UMA HEURÍSTICA PARA O ESCALONAMENTO ESTÁTICO DE TAREFAS EM AMBIENTES COMPUTACIONAIS HETEROGÊNEOS

Deolinda Fontes Cardoso
CASNAV – UFF
dcardoso@ic.uff.br

Eugene Francis Vinod Rebello
UFF – Universidade Federal Fluminense
Rua Passo da Pátria s/n – Boa Viagem - Niterói
vinod@ic.uff.br

Resumo

Atualmente diversas aplicações necessitam de poder computacional superior ao que um computador seqüencial pode fornecer para serem executadas em tempo aceitável. Face aos altos custos dos supercomputadores vetoriais, alternativas como *Clusters Computing* e mais recentemente os *Grid Computing,* se estabeleceram como formas de agregar poder computacional a custos acessíveis. Nessas plataformas o escalonamento eficiente de tarefas da aplicação paralela é o ponto crucial para se obter desempenho. Este artigo considera o Problema de Escalonamento de Tarefas (PET) em ambientes como as Grades Computacionais que são ambientes de alta latência, de recursos heterogêneos e de comportamento dinâmico. A abordagem escolhida propõe uma heurística da classe de *list scheduling* para um número limitado de processadores heterogêneos sob o modelo de comunicação *LogP*. A nova heurística *LSRGC* é uma extensão da metodologia proposta em [1], implementada em quatro versões. O objetivo é avaliar alternativas para minimizar os efeitos adversos que as sobrecargas decorrentes das comunicações causam ao ambiente. O parâmetro considerado nas comparações é o tempo total de execução paralelo da computação escalonada, conhecido como *makespan* do escalonamento, quanto menor for esse valor melhor será o desempenho da heurística.
Palavras-Chaves: Grades computacionais; Modelo *LogP*; e Escalonamento de Tarefas.

Abstract

Now several applications need higher performance to the that a sequential computer can supply for they be executed in acceptable time. Face at the high costs of the vectorial super computers, alternatives as Clusters Computing and more recently Grid Computing, they settled down as forms of joining power computing at accessible costs. In those platforms the efficient scheduling tasks of the parallel application is the crux to obtain performance. This article considers the Problem of Scheduling Tasks in environment as the Grid Computing that have high latency, heterogeneous resources and dynamic behavior. The chosen approach proposes a heuristics of the class of list scheduling for a limited number of heterogeneous processors under the communication LogP model. The new heuristics LSRGC is an extension of the methodology proposed in [1], implemented in four versions. The objective is to evaluate alternatives to minimize the adverse effects that the current overheads of the communications cause in this environment . The parameter considered in the comparisons is the parallel total time of execution, known as makespan of the scheduling, as minor goes that better value will be the performance of the heuristics.
Keywords: *Grid Computing; LogP model; and Scheduling Tasks.*

1. INTRODUÇÃO

Devido a crescente utilização do processamento de informações nas mais diversas áreas, um número cada vez maior de aplicações necessita de poder computacional superior ao que

um computador seqüencial pode fornecer. Por exemplo: Previsão do tempo, simulações físicas, computação gráfica, geometria fractal, criptografia e criptanálise. Tais aplicações se executadas em máquinas convencionais podem levar várias semanas ou até meses para executar e em alguns casos extremos nem executariam por falta de capacidade de memória**.** Por essa razão, 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. Assim, foram projetadas máquinas capazes de executar tarefas (de uma aplicação) 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.

Principalmente, com as melhorias nas tecnologias de redes de comunicação e aumento da largura de banda [2] aliadas a disponibilidade de computadores pessoais que dobram de velocidade a cada 18 meses, foi possível o desenvolvimento de sistemas computacionais paralelos e distribuídos como um caminho promissor na obtenção de maior desempenho. Surgiram assim, coleções de computadores tradicionais, como forma de agregar poder computacional. A este conceito de processamento paralelo denominamos de *Cluster Computing* [3]. Trata-se de um agrupamento ou coleção de máquinas PC tradicionais, em um mesmo domínio, conectadas por uma rede de comunicação de alta velocidade, dedicadas para compartilhar recursos e armazenamento de dados. Recentemente, é possível interconectar diversas coleções localizadas em domínios geograficamente distribuídos, interconectados por redes Internet, constituindo, assim, a tecnologia de *Grid Computing* [4]. Essa terminologia g*rid* 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 elétrica [4]. Sob esse aspecto, uma grade computacional parece ter o potencial latente para se tornar uma poderosa plataforma de computação paralela. Nessa plataforma, será possível a execução de uma vasta faixa de aplicações que demandam por grande poder computacional.

Grades computacionais são ambientes onde os recursos de comunicação e de computação são tipicamente compartilhados, distribuídos e de comportamento muito dinâmico. Por isso, projetar aplicações paralelas capazes de explorar esse potencial é, ainda, um trabalho desafiador [5]. Além disso, é imperativo que a execução das aplicações seja eficiente, ou seja, que o paralelismo existente nessas aplicações seja convenientemente explorado para que o potencial latente possa ser totalmente aproveitado [5]. Esta determinação é tratada em termos do Problema de Escalonamento de Tarefas (PET), alvo de muitos estudos [6] e foco deste trabalho. 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.

O Problema de Escalonamento de Tarefas, não é trivial e sim considerado *NP-Completo* em sua forma mais geral [7], 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 que permitem certas simplificações, por exemplo, um número ilimitado de processadores homogêneos que se comunicam com sobrecargas desprezíveis. Com isso, os resultados podem convergir para valores práticos pouco eficientes, uma vez que a maioria dos ambientes computacionais é composta por máquinas heterogêneas de diferentes capacidades de processamento e com a existência de sobrecargas da comunicação entre os processadores. O presente trabalho propõe uma heurística para o escalonamento estático de tarefas sob o modelo de comunicação *LogP* [8] para ambientes computacionais heterogêneos e limitados. O objetivo é minimizar o

makespan do escalonamento.

Este artigo está organizado da seguinte forma: A seção 2 descreve o problema de escalonamento de tarefas, os modelos de representação utilizados e uma possível classificação da área; a seção 3 apresenta as versões das heurísticas propostas e descreve os mecanismos gerais utilizados no projeto; na seção 4 é apresentada a análise dos resultados obtidos; a seção 5 descreve as conclusões e identifica os estudos que podem ser desenvolvidos a partir deste; a seção 6 apresenta as ilustrações mencionadas; e por fim, a seção 7 lista a bibliografia pesquisada.

2. O PROBLEMA DE ESCALONAMENTO DE TAREFAS

Esta seção descreve o problema de escalonamento de tarefas que é uma das questões mais importantes para o projeto de algoritmos paralelos eficientes. Inicialmente apresenta os modelos adotados para representar a aplicação e a arquitetura computacional alvo. Em seguida apresenta uma possível classificação do escalonamento de tarefas baseado em heurísticas.

