6

Referências bibliográficas

[1] ALDA, W.; DZWINEL, W.; KITOWSKI, J.; MOSCINSKI, J. ; YUEN, D. A.. **Penetration mechanics via molecular dynamics**. Research Report UMSI 93/58, University of Minnesota Supercomputing Institute, 1993.

[2] ALTILAR; TURGAY, D. ; P., Y.. **Optimal scheduling algorithms for communication constrained parallel processing**. Em: Monien, B.; Feldmann, R., editores, 8TH INTERNATIONAL EURO-PAR CONFERENCE, volume 2400 de **Lecture Notes in Computer Science**, p. 197–206, Paderborn, 2002. Springer-Verlag.

[3] BEAUMONT, O.; LEGRAND, A. ; ROBERT, Y.. **Optimal algorithms for scheduling divisible workloads on heterogeneous systems**. Research Report 2002-36, École Normale Supérieure de Lyon, 2002.

[4] BEAUMONT, O.; LEGRAND, A. ; ROBERT, Y.. **Optimal algorithms for scheduling divisible workloads on heterogeneous systems**. Em: 12TH HETEROGENEOUS COMPUTING WORKSHOP. IEEE Computer Society Press, 2003.

[5] BHARADWAJ, V.; GHOSE, D.; MANI, V. ; ROBERTAZZI, T. G.. **Scheduling Divisible Loads in Parallel and Distributes Systems**. IEEE Computer Society Press, 1996.

[6] BLANC, J. Y.; TRYSTRAM, D.. **Implementation of parallel numerical routines using broadcast communication schemes**. Em: JOINT INTERNATIONAL CONFERENCE ON VECTOR AND PARALLEL PROCESSING, volume 457 de **Lecture Notes in Computer Science**, p. 467–478, Zurique, 1990. Springer-Verlag.

[7] BLAZEWICZ, J.; DROZDOWSKI, M.. **Scheduling divisible jobs on hypercubes**. Parallel Computing, 21:1945–1956, 1995.

[8] BLAZEWICZ, J.; DROZDOWSKI, M.. **Distributed processing of divisible jobs with communication startup costs**. Discrete Applied Mathematics, 76:21–41, 1997.

[9] CAMP, W. J.; PLIMPTON, S. J.; HENDRICKSON, B. A. ; LELAND, R. W.. **Massively parallel methods for engineering and science problems**. Communications of the ACM, 37:30–41, 1994.

[10] CASANOVA, H.. **Simgrid: A toolkit for the simulation of application scheduling**. Em: IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, p. 430–437, Brisbane, 2001.

[11] CHENG, Y. C.; ROBERTAZZI, T. G.. **Distributed computation with communication delay**. IEEE Transactions on Aerospace and Electronic Systems, 24:700–712, 1988.

[12] CORMEN, T.; LEISERSON, C. ; RIVEST, R.. **Introduction to Algorithms**. MIT Press, 2001.

[13] DROZDOWSKI, M.; WOLNIEWICZ, P.. **Experiments with scheduling divisible tasks in clusters of workstations**. Em: Bode, A.; II, T. L.; Karl, W. ; Wismüller, R., editores, 6TH INTERNATIONAL EURO-PAR CONFERENCE, volume 1900 de **Lecture Notes in Computer Science**, p. 311–319, Munique, 2000. Springer-Verlag.

[14] DROZDOWSKI, M.. **Selected problems of scheduling tasks in multiprocessor computing systems**. Tese de Doutorado, Poznan University of Technology, Poznań, 1997.

[15] FOSTER, I.; KESSELMAN, C.; NICK, J. ; TUECKE, S.. **Grid services for distributed system integration**. Computer, 35:37–46, 2002.

[16] FOSTER, I.; KESSELMAN, C.. **The Grid: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann, 2004.

[17] GANTT, H. L.. **Wages and Profits**. The Engineering Magazine Company, 1910.

[18] LEGRAND, A.; MARCHAL, L. ; CASANOVA, H.. **Scheduling distributed applications: the simgrid simulation framework**. Em: 3RD INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, p. 138–145, Tóquio, 2003.

[19] RESENDE, M. G. C.; RIBEIRO, C.. **Greedy randomized adaptative search procedures**. Em: Glover, F.; Kochenberger, G., editores, HANDBOOK OF METAHEURISTICS, p. 219–249. Kluwer Academic Publishers, 2003.

