Caracterização de Desempenho do Simulador de Dinâmica de Fluídos Alya

Guilherme Antonio Camelo, Lucas Mello Schnorr

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{gacamelo,schnorr}@inf.ufrgs.br

Abstract. *This article presents a performance characterization of the Alya fluid dynamic simulator. Experiments are conducted in parallel using the MPI specification implemented by OpenMPI, and was traced using the Extrae tracing tool. By taking a closer look to the traces, it is possible to effectively see different performance patterns such as the communications among ranks, and the effective application load imbalance.*

Resumo. *Este artigo apresenta uma caracterização de desempenho do simulador de dinâmica dos fluidos Alya. As medidas são obtidas com execuções em paralelo usando a especificação MPI implementado por OpenMPI, sendo que a aplicação foi rastreada utilizando a ferramenta de rastreamento Extrae. Os resultados indicam ser possível a observação de diferentes padrões de desempenho, tais como as comunicações entre as processos, e o desequilíbrio de carga efetivo da aplicação.*

1. Introdução

Simulação numérica é utilizada na ciência para modelar e entender comportamentos da natureza. Geralmente, essas simulações são realizadas através de técnicas baseadas em dinâmica de fluidos. Em geral, é implementado um modelo da realidade que é resolvido de forma numérica e iterativa através de passos de tempo. Computadores de alto desempenho são frequentemente utilizados para a execução desses modelos numéricos, visto que execuções sequenciais tornariam a execução de certas simulações impraticáveis. Técnicas de programação distribuída e paralela disponibilizam um poder computacional elevado. Ferramentas como OpenMPI, por exemplo, possibilitam execuções de programas complexos em menos tempo, e são ideais par lidar com problemas físicos que simulam o mundo real. Este é o caso do arcabouço Alya[Vázquez et al. 2014], que é um projeto de código aberto do Barcelona Supercomputing Center (BSC) para resolver numericamente problemas físicos, fazendo parte do Benchmark Prace[Prace 2013].

A maioria das aplicações de simulação numérica, inclusive aquelas baseadas em Alya, tem cargas irregulares durante a execução. A irregularidade leva a um desbalanceamento de carga tanto entre os recursos quanto no tempo, conforme avança a simulação. Descobrir o real comportamento da aplicação em uma determinada plataforma é um passo inicial fundamental para se aplicar técnicas de otimização, tais como melhores algoritmos de balanceamento ou um padrão de comunicação mais adequado. A forma mais eficiente de se obter tais informações, quando não se conhece o código da aplicação, é aplicar técnicas de rastreamento. Elas registram em arquivos o comportanto da aplicação na forma de eventos, registrando o início e final de chamadas às funções MPI, por exemplo.

Neste artigo, a ferramenta de simulação de dinâmica de fluidos Alya é executada em uma plataforma de 4 nós totalizando 64 cores, com OpenMPI[Gabriel et al. 2004]. O objetivo é investigar se a ferramenta Alya, para uma dada entrada de dados, apresenta um comportamento de cálculo irregular entre os recursos e no tempo. Para tal, neste trabalho, a ferramenta de rastreamento Extrae é empregada com o objetivo de registrar todas as diretivas de comunicação MPI. Este tipo de estudo já foi conduzido [Rodríguez 2014] em uma plataforma de larga escala. Nosso objetivo secundário através deste trabalho é entender o comportamento em uma plataforma acessível mas de menor escala.

O texto tem a seguinte organização. A Seção 2 expõe a plataforma experimental utilizada. A metodologia empregada na caracterização de desempenho é detalhada na Seção 3. Os resultados são apresentados na Seção 4. Os principais resultados assim como os trabalhos futuros são enfim detalhados na Seção 5. Uma investigação científica reprodutível é fundamental[Legrand 2015]. Este trabalho é portanto fruto de um esforço de reprodutibilidade, disponível em `http://github.com/guiacamelo/alya/`.