2.1. O MODELO COMPUTACIONAL DE ESCALONAMENTO

Na área da computação, a modelagem do problema é fundamental para facilitar o projeto eficiente e a análise de desempenho dos algoritmos. Além disso, promove a consistência e organização durante a fase do projeto de programação. O principal objetivo de modelar o problema de escalonamento de tarefas é avaliar como uma aplicação será executada na máquina alvo destino; por isso o modelo proposto é muitas vezes composto por outros modelos de forma a retratar todo o ambiente envolvido no processo de escalonamento. Assim, o ambiente a ser considerado diz respeito a dois mini mundos fundamentais e distintos, no primeiro são consideradas as características relevantes ao desempenho da aplicação paralela, e no segundo as da arquitetura computacional onde será executada a aplicação.

2.1.1. O Modelo da Aplicação

Neste trabalho é considerada uma classe de aplicações paralelas que podem ser representadas por um Grafo Acíclico Direcionado denotado por *GAD* $G=(V,E,\varepsilon,\omega)$ onde V é o conjunto de n vértices do grafo e E o conjunto de arcos. Cada vértice $v \in V$ representa uma tarefa com peso de computação $\varepsilon(t)$ representando a quantidade de trabalho da tarefa e cada arco $(v_i, v_j) \in E$ representa a restrição de precedência entre as tarefas v_i e v_j, ou seja, a tarefa v_i deve completar sua execução antes que a v_j comece. Um peso $\omega(v_i, v_j)$ é associado ao arco representando a quantidade de dados a ser transmitida entre as duas tarefas. Cada tarefa é uma unidade de computação indivisível que pode ser uma instrução, uma sub-rotina, ou um programa inteiro, que quando alocada a um processador executa até terminar sem preempção, ou seja, sem interrupção. A Figura 1 apresenta um exemplo de um *GAD* de uma aplicação real utilizada nos testes.

2.1.2. O Modelo da Arquitetura Alvo

A arquitetura alvo pode ser representada por um grafo não direcionado $H=(P, C)$, onde $P = \{P_0, P_1, ..., P_k\}$ é o conjunto de k vértices que representam os processadores e C é o conjunto de arestas que representa os canais de comunicação que interligam processadores adjacentes. Neste trabalho, é considerado que os processadores estão totalmente conectados entre si e são heterogêneos, ou seja, possuem fatores de heterogeneidade distintos $h(P)$ representativos do atraso na execução de tarefas. O custo de computação, ou tempo de execução de uma tarefa t em um processador P é dado por: $custo\ (t,P) = \varepsilon(t).h(P)$.

As características da comunicação, consideradas relevantes, são definidas através de um modelo de comunicação que se incorpora ao modelo da arquitetura. É importante que o modelo resultante seja suficientemente abstrato, para que detalhes da máquina sejam

ignorados, mas ao mesmo tempo versátil para permitir a portabilidade do programa . Devido a variedade das máquinas paralelas existentes diversos modelos têm sido propostos, desde os inteiramente abstratos aos mais realísticos e práticos [9]. Os modelos abstratos são úteis por fornecerem uma visão mais ampla do problema, possibilitando a análise de propriedades gerais e da complexidade dos algoritmos neles formulados. Os realísticos permitem que a abstração ceda lugar a detalhes específicos da máquina alvo, possibilitando um melhor refinamento do algoritmo.

Um dos modelos realísticos criados, muito utilizado, é o modelo de *Latência* [13] que apenas considera o atraso ou latência da transmissão de uma mensagem. Posteriormente, surgiu o modelo *LogP* que considera a existência de parâmetros que permitem o cálculo preciso de cada comunicação realizada entre dois processadores. Este trabalho adota o modelo realístico *LogP* uma vez que os parâmetros considerados são necessários e suficientes para abordar o assunto do tema.

O Modelo LogP

Proposto por David Culler *et al.* em 1993 [8] é um modelo assíncrono que representa o fato de que nos multicomputadores, sendo a comunicação entre os processadores por troca de mensagens, cabe aos mesmos tratar ou ao menos iniciar cada uma delas. Por isso, um processador fica bloqueado um certo tempo preparando o envio e a recepção de mensagens, esse tempo caracteriza as sobrecargas impostas aos processadores envolvidos na transmissão e na recepção da mensagem.

Os parâmetros principais deste modelo são:

L - Latência: atraso no tempo de transmissão para enviar uma unidade de dados no canal de comunicação entre dois processadores.

o – *Overhead* ou sobrecarga: tempo que o processador gasta preparando a transmissão ou o recebimento de mensagens, tempo este em que o processador não pode realizar outras operações .

g - *Gap*: intervalo mínimo entre duas sobrecargas de envios ou de recebimentos consecutivos em um mesmo processador. Neste trabalho é assumido que $\boldsymbol{g = o}$.

P - Representa o número de processadores disponíveis.

A fim de se atingir maior precisão nos tempos de execução paralelos produzidos pelos algoritmos de escalonamento baseados neste modelo, convém especificar separadamente as sobrecargas de envio e de recebimento, denotadas por, respectivamente, O_s e O_r [22]. A Figura 2 ilustra o mapa resultante do escalonamento das tarefas do GAD numa arquitetura composta por 3 processadores sob o modelo *LogP*. Note que quando duas tarefas adjacentes são alocadas ao mesmo processador a comunicação entre elas se torna nula (tarefas 5 e 7).

2.2. O ESCALONAMENTO DE TAREFAS

Um escalonamento S é um conjunto finito de tuplas (v_i,P_j,t), cada um especificando que uma tarefa $v_i \in V$ é executada no processador $P_j \in P$ no tempo t. Dessa forma, cada tarefa $v_i \in V$ deve ser escalonada em pelo menos um processador $P_j \in P$ e essa tarefa somente pode ser executada depois que todos os seus predecessores imediatos tenham terminado e que os respectivos dados necessários para o início da sua execução estejam disponíveis no processador P_j. Além disso, é necessário definir as seguintes diretrizes para o escalonamento: Cada tarefa do Grafo deve ser executada em pelo menos um processador; duas ou mais tarefas não podem ser executadas ao mesmo tempo pelo mesmo processador; tarefas de computação não podem enviar qualquer resultado de seus dados, antes de terem terminado; uma mensagem de transferência de dados não pode ser recebida a menos que tenha sido enviada existe, portanto, uma demora de pelo menos L, entre uma sobrecarga de envio e a respectiva sobrecarga de recebimento; e todas as tarefas do grafo G são escalonadas. Uma vez definidos os modelos, da aplicação e da arquitetura, e estabelecidos os critérios sob os quais se realizará o escalonamento é possível determinar, através da heurística, o tempo de

execução paralelo final da aplicação, também conhecido como *Schedule Length* ou *Makespan*: $SL = max\{T_{inicial}(v_i, P_j) + \varepsilon(v_i).h(P_j)\}$ *onde* $v_i \in V$ e $P_j \in P$. O objetivo da heurística é minimizar esse valor.

