Accelerating Nyström Kernel Independent Component Analysis with Many Integrated Core Architecture

Lei Shan(✉), He Wang, Weixia Xu, Canqun Yang, and Minxuan Zhang

Institute of Microelectronics, Institute of Software, College of Computer, National University of Defence Technology, Changsha 410073, Hunan, China
ssll2999@163.com

Abstract. Kernel independent component analysis (KICA) penalizes the correlations among components in a reproducing kernel Hilbert space (RKHS) and performs well in many practical tasks such as speech separation due to its robustness on varying source distributions. Recently, Nyström-KICA (NKICA) incorporates a low-rank approximation and low-complexity sampling method to reduce the computational complexity of KICA. In this paper, we show that the computational complexity of NKICA can be further decreased by implementing the algorithm on the many integrated core (MIC) architecture to meet the requirement of large data processing. Particularly, we parallelize the critical segments with the OpenMP technology and perform the intensive matrix manipulations on a MIC coprocessor. This MIC-based approach has been evaluated on both simulated dataset and the TIMIT dataset. The experimental results confirm the efficiency of our implementation of NKICA on the MIC architecture, and show that it achieves a consistent speedup rate of around 10 on average, and of 12.3 at best, comparing with that performed on single CPU.

Keywords: Kernel ICA · Nyström method · MIC · OpenMP · MKL

1 Introduction

Independent component analysis (ICA) [1] is a signal processing technique of recovering sources from observation data. The observations are linear combinations of statistic independent sources. ICA aims to find out the latent independent components with a set of observations of random variables, which has been widely used in practice, e.g., blind source separation, speech separation and feature extraction [2,3], etc. The traditional ICA algorithms were based on objective functions defined in terms of expectations of a single fixed nonlinear function, which makes them suitable for some specific problems. Kennel ICA (KICA, [4]) define the objective function in a reproducing kernel Hilbert space (RKHS), and make use of the "kernel trick" to search over this space efficiently. The use of a function space makes it possible to adapt to a variety of sources and thus makes KICA algorithm more robust to varying source distributions.

W. Xu et al. (Eds.): NCCET 2016, CCIS 666, pp. 168–176, 2016.
DOI: 10.1007/978-981-10-3159-5_16

Although KICA algorithm is very effective, the high computational complexity prohibits it form practical applications [5]. The kernel matrices get larger and larger, and the computational complexity has a cubic growth when the scale of data to be processed increases. A usual solution to this problem is constructing low-rank approximations of the kernel matrices because the spectra of kernel matrices decays rapidly [6]. Recently, Wang et al. use Nyström [8] method to construct such low-rank approximation termed Nyström KICA (NKICA). NKICA is proposed to solve the computation problem of prior incomplete Cholesky decomposition KICA (IDC-KICA). With the increase of the scale of data, both time and space complexities of ICD-KICA are unacceptable. In contrast to ICD method, the Nyström method uses a low-complexity sampling technique [7]. NKICA cuts down the computational complexity of KICA effectively, from $O(m^3N^3)$ to $O(mM^2N)$.

Although NKICA reduces the computational complexity effectively, with the scale of practical data increases, the computation complexity of algorithm grows quickly. It prohibits NKICA form big data applications. To solve the problem, we consider hardware implementation to accelerate computation of NKICA. Nowadays, various accelerators developed quickly, including Intel MIC coprocessor [9,10] and general purpose computation on graphics processing units (GPGPU) [12,13], etc. With the help of the accelerators, highly parallel computations can be accelerated easily.

Intel MIC architecture provides many computational cores, and conveniently exploits parallelism with technology such as OpenMP and TBB. On the other hand, MIC is very suitable for matrix manipulations, which are the most manipulations in NKICA. In this paper, we implement the NKICA on Intel Many Integrated Core (MIC) architecture to reduce the computational complexity of KICA to meet the requirement of large data processing. Particularly, we parallelize the critical segments with the OpenMP technology and perform the intensive matrix manipulations on a MIC coprocessor. Experiments on both simulated dataset and the TIMIT dataset confirm its efficiency.

The remainder of the paper is organized as follows. Section 2 introduces related work including Nystrom Fast KICA and nd the Intel Many Integrated Core architecture. Section 3 discusses how to use MIC to accelerate Nystrom KICA algorithm and Sect. 4 presents the experimental results. Section 5 concludes the paper.