[20] WOLSEY, L. A.. **Integer Programming**. Wiley-Interscience, 1998.

[21] WOLNIEWICZ, P.. **Divisible Job Scheduling in Systems with Limited Memory**. Tese de Doutorado, Poznan University of Technology, Poznań, 2003.

[22] YANG, Y.; CASANOVA, H.. **RUMR: Robust scheduling for divisible workloads**. Em: 12TH IEEE SYMPOSIUM ON HIGH PERFORMANCE AND DISTRIBUTED COMPUTING, p. 114–125, Seattle, 2003.

[23] YANG, Y.; CASANOVA, H.. **UMR: A multi-round algorithm for scheduling divisible workloads**. Em: PROCEEDINGS OF THE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, p. 24b, Nice, 2003.

A

Visualizador de soluções

Com o objetivo de obter-se uma melhor visualização das soluções obtidas, bem como uma fácil análise e apresentação dos resultados, foi desenvolvido um sistema de visualização. Neste sistema pode-se escolher um arquivo de entrada contendo a descrição do sistema computacional, bem como o tamanho da tarefa divisível e as decisões de escalonamento tomadas. Como pode ser visto na Figura A.1, o sistema dispõe de várias opções de visualização. Seu recurso mais importante é a criação de diagramas temporais estilo Gantt [17], diagramas de barras que apresentam o progresso temporal de utilização dos recursos do sistema.

