Utilização de ambiente de alto desempenho para simulação de sistemas de comunicação utilizando linguagem JAVA

Maximiliano N. Biscaia, Cristhof J. R. Runge, André Angellis, André Leon S. Gradvohl

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – Universidade Estadual de Campinas (UNICAMP) – Campus de Limeira 13484-332 – Limeira – SP– Brasil

maxi.biscaia54@gmail.com, cristjrr@ft.unicamp.br, andre@ft.unicamp.br, gradvohl@ft.unicamp.br

Abstract. *This work shows the results using different strategies to implement a simulation software in Java to measure the performance of a communication system with the main aim of obtaining gains in simulation processing time. The results obtained using a high performance cluster computing and a multi core desktop are described. Three different approaches are used, serial programming , parallel multithreading and Fork/Join. The simulation results shows that, as the parallelism increases a saturation occurs, and even a performance degradation is observed, indicating that the time processing gains are not a monotonically increasing function of the parallelism.*

Resumo. Este trabalho busca mostrar os resultados obtidos na tentativa de se obter ganhos de tempo de processamento utilizando diferentes estratégias na implementação de um software de simulação para medida de desempenho de um sistema de comunicação escrito em linguagem Java. O artigo descreve os resultados obtidos através das simulações realizadas em um cluster de computação de alto desempenho, e utilizando um desktop multicore. Três diferentes abordagens de implementação são apresentadas, serial, paralela multithreading e Fork/Join. Os resultados obtidos apontam para uma saturação, e inclusive para uma perda de desempenho a partir de um certo grau de paralelismo, indicando que os ganhos de tempo de processamento não são uma função monotônica crescente do grau de paralelismo.

1. Introdução

Devido à natureza estocástica dos canais de telecomunicações, a simulação utilizando algoritmos de força bruta, que utilizam quantidades massivas de dados para obter resultados numéricos, é normalmente utilizada. No entanto, tal estratégia muitas vezes conduz a uma necessidade de processamento além da capacidade computacional dos sistemas convencionais, ou ainda acaba por demandar horas e as vezes dias de processamento para obtenção de um único ponto em uma curva de resultados de simulação. Nesse contexto, o processamento de alto desempenho (HPC – High Performance Computing) foi visto como uma alternativa para o tratamento do grande volume de dados requisitado por simulações dessa natureza. O ambiente de alto desempenho usado para o desenvolvimento desse trabalho é um Cluster IBM-AIX

pertencente à Faculdade de Tecnologia da Unicamp. A arquitetura de cluster pode ser definida como um conjunto de maquinas ou nós, que cooperam entre si na execução de aplicações utilizando a troca de mensagens pela rede [Buyya 1999]. O software desenvolvido e analisado nesse trabalho simula a geração de bits de informação transmitidos através de um canal de transmissão ruidoso[Lathi 1988] e utiliza um corretor de erro do tipo Reed Solomon(RS) [Lin e Costello 1983], que tem por intuito proteger os bits transmitidos e corrigi-los após a passagem pelo canal, assim calculando a taxa de erro de bit (TEB). Para geração dos bits de informação que serão transmitidos e recebidos o algoritmo do software utiliza o método estatístico de Monte Carlo [Rino e Costa 2013], amplamente empregado em modelos probabilísticos.

O desenvolvimento de softwares para arquiteturas de alto desempenho é comumente realizado em linguagem de programação C, devido à presença de bibliotecas de MPI (Message Passing Interface) que permitem a distribuição de tarefas em um ambiente de alto desempenho[Alsmadi, Khamaiseh, and Xu]. Contudo, a escolha da linguagem Java para a criação do software de simulação se justifica pelo grande número de bibliotecas presentes na linguagem que facilitam o desenvolvimento de programas, além da possibilidade de utilização de mecanismos de paralelismo como o uso de multithreadings ou estruturas do tipo Fork/Join, amplamente documentados pela empresa Oracle.[Oracle 2016].

O objetivo desse trabalho é comparar e analisar as etapas de teste do programa de simulação desenvolvido, através do seu desempenho em termos de tempo de processamento em diferentes ambientes e através do uso de diferentes estratégias de implementação.

