Paralelização da SVM em GPU

Henry E.L. Cagnini[1], Ana T. Winck[1], Rodrigo C. Barros[2]

[1] Universidade Federal de Santa Maria, Brasil
{henry,ana}@inf.ufsm.br
[2] Pontifícia Universidade Católica do Rio Grande do Sul, Brasil
rodrigo.barros@pucrs.br

Resumo. Máquinas de Vetores de Suporte (SVM) é um dos métodos mais eficazes para classificação de dados em aprendizado de máquina. Existem trabalhos que vêm buscando minimizar seu tempo de execução através da paralelização de trechos de seu código fonte, sobretudo com a utilização da *Graphics Processor Unit*, a GPU. Alguns destes trabalhos fazem uso do *framework* CUDA, ficando restritos a GPUs da fabricante NVIDIA. Este trabalho propõe a paralelização das Máquinas de Vetores de Suporte com a utilização do *framework* OpenCL, permitindo que a solução seja portável para diferentes GPUs. A etapa de treinamento do algoritmo foi otimizada através da paralelização de suas funções mais custosas. A solução final conseguiu aumentar com sucesso o desempenho das Máquinas de Vetores de Suporte em relação à versão sequencial.

Categorias e descrição dos tópicos: H.2.8 [**Gerenciamento de Banco de dados**]: Aplicações de Banco de dados; I.2.6 [**Inteligência Artificial**]: Aprendizagem

Palavras-chave: GPU, SVM, OpenCL

1. INTRODUÇÃO

A medida que o tamanho de uma base de dados aumenta, o poder computacional de um computador que execute um algoritmo de aprendizado de máquina sobre tal base também deve aumentar. Para problemas de *Big Data*, contudo, o simples aumento do poder de processamento da máquina não é suficiente para vencer a complexidade da geração de um modelo preditivo para tais volumes de dados. Para isto, novas abordagens de processamento de algoritmos precisam ser exploradas, tal como a paralelização de código fonte dos algoritmos de aprendizado na GPU.

Dentre os métodos de aprendizado de máquina passíveis de paralelização, as Máquinas de Vetores de Suporte (SVMs, do inglês *Support Vector Machines*) [Vapnik 1999] são atrativas sobre os outros devido à sua popularidade, conferindo-lhe uma diversidade de bibliotecas de código fonte. Um código fonte paralelo pode ser executado em diversos dispositivos simultaneamente, tal como máquinas em uma redes de computadores e *stream processors* (processadores paralelos) de placas gráficas (*Graphics Processing Unit*, ou simplesmente GPU). A GPU é mais atrativa para paralelização de SVMs sobretudo pela sua facilidade de acesso: a maioria dos computadores domésticos contemporâneos possui uma GPU, seja ela *onboard* (integrada ao *chip* da CPU) ou *offboard* (ligada à placa mãe por um barramento específico para tal).

Neste trabalho, propõe-se paralelizar a fase de treinamento de SVMs na GPU de forma a melhorar seu desempenho em relação à versão executada sequencialmente. O *framework* que será utilizado é o OpenCL [KHRONOS 2012], e o código fonte que representa a SVM é o LIBSVM [Chang and Lin 2011].

---

2. MOTIVAÇÃO

Desde a publicação da LIBSVM [Chang and Lin 2011], as SVMs têm se popularizado tanto como aplicação de aprendizado de máquina como objeto de estudo. Com um conjunto de ferramentas auxiliares (como pré-processador de bases de dados e múltiplas escolhas de função de *kernel*), a LIBSVM é disponibilizada em várias linguagens de programação. Essa robustez faz com que diversos trabalhos se dediquem a aumentar seu desempenho, através de técnicas como a paralelização de código fonte na GPU. Por exemplo, [Catanzaro et al. 2008] obtém um *speedup* de até 138×, ao custo de uma piora na acurácia dos modelos preditivos gerados; [Athanasopoulos et al. 2011] obtém *speedups* mais modestos, mas sem sacrificar a acurácia. Ambos utilizam o *framework* CUDA [NVIDIA 2014] para escrita do código fonte paralelo.

Deve-se perceber que CUDA opera apenas sobre GPUs da fabricante NVIDIA, impossibilitando que outras GPUs possam executar as versões otimizadas apresentadas pelos trabalhos citados. Para que mais GPUs possam se beneficar da execução da LIBSVM otimizada, é preciso utilizar um *framework* que não seja específico à um fabricante, como o OpenCL [KHRONOS 2012], configurando-se este como objetivo principal deste trabalho.