2.3. CLASSIFICAÇÃO DO ESCALONAMENTO

O escalonamento de tarefas inicialmente deve ser classificado de acordo com o número de processadores a serem considerados no sistema. Quando a atribuição das tarefas de um programa paralelo é feita a um único processador disponível, o escalonamento é dito local. No caso onde as tarefas são designadas a vários processadores, que compõem a plataforma computacional distribuída, o escalonamento é denominado global. Uma outra classificação e que diz respeito ao escalonamento global, é a divisão do escalonamento em dinâmico e estático. Essa divisão é baseada no momento das decisões para a alocação das tarefas.

No escalonamento dinâmico, as decisões se realizam durante a execução do programa (*on the fly*), e ainda, a topologia do grafo de tarefas, os custos de comunicação e de computação não podem ser totalmente conhecidos em tempo de compilação do programa. No escalonamento estático das tarefas, objeto deste trabalho, as informações relativas ao tempo de execução de cada tarefa, ou ao menos, uma boa aproximação, e as relações de precedência são previamente especificadas. Assim, o grafo de tarefas que representa a aplicação ou programa é analisado pelo escalonador e cada tarefa é designada a um processador antes da execução da aplicação.

De acordo com a taxonomia proposta por Casavant e Kuhl [14] o escalonamento estático pode ser dividido em ótimo e sub-ótimo, uma vez que devido à complexidade do problema são aceitas soluções próximas do ótimo. Dentro da categoria de escalonamentos sub-ótimos, é possível classificá-los em aproximados e heurísticos. O escalonamento aproximado utiliza algum modelo computacional formal para o algoritmo. O heurístico, por sua vez, utiliza parâmetros identificados empiricamente ou intuitivamente, como responsáveis por algum fator de desempenho.

2.4. HEURÍSTICAS DE ESCALONAMENTO ESTÁTICO

Existem diversas abordagens para formulação de heurísticas de escalonamento estático que podem ser classificadas em duas classes principais [15]: Busca aleatória ou construção. Aquelas baseadas em busca aleatória são chamadas de meta-heurísticas, nas quais vários escalonamentos são realizados a procura da melhor solução, como por exemplo, Algoritmos Genéticos [16], *Simulated Annealing* [19] e Busca Tabu [17]. As heurísticas de construção, definem somente um único escalonamento, podem ser divididas em quatro grupos: heurísticas *list scheduling* [18], heurísticas de aglomeração de tarefas [21], heurísticas de análise de caminho crítico do grafo [12] e particionamento de grafos [20].

Nas heurísticas *list scheduling* a idéia básica é construir duas listas, uma de tarefas livres ordenadas de acôrdo com uma prioridade pré-estabelecida e outra de processadores ociosos. A cada passo da heurística é selecionada a tarefa livre de mais alta prioridade e o processador ocioso favorito para alocar a tarefa candidata. Uma tarefa é dita livre se todos os seus predecessores imediatos foram escalonados, e um processador é dito ocioso se não está executando nenhuma tarefa e nem está bloqueado preparando o envio ou o recebimento de uma mensagem. O processador favorito para alocar a tarefa é aquele que permite minimizar uma pré-definida função de custo; em ambientes de processadores homogêneos a função a minimizar é o tempo de início da tarefa, em ambientes de processadores heterogêneos procura-se minimizar o tempo de término da tarefa, isto porque nem sempre o processador onde uma tarefa pode começar mais cedo (pode ser o mais lento) lhe permitirá terminar mais rápido. Exemplos de heurísticas formuladas nesta técnica podem ser encontradas em [11, 15].

Nos algoritmos baseados em aglomeração de tarefas são criados conjuntos de tarefas que devem ser executadas em um mesmo processador, com o objetivo de eliminar as

comunicações existentes. Isto é possível, pois a comunicação entre as tarefas de um mesmo conjunto tem custo de comunicação nulo. Em [21] é descrito um exemplo desse grupo.

As heurísticas que utilizam a análise do caminho crítico procuram diminuir o caminho mais longo do grafo designando todas as tarefas a ele pertencentes ao mesmo processador eliminando o custo de comunicação entre elas. Um exemplo pode ser encontrado em [12].

A estratégia que emprega o particionamento divide o grafo em várias partes com o objetivo de minimizar arestas que conectam vértices de diferentes partições. Em [10] é apresentada uma heurística desta metodologia.

A abordagem escolhida neste trabalho adota a técnica de *list scheduling*. Comparada com as outras estratégias descritas, é geralmente mais prática, promove resultados de *Makespan* aceitáveis com complexidade mais atrativa, é voltada para ambientes com número de processadores limitados totalmente conectados, e é facilmente adaptável a plataformas de processadores homogêneos e heterogêneos. [1, 15]. Essas características são totalmente favoráveis ao objetivo de um escalonamento estático numa grade computacional, que é realizar um pré-escalonamento de tarefas a fim de amenizar a carga de trabalho do escalonador dinâmico. Devido às características instáveis dos recursos da plataforma é necessário que as decisões de escalonamento sejam feitas durante a execução da aplicação porém, realizando-se uma pré-etapa, através do escalonador estático, restará ao dinâmico apenas os ajustes finais momentâneos.

3. A HEURÍSTICA PROPOSTA

Neste trabalho é apresentada uma heurística de escalonamento estático baseado no modelo de comunicação *LogP* para arquiteturas paralelas heterogêneas de memória distribuída com características semelhantes a grades computacionais. Nesses ambientes de alta latência, a comunicação para transmissão de dados (sempre que tarefas adjacentes forem designadas a processadores distintos) acarreta sobrecargas ao sistema. Assim, o objetivo principal da nova heurística é avaliar, através de versões, políticas alternativas para minimizar os efeitos adversos dessas sobrecargas. A metodologia apresentada é uma extensão do trabalho de Kalinowski, Kort e Trystram [1] que adaptaram a heurística *list scheduling ETF* (*Earliest Task First*) [11] formulada no modelo de latência aos parâmetros do modelo *LogP*, ambas para ambiente homogêneo. A heurística *ETF* apresenta elevada complexidade uma vez que determina o processador favorito para escalonar uma tarefa por exaustão. A tarefa candidata é testada em todos os processadores disponíveis, o que apresentar menor tempo de início para a tarefa é escolhido. A heurística dos autores é denominada *ETFRGC* (*Earliest Task First Reservation Garbage Collection)* . Os passos principais da metodologia podem ser resumidos da seguinte forma:

1) Após o escalonamento de cada tarefa v_i em P_j uma área de reserva R é prevista no processador 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. Conforme apresenta a Figura 3;

2) Conforme novas tarefas vão sendo escalonadas, caso seja sucessora de v_i e designada a processador diferente de P_j, então uma sobrecarga de envio O_S é escalonada em R e ao término desta sobrecarga é transmitida a mensagem de dados. Quando a mensagem chegar ao processador destino, a respectiva sobrecarga de recebimento O_r é escalonada;

3) As sobrecargas de recebimento são escalonadas de forma ordenada pelo *tempo de chegada da mensagem* (TCM) antes do escalonamento da tarefa sucessora de v_i; e

4) Considerando que tarefas sucessoras de v_i podem ser designadas ao mesmo processador P_j o espaço destinado a essas sobrecargas de envio não é utilizado, tornando-se inútil uma vez que não poderá ser utilizado para 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 os espaços não utilizados dentro de todas as áreas de reserva. Um exemplo é ilustrado na Figura 4.

