UNIVERSIDADE FEDERAL DO PAMPA – UNIPAMPA
CIÊNCIA DA COMPUTAÇÃO

PRÁTICAS EM PROGRAMAÇÃO

Trabalho 2

Mapeamento de Tarefas em MPSoc

Alunos:
Endrio Biasi – 141150013
Isadora Ferrão– 151151387
João Otávio Chervinski – 141150031
Sherlon Almeida - 151150179

Alegrete, 15 de Setembro 2017

1. INTRODUÇÃO

O segundo trabalho de Práticas de Programação consiste em implementar um simulador de mapeamento de tarefas. Para este fim, foi codificado um simulador de um MPSOC (Multiprocessor System-on-chip), que é um circuito integrado que utiliza múltiplos processadores. O objetivo do trabalho consiste na codificação de duas estratégias diferentes de alocação dos processos que serão executados no MPSoc. Existem vários métodos para que as tarefas sejam alocadas nos elementos de processamento (EPs), métodos estes que visam melhorar o desempenho do MPSOC como um todo e tendo como um dos principais objetivos evitar o gargalo dos canais de comunicação que conectam as tarefas. As técnicas implementadas pelo grupo foram o **(1)** First-Free e o **(2)** Path-Load.

(1) First-Free: Esta é uma técnica simples de mapeamento, que consiste apenas na alocação dos processos para os elementos de processamentos na ordem em que as mesmas chegam. Desse modo, a alocação vai preenchendo a matriz do MPSoc linha por linha, preenchendo as posições vazias até que a linha acabe, e então passando para a linha abaixo, até que todos os processos sejam alocados em algum elemento de processamento.

Esta técnica é muito rudimentar e na teoria não deve oferecer nenhum benefício à alocação das tarefas, pelo contrário, pois tarefas que se comunicam podem acabar ficando distantes, conectadas por canais sobrecarregados.

(2) Path-Load: Neste método de mapeamento, as tarefas são alocadas de acordo com um algoritmo que realiza o cálculo do custo dos canais de comunicação entre as tarefas, e essas então, são alocadas nas posições que apresentam os menores custos de comunicações de ida e volta possíveis. O cálculo do custo dos canais de ida e volta são necessários devido a natureza assíncrona dos caminhos de comunicação em um MPSoc. O custo do mapeamento em um EP é dado pela Equação 1 abaixo:

$$custo_k = \Sigma\, taxa_{c(i,j)} + \Sigma\, taxa_{c(j,i)} \qquad (1)$$

Onde custo é a soma dos custos de ida e volta do EP de origem até o EP de destino, e *c(i,j)* é o carga de ocupação do canal que liga o EP *i* ao *j*. Essa é uma técnica de mapeamento que apresenta um bom desempenho, e foi testada contra outras heurísticas na literatura (CARVALHO, 2007).

As seções seguintes do trabalho se apresentam como segue: na seção **(2)** são discutidos os aspectos relacionados ao desenvolvimento do trabalho e codificação das heurísticas; na seção **(3)** são apresentados os gráficos das execuções e discutidos os resultados obtidos; na seção **(4)** são realizadas conclusões acerca do trabalho desenvolvido; na seção **(5)** são apresentadas as referências utilizadas no desenvolvimento do trabalho e, por fim, no **Apêndice** são mostrados os códigos desenvolvidos para este trabalho.

2. DESENVOLVIMENTO

Para implementação das técnicas utilizadas foi determinado o uso da linguagem Python versão 3.5.2, por ser uma linguagem dominada pela maioria dos membros do grupo, bem como, pela sua legibilidade, poder expressivo e pela grande gama de estruturas oferecidas pela linguagem. Para ilustrar as saídas dos resultados, usou-se o Graphviz, um pacote de ferramentas de código aberto que tem como objetivo principal ilustrar gráficos especificados em scripts com a extensão .dot. Também foi utilizada a ferramenta Gnuplot, também de código aberto, para a elaboração dos gráficos comparativos das estratégias de mapeamento. A seguir apresenta-se a interface do sistema e as partes relevantes explicadas dos algoritmos First-Free e Path-Load.

2.1 INTERFACE DO SISTEMA

Nesta subseção serão apresentados os meios de interagir com o sistema, bem como explicações sobre o arquivo de entrada, execução e interpretação da saída.

Para executar o programa digite **<python3 mapeamento.py arquivo_de_entrada tamanho_MPSoc>**, como é possível ver na Figura 1. O arquivo de entrada possui uma configuração padrão, e este arquivo pode ser gerado de duas formas. A primeira forma é escrever em um arquivo os dados desejados para teste, e a segunda forma é gerando arquivos automaticamente com o código **geradorDeEntradas.py** em anexo.

Com este gerador automático podem ser geradas entradas de Pipeline, Árvores binárias e Grafos, e a execução é dada da seguinte forma **<python3 geradorDeEntradas.py nome_arquivo numero_arquivos numero_linhas opcao>,** onde:

- **nome_arquivo** = nome da sequência de arquivos a ser gerada.
- **numero_arquivos** = quantidade de arquivos que se deseja gerar;
- **numero_linhas** = quantidade de nós desejada
- **opcao** = define a estrutura a ser gerada Ex: Arvore, Pipeline, ...

Após gerar o arquivo desejado e executar o programa é possível escolher entre 1 (First Free) e 2 (Path Load). No momento que é executado, o programa gera dados para serem analisados pelo usuário e facilitar na compreensão dos resultados obtidos, estes dados são mostrados tanto no terminal, quanto em figuras e gráficos de saída, o que será apresentado a seguir.

Figura 1 - Interface do Sistema.

No terminal é apresentado o conteúdo das Figuras 2 e 3. Na Figura 2 é possível identificar a maneira como os dados estão sendo estruturados no computador, a interpretação é dada da seguinte forma:

A [B, C, [D, E, F, G, H] , [D, E, F, G, H]] , onde:

- **A = É o nó de partida, que se conecta com D;**
- **B = É a posição i na matriz do XY do MPSoc;**
- **C = É a posição j na matriz do XY do MPSoc;**
- **D = É o nó de destino, cuja partida é A;**
- **E = É a ocupação (Percentagem) do canal de ida;**
- **F = É a ocupação (Nº de pacotes) do canal de ida;** (Não está sendo usado)
- **G = É a ocupação (Percentagem) do canal de volta;**
- **G = É a ocupação (Nº de pacotes) do canal de volta;** (Não está sendo usado)

```
GRAFO POR PARTES
0 [0, 0, [2, 150, 22, 170, 20], [1, 340, 10, 300, 15]]
1 [0, 1, [3, 470, 10, 300, 15], [4, 300, 15, 480, 15]]
2 [0, 2, [5, 380, 10, 230, 10], [6, 240, 5, 140, 5]]
3 [0, 3, [9, 210, 15, 280, 25]]
4 [0, 4]
5 [0, 5, [8, 190, 25, 340, 20]]
6 [1, 0, [7, 120, 20, 360, 25]]
7 [1, 1]
8 [1, 2]
9 [1, 3]
```

Figura 2. Saída 1 no terminal

E na Figura 3 os dados representam a saída já computada da estratégia escolhida, para cada canal de comunicação do MPSoc são atribuídas tuplas **(Origem, Destino)** de identificação, por exemplo, na Figura 3 tem uma linha com os dados **(1,2) [47]**, o que representa o canal de comunicação entre 1 e 2, cuja percentagem de ocupação é 47. Desta forma interpretam-se todos os demais dados gerados.

A seguir são apresentadas as formas de visualização dos dados gerados, que são gráficos ilustrando o comportamento do MPSoc.

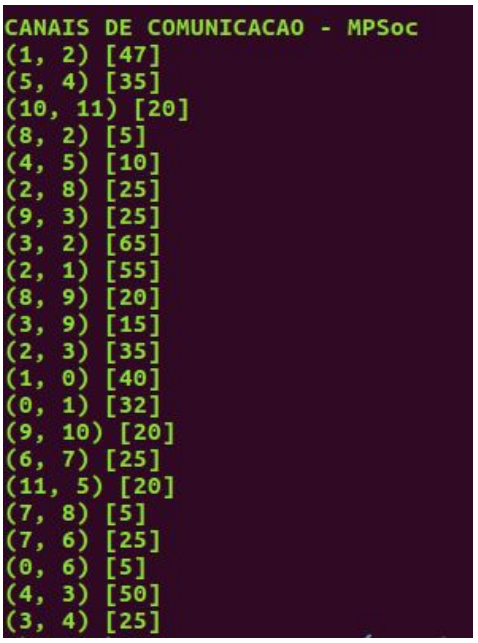

Figura 3. Saída 2 no terminal

2.2 First-Free

```
def firstFree(self):
    flag = 0
    for i in range(self.linhas):
        for j in range(self.colunas):
            if (i*self.linhas+j) < len(self.grafo):
                self.grafo[i*self.linhas+j][0] = i
                self.grafo[i*self.linhas+j][1] = j
            else:
                flag = 1
                break
        if flag == 1:
            break
```

Figura 4. Algoritmo First-Free

Como pode ser observado na figura acima, a função implementada mapeia no grafo as posições das tarefas de acordo com a heurística First-Free. O **for i in range(self.linhas):** percorre as linhas do MPSoc e o **for j in range (self.colunas):** percorre as colunas do MPSoc. Caso a interação do grafo for menor que o tamanho do grafo, mapeia o MPSoc. Se não, sai do laço **j** e autoriza a saída do laço **i**. Com isso, o **if flag ==1:** informa que saia do laço **i**, pois, já está autorizado.