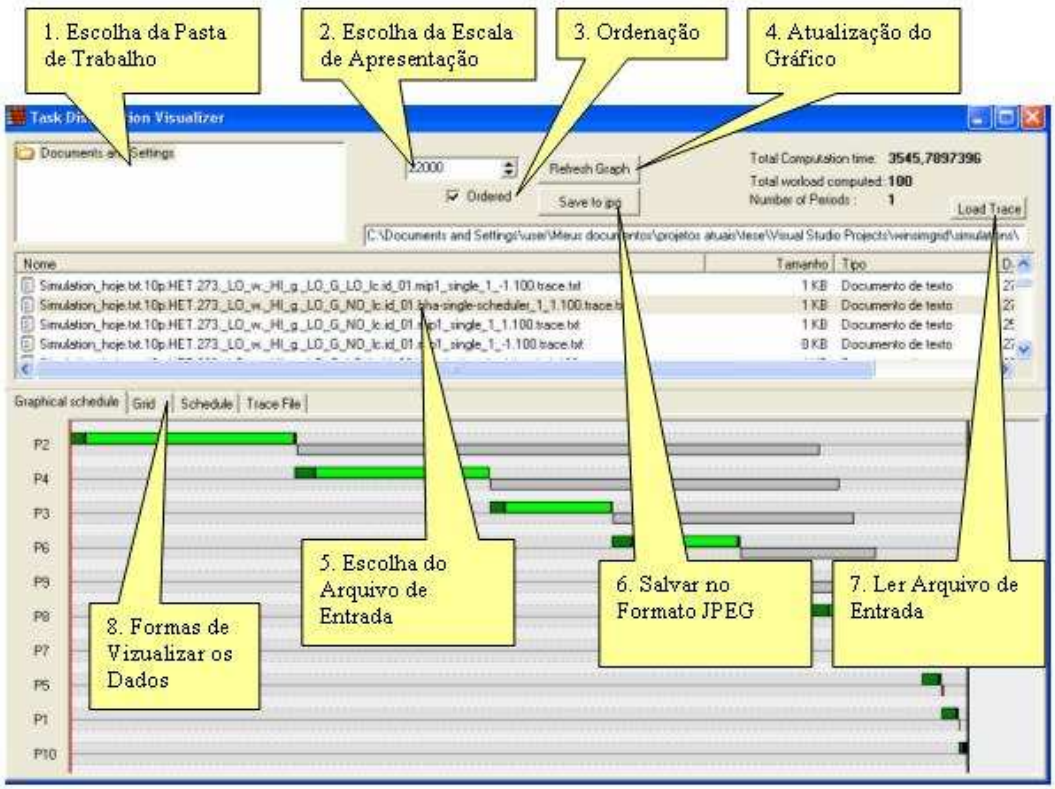

Figura A.1: Imagem da interface do visualizador

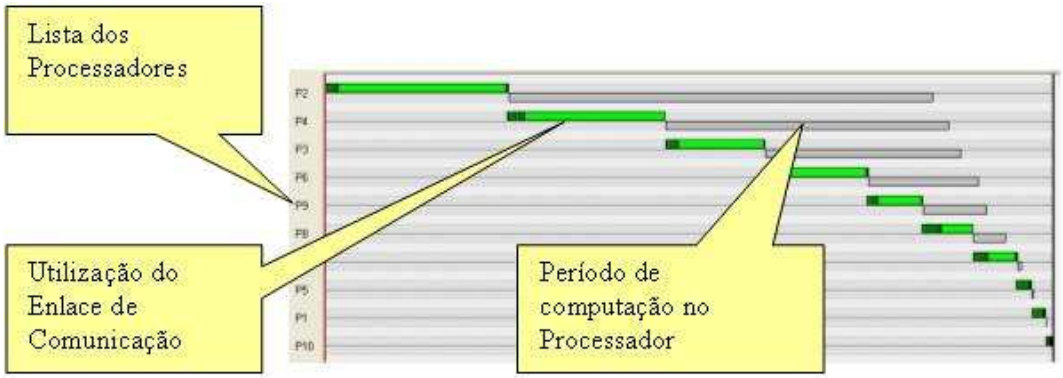

Figura A.2: Escalonamento não ótimo em apenas um período

Como pode ser visto na Figura A.2, o diagrama é composto por n linhas, cada uma representando um processador. Bem à esquerda é apresentado o rótulo do processador cuja utilização está sendo representada naquela linha. No diagrama existem três cores de barras horizontais: preto, cinza escuro e cinza claro.

Dada uma linha com rótulo P_i, a barra preta nesta linha representa o tempo gasto com a latência de comunicação g_i, a barra cinza escuro representa o tempo variável de comunicação $G_i\alpha_i$ e a barra cinza claro, o tempo variável de processamento $w_i\alpha_i$. Note que as barras cinza escuro que representam o tempo de comunicação nunca ocorrem concorrentemente, uma vez que o processador mestre não pode enviar dados de forma concorrente.

Considere o exemplo apresentado na Figura A.2, onde não houve uma distribuição ótima de carga e os processadores terminaram o processamento em momentos diferentes. Neste cenário, o escalonador optou pela utilização de apenas um período, ou seja, enviou as cargas apenas uma vez para cada processador. Pelo gráfico é possível verificar que o primeiro processador a receber tarefas é o processador P_2 (sendo que primeiro os dados são enviados -barra preta e cinza escuro- e depois são processados -barra cinza claro). O processador mestre envia dados para P_4 após o envio das tarefas para P_2, e assim por diante.

Ao contrário da Figura A.2, encontra-se em A.3 um exemplo de escalonamento ótimo em período único para outra instância. Note que neste cenário nem todos os processadores foram utilizados, o que causa a omissão dos rótulos de processadores que não participam da computação. Neste exemplo todos os processadores terminam a computação no mesmo instante de tempo, conforme pode ser visto pelas barras cinza claro.

