Primeiros Passos para Depuração de Aplicações BSP Desenvolvidas com SimGrid

Paulo Anderson Lara[1], **Rodrigo da Rosa Righi**[1]

[1] Programa de Pós-Graduação em Computação Aplicada – Unisinos
São Leopoldo – RS – Brazil

pauloalara@bol.com.br, rrrighi@unisinos.br

Resumo. *Depuração de programas paralelos que executam em rede é pertinente para que sejam analisadas situações de impasse bem como pontos de gargalo de comunicação e/ou processamento. Nesse sentido, o presente artigo apresenta os primeiros passos para a depuração de programas em fases (Bulk Synchronous Parallel) executados com o simulador Simgrid. A depuração é feita pela geração de rastros e a visualização deles na ferramenta Vite. O objetivo principal é destacar a importância da depuração para aplicações em fases e como ela pode ser viabilizada através do Simgrid e sua ferramenta de visualização.*

1. Introdução

Aplicações em fases no estilo BSP *(Bulk Synchronous Parallel)* são comuns entre as aplicações paralelas de sucesso, como aquelas que calculam a previsão do tempo e sequenciamento de DNA [Dutot et al. 2005, Miao and Tong 2007]. Elas são compostas por uma série de superetapas, cada qual com as fases de computação, comunicação e barreira de sincronização. O processo mais lento compromete o desempenho de uma aplicação desse tipo. Equipamentos com diferentes processadores e o uso de uma arquitetura hierárquica onde a comunicação intra-site possui um custo menor que outra entre *sites* são fatores que tornam a sintonia por desempenho em aplicações BSP um desafio. Isso sem falar no dinamismo em nível de arquitetura e aplicação que podem alterar o comportamento dos processos.

Nesse contexto, esse artigo aborda uma análise inicial para depuração de aplicações do tipo BSP. A depuração é *post-mortem* e acontece através da visualização de rastros obtidos com a execução de uma aplicação paralela sobre o simulador Simgrid [Casanova et al. 2009]. Para tal, dar-se-á a utilização da ferramenta gráfica chamada Vite [ViTE's Project Page 2010] para visualizar a execução dos processos. O programador pode apontar rapidamente quais os processos mais demorados, qual superetapa e se a ação lenta diz respeito a processamento, comunicação ou uma espera para sincronização.

2. A importância da Depuração

Depuração é uma das ferramentas mais importantes para o programador. Isso ganha ênfase na programação paralela, onde tem-se vários fluxos de execução concorrentes e problemas de gargalos, impasse e postergação indefinida *(starvation)* podem ocorrer mais facilmente [James et al. 2012]. A depuração paralela não siginifica somente detecção de erros. Uma vez que ela normalmente acontece com a geração de restros de execução e o suporte de ferramentas de visualização, ela se torna uma técnica pertinente para analisar e

comparar o desempenho dos processos e apontar anomalias e oportunidades de extração de melhor desempenho.

A depuração possui papel de destaque em duas situações: arquitetura de grid e exepressão do dinamismo. Um grid pode ser formado por recursos heterogêneos (em nível de memória principal e secundária, bem como CPU) e a depuração pode auxiliar na análise do escalonamento dos processos. A segunda situação trata de uma característica cada vez mais presente em aplicações de grande desafio. Processadores compartilhados podem ficar sobrecarregados em um determinado momento e processos de uma aplicações podem alterar o seu padrão de comunicação em função dos dados de entrada. As ações têm um impacto direto no desempenho e podem ser vistas na ferramenta de visualização.

Aplicações BSP possuem uma facilidade quanto a depuração, uma vez que são compostas por uma coleção de etapas [da Rosa Righi et al. 2010]. Em adição, cada etapa começa com os processos sincronizados entre si. A ideia básica numa execução eficiente é que os processos executem o mais rápido possível e em equilíbrio. Aquele que for mais lento compromete o desempenho dos demais. A análise da visualização pode interferir no escalonamento processos-recursos, bem como em ações de migração de processos para reagir a situações de dinamismo. Estas ações têm um impacto direto no desempenho da aplicação, podendo as métricas de computação, comunicação e memória dela serem vistas na ferramenta de visualização, possibilitando uma melhor ação no reescalonamento dos processos.

3. Testes Preliminares

Foi utilizado o simulador Simgrid para o desenvolvimento de aplicações do tipo BSP. Esse simulador possui uma interface que posibilita ao programador escrever uma aplicação paralela, explicitando as partes de computação e comunicação de cada processo. O Simgrid recebe como dados de entrada dois arquivos: um para o mapeamento de processos-recursos e outro que informa a arquitetura paralela com os recursos disponíveis. A Figura 1 ilustra o arquivo de plataforma, enfatizando a criação de um cluster com 5 máquinas. Em adição, o simulador também pode ser executado para gerar rastros de execução. A saída do programa é um único arquivo de rastros que é carregado em uma ferramenta gráfica compatível com o Vite. Muitas vezes é complicada a depuração de uma grande quantidade de informações e processos em uma única tela. Nesse sentido, além da forma automática, a depuração no Simgrid também pode ser controlada pelo programador que define as características e os processos que deseja analisar.

