Acceleration Techniques for Motion Estimation Algorithms Using Parallel and Distributed Computing

Jonas C Meinerz, Eduarda R Monteiro, Felipe M Sampaio, Sergio Bampi, Altamiro A Susin

Instituto de Informática, Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil

{jcmeinerz, ermonteiro, fmsampaio, bampi} @inf.ufrgs.br; altamiro.susin@ufrgs.br

***Abstract*— This work features different approaches on towards Motion Estimation (ME), looking for higher throughput to the procedure which is the main performance bottleneck of video encoders. In spite of consuming a high share of the total processing amount of an encoder, ME also provides its highest gains regarding compression. This work's purpose is to take advantage of the high potential for parallelism of the Motion Estimation algorithms, in this case focused on the Full Search algorithm, making use of combined parallel and distributed programming paradigms. We obtained gains up 7x with the propose implementation (OpenMP + MPI libraries) when compared to the serial version, considering different search areas.**

***Keywords* — Video coding; Motion Estimation; Full Search Algorithm; parallel; distributed; OpenMP; MPI**

I. INTRODUCTION

The heaviest load of data transmitted throughout the internet represents digital video [1]. Limited bandwidth and high amounts of digital data to be sent and received are, among others, factors for which digital videos must be compressed. Introduced by this context, video coding is a set of procedures that look for compressing digital videos.

While the digital video industry shows frequent innovation and growth regarding digital image quality and resolution both, the growth in the bandwidth of internet connections cannot keep its pace, thus creating a need to reduce the amount of transferring data in order to provide real-time broadcasting when dealing with high quality digital videos.

The digital video coding catches both academy's and industry's attention, having its importance shown in every kind of digital video broadcasting, such as through the internet, DVD, Digital Television, Blu-ray and so on. For that reason, a variety of standards has been developed as the need to compress data without losing valuable information gets more demanding with higher resolution and definition of digital images. The procedures here featured can be adapted to either consolidated standards (e.g., MPEG-2 [2], H264/AVC [3], and others) or up-and-coming standards (such as HEVC[1] - High Efficiency Video Coding [4]).

The video coding process's goal is to remove redundant information from the digital data of the image video. That is, exploiting the video's temporal and spatial redundancies in addition to entropy encoding [5].

Here in this work, the focus will be put onto the Motion Estimation (ME) process, which is one of the steps that accounts for the video coding as a whole [5]. As it is, ME may be considered a bottleneck for the whole process of the video coding, which makes that any time-related improvements made in this particular process strongly relevant in the whole coding process elapsed time [6].

As said before, with higher resolutions and higher amounts of data to be handled comes the need for more aggressive compression. The need to compress more and at the same time to care about valuable information is the motivation for creating standards that are more satisfying on regard of digital data compression. Although the newer standards offer more successful compression rates (HEVC saves about 80% bit rate in comparison to MPEG-2 and half this percentage when compared to H264/AVC [7]), they also demand a series of operations whose complexity can be quite concerning in situations when a recording must have a real time broadcast. On the other hand, the main part of video coding, as the most costing one, ME is a repetitive process that may operate on independent data, which means it has a large potential for parallel processing. Combined to the popularity of multi-threading and multi-core architectures nowadays, this potential enables a set of possibilities to be explored with the intention to reach high throughput rates in the coding process of a digital video. The literature brings us a variety of algorithms for ME, the Full Search (FS) algorithm and Diamond Search (DS) algorithm being the most popular ones.

This work will present different implementations of the Full Search algorithm, comparing the performance of serial, parallel, distributed and combined parallel and distributed versions. FS is an optimal and exhaustive algorithm, itself having a very high computation complexity, but it is completely based on data independence, this one being the most important factor regarding parallel computing. Our work exploits this peculiar characteristic of FS algorithm, in order to achieve a highly parallel ME in order to accelerate the video coding process as a whole. We obtained a gain up *4x* (in terms of speed-up) with the parallel OpenMP version and up *7x* (in terms of speed-up too) with the hybrid OpenMP + MPI parallel and distributed version in comparison to the sequential implementation.

