhor o comportamento da heurística quando submetida a grafos de granulosidade fina. Esta observação é de fato muito importante na questão do Problema de Escalonamento de Tarefas, dito NP-Completo. Por essa razão, heurísticas são aceitas para atingir valores próximos ao ótimo em tempos menores que o exponencial. Entretanto, uma heurística do grupo de *List Scheduling*, a ETF *Earliest Task First* ultrapassou essa limitação. Na classe de grafos diamantes com granulosidade grossa conseguiu atingir, em tempos polinomiais, escalonamentos ótimos. Dessa forma, é de considerada relevância trilhar o comportamento das heurísticas de *List Scheduling* quando submetidas a grafos de granulosidade fina. O objetivo é verificar sob que condições atingiriam resultados ótimos. Dessa forma, o estudo desta classe de grafo é de grande importância para esse objetivo.

Primeira Etapa de Testes

Os experimentos realizados, com esta classe de grafos, na arquitetura de 8 processadores estão apresentados na Figura 5.48. As prioridades que se destacam são *Nível* e *ALAP*, calculados de forma estática e dinâmica, utilizadas sem nenhum desempate e quando utilizadas com a dupla de desempate *Nível Dinâmico* e *Caminho Crítico Dinâmico*. O *makespan* gerado foi o mesmo para todas as oito heurísticas, com qualidade de 1,006. Observando a topologia dos grafos, pode ser verificado que apresentam arcos com pesos, em média, superiories, aos pesos das tarefas. Assim, ocorreu uma tendência a concentrar o escalonamento de cada grafo, em média, em poucos processadores, 2 ou 3, sendo estes os mais rápidos. Com isso a mesma distribuição de tarefas, de cada grafo, foi realizada pelas heurísticas desses trios de prioridades. Na arquitetura de 12 processadores, cujos resultados estão apresentados na Figura 5.49, se observou que 12 heurísticas atingiram o mesmo percentual de casos com o melhor *makespan*, 63,63%. Porém, nem todas as 12 heurísticas atin-

giram o mesmo valor no *makespan*, as baseadas na prioridade *Conível* e *Conível* apresentaram menor degradação em relação ao melhor valor que as outras oito heurísticas. Na arquitetura de 32 processadores, também, se observaram 16 empates, como mostra a Figura 5.50. Todas as 16 heurísticas geraram o mesmo *makespan*, utilizando o mesmo número de processadores. No máximo 7 processadores (os mais rápidos) foram utilizados, o que ocorreu ao escalonar o grafo KPSG3. A qualidade do *makespan* foi de 1,035 para cada heurística. Na arquitetura de 64 processadores, 4 heurísticas, compostas pela prioridade *Caminho Crítico Dinâmico* utilizada sem desempate ou com os três critério de desempates presentes, atingiram o melhor *makespan* em 63,63% dos casos com qualidade de 1,034. Conforme apresenta a Figura 5.51. Na média, pelas quatro arquiteturas, destacou-se a prioridade *Nível* sem adotar critério de desempate e a mesma prioridade quando adotou o critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* que atingiram cada uma 59,09% dos casos de melhor *mamespan* com qualidade de 1,049. A heurística de pior resultado foi a baseada na prioridade *conível dinâmico* sem critério de desempate. Os valores estão apresentados na Figura 5.52 e ilustrados no gráfico da Figura 5.53.

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	72,72%	1,006	72,72%	1,006	63,63%	1,01	63,63%	1,01
Conível	36,36%	1,096	36,36%	1,107	36,36%	1,11	36,36%	1,107
Caminho Crítico	45,45%	1,072	54,54%	1,047	54,54%	1,047	54,54%	1,047
ALAP	72,72%	1,006	72,72%	1,006	63,63%	1,01	63,63%	1,01
Nível Dinâmico	72,72%	1,006	72,72%	1,006	63,63%	1,01	63,63%	1,01
Conível Dinâmico	27,27%	1,105	36,36%	1,08	36,36%	1,08	36,36%	1,099
Caminho Crítico Dinâmico	63,63%	1,056	63,63%	1,056	63,63%	1,056	63,63%	1,056
ALAP Dinâmico	72,72%	1,006	72,72%	1,006	63,63%	1,01	63,63%	1,01

Figura 5.48: Percentuais de Melhor *Makespan* e Qualidade P = 8 - Grafos KPSG

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	63,63%	1,047	54,54%	1,04	63,63%	1,047	54,54%	1,067
Conível	63,63%	1,042	54,54%	1,04	63,63%	1,042	45,45%	1,08
Caminho Crítico	63,63%	1,047	54,54%	1,04	63,63%	1,047	54,54%	1,067
ALAP	54,54%	1,055	54,54%	1,04	54,54%	1,055	54,54%	1,067
Nível Dinâmico	36,36%	1,102	63,63%	1,047	36,36%	1,122	63,63%	1,047
Conível Dinâmico	27,27%	1,089	63,63%	1,042	27,27%	1,113	63,63%	1,042
Caminho Crítico Dinâmico	27,27%	1,089	63,63%	1,047	36,36%	1,102	63,63%	1,047
ALAP Dinâmico	27,27%	1,093	54,54%	1,055	36,36%	1,102	54,54%	1,055

Figura 5.49: Percentuais de Melhor *Makespan* e Qualidade P = 12 - Grafos KPSG

Segunda Etapa de Testes

Os grafos escolhidos para ilustrar alguns exemplos dos escalonamentos observados foram os grafos *KPSG7* e *KPSG8*. No apêndice *A* estão ilustrados todos

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	54,54%	1,035	54,54%	1,035	54,54%	1,079	54,54%	1,035
Conível	36,36%	1,089	36,36%	1,079	36,36%	1,06	36,36%	1,079
Caminho Crítico	36,36%	1,057	36,36%	1,06	36,36%	1,035	36,36%	1,06
ALAP	54,54%	1,035	54,54%	1,035	54,54%	1,035	54,54%	1,035
Nível Dinâmico	54,54%	1,035	54,54%	1,035	54,54%	1,035	54,54%	1,035
Conível Dinâmico	18,18%	1,117	27,27%	1,082	27,27%	1,082	27,27%	1,087
Caminho Crítico Dinâmico	45,45%	1,087	36,36%	1,09	36,36%	1,09	36,36%	1,09
ALAP Dinâmico	54,54%	1,035	54,54%	1,035	54,54%	1,035	54,54%	1,035

Figura 5.50: Percentuais de Melhor *Makespan* e Qualidade P = 32 - Grafos KPSG

	S/Desempate	Qualidade	ND/CCD	Qualidade	ConD/CCD	Qualidade	Conível/ALAP	Qualidade
Nível	45,45%	1,053	45,45%	1,053	45,45%	1,053	45,45%	1,053
Conível	36,36%	1,099	36,36%	1,089	36,36%	1,089	36,36%	1,089
Caminho Crítico	36,36%	1,054	36,36%	1,054	36,36%	1,054	36,36%	1,054
ALAP	45,45%	1,053	45,45%	1,053	45,45%	1,053	45,45%	1,053
Nível Dinâmico	45,45%	1,053	45,45%	1,053	45,45%	1,053	45,45%	1,053
Conível Dinâmico	18,18%	1,136	27,27%	1,116	27,27%	1,116	18,18%	1,125
Caminho Crítico Dinâmico	63,63%	1,034	63,63%	1,034	63,63%	1,034	63,63%	1,034
ALAP Dinâmico	45,45%	1,053	45,45%	1,053	45,45%	1,053	45,45%	1,053

Figura 5.51: Percentuais de Melhor *Makespan* e Qualidade P = 64 - Grafos KPSG

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	59,09%	*1,035*	56,81%	*1,034*	56,81%	*1,047*	54,54%	*1,041*
Conível	43,18%	*1,082*	40,91%	*1,079*	43,18%	*1,075*	38,63%	*1,089*
Caminho Crítico	45,45%	*1,058*	45,45%	*1,050*	47,72%	*1,046*	45,45%	*1,057*
ALAP	56,81%	*1,037*	56,81%	*1,034*	54,54%	*1,038*	54,54%	*1,041*
Nível Dinâmico	52,27%	*1,049*	59,09%	*1,035*	50,00%	*1,055*	56,81%	*1,036*
Conível Dinâmico	22,73%	*1,112*	38,63%	*1,080*	29,54%	*1,098*	36,36%	*1,088*
Caminho Crítico Dinâmico	50,00%	*1,067*	56,81%	*1,057*	50,00%	*1,071*	56,81%	*1,057*
ALAP Dinâmico	50,00%	*1,047*	56,81%	*1,037*	50,00%	*1,050*	54,54%	*1,038*

Figura 5.52: Médias considerando as quatro arquiteturas - Grafos KPSG

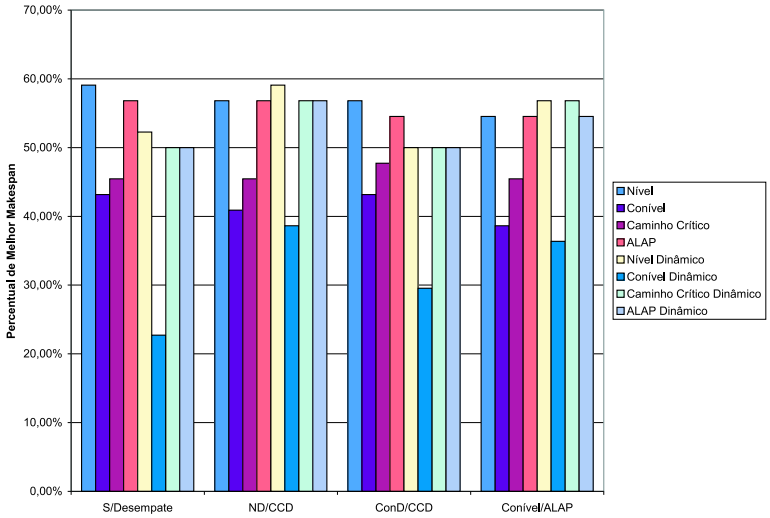

Figura 5.53: Médias por critérios de desempate - Grafos KPSG

os 11 grafos que compõem esta classe. Nos experimentos realizados com o grafo *KPSG7* de 11 tarefas, nenhuma das 32 versões de *List Scheduling* conseguiu se des-

tacar, todas atingiram o mesmo *makespan* de 11 unidades utilizando o processador P_2. Esse processador é o primeiro dos mais rápidos e a distribuição seqüencial das tarefas foi identica para todas as versões: v_0, v_1, v_3, v_2, v_6, v_4, v_5, v_7, v_8, v_9 e v_{11}. Em relação ao grafo *KPSG8* é composto por 9 tarefas e de granulosidade fina ($G(GAD) = 0.28 < 1$). Nesse grafo, o somatório dos custos de comunicação é, em média, 28% acima do somatório dos custos de computação. Fato interessante que pode ser observado nos mapas de escalonamento gerados, foi que, também, existiu por parte das versões a tendência de utilizar poucos processadores. Isso ocorreu para evitar as elevadas comunicações existentes entre as tarefas. Por exemplo, a versão com prioridade *Nível Dinâmico* sem critério de desempate; a versão com *Nível Dinâmico* com critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico*; a versão com *Caminho Crítico* sem critério de desempate; e uma das versões (que atingiu o melhor sucesso considerando as quatro arquiteturas) com prioridade *Nível* sem critério de desempate, apresentaram o mesmo escalonamnento com igual distribuição de tarefas em um só processador P_2 e *makespan* de 180 unidades. A versão que atingiu o melhor *makespan* de 159 unidades, para o grafo *KPSG8* foi a que utilizou *Conível Dinâmico* e sem critério de desempate. Essa versão distribuiu as tarefas nos dois processadores mais rápidos, disponíveis na arquitetura, P_2 e P_7.

5.9 Avalição Final

Na classificação final, em média, considerando todos os 65 grafos e as quatro arquiteturas disponibilizadas, apenas uma versão de *List Scheduling* apresentou o maior percentual de casos com o melhor *makespan*. Tal versão utiliza o trio de prioridades *Nível Dinâmico* em conjunto com o critério de desempate *Nível Dinâmico e Caminho Crítico Dinâmico*. Isto significou que não houve a necessidade de utilizar dois critérios de desempate; com apenas um critério de desempate incorporado a prioridade principal de *Nível Dinâmico* a versão de *List Scheduling* conseguiu, nas instâncias de testes efetuados, atingir melhores *makespans* que as concorrentes. Os valores estão apresentados na Figura 5.54 e ilustrados na Figura 5.55.

A versão de menor contribuição para a minimização do *makespan* foi a baseada na prioridade *Conível Dinâmico* sem utilizar nenhum critério de desempate. Em parte já era esperado, conforme pode ser observado no decorrer dos experimentos. Em média, as versões que utilizaram critérios de desempate conseguiam, na maior parte dos casos, decidir por tarefas mais relevates para a diminuição do *makespan*. Quanto a prioridade principal de *Conível*, a razão de não conseguir êxito é porque as heurísticas de *List Scheduling* baseadas nessa prioridade priorizam tarefas com o tempo de início mais breve possível. Com isso, existe por parte desse critério uma abordagem do grafo, em um único sentido, de cima para baixo. Partindo exclusivamente da origem, não se pode capturar a real importância de uma tarefa dentro do grafo e, conseqüentemente, não se pode determinar o seu grau de urgência no processo de escalonamento. Mesmo que recalculado a cada iteração do escalonamento, a prioridade *Conível Dinâmico* não atinge bons resultados, uma vez que não consegue identificar e priorizar tarefas realmente importantes, tais como, aquelas cujos caminhos até o fim do grafo são os mais longos. Esse fato somente pode ser observado, se o grafo for percorrido de baixo para cima. Segundo os resultados finais isso realmente é preciso, uma vez que a heurística de maior sucesso (por atingir o maior percentual de casos com o melhor *makespan*) foi a que utilizou essa abordagem de percorrer o grafo de baixo para cima, ou seja, utilizou o atributo *Nível*. Dessa forma, são priorizadas as tarefas seguidas das maiores cadeias de tarefas. As heurísticas de *List Scheduling* baseadas na prioridade *Nível* são aquelas que priorizam tarefas cujas distâncias para as tarefas de saída (finalizadoras do grafo) são maiores. Isso indica que as tarefas que pertencem aos caminhos mais longos (desde a tarefa em questão até uma tarefa de saída) devem, realmente, ser antecipadas pois o efeito final dessa ação repercutirá na diminuição do *makespan*, principal meta que se deseja alcançar. Efeito esse que se tornou mais significativo a medida que se recalculava a cada passo do escalonamento o *Nível* da tarefa, dito por isso *Nível Dinâmico*. Assim, foi possível capturar melhor a relativa importância da tarefa a cada iteração. A tarefa que melhor contribuía para a minimização do *makespan* era a preferida para o escalonamento. Aliás, como pode ter sido percebido ao longo dos experimentos, uma heurística de *List Scheduling*, é dita de construção por construir

um único escalonamento do grafo, que se espera ser o melhor possível. Esse escalonamento é construído escolhendo uma tarefa a cada iteração. Entretanto, para a heurística construir o melhor escalonamento final, a cada iteração, precisa escalonar a melhor tarefa, ou seja, aquela que mais contribui para minimizar o *makespan* da aplicação.

	S/Desempate	*Qualidade*	ND/CCD	*Qualidade*	ConD/CCD	*Qualidade*	Conível/ALAP	*Qualidade*
Nível	55,98%	*1,052*	64,14%	*1,048*	56,21%	*1,065*	49,42%	*1,061*
Conível	41,42%	*1,084*	48,09%	*1,087*	44,81%	*1,082*	38,22%	*1,090*
Caminho Crítico	53,27%	*1,048*	53,95%	*1,068*	43,81%	*1,065*	41,73%	*1,070*
ALAP	55,62%	*1,052*	63,65%	*1,050*	55,85%	*1,053*	49,80%	*1,061*
Nível Dinâmico	51,87%	*1,060*	66,19%	*1,047*	53,51%	*1,061*	53,81%	*1,055*
Conível Dinâmico	33,16%	*1,096*	39,57%	*1,082*	35,09%	*1,090*	39,39%	*1,086*
Caminho Crítico Dinâmico	49,41%	*1,073*	51,88%	*1,067*	49,03%	*1,072*	52,08%	*1,066*
ALAP Dinâmico	51,70%	*1,058*	65,83%	*1,046*	52,89%	*1,059*	53,45%	*1,055*

Figura 5.54: Média Final-Todos Grafos

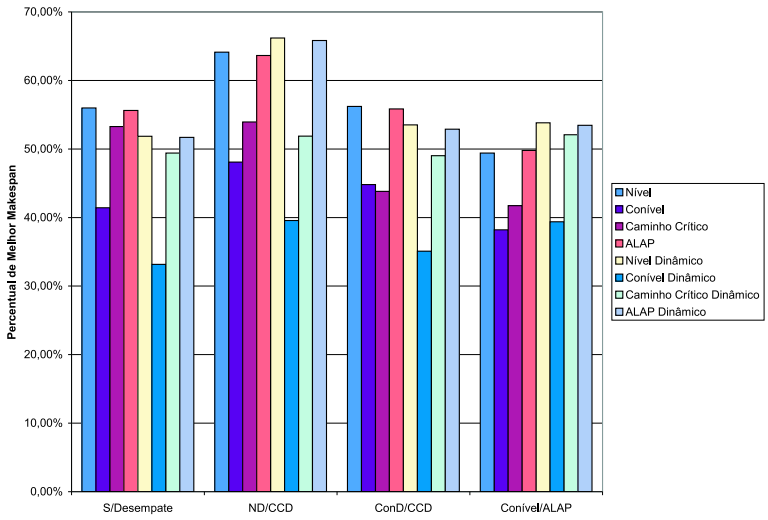

Figura 5.55: Média Final - Todos Grafos

A Figura 5.56 ilustra o percentual, em média, considerando as oito prioridades utilizadas em conjunto, que cada critério de desempate (A), (B), (C) e (D) atingiu. Os valores estão agrupados de acordo com a arquitetura utilizada. Pode ser observado que critério de desempate (B) *Nível Dinâmico/Caminho Crítico* se destaca ao atingir o maior percentual de casos onde foi alcançado o melhor *makespan*. O destaque ocorre nas quatro arquiteturas utilizadas. Entre a arquitetura de 8 processadores e de 12 processadores, o percentual de casos com o melhor *makespan*, para esse critério, sofre uma queda. Contudo, é possível observar que acima de 12 processadores o percentual de casos de melhor *makespan*, atingido por esse critério,

sofre melhorias gradativas. Confirmando assim, ser esse critério o de maior sucesso para a minimização do *makespan*, considerando as quatro arquiteturas propostas e sob o modelo de comunicação de latência.

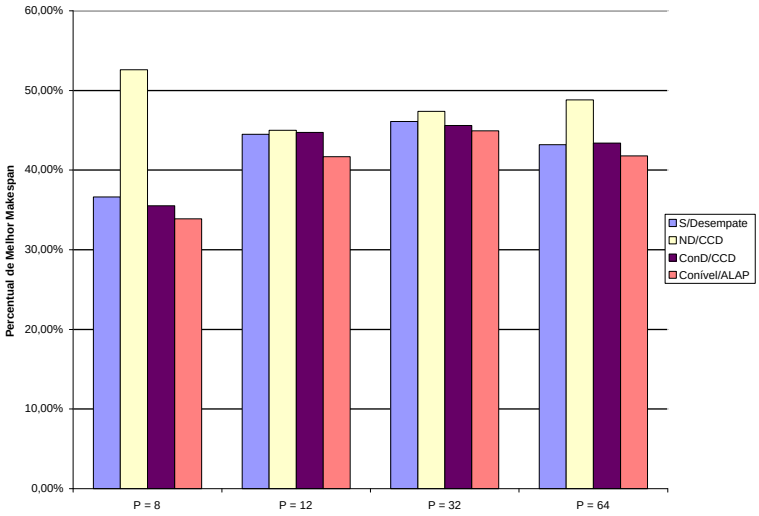

Figura 5.56: Critérios de Desempate Agrupados por Arquitetura

5.10 Resumo

Este capítulo apresentou os experimentos realizados, em ambientes heterogêneos sob o modelo de Latência. Foram avaliadas diferentes versões de *List Scheduling* compostas por uma prioridade principal e duas etapas sucessivas de critérios de desempate, ou nenhuma etapa de desempate. As oito prioridades disponibilizadas eram: *nível*, *conível*, *caminho crítico* e *ALAP*, calculadas de forma estática e dinâmica. O resultado da investigação indicou a versão que utilizou a prioridade *Nível Dinâmico* e a dupla de critérios de desempate *Nível Dinâmico e Caminho Crítico Dinâmico* como a de maior sucesso nos experimentos realizados. No próximo capítulo são apresentados os experimentos realizados em ambiente heterogêneo sob o modelo *LogP* que objetivam avaliar estratégias para melhor tratamento das sobrecargas de comunicação.

Capítulo 6

Experimentos em Ambiente Heterogêneo sob o Modelo *LogP*

Neste capítulo são apresentados os experimentos realizados com as quatro versões da heurística proposta *LSRGC*. São três os objetivos deste capítulo, o primeiro é avaliar qual das versões apresenta a melhor estratégia para minimizar os efeitos adversos que as sobrecargas de comunicação causam ao *makespan* das aplicações; o segundo é avaliar se a coleta dos espaços inúteis dentro das reservas R devem ser feitas ao final do escalonamento, como propõem as versões LSRGCV0 e LSRGCV1, ou a cada passo do escalonamento, como propõem as versões LSRGCV2 e LSRGCV3; e o terceiro objetivo é avaliar se as ordenações das sobrecagas de envio devem ser baseadas nas características do grafo, como propõem as versões LSRGCV0 e LSRGCV2, ou baseadas nas características do escalonamento gerado, versões LSRGCV1 e LSRGCV3. Quando as ordenações são baseadas nas características do escalonamento uma restrição é imposta: As mensagens são atrasadas em seus tempos de início, uma vez que somente podem ser transmitidas ao final dos espaços de reserva R. Em princípio, tal restrição pode parecer negativa para o escalonamento; entretanto os resultados demonstraram que aliada a outras táticas, resultou nos benefícios mais significativos para minimizar o *makespan*. A métrica utilizada na comparação dos valores é o *makespan* do escalonamento. Quanto me-

nor for esse valor melhor é o desempenho da heurística. A prioridade adotada neste conjunto de experimentos é *nível dinâmico* e o critério para primeiro desempate é *conível dinâmico* e para segundo desempate é *caminho crítico dinâmico*.

6.1 Considerações Iniciais

Os experimentos foram divididos pelas 6 classe do grafo considerados: diamantes, intree, outtree, randômicos, irregulares e KPSG (*Peer Set Graphs*). Em cada classe os testes foram realizados considerando as quatro arquiteturas compostas por 8, 12, 32 e 64 processadores heterogêneos, apresentadas no capítulo 5, e adaptadas para o modelo *LogP*. Para cada arquitetura variou-se os valores das sobrecargas de envio e de recebimento compondo-se os pares ($O_s = 1, O_r = 1$); ($O_s = 1$, $O_r = 4$); e ($O_s = 4$, $O_r = 1$). Além disso, foram utilizados dois fatores multiplicativos, um para a computação das tarefas, m e outro para a comunicação das tarefas F, formando pares (m, F) o que permitiu simular ambientes de distintas granulosidades. Para compôr ambientes de granulosidade grossa foram utilizados os pares $(1, 1)$; $(4, 1)$ e $(10, 1)$. Da mesma forma, para compôr o ambiente de granulosidade fina, foram utilizados os pares $(1, 4)$ e $(1, 10)$. Assim, foram realizadas 15 baterias de testes para cada arquitetura em cada classe de grafos. No total foram executados 15600 experimentos. Os resultados são apresentados sob a forma de gráficos comparativos com os percentuais de casos onde cada uma das quatro versões atingiu o melhor *makespan*. Os resultados, também, são avaliados em relação a qualidade do *makespan*. Assim, gráficos com a qualidade gerada por cada uma das heurísticas, agrupados pela respectiva bateria de testes são apresentados. O que permite verificar a degradação média do *makespan* em relação ao melhor *makespan* atingido por uma das versões. Quanto mais próxima de 1,00 for a degradação apresentada pela heurística melhor é a sua qualidade.

6.2 Grafos Diamantes

Com esta classe de grafos foram executados 2640 testes. Os resultados dos experimentos realizados na arquitetura de 8 processadores com as várias combinações de valores de parâmetros (O_s, O_r, m e F) escolhidos são apresentados na Figura 6.1. Em ambientes de granulosidade grossa, os escalonamentos da versão LSRGCV2 atingiram o valor de melhor *makespan* produzido pelas quatro versões em 71% dos casos, a LSRGCV3 58%, a LSRGCV1 17% e a LSRGCV0 16%. Em ambiente de granulosidade fina a versão LSRGCV2 conseguiu 79%, a versão LSRGCV3 53%, a LSRGCV0 26% e a LSRGCV1 24%. Na média ponderada dos dois ambientes, a versão LSRGCV2 apresentou 74%, e para as outras versões os valores foram: 56% para a LSRGCV3, 20% para a LSRGCV0 e 19% para a LSRGCV1. Na Figura 6.2 é ilustrada a qualidade relativa aos escalonamentos gerados pelas heurísticas. A qualidade média dos resultados da LSRGCV2 é de 1,007, da LSRGCV3 é de 1,019, da LSRGCV0 1,072 e da LSRGCV1 1,111. Através desses valores é possível verificar que, entre as duas melhores colocadas (versão LSRGCV2 e LSRGCV3) os resultados da versão LSRGCV3 não foram demasiadamente piores que os produzidos pela LSRGCV2.

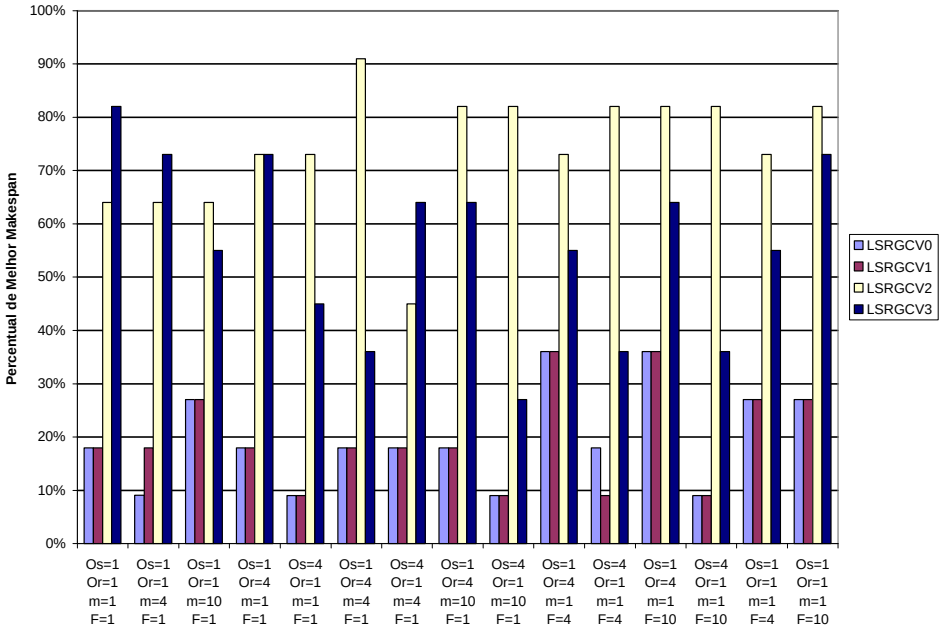

Figura 6.1: Comparação dos Percentuais com P=8 - Grafos Diamantes

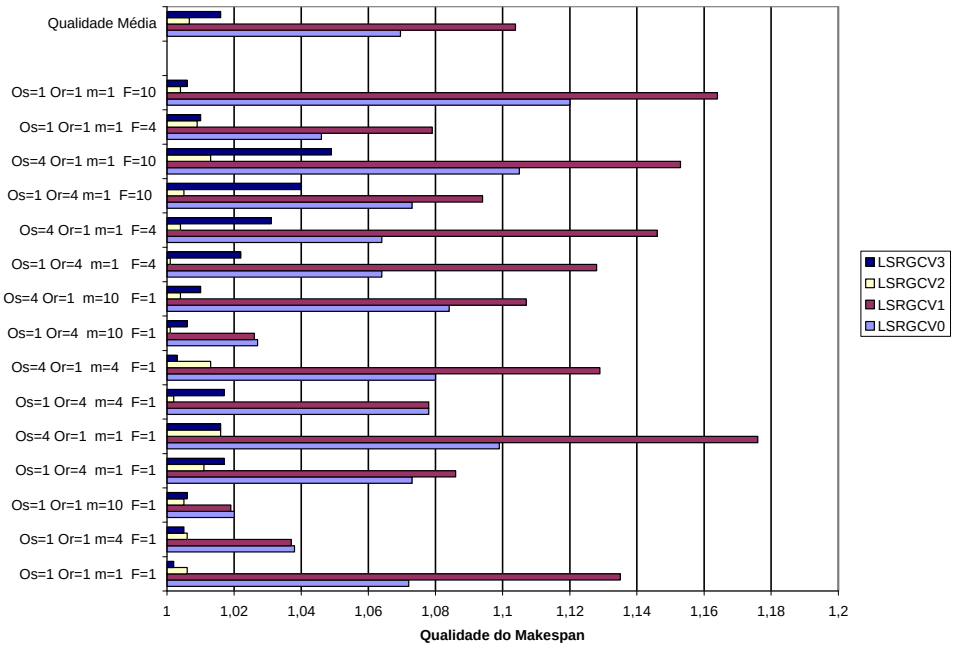

Figura 6.2: Comparação da Qualidade com P=8 - Grafos Diamantes

Os resultados obtidos na arquitetura de 12 processadores estão ilustrados na Figura 6.3, para ambientes de granulosidade grossa a versão LSRGCV3 consegue o melhor *makespan* em 59% dos casos, a LSRGCV2 em 56%, a LSRGCV0 com 27% e a LSRGCV1 com 25%. Em ambientes de granulosidade fina a LSRGCV2 apresentou 79%, a LSRGCV3 58%, a LSRGCV0 20% e a LSRGCV1 18%. Na média ponderada dos ambientes a LSRGCV2 apresentou 65%, a seguir a LSRGCV3 59%, a LSRGCV0 com 24% e a LSRGCV1 com 22%. A Figura 6.4 ilustra os valores de qualidade de *makespan* produzidos pelas heurísticas. Apesar da diferença de percentual, entre as duas versões melhores colocadas, pode ser observado que a qualidade média dos resultados produzidos pelas duas é igual 1,002. Quanto as outras duas versões a LSRGCV1 apresentou 1,014 e a LSRGCV0 1,016.