Os testes realizados têm por objetivo analisar a facilidade do uso de Simgrid e a visualização de seus rastros na ferramenta Vite. A primeira avaliação compreende a análise de uma aplicação mestre-escravo dentro de um cluster. Para tal, num primeiro momento foram usados 4 escravos além do mestre. Numa segunda etapa foram usados 80 processos escravos. Em ambos os casos, Vite possui um a representação horizontal para definir o comportamento de um processos e linhas entre eles para expressar a comunicação pela rede. A Figura 2 apresenta um corte na tela de visualização na execução com 4 processos. É possível observar que o transmissor realiza comunicação do tipo assíncrona e que a transmissão de fato é mais demorada na recepção do processo 3. Ainda, é possível observar o tempo que os processos que recebem mensagems ficam ociosos pois realizam chamadas bloqueantes de recepção.

```
<? xml version = '1.0' ?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforce.inria.fr/simgrid.dtd">
<platform version = "3">
<AS id = "ASO" routing = "Full"

<cluster id = "pipca" prefix = "host-" suffix = ".pipca" radical = "0-4" power = "100000"
bw = "100000" lat = "5E - 5" bb_bw = "100000" bb_lat = "0.0001" />

<link id = "lk0" bandwidth = "100000" latency = "0.0001" />

</AS>
</platform>
```

Figura 1. O arquivo *platform.xml* com os parâmetros largura de banda e latência.

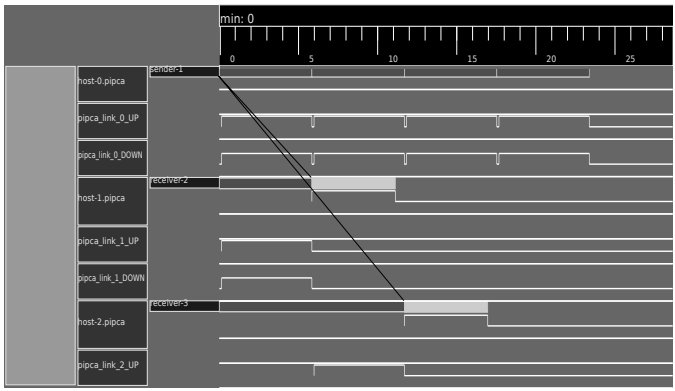

Figura 2. Visualização da execução de uma aplicação mestre-escravo

Diferentes rastros foram gerados a partir de alterações no código da aplicação. Mais especificadamente, é possível alterar o parâmetro de que define a quantidade de instruções a serem computadas na função Simgrid que cria uma tarefa *(MSG_Task_Create)*. Na mesma linha, a alteração do arquivo que define a máquina paralela também produz diferentes resultados. Diferentes capacidades de processamento e de largura de banda nas ligações de rede foram testadas e resultaram em diferentes gráficos no Vite.

4. Conclusão

Esse trabalho apresentou uma análise inicial do uso do simulador Simgrid para execução de aplicações paralelas e geração de rastros. O Simgrid é uma ferramenta simples que permite a escrita de diferentes aplicações paralelas e o mapeamento dos processos dela sobre diferentes plataformas. Apesar de simples, os testes realizados possibilitam a conclusão que a visualização de rastros possibilita a depuração de aplicações BSP. Foi constatado que o tempo de execução varia de acordo com a carga da aplicação, largura de banda e latência, bem como o número de nós empregados.

O trabalho mostrado nesse artigo representa os passos iniciais na pesquisa de

aplicações do tipo BSP. Os próximos passos passam pela análise de aplicações BSP que possui processos que migram entre processadores. A ideia é observar na linha horizontal de cada processo o possível ganho de desempenho. Para a execução de aplicações do tipo BSP, será usado o modelo de reescalonamento MigBSP [da Rosa Righi et al. 2010].

Referências

da Rosa Righi, R., Pilla, L. L., Carissimi, A., Navaux, P. A., and Heiss, H.-U. (2010). Observing the impact of multiple metrics and runtime adaptations on bsp process rescheduling. *Parallel Processing Letters*, 20(2):123–144.

Dutot, P., Goldman, A., Kon, F., and Netto, M. (2005). Scheduling moldable BSP tasks. In *11th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 3834 of *Lecture Notes in Computer Science*, pages 157–172. Springer Verlag.

James, D., Rosales, C., and Stanzione, D. (2012). Offline parallel debugging: a case study report. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, XSEDE '12, pages 41:1–41:2, New York, NY, USA. ACM.

Miao, W. and Tong, W. (2007). Agent based servicebsp model with superstep service for grid computing. *Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*, 00:255–260.

ViTE's Project Page (2010). ViTE - Visual Trace Explorer User Manual - Release 1.2. *http://vite.gforge.inria.fr/. Visitado em novembo de 2012.*

Casanova, H., Legrand, A., and Quinson, M. (2009). SIMGRID: a Generic Framework for Large-Scale Distributed Experiments. 9th International conference on Peer-to-peer computing IEEE P2P 2009 (2009)*, pages 1–6. Nancy University/Loria, France.*