We will be featuring more detailed explanations about Motion Estimation and the Full Search algorithm in Section 2. A quick explanation of our implementations will be given in the Section 3. The Section 4 will present the results of our implementations and comparisons to justify its worth. Section 5, for its part, is bringing this work's conclusions.

[1]The new video coding standard, which may be released in the first semester of the present year.

II. Motion Estimation

Among other steps, Motion Estimation is a component part of the so called video coding. There are plenty of ways to build a video coder, but an optimal algorithm for ME will always consume a relevant share of the total encoding processing [8].

ME is a lossless procedure as the decoding process recovers the original data without any losses. Nevertheless, it helps in the compression by changing the way the digital data is represented. While a not yet coded piece of video represents every block (e.g., a spatial-related chunk of a frame) by its literal value, after being coded, most blocks belonging to this piece of video may be represented as motion vectors, each vector representing a previously reconstructed block, in addition to a residue, representing the difference between that previously reconstructed block (found in an already coded frame of that piece of video) and the block currently being coded. After that, instead of storing the values of every pixel from every block, most blocks will be stored as a coordinate and a module. The compression level of the Motion Estimation is completely related to a video's temporal redundancy, which gives a good reliability for this process since most videos have a very strong temporal connection between their frames.

A. Motion Estimation Mechanics

Basically, the ME process for a block demands a reference frame, a search range and the current frame where the block itself is situated.

The currently selected block will be compared, based on a similarity criterion, to every block of the reference frame within a search range, and the coordinates of that one block which most resembles the currently selected block will be stored alongside its discrepancy in comparison to the currently selected block. There are several ways to define a search range and several more similarity criteria. It is up to the implementer to decide how the search area will be defined and which criterion will be used. These decisions will lead to the gains and losses of the process.

In the literature there are many different block-matching algorithms [9] (e.g., Full Search, Diamond Search, Three-Step Search and others), and the most remarkable difference among those algorithms is the way the search range is defined.

B. Full Search

As mentioned before, this work will focus the Full Search algorithm, which was chosen due to its high capacity for parallelism and for being optimal.

The FS is an exhaustive algorithm that gives us the optimal result within a search area. That means the search range will be composed from every possible block of the reference frame around the currently selected block position within a certain range. After comparing every block inside the search range, a coordinate and a residue will be stored representing the block which resembles the most the current block and its discrepancy to it. Despite being exhaustive and implying in the biggest number of comparisons that could be made in a search area, it is very important to state that the FS algorithm has no data dependency among the blocks yet to be coded. That implies every single block to be processed can be coded at the same time if there is hardware available for it. Theoretically speaking, if we can afford as many processing units as we need, the complexity of FS's ME parallel ideal version is *O(1)*, supposing we can dispatch each block to a different processing unit.

C. Sum of Absolute Differences

When we explained the ME mechanics we mentioned the need of a similarity criterion in order to measure the discrepancy between two blocks. It was arbitrarily chosen that the criterion to be used on this work's implementation of the Motion Estimation would be the Sum of Absolute Differences (SAD) [10].

III. Parallel and Distributed Full Search Implementations

Since the beginning of this text, ME's complexity computational has been emphasized, as well as its high potential for a parallel implementation. In this section, we will discuss different paradigms we used to build our parallel version of the Full Search algorithm, as well as the techniques used to distribute the computation among the available processing units we have.

The following implementations were based on a standard implementation of the Full Search algorithm made by us, using the C programming language.

A. OpenMP Parallel Full Search Implementation

OpenMP [11] is an open-source API for parallel programming that brings a high-level set of functions allowing the programmer to develop communication among the threads of a CPU.

From the standard serial implementation, the sort of changes made were the ones with the purpose to make use of every thread available on our CPU. Facing the possibility of a multi-core architecture, we decided to make each core responsible for a list of frames, in order to exploit every available processing unit at the same time avoiding stalling the computation, for example, by waiting for memory data, as much as possible. The threads within a core would be all responsible for blocks of a frame sent to that core, since these threads share the same memory. Another option would be sending a block to every core, but that would require more communication between memory and processor, and that is what we try to avoid. The figure below illustrates a flowchart representing this implementation.