Na arquitetura de 32 processadores, conforme pode ser verificado na Figura 6.5, em ambientes de granulosidade grossa, a LSRGCV2 apresentou 77%, a LSRGCV3 66%, a LSRGCV1 29% e a LSRGCV0 32%. Em ambientes de granulosidade fina a versão LSRGCV2 apresentou 70%, a versão LSRGCV3 apresentou 63%, a LSRGCV1 23% e a LSRGCV0 18%. Indicando valores relativamente mais favoráveis em ambientes de granulosidade grossa para a versão, em melhor colocação, LSRGCV2. Na

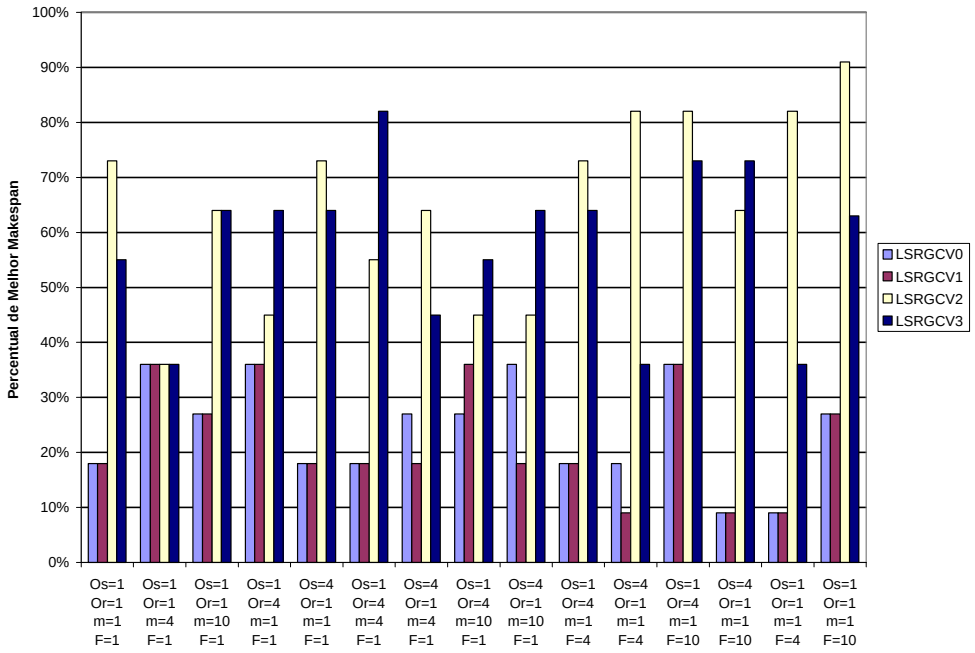

Figura 6.3: Comparação dos Percentuais com P=12 - Grafos Diamantes

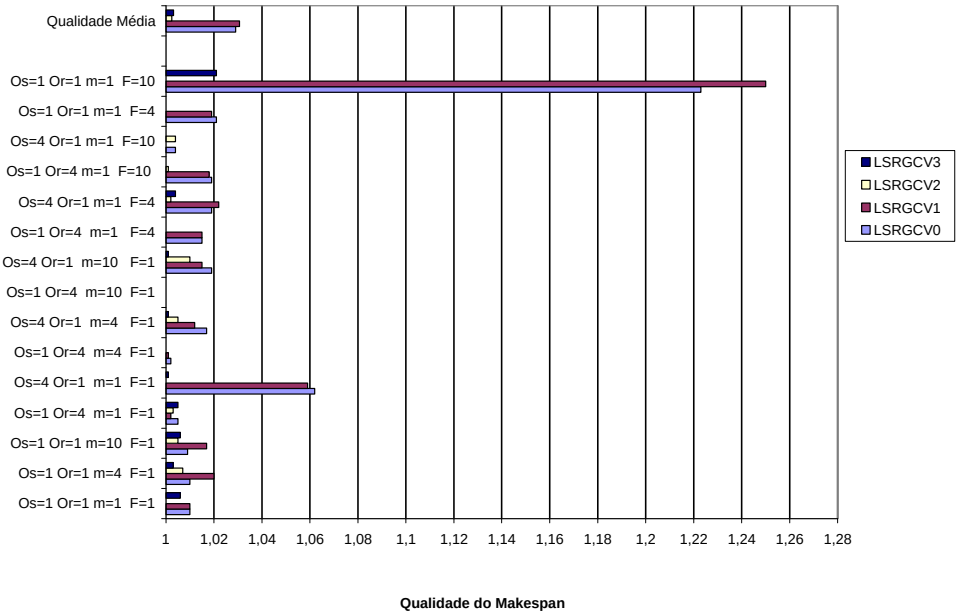

Figura 6.4: Comparação da Qualidade com P=12 - Grafos Diamantes

média ponderada dos ambientes, de granulosidade grossa e fina, observou-se 74% para a LSRGCV2, 65% para a LSRGCV3, 27% para a LSRGCV1 e 26% para a LSRGCV0. Na média dos resultados da qualidade, apresentados na 6.6 é possível

observar que a diferença de entre a versão LSRGCV2 e LSRGCV3 é mínima, 1,007 para a LSRGCV2 e 1,009 para a LSRGCV3. Com respeito a qualidade dos resultados produzidos pelas outras duas versões, os valores são: 1,079 para a LSRGCV01 e 1,104 para a LSRGCV1.

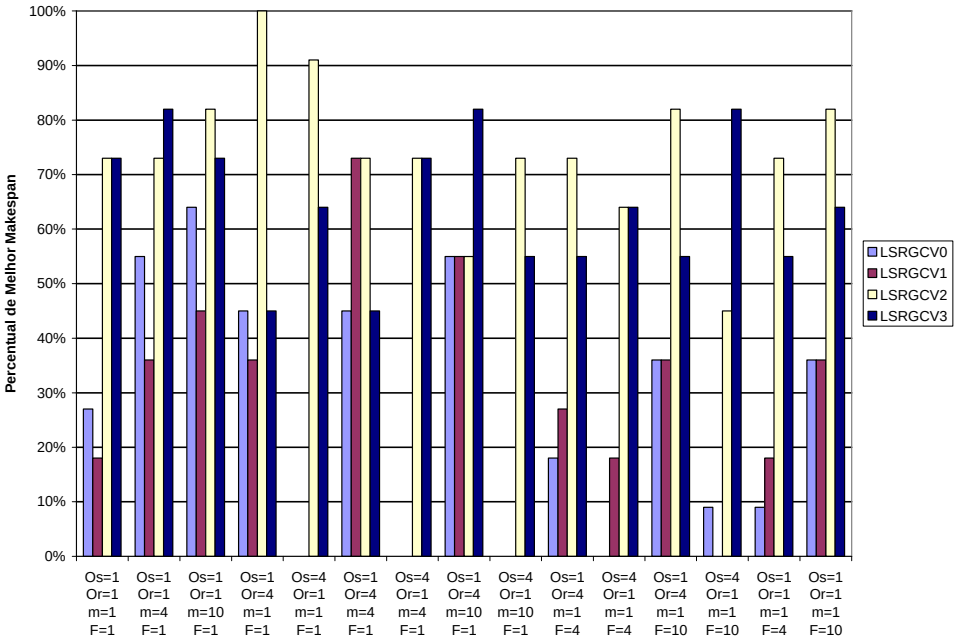

Figura 6.5: Comparação dos Percentuais com P=32 - Grafos Diamantes

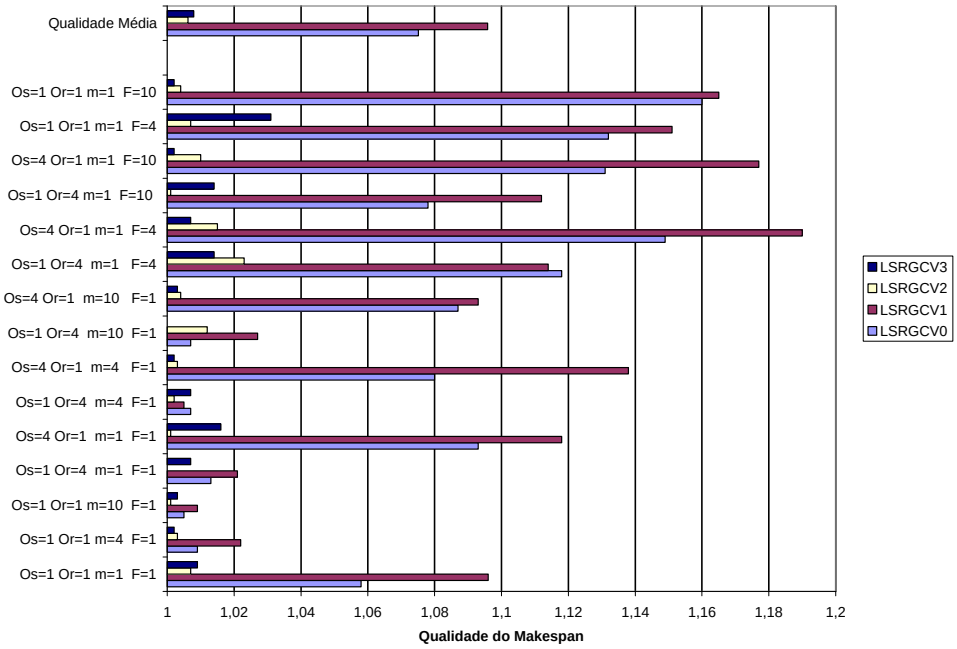

Figura 6.6: Comparação da Qualidade com P=32 - Grafos Diamantes

A Figura 6.7 ilustra os valores obtidos usando 64 processadores. Em ambientes de granulosidade grossa, a LSRGCV3 atingiu 75%, a LSRGCV2 66%, a LSRGCV0 65% e a LSRGCV1 49%. Em ambiente de granulosidade fina a LSRGCV2 atingiu 61%, a LSRGCV3 56%, a LSRGCV0 33% e a LSRGCV1 30%. Na média ponderada das duas granulosidades a LSRGCV3 apresentou 67%, a LSRGCV2 64%, a LSRGCV1 41% e a LSRGCV0 52%. Através da 6.8 podem ser verificados os valores da qualidade dos resultados produzidos. A degradação média em relação ao melhor valor de *makespan* produzido pela LSRGCV2 é de 1,023, da LSRGCV3 é ligeiramente superior com 1,026. A LSRGCV0 apresentou 1,070 e a LSRGCV1 apresentou 1,093.

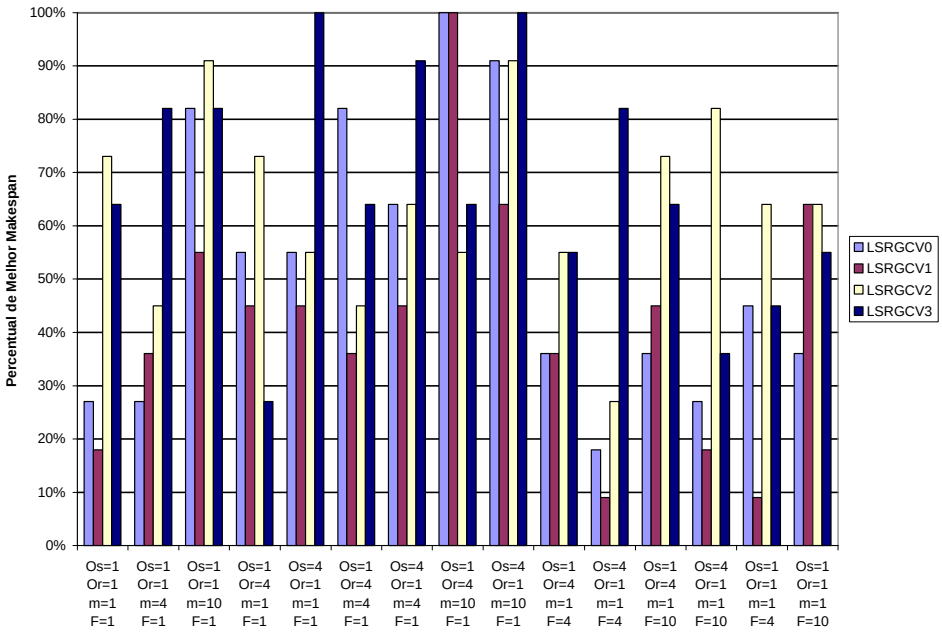

Figura 6.7: Comparação dos Percentuais com P=64 - Grafos Diamantes

Na média das quatro arquiteturas, a classificação final das versões indicou a LSRGCV2 com 69%, a LSRGCV3 com 62%, a LSRGCV0 com 30% e a LSRGCV1 com 27%. A degradação média dos valores é de 1,009 para a LSRGCV2, 1,011 para a LSRGCV3, 1,059 para a LSRGCV0 e 1,080 para a LSRGCV1. A política proposta pela LSRGCV2 foi melhor que a das concorrentes, uma vez que nos grafos desta topologia cada tarefa tem no máximo dois sucessores, com isso o tamanho das reservas R é relativamente pequeno. O envio das mensagens ao término de cada sobrecarga permitiu o escalonamento de tarefas sucessoras com predecessores distintos, em um

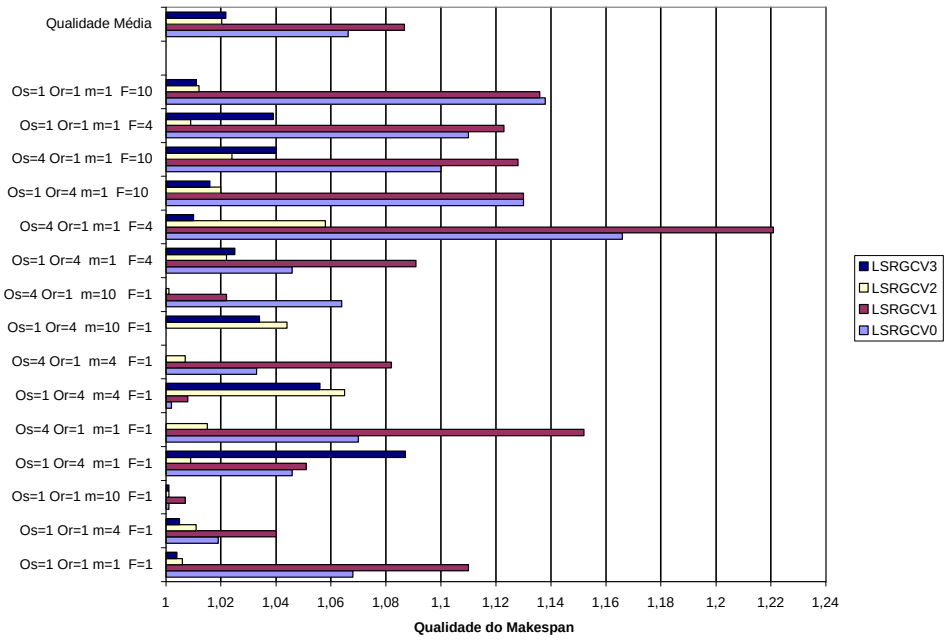

Figura 6.8: Comparação da Qualidade com P=64 - Grafos Diamantes

dos processadores que já alojavam algum desses predecessores, evitando a etapa de ordenação das sobrecargas restantes em R. Assim, com a coleta de lixo aplicada a cada passo do escalonamento, menor ficavam as reservar R. Além disso, tarefas com um único predecessor foram na maioria das vezes designadas ao mesmo processador de seu predecessor imediato, evitando as sobrecargas de comunicação.

Considerando, ainda, as médias das 15 baterias de testes, obtidas por cada versão, em cada uma das quatro arquiteturas disponibilizadas, foi realizada uma comparação do percentual de casos que cada uma das versões atingiu o melhor *makespan*. Conforme ilustra a Figura 6.9 pode ser visto que a versão LSRGCV2 se destaca com percentuais mais altos, nas arquiteturas de 8, 12 e 32 processadores, sendo portanto a que atinge a melhor média. Na arquitetura de 64 processadores, uma arquitetura que pode ser, praticamente, considerada homogênea devido aos 59 processadores rápidos, $h(p) = 1$, que apresenta; se destacou a versão LSRGCV3. Devido a esse resultado, foram realizados testes considerando as quatro arquiteturas compostas por processadores homogêneos, ou seja, todos tem $h(p) = 1$. Entretanto, apenas um par de parâmetros foi considerado ($O_s = O_r = m = F = 1$), uma vez que o objetivo era verificar se as versões se destacariam da mesma forma. Os resultados dos percentu-

ais atingidos por cada versão em ambiente homogêneo está apresentado na Figura 6.10. Nessa figura é possível verificar que nas arquiteturas de 8 e 12 processadores, também, lidera nos percentuais a LSRGCV2. Contudo, com a disponibilidade de mais processadores 32 e 64, a versão LSRGCV3 captura melhor a escalabilidade de processadores oferecidos ultrapassando a versão LSRGCV2. Atingindo, assim, a melhor média final.

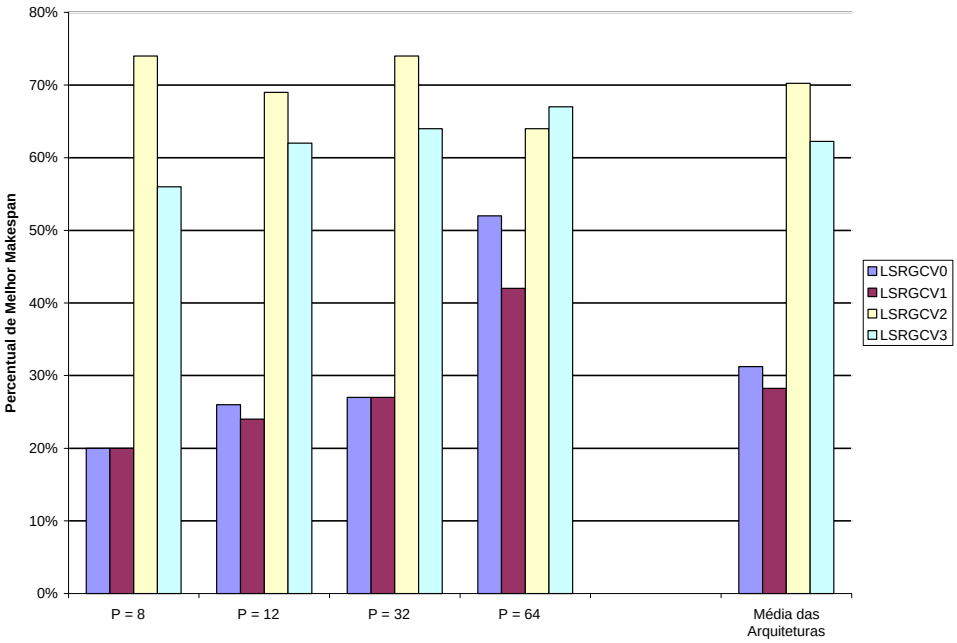

Figura 6.9: Comparação por Arquitetura Heterogênea - Grafos Diamantes

6.3 Grafos Intree

Com esta classe de grafos foram executados 1680 experimentos. Na arquitetura com 8 processadores, conforme apresenta a Figura 6.11, no ambiente de granulosidade grossa, na média dos valores obtidos nos testes efetuados, a LSRGCV3 e a LSRGCV2 empataram em 84%; e ambas as versões LSRGCV0 e LSRGCV1 empataram em 45%. Na média dos valores obtidos no ambiente de granulosidade fina, também, ocorreram empates, as versões LSRGCV3 e LSRGCV2 apresentaram 76% e as versões LSRGCV1 e LSRGCV0 apresentaram 45%. Na média ponderada, considerando os dois ambientes, as versões LSRGCV2 e LSRGCV3 apresentaram 81%

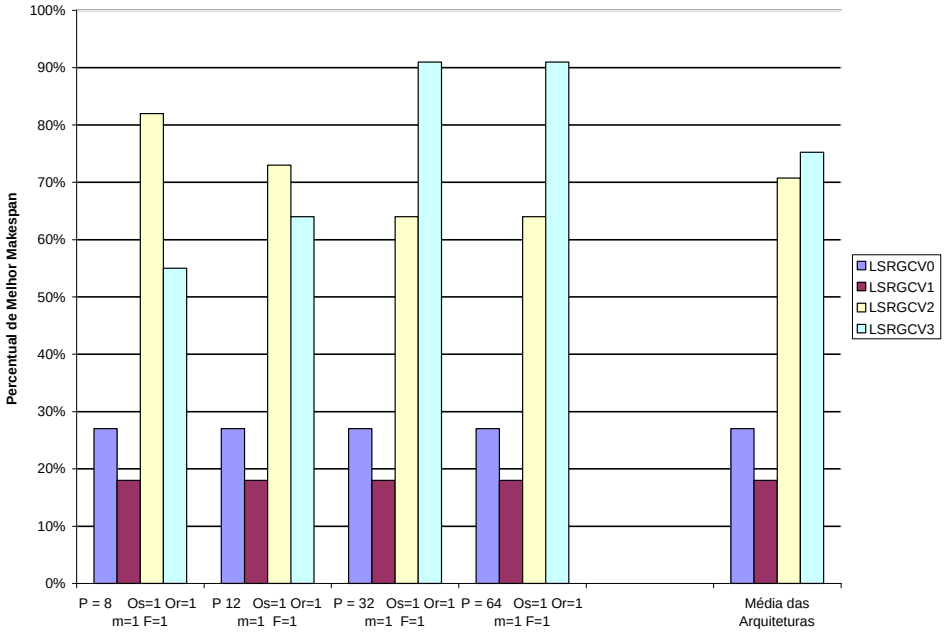

Figura 6.10: Comparação por Arquitetura Homogênea - Grafos Diamantes

e ambas as versões LSRGCV0 e LSRGCV1 apresentaram 45%. A Figura 6.12 ilustra a qualidade dos resultados produzidos por casa uma das versões por cada teste efetuado. A média da qualidade resultou em 1,026 para as versões LSRGCV2 e LSRGCV3 e 1,034 para as versões LSRGCV0 e LSRGCV1.

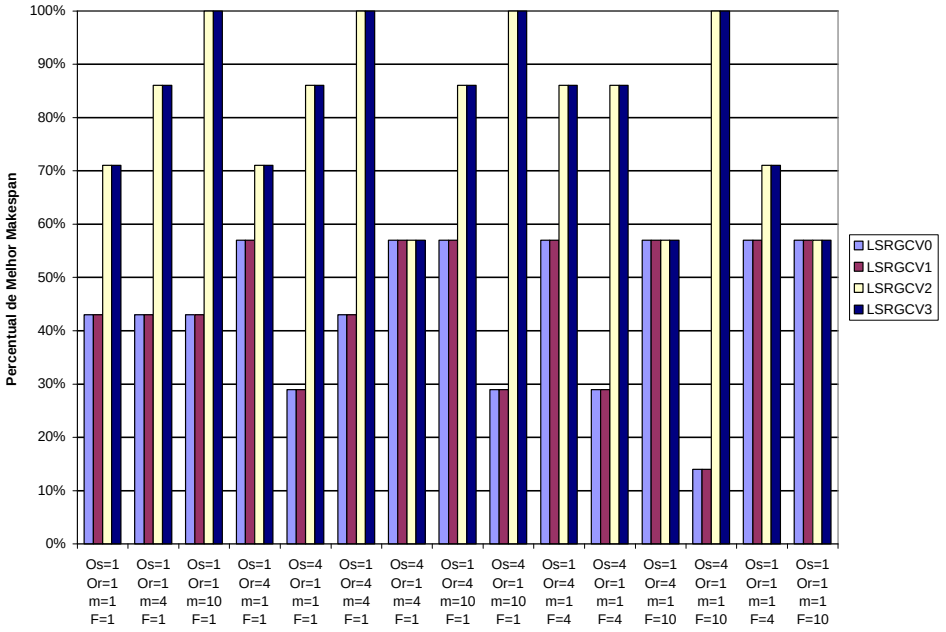

Figura 6.11: Comparação dos Percentuais com P=8 - Grafos Intree

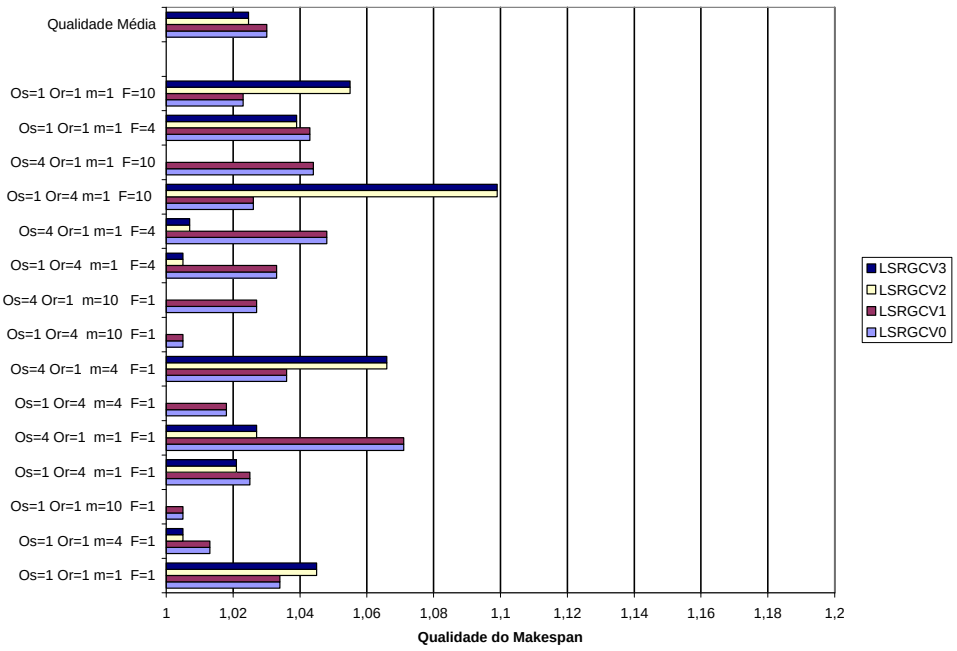

Figura 6.12: Comparação da Qualidade com P=8 - Grafos Intree

Na arquitetura de 12 processadores, conforme ilustra a Figura 6.13, em ambientes de granulosidade grossa, as versões LSRGCV2 e LSRGCV3 atingiram o valor de melhor *makespan* produzido pelas quatro versões em 83% dos escalonamentos, as versões LSRGCV0 e LSRGCV1 em 51%. Em ambientes de granulosidade fina, as versões LSRGCV2 e LSRGCV3 continuam liderando com 83% para cada uma, a seguir, também, com iguais valores estão as versões LSRGCV1 e LSRGCV0 com 34%. Na média ponderada pelos ambientes, a LSRGCV2 e LSRGCV3 atingiram 83% e as versões LSRGCV0 e LSRGCV1 44%. A Figura 6.14 ilustra as comparações da qualidade de *makespan* produzidos por cada uma das versões. Na média os valores foram 1,008 para a LSRGCV2 e LSRGCV3 e 1,051 para a LSRGCV0 e LSRGCV1.

Na arquitetura com 32 processadores, conforme apresenta a Figura 6.15, em ambientes de granulosidade grossa as versões LSRGCV2 e LSRGCV3 se destacam, ambas, com 95%. As outras versões LSRGCV0 e LSRGCV1 atingiram 54% cada uma. Em ambientes de granulosidade fina as versões LSRGCV2 e LSRGCV3 atingiram 81% e as outras duas versões 62%. Na média ponderada, considerando os dois ambientes, as versões LSRGCV2 e LSRGCV3 atingiram 89% e as versões LSRGCV0 e LSRGCV1 atingiram 57%, cada uma. Conforme ilustra a Figura 6.16 a qualidade

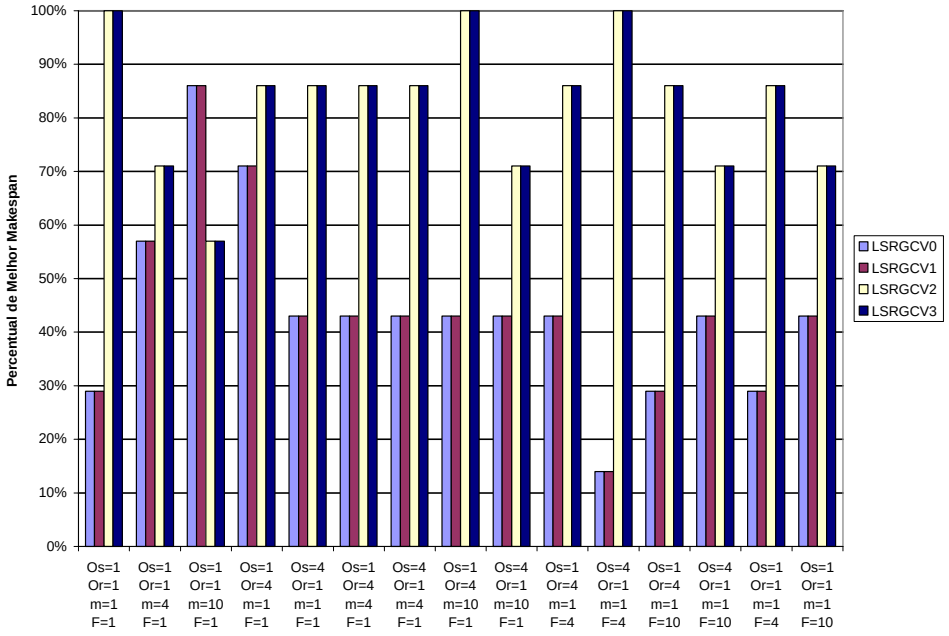

Figura 6.13: Comparação dos Percentuais com P=12 - Grafos Intree

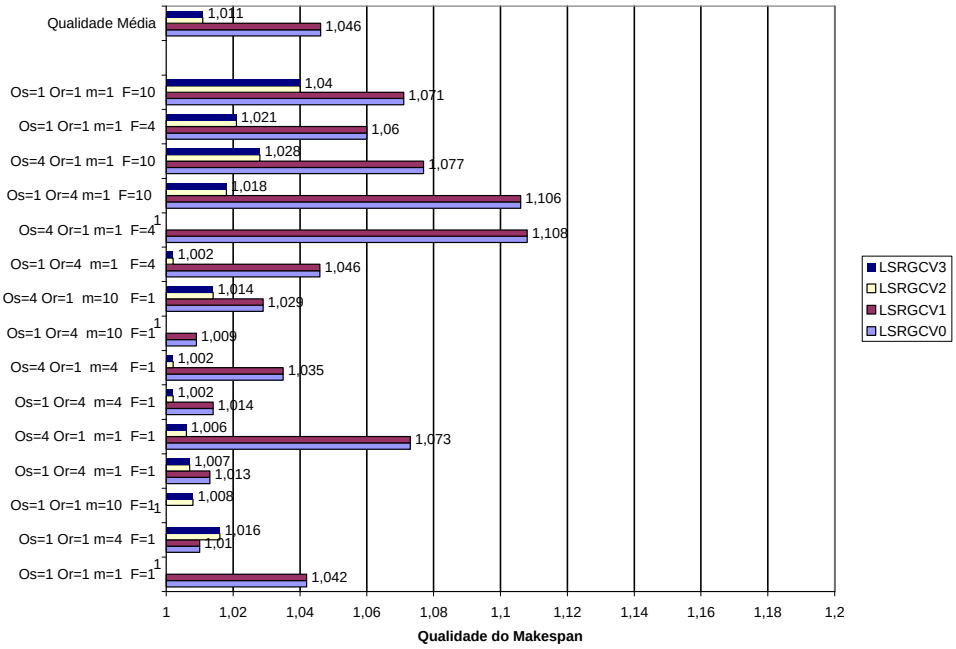

Figura 6.14: Comparação da Qualidade com P=12 - Grafos Intree

média dos *makespan* produzidos pelas quatro versões foram: LSRGCV2 e LSRGCV3 1,004; e LSRGCV0 e LSRGCV1 1,040.