2. Materiais e Métodos

Nesse trabalho foram utilizados um computador pessoal Desktop com as seguintes configurações de hardware: processador AMD E1-2100 com 2 cores de 1.0 GHz e 4GB de memória RAM e um ambiente de alto desempenho Cluster IBM AIX do Laboratório de Simulação e de Computação de Alto Desempenho da Faculdade de Tecnologia da Unicamp, [LaSCADo 2016]. O Cluster IBM contém as seguintes configurações: o ambiente é composto por cinco máquinas, cada qual contendo quatro processadores Power7 de 8 núcleos e cada núcleo de 3,0 GHz, de modo que cada núcleo processa 4 threads simultaneamente [IBM, 2013]. O software desenvolvido para realização dos testes gera um vetor de símbolos (vetor de números inteiros) que não apresentam dependência entre si (descorrelacionados), sendo que para o processamento paralelo esse vetor é quebrado em subvetores que são tratados de forma paralela. Para a comparação de tempo entre as plataformas, utilizaram-se três abordagens diferentes: serial, multithreading e Fork/Join. O código do software foi desenvolvido de acordo com o diagrama da Figura 1:

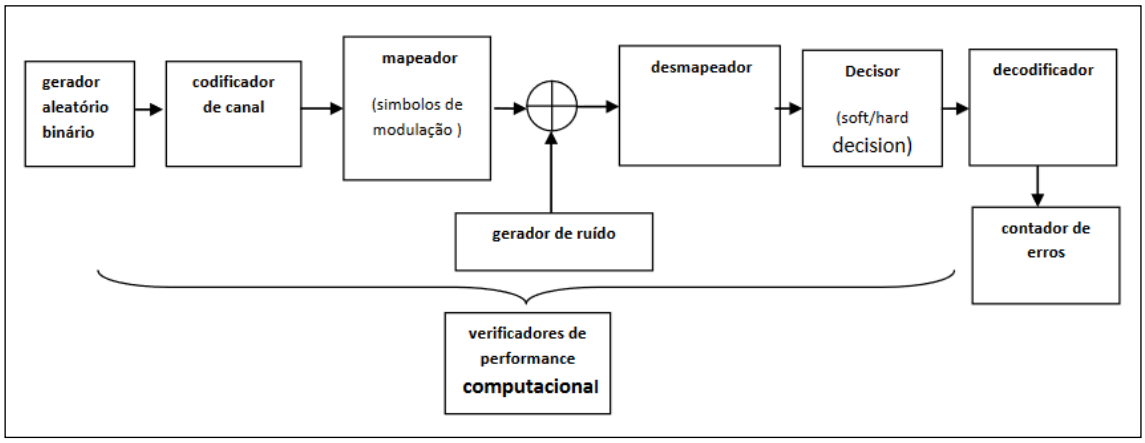

Figura 1. Diagrama de blocos do software de simulação

3. Utilização da Abordagem serial

Na primeira abordagem buscou-se verificar os ganhos obtidos através de um algoritmo de simulação serial, inicialmente testado em computador desktop e posteriormente portado para ambiente de alto desempenho.

3.1 Comparação entre os resultados no Cluster IBM AIX e no Desktop

O programa de simulação desenvolvido foi avaliado nas duas plataformas (Desktop e Cluster), a fim de traçar uma medida inicial de tempo de execução para a abordagem serial. A figura 2 apresenta a comparação entre os tempos de execução.