2.3 Path-Load

Para melhor visualização, o algoritmo Path-Load encontra-se nos anexos deste relatório. Inicialmente, a primeira tarefa encontra-se na posição **(0,0),** o **self.grafo**[0][0] = 0 e o **self.grafo**[0][1] = 0 inicializam a pos em zero. Para os vértices do grafo e para cada conexão do nó, o valor da posição **i** do nó atual, valor da posição **j** do nó atual, valor de ocupação do canal de ida até o destino e valor de ocupação do canal de volta do destino **N = self.linhas**. O **for ii in range (self.linhas**) percorre as linhas do MPSoc e o **for jj in range(self.colunas)** percorre as colunas do MPSoc. Se a posição que está sendo olhada é diferente da posição do nó atual, testa-se essa posição para saber o seu custo. O **if(j != jj**) percorre em x, enquanto não estiver na linha desejada, se a coordenada x atual for menor que a desejada e se esse canal ainda não tiver peso, adiciona o peso da ida. O **if(i != ii):** percorre em y, enquanto não estiver na coluna desejada e se a coordenada y atual for menor que a desejada, adiciona o peso da ida.

Por fim, são feitos alguns testes dos canais de volta. Primeiramente percorrendo em x e depois o y, enquanto não estiver na linha desejada, se a coordenada x atual for menor que a desejada e se o canal ainda não tiver peso, **custo += canalVolta** adiciona o peso da ida. Por fim, **custos = sorted(cargas.items(), key=operator.itemgetter(1))** ordena os custos das possíveis posições. Posteriormente o código escolhe a melhor posição que ainda não esteja ocupada para ser preenchida pela nova tarefa.

3. Resultados Obtidos

Ao todo foram realizados testes em 5 cenários diferentes, onde em 3 cenários as tarefas estavam dispostas em um formato de árvore, e nos outros 2 as tarefas estavam ordenadas simulando uma organização em pipeline, ou seja, uma tarefa depende de uma das anteriores e assim por diante. A organização em pipeline favorece a disposição no MPSoc utilizando o método First-Free. Para cada um dos cenários foi gerada uma visualização do estado do final do MPsoc, assim como a representação da entrada em um grafo, por fim foi gerado um gráfico mostrando os valores de ocupação dos canais do MPSoc e a média de ocupação total entre todos os canais.

As linhas entradas utilizadas foram organizadas segundo o seguinte modelo:

T U V X Y Z

Onde:

- T é o número do vértice (ID da tarefa)
- U representa o ID de uma das tarefas que está ligada à T
- V representa o número de ciclos de ida necessários*
- X representa a ocupação de ida dos canais que ligam T->U
- Y representa o número de ciclos de volta necessários*
- Z representa a ocupação de volta dos canais que ligam U->T

***OBS**: os valores de V e Y foram adotados como sendo 0 com a finalidade de facilitar a simulação. Inicialmente os valores V e Y estão sendo lidos e armazenados, e podem ser utilizados posteriormente para alguma heurística, mas não estão sendo utilizados nos cálculos das heurísticas até então implementadas.

A seguir encontram-se os casos de teste utilizados e sua análise de resultados, para isso, os 5 cenários de teste foram divididos em subseções. Cada subseção contém a entrada gerada (**Entrada**), a ilustração da estrutura utilizada (**Estrutura**), o MPSoc 6x6 com os canais de comunicação e suas percentagens (**FirstFree** e **Path Load**) e gráficos de resultados para cada uma das estratégias de mapeamento. Os gráficos contém a percentagem de ocupação de cada canal de comunicação, e uma linha preta que identifica a média de ocupação dos canais.

Visando identificar de maneira visual as conexões desocupadas, ocupadas e sobrecarregadas, foi definido um intervalo no gerador do script, caso a percentagem de ocupação do canal esteja entre:

- 0%, cor **VERDE FRACO** (Desocupado);
- 0% e 33%, a cor **VERDE** é utilizada;
- 33% e 66%, a cor **AMARELA** é utilizada;
- 66% e 99%, a cor **LARANJA** é utilizada;
- 100% ou maior, a cor **VERMELHA** é utilizada (Sobrecarregado);

3.1 Cenário de Teste 1 - Pipeline

O cenário de teste 1 representa um pipeline, como pode-se ver na Figura 5 e 6, temos respectivamente a entrada de texto e sua representação estruturada. A partir disso o programa interpreta essa entrada de texto da Figura 5 e armazena os dados como um grafo de conexões.

As Figuras 7 e 8 são a representação da interface gráfica do sistema, nestas figuras é possível observar de maneira mais legível as conexões existentes no MPSoc e quanto de ocupação tal canal de comunicação possui.

Entrada

```
0 1 0 11 0 50
1 2 0 46 0 48
2 3 0 33 0 24
3 4 0 16 0 49
4 5 0 26 0 56
5 6 0 29 0 15
6 7 0 42 0 24
7 8 0 13 0 31
8 9 0 26 0 15
9 10 0 47 0 42
```

Figura 5. Dados de entrada do Cenário 1.

Estrutura

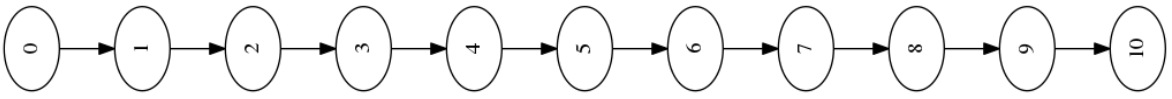

Figura 6. Grafo das tarefas do cenário 1.

Nas Figuras 7 e 8 é possível identificar o fluxo da comunicação e o quanto os canais estão sobrecarregados. Nota-se na Figura 7, First Free, que existem vários canais com a cor amarela e laranja. Bem como, na Figura 8, Path Load, há uma nova maneira de mapeamento das tarefas e nota-se que as percentagens dos canais de comunicação reduziu.

Estar observações à primeira vista podem não ser claras, o que pode ser melhor visto nos gráficos gerados, onde há a comparação da média da ocupação das estratégias de mapeamento utilizadas. Tais gráficos serão explicados a seguir.

First Free

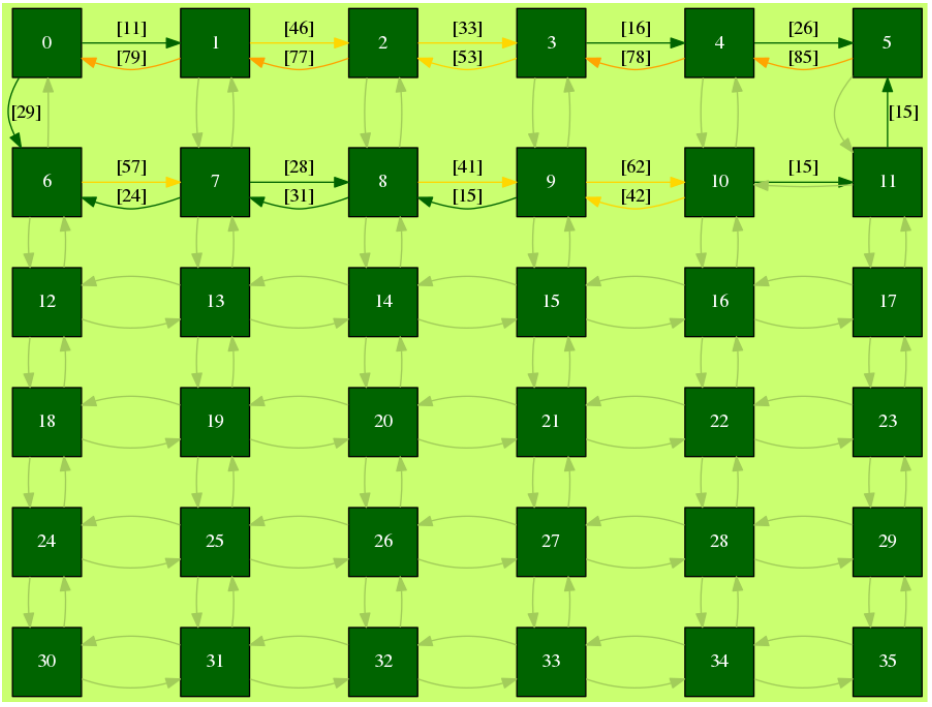

Figura 7. Representação do MPSoc do cenário 1(First-Free).

