Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(8):66-70

Research Article ISSN: 2394 - 658X

Optimistic Workload Configuration of Parallel Matrices On CPU

Apoorva Reddy Proddutoori

San Diego
Email: apoorvaproddutoori@gmail.com

ABSTRACT
This study compares and uses different feature parallelization techniques Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT) for classification of matrices. Convolutional Neural Network (CNN) is used to determine the classifications. In the classification, CNN is a unique technique that can be effectively used as a classifier. This study helps to extract features in the most efficient way with less computing time in real life. The framework provides comprehensive and flexible APIs that enable efficient implementation of multi-threaded applications. To meet the real-time performance requirements of these security applications, it is imperative to develop a fast parallelization technique for the algorithm. In this paper, we introduce a new memory-efficient parallelization technique that efficiently places and stores input text data and reference data in an on-chip shared memory and CPU texture cache. For better performance while reducing the power ratio, we extend the parallelization technology to support other major cores of the SOC. OpenCL, a heterogeneous parallel programming model, is used to communicate between CPU and other macro blocks.

Key words: FFT, DWT, CNN, framework, parallelization, classifier.

INTRODUCTION

Recently, multi utility microprocessors and micro architectures have become increasingly resourceful in advanced technology systems like artificial intelligence, augmented reality, gaming, video, and signal/image processing. A multiprocessing architectural system consists of at least two processing units accessing combined memory and diverse hardware macro units on the SoC depending upon the target requirements. As perse, a physical CPU is just a processor by itself efficiently handling serial and parallel workload data transmission for a variegate of applications.

To take full advantage of the available parallel matrices computing power, the algorithm is divided into an optimal number of threads and then tasks. A task is a simple set of threads that can run in the kernel. Tasks can be cached and scheduled between hardware threads. Task allocation is a complex task and requires optimization. Opposingforces are at work: too many partitions add extraand too few partitions waste CPUs, hence the need for optimal partitioning. The parallel feature extraction andclassification algorithm is divided into two main parts: Parallel feature extraction and parallel classification. Three different \techniques, Fast Fourier Transform (FFT), and Discrete Wavelet Transform (DWT) are compared to find the best information features. For classification, a parallel computational neural network (CNN) has been proposed as an efficient workload classification. The CNN approach combines a feedforward neural network and a genetic algorithm (GA). GA has proven to be very effective in practicefor optimizing functions and can efficiently search large and complex spaces to find a nearly global optimum. The search space associated with the neural network weight selection problem is an ideal GA usage target. The extracted features are used as inputs to the CNN to train the network. The training part of CNN is used to train the neural networkdata for a given number of iterations to produce the best chromosome.

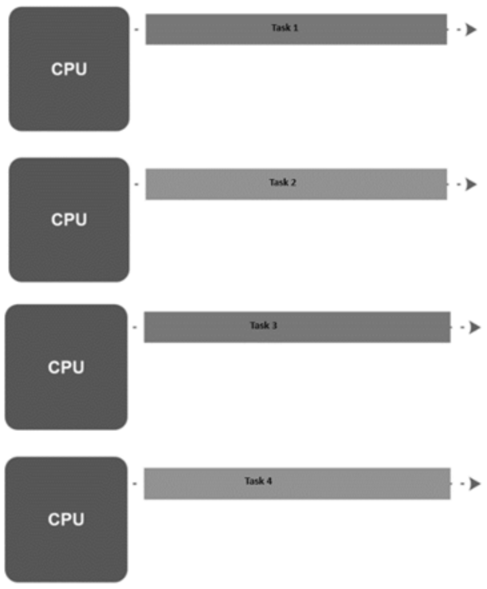

Figure 1: Independent CPU Operation

One of the main problems in parallel matrices computation is that the size of the sparse parallel matrix is very large Developing a parallel matrix subarray is very expensive Some elements of the parallel matrices are difficult to calculate or reconstruct, while others are not. Different parts of the parallel matrix can be calculated using different methods, so there are two ways One way is to precompute a small array, at the beginning. This option is useful for some parts of the parallel matrices that are difficult to reproduce. Taking this approach is limited by CPU memory Another approach is to calculate the elements of the parallel matrices constant matrix on the fly. This option is useful for some parts of the parallel matrices that are easy to rebuild. In this work, we implement an efficient and robust scheme tostore parallel matrices slot arrays on the CPU and GPU. The proposed format compresses small arrays to save a lot of CPU memory and GPU memory.

PARALLEL MATRIX SCHEDULER ALGORITHM

The calculation for parallel matrix includes extraction and classification is partitioned into primary computational parts, highlighting extraction and development of the computational neural network arrangement (CNN). Each portion is encouraged to be partitioned into sub-parts and each sub-part is isolated into ideal number of steps, to discover the best parallel matrix for parallelism. These steps ought to be divided, mapped, and planned on a multicore so that they can execute the whole calculation productively. Noting the limitations of current CPU-based parallel matrix research, this paper explains a CPU-based parallel matrix approach by fully incorporating parallelization and sub-parallel optimization algorithms.