Em ambientes como grades computacionais encontrar o melhor processador por tentativas, como propõem a *ETF* e *ETFRGC*, não representa uma boa estratégia face ao provável alto número de recursos existentes. Assim, a abordagem escolhida nesta investigação, para avaliar formas de tratamento das sobrecargas, é implementada numa heurística base de *list scheduling* de complexidade inferior a *ETF*. A nova heurística é denominada *LSRGC* (*List Scheduling Reservation Garbage Collection*) proposta em versões, que suportam ambientes homogêneos e heterogêneos:

A) LSRGCV0 (versão 0)

Versão criada com o objetivo de implementar a técnica da *ETFRGC* numa *list scheduling* para assim ser comparada, em ambientes heterogêneos, às outras 3 políticas de tratamento propostas. Os passos desta versão são idênticos aos descritos na *ETFRGC*.

B) LSRGCV1 (versão 1)

1) Uma área de reserva *R* é prevista após o escalonamento de cada tarefa. 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 ordenadas pelo TCM, antes do escalonamento das tarefas de computação;
3) Ao fim do escalonamento é aplicada a rotina de coleta de lixo; e
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 do caminho crítico escalonado 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 prioriza, para ocupar as primeiras posições dentro das reservas *R,* 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*. A Figura 5 apresenta um exemplo.

B) *LSRGCV2 (versão 2)*

1) Após o escalonamento de cada tarefa de computação são reservadas às áreas R;
2) A cada passo do algoritmo, caso uma tarefa sucessora seja designada ao mesmo processador que aloja um de seus predecessores imediatos, é aplicada a rotina de coleta de lixo. Essa rotina 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* a cada escalonamento de tarefa, com isso não existirão espaços ociosos durante o escalonamento;
3) As sobrecargas de envio são associadas às mensagens e ao fim de cada uma delas as respectivas mensagens são enviadas; e
4) As sobrecargas de recebimento são escalonadas ordenadas pelo TCM, antes do escalonamento das tarefas de computação. A Figura 6 ilustra os passos com um exemplo.

C) *LSRGCV3 (versão 3)*

Semelhante a *LSRGCV1*, porém, ao invés de realizar a coleta de lixo no final do escalonamento, esta é feita a cada passo da heurística, reajustando o tamanho da reserva *R*. A Figura 7 apresenta um exemplo desta versão.

4. RESULTADOS EXPERIMENTAIS

Nos experimentos foram utilizados 65 grafos representativos de aplicações numéricas. Para facilitar as análises, os grafos foram agrupados em 6 classes distintas, representativas da

sua topologia, são elas: Diamantes, *OutTree* (Árvores Binárias), *Intree* (Árvores Binárias Invertidas), Randômicos, Irregulares e PSG (*Peer Set Graphs*). As quatro primeiras classes são compostas por grafos de tarefas e arcos unitários, o que permitiu formar o conjunto de grafos unitários; as duas últimas classes são compostas por grafos onde as tarefas e arcos são arbitrários formando, o conjunto de grafos não unitários.

Nos testes iniciais utilizou-se duas arquiteturas compostas por 8 e 12 processadores. Em cada uma das arquiteturas variou-se os parâmetros *o* e *L: (o=1, L=1), (o=1, L=4) e (o=4, L=1)* formando assim 3 baterias de testes para cada arquitetura. Os testes foram divididos pela classe do grafo e os percentuais apresentados representam o valor de melhor ou igual *makespan* obtidos, por cada heurística, considerando todos os grafos da classe. Com o intuito de condensar os resultados obtidos, são descritas as médias, considerando as 3 baterias de testes, para cada classe de grafos. Na arquitetura de 12 processadores, ainda, se detalhou mais os testes considerando separadamente os valores das sobrecargas O_s e O_r *(O_s =1, O_r =4 L =1), (O_s =4, O_r =1 L = 1), (O_s =1 O_r = 4 L= 4)* e *(O_s=4 O_r =1 L =4)* por conjunto de todos os grafos, conforme está ilustrado na Figura 8. Numa etapa posterior foram avaliadas as heurísticas em arquiteturas com 32 e 64 processadores.

Grafos Diamantes

Nos grafos diamantes todas as tarefas são relacionadas por predecessores e sucessores comuns, são muito utilizados para representar multiplicação de matrizes. Foram utilizados 11 grafos com um número de tarefas variando de 9 a 1024. Nos experimentos realizados com a arquitetura de 8 processadores, em média nas 3 baterias, a heurística *LSRGCV2* atingiu 64% de melhor ou igual *makespan*, a *LSRGCV3* atingiu 30%, a *LSRGCV1* 27% e a *LSRGCV0* 18%. Na arquitetura com 12 processadores a *LSRGCV2* atingiu 70%, a *LSRGCV3* 48%, a *LSRGCV0* 33% e a *LSRGCV1* 30%.

Grafos *Intree*

Nesta classe, também conhecida como árvores binárias completas invertidas, cada tarefa têm dois predecessores imediatos independentes, com exceção das tarefas origens; e apenas um sucessor imediato, com exceção da tarefa de saída. Esta topologia de grafos é muito utilizada para representar redução nas operações como a soma em paralelo. Para os testes foram utilizados 7 grafos com um número de tarefas variando entre 3 e 511. Nos experimentos com 8 processadores, na média, as heurísticas *LSRGCV2* e *LSRGCV3* empataram em 95% de melhor ou igual *makespan*, a *LSRGCV1* e a *LSRGCV0*, também, empataram em 52%. Na arquitetura com 12 processadores, as duas primeiras empataram em 91% e as duas segundas em 43%. A queda de desempenho das heurísticas é devida a terem sido escalonadas tarefas sucessoras em processadores que não possuíam nenhum dos dois predecessores imediatos escalonados. Com isso mais sobrecargas ocorreram contribuindo para o aumento do *makespan*.