Path Load

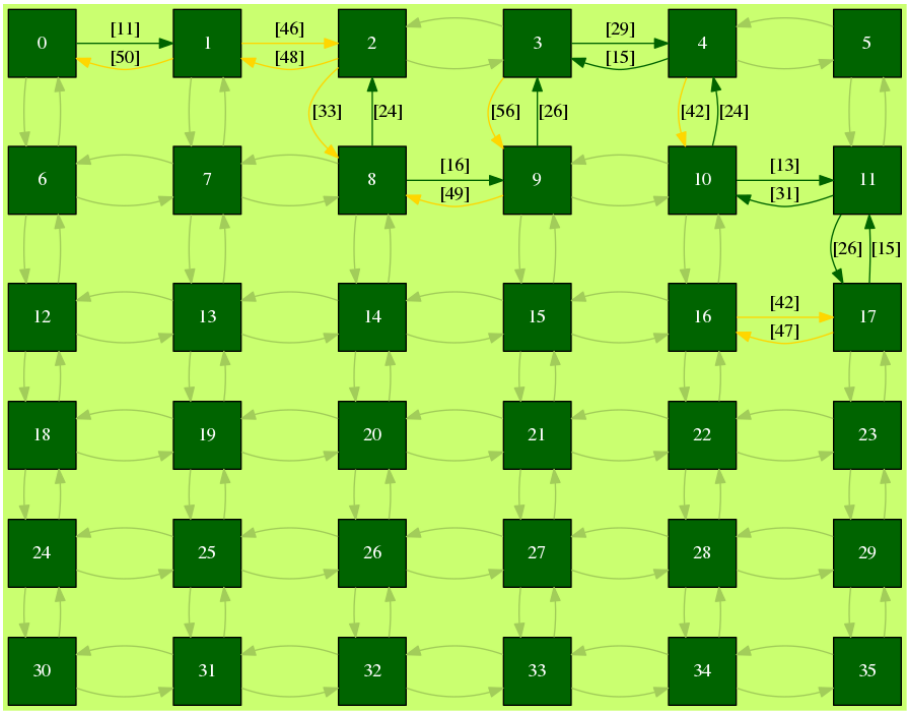

Figura 8. Representação do MPSoc do cenário 1(Path Load).

É possível observar nas Figuras 9 e 10 o comportamento das heurísticas utilizadas. Os gráficos expressos por estas figuras apresentam no eixo X cada uma das conexões dadas por **(Origem, Destino)** na parte superior de cada coluna, e no eixo Y a porcentagem ocupada no canal. Como já foi dito, os canais sobrecarregados possuem percentagem igual ou superior a 100%.

Ao analisar ambas as figuras, observa-se que a percentagem de ocupação dos canais varia, na heurística First Free a média de percentagem ficou próximo à 25%, enquanto a média da heurística Path Load ficou abaixo de 20%. Também é possível identificar que o canal mais sobrecarregado na heurística First Free possui percentagem de ocupação entre 80% e 90%, enquanto na heurística Path Load esse valor fica entre 50% e 60%.

Esta metodologia de análise foi aplicada em cada um dos 5 cenários de teste e a conclusão geral obtida sobre todas estas perspectivas encontra-se na seção 4. A seguir apresentam-se os resultados dos demais cenários de teste.

First Free - Gráfico

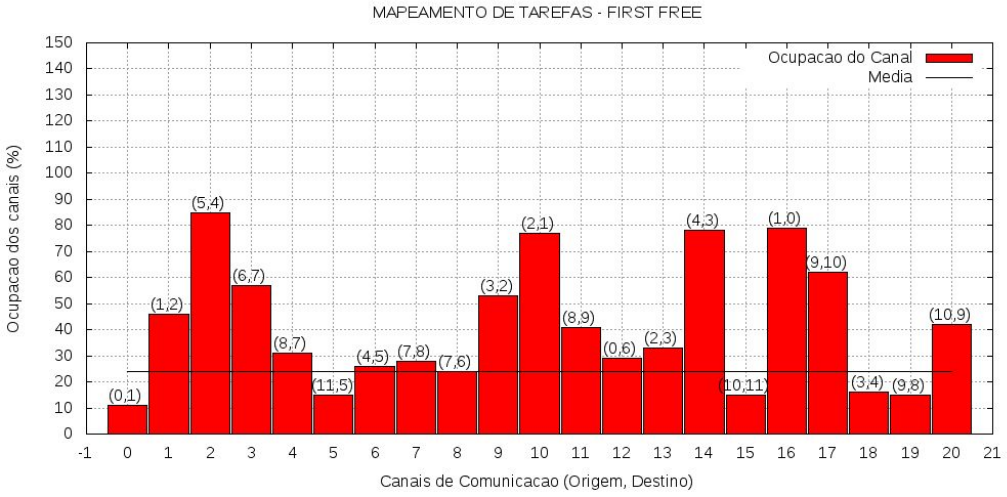

Figura 9. Gráfico de ocupação dos canais do cenário 1(First-Free).

Path Load - Gráfico

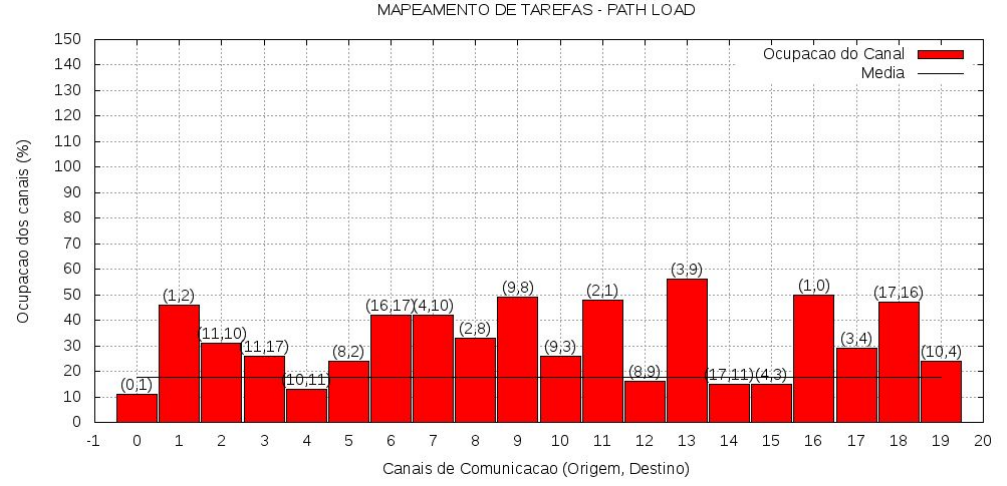

Figura 10. Gráfico de ocupação dos canais do cenário 1(Path Load).

3.2 Cenário de Teste 2 - Pipeline

No cenário 2 também temos uma representação de pipeline, demonstrada pelos dados e pelo formato distinguível do grafo da Figura 9.

Entrada

```
0 1 0 42 0 76
1 2 0 39 0 80
2 3 0 48 0 90
3 4 0 87 0 44
4 5 0 69 0 43
5 6 0 70 0 83
6 7 0 60 0 51
7 8 0 27 0 34
8 9 0 50 0 88
9 10 0 76 0 83
10 11 0 32 0 19
11 12 0 71 0 66
12 13 0 77 0 61
13 14 0 35 0 46
14 15 0 73 0 68
```

Figura 11. Dados de entrada do Cenário 2.

Estrutura

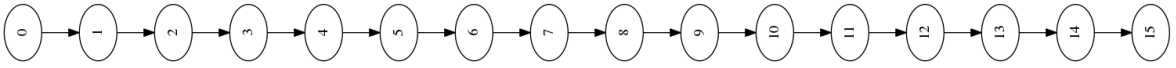

Figura 9. Grafo das tarefas do cenário 2.

First Free

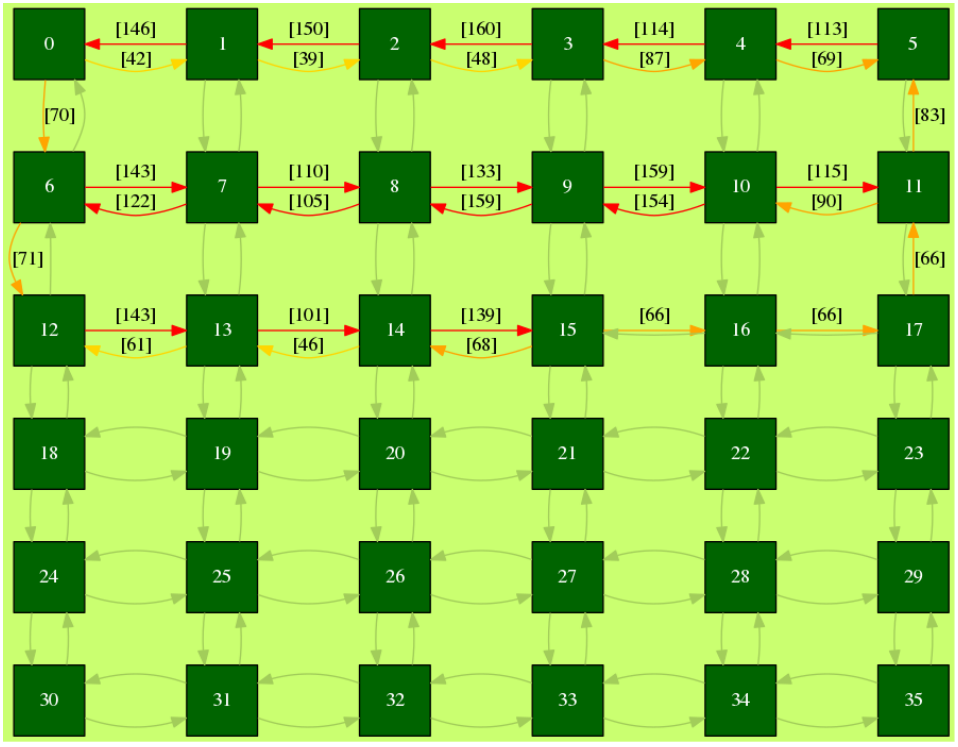

Figura 12. Representação do MPSoc do cenário 2(First-Free).