A Figura 6.17 apresenta os resultados obtidos na arquitetura de 64 processadores.

Em ambientes de granulosidade grossa, as versões LSRGCV2 e LSRGCV3 atingiram 98% e as versões LSRGCV0 e LSRGCV1 com cerca de 40 pontos percentuais abaixo das primeiras, atingiram 56%. Da mesma forma, em ambientes de granulosidade fina, as versões LSRGCV2 e LSRGCV3 atingiram 93% e as versões LSRGCV0 e

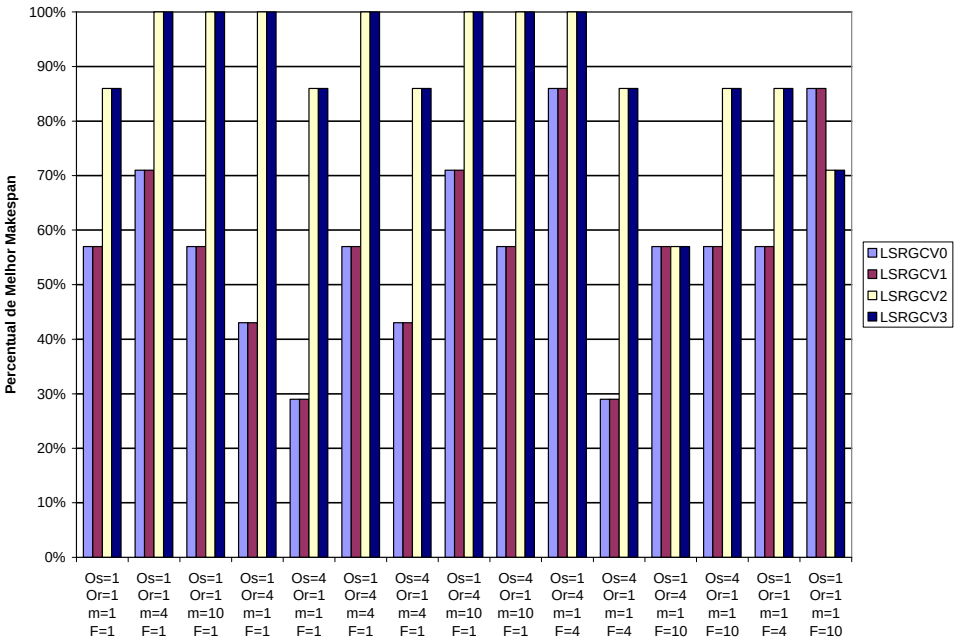

Figura 6.15: Comparação dos Percentuais com P=32 - Grafos Intree

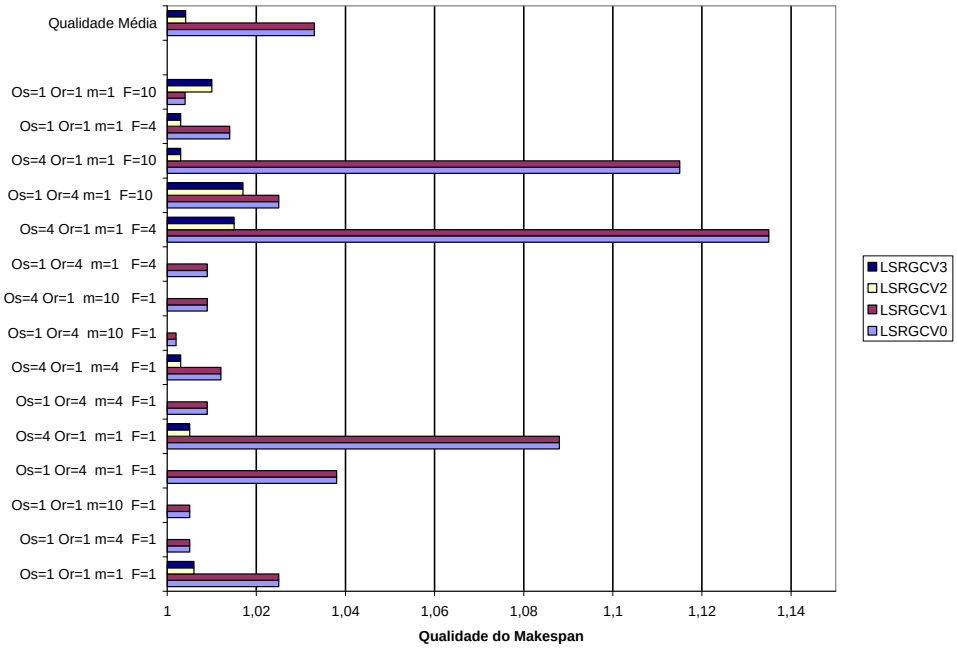

Figura 6.16: Comparação da Qualidade com P=32 - Grafos Intree

LSRGCV1, também, empatadas em 55%. Na média ponderada pelo ambiente de granulosidade grossa e fina, as versões LSRGCV2 e LSRGCV3 atingiram, cada uma, 96% e as versões LSRGCV0 e LSRGCV1 atingiram 57%. Pela Figura 6.18 é possível verificar a qualidade média dos resultados produzidos pelas quatro versões. Para as versões LSRGCV2 e LSRGCV3 a degradação em relação ao melhor *makespan* foi insignificante 1,001, para cada uma; para as versões LSRGCV0 e LSRGCV1 os valores foram bem mais significativos 1,049 de degradação do melhor valor para cada uma delas.

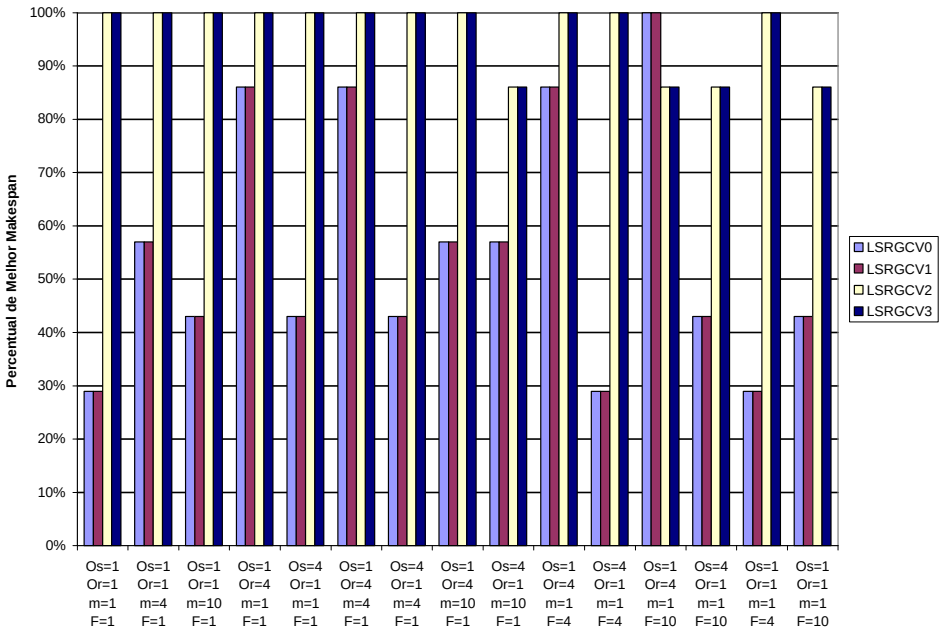

Figura 6.17: Comparação dos Percentuais com P=64 - Grafos Intree

A classificação final das versões para esta classe de grafos, considerando as quatro arquiteturas, indicou ambas as versões LSRGCV2 e LSRGCV3 com 87% e ambas as versões LSRGCV0 e LSRGCV1 com 51%. Da mesma forma, a qualidade média dos resultados, considerando s quatro arquiteturas foi de 1,009 para as versões LSRGCV2 e LSRGCV3; e 1,043 para as versões LSRGCV0 e LSRGCV1. O empate dos resultados observados entre as versões LSRGCV2 e LSRGCV3, melhores colocadas, foi devido a nesta topologia de grafos cada tarefa possuir no máximo um sucessor imediato, assim o envio das mensagens ao término da reserva R ou ao fim da sobrecarga foi irrelevante. Além disso, existindo no máximo uma sobrecarga em cada reserva R a etapa de ordenação não beneficiou o escalonamento. A superio-

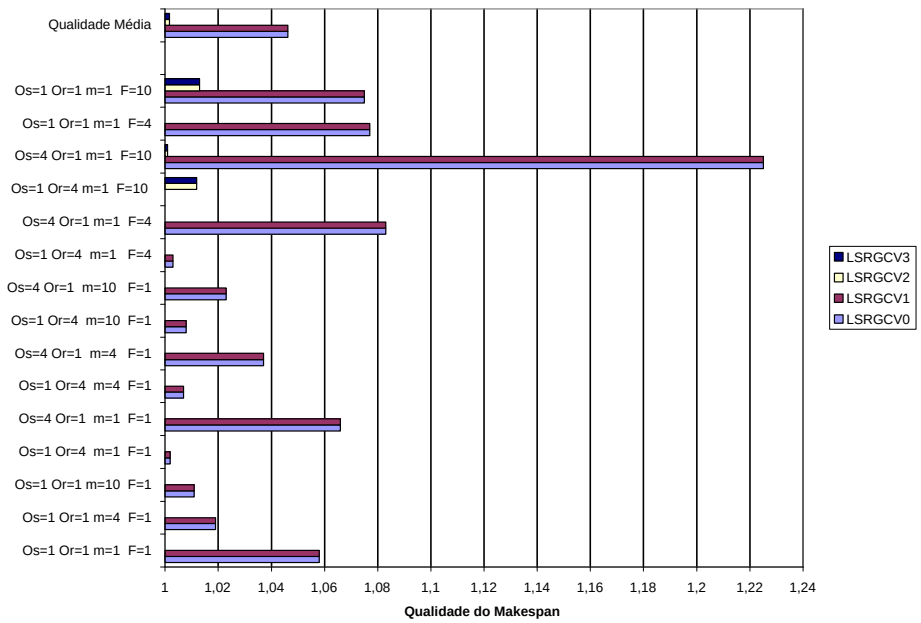

Figura 6.18: Comparação da Qualidade com P=64 - Grafos Intree

ridade de resultados, dessas duas versões em relação às concorrentes LSRGCV0 e LSRGCV1 foi decorrente de ser realizada a coleta de lixo a cada passo do escalonamento. Com isso tarefas sucessoras foram, na grande maioria dos casos, escalonadas em um processador que já alojava algum dos seus dois predecessores imediatos.

Na abordagem comparativa pelas quatro arquiteturas heterogêneas disponibilizadas as versões se destacaram em dupla, LSRGCV0 e LSRGCV1; e a dupla LSRGCV2 e LSRGCV3. Conforme ilustra a Figura 6.19, é possível perceber que as versões LSRGCV2 e LSRGCV3 aproveitam a escalabilidade dos processadores disponibilizados. Uma vez que gradativamente aumentam o percentual de casos onde atingiram o melhor *makespan*. Da mesma forma que foi descrito para os grafos da classe de diamantes, também, foi verificado o comportamento das versões quando submetidas a ambientes de processadores homogêneos. A Figura 6.20 ilustra as comparações. Nesse ambiente, nas quatro arquiteturas as versões LSRGCV2 e LSRGCV3, utilizando o mesmo número de processadores, atingiram o mesmo *makespan* para cada grafo. Na arquitetura homogênea de 12 processadores as versões LSRGCV0 e LSRGCV1 em três grafos *intree* não conseguem atingir o melhor *makespan* gerado pelas versões LSRGCV2 e LSRGCV3.

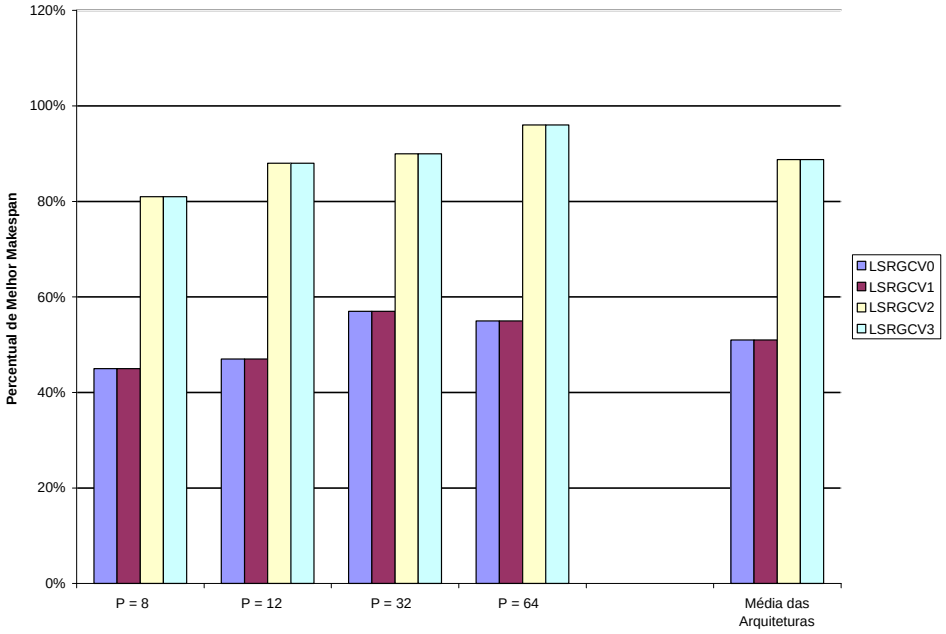

Figura 6.19: Comparação por Arquitetura Heterogênea - Grafos Intree

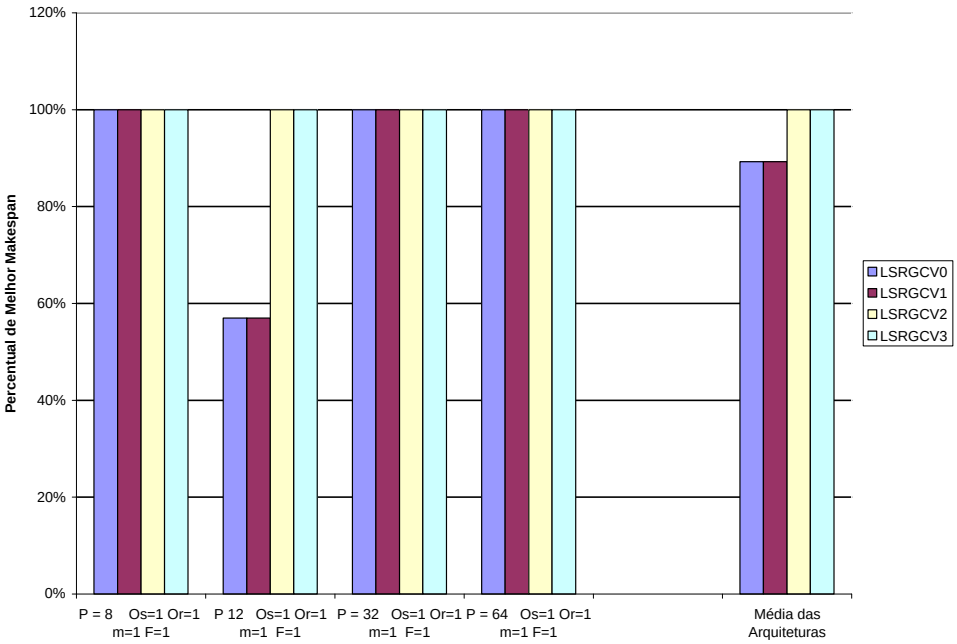

Figura 6.20: Comparação por Arquitetura Homogênea - Grafos Intree

6.4 Grafos OutTree

Com esta classe de 8 grafos foram executados 1920 experimentos, considerando as quatro arquiteturas heterogêneas disponibilizadas. A Figura 6.21 ilustra os valores

apresentados pelas versões na arquitetura de 8 processadores. Considerando os testes efetuadas em ambiente de granulosidade grossa, na média, a LSRGCV3 atingiu 86%, a LSRGCV2 72%, a LSRGCV1 e a LSRGCV0 33%. No ambiente de granulosidade fina, na média, a LSRGCV3 atingiu 94%, a LSRGCV2 69%, a versão LSRGCV0 44% e a LSRGCV1 42%. Na média ponderada, considerando os dois ambientes, a LSRGCV3 atingiu 89%, a LSRGCV2 71%, a LSRGCV0 37%, e a LSRGCV1 36%. De acôrdo com a Figura 6.22 a degradação média obtida pelo conjunto de todos os testes foi de 1,005 para a LSRGCV3, 1,009 para a LSRGCV2, 1,083 para a LSRGCV0 e 1,089 para a LSRGCV1.

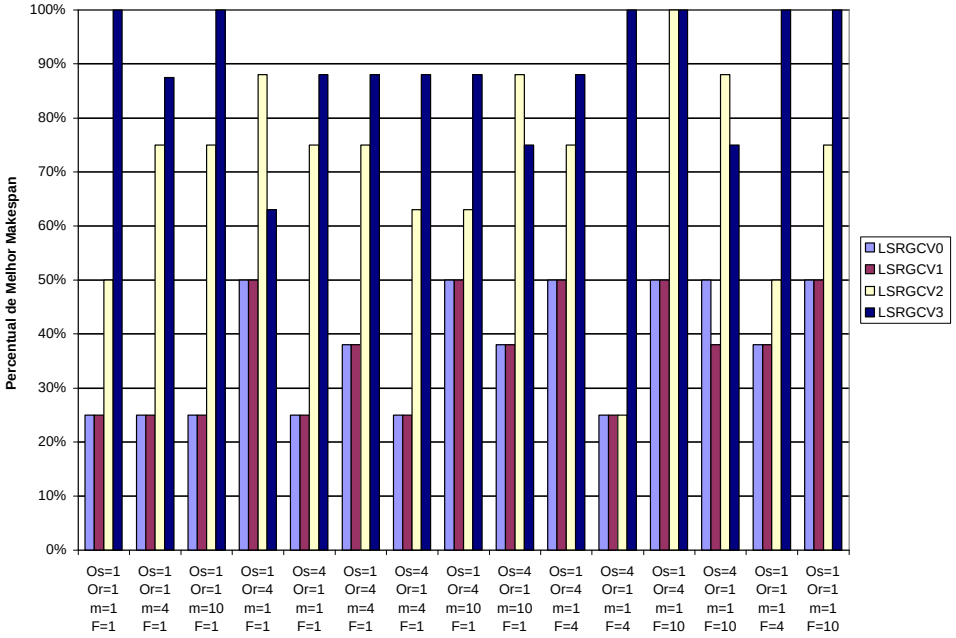

Figura 6.21: Comparação dos Percentuais com P=8 - Grafos OutTree

Conforme ilustra a Figura 6.23 usando 12 processadores, na média pelo ambiente de granulosidade grossa, a versão LSRGCV3 atingiu 86%, a LSRGCV2 75%, a LSRGCV1 e a LSRGCV0 31%. Na média pelo ambiente de granulosidade fina, houve uma inversão na colocação das duas versões em melhores posições, a versão LSRGCV2 liderou com 80%, a seguir a LSRGCV3 com 71%. Com resultados inferiores estão as versões LSRGCV1 com 44% e a LSRGCV0 com 40%. Na média ponderada, pelos dois ambientes, a LSRGCV3 atingiu 80%, a LSRGCV2 77%, a LSRGCV1 36% e a LSRGCV0 35%. Na Figura 6.24 são apresentados os valores

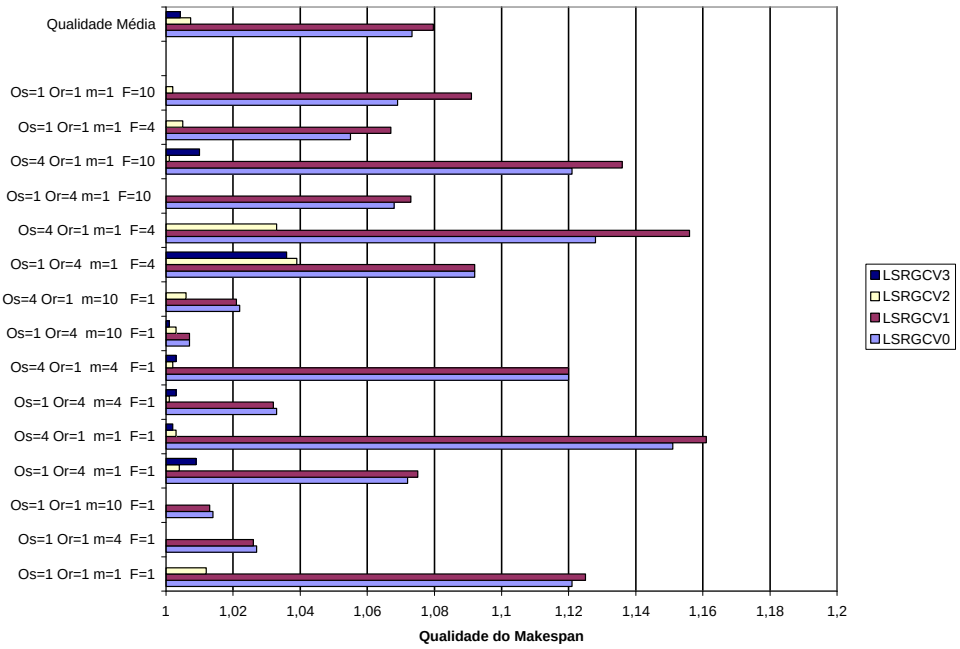

Figura 6.22: Comparação da Qualidade com P=8 - Grafos OutTree

de qualidade de resultados das quatro versões. Na média as versões LSRGCV3 e LSRGCV2 empataram em 1,007, a LSRGCV0 apresentou 1,094 e a LSRGCV1 apresentou 1,101.

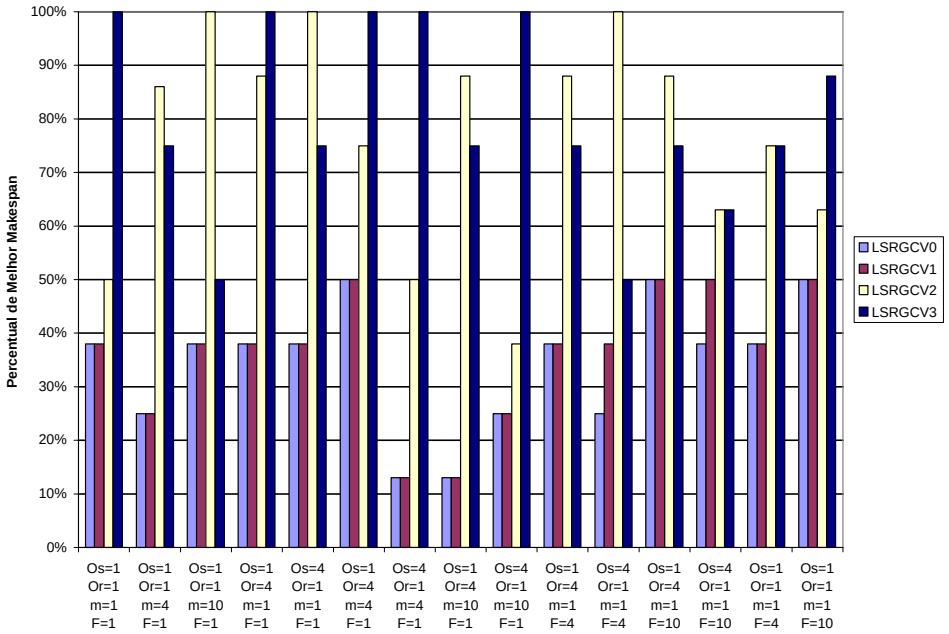

Figura 6.23: Comparação dos Percentuais com P=12 - Grafos OutTree

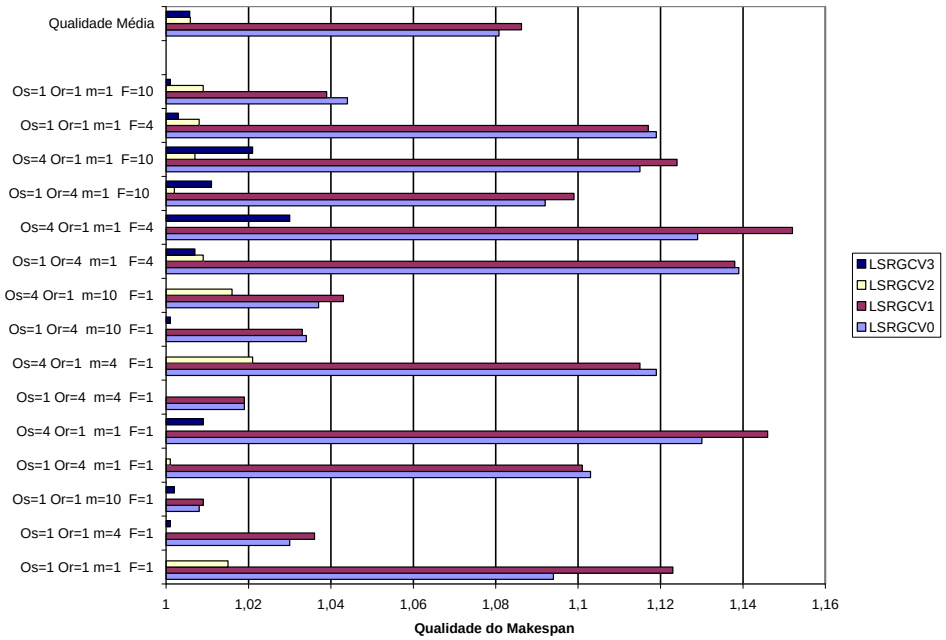

Figura 6.24: Comparação da Qualidade com P=12 - Grafos OutTree

A Figura 6.25 apresenta os percentuais atingidos por cada versão pelas 12 testes consideradas. Na média pelo ambiente de granulosidade grossa a LSRGCV2 atingiu 86%, a LSRGCV3 73%, a LSRGCV1 43% e a LSRGCV0 41%. No ambiente de granulosidade fina na média, a versão LSRGCV2 continua liderando com 79%, a versão LSRGCV3 atingiu 67%, a LSRGCV1 41% e a LSRGCV0 40%. Sintetizando os resultados pela média ponderada dos dois ambientes a LSRGCV2 atingiu 83%, a LSRGCV3 71%, a LSRGCV1 42% e a LSRGCV0 41%. Analisando a qualidade dos resultados pela Figura 6.26 é possível observar que na média, as duas versões LSRGCV2 e LSRGCV3 apresentaram degradações em relação ao melhor bastante próximas; a LSRGCV2 com 1,009 e a LSRGCV3 com 1,014. Por outro lado, a versão LSRGCV0 apresentou 1,113 e a versão LSRGCV1 1,145.

Os resultados dos testes realizados na arquitetura de 64 processadores, estão ilustrados na Figura 6.27. Em ambientes de granulosidade grossa, em média a LSRGCV2 apresentou 93%, seguir está a versão LSRGCV3 com 85%, a LSRGCV0 com 50% e a LSRGCV1 com 47%. Em ambientes de granulosidade fina, na média a LSRGCV2 atingiu 75% dos casos, a seguir está a LSRGCV3 com 71%, a LSRGCV0 com 54% e a LSRGCV1 com 46%. Na média ponderada dos ambientes a versão

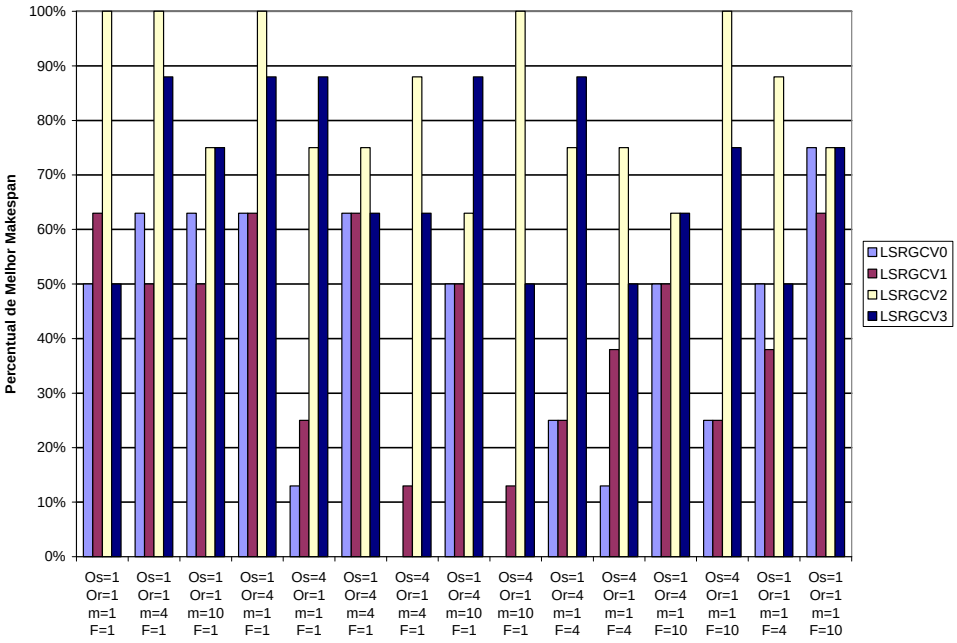

Figura 6.25: Comparação dos Percentuais com P=32 - Grafos OutTree

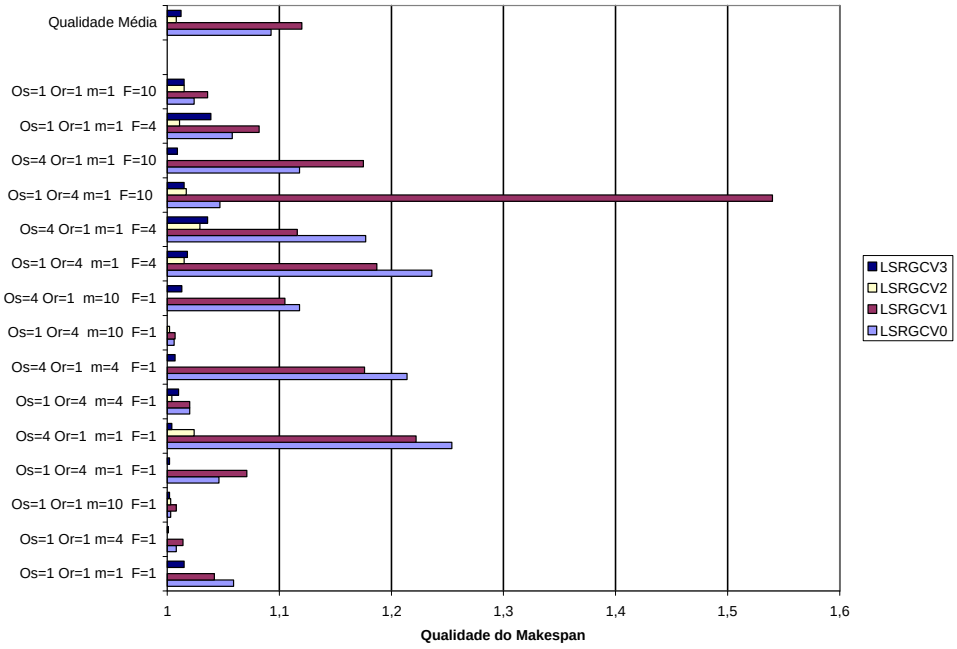

Figura 6.26: Comparação da Qualidade com P=32 - Grafos OutTree

LSRGCV2 atingiu 86%, a LSRGCV3 79%, a LSRGCV0 52% e a LSRGCV1 47%. A Figura 6.28 ilustra a qualidade dos resultados gerados por cada versão; na média apesar da LSRGCV3 ficar em segundo lugar quando se avaliou o percentual de melhor *makespan* foi a que apresentou menor degradação em relação ao melhor

makespan, 1,012. A LSRGCV2 apresentou 1,021, a LSRGCV0 apresentou 1,054 e a LSRGCV1 apresentou 1,082.