Grafos *OutTree*

Topologia representativa das árvores binárias completas, onde cada tarefa tem apenas um predecessor imediato, com exceção da tarefa origem (raiz) e possuem dois sucessores imediatos, com exceção das tarefas de saída. Algoritmo de difusão e divisão são exemplos deste tipo de grafo. Para os testes foram utilizados 8 grafos com número de tarefas variando entre 7 e 511. Na arquitetura com 8 processadores, na média, as heurísticas LSRGCV2 e LSRGCV3 apresentaram 67% de melhor ou igual makespan. As *LSRGCV1* e *LSRGCV0* empataram em 52%. O aumento de processadores para 12 favorece mais o desempenho da *LSRGCV2* que atinge 84% seguida da *LSRGCV3* com 67%; ambas as *LSRGCV0* e *LSRGCV1* empatam em 30%.

Grafos Randômicos

Os grafos desta classe são gerados aleatoriamente e tem como objetivo representar qualquer estrutura para que se possa avaliar o desempenho da heurística sobre grafos com estruturas irregulares. Para os testes foram utilizados 21 grafos com um número de tarefas variando entre 80 e 546. Nos experimentos com 8 processadores a *LSRGCV0 e a LSRGCV1* apresentaram o pior desempenho 0% perdendo em todos os 8 grafos (nas 3 baterias de testes) para as outras heurísticas; a heurística *LSRGCV3* apresentou o melhor valor 73% seguida da *LSRGCV2* com 33%. O aumento de processadores para 12 favoreceu a *LSRGCV3* com 78%, a seguir a *LSRGCV2* com 35%; a *LSRGCV0 e a LSRGCV1* continuaram com 0%.

Grafos Irregulares

Conjunto de 7 grafos de topologia irregular variando de 7 a 41 tarefas caracterizados por possuírem tarefas e arestas com pesos de grande valor. Na arquitetura de 8 processadores a heurística com melhor ou iguais valores de *makespan* foi a *LSRGCV3* que apresentou 95%, seguida da *LSRGCV2* com 86%, *LSRGCV1* com 76% e a *LSRGCV0* com 72%. Na arquitetura com 12 processadores, ainda persiste a *LSRGCV3* com o melhor resultado 86%, a seguir a *LSRGCV2* com 81%, a *LSRGCV1* com 67% e a *LSRGCV0* com 62%.

Grafos PSG (*Peer Set Graphs*)

Conjunto de 11 grafos de topologia variada compostos por poucas tarefas, que variam de 7 a 18, e considerados de granulosidade fina, ou seja, quando os pesos de comunicação superam os de computação. Em ambas as arquiteturas a heurística de melhor desempenho foi a *LSRGCV3* com 85%. Na arquitetura de 8 processadores a *LSRGCV1* atingiu 82%, a *LSRGCV0* 79% e a *LSRGCV2* 76%. Com o aumento de processadores para 12 a LSRGCV3 continua liderando com 85% seguida da *LSRGCV2* com 82%, a *LSRGCV1* com 79% e a *LSRGCV0* com 64%.

Na arquitetura de 12 processadores foram, ainda, realizados testes em que se considerou distintos valores para as sobrecargas de envio e recebimento a *LSRGCV3* apresentou, na média, o melhor resultado atingindo 72%, seguida da *LSRGCV2* com 64% e ambas as *LSRGCV1* e *LSRGCV0* empatadas em 32%. Conforme pode ser observado na Figura 8. Nova etapa de testes foi realizada em arquiteturas com 32 e 64 processadores. Na média, considerando as quatro arquiteturas nos grafos unitários a *LRGCV2* apresentou 62% seguida da *LSRGCV3* com 60%, a *LSRGCV0* com 33% e a *LSRGCV1* com 26%. No conjunto de grafos não unitários a *LSRGCV3* apresentou 86%, a *LSRGCV2* 81%, a *LSRGCV1* 74% e a *LSRGCV0* 68%.

As Tabelas 1, 2 e 3 apresentam as comparações das heurísticas, o objetivo é verificar em quantos grafos uma heurística foi melhor que as outras.

	LSRGCV0	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*
M	*	14	1	6		*	17	10	13		*	18	8	22
LSRGCV0 I	*	43	21	17		*	39	25	18		*	43	37	24
P	*	8	43	42		*	9	30	34		*	4	20	19
M	8	*	4	3		9	*	12	11		4	*	7	16
LSRGCV1 I	43	*	18	20		39	*	21	20		43	*	31	26
P	14	*	43	42		17	*	32	34		18	*	27	23
M	43	43	*	16		30	32	*	18		20	27	*	22
LSRGCV2 I	21	18	*	28		25	21	*	31		37	31	*	34
P	1	4	*	21		10	12	*	16		8	7	*	9
M	42	42	21	*		34	34	16	*		19	23	9	*
LSRGCV3 I	17	20	28	*		18	20	31	*		24	26	34	*
P	6	3	16	*		13	11	18	*		22	16	22	*

Tabela 1 – Comparação *Pair-Wise* do número de Melhor (M), Igual (I), e Pior (P) *makespan* de todos os grafos em arquiteturas heterogêneas de 12, 32 e 64 processadores com valores de *o = 1 e L = 1.*

	LSRGCV0	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*
M	*	17	5	6		*	14	12	17		*	24	16	21
LSRGCV0 I	*	35	22	19		*	44	31	24		*	37	30	29
P	*	13	38	40		*	7	22	24		*	4	19	15
M	13	*	6	5		7	*	8	11		4	*	13	14
LSRGCV1 I	35	*	20	22		44	*	28	27		37	*	28	31
P	17	*	39	38		14	*	29	27		24	*	24	20
M	38	39	*	15		22	29	*	10		19	24	*	17
LSRGCV2 I	22	20	*	34		31	28	*	37		30	28	*	42
P	5	6	*	16		12	8	*	18		16	13	*	6
M	40	38	16	*		24	27	18	*		15	20	6	*
LSRGCV3 I	19	22	34	*		24	27	37	*		29	31	42	*
P	6	5	15	*		17	11	10	*		21	14	17	*

Tabela 2 – Comparação *Pair-Wise* do número de Melhor (M), Igual (I), e Pior (P) *makespan de* todos os grafos em arquiteturas heterogêneas de 12, 32 e 62 processadores com valores de *o = 1 e L = 4*.