2. Ambiente de Execução

Os experimentos foram executados nos computados que formam o cluster Draco do Instituto de Informática da Universidade Federal do Rio Grande do Sul. O cluster é formado por 8 computadores, cada computador possui 16 *cores* físicos (32 com hyperthreading), 64 gigabytes de memória RAM, processadores Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz, rodando Ubuntu 14.04.4 LTS. Nas execuções foram utilizados somente os *cores* físicos, e nas experimentações, diferentes configurações foram testadas, variando o número de *cores*, número de nós, e quantidade de passos de tempo (*timestep*s). Para a criação do traço foi utilizado a ferramenta *Extrae* 3.3.0, e para a execução em paralelo, utilizamos a versão 1.10.2 do OpenMPI.

3. Metodologia Experimental

Os experimentos foram conduzidos usando quatro nós para um total de 64 *cores*. A simulação rodou somente até o fim do terceiro *timestep* devido ao grande tamanho dos arquivos criados no traço. A ferramenta *Extrae* cria traços individuais de cada processador, mantendo informações relativas a comunicação e execução. São criados vários arquivos em formato *.mpits* que contem informações sobre o que foi executado por um determinado processo. Em seguida é necessário então converter os arquivos usando a ferramenta *mpi2prv* que faz a união dos arquivos e os converte para o formato Paraver (*.prv*). Após isso, um script em *perl* é utilizado para filtrar as informações relevantes em uma saída no formato *Comma-Separated Values* (CSV), utilizado pelos scripts de análise. O CSV resultante do processo de execução/rastreamento tem quatro colunas de dados: processo, referente ao rank MPI (*Rank*), o tempo do início do estado do processo (*Start*), tempo do fim do estado (*End*), e qual o nome do estado *MPI* (*State*). Com estas informações é possível calcular a carga de trabalho efetiva (em tempo de execução) de cada processo, assim como construir gráficos espaço/tempo que nos informa a ordem e em que tempo cada estado ocorreu em cada processo. Todos os cálculos são feitos a partir de scripts escritos na linguagem R, fazendo uso das bibliotecas *ggplot2* e *dplyr*. A Figura 1 resume esses passos para se gerar a execução e análise do experimento.

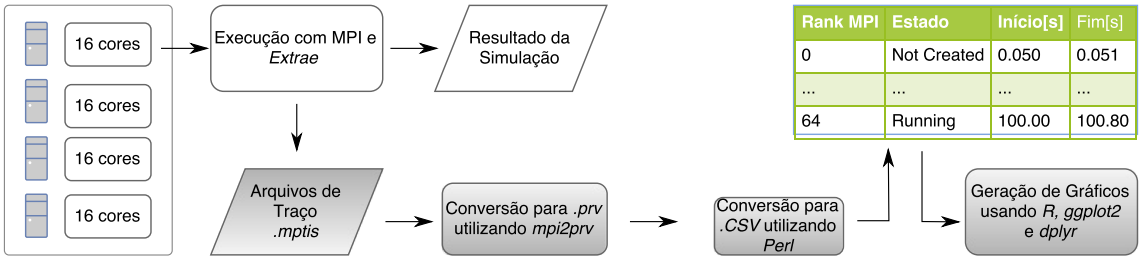

Figura 1. Metodologia usada para a execução e análise do experimento.

4. Resultados

A caracterização de tempo se baseia principamente nos resultados obtidos a partir de uma execução com quatro nós e 64 *cores*. Escolheu-se um número reduzido de nós pois os arquivos de rastro gerados pelo *Extrae* são minados de informação e facilmente alcançam tamanhos grandes (na ordem de Gigabytes). Tentou-se então evitar arquivos demasiadamente grandes, facilitando o processo de *parsing* e filtragem de dados. Pelo mesmo motivo, utilizou-se somente três *timestep*s. Essa abordagem reduzida tem o objetivo inicial de identificar características relevantes da aplicação, e avaliar se é interessante realizar o mesmo experimento para problemas maiores (maior número de *timestep*s). São apresentados a seguir uma visão geral e um detalhamento do comportamento da aplicação.