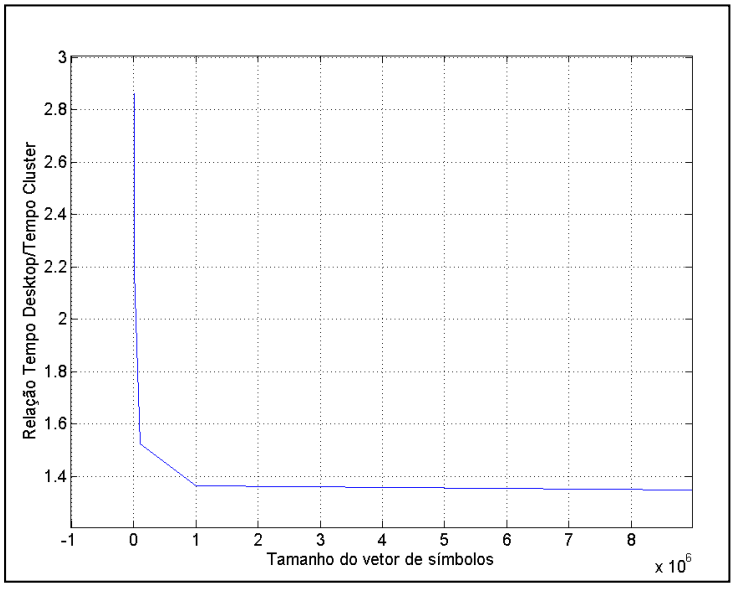

Figura 2. Relação de tempo gasto pelo Cluster e PC

Analisando o gráfico observa-se que os ganhos de tempo de processamento obtidos no Cluster em relação ao desktop não são significativos, sendo esses em torno de 3 vezes para vetores de simulação da ordem de 10^3 símbolos, caindo para menos de uma vez e meia em torno de 10^7 símbolos. A partir desse primeiro resultado pensou-se em uma

forma de se obter maiores ganhos no tempo de processamento da simulação, assim, buscou-se incorporar ao programa métodos de paralelismo.

4. Abordagem Multithreading

Visto os resultados da abordagem serial, pensou-se no uso da abordagem multithread a fim de verificar seu ganho de desempenho e compará-lo aos testes anteriores feitos em Desktop e Cluster. Através do uso desse mecanismo, o programa passou a quebrar o vetor de simulação em vetores menores e processá-los através de múltiplas threads.

4.1. Resultados de simulação

A Figura 3 apresenta o resultado do tempo de execução do teste no Cluster aumentando-se o número de threads de 1 até 8 e com um vetor com comprimento de 10^7 símbolos.

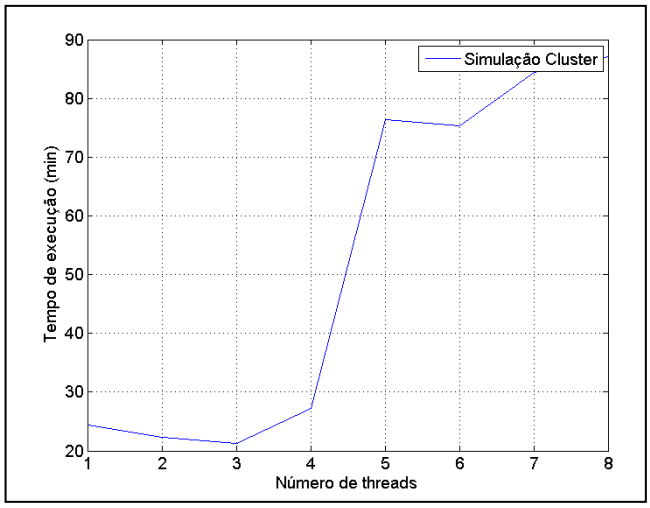

Figura 3. Tempo de execução multithreading IBM AIX

A Figura 4 apresenta o resultado do teste no computador desktop aumentando-se o número de threads de 1 até 8.

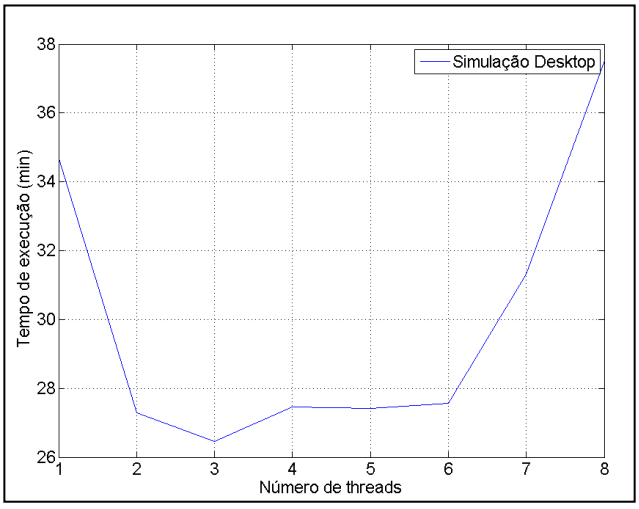

Figura 4. Tempo de execução multithreading desktop

Através da análise das figuras observa-se que nos dois ambientes, há ganho de desempenho somente até cerca de 3 threads, sendo que aumentando-se o número de threads observa-se um aumento no tempo de execução. Verifica-se também que o tempo de execução em 3 threads foi menor no Cluster.