Path Load

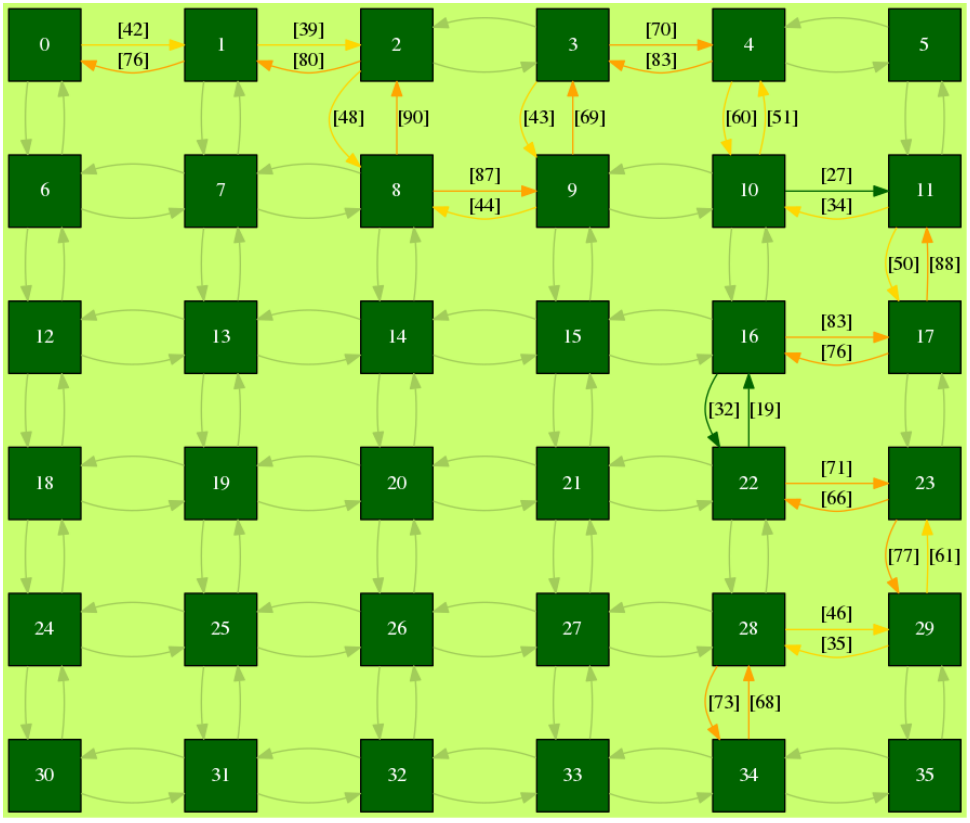

Figura 13. Representação do MPSoc do cenário 2(Path Load).

First Free - Gráfico

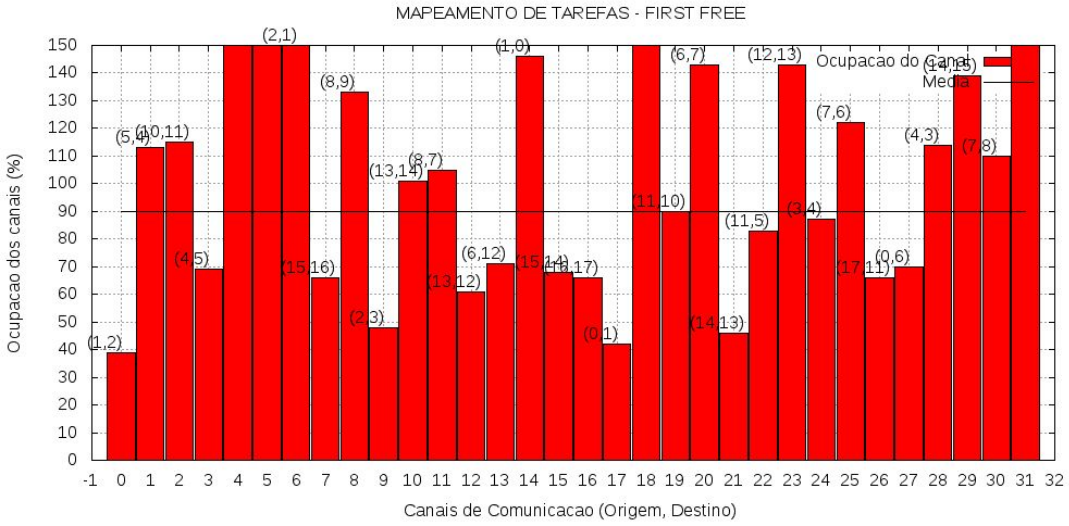

Figura 14. Gráfico de ocupação dos canais do cenário 2(First-Free).

Path Load - Gráfico

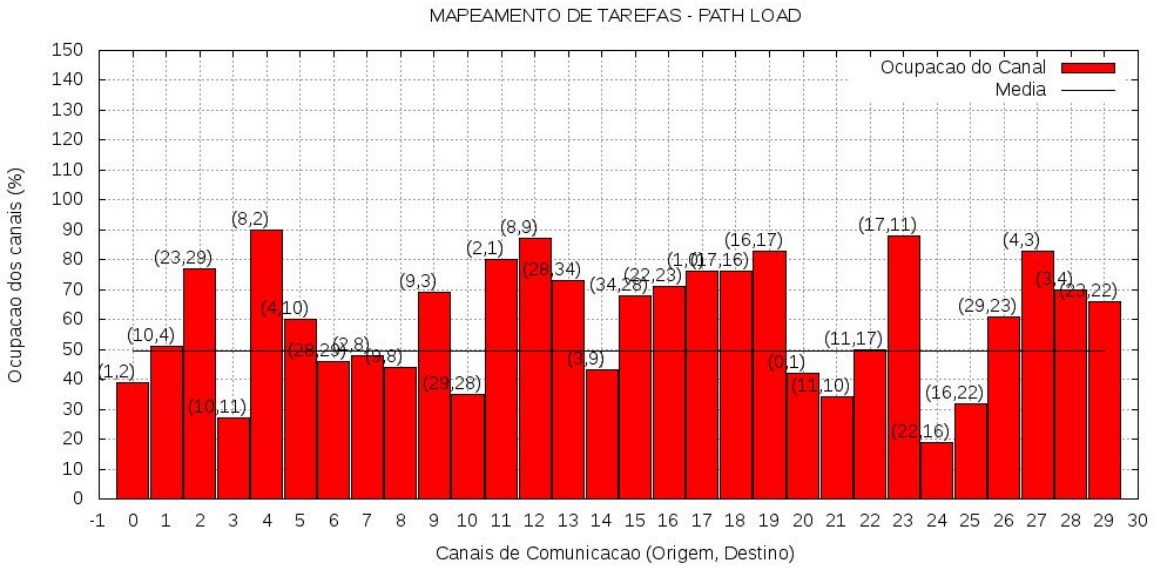

Figura 15. Gráfico de ocupação dos canais do cenário 2(Path Load).

3.3 Cenário de Teste 3 - Árvore

O cenário 3 e os seguintes são todos compostos por entradas no formato de árvore, onde as tarefas perdem a característica de dependência que foi vista nos cenários anteriores.

Entrada

```
1  0 2 150 22 170 20
2  0 1 340 10 300 15
3  1 3 470 10 300 15
4  1 4 300 15 480 15
5  2 5 380 10 230 10
6  2 6 240 5 140 5
7  3 9 210 15 280 25
8  5 8 190 25 340 20
9  6 7 120 20 360 25
```

Figura 16. Dados de entrada do Cenário 3.

Estrutura

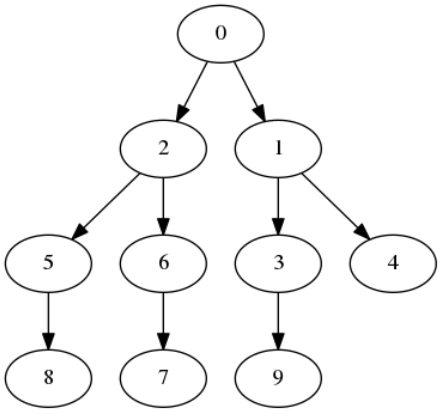

Figura 17. Grafo das tarefas do cenário 3.

First Free

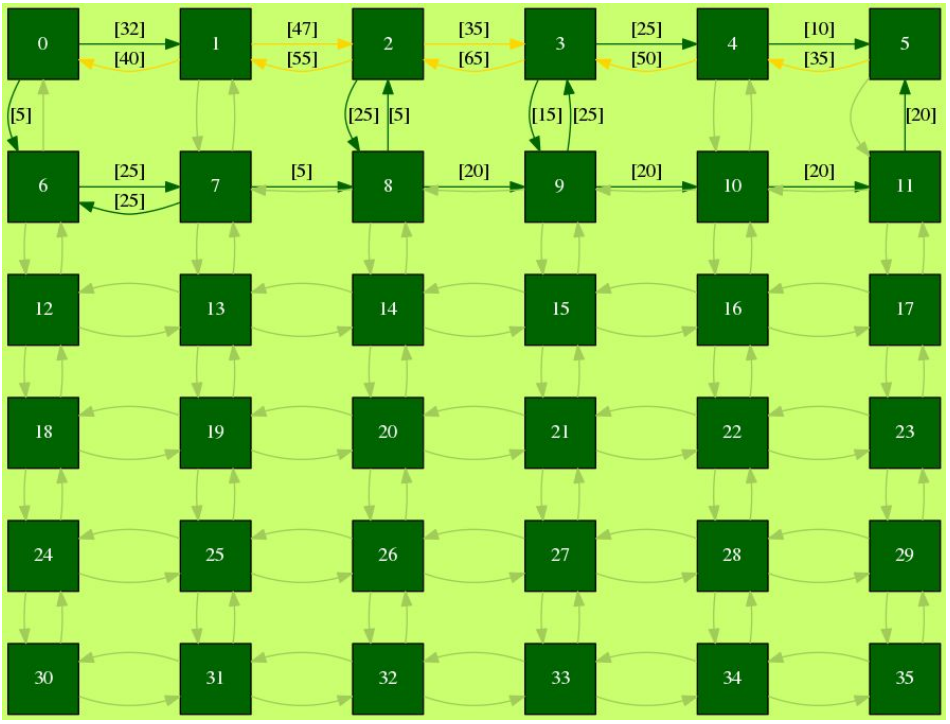

Figura 18. Representação do MPSoc do cenário 3(First-Free).