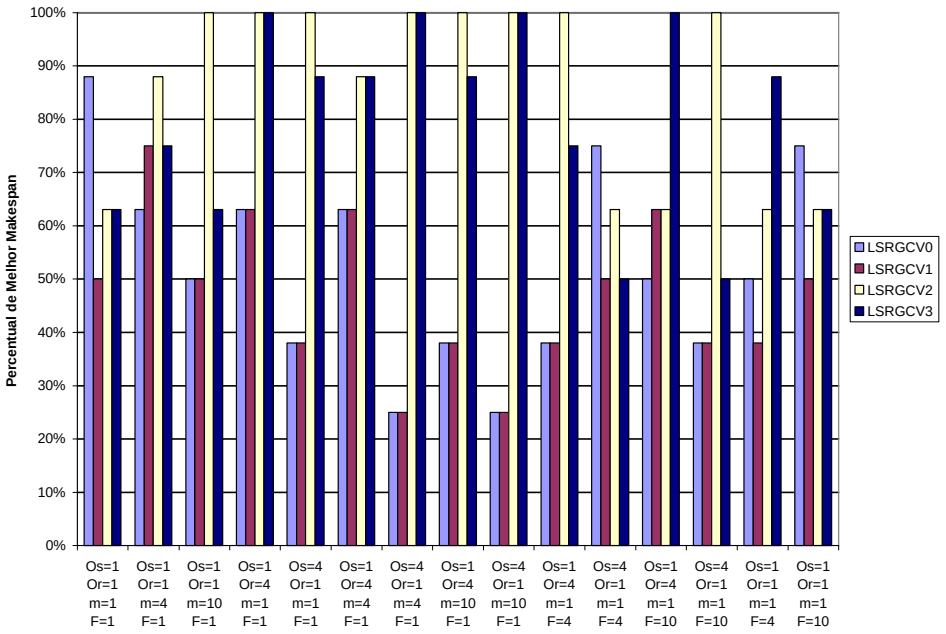

Figura 6.27: Comparação dos Percentuais com P=64 - Grafos OutTree

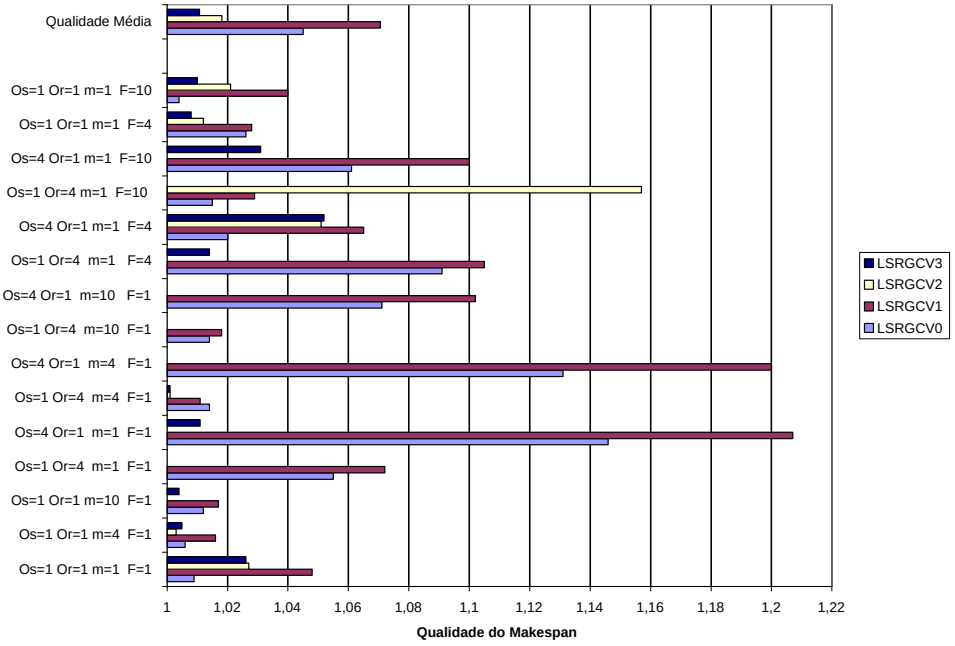

Figura 6.28: Comparação da Qualidade com P=64 - Grafos OutTree

A classificação final, nesta classe de grafos outtree, resultou nos seguintes percentuais: 80% para a LSRGCV3, 79% para a LSRGCV2, e 41% para cada uma

das versões LSRGCV1 e LSRGCV0. Os índices de qualidade dos resultados foram: 1,009 para a LSRGCV3, 1,011 para a LSRGCV2, 1,086 para a LSRGCV0, e 1,104 para a LSRGCV1. Assim, as estratégias que adotaram a política de realizar a coleta de lixo a cada passo surtiram em maiores benefícios para o escalonamento. Devido a topologia destes grafos, onde cada tarefa tem dois sucessores, as reservas R, eram relativamentes pequenas. Com a coleta de lixo realizada a cada escalonamento de tarefa, menores ficavam com a alocação de uma das duas tarefas sucessoras no mesmo processador que já alojava seu predecessor imediato. Nos casos onde isto não ocorreu, a ordenação das sobrecargas, proposta na versão LSRGCV3, permitiu o escalonamento de tarefas importantes para a minimização do *makespan*. Quando se ofereceu mais processadores, e sendo eles na maioria mais rápidos houve uma tendência a espalhar mais as tarefas. Com isso a política da LSRGCV2 foi superior a da LSRGCV3, com envio das mensagens ao fim de cada sobrecarga. Além disso, a ordenação realizada pela LSRGCV3 nem sempre resultou em sucesso devido aos vários caminhos críticos escalonados existentes em um mesmo grafo deste tipo.

Após as análises nas quatro arquiteturas, foi realizada uma comparação do comportamento, médio, de cada versão. Conforme ilustra a Figura 6.29, é possível verificar que nas duas primeiras arquiteturas de 8 e 12 processadores, a heterogeneidade do sistema foi favorável a versão LSRGCV3 que lidera nos percentuais de casos onde atingiu o melhor *makespan*. A mesma situação não se observou nas outras duas arquiteturas. A versão LSRGCV2 lidera sobre as demais, inclusive aumenta o percentual de casos onde atingiu o melhor *makespan* quando se enfoca a escalabilidade de 32 para 64 processadores. Em virtude disso, experimentos onde as quatro arquiteturas são homogêneas foram realizados. Os resultados estão ilustrados na Figura 6.30. Nesse ambiente, a versão LSRGCV3 lider na grande maioria dos testes realizados neste capítulo não consegue atingir bons escalonamentos. Apenas na arquitetura homogênea de 8 processadores ultrapassou as versões concorrentes. Nas outras 3 arquiteturas sofre queda, sucessiva, nos percentuais atingidos. Inclusive, nas arquiteturas de 32 e 64 processadores é ultrapassada pelas outras três versões. Na arquitetura de 64 processadores homogêneos a versão LSRGCV0 surpreende nos resultados, atingindo mais de 80% dos casos com o meslhor *makespan*. Na média fi-

nal, no ambiente homogêneo, foram observados os seguintes valores: LSRGCV2 82%; LSRGCV0 66%; LSRGCV3 63%; e LSRGCV1 56%. Com o intuito de esclarecer os valores observados foi verificado o escalonamento gerado pelas versões LSRGCV0 e LSRGCV3, para o grafo *Outtree255* de 255 tarefas. A versão LSRGCV3 perdeu no escalonamento para a versão LSRGCV0 por priorizar a sobrecarga O_{s22} ao invés da sobrecarga O_{s21} com dados transmitidos pela tarefa v_{10} para a tarefa v_{22} e v_{21}, respectivamente. Essa ordenação repercutiu num atraso para a cadeia de tarefas sucessoras da v_{21}. Além disso, nova ordenação errada é realizada pela LSRGCV3, nas duas sobrecargas de envio da tarefa v_{21}. A LSRGCV3 prioriza a sobrecarga O_{s44} que transmite dados para a v_{44} ao invés de priorizar a sobrecarga O_{s43}, como feito pela LSRGCV0. Devido a essas duas ordenações iniciais o valor final de *makespan* foi favorável para a versão LSRGCV0, 30 unidades e para a LSRGCV3 foi de 33 unidades, ambas utilizando os 64 processadores. Outros grafos onde a versão LSRGCV0 apresentou menor *makespan* que a versão LSRGCV3 foram os seguintes: *Outtree127*, LSRGCV0: 23(36), LSRGCV3: 27(33); grafo *Outtree63*, LSRGCV0: 19(18), LSRGCV3: 20(18). Onde o número representa o *makespan* e entre parenteses o número de processadores utilizados pela versão. Tais valores foram devido a ordenações inoportunas para a minimização do *makespan* pela LSRGCV3.

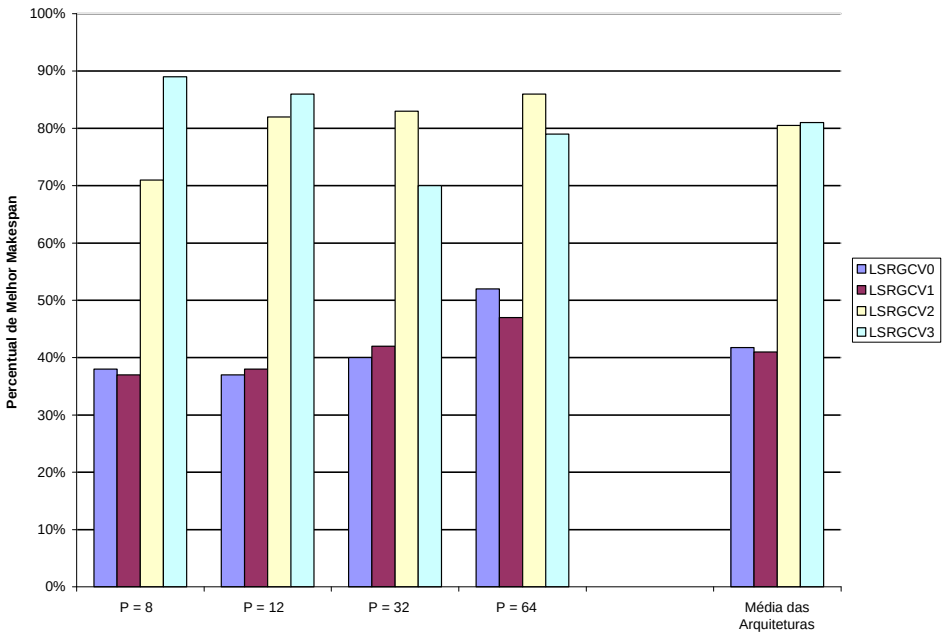

Figura 6.29: Comparação por Arquitetura Heterogênea - Grafos Outtree

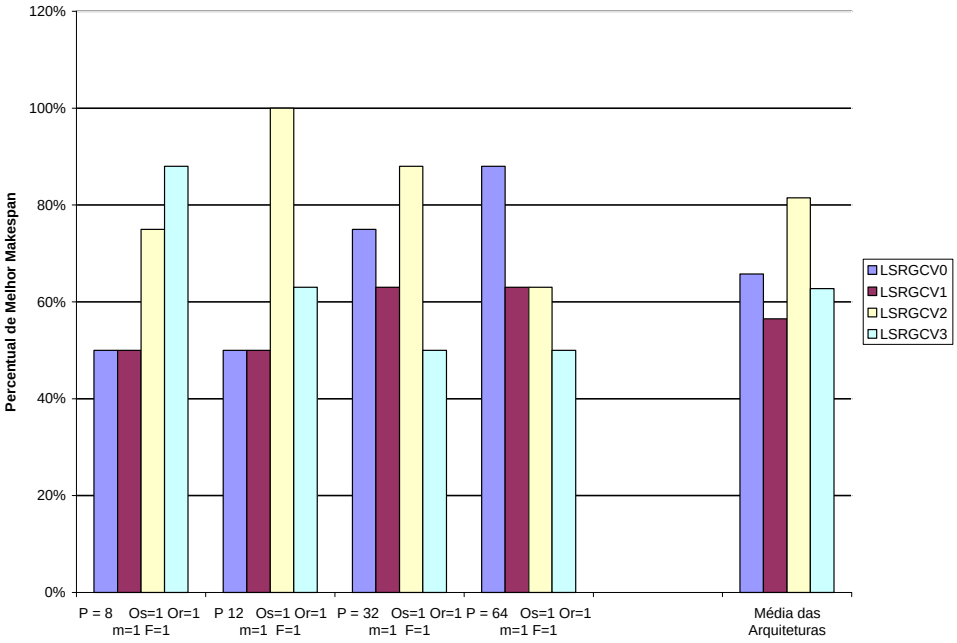

Figura 6.30: Comparação por Arquitetura Homogênea - Grafos Outtree

6.5 Grafos Randômicos

Nesta classe composta por 21 grafos foram realizados 5040 experimentos. A Figura 6.31 ilustra os resultados dos experimentos para avaliar o percentual de melhor *makespan* na arquitetura de 8 processadores. Conforme pode ser observado no ambiente de granulosidade grossa, em média, a LSRGCV3 disparou atingindo 83%, a LSRGCV2 ficou bem abaixo desse valor com 28%, mais abaixo, ainda, ficaram as versões LSRGCV1 e LSRGCV0 com 2% cada. No ambiente de granulosidade fina, os valores médios apresentados pelas versões foram: 80% e 16%, respectivamente para a LSRGCV3 e LSRGCV2; 10% para a LSRGCV0 e 5% para a LSRGCV1. A média ponderada, pelos dois ambientes, foi de 82% para a versão LSRGCV3, 23% para a LSRGCV2, 5% para a versão LSRGCV0 e 3% para a LSRGCV1. A Figura 6.32 ilustra a qualidade dos *makespan* gerados por cada versão. A de melhor degradação é a LSRGCV3 com apenas 1,004; a LSRGCV2 com 1,056, a LSRGCV0 com 1,091 e a LSRGCV1 com 1,097.

Os experimentos com 12 processadores estão ilustrados na Figura 6.33 Na média dos testes realizados em ambientes de granulosidade grossa a LSRGCV3 atingiu o

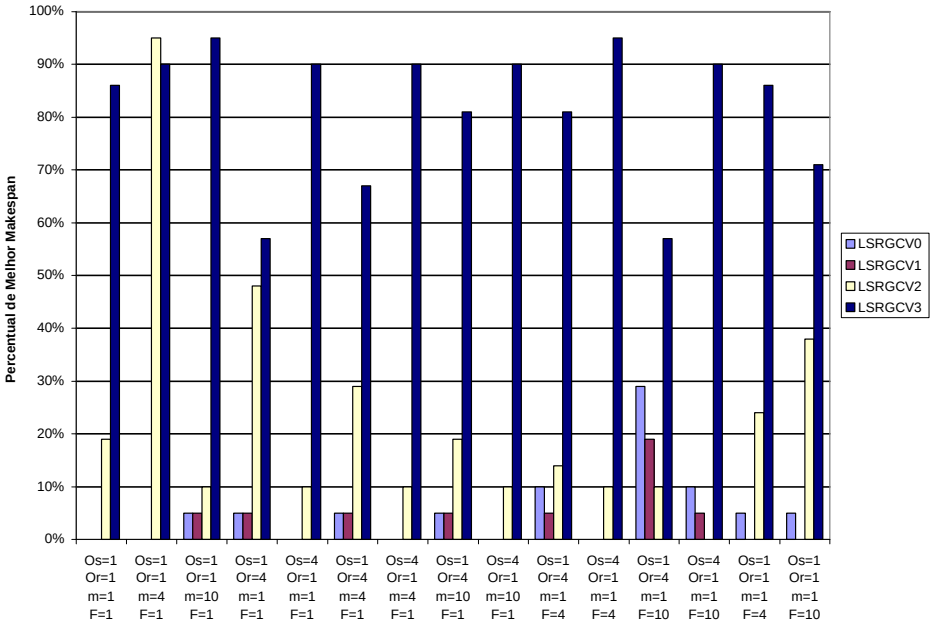

Figura 6.31: Comparação dos Percentuais com P=8 - Grafos Randômicos

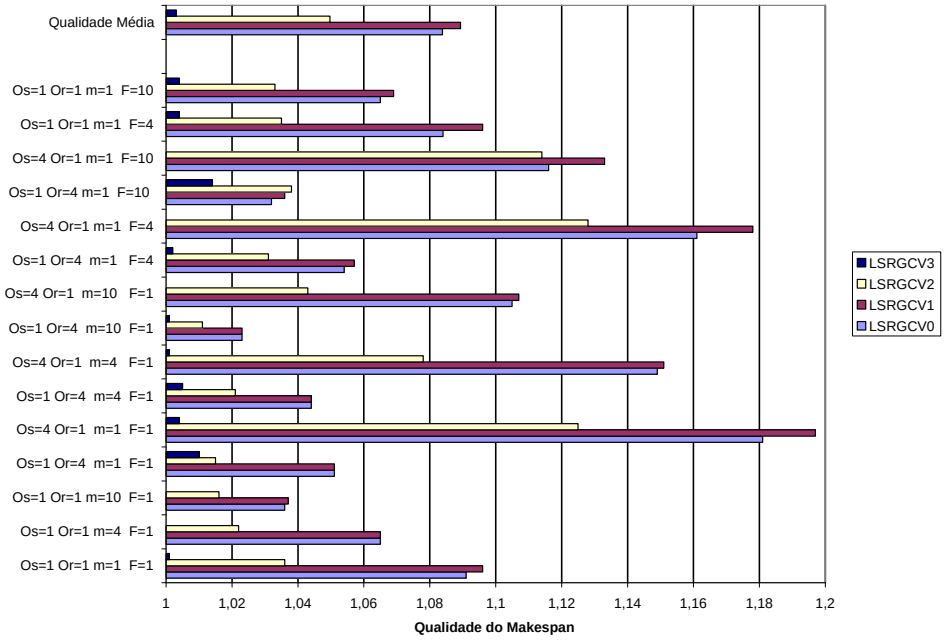

Figura 6.32: Comparação da Qualidade com P=8 - Grafos Randômicos

melhor *makespan* em 82% dos casos, a LSRGCV2 em 16%, a LSRGCV0 em 8% e a LSRGCV1 em 7%. Em ambientes de granulosidade fina, em média, a versão LSRGCV3 continua liderando com 65% dos casos, a LSRGCV2 com 30%, a LSRGCV1 com 10% e a LSRGCV0 com 9%. Considerando a média ponderada pe-

los dois ambientes a LSRGCV3 atingiu 75%, a LSRGCV2 22%, a LSRGCV0 9% e a LSRGCV1 8%. Os experimentos comparativos da qualidade dos *makespan* produzidos pelas versões estão ilustrados na Figura 6.34. Na média a LSRGCV3 apresentou 1,007 de degradação ao melhor *makespan*, a LSRGCV2 apresentou 1,055, e 1,068 e 1,076 para a LSRGCV0 e LSRGCV1, respectivamente.

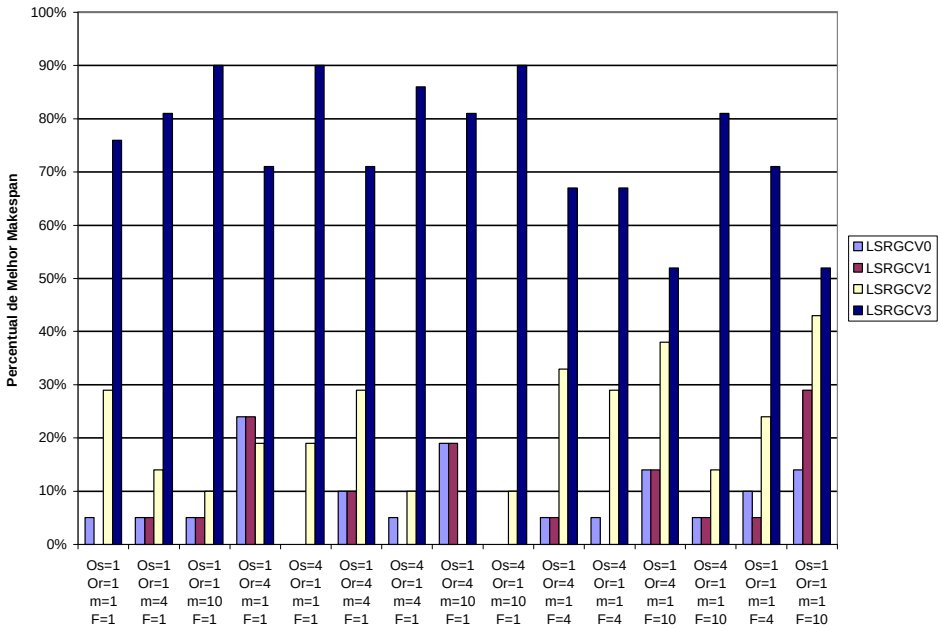

Figura 6.33: Comparação dos Percentuais com P=12 - Grafos Randômicos

Os experimentos usando 32 processadores estão ilustrados na Figura 6.35. Em ambientes de granulosidade grossa, na média dos testes realizados a LSRGCV3 atingiu 62% dos casos com melhor *makespan*; a LSRGCV0 apresentou 28%; a LSRGCV2 24% e a LSRGCV1 17%. Em ambientes de granulosidade fina, na média dos testes realizados, a LSRGCV3 continuou liderando atingindo o melhor *makespan* em 44% dos casos; a LSRGCV2 atingiu 43%; a LSRGCV0 23% e a LSRGCV1 12%. Considerando a média ponderada, pelos dois ambientes, a LSRGCV3 atingiu 55%, a LSRGCV2 32%, a LSRGCV0 26% e a LARGCV1 15%. A qualidade dos resultados, de cada versão, pode ser observada na Figura 6.36 na média, a LSRGCV3 apresentou 1,016 de degradação do melhor *makespan*, a LSRGCV2 apresentou 1.040, e as versões LSRGCV0 e LSRGCV1 apresentaram, respectivamente, 1,038 e 1,065.

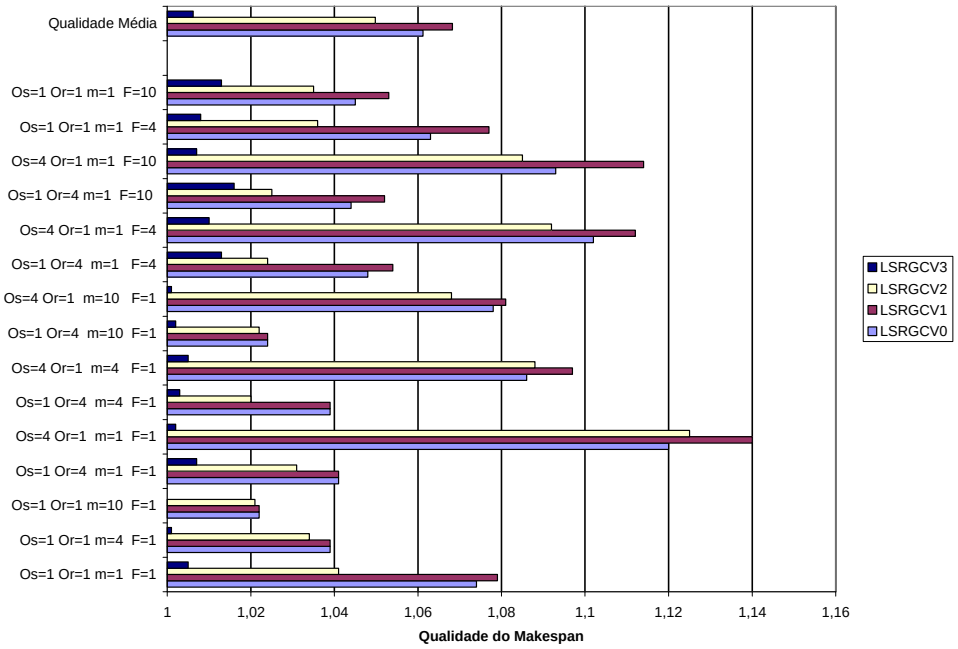

Figura 6.34: Comparação da Qualidade com P=12 - Grafos Randômicos

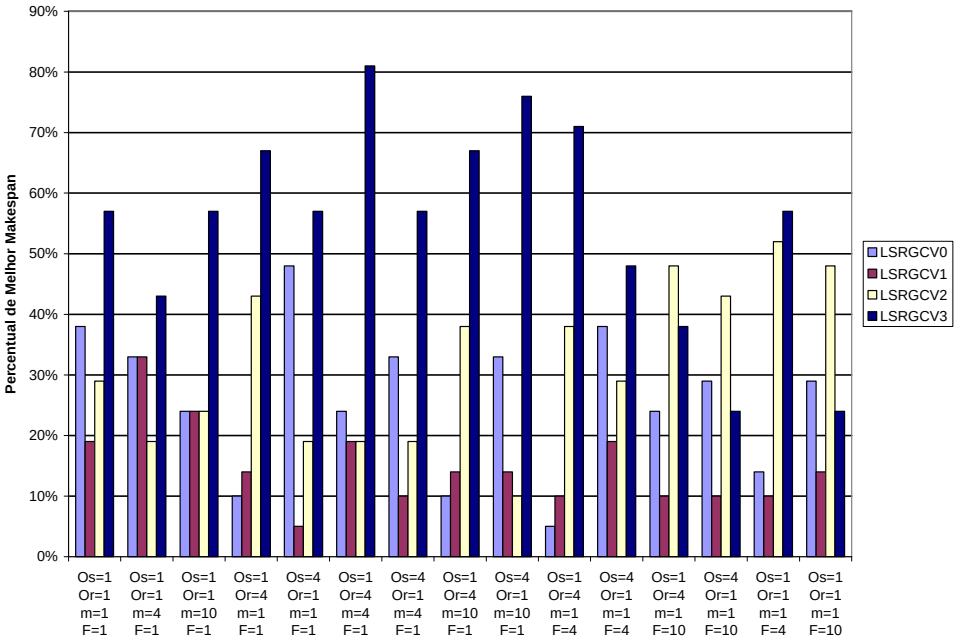

Figura 6.35: Comparação dos Percentuais com P=32 - Grafos Randômicos

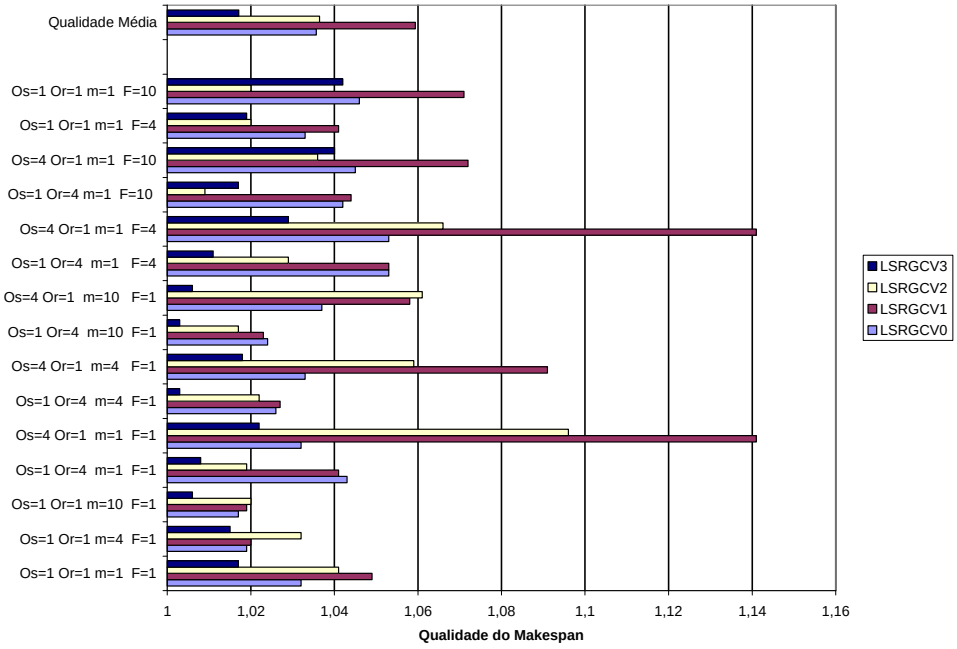

Figura 6.36: Comparação da Qualidade com P=32 - Grafos Randômicos

Os experimento usando 64 processadores estão ilustrados na Figura 6.37. Na média, em ambientes de granulosidade grossa destacou-se a LSRGCV2 com 52%, seguida da LSRGCV3 com 38%, a LSRGCV0 apresentou 39% e a versão LSRGCV1 16%. Na média pelo ambiente de granulosidade fina, a colocação das versões manteve

a mesma ordem, e os valores foram os seguintes: 57% para a LSRGCV2, 37% para a LSRGCV3, 33% para a LSRGCV0 e 12% para a LSRGCV1, respectivamente. Englobando os dois ambientes, e considerando a média ponderada, a LSRGCV2 obteve 54%, a LSRGCV3 38%, a LSRGCV0 37% e a LSRGCV1 14%. De acôrdo com a Figura 6.38 são ilustrados os valores da qualidade do *makespan* produzido por cada versão. Na média a LSRGCV3 e LSRGCV0 empataram com 1.036, a menor degradação foi para a LSRGCV2 com 1,025, e LSRGCV1 apresentou 1,084.