5. Abordagem Fork/Join

A partir dos resultados dos testes obtidos com o programa de simulação utilizando o paralelismo multithreading, foi realizada então uma investigação do uso de processamento paralelo por meio da estrutura Fork/Join do Java. O mecanismo Fork/Join quebra as tarefas em pequenas partes e as executa recursivamente, sendo que após o término da execução o programa une os resultado de cada parte. O objetivo é usar o máximo de processamento dos núcleos disponíveis no processador da máquina para melhorar o desempenho da aplicação.

5.1 Resultados de simulação

As figuras 5 e 6 mostram os resultados de simulação para um vetor de 10^7 símbolos, no cluster e no desktop respectivamente. O tamanho do vetor é expresso pelo parâmetro "Total Units". Já os parâmetros "Target Paralelism" e "Sequential threshold" correspondem respectivamente ao número de processadores encontrados pelo programa e em quantos pedaços o programa quebrou a tarefa.

```
Simulating ..: Test 001; Repetition: 1; SimulationParameters [simulationName=Test
001 dataSize=9, signalNoiseRatio=6.5, repetitions=1] Total Units:10000000; Target
                  parallelism: 128; Sequential threshold: 9765
                       Error rate: 1.78364423076924l6E-4
       Time elapsed for this repetition: 5 h. 54 min. 35 sec. (21275115 ms.)
      ================================================================
           Overall time elapsed: 5 h. 54 min. 35 sec. (21275159 ms.)
```

Figura 5. Resultados de simulação Fork/Join IBM AIX

```
Simulating ..: Test 001; Repetition: 1; SimulationParameters [simulationName=Test
             001, dataSize=9, signalNoiseRatio=6.5, repetitions=1]
     Total Units:10000000; Target parallelism: 2; Sequential threshold: 625000
                       Error rate: 1.791778846153844E-4
        Time elapsed for this repetition: 0 h. 37 min. 13 sec. (2233467 ms.)
=========================================================================Overall
                time elapsed: 0 h. 37 min. 13 sec. (2233625 ms.)
```

Figura 6. Resultados de simulação Fork/Join Desktop

Observa-se pelos resultados obtidos através da abordagem Fork/Join, que também nesse caso não foram obtidos ganhos no tempo de processamento, sendo que no caso do cluster o resultado é ainda pior que o caso da abordagem multithreading.

6. Tempo de criação das threads e tempo total de execução

Baseado nos resultados dos testes com a abordagem multithreading e abordagem Fork/Join levantou-se a hipótese de que o tempo de criação das threads poderia estar contribuindo de forma significativa no tempo global de execução, o que justificaria o

comportamento de aumento do tempo de execução com o aumento do paralelismo a partir de um certo valor. Verificou-se, no entanto, que o mesmo não é significativo em relação ao tempo global de execução observado nas simulações, tanto no Cluster quanto em ambiente Desktop. É possível constatar também que os valores de tempo de criação apresentam alto índice de flutuação, dessa forma, não seguindo uma tendência possível de ser comparada aos resultados anteriores. A figura 7 apresenta o gráfico do resultado de tempo de criação de threads para em ambiente cluster e a Figura 8 apresenta o tempo de criação para o ambiente Desktop.

