Performance Analysis of Machine Learning Algorithms With Different Thread and Data Affinity Approaches

Arthur Mittmann Krause, Eduardo H. M. Cruz,
Matheus S. Serpa, Philippe O. A. Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informatica
{amkrause, ehmcruz, msserpa, navaux}@inf.ufrgs.br

Abstract

The popularity of machine learning algorithms is growing together with their data sets, and so is the need to find ways to accelerate them. Data and Thread mapping is an useful technique to improve performance in those applications. Placing threads that communicate more with each other closer in the memory hierarchy can be beneficial to performance by reducing the latency in the communication, while placing threads that use intensively the same resources in separate cores can reduce competition between them. As a main contribution, we analyze the performance impact of data and thread mappings on popular machine learning algorithms. We managed to reduce the execution time by 87% in some situations using only thread mapping, mostly due to reduction in QPI traffic.

1. Introduction

Machine learning is one of the most rapid growing subjects in computer science, helping nearly all fields of knowledge by bringing a new approach to understanding large data sets. The abundance of training data and techniques currently available are allowing many companies to develop revolutionary products such as self driving cars, personal assistants and precise product recommendations tools. Usually, Machine Learning algorithms are accelerated through cloud services, ASIC or GPU based environments. However, the use of CPU systems can also be interesting because of their flexibility, memory capacity and scalability, and with the new many-core processors, CPUs can be a better choice for largely parallel workloads, which were previously dominated by GPUs.

Threads in multithreaded applications such as Machine Learning algorithms usually share data, forcing it to be moved through the system either by the inter-chip or intra-chip interconnections, depending on the memory hierarchy and the threads location [3]. It means that threads that communicate a lot with each other will experience lower latency from communication if placed closer in the chip. On the other hand, the bandwidth is also shared, which means that threads will compete for it, so threads that are communicating more should be placed far apart from each other, in order to not compete for the interconnections. On NUMA systems, ubiquitous in HPC, data present in the main memory can be closer or farther away, depending on the node it is stored and the physical CPU the thread that is accessing it is running on. Reading data from a distant memory bank results in higher latencies and more traffic in the interconnections.

This paper aims to analyze the performance variations of popular machine learning algorithms, as well as the traffic on the interconnections, when executed using different combinations of thread and data mappings on a NUMA machine, to understand if these applications can be accelerated by a smarter placing of its threads and data and for what reason.

2. Related Work

Mazouz et al. analyses the speedup of the SPEC OMP benchmark suite in different machines using various thread affinity strategies. However, no machine learning applications are tested [5]. Bottesch et al. introduces a new method to approximate Euclidian distances in order to significantly accelerate the k-Means algorithm [1]. Morais accelerates the k-Nearest Neighbors algorithm through the use of the Fast Fourier Transform and removing unfavorable training vectors [6]. Both papers provide application specific optimizations, not easily usable in other programs. In this context, this paper aims to analyze the performance implications of different thread and data mappings on popular machine learning algorithms.

3. Machine Learning Algorithms

Machine Learning techniques are used to classify elements in large datasets. There are two approaches that are used: supervised and unsupervised [8]. A supervised model aims to classify elements in previously defined categories. In order to achieve this, a previous dataset of values already classified are needed to train the algorithm. This allows the model to know the classes it should attribute to the elements in the data set and the expected features the elements on each class should have. For example, given a table containing the species, length, width, color and quantity of petals of various flowers, a supervised model that has been fed with this data can predict the species of other flowers given their petals characteristics.

Unsupervised approaches are used to group elements in generic clusters. Those algorithms receive some data and cluster the points that have similar features. They are better suited for cases where there is little or nothing known about the dataset and it is wanted to detect patterns within the data. Using the previous example, those models would be fed with just the latter dataset, containing unclassified flowers with the features of their petals, and label them according to what cluster it was found to be part of. The number of clusters can be predefined or variable depending on the technique used. The clusters formed may or may not be equivalent to the original species of the flowers.

In this work, four of the most popular Machine Learning algorithms, two being supervised and the other two unsupervised methods, were used for testing the impact on performance and QPI traffic of thread and data affinity policies. The supervised methods were k-Nearest Neighbors and Back Propagation Neural Network, and the unsupervised were k-Means and Streamcluster. In k-Nearest Neighbors, an element receives the label that is the most frequently occurring between its k-Nearest Neighbors, where k is a number defined beforehand. Back Propagation is an algorithm to train a neural network. The technique is to change the weights of each neuron in the network multiple times in order to minimize the error between the actual output and the desired output of the training set. Once the error is within the tolerable range, the training is complete and the network should be capable of classifying its inputs from outside the training set [7]. In k-Means, a set of k initial centers is generated in the same multidimensional space as the data set one wishes to cluster. The points are then clustered according to their nearest center. New centers are then calculated for each new cluster, and the process is repeated multiple times [4]. The Streamcluster finds a predetermined amount of centers in order to cluster a stream of points in a way that minimizes the sum of the distances between each point and its center. The applications tested are the OpenMP implementations present in the Rodinia Benchmark Suite version 3.1 [2]. We inverted the main loops in k-Nearest Neighbor in order to allow it to run with more than 16 threads.

