Parallelization of the Hierarchical Search in Python for High Performance Embedded Systems

Radoglou Grammatikis Panagiotis, Manganopoulou Evdoxia and Dasygenis Minas
Department of Informatics and Telecommunications Engineering
University of Western Macedonia
Kozani, 50100, Greece
radg.pan@ieee.org, manganopoulou_evi@hotmail.com, mdasyg@ieee.org

***Abstract*—The number of high performance embedded systems that are used for multimedia applications, like video encoding or decoding, has erupted. A key component in video encoding is the motion estimation, which exhibits high computational complexity and hard to meet deadlines. The most popular technique for motion estimation is block matching. The hierarchical search (HS) is a popular and a very fast block matching algorithm that achieves the best image quality, with a very high computational complexity. This complexity is usually handled using parallelization. Our work differentiates from other authors, because it targets parallelization on embedded systems using the Python framework and specifically the Multiprocessing module. The experimental results on parallelization of the HS algorithm on a high performance multi core embedded systems, illustrate the usefulness of our methodology, with speedup up to 1.4.**

***Keywords*— Motion Estimation, Block Matching Algorithms, Hierarchical Search, Python, Multimedia, Mobile, High Performance**

I. INTRODUCTION

With the development of the network and communication technologies, multimedia is becoming more and more popular, among desktop or mobile users. Mobile multimedia requires a high performance embedded system, like a multi core mobile phone having a number of ARM processors. High quality multimedia transmission requires high speed bandwidth. Thus, the basic problem during video communication is bandwidth demand. Sending several frames per second in order to create the illusion of a continuous moving sequence with high resolution, requires high bandwidth. As a conclusion, video compression was considered as a solution to a problem like that.

Since the early 1990s, when video coding technology was in its infancy, international video compression standards such as H.261, MPEG-1, H.262/MPEG-2 Video, H.263, and MPEG-4 Part 2 have been used as powerful engines behind the commercial success of digital video [6].

The basic efficient of video compression is block matching motion estimation [7]. In the motion estimation process, a frame is divided into many non overlapping blocks with different block sizes. According to the motion contents, the target block in the current frame is compared with the candidate blocks in the reference frame. Under the constraint of cost function, we can obtain the best matched block by minimizing the functional cost. Finally, the motion vector, which represents the displacement between the current block and the best matched block, along with the residual signal (which is the pixel difference between the current block and the best matched block) are transported to the next process to be coded.

There are many popular algorithms for motion estimation such as Full Search [2], Hierarchical Search [5], 3-step Logarithmic Search [10] and Parallel Hierarchical One Dimensional Search [2]. All these algorithms operate on blocks of equal size, which they have partitioned both current and previous frame in a successive video stream and estimate the motion vectors.

Until now, many researchers have presented their works on parallelizing block motion estimation. They have used either OpenMP or OpenMPI or GPU [16], or custom architectures [15] or special FPGA architectures [14]. None of them have ever tried to parallelize a block matching algorithm on an existing high performance embedded system, like a smart mobile phone. The reason is the difficulty of utilizing all these frameworks on such a platform. Smart mobile phones do not have a GPGPU, or FPGA, or OpenMP or OpenMPI. Thus, their methodologies cannot be applied.

In this paper, we present our methodology and our findings on the parallelization of the hierarchical search block matching algorithm. All the results were extracted in an embedded system and also the implementation of the parallelized HS was written in Python. The main reason we used the Python programming language is that it provides portability between different embedded platforms. Also, many Python modules add a hardware abstraction layer which helps programmers who don’t have any previous experience in embedded systems, to write code easily.

P. Radoglou-Grammatikis, E. Manganopoulou and M. Dasygenis, “P. Radoglou-Grammatikis, E. Manganopoulou, and M. Dasygenis, “Parallelization of the hierarchical search in python for high performance embedded systems,”in2016 5th International Conference on Modern Circuits and Systems Technologies (MOCAST), May 2016, pp. 1–4.

The published version is available at IEEE Xplore, May 2016, pp. 1-4, DOI: 10.1109/MOCAST.2016.7495138

II. RELATED WORK

Initial efforts for the parallelization of the block matching algorithms in old parallel processing platforms have been presented in [1, 11, 12, 14, 15, 16]. Specifically, as far as the HS algorithm is concerned, very few research works have been published, referring to systolic arrays [5, 11, 12] and not modern parallelization frameworks.