2 Background

2.1 Kernel Independent Component Analysis

Independent component analysis (ICA) [3] aims to recover a latent random vector $s = (s1; \ldots; sm)$ from observations of m unknown linear functions of that vector. The components of s are assumed to be mutually independent, and their distributions are usually assumed unknown. x is modeled as

$$x = As, \tag{1}$$

where s is the independent components, and A is A is an $m \times m$ mixing matrix of parameters. ICA finds the optimal demixing matrix $W = A^{-1}$ by solving a objective function, and recovers $s = Wx$. The traditional ICA algorithms were based on objective functions defined in terms of expectations of a single fixed nonlinear function, such as kurtosis, negentropy. KICA define the objective function in a reproducing kernel Hilbert space(RKHS), and make use of the "kernel trick" to search over this space efficiently. The objective function of KICA is relates to the kernelized first canonical correlation between variables. Mathematically, the kernalized first canonical correlation can be obtained by finding the minimal eigenvalue of the following matrix.

$$\widetilde{K_k} = \begin{pmatrix} I & r_k(K_1)r_k(K_2) & \cdots & r_k(K_1)r_k(K_m) \\ r_k(K_2)r_k(K_1) & I & \cdots & r_k(K_2)r_k(K_m) \\ \vdots & \vdots & & \vdots \\ r_k(K_m)r_k(K_1) & r_k(K_m)r_k(K_2) & \cdots & I \end{pmatrix} \tag{2}$$

where $K_1, \ldots, K_m$ denote the kernel matrices of the observations, where $r_k(K_i) = K_i(K_i + \frac{Nk}{2}I)^{-1}$, and $K_i + \frac{Nk}{2}I$ denotes the regularization of K_i and $\frac{Nk}{2}$ is called jitter factor [6]. The minimal generalized eigenvalue, denoted as λ_{min}, and the objective function can be defined as $C(W) = -\frac{1}{2}\log\lambda_{min}$. The value of the objective function is nonnegative, and equals zero if and only if the variables are pairwise independent [4].

An ICA objective function is actually a function of demixing matrix W. Estimating the independent components means minimizing the objective function with respect to W. It is quite challenging because the scale of the constructed kernel matrices are quite large. Recently, Wang et al. proposed Nyström KICA(NICA) to perform low-rank approximations of the kernel matrices. NKICA randomly sample M data points form the observations consist of N samples, $M \ll N$. The NKICA algorithm is shown in Algorithm 1. The total computational complexity of NKICA Algorithm 1 is $O(mM^2N)$.

2.2 Intel Many Integrated Core Architecture

Intel many integrated core (MIC) architecture is designed to accelerate highly parallel and computational intensive applications. Intel Xeon Phi coprocessor is a commercial product based on MIC architecture. One coprocessor consists of up to 61 cores. Each core has two level caches, includs a 32 KB L1 data cache and L1 instruction cache, and a 512 KB private L2 cache. A 512 bits vector processor unit (VPU) based on Single Instruction Multiple Data (SIMD) architecture is implemented on MIC.

Intel MIC architecture is easy to program for developers. Applications on other platform can also be ported to MIC with little modification. The MIC coprocessor is supported by a variety of libraries, compilers and tuning tools, etc. In practical, developers usually use the parallel programming interface, such as OpenMP and TBB, to utilize the abundant computational core resources. The

Algorithm 1. NKICA Algorithm[8]

Input : Data vectors $x^1, x^2, \cdots, x^N$,
Kernel function $K(x, y)$.
Output: Estimated independent components s.

1 Whiten the data
2 perform the low-rank approximation via Nyström method, randomly choose $M \times M$, $K_i \approx P_i A_i^{-1} P_i^T$
3 Compute the orthogonal eigenvectors and eigenvalues of K_i, U_i and L_i, $K_i \approx U_i L_i U_i^T$
4 Compute the kernel matrix R_K based on U_i and L_i, find the minimal eigenvalue of R_K λ_{min}
5 Define objective function $C(W) = -\frac{1}{2} \log \lambda_{min}$
6 Minimize the objective function, obtain the demixing matrix W
7 Output the estimated independent components $s = Wx$.