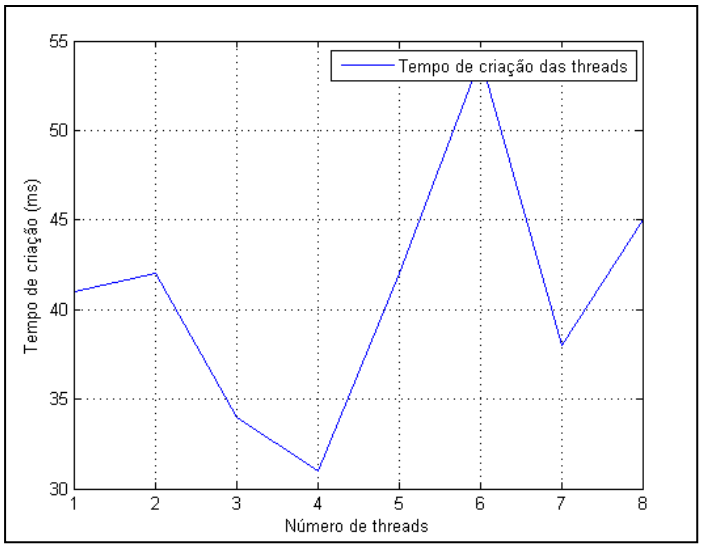

Figura 7. Tempo de criação das threads no Cluster

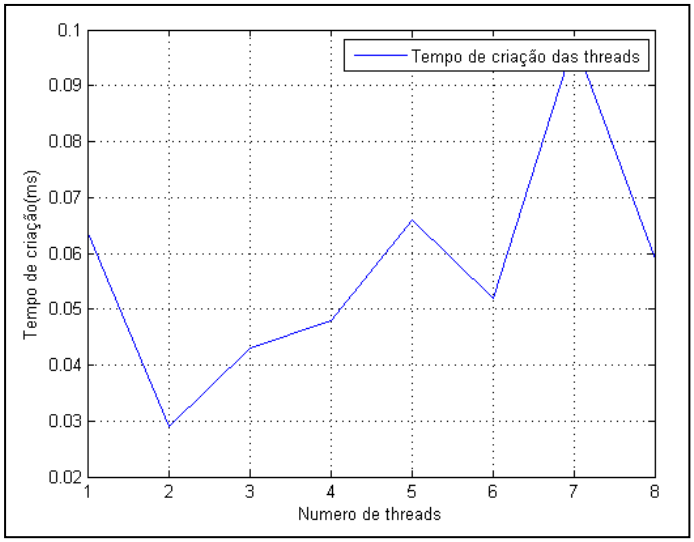

Figura 8. Tempo de criação das threads em Desktop

7. Conclusão

Após os testes com o programa de simulação por meio das três abordagens de processamento, é possível comparar e analisar os resultados obtidos tanto em ambiente Desktop, quanto em ambiente Cluster. Observando os resultados dos testes feitos nesse trabalho, é possível constatar que o melhor caso de desempenho de tempo ocorreu na abordagem de processamento paralelo em Cluster, já que o tempo de execução foi de cerca de 20 minutos contrapondo o tempo de execução do Desktop que foi de cerca de 26 minutos. Contudo, para o caso Fork/Join o computador Desktop apresentou melhor desempenho de tempo, executando o programa em 37 minutos, contrapondo o tempo de execução do Cluster, que foi de 5h. A partir dos testes com abordagem paralela, é possível constatar também que o ganho de desempenho se dá até certo número de threads, sendo 3 threads tanto para o cluster quanto para desktop, de modo que a partir daí o tempo de execução do programa aumenta.

Durante a investigação levantou-se a hipótese de que o aumento do número de threads interferiria diretamente no tempo global de execução do programa. Os gráficos de tempo de criação das threads mostraram que o tempo exigido pelo programa para criar cada thread é quase nulo. Dessa forma a hipótese não se confirma.

Até onde se conseguiu investigar, não foi obtida uma explicação definitiva para a perda de desempenho observado a partir do aumento do grau de paralelismo. Uma possível justificativa para os resultados levantados talvez esteja relacionada aos mecanismos internos da máquina virtual Java (JVM) ou mesmo os mecanismos de distribuição de threads por parte do sistema operacional da máquina, no entanto, esse tema deverá ser objeto de investigação para trabalhos futuros.

Referências

Oracle e afiliados, “Java Platform Standard Edition 8”. Disponível: http://docs.oracle.com/javase/8/docs/, Dezembro 2016.

Lathi, B.P. Modern Digital and Analog Comunication Systems, Oxford University Press, 1988.

Lin, S. e Costello. D. Error Control Coding: Fundamentals and Applications, Prentice-Hall, 1983.

Rino, J. e Costa. ABC da Simulação Computacional, Livraria da Física, 1ª Edição, 2013.

LaSCADo, “Hardware”. Disponível: http:// cluster.ft.unicamp.br/ wiki/ doku.php? id= hardware , Dezembro de 2016.

IBM. “IBM Power 755 server”. Disponível: https://www-03.ibm.com/systems/power/hardware/755/ , Dezembro, 2016.

Buyya, R. (1999). High-Performance Cluster Computing: Architectures and Systems. Prentice Hall, USA.

Alsmadi, I. Khamaiseh, S. and Xu, D "Network Parallelization in HPC Clusters," *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, 2016, pp. 584-589.