4. Methodology

The applications were executed using different thread and data mappings. The thread affinity was configured through Intel Compiler's OpenMP runtime library predefined mappings. Applications were executed with compact, scatter and no affinity. In compact, the system tries to place the threads as physically close as possible, meaning that the communications between neighbor threads will have less latency, but there will be more competition for resources. With scatter, the threads are placed as far as possible, resulting in more latency from communications, but more resources such as caches, bandwidth and functional units are available. The memory affinity was set using numacontrol. With the interleave policy, memory will be allocated using round robin on nodes, meaning that the data will be spread evenly between the memory banks. With membind, all the data is present in the same node. With no policies set in numacontrol, the default NUMA balance policy of the Linux kernel is active. We also tested the applications with the kernel balancing turned off.

Experiments were done on a machine with four Intel Xeon X7550 processors and 128 GB of DDR3 RAM, being 32 GB for each NUMA node. The applications parameters were selected in order to have a similar execution time and memory footprint. The results shown are based on the mean of six executions. More repetitions were not done because the variance was low and due to time constraints.

5. Results

Figure 1 shows the percentage variation of the execution time with different combinations of thread and data mappings, for different amount of threads. Figure 2 shows the percentage variation of the QPI traffic measured using the PCM tool. Each value is calculated from the mean of 6 executions, having as the base point the values for the default thread mapping policy of the Linux kernel and the automatic NUMA memory balancing turned off, for the sake of finding the relative improvement of different mapping combinations in relation to no specific mapping at all. It is possible to observe significant performance improvements with the use of data and thread mappings, specially when using 8, 16 or 32 threads, achieving in some cases more than 87% of reduction on the execution time. In those cases, the thread mapping appeared as the most important factor, while data mapping was less relevant.

