ISSN: 1983-8174 *Universidade Regional do Cariri - URCA - Crato, Ceará*

Um Estudo Comparativo de Desempenho em Cálculos de Média Aritmética Considerando Múltiplas Tarefas Computacionais

Josuel Feitosa Rodrigues, André Ricardo Dantas Bezerra, Mateus Alves Vieira Neto, Guilherme Álvaro Rodrigues Maia Esmeraldo

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *Campus* Crato.

Introdução

Os volumes de dados que devem ser processados têm se tornado cada vez maiores [1]. Programas de computador têm consumido cada vez mais tempo para realizar operações sobre esses dados para gerar novas informações. Com isso surgiram os programas paralelos [2].

Programação paralela consiste em subdividir uma tarefa computacional em tarefas menores, as quais poderão ser realizadas em paralelo, permitindo assim um maior aproveitamento dos recursos computacionais e, consequentemente, aumento de desempenho[2].

Para a implementação de tarefas paralelas utiliza-se processos e/ou *threads* [3]. Um processo pode ser definido como um programa em execução [3-4]. Já uma *thread*, é um fluxo de execução independente de um processo [5], de forma que este pode ter uma ou várias *threads*. Em um sistema *multithread*, as *threads* compartilham os segmentos de código e de dados do processo.

Este artigo apresenta um estudo comparativo entre o desempenho de uma tarefa de cálculo de médias aritméticas, quando implementada em múltiplos processos e múltiplas *threads*.

Metodologia

O estudo consistiu, inicialmente, no desenvolvimento de um programa de cálculo de média tradicional, o qual contém apenas um processo, e de programas com múltiplos processos (2, 4 e 8 processos) e com múltiplas *threads* (2, 4 e 8 threads). Cada um dos programas foi executado 1000x e, dos tempos obtidos para o cálculo de média, em cada um deles, computou-se a média aritmética. Essas médias foram comparadas de forma a se verificar em qual dos cenários obteve-se maior desempenho.

Para os experimentos, todos os programas foram codificados com a linguagem de programação C (padrão ANSI C [6]). Utilizou-se memória compartilhada para comunicação interprocessos [3] e, para os cálculos de média, utilizou-se uma base de dados com um milhão de números inteiros. A plataforma utilizada para os experimentos consistiu de um processador dual core Intel Core i5 de 2,3 GHz, 4GB de memória RAM DDR3 e sistema operacional GNU/Linux.

Resultados e Discussão

A Tabela 1 mostra os tempos médios obtidos nos cálculos da média aritmética do conjunto de inteiros, utilizando cenários com múltiplos processos e múltiplas threads.

Analisando a Tabela 1, o algoritmo tradicional, que é aquele que considera apenas um processo e uma *thread*, consumiu o tempo médio de *113 ms*. Porém, ainda na Tabela 1, percebe-se que foi possível reduzir esse tempo ao se utilizar 4 e 8 processos, sendo que o melhor tempo médio (*105,1 ms*) foi obtido com o primeiro.

Tabela 1 – Comparativo de desempenho para cálculo de média entre múltiplos processos e múltiplas *threads*.

No. de Tarefas	Processos (Tempo em ms)	*Threads* (Tempo em ms)
1	113,0	
2	115,0	138.3
4	105,1	127.4
8	106,3	124.7

Ainda analisando esses resultados, percebe-se uma degradação no tempo de cálculo ao se utilizar *threads*. De acordo com Ranade, em [5], o uso de *threads* somente é adequando quando existe comunicação considerável entre as tarefas, o que não é o caso do nosso experimento.

Conclusões e Perspectivas

Este trabalho apresentou um estudo comparativo de desempenho para um programa de cálculos de médias aritméticas, considerando múltiplos processos e múltiplas *threads*. Os resultados mostraram que é possível ter aumento de desempenho quando os cálculos são divididos entre tarefas computacionais paralelas. Planeja-se o uso de técnicas de gerenciamento de escalonamento, como atribuir prioridades e afinidade aos processos, visando aumentar mais ainda o desempenho da aplicação.

Agradecimentos

Ao Instituto Federal de Educação do Ceará (IFCE) - campus Crato por fomentar parte desta pesquisa.

Referências

[1] Manyika, J., McKinsey Global Institute, Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C, and Byers, A. H. **Big data: The next frontier for innovation, competition, and productivity**. McKinsey Global Institute, 2011.

[2] Gebali, F. **Algorithms and Parallel Computing**. John Wiley & Sons, Inc, 2011.

[3] Tanenbaum, A. S., Woodhull, A. S. **Operating Systems Design and Implementation, 3e**. Pearson Education, 2011.

[4] Null, L. and Lobur, J. **The Essentials of Computer Organization and Architecture, 2e**. Bookman, 2006.

[5] Ranade, D. M. **Shared Data Clusters: Scaleable, Manageable, and Highly Available Systems (VERITAS Series)**. John Wiley & Sons, 2003.

[6] ISO/IEC. ISO/IEC 9899:1999(E): **Programming Languages - C** §7.19.1 para 1, 1999.

