Parallelization of Adaboost Algorithm on Intel MIC Architecture

Haibiao Luo, Haojie Yuan, Shendong Cheng, Ying Li, Feng Yuan and Mingzhu Wei
Institute of Software Application Technology, Guangzhou & CAS, Guangzhou, China

***Abstract*—The Adaboost algorithm plays important role in many machine learning applications. But the computation cost is real expensive when the candidate features are in large amount. In this paper, we introduce a parallel strategy of the Adaboost algorithm on Intel CPU+MIC system, where Intel MIC works as a coprocessor. Open MP directive was used to parallelize the program in both CPU and MIC. The paper achieved a speedup of 5.2 on CPU+MIC with respect to CPU alone.**

Keywords- Intel MIC; adaboost; parallel computation

I. INTRODUCTION

The Adaboost algorithm [1] is an important method in ensemble learning. The idea behind it is to build an arbitrarily strong classifier by combining multiple weak classifiers whose accuracies are slightly better than random guessing. The algorithm can be applied to almost all of the prevalent machine learning algorithms to improve the accuracy of prediction. However, it takes enormous computation power to achieve reasonable precision from a large training dataset and a large feature set.

Intel Many Integrated Core Architecture (Intel MIC) is Intel's latest effort of Intel toward exascale computing. Intel MIC is a coprocessor of up to 61 processing cores and has been used in the world's fastest supercomputer Tianhe II. It is nature to use Intel MIC architectures to speed up the Adaboost algorithm. This paper studied the parallelization paradigm of the Adaboost algorithm that best suited for Intel MIC.

II. RELATED WORK

Since its importance and enormous computing cost of the Adaboost algorithm, there have been a number of researchers investigating in its parallelization. Merler etc. [2] developed a P-Adaboost algorithm based on weights dynamics. Lazarevic and Obradovic [3] proposed a parallelized Adaboost algorithm for distributed homogeneous database that cannot be merged at a single location. These parallel algorithms changed the structure of the Adaboost algorithm to make it compatible with a parallel framework. The computing results changed slightly as well. Some researchers parallelized the Adaboost algorithm without modifying the original Adaboost training process. Galtier etc. [4] developed a parallel framework based on MPJ and the JavaSpace for PC clusters. Zeng etc. [5] proposed a hybrid parallel framework by MPI/OpenMP on distributed shared memory systems. Krpec etc. [6] trained the face dataset of MIT CBCL by the Rank-Adaboost algorithm. It was parallelized on CPU+GPU architecture and obtained up to 20 times speedup.

III. PARALLELIZATION OF THE ADABOOST ALGORITHM

A. The Sequential Adaboost Algorithms

The Adaboost training process is intrinsically sequential [3]. The weights of the training set are updated at each step. This leads to strong dependency between iterations. The sequential Adaboost algorithm is given as [1]:

Input: Given number of iteration T, weak classifier algorithm hj, training set $S=\{(xi, yi)\}$, where $i = 1, \ldots, n$, $xi \in X$ and $yi \in \{0, 1\}$, n is the number of positive samples, m is the number of negative samples.

Initialize $t = 1$, weight vector $Wt, i = 1/2m$, 1/2l for $yi = 0, 1$ respectively, $i = 1 \ldots m$

while $t \leqslant T$ do

1) Normalize the weights: $W_{t,i} = W_{t,i} / \sum_{i=1}^{m} W_{t,j}$

2) For each feature, j, train a weak classifier hj

3) Calculate error of hj: $\varepsilon_j = \sum_i W_i \left| h_j(x_i) - y_i \right|$

4) Choose the weak classifier ht, with the lowest error ε_j

5) Update the weights: $W_{t+1,i} = W_{t,i} \beta_t^{1-e_i}$, where $ei = 0$ if example xi is classified correctly, $ei = 1$ otherwise, and $\beta_t = e_t / (1 - e_t)$

Output:

The final strong classifier is:

$$h(x) = \begin{cases} 1, \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, \text{otherwise} \end{cases}$$, where $\alpha_t = \log \frac{1}{\beta_t}$

FIGURE I. THE SEQUENTIAL ADABOOST ALGORITHM

In the above algorithm, step 2) the weak classifier is designed to select a single feature which best separates the positive and negative examples. The weak classifier determines an optimal threshold classification function, such that the minimum number of examples are misclassified. A weak

classifier $h_j(x)$ thus is a function of a feature f_j, a threshold and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The weak classifier training process of the Adaboost algorithm counts about 80~90% of the whole computing time. Because it is expensive to enumerate all the features possible. Therefore, it is very necessary to parallelize the weak classifier training process to make the Adaboost algorithm run faster on modern computers.