4.1. Balanceamento de Carga

A Figura 2 mostra o tempo agregado de cálculo efetivo para cada processo participante da aplicação. Os tempos são calculados efetuando-se o somatório de todos os períodos de tempo em que o processo realiza o processamento dos dados. Não fazem parte dessas medidas o tempo de comunicação MPI ou de sincronização ponto-a-ponto ou coletiva, por exemplo. Nota-se que a maioria dos processos tem um total de computação na ordem de 150 segundos. Em alguns casos, no entanto, os tempos são maiores, até 300 segundos para o caso do processo 60. Isso é um indício de desbalanceamento de carga espacial entre os processos considerando este estudo de caso específico.

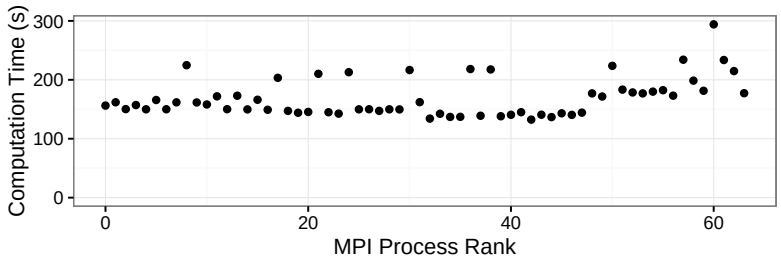

Figura 2. Tempo agregado de cálculo efetivo (eixo Y) por processo (X).

4.2. Espaço/Tempo da Execução

Na figura 3 é apresentado o detalhamento do balanceamento de carga, mostrando o intervalo de tempo que cada processo ficou em estado de cálculo efetivo. No início da execução, o processo zero se engaja em dividir as tarefas entre todos processos. Até o fim da divisão de tarefas, que acontece a marcação temporal de 150 segundos, somente o processo zero é executado. Após este momento, os outros 63 processos começam a trabalhar.

A partir desse momento, o processo raiz assume um papel de organizador, ficando ocioso na espera da resposta dos demais processos.

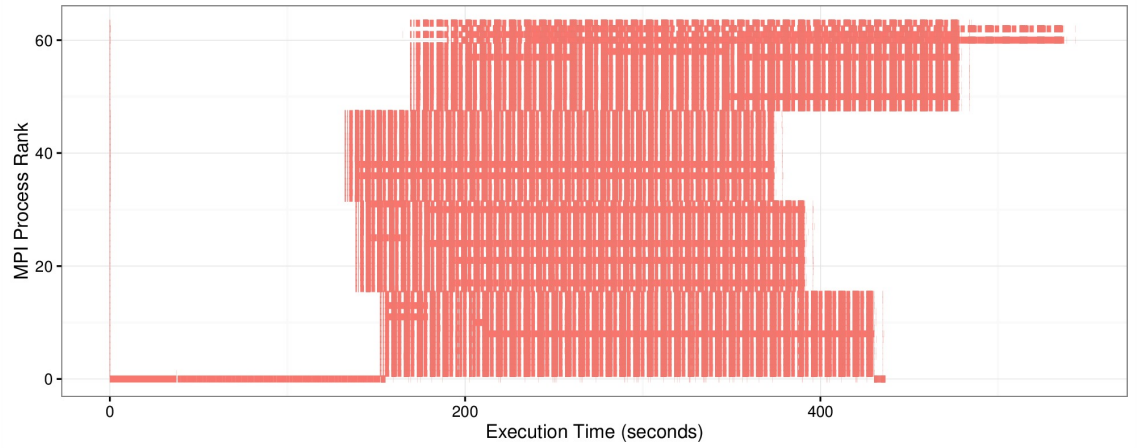

Figura 3. Processo (eixo X) por tempo em estado de calculo efetivo.

Uma particularidade interessante que pode também ser observada na Figura 3 é a existência de quatro grupos de processos: do 0–15, do 16–31, e assim por diante. Esse comportamento se correlaciona com a quantidade de nós utilizada no experimento, indicando que internamente em um nó todos os processos começam seu cálculo praticamente ao mesmo tempo. Isso leva a indicar que a distribuição inicial da carga a partir do processo zero não é escalável, pois visivelmente os cálculos nas quatro máquinas começam de maneira sequencial: primeiro o grupo entre os processos 32–47, depois o grupo dos processos 16–31, em seguida o grupo que contém o processo zero (0–15) e enfim o grupo dos processos 48–63. Neste último grupo, podemos observar também uma anomalia nos processos 60 e 62, visto que eles começam/terminam o cálculo após os demais de seu nó.