Path Load

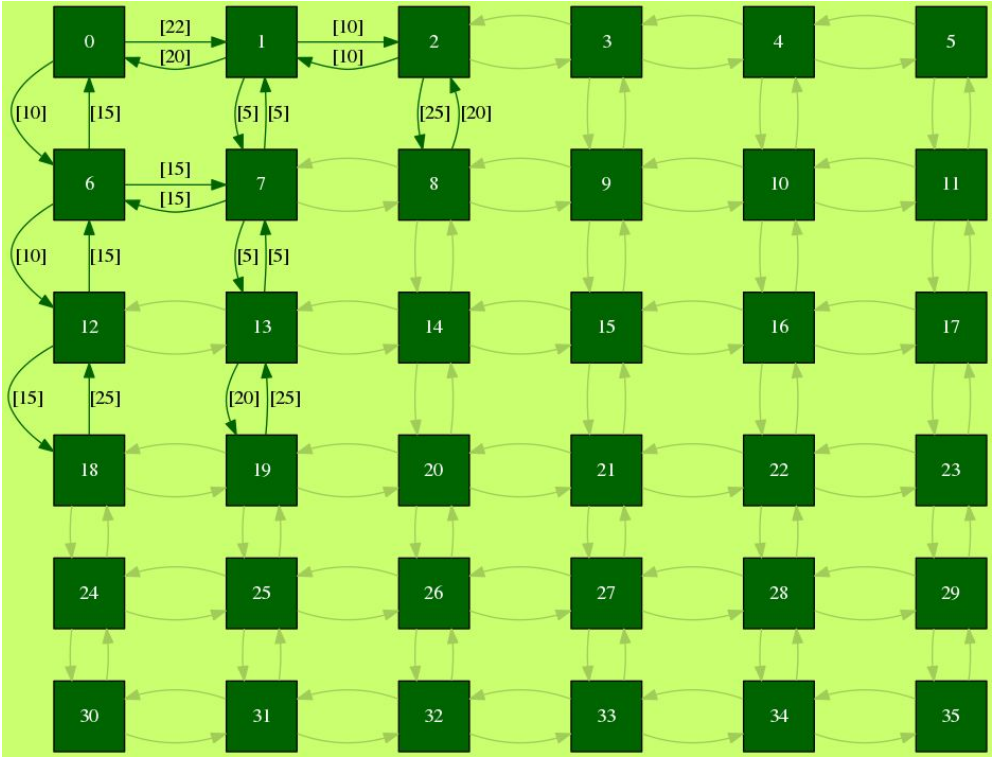

Figura 19. Representação do MPSoc do cenário 3(Path Load).

First Free - Gráfico

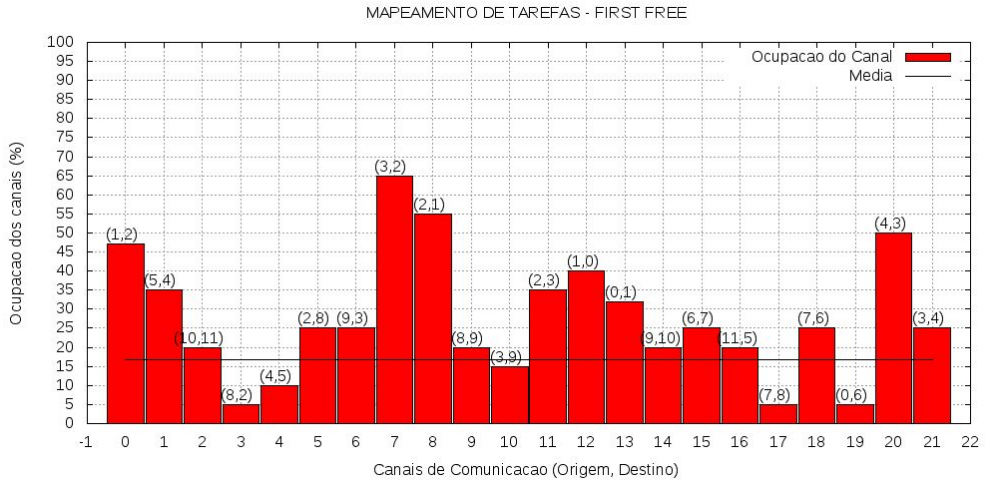

Figura 20. Gráfico de ocupação dos canais do cenário 3(First-Free).

Path Load - Gráfico

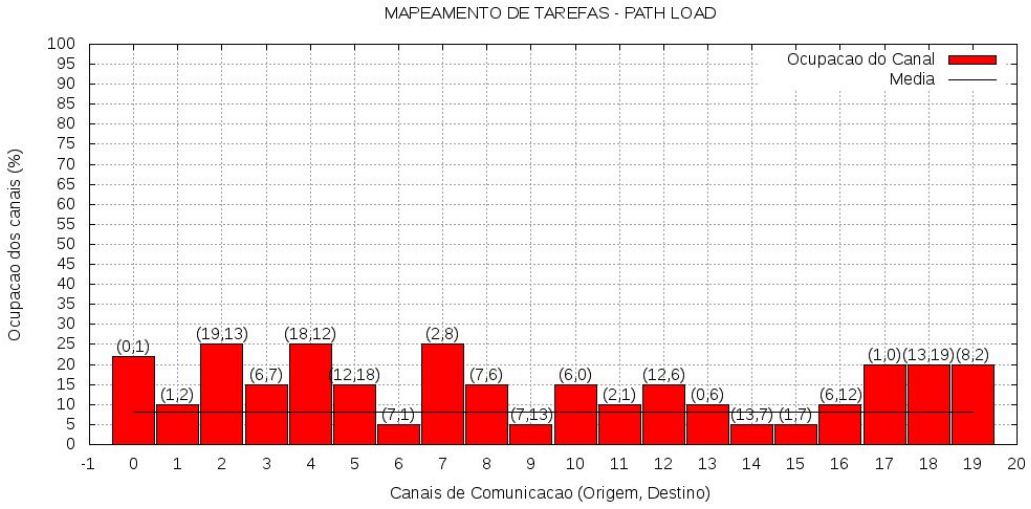

Figura 21. Gráfico de ocupação dos canais do cenário 1(Path Load).

3.4 Cenário de Teste 4 - Árvore

Entrada

```
1  0 1 0 26 0 22
2  0 2 0 26 0 19
3  1 3 0 24 0 25
4  1 4 0 24 0 7
5  2 5 0 26 0 15
6  2 6 0 27 0 20
7  3 7 0 29 0 28
8  3 8 0 8 0 19
9  4 9 0 7 0 22
10 4 10 0 26 0 12
```

Figura 22. Dados de entrada do Cenário 4.

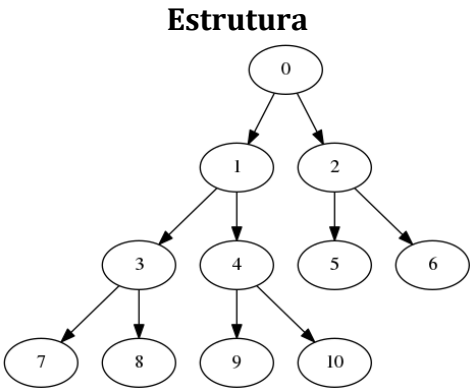

Figura 23. Grafo das tarefas do cenário 4.

Figura 24. Representação do MPSoc do cenário 4(First-Free).