A. Parallel Scheduling

First, parallel scheduling can be performed either with memory or workload. Indulging deeper into memory architecture parallelization is by developing a symmetric multi-matrix processing (SMMP) architecture, Using SMMP, various parallel matrices of the CPU are joined together forming a consolidated gigantic memory unit leading to single instance of the operating system, proving scope for working as unified processor or multiple processors.

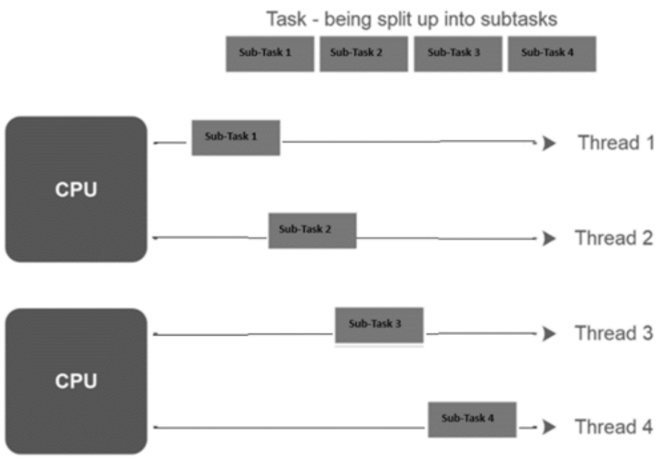

Figure 2: Parallel Matrix Workload

In the computational approach to parallel matrix programming, the simplest and easiest way to implement parallel matrix applications is to use multi-matrix threading. All real programming libraries are multithreading capable and provide mechanisms to automatically manage multiple threads on different cores. Open multi-matrix threading is an API for the shared memory model. It is one of the most powerful high-level parallel languages.

Secondly, parallelization can be exploited at workload level, almost all image-processing algorithms use grayscale as input. However, almost all hardware video sources provide images in RGB format. Hence, the grayscale transform is a very popular transformation for workload parallel matrices.

B. Parallel Concurrent Scheduling

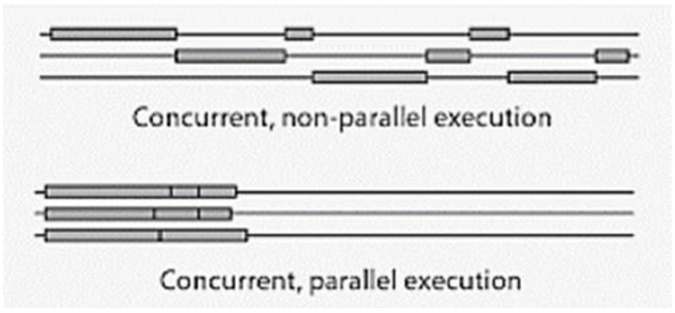
Concurrent, non-parallel execution
Concurrent, parallel execution

Figure 3: Parallel Concurrent Execution

The algorithm necessarily implicate that the multi-matrix threads would all be running at the same instant, but concurrency algorithm can be initiated on multiple single core's of the CPU. Benefiting from this idea, an algorithm for parallel concurrency scheduling has been developed to potentially utilize multi-threading multi-core CPUs processors. Compared to single core CPU, which cannot execute and parallelize the matrices for execution at the same time, multi-core CPUs have an advantage of allocating multi-threaded parallel matrix in memory and distribute the workload decomposition knowledgeably. Therefore, introducing the concept of parallel concurrency would reduce the latency of the CPU and processor higher workload.

MAPPING THE SCHEDULER

A. Parallel Fast Fourier Transform

When parallelizing the FFT algorithm, the recursive approach of the FFT algorithm is better to implement. However, there are two reasons for using the crossover method in the FFT algorithm. First, the inverted version of the FFT algorithm can perform fewer arithmetic calculations. Second, it is easier to generate a parallel FFT algorithm than any sequential algorithm. We already know that the output index is a bit-by-bit inversion of the input index. So, use this idea to rearrange the parallel matrix alignment.

From figure 4, top of the sequence is considered to be the input with n variables being fed into the parallel matrix, while bottom sequence of the matrix is considered to be the output. The gray block depicts each of the matrix running in parallel. There are three statements for a parallel matrix FFT algorithm. Considering n be the number of input elements and p be the number of processes running in parallel on the CPU core. First, the processes modifies the input sequence and reorders the indexes. In the second statement, the processes perform the first log n - log p iterations of the FFT, performing the necessary convolutional factors on the complex numbers. In the third statement, the processes perform the final protocols of the FFT and exchange values between the dimensions of the hypercube with the partner. Thus, each process directs n/p elements of the input sequence a. There are protocol p iterations in which each process exchanges n/p values with a partner process. The time complexity of the general communication is O((n/p) log p) and the computational complexity of the parallel algorithm is O(n log n/p).