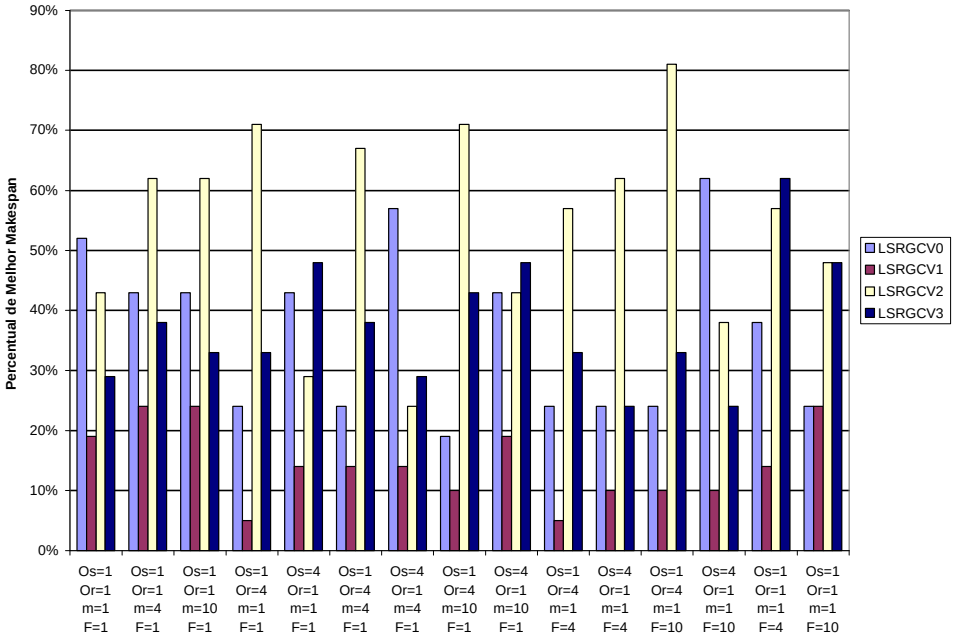

Figura 6.37: Comparação dos Percentuais com P=64 - Grafos Randômicos

A classificação final das quatro versões, considerando a média pelas quatro arquiteturas utilizadas, foi de 63% dos casos para a LSRGCV3, 33% para a LSRGCV2, 19% para a LSRGCV0, e 10% para a LSRGCV1. A média dos valores de qualidade foram de 1,015 para a LSRGCV3, 1,044 para a LSRGCV2, 1,058 para a LSRGCV0 e 1,080 para a LSRGCV1. Pela topologia desta classe de grafos, onde a maioria das tarefas possui um número relativamente elevado de sucessores, a política da LSRGCV3 através da estratégia de ordenação das sobrecargas, aliada a coleta de lixo instantânea surtiu benefícios consideráveis para a minimização do escalonamento.

Na análise com abordagem no comportamento das versões, considerando as quatro arquiteturas heterogêneas disponibilizadas foi observado que a versão LSRGCV3

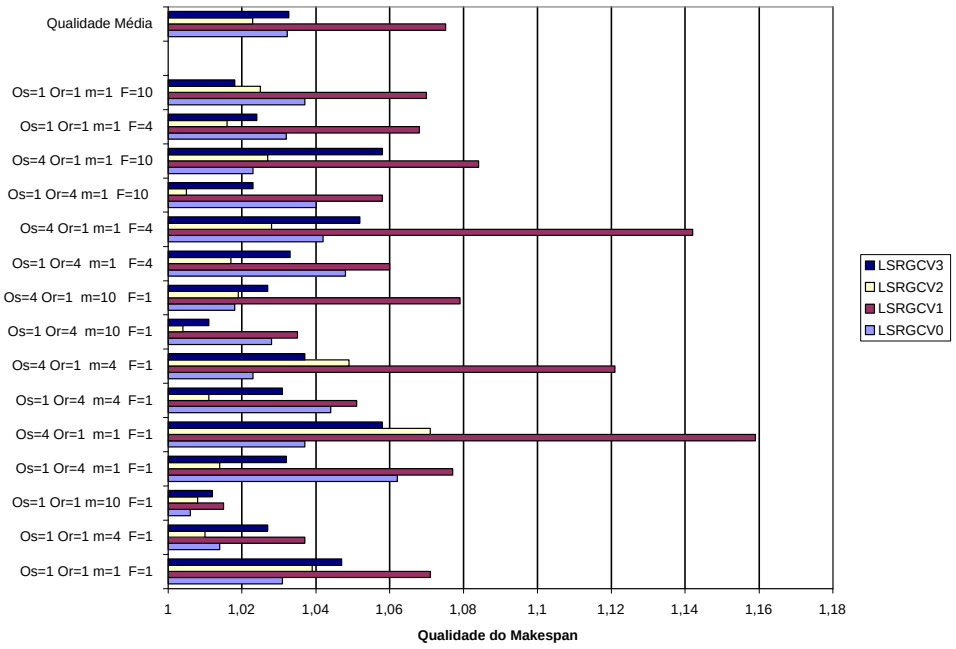

Figura 6.38: Comparação da Qualidade com P=64 - Grafos Randômicos

liderou nos resultados. A Figura 6.39 ilustra os valores. Nas duas primeiras arquiteturas a diferença de percentuais atingidos em comparação aos atingidos pela LSRGCV2 são bastante significativas. Na arquitetura de 32 processadores a LSRGCV3 sofre uma queda em relação ao percentual atingido nas arquiteturas de 8 e 12 processadores. Porém, mantém a liderança sobre as demais. É, também, possível observar que a LSRGCV2 sofre um acréscimo de percentual na arquitetura de 32 que se torna mais significativo na arquitetura de 64 processadores. Inclusive, na arquitetura de 64 processadores a LSRGCV2 é a de maior percentual. Fato interessante é que nessa arquitetura a LSRGCV0 quase atinge o percentual apresentado pela LSRGCV3. De acordo com os escalonamentos observados, pode-se concluir que a ordenação das sobrecargas de envio, com base no caminho crítico escalonado realizada pela versão LSRGCV3 não foi, na maioria das vezes, bem sucedida para a minimização do *makespan*. Da mesma forma que apresentado para as outras classes de grafos, experimentos em ambientes de processadores homogêneos foram realizados. O resultado está ilustrado na Figura 6.40. Da mesma forma que observado nas arquiteturas de 8 e 12 processadores heterogêneos, a LSRGCV3 atinge os maiores percentuais nas duas arquitetutas de 8 e 12 processadores homogêneos. Fato interessante é que a

LSRGCV2 não consegue acompanhar os índices apresentados pela LSRGCV3. A versão que se destaca é a LSRGCV0 que, inicialmente, com percentual 0% na arquitetura de 8 processadores, gradativamente aumenta seus percentuais chegando a ultrapassar as outras três versões nas arquiteturas de 32 e 64 processadores. A razão é devida as ordenações das sobrecargas de envio feitas pela LSRGCV3 não resultarem em benefícios para a redução do *makespan*. A estratégia apresentada pela LSRGCV0, coleta de lixo ao final do escalonamento e não atrasar o envio das mensagens resultou, em média, em escalonamentos com menor *makespan*.

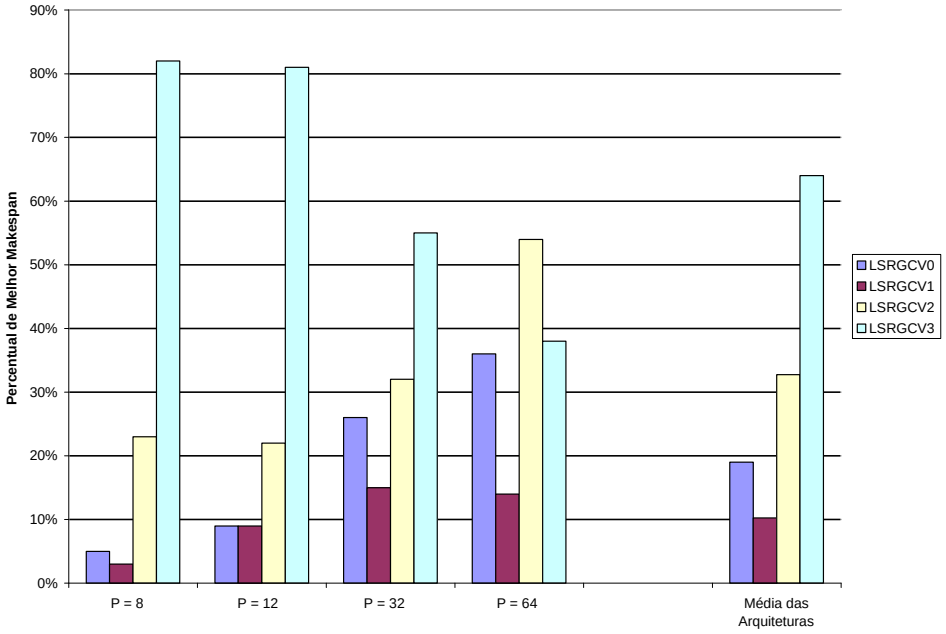

Figura 6.39: Comparação por Arquitetura Heterogênea - Grafos Randômicos

6.6 Grafos Irregulares

Com esta classe, composta por 7 grafos, foram realizados 1680 experimentos, considerando quatro arquiteturas de processadores heterogêneos. Nos experimentos realizados utilizando 8 processadores, conforme ilustra a Figura 6.41. Em ambientes de granulosidade grossa, em média, a LSRGCV3 atingiu o melhor *makespan* em 87% dos casos, a LSRGCV2 atingiu em 76%, a LSRGCV1 em 68% e a LSRGCV0 em 62%. Em ambiente de granulosidade fina, em média, a LSRGCV2 e a LSRGCV3 atingiram

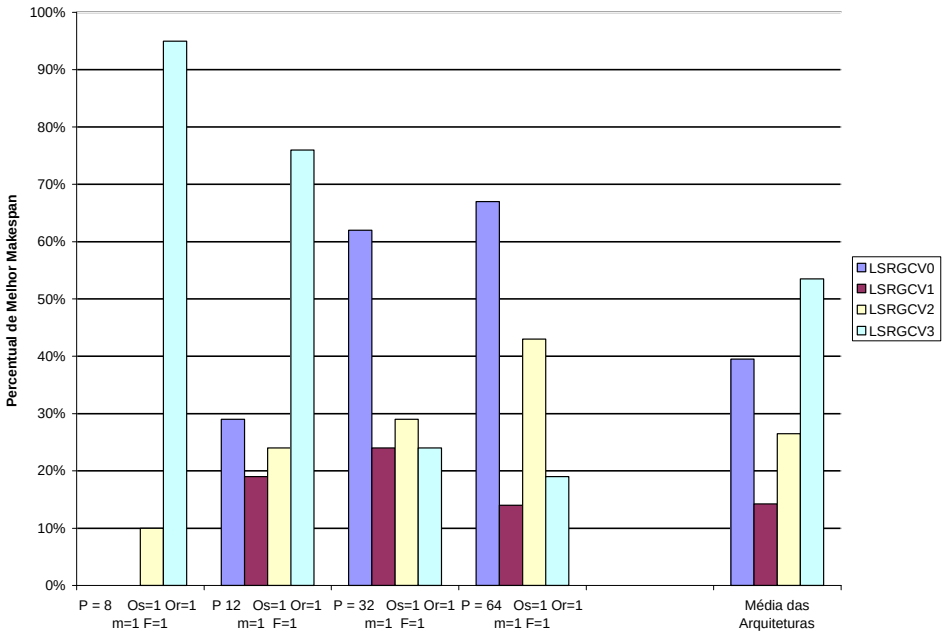

Figura 6.40: Comparação por Arquitetura Homogênea - Grafos Randômicos

o melhor *makespan* em 95% dos casos, as versões LSRGCV1 em 81% e LSRGCV0 em 79%. Na média ponderada considerando os dois ambientes, a liderança ficou para a LSRGCV3 que atingiu 90%, a LSRGCV2 84%, a LSRGCV1 73% e a LSRGCV0 69%. Na Figura 6.42 é possível observar os valores de qualidade do *makespan* produzido por cada versão, a de melhor qualidade foi a LSRGCV2 com 1,003, a LSRGCV3 apresentou valor ligeiramente superior 1.005, a LSRGCV0 apresentou 1,044 e a LSRGCV1 1,046.

A Figura 6.43 apresenta os valores obtidos nos experimentos usando 12 processadores. Considerando a média dos testes realizados, em ambientes de granulosidade grossa a LSRGCV3 atingiu melhor *makespan* em 89% dos casos, a LSRGCV2 em 73%, a LSRGCV1 em 65% e a LSRGCV0 em 57%. Na média, em ambiente de granulosidade fina, a versão LRGCV2 atingiu 88% e a LSRGCV3 86% dos casos, sendo ultrapassadas pela versão LSRGCV0 que atingiu melhor makespan em 90% dos casos e a LSRGCV1 atingiu 84%. Na média ponderada, considerando os dois ambientes, a LSRGCV3 atingiu 88%, a LSRGCV2 79%, a LSRGCV1 73% e a LSRGCV0 71%. A qualidade, em média, considerando os 12 experimentos realizados foi de 1,004 para a LSRGCV3, 1,006 para a LSRGCV2, 1,024 para a LSRGCV0 e 1,027 para a

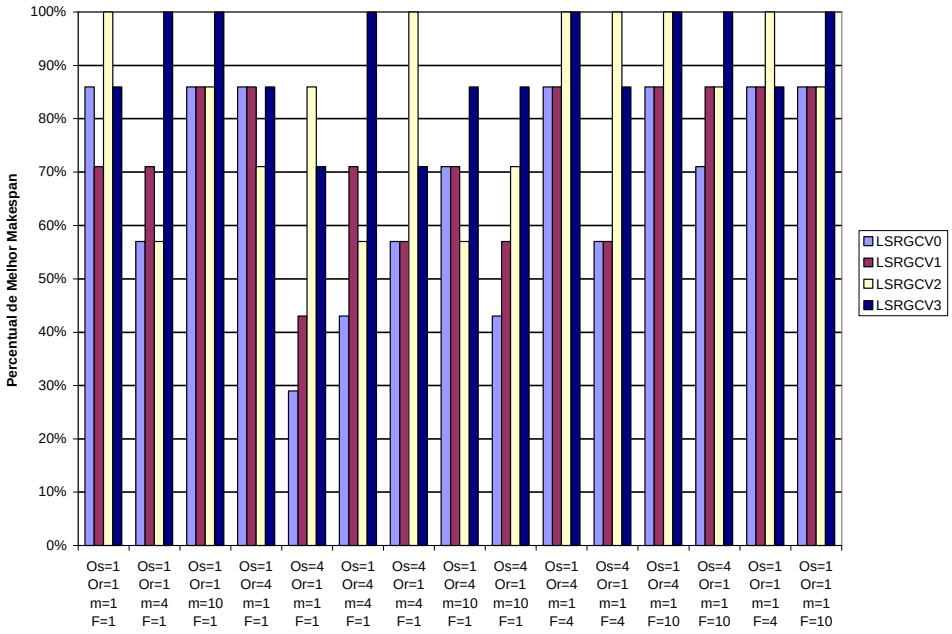

Figura 6.41: Comparação dos Percentuais com P=8 - Grafos Irregulares

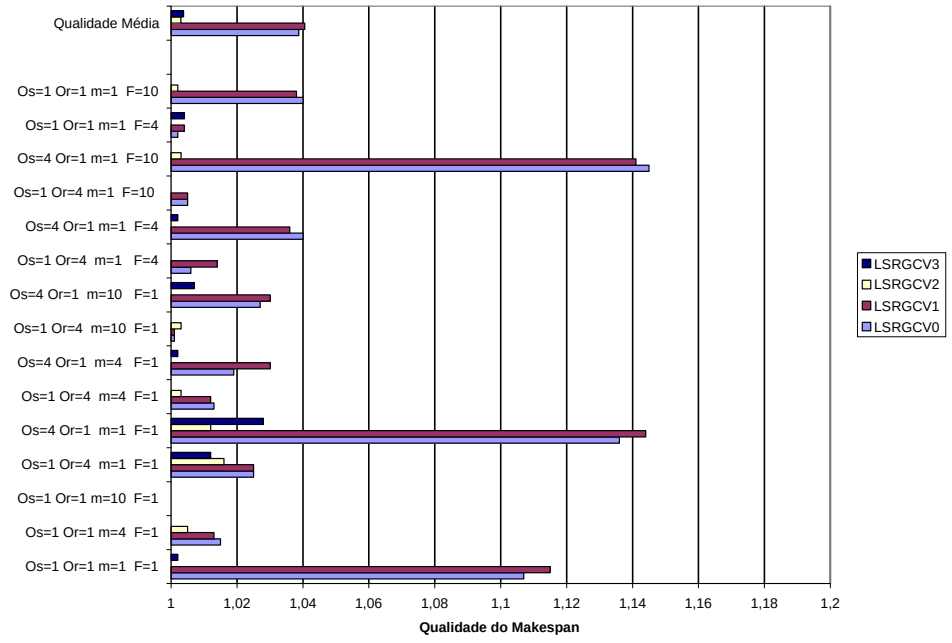

Figura 6.42: Comparação da Qualidade com P=8 - Grafos Irregulares

LSRGCV1.

A Figura 6.45 ilustra os percentuais de melhor *makespan* usando 32 processadores. É possível observar, que em média, no ambiente de granulosidade grossa a LSRGCV3 apresentou 92%, a LSRGCV2 70%, a LSRGCV1 49% e a LSRGCV0 44%.

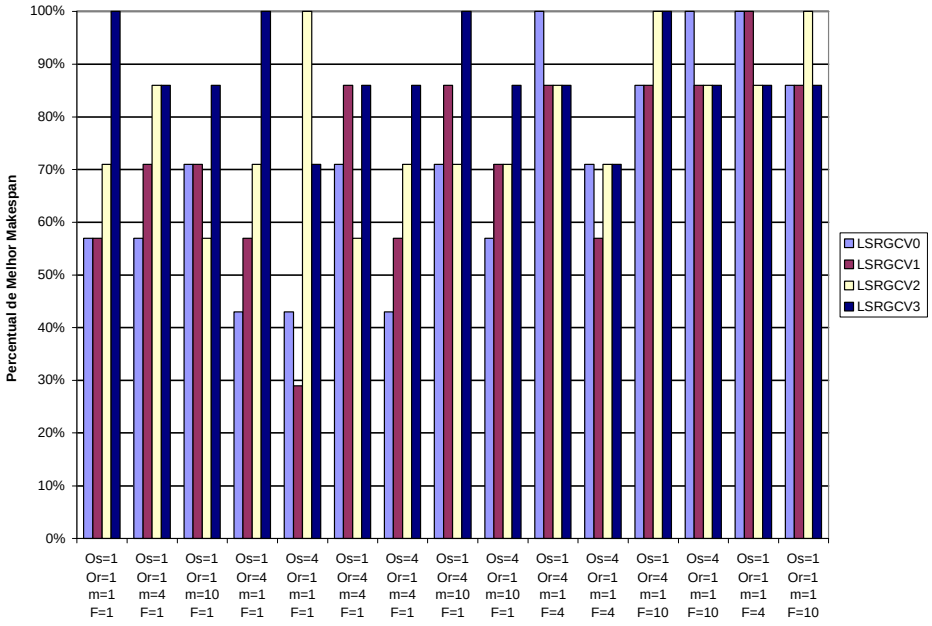

Figura 6.43: Comparação dos Percentuais com P=12 - Grafos Irregulares

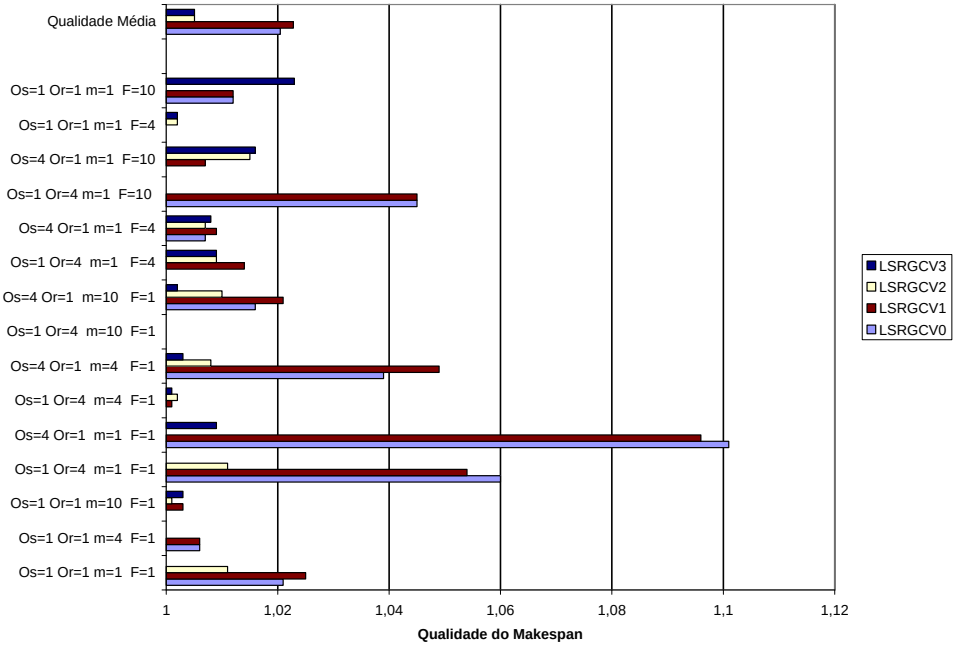

Figura 6.44: Comparação da Qualidade com P=12 - Grafos Irregulares

No ambiente de granulosidade fina, na média dos testes realizados, a LSRGCV2 ultrapassou a LSRGCV3 atingindo 93% dos casos, a LSRGCV3 ficou com 88%. As versões LSRGCV0 e LSRGCV1 atingiram, respectivamente, 74% e 71% dos casos de melhor *makespan*. Na média ponderada, considerando os dois ambientes, a

LSRGCV3 atingiu 90%, a LSRGCV2 79%, a LSRGCV1 58% e a LSRGCV0 56%. Na Figura 6.46 ilustra a comparação da qualidade do *makespan* produzido pelas versões, a LSRGCV3 foi a de menor degradação apresentando 1,003, a LSRGCV2 com 1,018, e ambas as versões LSRGCV0 e LSRGCV1, com cada uma, 1,082 e 1,083, respectivamente.

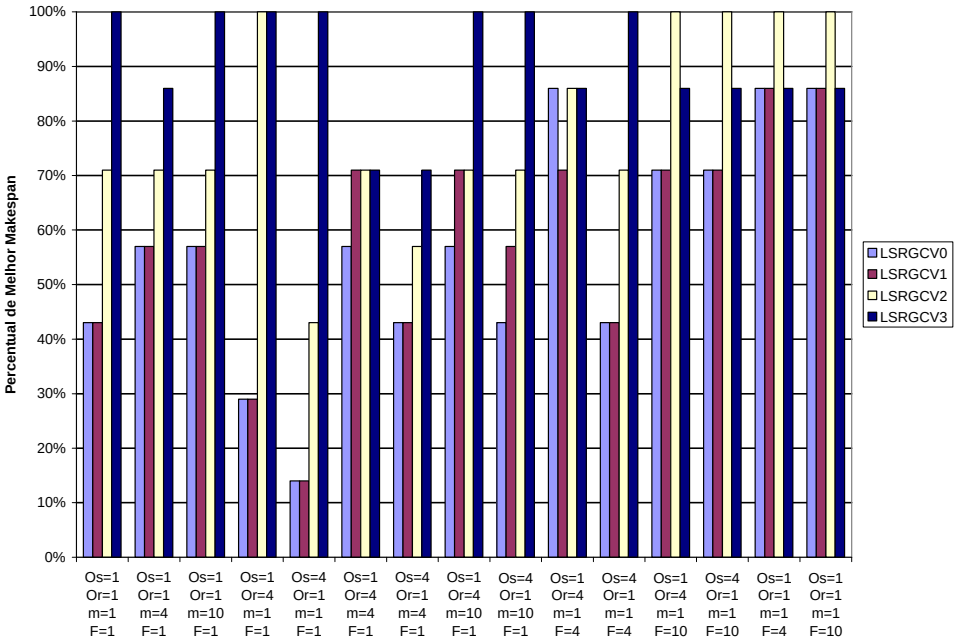

Figura 6.45: Comparação dos Percentuais com P=32 - Grafos Irregulares

Os resultados dos testes realizados com 64 processadores são ilustrados na Figura 6.47. Considerando a média dos melhores *makespan* em ambientes de granulosidade grossa, a LSRGCV3 atingiu 92% dos casos, a LSRGCV2 69% e empatadas em 55% ficaram as versões LSRGCV0 e LSRGCV1. No ambiente de granulosidade fina, em média a LSRGCV2 atingiu 93%, a LSRGCV0 91%, a LSRGCV3 88% e a LSRGCV1 84%. A média ponderada considerando os dois ambientes foi de 90% dos casos de melhor *makespan* para a LSRGCV3, 79% para a LSRGCV2, 69% para a LSRGCV0 e 67% para a LSRGCV1. A Figura 6.48 apresenta a qualidade dos *makespan* produzidos por cada versão. A heurística de resultados com melhor qualidade é a LSRGCV3, na média, 1,003, uma vez que apesar de perder em um grafo o irr41 para a LSRGCV0, e em outro teste no mesmo grafo perdeu para as três versões, a diferença foi de apenas uma unidade no *makespan*. A seguir a versão

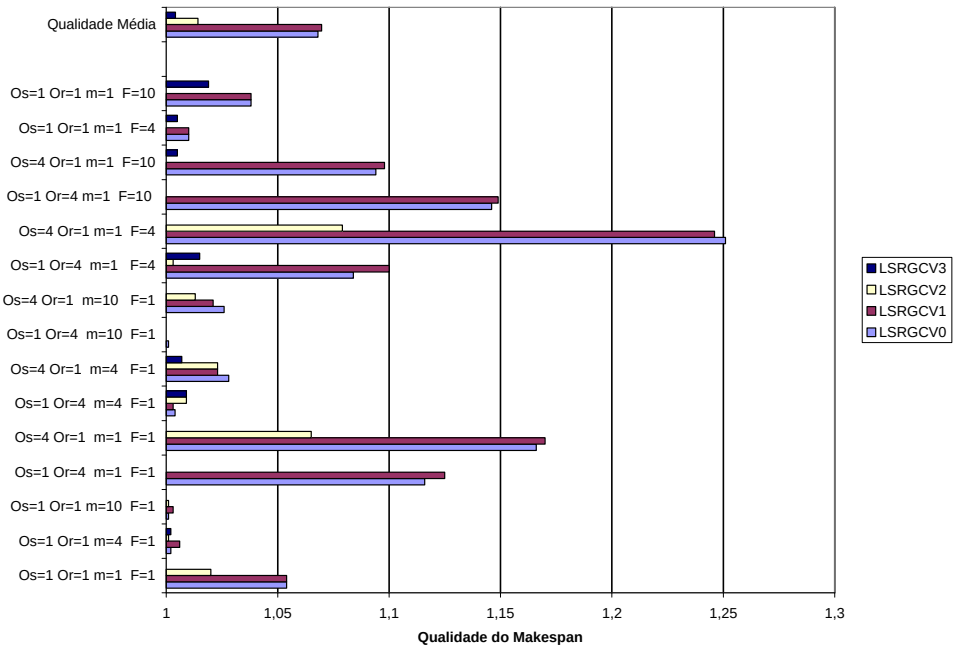

Figura 6.46: Comparação da Qualidade com P=32 - Grafos Irregulares

de melhor qualidade é a LSRGCV2 com 1,010, a LSRGCV0 apresentou 1,015 valor este que indicou uma degradação, em relação ao melhor, cerca de 5 vezes maior que a degradação da LSRGCV3. A LSRGCV1 apresentou 1,020.

A média geral obtida, considerando as quatro arquiteturas, foi: 90% para a LSRGCV3; 80% para a LSRGCV2; 68% para a LSRGCV1; e 66% para a LSRGCV0. Da mesma forma, a degradação em relação ao melhor *makespan* foi: 1,003 para a LSRGCV3; 1,009 para a LSRGCV2; 1,041 para a LSRGCV0; e 1,044 para a LSRGCV1. Pela topologia desta classe de grafos, é possível observar que o número de sucessores por tarefa não é alto. Entretanto, a comunicação entre as tarefas é bastante elevada. Dessa forma, apesar das reservas R serem medianas, o sucesso da LSRGCV3 sobre as demais, foi decorrente de enviar as mensagens (bastante custosas), ao fim da reserva R e da ordenação das sobrecargas baseadas nas características do escalonamento. No geral, observou-se uma tendência de colocar no processador que alojava uma tarefa pai a maioria das tarefas sucessoras recebedoras de dados. Tendência esta, cada vez mais acentuada com a coleta de lixo realizada a cada escalonamento de tarefa. O passo de ordenação das sobrecargas de envio existentes surtiu benefícios consideráveis uma vez que a priorização de sobrecargas para tarefas

críticas permitiu a redução no *makespan* do grafo.

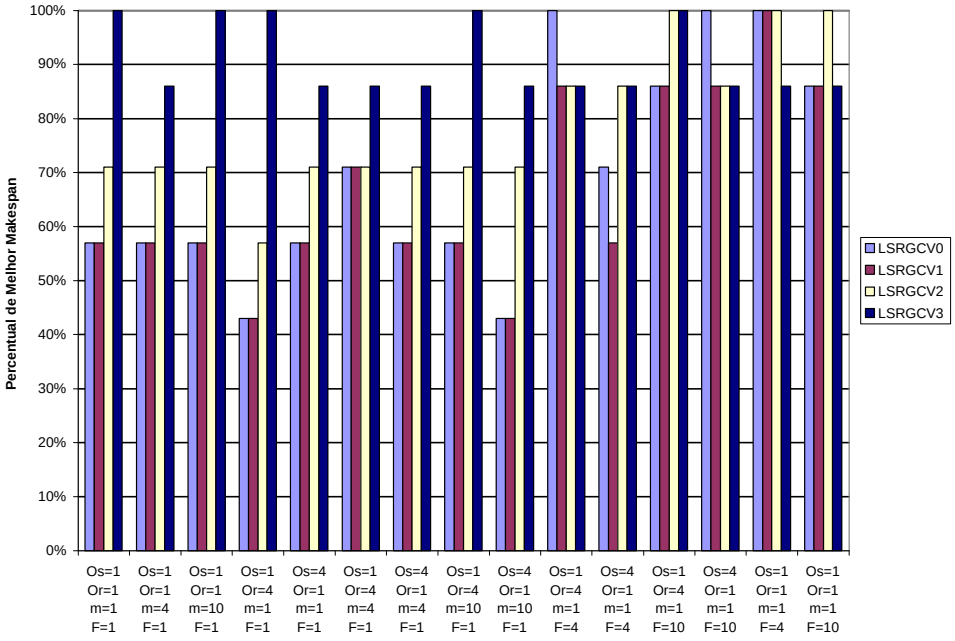

Figura 6.47: Comparação dos Percentuais com P=64 - Grafos Irregulares

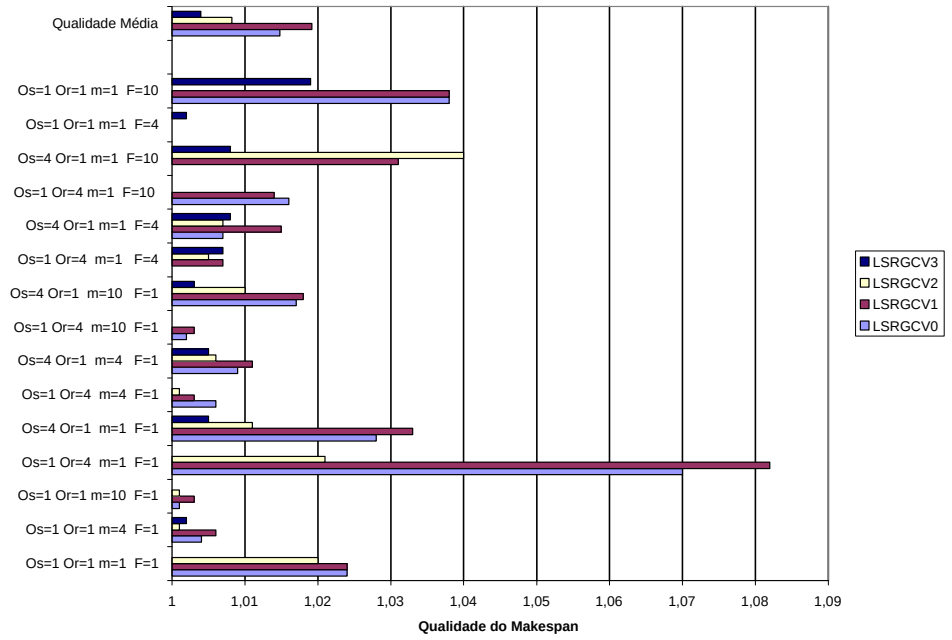

Figura 6.48: Comparação da Qualidade com P=64 - Grafos Irregulares

Após a análise em cada arquitetura, foi realizada uma comparação dos percentuais atingidos por cada versão. A Figura 6.49 ilustra os percentuais, médios, atingidos

por cada versão agrupados pelas arquiteturas. A LSRGCV3 lidera nas quatro arquiteturas. Não cabe aqui ressaltar percentuais, mas sim compreender corretamente o que isto significou. A fim de evitar precipitação na interpretação dos resultados, experimentos foram, também, realizados nas quatro arquiteturas com processadores homogêneos. Os resultados estão apresentados na Figura 6.50 e quase a mesma disposição das barras, observadas no ambiente heterogêneo, pode ser observada. Os percentuais de casos onde foi atingido o melhor *makespan* é maior, em cada uma das arquiteturas, para a versão LSRGCV3. Convém ressaltar, que esta classe é composta por grafos nos quais, em média os custos de comunicação são superiores aos custos de computação. Essa característica atibui à classe de grafos irregulares ser considerada de granulosidade fina. Dessa forma, pode-se concluir que para grafos de granulosidade fina a LSRGCV3 realiza escalonamentos mais eficientes que os realizados pelas outras três versões. Fato este de grande importância nas análises de escalonamentos realizados por heurísticas de *Construção* formuladas na técnica de *List Scheduling*.