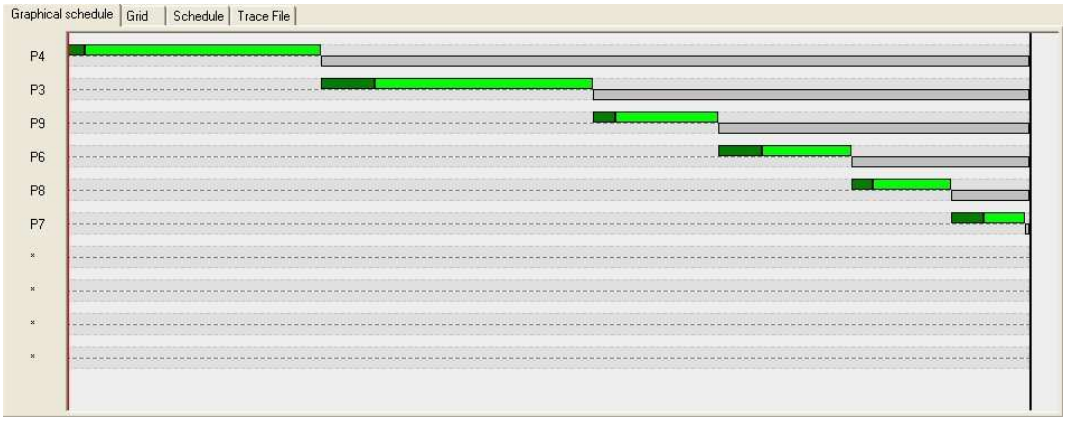

Figura A.3: Escalonamento ótimo em apenas um período

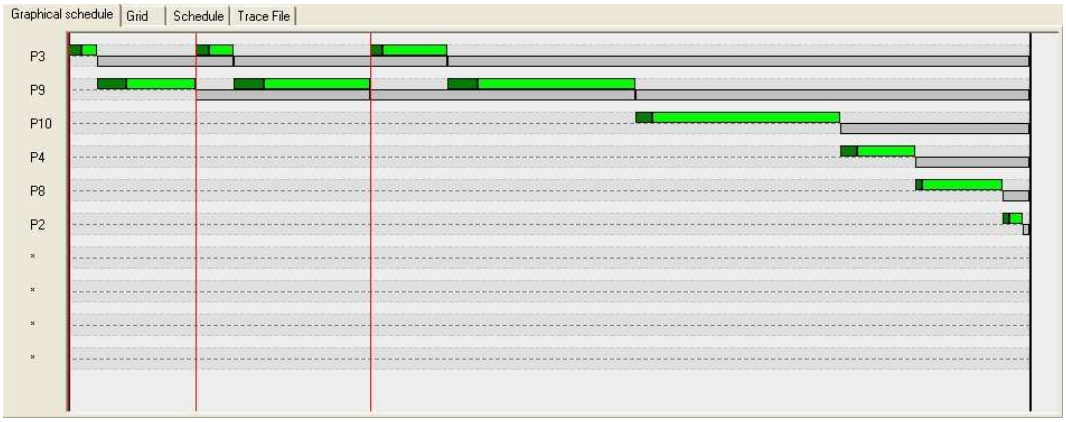

Figura A.4: Escalonamento em três períodos

Diferentemente das imagens anteriores, na Figura A.4 é apresentado o escalonamento de tarefas em múltiplos períodos. Neste exemplo, foram utilizados três períodos de comunicação. No primeiro e segundo períodos apenas os processadores P_3 e P_9 receberam dados, enquanto que no último foram utilizados ainda mais quatro computadores, P_{10}, P_4, P_8 e P_2.

Note que existem linhas verticais delimitando os períodos de envio de dados, bem como uma linha grossa preta marcando o momento que todo o processamento foi terminado.

A.1

Formato do arquivo de entrada

O arquivo de entrada deve ser um arquivo no formato texto contendo a descrição do sistema computacional, da tarefa divisível que foi escalonada, bem

como das decisões de escalonamento. Para ser um arquivo de entrada válido, ele deve seguir o seguinte formato:

Simulation
<tamanho da tarefa divisível> <número de processadores>
<id1 do processador> <w_{id1}> <NU> <NU> <G_{id1}><g_{id1}>
<id2 do processador> <w_{id2}> <NU> <NU> <G_{id2}><g_{id2}>
<id3 do processador> <w_{id3}> <NU> <NU> <G_{id3}><g_{id3}>
. . .
<idn do processador> <w_{idn}> <NU> <NU> <G_{idn}><g_{idn}>
0 0 <momento de chamada do escalonador>
<id do 1º processador a receber dados> <qta. de dados>
<id do 2º processador a receber dados> <qta. de dados>
. . .
<id do nº processador a receber dados> <qta. de dados>
0 0 <hora da chamada do escalonador>
. . .

Abaixo segue uma descrição mais detalhada do arquivo de entrada:

- Na primeira linha do arquivo de entrada deve aparecer a palavra "Simulation", para identificar que se trata de um arquivo neste formato.
- Na segunda linha aparecem o tamanho da tarefa divisível e o número n de processadores do sistema.
- Após a segunda linha, estão presentes n linhas contendo informações de cada processador e de cada enlace. Existem também campos ainda não utilizados denotados <NU>, que estão reservados para futuros melhoramentos.
- Por fim, devem existir as seguintes linhas para cada rodada (ou período de envio de dados) de escalonamento, até o final do arquivo:
  - Dois números zero ("0 0") seguidos por um número real que representa o momento que o escalonador foi chamado
  - Múltiplas linhas contendo o identificador do processador que deve receber os dados e a quantidade de dados.