	LSRGCV0	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*		*LSRGCV0*	*LSRGCV1*	*LSRGCV2*	*LSRGCV3*
M	*	19	4	5		*	18	15	15		*	19	16	14
LSRGCV0 I	*	32	21	17		*	40	13	13		*	44	24	26
P	*	14	40	43		*	7	37	37		*	2	25	25
M	14	*	6	5		7	*	12	9		2	*	13	9
LSRGCV1 I	32	*	17	19		40	*	13	15		44	*	23	25
P	19	*	42	41		18	*	40	41		19	*	29	31
M	40	42	*	13		37	40	*	20		25	29	*	16
LSRGCV2 I	21	17	*	27		13	13	*	29		24	23	*	35
P	4	6	*	25		15	12	*	16		16	13	*	14
M	43	41	25	*		37	41	16	*		25	31	14	*
LSRGCV3 I	17	19	27	*		13	15	29	*		26	25	35	*
P	5	5	13	*		15	9	20	*		14	9	16	*

Tabela 3 – Comparação *Pair-Wise* do número de Melhor (M), Igual (I), e Pior (P) *makespan* de todos os grafos em arquiteturas heterogêneas de 12, 32 e 64 processadores com valores de *o = 4 e L = 1*.

5. CONCLUSÃO E TRABALHOS FUTUROS

O objetivo deste trabalho foi estudar formas de tratamento para minimizar os efeitos adversos que as sobrecargas das comunicações, entre processadores, causam em ambientes computacionais heterogêneos. Foi proposta uma heurística de escalonamento para aplicações que podem ser representadas por grafos acíclicos direcionados (GAD) em um número limitado de processadores heterogêneos sob os parâmetros do modelo *LogP*, denominada *LSRGC*. A nova heurística do grupo de *list scheduling* é apresentada em quatro versões que implementam distintas políticas para tratamento das sobrecargas. O quesito considerado na avaliação do melhor desempenho, é o *makespan* do escalonamento.

De acordo com os resultados as heurísticas *LSRGCV2* e *LSRGCV3* que propuseram realizar a coleta de espaços, não utilizados, a cada passo do escalonamento apresentaram os índices mais significativos. Considerando a média final, nas 4 arquiteturas analisadas, a *LSRGCV3* atingiu 68% de melhores ou iguais valores de *makespan*, a seguir a *LSRGCV2* com 67%, a *LSRGCV0* com 42% e a *LSRGCV1* com 39%. A Figura 9 ilustra essa comparação.

Avaliando-se a qualidade da *LSRGCV3* em relação a escalabilidade de poder computacional, o aumento do número de processadores, também, foi favorável a essa versão possibilitando a estabilidade de seu desempenho, mantendo a qualidade dos resultados.

Nesta investigação foram realizados cerca de 150 testes que permitiram concluir que a heurística *LSRGCV3* está indicada para realizar o escalonamento estático de tarefas em grades computacionais. Visto que apresentou resultados de qualidade quando aplicada em ambientes

de latência superior *(L=4)* à computação das tarefas e manteve a consistência nos resultados quando se considerou a escalabilidade do sistema. Essas características, alta latência e escalabilidade, são condições intrínsecas às plataformas de grades computacionais.

Os trabalhos que podem ser identificados a partir deste são:

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

ii) Investigar novas alternativas para minimizar efeitos negativos que as sobrecargas causam ao sistema, inclusive para aquelas existentes quando tarefas adjacentes são escalonadas no mesmo processador, que neste estudo foram desconsideradas;

iii) Especificar um modelo extensivo ao *LogP* capaz de suportar processadores heterogêneos, como o *HlogP*; e

iv) Investigar alternativas para tratar os efeitos adversos das sobrecargas, como aplicar a técnica de coleta dos espaços, não utilizados, ao término de cada *nível topológico* do grafo.

6. ILUSTRAÇÕES

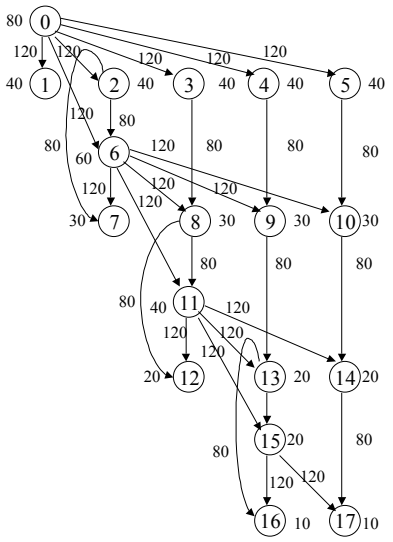

Figura 1 – Grafo da Eliminação Gaussiana, pertence à classe de Irregulares com 18 tarefas (*Ir18*).

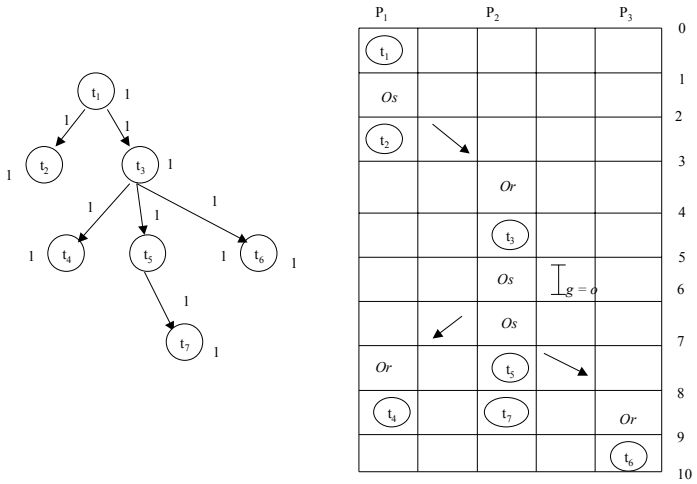

Figura 2 – Escalonamento final obtido utilizando-se o modelo *LogP*. O peso dos arcos, o tempo de computação das tarefas e a latência L, e as sobrecargas O_s e O_r e *gap* são unitários. O custo de comunicação entre duas tarefas é dado pelo *peso* do *arco x latência. Makespan = 10*

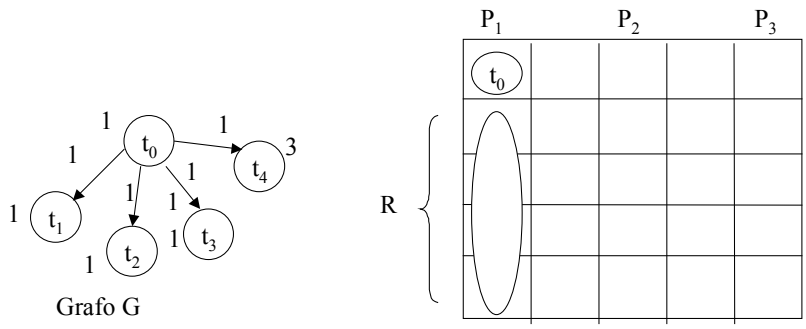

Figura 3 – Exemplo do escalonamento da tarefa t_0 do GAD utilizado nos próximos exemplos. Após a tarefa t_0 é reservada uma área R de tamanho 4, capaz de acomodar as sobrecargas de envio para os 4 sucessores, caso sejam alocados a processadores distintos de P_1.