Path Load

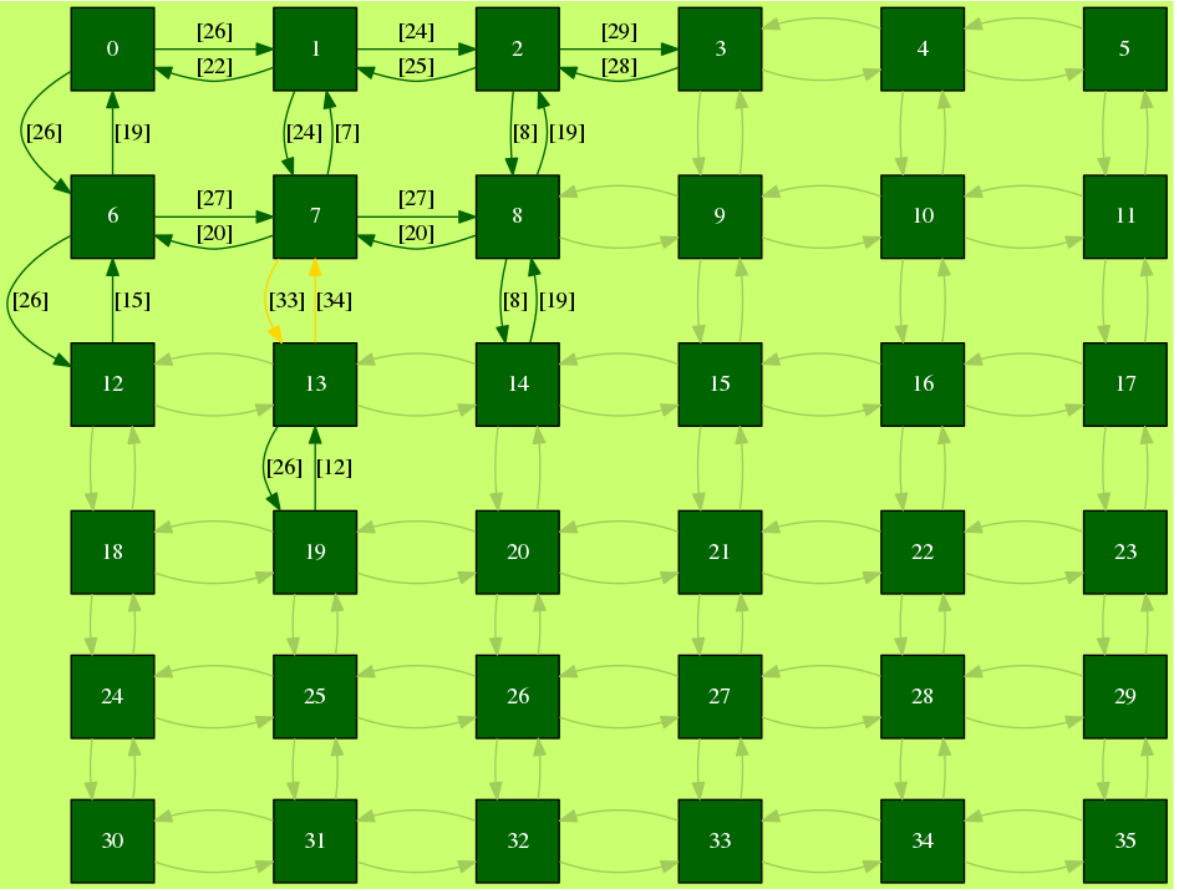

Figura 25. Representação do MPSoc do cenário 4(Path Load).

First Free - Gráfico

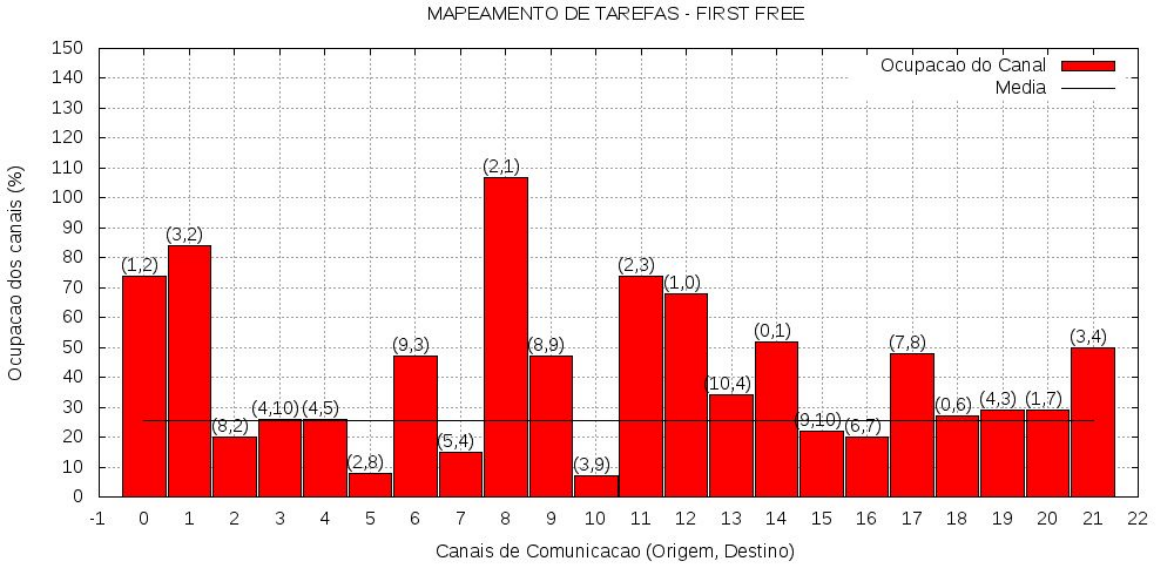

Figura 26. Gráfico de ocupação dos canais do cenário 4(First-Free).

Path Load - Gráfico

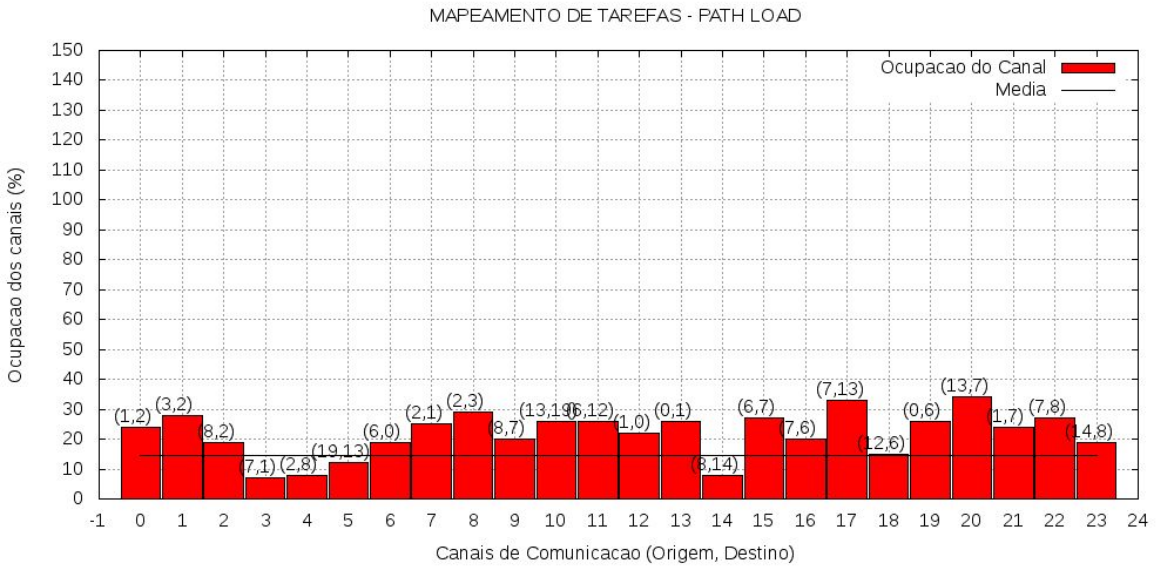

Figura 27. Gráfico de ocupação dos canais do cenário 1(Path Load).

3.5 Cenário de Teste 5 - Árvore

Entrada

```
 1  0 1 0 37 0 54
 2  0 2 0 38 0 15
 3  1 3 0 31 0 49
 4  1 4 0 11 0 41
 5  2 5 0 43 0 33
 6  2 6 0 21 0 36
 7  3 7 0 47 0 59
 8  3 8 0 11 0 15
 9  4 9 0 13 0 46
10  4 10 0 14 0 51
11  5 11 0 20 0 22
12  5 12 0 47 0 11
13  6 13 0 52 0 46
14  6 14 0 14 0 27
15  7 15 0 37 0 57
16  7 16 0 43 0 43
17  8 17 0 20 0 26
18  8 18 0 48 0 10
19  9 19 0 54 0 24
20  9 20 0 56 0 51
```

Figura 28. Dados de entrada do Cenário 5.

Estrutura

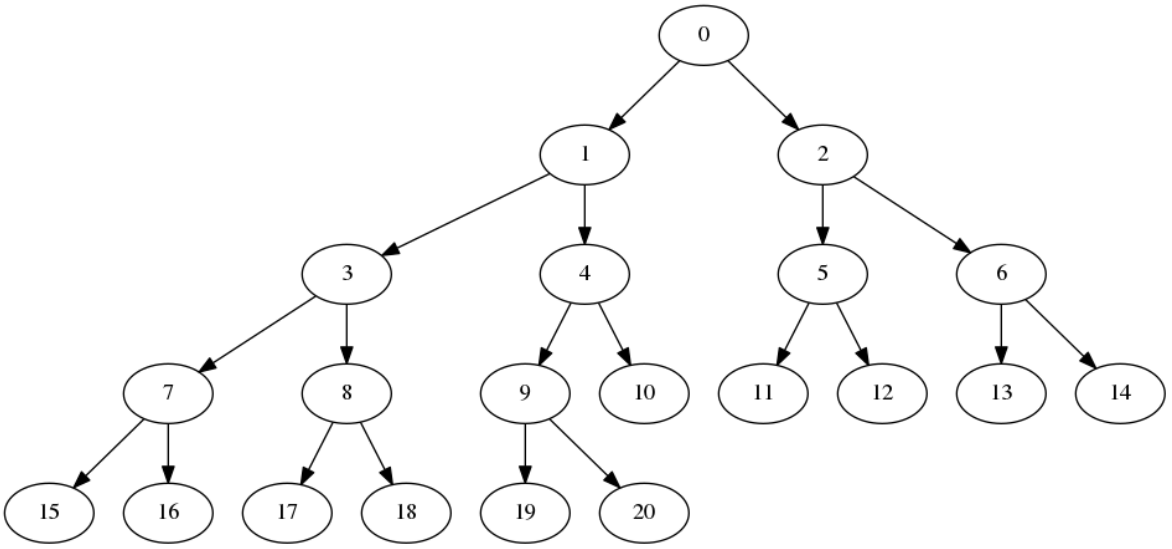

Figura 29. Grafo das tarefas do cenário 5.