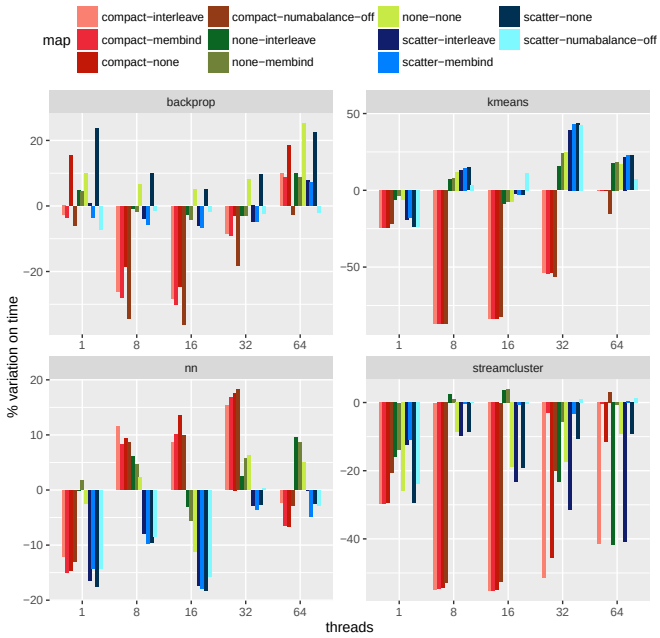

Figure 1. Percentage variation of the execution time

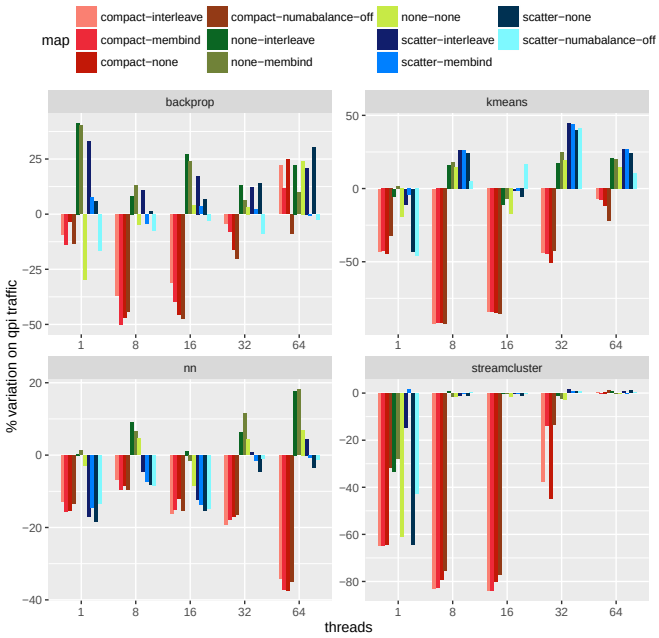

Figure 2. Percentage variation of the QPI traffic

The different combinations of thread and data mappings showed different results depending on the application and even thread amount. The compact thread affinity policy yielded the best results on most cases by pinning the threads closer physically and reducing the communication penalties. On k-Means, with 8 and 16 threads, the reduction in execution time was more than 87%, while the QPI traffic was reduced almost equally. A lesser improvement is observed with 32 threads and almost none with 64. Placing the threads closer was also interesting for the Backpropagation with 8, 16 and 32 threads, as well as for the Streamcluster with every thread amount but 64. On the other hand, in the Nearest Neighbors algorithm, where there is little communication between the threads, the scatter policy resulted in better performance, specially with 8 and 16 threads, presumably because there are effectively more physical cores at use, so the competition between resources such as cache lines or functional units was reduced. The similar QPI traffic with compact and scatter mappings on these cases reinforces this conclusion.

Different memory mappings yielded only marginal gains comparad to thread mappings in mosts cases. Nonetheless, With all the 64 cores in use on Streamcluster, the memory interleave policy provided a reduction of more than 40% of the execution time, while the thread mapping policies had no significant impact. There is no reduction in total QPI traffic in this case, because the memory loaded is still the same and the communication between threads is not improved by different mappings when all cores are in use.

As it can be seen by analyzing the Pearson correlation coefficient between the QPI traffic and execution time of the applications when varying the data and thread mappings, as shown in Figure 3, the performance was improved due to the reduction in the QPI traffic in most cases. The exceptions were the Streamcluster with 64 threads, where the QPI traffic didn't change, and the k-Nearest Neighbors, where the performance improvement was due to less sharing of resources and not less traffic on the interconnections.

6. Conclusions

Thread and data mappings were found to be effective ways to improve performance of the popular Machine Learning algorithms tested, depending on the applications' communication pattern. We managed to observe an 87% reduction on execution time for k-Means in some cases, and an 82% reduction in QPI traffic for Streamcluster, only with thread mapping. With the interleave memory mapping, reductions as big as 40% on execution time were achieved on Streamcluster when all 64 threads were running. Reductions on execution time were often coupled with similar reductions on QPI traffic. Performance could be further

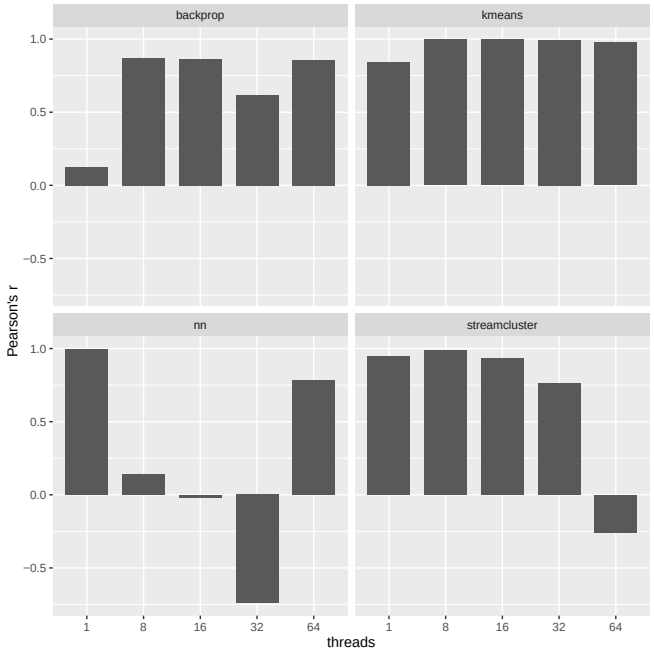

Figure 3. Pearson's r of the relationship between execution time and QPI traffic

improved by using more specific data and thread mappings or an online mechanism. Future work will analyze the impact of data and thread affinity on manycore architectures.

7. Acknowledgements

This research received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E project, grant n.o 689772. It was also supported by Intel under the Modern Code Project.

References

[1] T. Bottesch, T. Bühler, and M. Kächele. Speeding up k-means by approximating euclidean distances via block vectors. In *International Conference on Machine Learning*, pages 2578–2586, 2016.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 44–54, Washington, DC, USA, 2009. IEEE Computer Society.

[3] M. Diener, E. H. Cruz, P. O. Navaux, A. Busse, and H.-U. Heiß. Communication-aware process and thread mapping using online communication detection. *Parallel Computing*, 43:43–63, 2015.

[4] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[5] A. Mazouz, D. Barthou, et al. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of spec omp applications on intel architectures. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 273–279. IEEE, 2011.

[6] J. P. d. Morais Neto et al. Aceleração de uma variação do problema k-nearest neighbors. 2014.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[8] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.