IV. PARALLELIZATION OF THE ADABOOST ALGORITHM ON INTEL MIC

The MIC architecture is an x86-based many-core processor architecture based on small in-order cores that uniquely combines full programmability of Intel CPU with compute-throughput and memory bandwidth capabilities of modern GPU architectures. Each core is a general-purpose processor, which has a scalar unit based on the Pentium processor design, as well as a vector unit that supports 16 32-bit float or integer operations per clock. The MIC architecture has two levels of cache: low latency L1 cache and larger globally coherent L2 cache that is partitioned among the cores. Cores are connected through up to 16 GDDR5 transmission channels and can support 5.5GT/s transmission speed theoretically. It has extended vector processing units and larger memory bandwidth. MIC uses Symmetric Multi-Processing architecture (SMP). One main advantage of MIC is that programs on CPU can run on MIC without changing the main body of the programs.

MIC and CPU can work together in different ways: naïve mode of treating MIC as an equal partner of CPU, or offload mode of treating CPU as host and offload heavy duty work to MIC. Naïve mode requires delicate workload balance between big cores and little cores. Currently the under development topology-aware MPI interface MVAPICH2 works better in this mode. Since the heaviest job of the Adaboost algorithm is the weak classifier training and other jobs occupies only 10~20% of the computing time. We used the offload mode so that weak classifier training is throw to MIC and the other computing steps remain in CPU.

After the data are input, the weights and image data are transmitted to MIC to compute step 2) weak classifier training. The transmission occurs during computation of normalization of weights to overlap computation and communication. After that the normalized weights are transmitted. The results will be transmitted back to CPU to continue the next step.

A. Parallelization of Integral Images

Haar-like features are used in the Adaboost algorithm to construct strong classifiers [1]. A haar-like feature is represented by adjacent rectangular regions at a specific location in a detection window of a image. Pixel intensities in these regions are sum up and the difference between them is calculated. This difference is then used to categorize subsections of an image. In our study we used 4 types of haar-like features (Fig. 2). Haar features can be calculated from integral images through the integral values of the vertices of the rectangular regions. Since there is no dependency between different samples, integral images can be computed in parallel by multithreading of OpenMP directive directly.

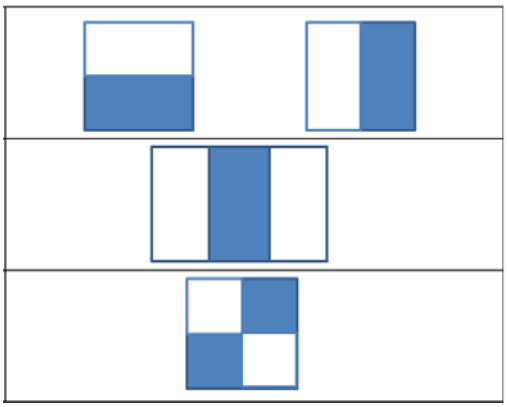

FIGURE II. EXAMPLES OF HAAR-LIKE FEATURES

B. Parallelization of Quick Sort of the Evaluated Features

After integral images are calculated, the second step is to sort the features according to their value. Here we used Parallel Sorting by Regular Sampling method (PSRS) [7] to sort the features. Every image sample has tens of thousands features. It is expected that it is very time consuming.

PSRS was parallelized by OpenMP on MIC. The parallel algorithm can be summarized in the following:

It is assumed that the total number of sorted elements is n, and total number of cores in the system is p. Firstly, n elements are evenly divided into p groups. Then each core named pi (i=0…p-1) sorts its own n/p elements using radix sort. In order to divide evenly, the master core selects p-1 pivots from each group. Then divide the array into p parts according to the pivots and exchange globally the corresponding elements of every group. Finally merge sort these p segments using p–way merge and produce the sorted array.

In MIC, there is 61 cores and one will be chosen as the master core. Open MP directive was used. Every core has its own private memory and 2 threads were used in every core. Auto vector compilation option was turned on in Intel C++ compiler to vectorize the loops in the code.