Analysing the Fig. 1 we observe that one thread (called the Master Thread in our diagram) will read the first frame of the video sequence and store it, taking it as the reference frame. After that, each thread will fetch a block and perform the ME, comparing this block to every block of the reference frame within the search range, saving the coordinates and residue from its best match. This procedure is repeated till every block has been compared and finally resynchronize all the threads and finish the computation.

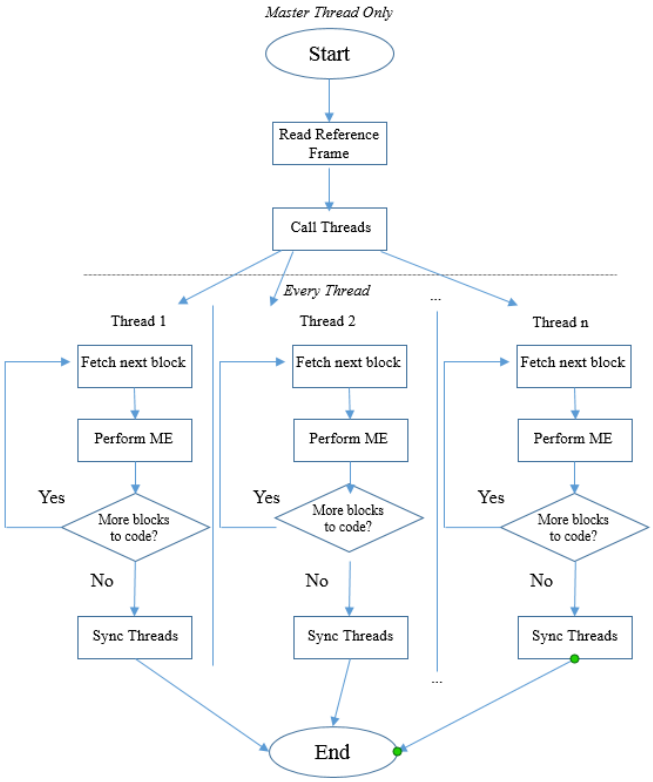

Figure 1: Diagram of computation for the OpenMP Parallel implementation of the Full Search algorithm.

B. Hybrid MPI and OpenMP Full Search Implementation

MPI (Message Passing Interface) [12] is a standardized message-passing interface. In this work it was used in order to establish communication among the nodes of a distributed PC cluster. The implementation used in this work for the MPI was MPICH2 [13].

The logic applied on the division of the labour across the processing units is the very same described in the subsection above. For this implementation, a node on the cluster receives a list of frames to code and a reference frame. The distribution algorithm is arranged in a way that guarantees balance regarding the workload of the nodes in the cluster, that is, the maximum discrepancy among the number of frames that a node will process is one.

Fig. 2 shows the diagram related to hybrid implementation distributed and parallel of the Full Search algorithm presented in this work. This hybrid version uses the OpenMP parallel implementation (described in the subsection above) with the distributed version.

In accordance with the Fig. 2, each node on the cluster will code its list of frames in a parallel way. Another level of parallelism is added inside each node: a process analogue to the process described on Fig. 1 is activated in order to make use of every thread inside that node. MPI itself foresees using every thread available on the processors of a cluster but our implementation tries to force a node to work on data it already has, by using the OpenMP framework, thus avoiding two blocks of a same frame to be coded elsewhere, reducing communication among nodes.