3. METODOLOGIA

A primeira etapa do trabalho diz respeito à identificação das funções mais executadas na fase de treinamento do processo de classificação da SVM e posterior adaptação para execução na GPU. Para as bases de dados apresentadas na Tabela I [Bache and Lichman 2013], a função de *kernel* – responsável por projetar os dados do espaço de atributos para um nível dimensional superior – é a função que mais demanda recursos computacionais. A função de *kernel* utilizado pelo código fonte é dependente da tarefa realizada pela LIBSVM; como apenas a classificação binária é levada em consideração neste trabalho, a função de *kernel* utilizada pelo código fonte é a *Radial Basis Function* (RBF, na sigla em inglês), sendo esta adaptada para execução na GPU. Contudo, isto não impossibilita a paralelização de outras funções de *kernel*.

O computador no qual o código fonte foi desenvolvido – e subsequentemente os tempos de execução das versões original e paralelizada da LIBSVM foram aferidos – possui placa mãe ASUS M5A88-M, processador AMD FX 4100, 8GB de memória RAM e placa de vídeo AMD R7 260X.

Característica	Bases de dados[1]				
	a1	a6	a7	a8	a9
Classes	2	2	2	2	2
Atributos	123	123	123	123	123
Número de instâncias de treino	1.605	11.220	16.100	22.696	32.561
Número de instâncias de teste	30.956	21.341	16.461	9.865	16.281

Tabela I. Bases de dados utilizadas na otimização do código fonte.

4. IMPLEMENTAÇÃO

Em relação ao código fonte original da LIBSVM, duas modificações foram realizadas: nova representação da base de dados e substituição das funções auxiliares da *kernel* RBF. A nova representação visa aproveitar-se de uma particularidade das bases de dados utilizadas neste trabalho: uma vez que todos atributos preditivos são binários, é permitido que uma instância possa ser representada sem perda de informações como um vetor de bits. Essa modificação é particularmente vantajosa por proporcionar

[1]Disponíveis em `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

um novo processamento do produto escalar (utilizado pela *Radial Basis Function*) entre instâncias: uma operação AND entre os vetores de bits seguida de um somatório dos bits ativados produz o mesmo resultado que o código fonte original. Essa nova representação, além tornar a computação do produto escalar mais rápida que a original, faz também com que o espaço em memória principal necessário para armazenar a base de dados seja reduzido em 98,37%: se na versão original 8 bytes são necessários para armazenar o valor de um único atributo, com a nova representação cada byte armazena até 8 atributos.

A substituição das funções auxiliares, por sua vez, serve ao único propósito de trocar as funções de bibliotecas auxiliares ao código fonte original por funções nativas do OpenCL. Apesar disso não alterar o funcionamento da função de *kernel* (de fato, ele apenas é transferido do código fonte original para o código fonte do OpenCL), as funções auxiliares são responsáveis por uma depreciação na acurácia do modelo preditivo gerado para cada base de dados (mais detalhes são apresentados na Seção 5).

5. RESULTADOS OBTIDOS

Foram geradas duas versões otimizadas da LIBSVM: ambas contém a nova representação de dados, porém uma é executada sequencialmente na CPU e outra possui processamento paralelo na GPU[2]. O aumento no tempo de execução (denotado por um *speedup* de ordem inferior a 1× na Tabela II) para conjuntos de dados pequenos dá-se pela necessidade de conversão da representação de dados (presente em ambas versões) e pelo custo de alocação e envio de dados à memória da GPU (presente apenas na versão paralela). Contudo, este custo é vencido para conjuntos maiores, onde a versão paralela obtém um desempenho melhor que as versões original e sequencial. O gráfico da Figura 1 sugere ainda uma tendência da versão paralela a executar mais rapidamente com conjuntos de dados maiores que a9a.

Dataset	Original		Otimizado sequencial			Otimizado paralelo		
	Acurácia	Tempo	Acurácia	Tempo	*speedup*	Acurácia	Tempo	*speedup*
a1a	83,59%	634,23ms	83,59%	692,87ms	0,91×	83,59%	2.753,29ms	0,23×
a6a	84,17%	17.327,07ms	84,17%	14.229,50ms	1,22×	75,87%	18.238,22ms	0,95×
a7a	84,56%	34.278,50ms	84,58%	28.242,03ms	1,21×	76,17%	28.683,22ms	1,19×
a8a	85,01%	66.831,44ms	85,01%	51.730,57ms	1,29×	76,33%	40.735,22ms	1,64×
a9a	84,82%	133.877,07ms	84,82%	103.825,86ms	1,29×	76,38%	80.744,78ms	1,66×

Tabela II. Tempos de execução e acurácia dos modelos preditivos gerados das versões original, otimizada com processamento em CPU e otimizada com processamento em GPU do código fonte da LIBSVM. O *speedup* é comparado à versão original para cada base de dados.

Os modelos preditivos da versão otimizada em GPU obtêm uma piora de em média 6,76% na acurácia. Essa piora decorre de uma função auxiliar (*exp*, responsável por calcular e^x) utilizada pela função de *kernel*: enquanto na versão otimizada para CPU esta função é proveniente da mesma biblioteca de código fonte da versão original, na GPU ela é provida pelo OpenCL, que possui precisão limitada para operações de ponto flutuante. Note que, se a versão em GPU da função *exp* tivesse precisão similar à da CPU, os resultados em termos de acurácia seriam os mesmos para todas as versões.