First Free

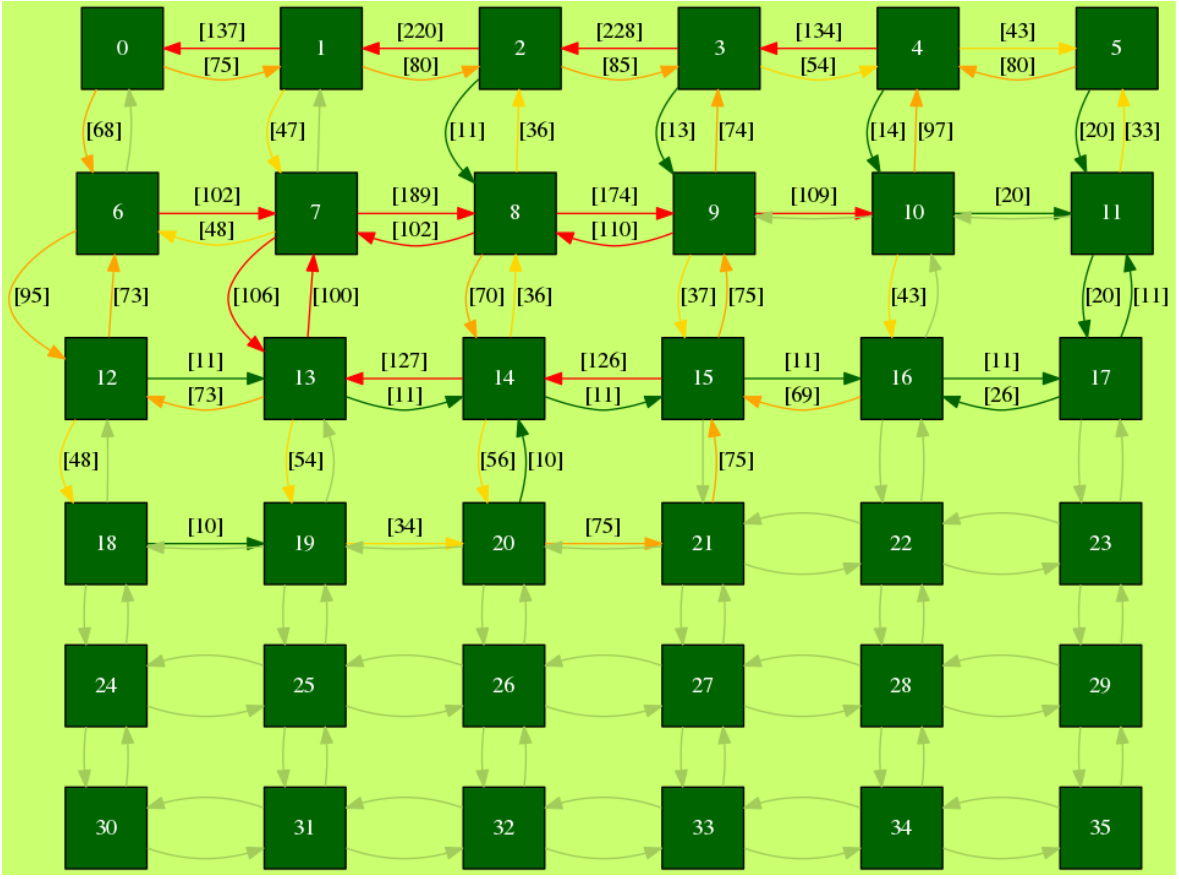

Figura 30. Representação do MPSoc do cenário 5(First-Free).

Path Load

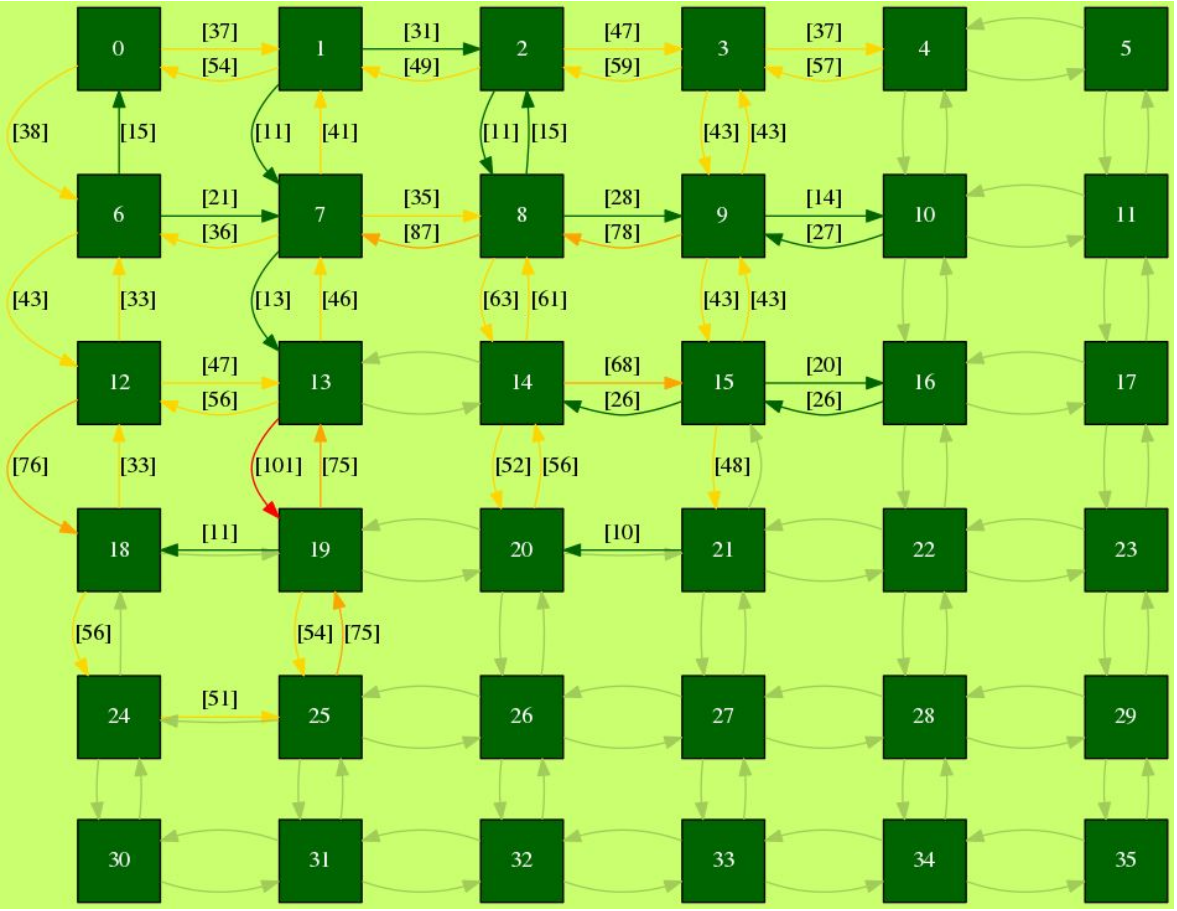

Figura 31. Representação do MPSoc do cenário 5(Path Load).

First Free - Gráfico

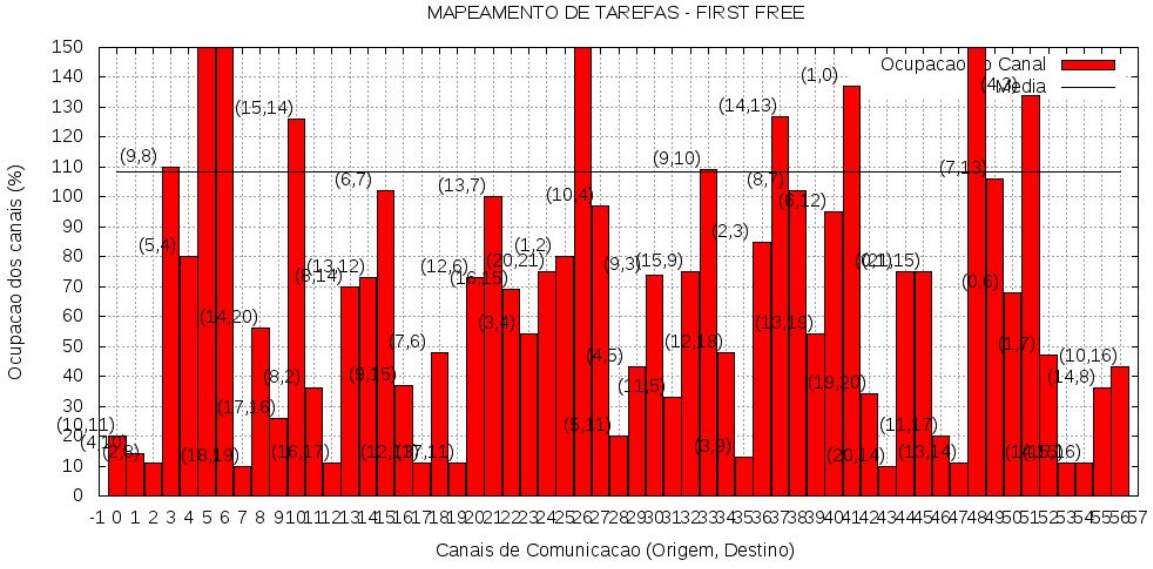

Figura 32. Gráfico de ocupação dos canais do cenário 5(First-Free).