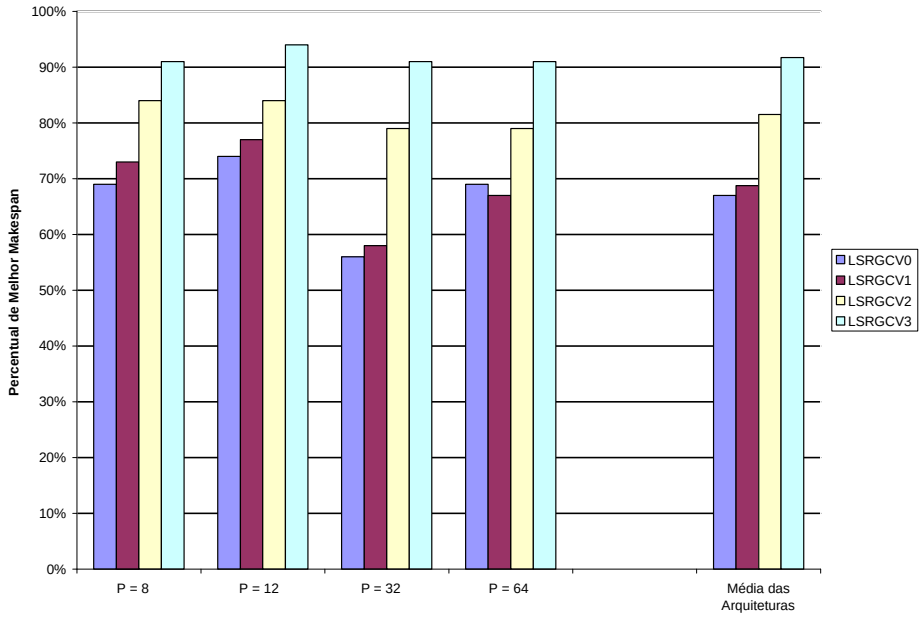

Figura 6.49: Comparação por Arquitetura Heterogênea - Grafos Irregulares

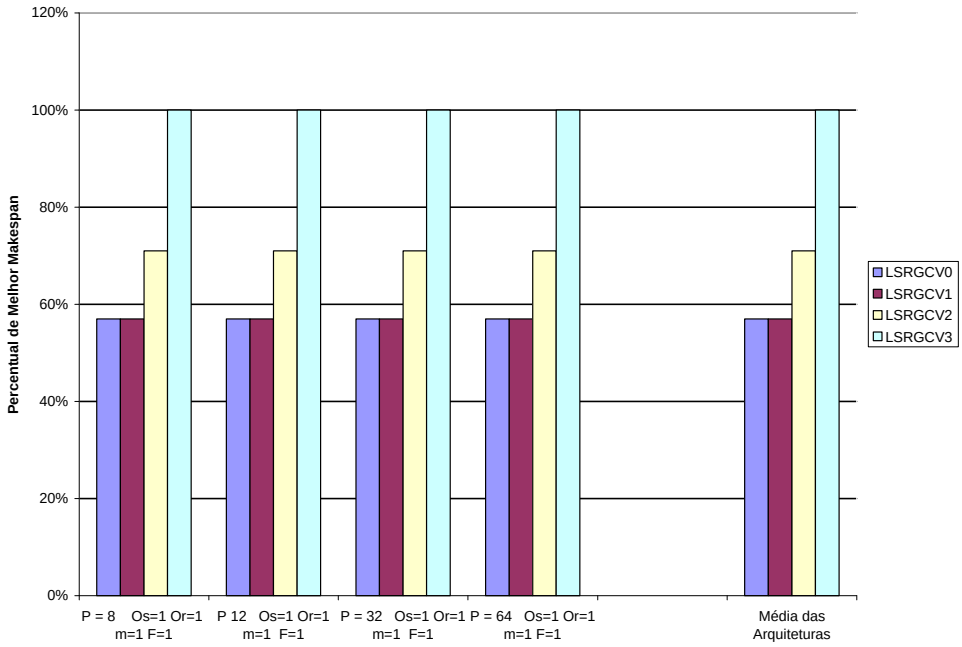

Figura 6.50: Comparação por Arquitetura Homogênea - Grafos Irregulares

6.7 Grafos KPSG (*Kwok Peer Set Graphs*)

Com esta classe, composta por 11 grafos, foram realizados 2640 testes. Os experimentos realizados usando 8 processadores estão ilustrados na Figura 6.51. Em ambientes de granulosidade grossa, na média a LSRGCV3 atingiu o melhor *makespan* em 90% dos casos, a LSRGCV2 em 81%, a LSRGCV1 em 76% e a LSRGCV0 em 70%. No ambiente de granulosidade fina, na média, a LSRGCV1 atingiu o melhor em 95%, a LSRGCV3 em 94%, a LSRGCV0 em 85% e a LSRGCV2, também em 85%. A média ponderada, considerando os dois ambientes, os resultados foram: 92% dos casos para a LSRGCV3, 83% para a LSRGGCV2, 84% para a LSRGCV1 e 76% para a LSRGCV0. Através da Figura 6.52 é possível verificar a qualidade dos resultados produzidos por cada uma das versões. A de menor degradação é a LSRGCV3 com 1,003, a seguir está a LSRGCV2 com um valor ligeiramente superior 1,005; a LSRGCV1 apresentou 1,015; e a LSRGCV0 apresentou 1,019.

Os experimentos realizados usando 12 processadores estão ilustrados na Figura 6.53. Na média, em ambiente de granulosidade grossa, a LSRGCV3 atingiu 92%, a LSRGCV2 atingiu 84%, a LSRGCV1 78% e a LSRGCV0 73%. No ambiente de gra-

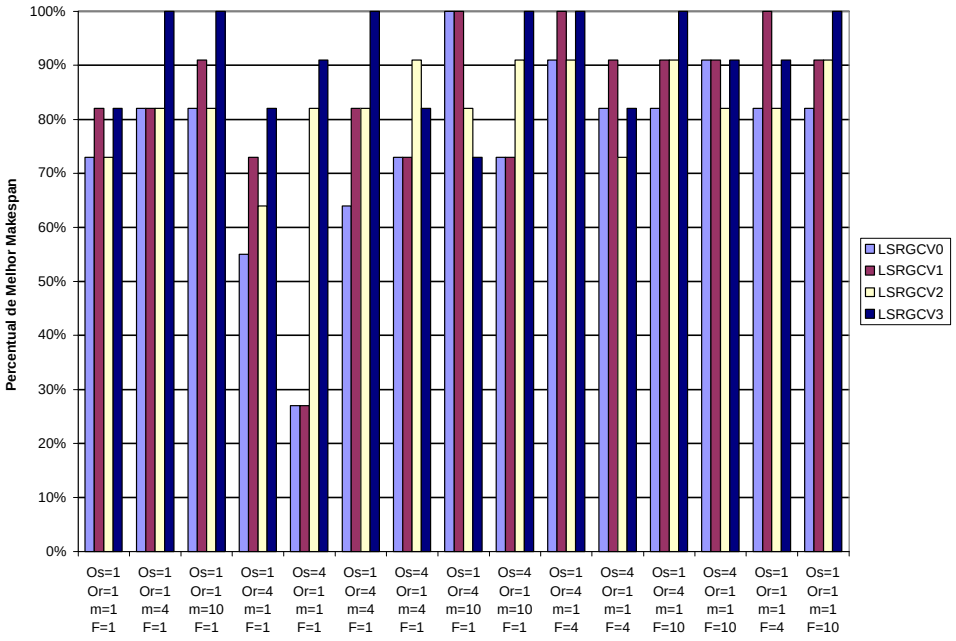

Figura 6.51: Comparação dos Percentuais com P=8 - Grafos KPSG

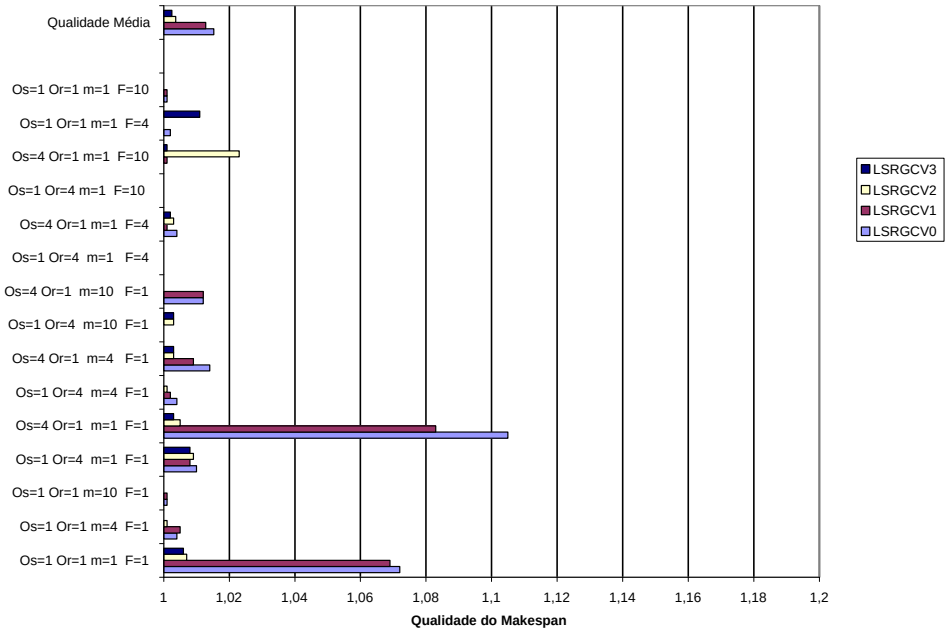

Figura 6.52: Comparação da Qualidade com P=8 - Grafos KPSG

nulosidade fina os valores em média observados foram 96% para a LSRGCV3, 88% para a LSRGCV1, 85% para a LSRGCV2 e 75% para a LSRGCV0. Na média ponderada, considerando os dois ambientes, os valores foram os seguintes: 94% para a LSRGCV3, 84% para a LSRGCV2, 82% para a LSRGCV1 e 74% para a LSRGCV0.

A Figura 6.54 apresenta os valores da qualidade dos *makespan* produzidos por cada versão. A média observada foi de 1,002 para as versões LSRGCV2 e LSRGCV3; a LSRGCV1 apresentou 1,014 e ligeiramente acima deste valor está a LSRGCV0 com 1,016.

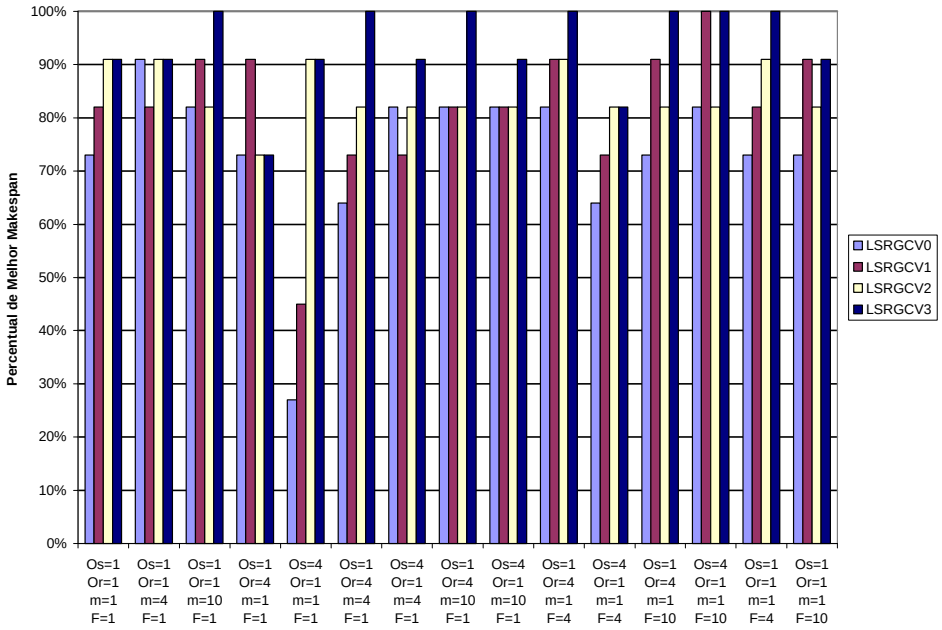

Figura 6.53: Comparação dos Percentuais com P=12 - Grafos KPSG

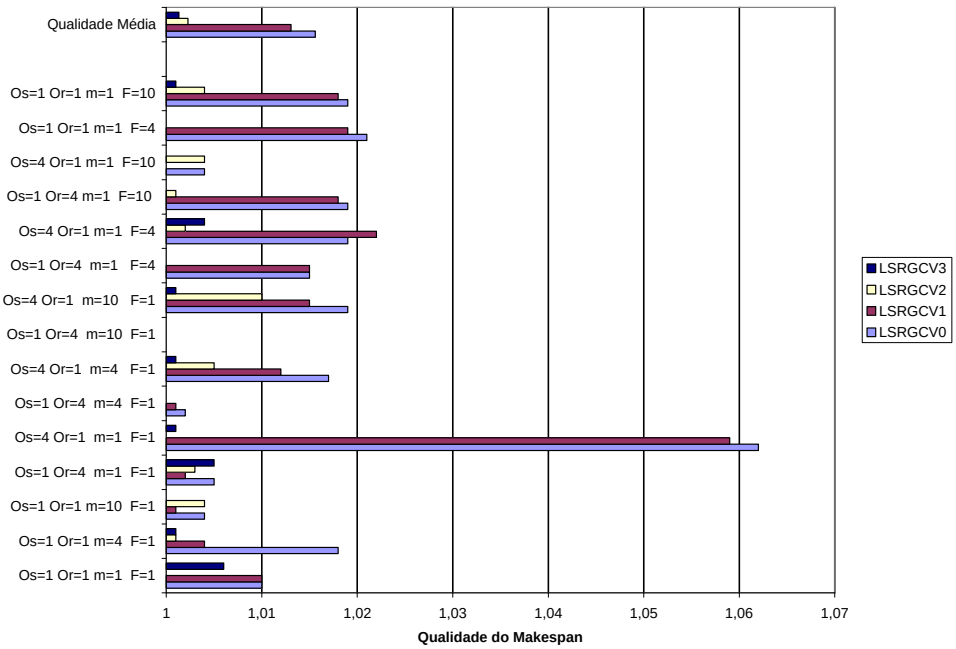

Figura 6.54: Comparação da Qualidade com P=12 - Grafos KPSG

A Figura 6.55 ilustra os experimentos relizados usando 32 processadores. Considerando o ambientes de granulosidade grossa, em média a LSRGCV3 atingiu 92% dos casos de melhor *makespan*, a LSRGCV2 atingiu 77%, a LSRGCV1 58% e a LSRGCV0 54%. No ambiente de granulosidade fina, em média, a LSRGCV2 destacou-se com o maior percentual dos casos 87%, a seguir a LSRGCV3 com 85%, a LSRGCV1 com 76% e a LSRGCV0 com 72%. A média ponderada, considerando os dois ambientes foi: 89% para a LSRGCV3; 81% para a LSRGCV2; 65% para a LSRGCV1; e 61% para a LSRGCV0. A comparação da qualidade dos *makespan* produzido por cada versão está ilustrada na Figura 6.56. A LSRGCV3, na média, apresentou a menor degradação em relação ao melhor, 1.005, a seguir com valor um pouco superior está a LSRGCV2 com 1,011, ambas as versões LSRGCV0 e LSRGCV1 com valores próximos apresentaram 1,062 e 1,058, respectivamente.

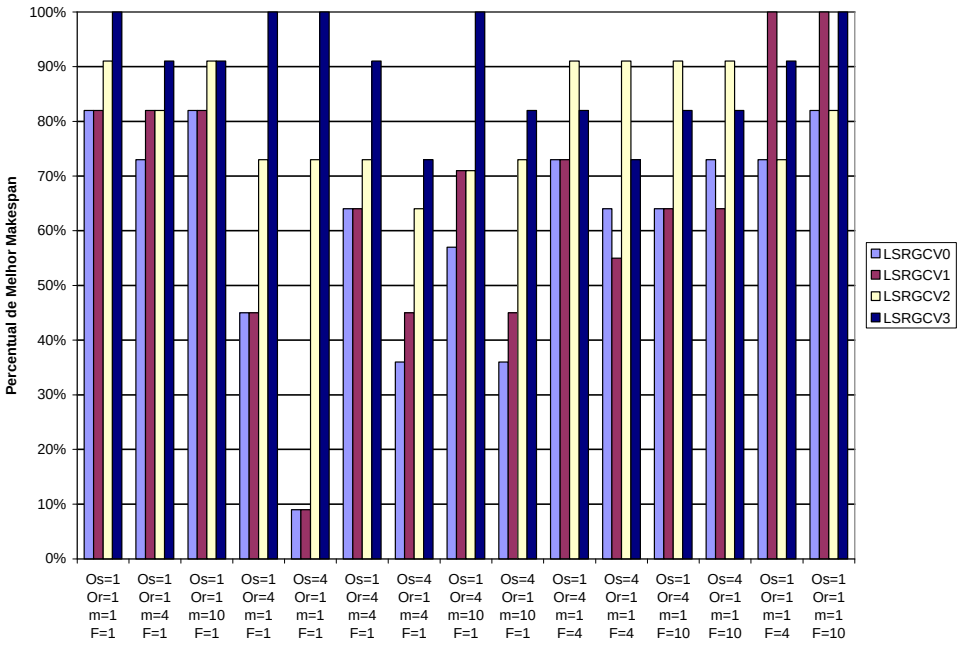

Figura 6.55: Comparação dos Percentuais com P=32 - Grafos KPSG

Os experimentos realizados usando 64 processadores estão ilustrados na 6.57. Considerando a média dos valores, no ambiente de granulosidade grossa, a versão LSRGCV3 atingiu o melhor *makespan* em 89% dos casos, a LSRGCV2 em 84%, a LSRGCV1 em 75% e a LSRGCV0 em 71%. No ambiente de granulosidade fina, a média dos valores foi de: 96% para a LSRGCV3; 90% para a LSRGCV1, 89% para

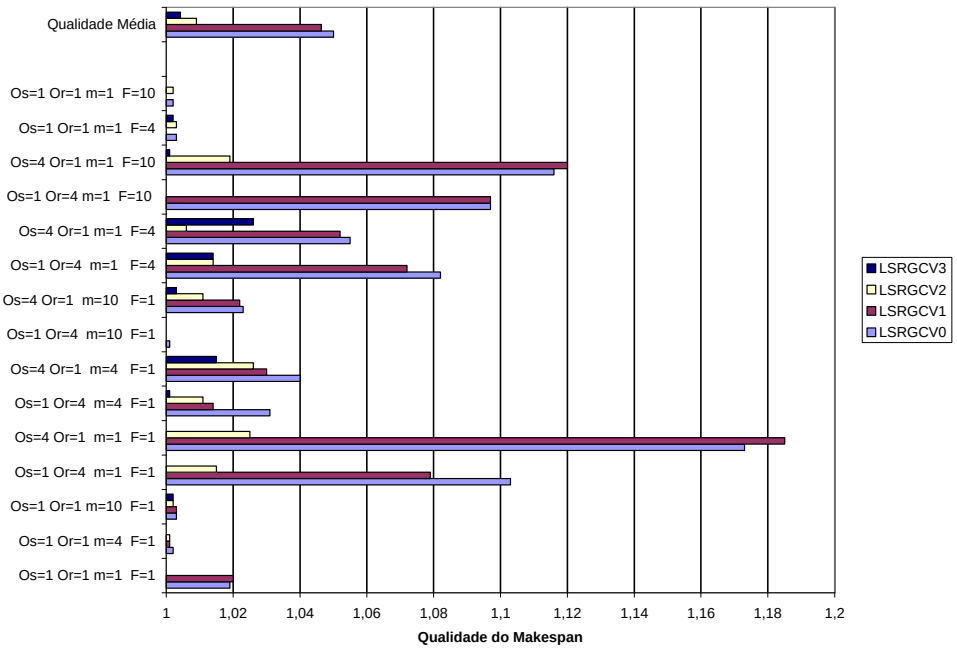

Figura 6.56: Comparação da Qualidade com P=32 - Grafos KPSG

a LSRGCV2 e 83% para a LSRGCV0. Na média ponderada pelos dois ambientes, a LSRGCV3 continua liderando com 92%; a LSRGCV2 atingiu 86%; a LSRGCV1 81% e a LSRGCV0 76%. A Figura 6.58 ilustra a qualidade dos resultados gerados por cada versão, a LSRGCV3 foi a de menor degradação com 1,003, a seguir com valor ligeiramente superior está a LSRGCV2 com 1,004, as versões LSRGCV1 e LSRGCV0 apresentaram, respectivamente, 1,007 e 1,009 de degradação em relação ao melhor *makespan*.

A classificação final das versões, considerando as quatro arquiteturas utilizadas, foi de 92% dos casos de melhor *makespan* para a LSRGCV3, 84% para a LSRGCV2, 78% para a LSRGCV1 e 72% para a LSRGCV0. A degradação em relação ao melhor *makespan* foi de 1,003 para a LSRGCV3; ligeiramente acima desse valor com 1,005 está a LSRGCV2, e com valores superiores a esses estão as versões LSRGCV1 e LSRGCV0, com respectivamente, 1,023 e 1,026. Analisando a topologia dos grafos pertencentes a esta classe, foi possível verificar que são compostos por poucas tarefas nas quais o número de sucessores não é alto. Porém, a comunicação entre as tarefas é bastante elevada. Com isso, apesar das reservas R serem relativamente medianas, foi deduzido que o sucesso da LSRGCV3 sobre as demais versões; ocor-

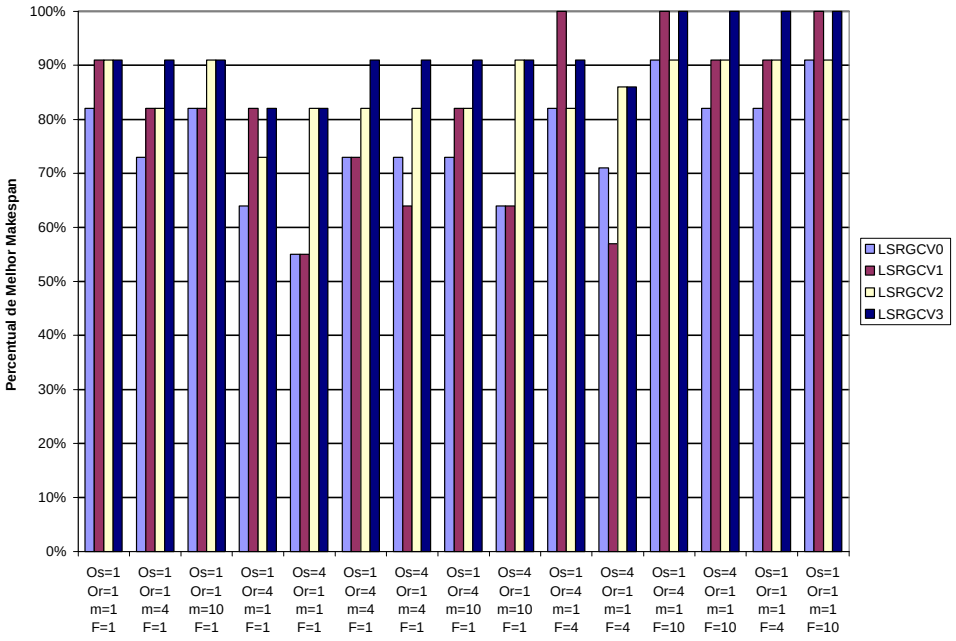

Figura 6.57: Comparação dos Percentuais com P=64 - Grafos KPSG

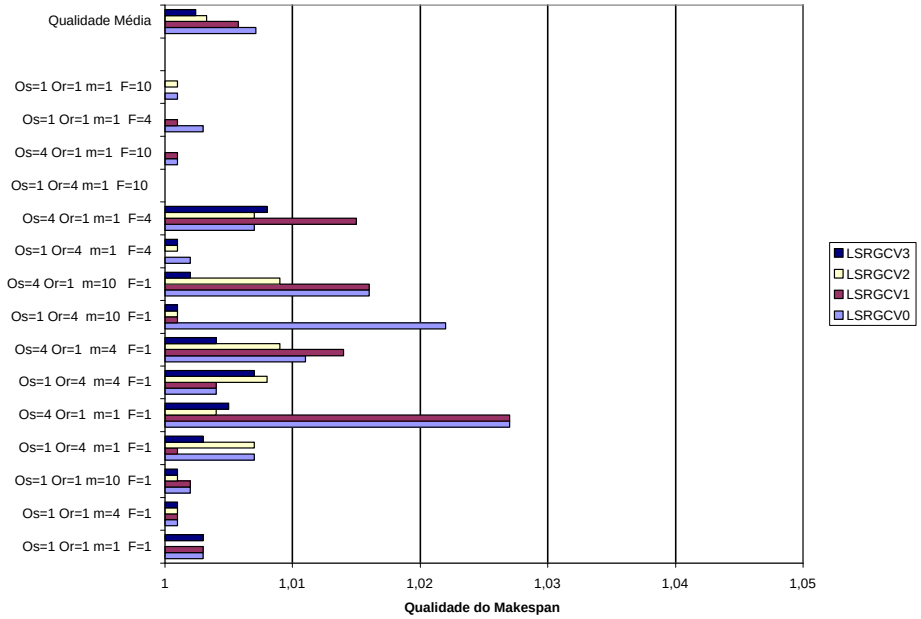

Figura 6.58: Comparação da Qualidade com P=64 - Grafos KPSG

rido em todas as quatro arquiteturas utilizadas, foi decorrente da oportunidade de ordenar as sobrecargas de envio de acôrdo com o escalonamento gerado. A fase de ordenação das sobrecargas de envio existentes, priorizando as mais críticas permitiu antecipar tarefas relevantes para a redução do *makespan* na maioria dos casos

avaliados. A segunda colocação para a LSRGCV2, em todas as quatro arquiteturas, permitiu concluir que a coleta de lixo a cada passo do escalonamento foi de grande importância.

Após a etapa de análise isolada em cada uma das quatro arquiteturas heterogêneas disponibilizadas, foram realizados experimentos para comparar o comportamento, médio, das versões. A comparação foi avaliada em relação a arquitetura heterogênea utilizada. Conforme ilustra a Figura 6.59 é possível verificar que a versão LSRGCV3 lidera com os maiores percentuais de casos onde se atingiu o melhor *makespan* nas quatro arquiteturas. Experimentos, também, foram realizados considerando as quatro arquiteturas homogêneas disponibilizadas. Os resultados estão ilustrados na Figura 6.60. É possível verificar que a versão LSRGCV3, da mesma forma ao observado no ambiente heterogêneo, lidera nas quatro arquiteturas atingindo o maior percentual (100%) de casos com o melhor *makespan*. Com base nas duas figuras apresentadas é possível concluir que a versão LSRGCV3 apresenta resultados bastante eficientes quando se avaliam grafos de granulosidade fina. Como dito anteriormente, os grafos KPSG são considerados de granulosidade fina por possuirem, em média, custos de comunicação superiores aos custos de computação. Fato importante a ser ressaltado, com base, nestas análises é que a versão LSRGCV3 é das quatro versões a mais indicada para o escalonamento de grafos considerados de granulosidade fina. Essa observação enfatiza os resultados anteriormente verificados para a classe de grafos Irregulares, considerados, também, de granulosidade fina.

