Processando Consultas de Forma Eficiente Utilizando Paralelização em Placa de Vídeo

Lucas Tavares de Lima[1], Caio Moura Daoud e Rafael Freitas Schmid

Instituto Federal de Mato Grosso do Sul – Aquidauana , MS

lucassendsbullet@gmail.com[1], caio.daoud@ifms.edu.br e rafael.schmid@ifms.edu.br

Palavras-chave: Recuperação de Informação, Processamento de Consultas, Índices Invertidos, Sistemas de Busca, Paralelização de Código.

Introdução

Sistemas de busca são mecanismos que buscam informações relevantes para uma consulta em coleções de documentos. Esses sistemas estão presentes em muitos cenários, como em sites de vendas onde o sistema de busca retorna os produtos considerados mais relevantes para uma consulta do usuário. Outro exemplo é o popular sistema de busca da Google que retorna às páginas Web mais relevantes para um conjunto de termos consultado.

A popularização dos sistemas de busca e o constante crescimento de meios eletrônicos para armazenar informação motivam a busca por soluções que possam reduzir o custo para processar consultas. A paralelização do código entra nesse contexto como a estratégia que utilizaremos para otimizar o processamento da consulta. O código será dividido em partes independentes que serão processadas por diversas threads simultaneamente.

Metodologia

O objetivo principal do projeto é comparar as implementações sequenciais e paralelas de um processador de consulta. Para as versões paralelas serão utilizadas as ferramentas OpenMP [Leonardo e Ramesh 1998] e CUDA [NVIDIA 2011]. Ambos tratam-se de APIs para processamento paralelo, a primeira em CPU e a segunda nas placas de vídeo da NVIDIA.

Para computar a relevância dos documentos para a consulta adotaremos um dos modelos mais tradicionais na área de RI (Recuperação de informação), o Modelo de Espaço Vetorial proposto por Salton et al. [1974]. A implementação do processador de consultas usará a estratégia TAAT (Term-At-A-Time).

Análise e Discussão

Uma consulta é composta por um conjunto de termos, o processador de consultas calcula um peso para cada documento da coleção onde há ocorrência desses termos e então gera um ranking dos documentos para consulta. O índice invertido é uma estrutura adotada por muitos processadores de consultas. Essa estrutura mantém uma lista, chamada lista invertida, para cada termo da coleção de documentos. Cada lista armazena os IDs dos documentos onde o termo ocorre e a frequência do termo nos documento.

Termos	A FT	A FT	A FT	A FT
Instituto	(0,10),	(1,5),	(2,15),	(3,14)
Federal	(1,5),	(2,15),	(3,11)	
Aquidauana	(0,13),	(2,15),	(3,2)	

A - ID do documento
FT - (FT) Frequência do termo no documento

Figura 1 - Índice invertido

A Figura 1 ilustra a estrutura de um índice invertido ordenado por id dos documentos. Para cada termo há uma lista de documentos e a frequência que o termo aparece em cada documento, o primeiro termo tem ocorrência no arquivo 0 como frequência 10, no arquivo 1 com frequência 5, no arquivo 2 com frequência 15 e no arquivo 3 com frequência 14.

Na implementação sequencial, o programa percorrerá um a um os termos sendo pesquisados calculando um peso para cada documento. Como a arquitetura do dispositivo pode limitar o número de threads de uma implementação paralela eficiente, dividiremos as listas em blocos para serem processados pelas threads, que farão o cálculo dos pesos em paralelo.

Conclusão

Os processadores de consulta estão presentes em várias áreas de atuação e lidam com grandes quantidades de dados. Por isso, é importante que tenhamos processadores de consulta eficientes que além de responder com qualidade também sejam capazes de responder rapidamente.

Referências

Nvidia, CUDA (2011). Nvidia cuda c programming guide. *Nvidia Corporation.*

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE.*

Salton, G.; Wong, A. & Yang, C. S. (1974). A vector space model for automatic indexing. Relatório técnico, Ithaca, NY, USA

Robertson, S. E. & Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. Em *ACM SIGIR*, pp. 232--241