MIC coprocessor is often treated as an accelerator to perform the computational intensive tasks offloaded from CPU. Lager scale matrix computations are main tasks for MIC coprocessor. With the help of Intel math kernel libraryMKL, matrix computations can be performed effectively on MIC.

3 Analysis of Parallelism

This paper aims to parallelly accelerate NKICA algorithm on MIC. So the main work is to parallelize the most time consuming parts, i.e. the critical segments, of the algorithm, and perform the computations on the MIC. By analysing the NKICA algorithm in Algorithm 1, we find the critical segment is the computation of the objective function. The objective function is computed many times in Algorithm 1 to find the minimal value. Table 1 shows the simulated execution of NKICA, the components number sets to 2, 6, 10. We can see the total time of computing objective function accounts for more than 80% of the execution time of NKICA algorithm. Therefore, the computation of objective function is the critical segment needs to be parallelized.

Table 1. Simulated excution of NKICA

Number of independent components	2	6	10
Execution time of NKICA Algorithm	8.25 s	86.06 s	316.68
Total time of computing objective function	7.33 s	73.98 s	267.51
Percentage	88.85%	85.96%	84.47%

There are two levels of parallelism inhered in NKICA algorithm. The first level is the parallelism of computing K_i of each observation, and the second level

is the parallelism inhered in matrix manipulations, including matrix multiplication, singular value decomposition and eigenvalues decomposition. In practical, We use OpenMP technology to exploit the first level of parallelism, and Intel math kernel library (MKL) to realize the second level of parallelism. MKL has rich functions and can perform matrix manipulation with high-efficiency. The data needed in the algorithm are constructed as matrix, so we can utilize the computational ability of coprocessors VPU. Algorithm 2 summarizes the NKICA algorithm with MIC.

Algorithm 2. NKICA algorithm with MIC

Input : Data vectors $x^1, x^2, \cdots, x^N$,
Kernel function $K(x, y)$.
Output: Estimated independent components s.

1 Whiten the data
2 Generate the initial demixing matrix W', compute the corresponding source $s' = W'y$
3 offload s' and Kernel function $K(x, y)$ from CPU to MIC
4 perform the low-rank approximation via Nyström method, randomly choose $M \times M$, $K_i \approx P_i A_i^{-1} P_i^T$
5 Compute the orthogonal eigenvectors and eigenvalues of K_i, U_i and L_i, $K_i \approx U_i L_i U_i^T$
6 Compute the kernel matrix R_K based on U_i and L_i, find the minimal eigenvalue of R_K λ_{min}
7 Return objective function $C(W) = -\frac{1}{2} \log \lambda_{min}$ to CPU
8 Minimize the objective function, obtain the demixing matrix W
9 Output the estimated independent components $x = Wy$.

In Algorithm 2, Step 3 to step 7 are performed on MIC, use OPENMP technology to realize parallelization, and MKL to perform the matrix manipulation. We adopt offload mode of MIC in Step 3. Step 5 and step 6 perform singular value decomposition and eigenvalues decomposition on A_i and R_K respectively. In practice, the regularization scheme in NKICA enables us to ignore the eigenvalues less than the jitter factor, which significantly reduces the computational complexity.

4 Experiments

To verify the performance of the accelerated algorithm, we compare the execution time between MIC-based NKICA algorithm and original non-accelerated one only on CPU. The CPU is Intel Xeon server CPU, 2.6 GHz, two way 8 cores. the only CPU-based algorithm also uses MKL to perform matrix manipulation. The datasets used in experiments include the simulated dataset and the TIMIT dataset. The input kernel function is Gaussian Kernel in all experiments. All the experiment results are averages of 10 replications.

There are three parameters have significant influence on execution time, which are components number m, observations data size N and sampling number M. When m and N increases, we can perform NKICA algorithm on larger datasete. When M increases, the results will be more accurate. Meanwhile, the computational complexity increases along with the increase of the three parameters. We set the three parameters to different values respectively, and record the execution time of the MIC-based algorithm and the only CPU-based algorithm on different dataset.

4.1 Simulated Dataset

The simulated dataset consists of data obtained from a variety of source distributions. In this paper, we use five basic distributions to generate a large dataset. Figure 1 shows the five basic distributions. We set different parameters to the distributions to get different data.