Como exemplo de arquivo de entrada, segue abaixo o arquivo que gerou o gráfico Gantt apresentado na Figura A.5. Nele observa-se o escalonamento em dois períodos de uma tarefa de tamanho 100, que resultou num *makespan* de

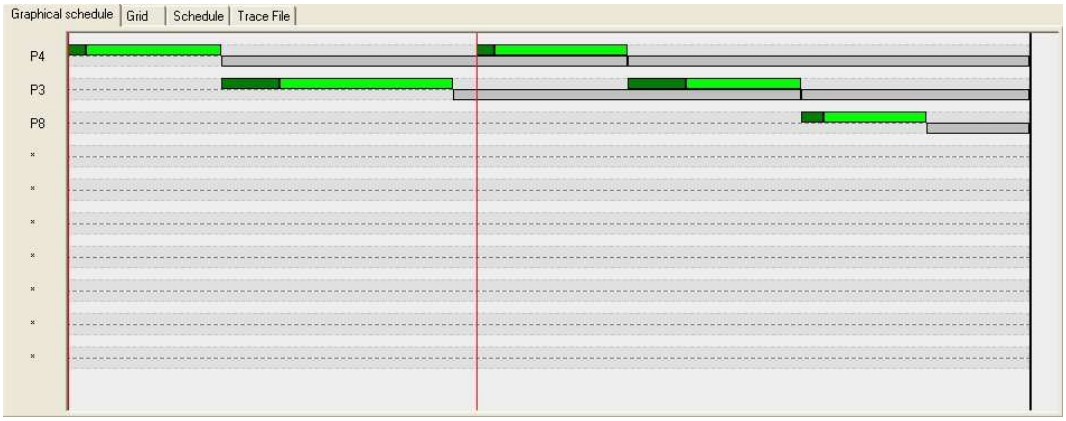

Figura A.5: Escalonamento em dois períodos

1.648,2978723. Salienta-se que esta entrada de dados independe das unidades de medida (tanto de tempo quando de unidade de informação), uma vez que todos os valores expressos devem ser referentes a um mesmo padrão de medidas.

```
Simulation
100,0000000 10
1 50,0000000 0,0000000 0,0000000 60,0000000 60,0000000
2 30,0000000 0,0000000 0,0000000 90,0000000 40,0000000
3 20,0000000 0,0000000 0,0000000 100,0000000 10,0000000
4 30,0000000 0,0000000 0,0000000 30,0000000 10,0000000
5 30,0000000 0,0000000 0,0000000 70,0000000 60,0000000
6 60,0000000 0,0000000 0,0000000 80,0000000 30,0000000
7 10,0000000 0,0000000 0,0000000 60,0000000 90,0000000
8 40,0000000 0,0000000 0,0000000 40,0000000 40,0000000
9 90,0000000 0,0000000 0,0000000 40,0000000 30,0000000
10 80,0000000 0,0000000 0,0000000 60,0000000 100,0000000
0 0 0,0000000
4 23,244681
3 29,783688
0 0 700,2836879
4 22,950355
3 19,617021
8 4,404255
0 0 1648,2978723
```

B

Exemplo de utilização das classes criadas

Como exemplo de utilização de algumas das classes criadas, tem-se o seguinte trecho de código:

```
// declaração dos objetos
scheduler_HeuRet  sched_f;
simulation        s;
solution          sol;
ofstream          fgrid("grid01.txt");
ofstream          ftrace("grid01.trace.txt");

// inclui hosts, informando seus nomes e respectivos wi
s.grid.add_host( host("Host 0",0) );
s.grid.add_host( host("Host 1",2) );
s.grid.add_host( host("Host 2",2) );

// inclui links, cada um com seu nome, gi, Gi, host origem e destino
// (os links são sempre bidirecionais)
s.grid.add_link( link("Link 1", 10, 20, s.grid.Host(0), s.grid.Host(1));
s.grid.add_link( link("Link 2", 10, 20, s.grid.Host(0), s.grid.Host(2));

// inclui uma tarefa divisível de tamanho 1000 (W=1000)
s.add_task( divisible_task("dt1", 1000));

// configura a simulação para gravar um arquivo de trace
// com informações sobre as decisões de escalonamento tomadas
// ( este arquivo será usado como entrada para o
// visualizador de soluções )
s.set_trace_on(ftrace);
```

```
// gera uma solução usando o escalonador sched_f agindo na simulação s
sol        = sched_f.create_solution(s);

// salva toda a topologia do sistema computacional em arquivo
// para que o experimento possa ser reproduzido futuramente
s.grid >> fgrid;
```