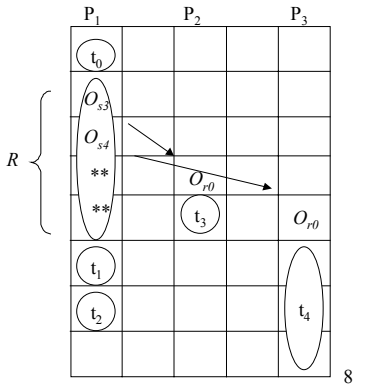

Figura 4 – Exemplo das heurísticas *ETFRGC* e *LSRGCV0*. Após t_0 é reservada a área R. As tarefas sucessoras t_1 e t_2 são escalonadas no mesmo processador de t_0, as reservas (**) tornam-se inúteis e serão removidas ao final do escalonamento. As tarefas t_3 e t_4 são designadas a processadores diferentes de P_1 e são escalonadas as respectivas sobrecargas O_s e O_r. As mensagens são enviadas após o término de cada sobrecarga de envio. *Makespan = 8.*

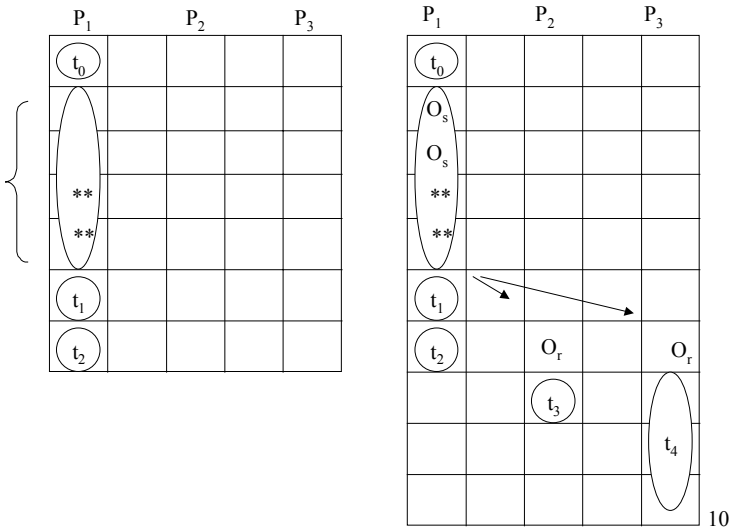

Figura 5 – Exemplo da heurística *LSRGCV1*. O mapa da esquerda ilustra o escalonamento das tarefas t_1 e t_2, os asteriscos indicam que os espaços não foram utilizados para escalonar as sobrecargas de envio para t_1 e t_2. Esses espaços serão removidos após o escalonamento de todo o grafo. A direita está o escalonamento de t_3 e t_4 com as sobrecargas anônimas alocadas e as mensagens sendo enviadas ao fim da reserva. Após a coleta de lixo as sobrecargas serão ordenadas priorizando para ocupar a primeira posição em R, a sobrecarga que envia mensagem para t_4 (tarefa do caminho crítico escalonado).
***Makespan* final (após a ordenação) = 7.**

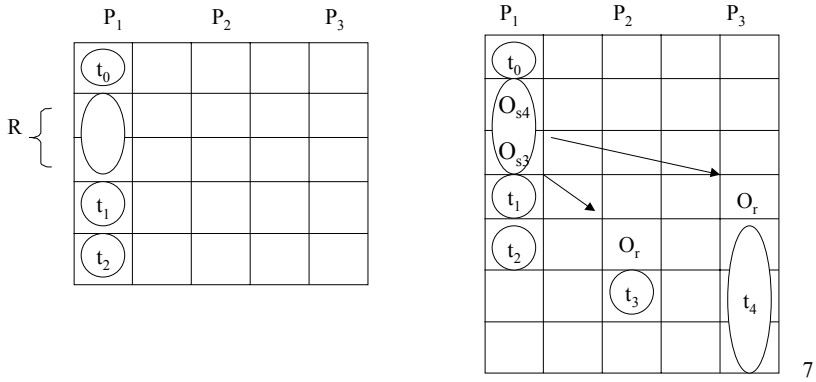

Figura 6 – Exemplo da heurística *LSRGCV2*. A esquerda o mapa ilustra a coleta de espaços inativos a cada escalonamento de tarefa. O mapa da direita ilustra que as sobrecargas enviam as mensagem após seu término. *Makespan = 7.*

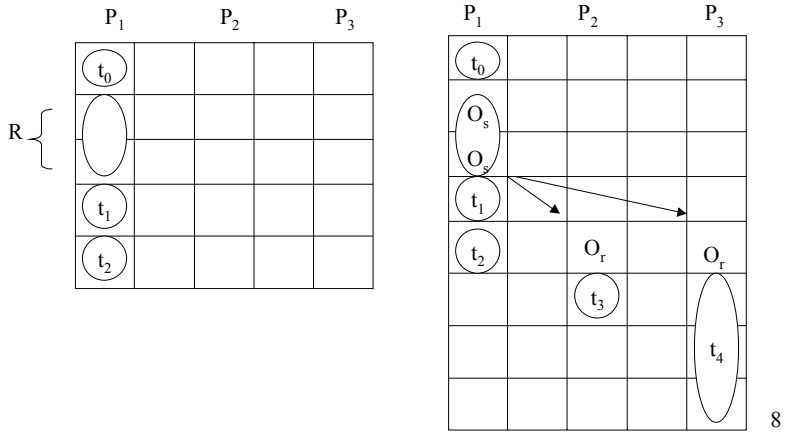

Figura 7 – Exemplo da heurística *LSRGCV3*. O mapa da esquerda ilustra a coleta de lixo a cada tarefa escalonada. A direita indica que as sobrecargas são inicialmente anônimas e as mensagens somente circulam ao fim da reserva R. Numa fase posterior serão ordenadas priorizando as que enviam dados para tarefas do caminho crítico escalonado. Neste caso t_4 é tarefa do caminho crítico escalonado. *Makespan* final (após a ordenação) = 7.

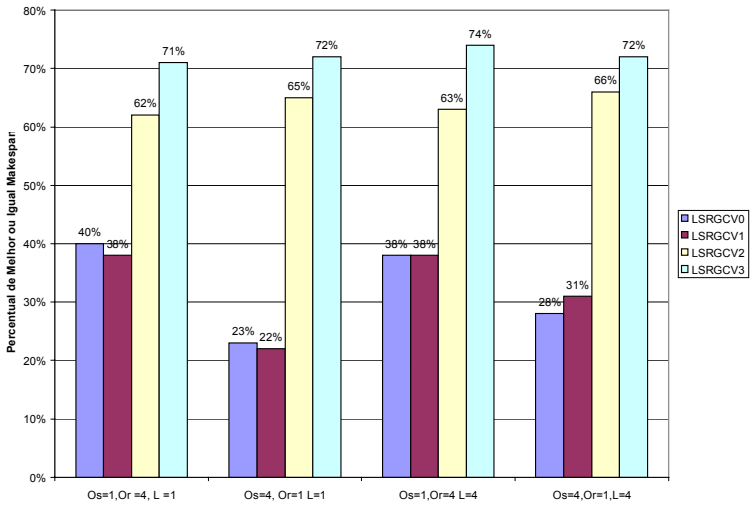

Figura 8 – Comparação considerando o conjunto de todos os grafos, os percentuais indicam o somatório de melhor ou igual *makespan* na arquitetura de P = 12 onde se variou os valores de *Os*, *Or e L*.