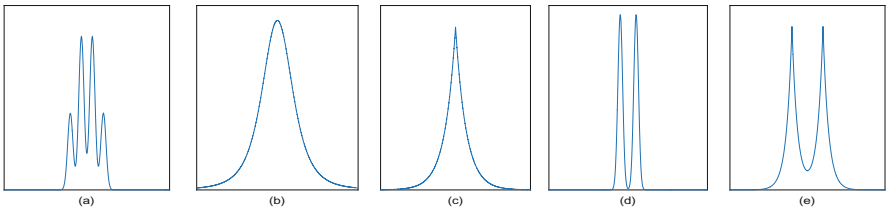

Fig. 1. Probability density functions of sources in simulated dataset with their kurtoses: (a) Mixtures of four Gaussians; (b) Student; (c) Double exponential; (d) Mixture of two Gaussians; (e) Mixtures of two double exponential.

Figure 2 shows the results of two experiment sets on simulated dataset. We compare the execution time between the MIC-based and the only CPU-based NKICA algorithm. The results show that the MIC-based algorithm runs much faster than the only CPU-based one, an around 10 times speedup is achieved by MIC-based algorithm, when the three parameters set to different values.

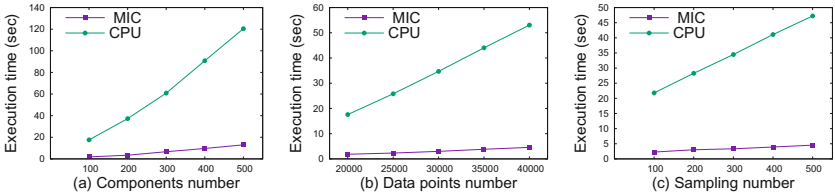

Fig. 2. Execution time with different parameter values: (a) Components; (b) Data points; (c) Sampling points.

4.2 TIMIT Dataset

TIMIT dataset [14] consists of large amount of speech segments of different speakers. We use it to test our MIC-based algorithm for practical applications, such as speech separation [15]. Speech segments are all sampled at a rate of 16 kHz.

Figure 3 compares the original speech signals, which used to generate observation data, and the recovered speech signals by MIC-based NKICA algorithm. The first row shows the time-domain signals, while the second row shows the frequency-domain signals. The estimated 1 signal is the corresponding recovery of the source 1, while the estimated 2 corresponds to source 2. The results demonstrate that MIC-based NKICA algorithm can perform speech separation effectively.

Figure 4 shows the results of the two experiment sets on TIMIT dataset. We also compare the execution time between the MIC-based and only CPU-based NKICA algorithm. The results show that the performance of MIC-based algorithm is also much better than the only CPU-based one on TIMIT dataset. An around 10 times speedup is achieved by MIC-based algorithm, when the three parameters set to different values. The best speedup is 12.3, when m set to 100, M set to 40, and N set to 50000. In the two experiments, the MIC architecture shows its great effectiveness in performing NKICA algorithm. The

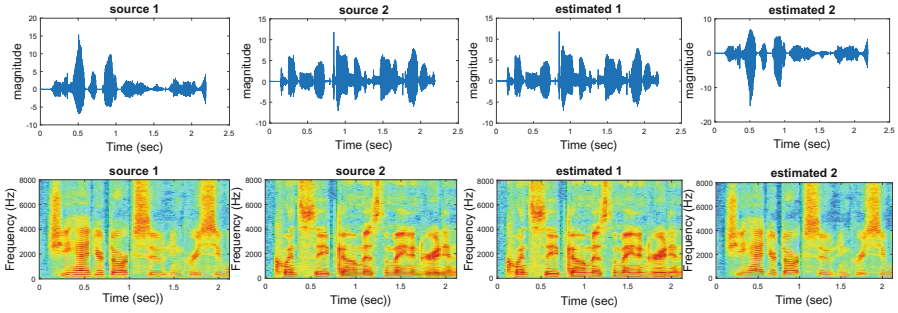

Fig. 3. Decomposition result on TIMIT dataset by NKICA. The first row shows the time-domain signals of original speech and recovered speech, and the second row shows the frequency-domain signals of original speech and recovered speech