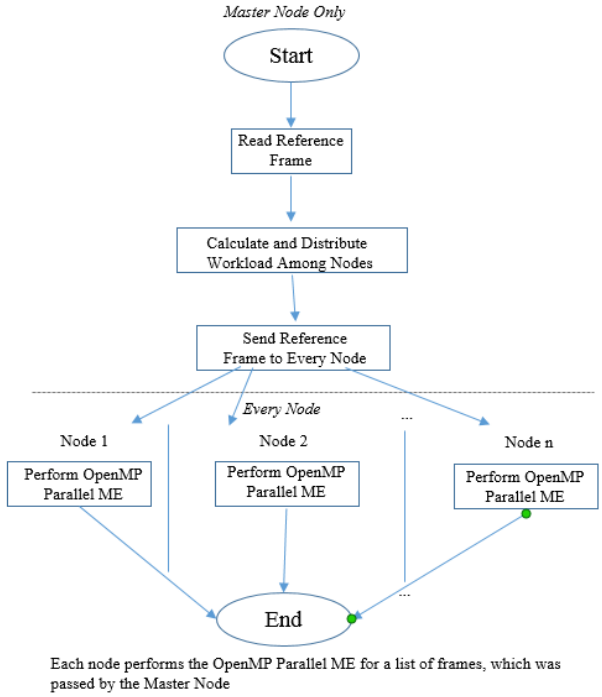

Figure 2: Diagram of computation for the hybrid MPI and OpenMP implementation of the Full Search algorithm.

IV. Results Obtained

In this section we present the obtained results with the execution of our Full Search parallelized and distributed algorithm.

Both serial and OpenMP tests were executed on a computer running an Intel Core i7 3770 (@ 3.4 GHz) [14]. The tests related to MPI ran in a cluster and used 4 nodes Intel Xeon CPU E5310 (@ 1.60GHz) [14], where each node can handle up to 8 processes. The time represented on the results refers to the time elapsed on the ME of 150 frames using the resolution HD720p (*1280x720* pixels) considering the *FourPeople* video sequence.

The block size adopted was 4x4 pixels and the search area considered in these experimental ranging from *16x16* pixels to *256x256* pixels. The performance is measured considering the time execution in seconds and performance calculate by the speed-up [15] obtained.

Fig. 3 shows a comparison with different versions of the Full Search algorithm implemented in this work: Sequential (red), OpenMP (green), the hybrid OpenMP + MPI tests ranging the number of processes in the cluster (in different shades of blue). The x-axis is related to the search area dimensions (*32x32 – 128x128 pixels*) and the y-axis represents the execution time, in seconds.

Analyzing the Fig. 3, we can observe that this graphic states that parallel versions of the FS algorithm have reached higher throughput rates than the serial one, the OpenMP version performing up to 75% faster (in terms of time execution) than the sequential version and the hybrid OpenMP

+ MPI versions performing: ***(a)*** 8 processes: up to 12%; ***(b)*** 16 processes: up to 60%; and ***(c)*** 32 processes: up to 85% faster compared to the sequential version. The values of elapsed time regarding the parallel and distributed implementations (OpenMP vs. OpenMP + MPI – 8 processes) of the FS represented in the graphic above are approximated. Despite of both architectures having a similar computation power, the distributed architecture has a fairly high communication cost among nodes, which justifies the gain of up to 22% of the OpenMP-only version, that runs the whole application inside the same processor.

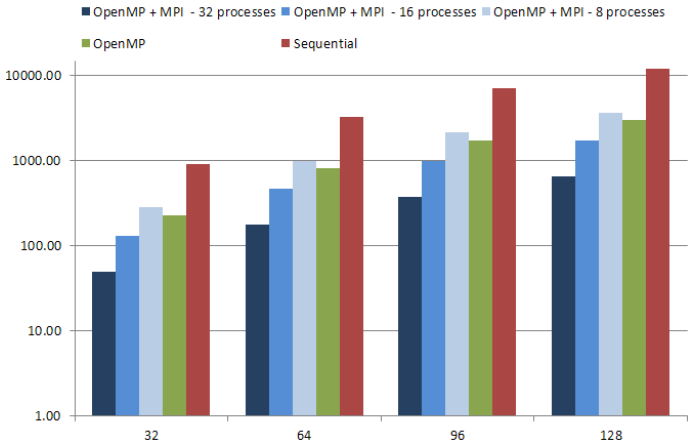

Figure 3: HD720p Time Execution Results: Sequential vs. OpenMP vs. OpenMP + MPI (8 – 32 processes).

Fig. 4 presents the hybrid OpenMP + MPI version ranging the number of processes (x-axis), in terms of speed-up (y-axis). Each curve illustrated represents a search area dimension.