6. CONSIDERAÇÕES FINAIS

As versões otimizadas do código fonte da LIBSVM são capazes de aumentar o desempenho em relação à versão original. Apesar da piora da acurácia para a versão executada em GPU e do *speedup* ser pequeno para bases de dados pequenas, algumas modificações podem ser realizadas de forma a aprimorar seu funcionamento:

[2]Disponível em `https://bitbucket.org/hcagnini/parallel_libsvm_train`.

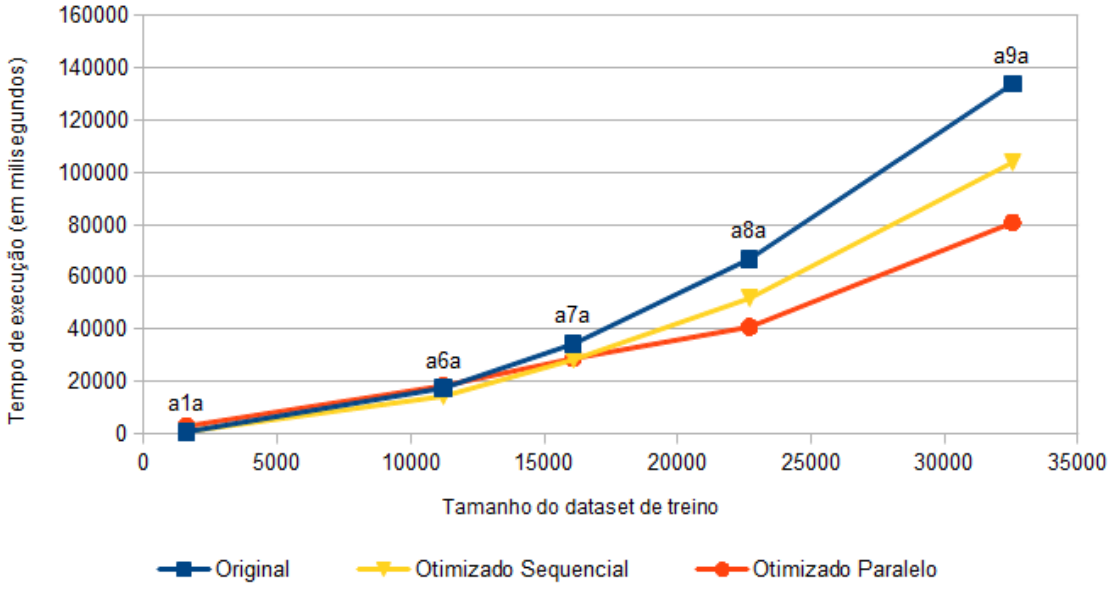

Fig. 1. Comparação entre as versões original, otimizada com processamento sequencial e otimizada com processamento paralelo do código fonte da LIBSVM. À medida que o tamanho da base cresce, o aumento de desempenho (área entre as curvas de cada versão da LIBSVM) também cresce para as versões otimizadas.

(1) Implementação própria das funções auxiliares à função de *kernel*, substituindo as providas pelo OpenCL e garantindo uma acurácia constante dentre as três versões (original, otimizada em CPU e otimizada em GPU) de código fonte;
(2) Paralelização de outras funções de *kernel*, possibilitando que tarefas como regressão e classificação multiclasse sejam abordadas;
(3) Multiplicação de matrizes para computação do produto escalar entre instâncias, possibilitando que bases de dados com atributos contínuos possam ser utilizadas;
(4) Utilização de bases de dados maiores, tanto em termos de dimensionalidade como em número de instâncias, com o intuito de escrutinar melhor o ganho de desempenho da versão em GPU.

REFERÊNCIAS

Athanasopoulos, A., Dimou, A., Mezaris, V., and Kompatsiaris, I. GPU Acceleration for Support Vector Machines. In *Proceedings of 12th International Workshop on Image Analysis for Multimidia Interactive Services*. Delft, Holanda, 2011.

Bache, K. and Lichman, M. UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>, 2013.

Catanzaro, B., Sundaram, N., and Keutzer, K. Support Vector Machine Training and Classification on Graphics Processors. In *Proceedings of the International Conference on Machine Learning*. Helsink, Finlândia, pp. 104–111, 2008.

Chang, C.-C. and Lin, C.-J. LIBSVM: A library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology* vol. 2, pp. 27:1–27:27, 2011.

KHRONOS. The OpenCL 1.2 Specification. <http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf>, 2012.

NVIDIA. CUDA Toolkit Documentation. <http://docs.nvidia.com/cuda/>, 2014.

Vapnik, V. An overview of statistical learning theory. *IEEE Transactions on Neural Networks* vol. 10, pp. 988–999, 1999.