Path Load - Gráfico

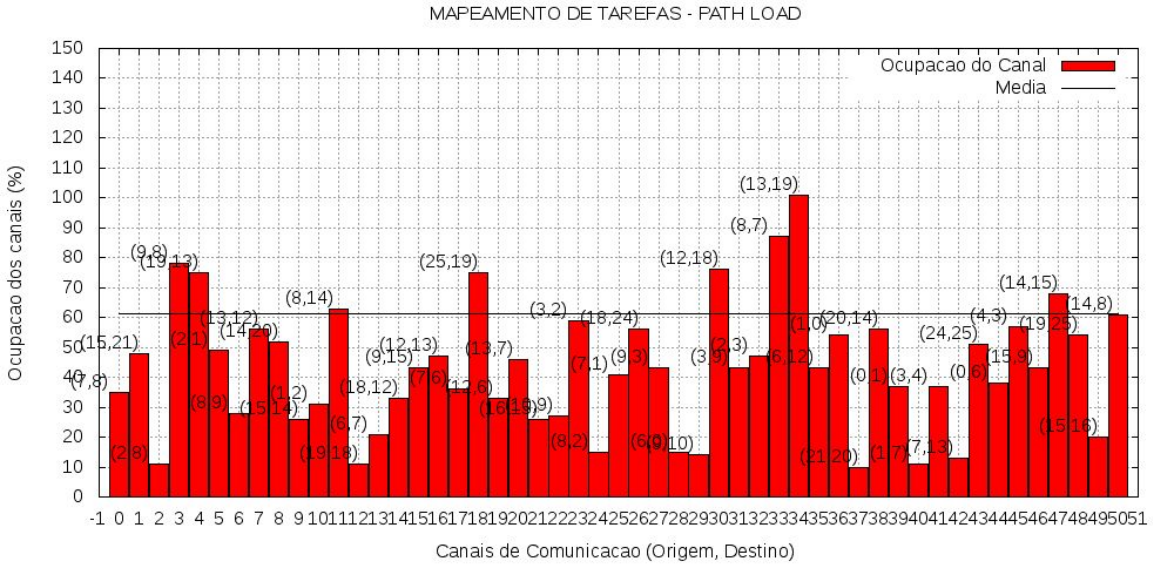

Figura 33. Gráfico de ocupação dos canais do cenário 1(Path Load).

3.5 Discussões

De modo geral, em todas as simulações, os resultados da ocupação dos canais foi favorável à heurística do Path Load. A ocupação dos canais nos cenários de simulação utilizando First-Free várias vezes excedem 100% da capacidade do canal(ligações em vermelho). Em alguns casos como na figura 30, os pesos excederam o dobro da capacidade do canal, o quê em prática, causaria lentidão na execução das tarefas.

Diferente dos cenários que utilizaram First-Free, somente no último caso de testes a heurística apresentou um esquema de mapeamento que deixou canais acima do seu limite de capacidade.

A distribuição da carga entre os canais também foi mais uniforme nos cenários de teste utilizando a heurística Path Load. Nos casos onde foi empregado o First-Free existem picos mais noticiáveis, que demonstram que a distribuição da carga foi mal realizada, deixando alguns canais sobrecarregados enquanto outros caminhos ainda poderiam ser utilizados de modo a melhorar o desempenho das tarefas.

A heurística First-Free não apresentou bom desempenho nem mesmo nos primeiros casos de teste, que são mais simples em termos de conexões. Isso demonstra a inviabilidade do método. Em contraste, o Path Load, ainda que seja um algoritmo simples, apresentou um desempenho excelente.

4. Conclusão

O trabalho proposto proporcionou maior proximidade com a linguagem Python e maior compreensão sobre o funcionamento de cada uma das heurísticas, proporcionando assim, maior conhecimento sobre elas e sobre o funcionamento do mapeamento de tarefas em um MPSoc.

Este trabalho possibilitou aos alunos a obtenção de conhecimentos tecnológicos, transmitiu ao grupo maior experiência e capacidade para lidar com outros desafios, principalmente, na questão do trabalho em grupo. Ao mesmo tempo em que o faz pensar sobre as possibilidades que o cercam, ao mesmo tempo em que se depara com um trabalho deste e adquire conhecimento para poder ir, quem sabe, muito além de sala de aula para buscar conhecimento e aprender. Através do incentivo proporcionado na cadeira e em seus trabalhos, o aluno se torna capaz de buscar por si só, pois o conteúdo é dinâmico e estimula o aprendizado.

5. Referências

[1] CARVALHO, Ewerson; CALAZANS, Ney; MORAES, Fernando. Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs. In: **Rapid System Prototyping, 2007. RSP 2007. 18th IEEE/IFIP International Workshop on**. IEEE, 2007. p. 34-40.

Anexos

1) mapeamento.py

Em virtude do tamanho do código fonte, o mesmo não será exibido nos apêndices.
O código encontra-se na pasta junto com este documento pdf.

2) FirstFree.gnu

```
set terminal postscript eps enhanced color
set boxwidth 1.9 relative
set style fill solid border -1
set style data histogram
set style histogram cluster gap 1.5

set ytics 10
set xtics 1
set terminal png size 1000,500

set grid ytics
set grid xtics
#set nokey

set yrange [0:150]
set xrange [-1:]

set title "MAPEAMENTO DE TAREFAS - FIRST FREE"

set ylabel "Ocupacao dos canais (%)"
set xlabel "Canais de Comunicacao (Origem, Destino)"

plot 'FirstFree.dat' using 2 linecolor rgb "red" title "Ocupacao do Canal"
replot 'FirstFree.dat' using 5 linecolor rgb "black" with lines title "Media"
replot 'FirstFree.dat' using 1:2:(sprintf("(%d,%d)", $3, $4)) with labels offset -4,0.5 notitle

set terminal png
set output 'FirstFreeGrafico.png'
replot
```

3) PathLoad.gnu

```
set terminal postscript eps enhanced color
set boxwidth 1.9 relative
```

```
set style fill solid border -1
set style data histogram
set style histogram cluster gap 1.5

set ytics 10
set xtics 1
set terminal png size 1000,500

set grid ytics
set grid xtics
#set nokey

set yrange [0:150]
set xrange [-1:]

set title "MAPEAMENTO DE TAREFAS - PATH LOAD"

set ylabel "Ocupacao dos canais (%)"
set xlabel "Canais de Comunicacao (Origem, Destino)"

plot 'PathLoad.dat' using 2 linecolor rgb "red" title "Ocupacao do Canal"
replot 'PathLoad.dat' using 5 linecolor rgb "black" with lines title "Media"
replot 'PathLoad.dat' using 1:2:(sprintf("(%d,%d)", $3, $4)) with labels offset -4,0.5 notitle

set terminal png
set output 'PathLoadGrafico.png'
replot
```

4) geradorDeEntrada.py

```
#!usr/bin/env/python3
# -*- coding: utf-8 -*-
from random import randint
import argparse, sys

parser = argparse.ArgumentParser(description='Gerador de casos')
parser.add_argument('nome_entrada', type=str, help='Nome dos arquivos de teste')
parser.add_argument('casos',type=int, help='Número de casos de teste')
parser.add_argument('linhas',type=int, help='Número de linhas')
parser.add_argument('simulacao', type=str, help='Qual o tipo de simulacao (p = pipeline,  a = arvore, g = grafo)')

args = parser.parse_args()

if len(sys.argv) < 5:
   parser.print_help()
   sys.exit(1)

num = 1
node = 0
for i in range(args.casos):
   with open (args.nome_entrada+str(num)+'.txt', 'w')  as f:
      #Gera os arquivos de Pipeline
      if (args.simulacao == 'p'):
         for j in range(args.linhas * (i+1)):
            f.write("{} {} 0 {} 0 {}\n".format(node, node+1, randint((i+1)*5,(i+1)*30),randint((i+1)*5,(i+1)*30)))
            node += 1
      #Gera os arquivos de Arvore
      elif (args.simulacao == 'a'):
         control = [] #armazena as conexoes para evitar repeticao
         for j in range(args.linhas * (i+1)):
            f.write("{} {} 0 {} 0 {}\n".format(j, (2*j)+1, randint((i+1)*5,(i+1)*30),randint((i+1)*5,(i+1)*30)))
            f.write("{} {} 0 {} 0 {}\n".format(j, (2*j)+2, randint((i+1)*5,(i+1)*30),randint((i+1)*5,(i+1)*30)))
            node += 1
      #Gera os arquivos de Grafo
      elif (args.simulacao == 'g'):
         control = [] #armazena as conexoes para evitar repeticao
         for j in range(args.linhas * (i+1)): #Para cada um dos elementos
            for k in range(randint(1, 3)): #Sorteia o numero de ligacoes que ele tem
```

```
        while True:
            valA = randint(1,args.linhas)
            if (j != valA) and ((j,valA) not in control):
                        control.append(((j,valA)))
                        break

        f.write("{} {} 0 {} 0 {}\n".format(j, valA, randint((i+1)*5,(i+1)*30),randint((i+1)*5,(i+1)*30)))
        node += 1
num += 1
node = 0
```