O processo 60 apresenta um comportamento peculiar quando comparado aos demais. Na Figura 2, ele apresenta um tempo de cálculo efetivo total de cerca de 300 segundos, o dobro da maioria dos demais processos. Na figura 3, ele é um dos processos que tem um comportamento anômalo, começando e terminando sua execução por último. Além disso, os estados de execução (*Running*), onde o cálculo é efetivamente realizado, parece não conter espaços como os demais processos. Isso provavelmente indica que o tempo gasto nas funções de comunicação para este processo em particular seja menor.

4.3. Comportamento de cada Processo por Estado

A Figura 4 apresenta um sumário do tempo (eixo Y) dedicado por processo (eixo X) a cada estado comportamento (diferentes facetas). Somente os cinco estados mais relevantes são apresentados (*Blocking Send*, *Group Communication*, *Running*, *Send Receive* e *Waiting a message*). Os demais apresentam ou tempos muito pequenos ou inexistentes. Os envios bloqueantes (*Blocking Send*) são menos representativos e de certa forma semelhantes entre todos os processos (salvo para o grupo 48–63, um pouco maior). As comunicações de grupo (*Group Communication*) tem tempos bastante diferentes entre os processos e um tempo bastante grande para o processo zero, como detalhado nas seções anteriores. O desbalanceamento de carga é explícito pela faceta de cálculo (*Running*, semelhante aos dados apresentados na Figura 2). Os tempos de envio e recepção (*Send Re-*

ceive) são relativamente homogêneos entre os processos, salvo para alguns onde o tempo é menor. Enfim, podemos observar que a razão potencial da anomalia dos processos 60 e 62 é devido ao tempo adicional gasto no estado *Waiting a message*, como pode ser visto na faceta mais a direita deste gráfico.

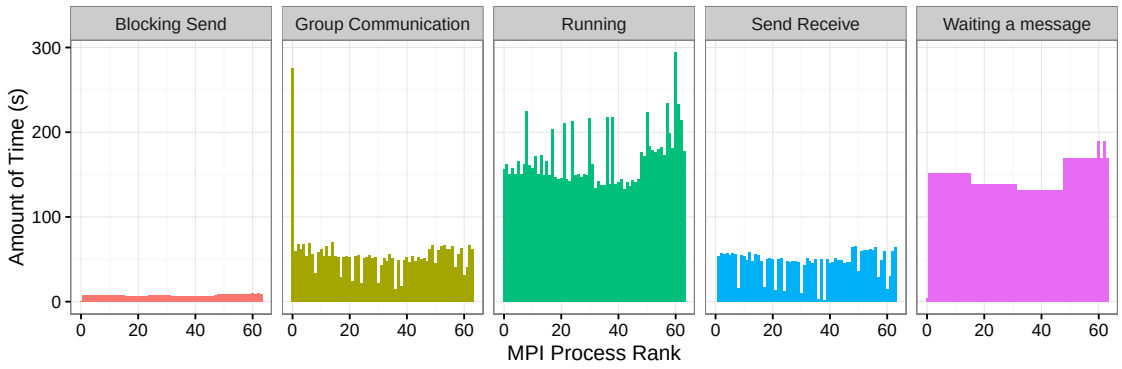

Figura 4. Tempo dedicado a cada estado por Processo.

4.4. Porcentagem de Desbalanceamento

Segundo [Pearce et al. 2012], em seguida revalidado por Alles [Rodrigues 2016], usa-se a fórmula da porcentagem de desbalanceamento de carga (*percent imbalance*) para avaliar formalmente o balanceamento de carga. Ela é descrita pela Equação 1, onde λ é o valor que se deseja calcular, L_{max} tem o valor do processo com a maior carga, $\overline{L}$ tem a média da carga computacional entre todos os processos participantes. A métrica pode ser calculada tanto para toda a execução da aplicação quanto para diferentes fases.

$$\lambda = (\frac{L_{max}}{\overline{L}} - 1) * 100 \quad (1)$$

Neste trabalho foi realizado o cálculo da métrica considerando todo o tempo de execução, sendo portanto uma indicação global de desbalanceamento. A métrica caracteriza a distribuição desigual de trabalho, e ao aplicar ao conjunto de dados, temos $\lambda = 74.25161$. Isso indica que existe um desbalanceamento de cerca de 75% da carga de trabalho, indicando que existe um potencial de melhora caso a carga seja melhor balanceada entre os recursos computacionais.