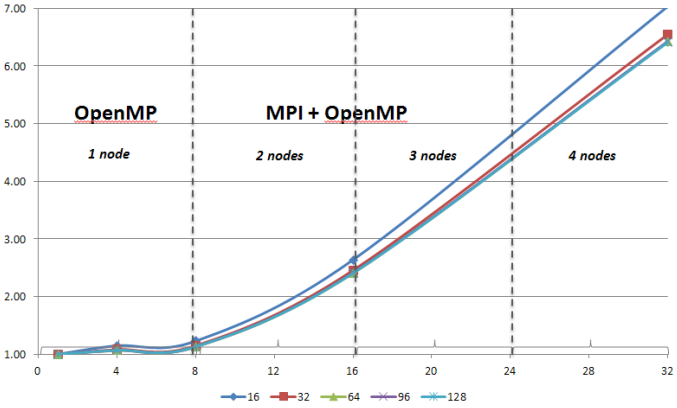

Figure 4: HD720p Speed-up Results – Time Execution: OpenMP vs. OpenMP + MPI (1 – 32 processes).

By making an analysis of the graphic presented in Fig. 4, we can notice the sum of the distributed and parallel computing paradigms, which was from the beginning the goal of this work, was successfully accomplished. Considering that the processing power of the conventional processor used in the OpenMP tests is limited to 8 threads, it is possible to accredit the speed-up boost brought until the 8th process to the parallel OpenMP implementation. In addition, as the OpenMP-only version doesn't present higher performance with the growth of search areas nor it presents reaction to the varying of search areas in the speed-up ratio. On other hand, when the hybrid parallel and distributed scope is introduced, it is possible to visualize a significant growth in terms of speed-up when more processes are dedicated to execute the application.

V. Conclusion

Our objective was to develop a merged implementation where parallel and distributed computing would coexist and lead to a more efficient processing of the ME and that objective was successfully accomplished. In addition, we have shown that the exploration of the parallelism in architecture distributed is a good alternative to accelerate the motion estimation process. Our hybrid OpenMP + MPI solution achieve up to *3x* gains in terms of speed-up increase when compared with the version OpenMP-only, and up to *7x* faster than the sequential version.

It is visible that the approach discussed on this paper has a lot to offer and great performance may be reached after refining our implementations.

Future works could focus on improving the way the work is distributed among the processing units in order to enhance the throughput and find ways to amortise the cost of communications.

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2010-2015", June 2011.

[2] ISO/IEC 13818-1:2000, "Information technology -- Generic coding of moving pictures and associated audio information: Systems".

[3] ISO/IEC 14496-10, "MPEG-4 Part 10, Advanced Video Coding".

[4] G. J. Sullivan, J. R. Ohm, W. J. Han, T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard" , IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No 12, December 2012

[5] V. Bhaskaran, K. Konstantinides, "Image and video compression standards - algorithms and architectures", Kluwer Academic Publishers, 1996.

[6] Y. Cheng, Z. Chen, P. Chang, "An H.264 Patio- Temporal Hierarchical Fast Motion Estimation Algorithm for High-Definition Video", IEEE International Symposium on Circuits and Systems, ISCAS, pp. 880-883, 2009.

[7] J. R. Ohm, G. J. Sullivan, H. Schwartz, T. K. Tan, T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No 12, December 2012.

[8] Z. Yang, J. Bu, C. Chen, L. Mo "Configurable Complexity-Bounded Motion Estimation for Real-Time Video Encoding", Lecture Notes in Computer Science Volume 3708, pp 555-562, Springer, 2005.

[9] P. M. Kuhn: "Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation ", Kluwer Academic Publishers, 1999.

[10] I. E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia", John Wiley & Sons Ltd, 2003.

[11] OpenMP Official Website - http://openmp.org/wp/

[12] The Message Passing Interface (MPI) Standard Official Website: http://www.mcs.anl.gov/research/projects/mpi

[13] MPICH Official Website - http://www.mpich.org/

[14] Intel official specifications: http://ark.intel.com/

[15] I. H. Tuncer, "Parallel Computational Fluid Dynamics: Implementations and Experiences on Large Scale and Grid Computing", Springer, 2007.