C. Parallelization of the Best Weak Classifier

The best weak classifier was chosen with the lowest weighted error rate. The process can be parallelized by dividing the feature set into a number of sub sets and distribute the sub sets to threads of MIC. The winner of each sub set will compete in the master thread and find out the best weak classifier.

D. Performance Evaluation

We now evaluate the performance of the parallel algorithm of the Adaboost algorithm on CPUs and an Intel MIC architecture. The CPU is Intel i5-3470 of 3.20GHz; Intel MIC is Intel Xeon Phi E5-2650 0 @2.00GHz, 7110P, 8GB GDDR5, 61 cores.

The training dataset used in our experiment is a car repository collected from internet and real time road and parking lot surveillance video in the city. It contains 2320 positive samples and 3450 negative samples of unified resolution 18×18. There are 7550 negative samples randomly taken from internet. To make up the insufficiency of the samples, we did left and right mirror transformation to the positive samples, and did left and right, up and down mirror transformation to the negative samples. The tested feature set contains 22710 haar-like features. The training loop was executed 1000 times in total.

The sequential version of the Adaboost algorithm was run on a CPU core while the MIC version was run on a CPU core + a MIC. We developed the parallelized training program carefully and made sure the parallel MIC version produced exactly the same strong classifier as the sequential version did.

The following table illustrated the computing time and speedup of the sequential version and parallel MIC version of the Adaboost algorithm. The MIC version was slower than the sequential version when the number of samples is 5120. This is because the data transmission between CPU and MIC occupied relatively large amount of time compared to the computing time in MIC when the data is not large enough. As more samples were input, MIC started to show better performance than CPU. The MIC version achieved the best speedup of 5.2 at the top of the sample size.

TABLE I. COMPUTING TIME OF DIFFERENT SAMPLES

Sample Number.	Sequential (S)	MIC (S)	Speed Up
5120	26.3	27.1	0.97
7680	39.2	32.7	1.2
10200	54.8	30.4	1.8
12800	72.1	28.8	2.5
15300	89.3	27.9	3.2
20480	134	31.2	4.3
25600	172	33.0	5.2

V. CONCLUSION

This paper analyzed the Adaboost algorithm and proposed a parallel strategy on Intel MIC architecture. The experiment result indicated that a CPU + a MIC coprocessor can increase the computing speed by 5.2 times. From the trend of speedup, one can see that the speedup of the MIC version still has potential to increase. Our future work will be to increase sample size to see where the speedup limit is.

ACKNOWLEDGEMENT

Authors acknowledge support from the international cooperation project of Nansha District (NO. 2014GJ037), the Industry-university-research Cooperative Innovation major projects of Guangzhou (NO. 201508010009, NO. 201508030026), the Industry-university-research Cooperative Project of Dongguan (NO. 2014509105105), National Science-technology Support Plan Project (NO. 2015BAK36B06).

REFERENCES

[1] P. Viola, M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features,” Proc of the Conf on Computer Vision and Pattern Recognition, pp.511-518, 2001

[2] S. Merler, B. Caprile, C. Furlanello. “Parallelizing adaboost by weights dynamics,” Computational Statistics & Data Analysis, 51(5):2487-2498, 2007.

[3] A. Lazarevic, Z. Obradovic. “Boosting algorithms for parallel and distributed learning,” Distrib Parallel Databases 11(2):203–229, 2002.

[4] V. Galtier, S. Vialle, and S. Genaud. “Implementation of the adaboost algorithm for large scale distributed environments,” Comparing javaspace and mpj, International Conference on Parallel & Distribut Systems, 655-662 Shenzhen, China, 2009.

[5] K. Zeng, Y. Tang, F. Liu. “Parallization of Adaboost Algorithm through Hybrid MPI/OpenMP and Transactional Memory,” Parallel, Distributed and Network-Based Processing , pp. 94-100, 2011.

[6] Jaromír Krpec, Martin Němec. “Face Detection CUDA Accelerating,” The Fifth InternationalConference on Advances in Computer-Human Interactions (ACHI), pp. 155-160, 2012.

[7] Shi H, Schaeffer J. “Parallel sorting by regular sampling,” Journal of Parallel and Distributed Computing. Vol.14. 1992.

[8] N. Satish, C. Kim, etal. “Fast Sort on CPUs, GPUs and Intel MIC Architectures,” Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010.