5. Conclusão e Trabalhos Futuros

Este artigo apresenta uma análise de desempenho da ferramenta de simulação de dinâmica de fluidos Alya com o intuito de melhor conhecer seu comportamento principalmente no que diz respeito ao balanceamento de carga. Para tal, uma simulação foi executada em uma plataforma de quatro nós computacionais totalizando 64 *cores*, levando a uma execução de aproximadamente 450 segundos. O comportamento da execução foi rastreado utilizando a ferramenta Extrae, permitindo descobrir informações relevantes sobre o funcionamento do core do *Alya* para o caso executado. Dentre os resultados obtidos, salientamos que uma boa parte do tempo (cerca de 34% para uma execução com três *timesteps*) é de certa forma desperdiçada no começo da execução para dividir o problema, criando

uma sobrecarga considerável. Outros resultados incluem a detecção de anomalias em alguns processos e a percepção que o processo raiz (zero) envia sequencialmente os dados particionados aos diferentes nós, tornando o início da aplicação demasiadamente lento. O cálculo do percentual de desabalanceamento segundo Pearce nos indica que existe um potencial de melhoria caso as cargas sejam melhor distribuídas entre os participantes.

Como trabalhos futuros, estudaremos possíveis mudanças no código que proporcionem um balanceamento distribuído de forma mais igualitária. Executaremos também experimentos similares com outras configurações, variando o número de nós, número de *cores*, e número de *timestep*s com o objetivo de confirmar o comportamento adquirido neste experimento. Esperamos também marcar manualmente o início das iterações do algoritmo (alterando o código da aplicação) de forma que as métricas de balanceamento possam ser calculadas para cada fase de cálculo/comunicação. Esse refinamento permitirá verificar o desbalanceamento ao longo do tempo.

Reconhecimentos

Esta investigação recebe fundos do projeto HPC-ELO, do programa H2020 da União Européia e do MCTI/RNP-Brasil através do projeto HPC4E com o código 689772, do projeto FAPERGS/Inria ExaSE, do projeto universal do CNPq 447311/2014-0, e do laboratório internacional CNRS/LICIA. Nós também agradecemos Flavio Alles pelas discussões referentes às métricas de balanceamento de carga.

Referências

[Gabriel et al. 2004] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al. (2004). Open mpi: Goals, concept, and design of a next generation mpi implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer.

[Legrand 2015] Legrand, A. (2015). Scientific methodology and performance evaluation. `https://github.com/alegrand/SMPE`.

[Pearce et al. 2012] Pearce, O., Gamblin, T., De Supinski, B. R., Schulz, M., and Amato, N. M. (2012). Quantifying the effectiveness of load balance algorithms. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 185–194. ACM.

[Prace 2013] Prace (2013). Prace Research Infrastructure unified european applications benchmark suite - prace research infrastructure. `http://www.prace-ri.eu/ueabs/`. Publicado: 2013-10-17, Acessado: 2016-07-15.

[Rodrigues 2016] Rodrigues, F. A. (2016). Study of Load Distribution Measures for High-performance Applications. Master's thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

[Rodríguez 2014] Rodríguez, J. (2014). Performance analysis of alya on a tier-0 machine using extrae. Technical Report 151, Prace White Papers.

[Vázquez et al. 2014] Vázquez, M., Houzeaux, G., Koric, S., Artigues, A., Aguado-Sierra, J., Aris, R., Mira, D., Calmet, H., Cucchietti, F., Owen, H., Taha, A., and Cela, J. M. (2014). Alya: Towards exascale for engineering simulation codes. *arXiv.org*.