6.8 Avaliação Final

Nesta seção são apresentadas as considerações sobre os resultados dos experimentos realizados com os 65 grafos nas quatro arquiteturas disponibilizadas. As avaliações estão agrupadas pela arquitetura utilizada em cada uma, são apresentads figuras ilustrativas com o percentual do melhor *makespan* de cada versão pela classe do grafo. Os índices de qualidade não serão ilustrados sob a forma de gráficos, e sim, apenas mencionados. O objetivo da avaliação final é focar as análises no aspecto

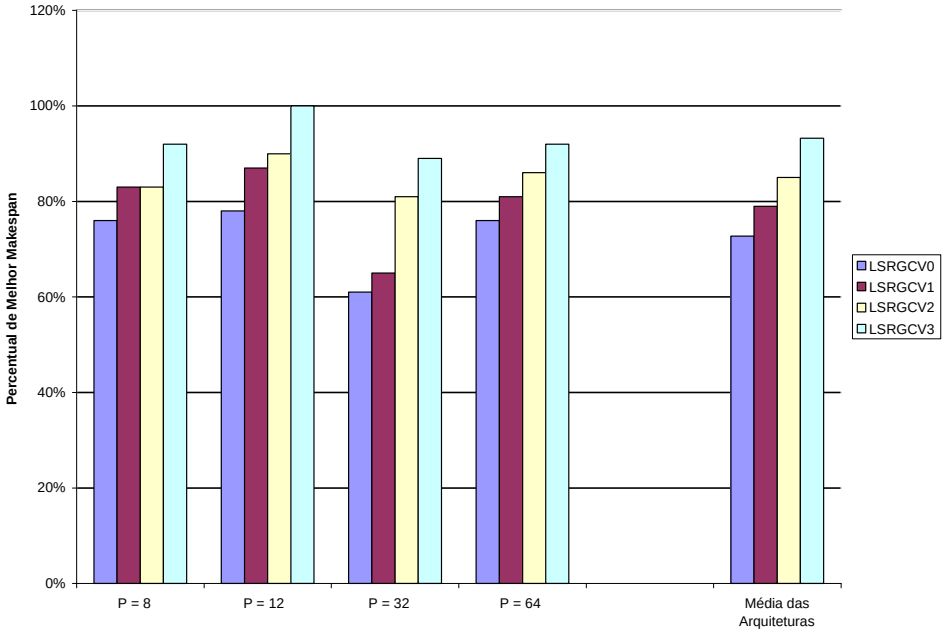

Figura 6.59: Comparação por Arquitetura Heterogênea - Grafos KPSG

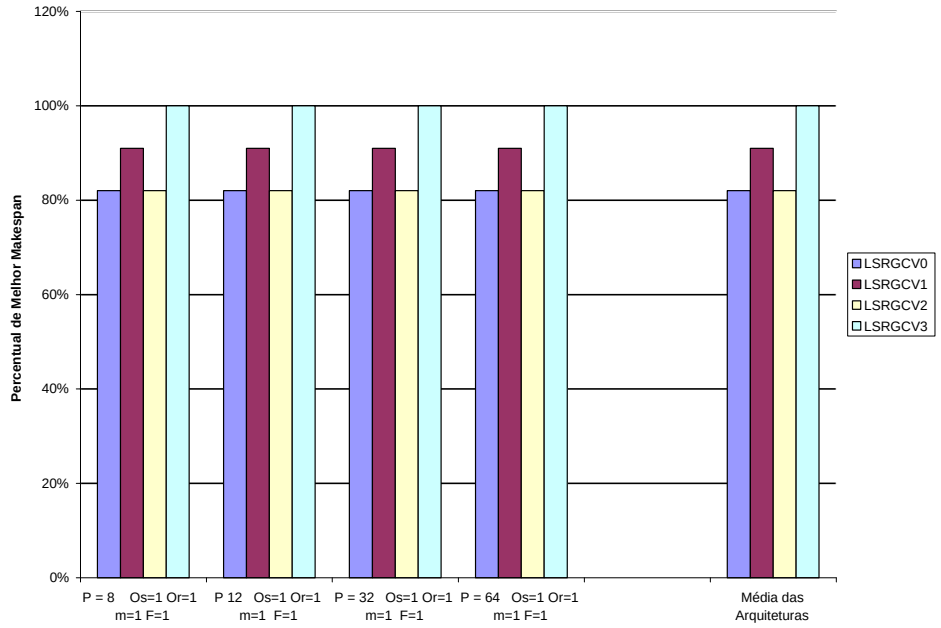

Figura 6.60: Comparação por Arquitetura Homogênea - Grafos KPSG

da escalabilidade do sistema e no comportamento das versões em ambientes de alta latência. Considerando todos os 65 grafos que compõem este conjunto foi possível observar que nos experimentos realizados utilizando 8 processadores, em média, a LSRGCV3 atingiu o melhor *makespan* em 80% dos casos com qualidade de 1,010;

a LSRGCV2 em 69% com qualidade de 1,017; e as versões LSRGCV0 e LSRGCV1, com valores próximos, atingiram 42% com qualidade de 1,057 e 43% com qualidade de 1,065, respectivamente. Os resultados médios estão apresentados na Figura 6.61.

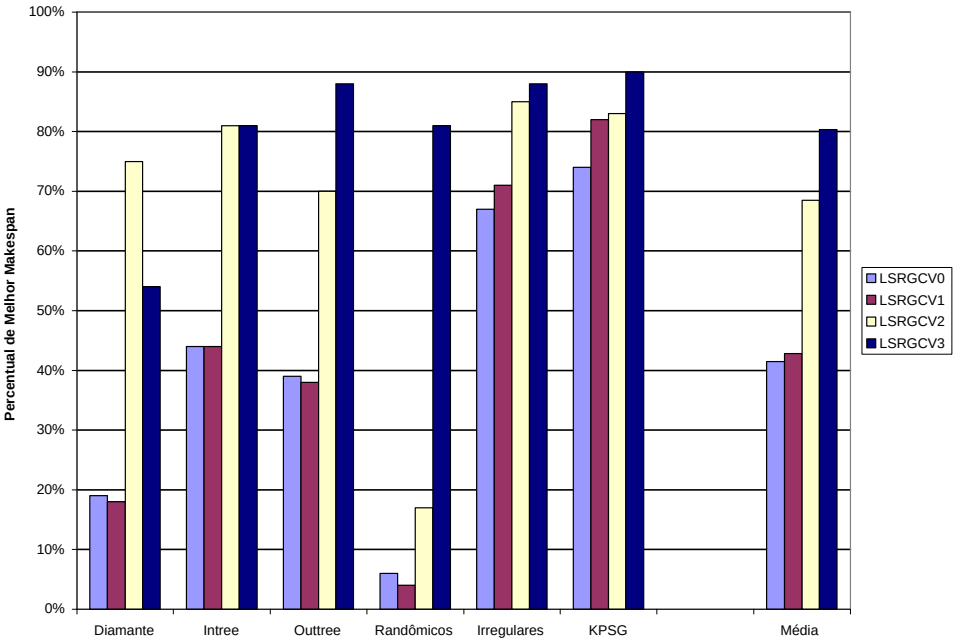

Figura 6.61: Comparação dos Percentuais com P=8 - Todos os Grafos

Os resultados médios observados nos experimentos usando 12 processadores estão ilustrados na Figura 6.62. Pode ser verificado que o aumento de poucos processadores surtiu quase nenhuma influência nos percentuais de cada uma das versões. Os valores foram praticamente os mesmos. A LSRGCV3 manteve a liderança atingindo o melhor *makespan* em 80% dos casos; entretanto melhorou a qualidade dos resultados apresentando menor degradação: 1,005. A LRGCV2 continua em segunda colocação com 68% dos casos e, também, com melhor qualidade de resultados: 1,013. A versão LSRGCV1 atingiu 42% com qualidade de 1,047; e a versão LSRGCV0 atingiu 41% com 1,044 de qualidade.

As médias dos experimentos realizados utilizando 32 processadores estão ilustrados na Figura 6.63. A LSRGCV3 continua na liderança dos melhores *makespan* em 76%, sofrendo uma pequena queda de 4 pontos percentuais, em relação ao percentual observado na arquitetura de 12 processadores. Da mesma forma, a qualidade foi ligeiramente inferior: 1,008. A LSRGCV2 sofreu um aumento de 5 pontos percen-

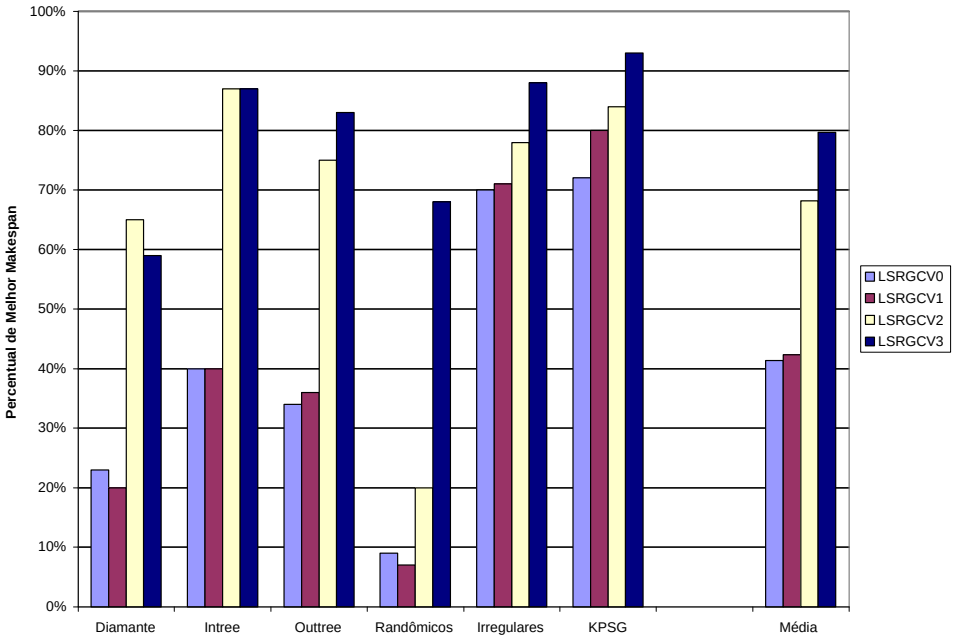

Figura 6.62: Comparação dos Percentuais com P=12 - Todos os Grafos

tuais, atingindo o melhor *makespan* em 73% dos casos, com qualidade praticamente a mesma: 1,014. A razão desse aumento no percentual da LSRGCV2 foi devido ao número de vitórias que conseguiu na classe de grafos Diamantes e Outtree. Em ambas as classes, as tarefas foram distribuídas pelos processadores disponíveis de tal forma que a estratégia do envio das mensagens após cada sobrecarga e a coleta de lixo instântanea surtiu maiores benefícios para a redução do *makespan*. Ambas as versões LSRGCV1 e LSRGCV0 atingiram 41% dos casos, com qualidade de 1,082 e 1,069.

Os valores médios dos experimentos realizados com 64 processadores estão ilustrados na Figura 6.64. O aumento de processadores permitiu um ligeiro aumento nos percentuais de todas as quatro versões. Na liderança continuam a LSRGCV3 e a LSRGCV2 com iguais 77% dos casos, a qualidade dos resultados da LSRGCV3 foi melhor (1,013) que a da LSRGCV2 (1,014). A LSRGCV0 atingiu 57% com qualidade de 1,038 e a LSRGCV1 atingiu 50% com qualidade de 1,055.

A última ilustração deste capítulo, Figura 6.65, apresenta as médias atingidas por cada versão agrupadas pela arquitetura utilizada na respectiva instância de testes e a média geral obtida. Pode ser observado que a versão LSRGCV3 é, em todas as

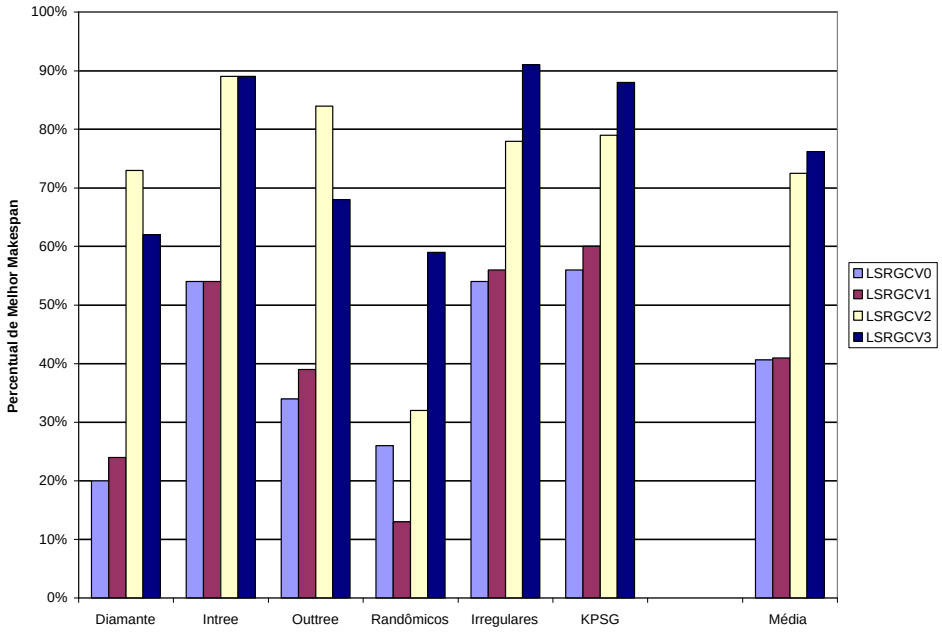

Figura 6.63: Comparação dos Percentuais com P=32 - Todos os Grafos

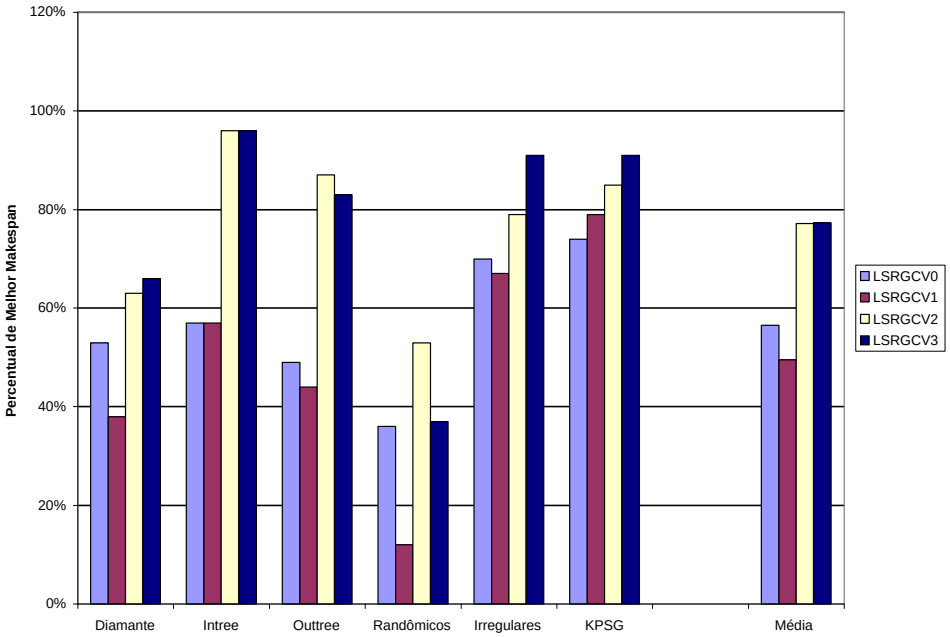

Figura 6.64: Comparação dos Percentuais com P=64 - Todos os Grafos

quatro arquiteturas, a que atinge melhores percentuais de sucesso. Nas arquiteturas de 8 e 12 processadores mantém o percentual de 80%, com o aumento do número de processadores para 32, sofre um ligeiro decréscimo para 76%. Isto foi devido, aos decréscimos nos percentuais verificados para essa versão nas classes de grafos Outtree

e Randômicos. Na arquitetura de 64 processadores ocorre um ligeiro aumento, para 77% indicando que o desempenho da LSRGCV3 correspondeu favoravelmente a escalabilidade do sistema. Com base nos resultados apresentados é possível enfatizar que as melhores estratégias para tratar as sobrecargas num processo de escalonamento foram as apresentadas pelas versões LSRGCCV2 e LSRGCV3 que propõem realizar a coleta dos espaços não utilizados (dentro das reservas R) a cada passo do escalonamento. A versão LSRGCV3 apresentou, em média, resultados superiores aos da LSRGCV2. Indicando que o efeito inicial negativo de atrasar o envio das mensagens (uma vez que somente podiam ser transmitidas ao final das reservas R) não surtiu prejuízos ao escalonamento final. Muito pelo contrário, contribuiu sobremaneira para a redução do *makespan*. Uma vez que as ordenações das sobrecargas baseadas nas características do escalonamento permitiram priorizar tarefas críticas e relevantes para a minimização do *makespan*.

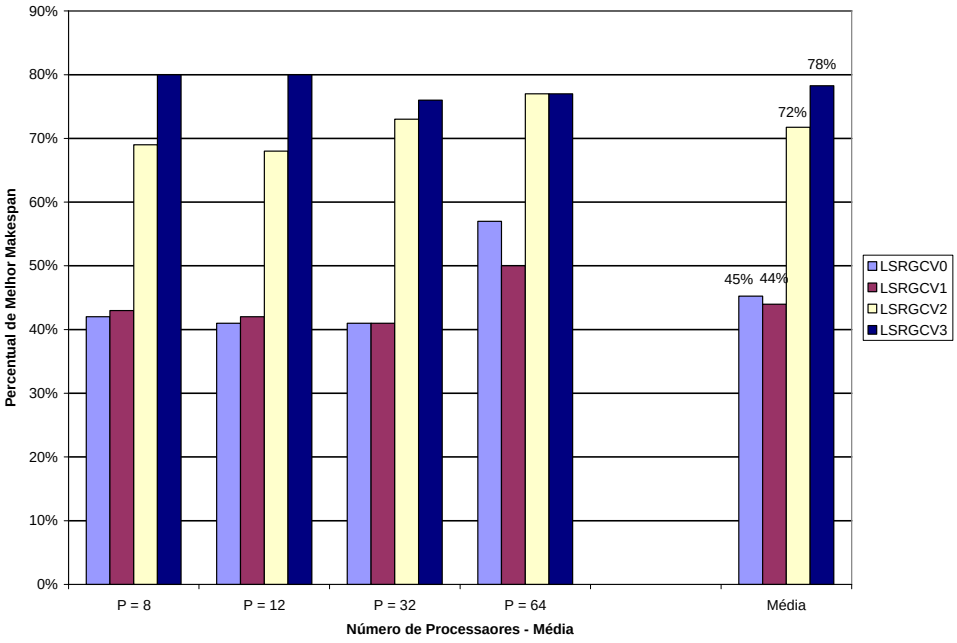

Figura 6.65: Comparação pela Escalabilidade - Todos os Grafos

6.9 Resumo

O objetivo principal deste capítulo foi avaliar as quatro estratégias propostas para minimizar os efeitos das sobrecargas no *makespan*. As quatro estratégias foram implementadas em heurísticas de *List Scheduling* com a prioridade, para ordenação da lista de tarefas livres, usando o atributo *nível dinâmico*. Além disso, dois critérios de desempate foram previstos, o primeiro usa o atributo de *conível dinâmico* e o segundo usa o atributo de *caminho crítico dinâmico*. As quatro arquiteturas heterogêneas utilizadas apresentam, respectivamente, 8, 12, 32 e 64 processadores. Os experimentos foram divididos pelas seis classes de grafos existentes no conjunto disponibilizado. Os tipos de experimentos realizados avaliaram distintos valores para os parâmetros O_s e O_r; e para (m, F) permitindo criar ambientes de distintas granulosidades. Na classificação final a LSRGCV3 foi a primeira colocada atingindo os melhores *makespan* em 78% dos casos, e com 1,009 de qualidade nos resultados; em segundo lugar ficou a LSRGCV2 com 72% dos casos e com qualidade de 1,014, em terceiro e quarto lugar, com valores quase iguais, ficaram as versões LSRGCV0 com 45% e qualidade 1,052; e a LSRGCV1 com 44% e qualidade 1,062. Além dessa classificação, foram verificados os percentuais de cada versão em relação a classificação pelo ambientes de granulosidade fina. Tais ambientes são bem mais relevantes a este trabalho, que o ambiente de granulosidade grossa. Isto decorre da característica de sistemas distribuídos, semelhantes às grades computacionais, onde existe uma tendência a alta latência devido a interconexão de distintos domínios. O que se verificou, na média dos valores, é que a LSRGCV3 atingiu o melhor *makespan* em ambientes de granulosidade fina em 76% dos casos. A seguir com valor bem próximo está a LSRGCV2 com 75% dos casos. As versões LSRGCV1 e LSRGCV0 com valores próximos, atingiram 47% e 48%, respectivamente. No próximo capítulo são apresentadas as conclusões e os trabalhos futuros decorrentes desta dissertação.

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho apresentou um estudo sobre o Problema de Escalonamento Estático de Tarefas em ambientes distribuídos com características semelhantes às grades computacionas. Nesses ambientes devido ao comportamento, tipicamente, instável dos recursos é necessário que a designação das tarefas aos mesmos seja realizada por um escalonador dinâmico. Uma tática para aliviar a carga de trabalho desse escalonador é utilizar um escalonador híbrido composto por um estático e outro dinâmico. O estático realiza um pré-escalonamento das tarefas e o dinâmico, com base nessas informações mais as referentes ao sistema computacional, realiza os ajustes finais necessários, durante a execução da aplicação. O objetivo principal desta dissertação é desenvolver o escalonador estático para ser incorporado ao *Framework* do escalonador híbrido do Projeto *Easygrid*. A meta do *Framework* é a execução eficiente de aplicações paralelas em ambientes de grades computacionais. Assim, o escalonador híbrido é projetado, especificamente, para cada aplicação paralela e deve apresentar baixa complexidade. Por isso, a técnica utilizada na heurística proposta foi a de *list scheduling* que comparada à outras existentes é a de mais baixa complexidade. Além disso, por ser facilmente adaptável a ambientes com um número de processadores limitados e heterogêneos.

O modelo de computação adotado abordou uma classe de aplicações que podem

ser representadas por grafos acíclicos direcionados (GAD) e um modelo de comunicação denominado *LogP*. Este modelo é considerado mais adequado e realístico que o modelo de Latência, para representar ambientes de *clusters* e *Grids* computacionais. O Modelo *LogP* adota parâmetros, em adição a Latência, que permitem o cálculo preciso das comunicações entre os processadores do sistema distribuído. Tais parâmetros, entre outros, são conhecidos como as sobrecargas da comunicação, para transmissão e recepção de dados, impostas aos processadores. A investigação realizada, por este trabalho, utilizou estratégias que visaram minimizar os efeitos adversos que essas sobrecargas causam no *makespan* de aplicações paralelas. A abordagem desenvolvida empregou numa heurística *list scheduling* distintas estratégias para tratamento das sobrecargas de comunicação. Assim, a heurística proposta *LSRGC* foi apresentada em três novas versões e em uma versão com uma estratégia existente na literatura. A métrica considerada na avaliação do melhor desempenho foi o *makespan* do escalonamento.

Duas fases distintas de testes foram realizadas em ambientes de processadores limitados e heterogêneos. Na primeira fase os experimentos foram efetuados considerando o modelo de latência e na segunda fase considerando o modelo *LogP*. Os primeiros testes objetivaram avaliar distintas prioridades para a ordenação da lista de tarefas livres do algoritmo *List Scheduling*. As prioridades avaliadas foram oito: *Nível*, *Conível*, *Caminho Crítico* e *ALAP*; calculadas de duas formas distintas, estática e dinâmica. A forma estática refere-se ao cálculo das prioridades antes do início da execução da primeira tarefa do grafo. Nesta forma uma vez construída a lista de tarefas livres a ordenação não pode mais ser alterada. A forma dinâmica permite que a cada passo do escalonamento as prioridades sejam recalculadas e, assim, as tarefas da lista de livres podem ser reordenadas; refletindo a real importância das tarefas no momento do escalonamento. Outro aspecto observado nessa fase de experimentos foi verificar se ocorreria alguma melhoria no *makespan* com a adoção de critérios de desempate. Foram incorporadas duas etapas de desempate com distintos critérios, primeiro desempate e segundo desempate, formando duplas de critérios para serem aplicadas em conjunto com a prioridade principal de ordenação das tarefas livres. Dentre as várias possibilidades de escolhas de duplas de desem-

pate, três foram escolhidas: *Nível Dinâmico e Caminho Crítico Dinâmico*, *Conível Dinâmico e Caminho Crítico Dinâmico* e *Conível e ALAP*. Os experimentos que avaliaram a forma de cálculo estática e a forma de cálculo dinâmica das prioridades, resultaram numa grande maioria de valores de *makespan* coincidentes. Entretanto, a média final obtida, considerando todos os 65 grafos avaliados, mostrou que o grupo de prioridades calculadas de forma dinâmica atingiram um percentual de casos com melhor *makespan* ligeiramente superior que o percentual atingido pelas prioridades calculadas de forma estática.

Os experimentos realizados para avaliar o sucesso da heurística de *list scheduling* com a utilização de uma prioridade principal e uma dupla de critérios de desempate, consistiram de cerca de 8320 testes, conforme os resultados apresentados no capítulo 5. Na média final, a prioridade principal que se destacou foi a de *Nível Dinâmico* com critérios de primeiro desempate de *Nível Dinâmico* e de segundo desempate *Caminho Crítico Dinâmico*.

Nova etapa de testes foi realizada em ambientes heterogêneos sob o modelo *LogP*. Nas quatro versões da heurística LSRGC foi mantida a prioridade principal de *Nível Dinâmico*. O critério de primeiro desempate utilizado foi de *Conível Dinâmico* e de segundo desempate foi de *Caminho Crítico Dinâmico*. Os objetivos são três: Avaliar qual é a melhor estratégia, dentre as apresentadas para minimizar os efeitos adversos que as sobrecargas de comunicações causam ao *makespan*; avaliar se a coleta de espaços não utilizados dentro das reservas R deve ser realizada ao final ou durante o escalonamento; e avaliar se a ordenação das sobrecargas deve ser baseada nas características do grafo ou nas de escalonamento. Cerca de 15600 testes foram executados, utilizando quatro arquiteturas compostas por, respectivamente, 8; 12; 32; e 64 processadores. Os experimentos foram agrupados pela classe do grafo e objetivaram avaliar, para distintos pares de parâmetros (O_s, O_r, m e F), as estratégias propostas pelas novas heurísticas: LSRGCV1, LSRGCV2 e LSRGCV3 e pela versão LSRGCV0 que implementa um estratégia proposta na literatura utilizada na heurística ETFRGC [40]. As quatro estratégias adotam distintas políticas para tratar as sobrecargas de comunicação de forma a amenizar os efeitos adversos que causam no *makespan* das aplicações. Em cada uma das arquiteturas de processadores utilizadas

distintos valores foram atribuídos as sobrecargas de envio O_s e de recebimento O_r, previstas no Modelo *LogP*. Além disso, foram utilizados dois fatores multiplicativos, um fator m para a computação das tarefas, e outro fator F para a comunicação entre as tarefas, o que permitiu simular ambientes de distintas granulosidades.

As comparações dos resultados apresentados por cada uma das políticas adotadas em cada versão, foram baseados nos percentuais de melhor *makespan* considerando os resultado produzidos pelas quatro versões. Além disso, foram realizados experimentos para avaliar a qualidade dos resultados fornecidos por cada política. Quanto menor for o valor da degradação em relação ao melhor *makespan* gerado, melhor é a qualidade do resultado produzido. As médias finais dos resultados, observadas em cada classe de grafos foram as seguintes: Na classe Diamante a versão LSRGCV2 foi a que atingiu melhores *makespan*. Na classe Intree observou-se o empate entre as duas versões de melhores resultados, a LSRGCV2 e a LSRGCV3. Em relação aos grafos Outtree, a política proposta pela LSRGCV3 surtiu maiores benefícios para a diminuição do *makespan*. Nos grafos Randômicos, novamente se destacou sobre as demais versões, a política adotada pela LSRGCV3. Na classe de grafos Irregulares a versão LSRGCV3 foi a que produziu melhores *makespans*. Com respeito aos grafos KPSG, a LSRGCV3 foi novamente a de maior sucesso. Na classificação final a LSRGCV3 foi a que mais minimizou os efeitos adversos das sobrecargas no *makespan* das aplicações atingindo 78% de sucesso nos escalonamentos com qualidade de 1,009; a LSRGCV2 atingiu 72% com qualidade de resultados de 1,014; a LSRGCV0 atingiu o melhor *makespan* em 45% dos escalonamentos com qualidade de 1,052; a LSRGCV1, apresentou percentual ligeiramente inferior ao da LSRGCV0, 44% com qualidade de 1,062. Além disso, a LSRGCV3 foi a de melhores *makespans* em ambientes de alta latência, atingindo 76%.

A conclusão desta investigação, com base nos 23920 experimentos realizados, indica que a estratégia proposta pela versão LSRGCV3 foi a que mais contribuiu para minimizar os efeitos que as sobrecargas de comunicação causam ao *makespan* das aplicações paralelas. Uma vez que sua estratégia de realizar a coleta de lixo durante o escalonamento contribuiu sobremaneira para se atingir resultados de *makespan* mais favoráveis. Além disso, o atraso inicial das mensagens ao se impôr a restrição

de somente permitir os envios ao término das reservas R, não causou prejuízos ao escalonamento final, e sim contribuiu sobremaneira para a sua minimização. Isto porque permitiu que as ordenações das sobrecargas fossem realizadas com base nas características do escalonamento, ao invés de as basear nas características do grafo. Apesar da LSRGCV1, também, adotar esta política de atraso com ordenação, o insucesso da versão LSRGCV1 foi realizar a coleta de lixo ao final do escalonamento. Sendo assim, a LSRGCV3 é potencialmente indicada para realizar o escalonameto estático em ambientes computacionais com transmissão de dados por troca de mensagens, como as grades computacionais. Visto que apresentou maior percentual de melhores *makespans* que as outras três versões quando se avaliou a escalabilidade do sistema, com qualidade média de no máximo 1,020. Além disso, considerando ambientes de alta latência foi que atingiu o melhor *makespan* em o maior número de casos. Essas característica, escalabilidade alta latência são condições intrínsecas às plataformas se sistemas distribuídos como as grades computacionais.

Os trabalhos futuros que podem ser desenvolvidos a partir deste são:

a) Especificar uma abordagem para utilizar a técnica de *inserção de tarefas*, com o modelo *LogP*, a fim de utilizar espaços livres existentes entre tarefas já escalonadas;

b) Desenvolver novas heurísticas para um modelo mais realístico que o *LogP* capaz de suportar recursos heterogêneos, como o *HLogP* [52] voltado para ambientes de grades;

c) Investigar novas alternativas para minimizar os efeitos adversos das sobrecargas, decorrentes das comunicações, em sistemas distribuídos. Como, por exemplo, considerar os tempos de inícios das tarefas variáveis e, assim, inserir (cajo seja necessária) a sobrecarga de envio, a cada iteração do escalonamento.

Apêndice A

Topologia de Grafos

Neste apêndice são apresentadas algumas estruturas dos Grafos Acíclicos Direcionados (GAD) utilizados na avaliação dos resultados obtidos pelas heurísticas analisadas. Dois *Benchmark* de grafos, disponíveis na literatura, foram utilisados, os grafos do Projeto *EasyGrid* [8] e os grafos *KPSG -Kwok Peer Set Graphs* disponibilizados em [46]. As topologias dos grafos são variadas, consistindo de ávores binárias invertidas e diretas, diamantes, formas irregulares como a gaussiana e o código molecular e randômicas . A finalidade é permitir verificar como são classificadas as topologias das aplicações numéricas e científicas utilizadas nos experimentos.