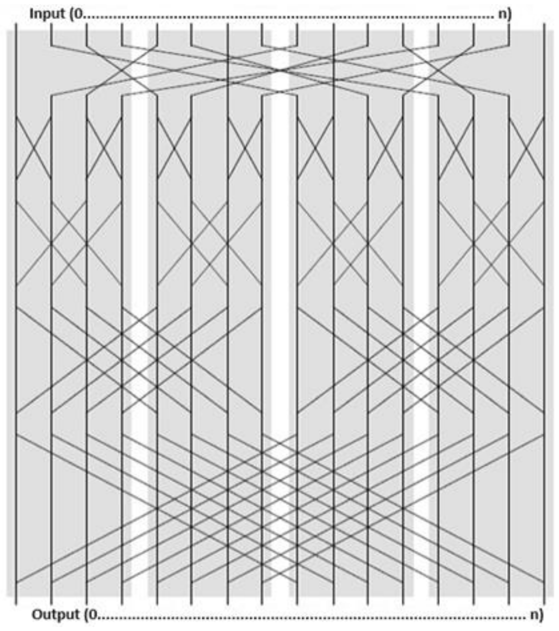
Input (0.. n)
Output (0.. n)

Figure 4: Parallel FFT Technique

B. Discrete Wavelet Transform

Discrete Wavelet Transform have been one of the vital flag handling advancements within the parallel matrix, particularly for applications such as time, power, bandwidth and frequency investigation, information compression, division and vision. In spite of the fact that a few proficient executions of wavelet changes have been inferred, their computational burden is still significant. The paper depicts nonexclusive parallel matrix usage of wavelet changes, based on the pipeline processor cultivating strategy, which have the potential to attain real-time execution. Comes about appear that the parallel usage of the oversampled wavelet change accomplishes for all intents and purposes direct speedup, whereas the parallel matrix execution of the discrete wavelet change (DWT) too outflanks the successive adaptation, given that the channel arrange is huge. The DWT parallelization execution moves forward with expanding information length and channel arrange, whereas the frequency-domain usage execution is autonomous of wavelet channel arrange. Parallel matrix pipeline executions are right now reasonable for handling multidimensional pictures with information length at slightest 480 pixels.

CONCLUSION

Parallel extraction and parallel clustering implemented on single and multiple cores. We extract salient features using three different feature extraction techniques (FFT and DWT). CNN was used for classification. With more iterations the algorithm uses more available multicore resources compared to fewer iterations. We can see that the parallel algorithms for all feature extraction methods are optimized faster on consumer-level multi-core computers compared to single-core systems. Algorithmic calculations using DWT feature extraction have also been found to provide higher accuracy than FFT and MFCC. DWT provides both the nature and timing of signals, which can provide a more accurate representation of global signals.

FUTURE SCOPE

In future work, this algorithm will be implemented on many different systems, and the results will be compared to systems on the same chip. Furthermore, Computer vision applications based on face recognition are highly non-trivial tasks that are intrinsically parallel. In this paper, we investigated the software parallelism and performance of the parallel matrix algorithm optimizing directives for task parallelism and instructions for data parallelism. To further improve the performance/energy criterion.

REFERENCES

[1]. Mohammed Mahmoud, Mark Hoffmann, Hassan Reza, “An Efficient Storage Format for Storing Configuration Interaction Sparse Matrices on CPU/GPU, International Conference on Computational Science and Computational Intelligence, IEEE Xplore, August 2017

[2]. Mohammad Wadood Majid, Golrokh Mizaei, Mohsin M. Jamali, “Parallelization of Feature Extraction Techniques on Consumer–Level Multicore System”, IEEE, February 2012

[3]. Nhat-Phuong Tran, Myungho Lee, Sugwon Hong, Minho Shin, “Memory Efficient Parallelization for Aho-Corasick Algorithm on a GPU”, IEEE 14th International Conference on High Performance Computing and Communications, 2012

[4]. Agnes Ghorbel, Nader Ben Amor and Mohamed lallouli, “Towards a parallelization and performance optimization of Viola and Jones algorithm in heterogeneous CPU -GPU mobile system”, IEEE, May 2015

[5]. Xueneng Su, Chuan He, Tianqi Liu, Lei Wu, “Full Parallel Power Flow Solution: A GPU-CPU-Based Vectorization Parallelization and Sparse Techniques for Newton–Raphson Implementation”, IEEE Transactions on Smart Grid, Vol. 11, No. 3, May 2020

[6]. https://jenkov.com/tutorials/java-concurrency/concurrency-vs-parallelism.html

[7]. https://blog.terramate.io/how-to-use-multithreading-and-multiprocessing-a-beginners-guide-to-parallel-and-concurrent-a69b9dd21e9d#608f