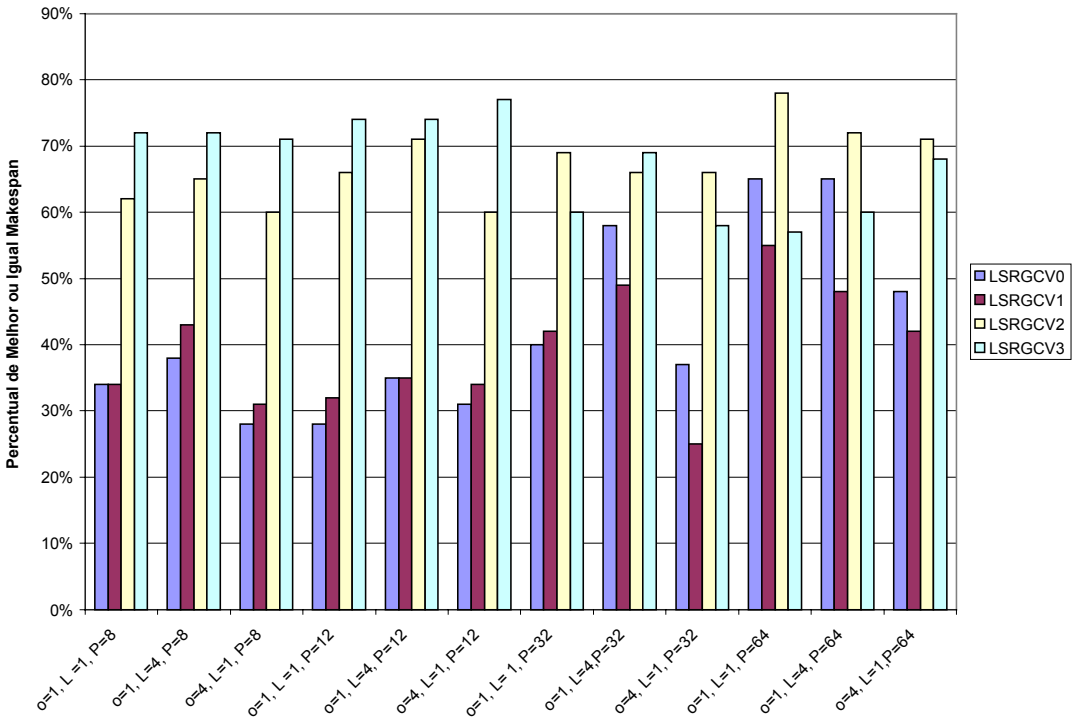

Figura 9 – Comparação considerando o conjunto de todos os grafos. Os percentuais indicam o somatório de melhor ou igual *makespan* em arquiteturas onde se variou os valores de *o*, *L* e *P*.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] KALINOWSKI T., KORT I. e TRYSTRAM D. List scheduling of general task graphs under LogP. *Parallel Computing 26 1109 – 1128* (1999).

[2] FOSTER I. The Grid: A new infrastructure for 21st century science. *Physics Today* (2002).

[3] BUYYA R. High Performance Cluster Computing: Programming and Applications. *Prentice Hall PTR, New Jersey* (1999).

[4] FOSTER I., KESSELMAN C. e TUECKE S. The anatomy of the grid. *Proceedings of International J. Supercomputer Applications* (2001).

[5] BOERES C., REBELLO V. E. F. Towards optimal task scheduling for realistic machine models: Theory and practice. *The International Journal of High Performance Applications 17, 2 173-190* (2003).

[6] KWOK, Y. e AHMAD, I. Static scheduling algorithms for allocating direct task graphs to multiprocessors. *ACM Computing Surveys 31, 4 367-432* (1999).

[7] GAREY, M. R. e JOHNSON, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. *W. H. Freeman & Co*., (1979).

[8] CULLER, D. E., KARP, R. M., PATTERSON, D. A, SAHAY, A ., SCHAUSER, K. E., SANTOS, E., SUBRAMONIAN, R. e VON EICKEN, T. LogP: Towards a realistic model of parallel computation. *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1-12* (1993).

[9] RANAWEERA, S. e AGRAWAL, D. P. A task duplication based scheduling algorithm for heterogeneous systems. *Proceedings of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS-00), 445-450* (2000).

[10] EHRENFEUCHT, A. e ROZEMBERG, G. Theory of 2-structures, part i: Clans, basic subclasses and morphisms. *Theoretical Computer Science 70, 3, 277-303* (1999).

[11] HWANG, J., CHOW Y., ANGER, F. e LEE, C. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing 18, 2, 244-257* (1989).

[12] KWOK, Y. e AHMAD I. Dynamic critical-path scheduling: Na effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems 7, 5, 506-521* (1996).

[13] VALIANT L. G. A bridging model for parallel computation. *ACM, 33: 103-111* (1990).

[14] CASAVANT, T. L. e KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering 14, 2, 141-154* (1988)

[15] TOPCUOGLU, H., HARIRI, S. e WU, M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *In IEEE Transactions on Parallel and Distributed Systems 13 3* (2002)

[16] 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).

[17] PORTO, S. C. S., KITAJIMA, J. P. e RIBEIRO, C. C. Peformance evaluation of a parallel tabu search task scheduling algorithm. *Parallel Computing 26, 1, 73-90* (2000).

[18] YANG, T. e GERASOULIS, A. List scheduling with and without communication delays. *Parallel Computing Journal 19, 12, 1321-1344* (1993).

[19] SHROFF, P., WATSON, D. W., FLANN, N. S., FREUND, R. F. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environment. *In 5th (HCW`96) 98-117* (1996).

[20] McCREARY, C. L., THOMPSON, J., GILL H., SMITH, T. e ZHU, Y. Partitioning and scheduling using graph decomposition. *Relatório Técnico CSE-93-06, Department of Computer Science and Engineering, Auburn University* (1993).

[21] PALIS, M. A., LIOU, J. C., RAJASEKARAN, S., SHENDE, S. e WEI, D. S. L. Online scheduling of dynamic trees. *Parallel Processing Letters* 5, 4, 635-646 (1995).

[22] REBELLO, V.E.F. *Personal Communication*. *UFF*. 2004