Autor correspondente: Dr. Guilherme Esmeraldo (guilhermealvaro@ifce.edu.br)

Utilizando Programação Orientada a Aspectos para Separação da Lógica de Negócio da Lógica de Suporte

Eduardo Rubens de Alencar Maia[1,2], Guilherme Esmeraldo[2]

1 - Faculdade Paraíso do Ceará (FAP), 2 - Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *campus* Crato.

Introdução

Com o avanço da computação e a demanda crescente por aplicações cada vez mais complexas, no desenvolvimento de software, o paradigma de programação orientado a objetos (POO) [1] sugiu para agregar mais recursos ao paradigma puramente estruturado. Com o uso de POO, características como modularidade, reusabilidade e modificabilidade, tornaram-se mais evidentes no ciclo de desenvolvimento de software [2]. Entretanto, a POO encontra desafios a superar, como a separação da lógica de negócio de uma determinada funcionalidade (código primário) da lógica de suporte (código secundário) [3].

A programação orientada a aspectos (POA) [3] é uma alternativa para resolver problemas encontrados pela POO e pelo paradigma estruturado, permitindo a separação de interesses do sistema e diminuindo a complexidade dos componentes. Com o uso da POA, informa-se que partes do código fonte da aplicação devem ser interceptadas pelo código dos aspectos, permitindo a solução de problemas como a separação da lógica de negócio da lógica de suporte [3]. Um problema recorrente no desenvolvimento de aplicações é a codificação para geração de arquivos de logs [4]. Esses arquivos registram eventos importantes definidos pela aplicação e, com eles, pode-se, por exemplo, realizar auditoria e diagnóstico [5].

Este trabalho propõe um modelo de extração e de registro de determinados tipos de dados de uma aplicação, com o uso de POA. Com isso, objetiva-se retirar a responsabilidade de registro dessas informações da lógica de negócio e atribuir a aspectos [3]. Assim, o desenvolvedor pode concentrar-se apenas no desenvolvimento da aplicação em si e, consequentemente, alcançar maior abstração, menor acoplamento e tornar a aplicação mais expansível.

Metodologia

Para este trabalho, o método para o desenvolvimento da pesquisa foi dividido em quatro etapas, que são: 1) Levantamento bibliográfico acerca da teoria e de frameworks que permitem acoplar programação orientada a aspectos a programas desenvolvidos com a abordagem de programação orientada a objetos; 2) Levantamento bibliográfico dos principais métodos de extração de logs de execução e de falhas, em diversos tipos de aplicações. Além disso, fez-se um estudo de ferramentas e de como estas implementam os diversos métodos de extração; 3) Análise e projeto de uma solução que implementa a abordagem proposta neste trabalho; 4) Definição de estudo de caso, onde é aplicada a abordagem proposta neste trabalho para a separação da lógica de negócio da lógica de suporte.

Resultados e Discussão

A aplicação proposta no estudo de caso, que consistiu de uma aplicação de automação comercial, foi desenvolvida considerando o método proposto neste trabalho. No estudo, a aplicação foi dividida em lógica de negócio e nos aspectos relacionados à extração dos logs. Dentre os tipos de informações registradas, estão os relacionados à lógica de negócio (e.g. dados de autenticação de usuários, como data/hora e *login*), bem como os relacionados às transações no sistema (e.g. tipos de transações mais frequentes e horários de maior frequência). Para avaliar a abordagem proposta, verificou-se a possibilidade de se adicionar novos módulos de extração de dados e registro de logs. Na solução proposta para o estudo de caso, o código fonte que realiza a extração e registros, desenvolvido com POA, é disjunto do restante, o que permitiu, consequentemente, um aumento da manutenabilidade e suporte à evolução do sistema.

Conclusões e Perspectivas

A cada dia as aplicações estão se tornando cada vez mais complexas. Programação orientada a objetos, apesar de ser muito utilizada para atacar este problema, através de modularidade, reusabilidade e modificabilidade, não dá suporte à separação de código principal de secundário e, com isso, abordagens complementares têm surgido. Uma delas é a programação orientada a aspectos. Neste trabalho, propôs-se o uso de POA para retirar obrigações secundárias do código fonte de uma aplicação, inserindo-as em aspectos.

Referências

[1] Cox, B. J. **Object oriented programming**. Addison-Wesley, 2008.
[2] Varejão, F. M. **Linguagem de programação: conceitos e técnicas**. Elsevier,2004
[3] Kiczales, G.; Lamping, J; Mehdhekar, A; Maeda, C; Lopes, C. V.; Loingtier, J; Irwin, J. **Aspect-Oriented Programming**. Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241. June 1997.
[4] Laddad, R. **AspectJ in action: practical aspect-oriented programming**. Vol. 512. Greenwich: Manning, 2003.
[5] Viega, J.; Vuas, J. **Can aspect-oriented programming lead to more reliable software?.** *Software, IEEE* 17, no. 6 (2000): 19-21.

Autor correspondente: Dr. Guilherme Esmeraldo (guilhermealvaro@ifce.edu.br)