A.1 Grafos Acíclicos Direcionados

O *Benchmark* do projeto *EasyGrid* é composto por dois conjuntos de grafos: Os regulares unitários, nos quais todas as tarefas e arcos tem pesos unitários e o conjunto dos irregulares não unitários, cujas tarefas e arcos possuem pesos arbitrários. Os grafos regulares podem ser divididos em quatro classes: Diamantes, Intree, OutTree e Randômicos; o outro conjunto de grafos não unitários é composto dos irregulares. Ao conjunto não unitário é adicionado o *Benchmark KPSG*, também, composto por grafos irregulares não unitários. A terminologia representativa do nome do GAD

diz respeito a sua classificação em árvores diretas Outtree *Out*, e indiretas Intree *In*, Diamantes *Di*, Randômicos *Ran*, irregulares *Irr* e KPSG *KPSG* e ao número total de tarefas.

A.1.1 Classe de Grafos Diamante

Semelhantes a uma estrutura de distribuição em forma de malha encontra-se esta categoria de grafos muito utilizados nas aplicações numéricas e científicas. Tais grafos podem representar uma subcomputação na *Decomposição LU* [66], multiplicação de matrizes e *Systolic Arrays* [54]. Nos grafos diamantes todas as tarefas são relacionadas por predecessores e sucessores comuns, não necessariamente imediatos. A Figura A.1 apresenta uma topologia de grafo diamante 4X4. São compostos por 11 grafos cada um referenciado pelas iniciais *Di* seguido do número de tarefas que apresentam: *Di*9, *Di*16, *Di*25, *Di*36, *Di*64, *Di*100, *Di*144, *Di*225, *Di*256, *Di*400 e *Di*1024.

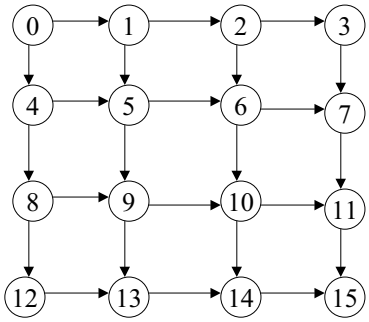

Figura A.1: Grafo Diamante 4X4

A.1.2 Classe de Grafos Intree - Árvores Invertidas

Esta classe de Grafos é conhecida como árvores binárias completas invertidas, cada tarefa tem dois predecessores imediatos independentes (com exceção das tarefas origens) e apenas um sucessor imediato (com exceção da tarefa de saída). Algoritmos que utilizam uma representação onde os dados partem das folhas para a raiz, indicando redução nas operações, tais como soma em paralelo e conquista de dados são bons exemplos da topologia de árvore binária invertida *in-tree*. A Figura A.2

ilustra uma árvore. São compostos por 7 grafos, cada um referenciado pelas iniciais In seguido do número de tarefas que apresentam: $In3$, $In15$, $In31$, $In63$, $In127$, $In255$ e $In511$ tarefas

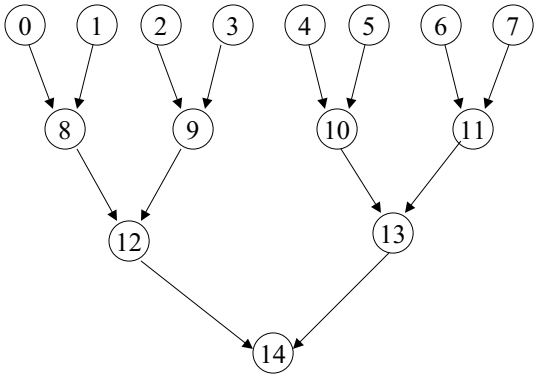

Figura A.2: Árvore do tipo *in-tree* - genérica

A.1.3 Classe de Grafos Outtree - Árvores

A estrutura de dados denominada árvore é um dos principais paradigmas da computação paralela. Algoritmos de difusão, soma de prefixos, soma paralela, divisão e conquista e *pointer jumping* são alguns dos exemplos que empregam esta topologia de grafo para modelagem do problema. A estrutura utilizada na difusão, divisão e particionamento de dados se ajusta a uma topologia de árvore onde os dados partem da raiz até as folhas (nós sem sucessores). Essa topologia é conhecida com o nome de *out-tree*. A Figuras A.3 ilustra uma árvore regular *out-tree* de 15 tarefas ou nós. A característica principal desta topologia é que cada tarefa tem apenas um predecessor imediato (com exceção da tarefa origem ou raiz) e dois sucessores imediatos (com exceção das tarefas de saída) São compostos por 8 grafos, cada um referenciado pelas iniciais Out seguido do número de tarefas que apresentam: $Out3$, $Out7$, $Out15$, $Out31$, $Out63$, $Out127$, $Out255$ e $Out511$.

A.1.4 Classe de Grafos Randômicos

Os grafos desta classe são gerados aleatoriamente com a finalidade de representar qualquer estrutura irregular. O menor deles apresenta 80 tarefas e por isso não será

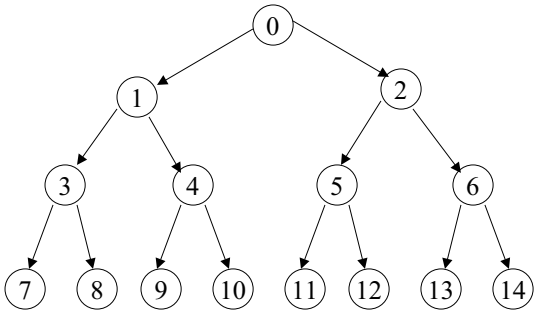

Figura A.3: Árvore Binária do tipo *out-tree* genérica

ilustrado. São compostos por 21 grafos, cada um referenciado pelas iniciais *Ran* seguido pelo número de tarefas que apresentam: *Ran*80, *Ran*98, *Ran*108, *Ran*124, *Ran*135, *Ran*140, *Ran*152, *Ran*153, *Ran*154, *Ran*170, *Ran*186, *Ran*223, *Ran*234, *Ran*256, *Ran*286, *Ran*298, *Ran*310, *Ran*357, *Ran*364, *Ran*510 e *Ran*546.

A.1.5 Classe de Grafos Irregulares

Esta classe é composta por grafos, como o próprio nome indica, de estrutura irregular. A característica principal dos grafos é apresentarem tarefas e arestas com pesos de grande valor. São amplamente utilizadas para representar problemas matemáticos do mundo real. Dentre os quais podemos citar: Algoritmo da Eliminação Gaussiana [76, 21], Transformada Rápida de Fourrier [19, 16], e aplicações da física molecular [42]. A Figura A.4 representa o grafo da Eliminação Gaussiana. Esta classe é composta por 7 grafos, cada um referenciado pelas iniciais *Irr* seguido pelo número de tarefas que apresentam. Como três grafos são compostos, cada um, por 7 tarefas para diferenciá-los, foram acrescentados índices (a, b e c) ao lado de suas referências . Os grafos são: *Irr*7*a*, *Irr*7*b*, *Irr*7*c*, *Irr*10, *Irr*13, *Irr*18 e *Irr*41.

A Figura A.5 ilustra a representação de uma aplicação da Física molecular, apresentada em [42].

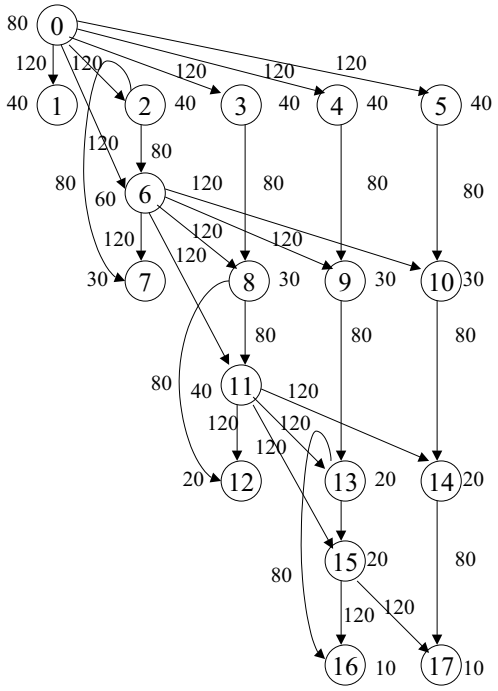

Figura A.4: Eliminação Gaussiana - Irr18

A.1.6 Classe de Grafos KPSG *Kwok Peer Set Graphs*

Esta classe é composta por grafos de topologia variada com poucas tarefas de pesos medianos. A característica principal dos GADs é que as arestas apresentam pesos com valores, em média, superiores aos pesos das tarefas; o que atribui ao conjunto ser considerado de granulosidade fina. As Figuras de A.6 ao A.16 ilustram os 11 grafos que compõem este *Benchmark*. Os grafos são referenciados pelas iniciais $KPSG$ seguidas de um número inteiro, representativo da ordem seqüencial dos grafos.

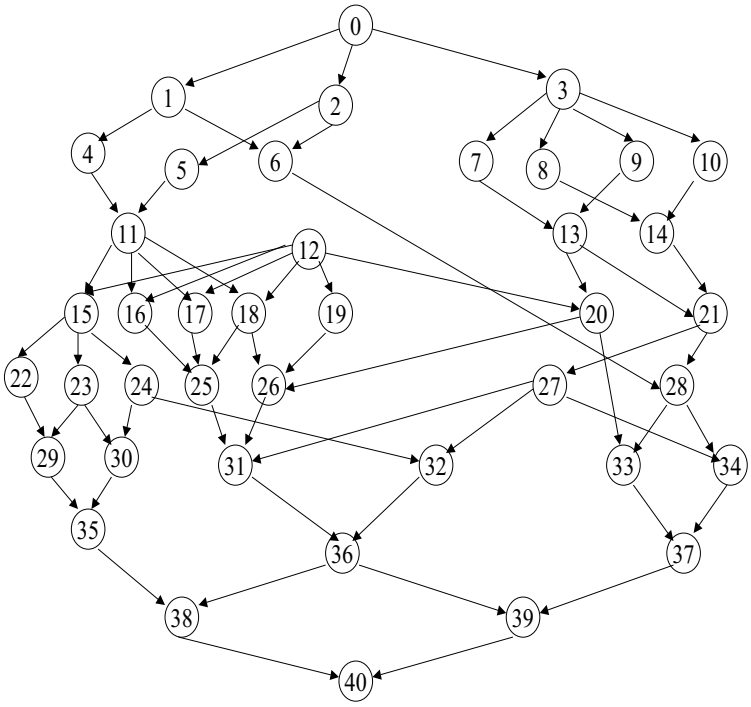

Figura A.5: Código Molecular Dinâmico - Irr41

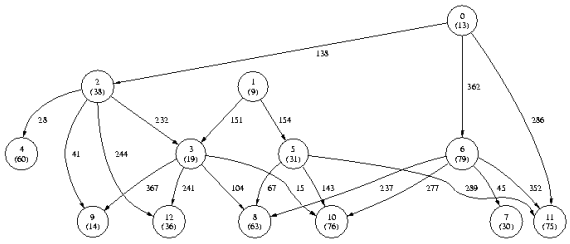

Figura A.6: KPSG1 - Ahmad and Kwok

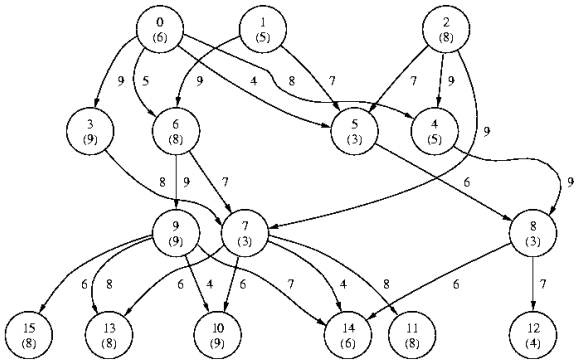

Figura A.7: KPSG2 - Al-Maasarani

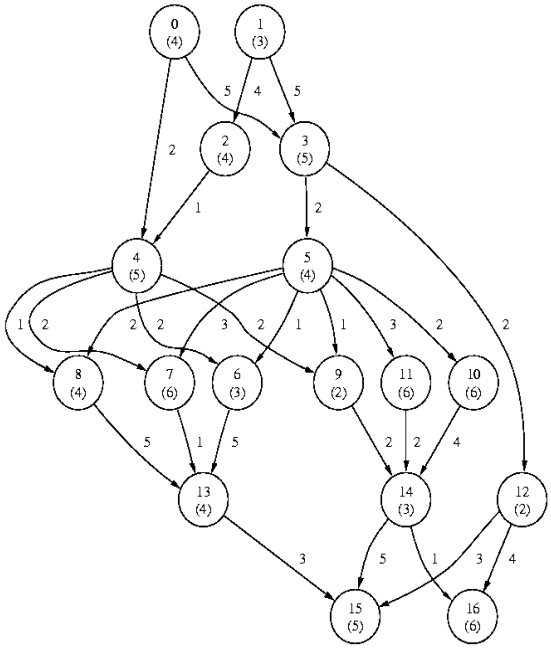

Figura A.8: KPSG3 - Al-Mouhamed

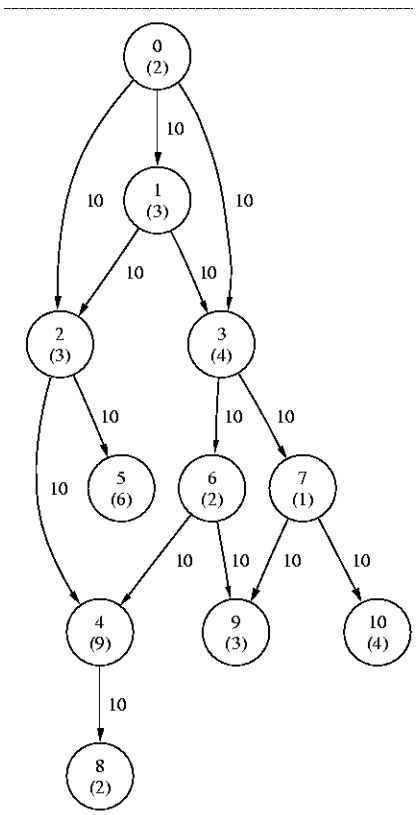

Figura A.9: KPSG4 - Shirazi *et alli*

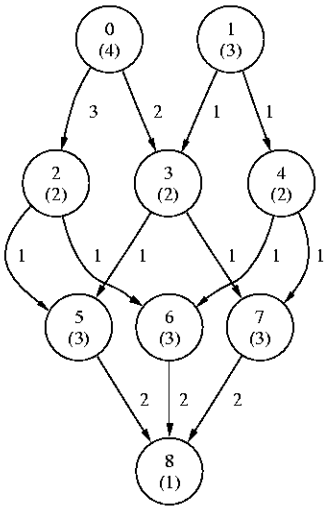

Figura A.10: KPSG5 - Colin and Chretienne

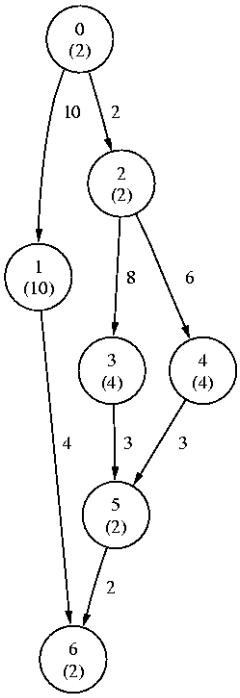

Figura A.11: KPSG6 - Gerasoulis and Yang

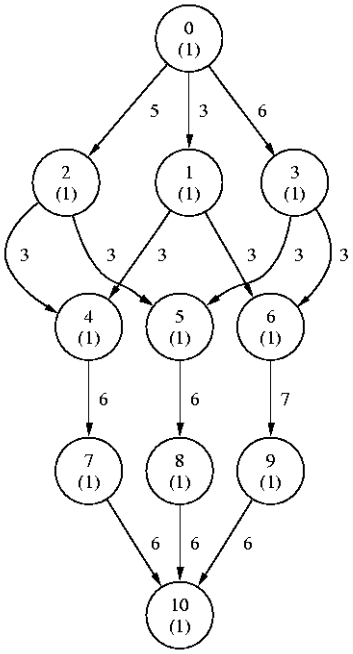

Figura A.12: KPSG7 - Kruatrachue and Lewis

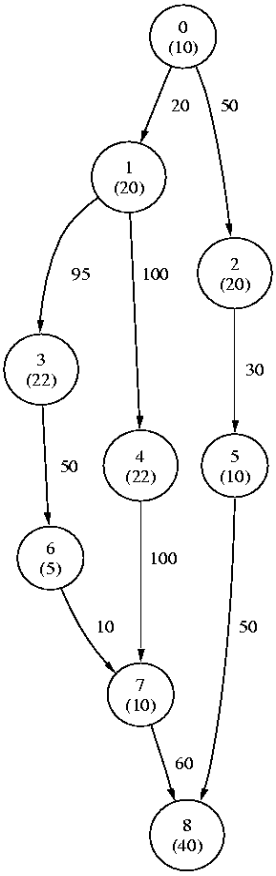

Figura A.13: KPSG8 - McCreary and Gill

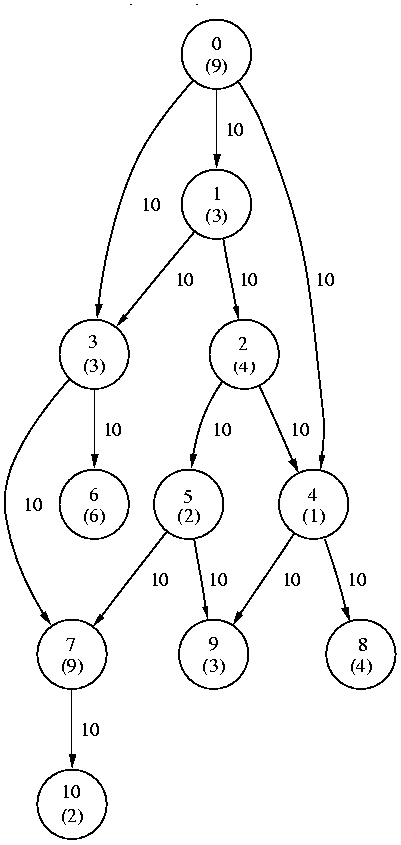

Figura A.14: KPSG9 - Chung and Ranka

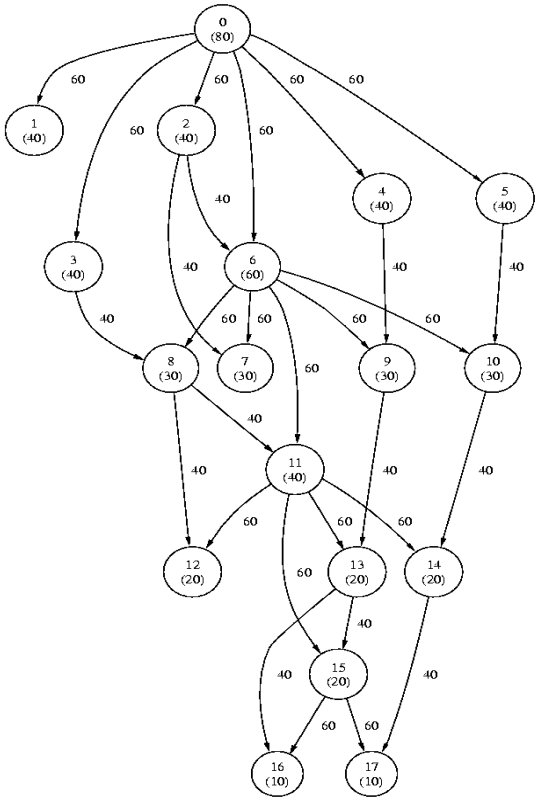

Figura A.15: KPSG10 - Wu and Gajski

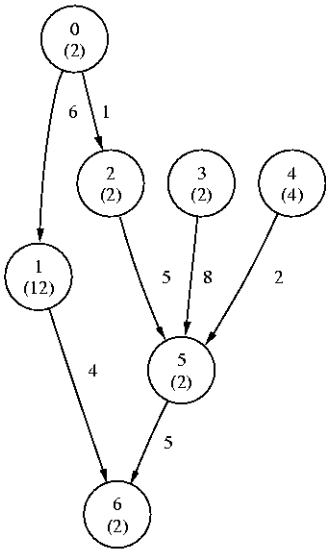

Figura A.16: KPSG11 - Yang and Gerasoulis

Referências Bibliográficas

[1] Adam T., Chandy K. e Dickson J. A comparison of list schedulers for parallel processing systems. *Communication ACM 17*(2): 685–690, 1974.

[2] Ahmad I. e Kwok Y-K. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Systems 9*(9): 872–892, 1998.

[3] Akl S. G. *Parallel Computations: Models and Methods.* Prentice Hall, 1997.

[4] Alexandrov A., Ionescu M., Schauser K. e Scheiman C. LogGP: Incorporating long messages into the LogP model - One step closer towards a realistic model for parallel computation. In *Proceedings of the 7th Annual Symposium on Parallel Algorithms and Architectures (SPAA'95)*, 1995.

[5] Anderson T., Culler D. e Patterson D. A case for NOW (Networks of Workstations). *IEEE Micro 15*(1): 23–39, 1995.

[6] Beaumont O., Legrand A. e Robert Y. Static scheduling strategies for heterogeneous systems. *Computing and Informatics 21*: 413–430, 2002.

[7] Boeres C. Uma introdução ao escalonamento em multicomputadores. Apostila do Curso de Algoritmos Paralelos. Instituto de Computação, Universidade Federal Fluminense, 1998.

[8] Boeres C. *Versatile Communication Cost Modelling for Multicomputer Task Scheduling Heuristics.* Tese de Doutorado, Department of Computer Science, University of Edinburgh, 1997.

[9] Boeres C., Lima A. e Rebello V. E. F. Hybrid task scheduling: Integrating static and dynamic heuristics. *In the Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2003.

[10] Boeres C. e Rebello V. E. F. LogP cluster-based scheduling: Theory and practice. In *The Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002)*, IEEE Computer Society Press, 2002.

[11] Boeres C., Rebello V. E. F. e Skillicorn D. Static scheduling using task replication for LogP and BSP models. In *The Proceedings of the 4th International Euro-Par Conference on Parallel Processing* (UK), D. Pritchard and J. Reeve, Eds., LNCS 1470: 337–346, 1998.

[12] Buyya R. *High Performance Cluster Computing: Architectures and Systems.* Prentice Hall, 1999.

[13] Casavant T. e Kuhl J. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engeneering 14*(2): 141–154, 1988.

[14] Chochia G., Boeres C., Norman M. e Thanisch P. Analysis of multicomputer schedules in a cost and latency model of communication. In *Proceedings of the 3rd Workshop on Abstract Machine Models for Parallel and Distributed Computing* (Leeds, UK.), IOS press, 1996.

[15] Choe T. e Park C. A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. *Proceedings of the International Conference on Parallel Processing Workshops*, 531–536, 2002.

[16] Chung Y. e Ranka S. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. *Proc. Supercomputing 1*(7): 512–521, 1992.

[17] Coffman E. Computer and job-shop scheduling theory. *John Wiley & Sons. New York*, 1976.

[18] Coffman E. e Graham R. Optional scheduling for two processor systems. *Acta Informática 1*, 200–213, 1972.

[19] Cormen T., Leiserson C. e Rivest R. *Introduction to Algorithms*. MIT Press, 1990.

[20] Corrêa R., Ferreira A. e Rebreyend P. Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems 10*(8): 825–837, 1999.

[21] Cosnard M., Marrakchi M., Robert Y. e Trystram D. Parallel gaussian elimination on an MIMD computer. *Parallel Computting 6*: 275–295, 1988.

[22] Culler D., Karp R., Patterson D., Sahay A., Schauser K., Santos E., Subramonian R. e Von Eicken T. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (San Diego, CA, USA), 1993.

[23] Dogan A. e Özgüner F. LDBS: A duplication based scheduling algorithm for heterogeneous computing systems. *Proceedings of the International Conference on Parallel Processing (ICPP 02)*, 2002.

[24] Ehrenfeucht A. e Rozenberg G. Theory of 2-structures, part i: Clans basic subclasses and morphisms. *Theoretical Computer Science 70*(3): 277–303,1990.

[25] Esteves, M. A. N. Rumo a uma heurística rápida para geração de escalonamentos eficientes em grids computacionais. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2003.

[26] Feo T. e Resende M. Greedy randomized adaptative search procedures. *Journal of Global Optimization 6*: 109–133, 1995.

[27] Fernandez E. e Bussel B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions Computer c-22*(8): 745–751, 1973.

[28] Filho J. V. Estratégia para escalonamento de tarefas em dois estágios para arquiteturas heterogêneas. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2004.

[29] Fishburn P. *Interval orders and interval graphs.* John Wiley & Sons. New York, 1975.

[30] Fortune S. and Wyllie J. Parallelism in random access machines. In *The Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, 114–118, 1978.

[31] Foster I. The Grid: A new infrastructure for 21st century science. *Physics today* 2002.

[32] Foster I. e Kesselman C. *The GRID: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999.

[33] Garey M. e Johnson D. *Computers and intractability: A Guide to the theory of NP-Completeness. W.H. Freeman*, 1979.

[34] Gerasoulis A. e Yang T. A comparison of clustering heuristics for scheduling Directed Acyclic Graphs on multiprocessors. *Journal of Parallel and Distributed Computing 16*: 276–291, 1992.

[35] Glover F. Tabu search - part i. *ORSA Journal on Computing 1*: 190–206, 1989.

[36] Glover F. Tabu search - part ii. *ORSA Journal on Computing 1*: 4–32, 1990.

[37] Hansen P. e Mladenović N. A tutorial on variable neighborhood search. Tech Report GERAD2003-46, Universidade de Montreal, 2003.

[38] Holland J.H. Adaptation in natural and artificial systems. Tech report, University of Michigan Press, 1975.

[39] Hwang J., Chow Y., Anger F. e Lee C. Scheduling precedence graphs in systems with interprocessor communication times. *SIAMJ Comp. 18*: 244–257, 1989.

[40] Kalinowski T., Kort I. e Trystram D. List scheduling of general task graphs under LogP. *Parallel Computing 26*(9): 1109–1128, 2000.

[41] Khan A., MacCreary, C. e Jones M. A comparison of multiprocessor scheduling heuristics. *Proceedings of International Conference on Parallel Processing 2*: 243–250,1994.

[42] Kim S. e Browne J. A general approach to mapping of parallel computation upon multiprocessor architectures. *In Proceedings of the IEEE International Conference on Parallel Processing 2*: 1–8, 1988.

[43] Kirkpatrick S., Gelatt C. e Vecchi M. P. Optimization by simulated annealing. *Science, 4598*(220): 671–680, 1983.

[44] Kruatrachue B. e Lewis T. Grain size determination for parallel processing. *IEEE Software*, 23–32, 1988.

[45] Kwok Y-K. e Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating tasks graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems 7*(5): 506–521, (1996).

[46] Kwok Y-K. e Ahmad, I. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing 59*(3): 381–422, 1999.

[47] Kwok Y-K. e Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys 31*, 4, 1999.

[48] Kwok Y-K., Ahmad I. e Gu J. Fast: A low-complexity algorithm for efficient scheduling of DAGs on parallel processors. *Proc. Int´l Conf. Parallel Processing (ICPP) 2*: 150–157, 1996.

[49] Lee C., Hwang J., Chow Y. e Anger F. Multiprocessor scheduling with interprocessor communication delays. *Operations Research Letters 7*: 141–147, 1988.

[50] Macey B. e Zomaya A. A performance evaluation of CP list scheduling heuristics for communication intensive task graphs. In *Proceedings of IPPS/SPDP*, 538–541, 1998.

[51] McCreary C., Thompson J., Gill H., Smith T. e Zhu Y. Partitioning and scheduling using graph decomposition. Technical Report, Department of Computer Science and Engineering, Auburn University, 1993.

[52] Mendes H. HLogP: Um modelo de escalonamento para a execução de aplicações MPI em grades computacionais. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 2004.

[53] Nascimento A. Aglomeração de tarefas em arquiteturas paralelas com memória distribuída. Dissertação de Mestrado, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brasil, 1999.

[54] Norman M., Chochia G., Thanisch P. e Issman E. Predicting the performance of the diamond DAG computation. Technical Report EPCC-TR-92-07, Edinburgh Parallel Computing Centre, 1992.

[55] Norman M. e Thanisch P. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys 25*(3): 263–302, 1993.

[56] Oh H. e Ha S. A static heuristic for heterogeneous processors. In *Euro-Par*(2): 573–577, 1996.

[57] Palis M., Liou J. e Wei D. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems 7*(1): 46–55, 1996.

[58] Papadimitriou C. e Yannakakis M. Scheduling interval-ordered tasks. *SIAMJ Comp. 8*: 405–409, 1979.

[59] Papadimitriou C. e Yannakakis M. Towards and archicteture independent analysis of parallel algorithms. *SIAMJ Comp. 19*: 322–328, 1990.

[60] Park G. Shirazi B., Marquis J. e Choo H. Decisive path scheduling: A new list scheduling method. *International Conference Parallel Processing*, 472–480, 1997.

[61] Radulescu A. e Van Gemund A. Low-cost task scheduling for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems 13*(6): 648–658, 2002.

[62] Rebello V. E. F. Página do Projeto EasyGrid. http://easygrid.ic.uff.br.

[63] Rebello V. E. F. e Boeres C. Easygrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience. John Wiley and Sons 16*(5): 425–432, 2004.

[64] Ridge D., Becker D., Merkey P. e Sterling T. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs. In *Proceedings, IEEE Aerospace*, 1997.

[65] Sarkar V. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.* MIT Press, 1989.

[66] Selim G. A. *Parallel Computations: Models and Methods.* Prentice Hall, 1997.

[67] Shirazi B., Hurson A. e Kavi K. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press, CA, USA, 1995.

[68] Sih G. e Lee E. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems 4*(2): 175–187, 1993.

[69] Sinnen O. e Souza L. A classification of graph theoretic models for parallel computing. Technical Report, RT/005/99, Instituto Superior Técnico - Universidade Técnica de Lisboa, 1999.

[70] Souza P. *Escalonamento de processos: uma contribuição para a covergência da área.* Tese de Doutorado, Departamento de Ciências de Computação e Estatística, Universidade de São Paulo, 2000.

[71] YANG T. E GERASOULIS A. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems 5*(9): 951–967, 1994.

[72] TAM A. E WANG C. Realistic communication model for parallel computing on cluster. In *The Proceedings of the 1st IEEE International Workshop on Cluster Computing*, IEEE Computer Society Press, 92–101, 1999.

[73] TOPCUOGLU H., HARIRI S. E WU M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems 13*(3): 260–274, 2002.

[74] ULLMAN J. NP-Complete scheduling problems. *Journal of computer and system science 10*: 384–393, 1975.

[75] VALIANT L. A bridging model for parallel computation. *Communication ACM 33*: 103–111, 1990.

[76] WU M. E GAJSKI D. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems 1*(3): 101–119, 1990.

[77] YANG T. E FU C. Heuristics algorithms for scheduling iterative task computations on distributed memory machines. *IEEE Transactions on Parallel and Distributed Systems 8*(6): 608–622, 1997.