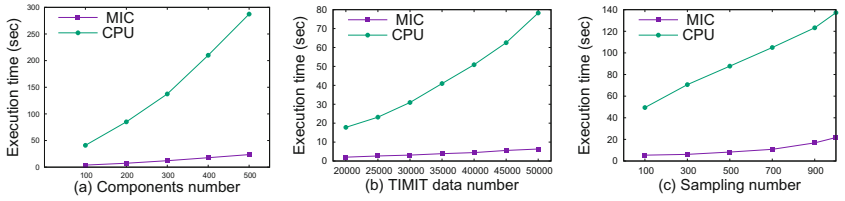

Fig. 4. Execution time with different parameter values: (a) Components; (b) Data points; (c) Sampling points.

intensive matrix manipulations are offloaded to coprocessor and performed parallel, which makes the NKICA algorithm has a much shorter execution time on MIC. In summary, MIC implementation is a very effective way to accelerate NKICA algorithm.

5 Conclusion

The NKICA algorithm reduces the computational complexity of the KICA algorithm effectively. However, with the scale of the dataset increases, the computational complexity of NKICA algorithm grows quickly. This paper presented a MIC-based implementation of the NKICA algorithm. We analyzed the parallelism of the NKICA algorithm, and parallelized the critical segments by offloaded the intensive computations to MIC. Our implementation took advantage of the OPENMP and MKL technology, and made best use of the parallelism provided by MIC. The experiment results on simulated dataset and TIMIT dataset show that MIC-based NKICA algorithm costs much less execution time, When comparing with the only CPU-based one. The speedup is around 10 on average, and 12.3 at best. The experiment results confirmed that MIC-architecture is suitable for accelerating NKICA algorithm.

References

1. Hyvärinen, A., Oja, E.: A fast fixed-point algorithm for independent component analysis. Neural Comput. **9**(7), 1483–1492 (1997)
2. Du, K.L., Swamy, M.: Independent Component Analysis: Neural Networks and Statistical Learning, pp. 419–450. Springer, London (2014)
3. Hyvärinen, A., Karhunen, J., Oja, E.: Independent Component Analysis. Wiley, New York (2004)
4. Bach, F.R., Jordan, M.I.: Kernel independent component analysis. J. Mach. Learn. Res. **3**, 1–48 (2003)
5. Shen, H., Jegelka, S., Gretton, A.: Fast kernel-based independent component analysis. IEEE Trans. Sign. Process. **57**(9), 3498–3511 (2009)
6. Williams, C., Seeger, M.: The effect of the input density distribution on kernel-based classifiers. In: Proceedings of the 17th International Conference on Machine Learning, no. EPFL-CONF-161323, pp. 1159–1166 (2000)
7. Kumar, S., Mohri, M., Talwalkar, A.: Sampling techniques for the Nyström method. In: International Conference on Artificial Intelligence and Statistics, pp. 304–311 (2009)
8. Wang, H., Xu, W., Guan, N., Yang, C.: Fast kernel independent component analysis with Nyström method. In: International Conference on Signal Processing (2016)
9. Duran, A., Klemm, M.: The intel many integrated core architecture. In: 2012 International Conference on High Performance Computing and Simulation (HPCS), pp. 365–366. IEEE (2012)
10. Jeffers, J., Reinders, J.: Intel Xeon Phi coprocessor high-performance programming. Newnes (2013)
11. Chrysos, G.: Intel xeon phi coprocessor-the architecture. Intel Whitepaper (2014)

12. Tarditi, D., Puri, S., Oglesby, J.: Accelerator: using data parallelism to program gpus for general-purpose uses. In: ACM SIGARCH Computer Architecture News, vol. 34, no. 5, pp. 325–335. ACM (2006)
13. Lee, S., Min, S.J., Eigenmann, R.: OpenMP to GPGPU: a compiler framework for automatic translation and optimization. ACM Sigplan Notices **44**(4), 101–110 (2009)
14. Garofolo, J.S., Lamel, L.F., Fisher, W.M., Fiscus, J.G., Pallett, D.S.: DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. nist speech disc 1–1.1, NASA STI/Recon Technical report N, vol. 93 (1993)
15. Guan, N., Lan, L., Tao, D., Luo, Z., Yang, X.: Transductive nonnegative matrix factorization for semi-supervised high-performance speech separation. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2534–2538. IEEE (2014)