Our work differs from the previous researches in two key points: (a) by the fact that we achieve to parallelize the HS algorithm using Python programming, while all the previous works utilized C programming language, and (b) our work is the only work that targets a real high performance embedded system like a mobile phone.

One drawback of using Python in one parallelization task is that it is slower than C but on the other hand, allows programmers to express concepts in fewer lines of code than it would be possible using languages such as C or C++ [8]. Additionally, an important advantage using Python is that it can be used in many system architectures such as BSD, Linux and Windows operating systems [9]. Also, Python scripts can be executed in Android platforms and in other embedded machines [9].

III. HIERARCHICAL SEARCH ALGORITHM

Hierarchical search is a motion estimation algorithm that uses a combination of both fewer search locations and fewer pixels in computing the matching block. In this algorithm, two low-resolution versions of the current picture and the reference picture are formed by subsampling both of them by a factor of two and four [3,4]. The motion vector search begins at the lowest resolution picture (level 2). The search space is one fourth of the original one. At level one, the motion vector search is performed with the origin being the block that corresponds to the level 2 block where there is a close match and a search region of [-1,1] pixel around it. At level zero the search origin is located on the block which corresponds to the level one block where the corresponding distance criterion is minimized and the search region is again [-1,1] pixels around the block. The location that yields the smallest distance criterion corresponds to the final motion vector output.

IV. PARALLELIZATION METHOLOGY

Parallelization is defined as the concurrent execution of blocks of code [8]. As a result, the first thing that has to be paid attention is the way that the Python interpreter behaves under those circumstances and what principles it obeys.

A. *Python parallelization attributes*

Python interpreter has some distinct differences from other programming languages when it comes to issues that have to do and are related to parallelizing a program. One of the most important issues is the Global Interpreter Lock (GIL), which it cannot run more than one thread at a time [8]. However, there are numerous techniques that manage to create and execute many threads such as the Parallel Python module (PP).

B. *Multiprocessing module attributes*

There are a lot of different parallelization techniques that seem plausible, such as the Message Passing Interface for Python (MPI) [8], the Threading module [8], the Multiprocessing module [8], etc. We chose to use the Multiprocessing module for the Python 3. This module has an easy way of passing the data to and from among various processes. For this job, pipes and queues are used. The queue is generated using the Multiprocessing Queue class and its basic methods, the ones that we have extensively used in our code, are put() and get() [8]. Also, the queues are a FIFO structure as well, because the processes run concurrently. Additionally, another feature of Multiprocessing module is that it does not create separate threads, but instead, it uses spawning processes. This means that no problem can be created with GIL.

C. *Dependencies and their implications*

The HS algorithm cannot be easily parallelized, because there are many dependencies between different tasks. Specifically, the processes that execute the subsampling task can run concurrently, while the process that runs the motion estimation phase has to wait in order to collect the data from the previous ones. This is achieved with the Multiprocessing Queue class, where each process adds data in a FIFO structure, in order for another process to collect this data. The algorithm 1 defines the five main tasks of the HS algorithm that are executed concurrently and algorithm 2 shows the motion estimation task. Note that each main task is executed by one process, while the remaining processes that may exist undertake a workload for every main task.

Algorithm1 Procedure Parallel Hierarchical Search

```
1.  process_1 = create_current_frame_subsampled_by_two()
2.  process_2= create_current_frame_subsampled_by_four()
3.  process_3=create_previous_frame_subsampled_by_two()
4.  process_4=create_previous_frame_subsampled_by_four()
5.
6.  process_1.start()
7.  process_3.start()
8.  process_1.join()
9.  process_3.join()
10. process_2.start()
11. process_4.start()
12. process_2.join()
13. process_4.join()
14. process_5 = HS_Motion_Estimation()
```

```
process_5.start()
```

Algorithm 2 Procedure HS_Motion_Estimation

```
/***Level 2 ***/
for ( i=-p/4 …..) /* ME at fr.subsamp. By 4 */
for ( j=-p/4 …..)
   {
   for ( m=0 ….. )
   for ( n=0 …..)
      {
         read_from_current_frame_4()
         Check_Bound_Condition
         read_from_previous_frame_4()
         distance_criterion_check()
      }
   }
/*** Level 1 ***/
for (i=-1 …..) /*ME at fr.subsamp. By 2 */
for (j=-1 …..)
   {
   for (m=0 …..)
   for(n=0 …..)
      {
         read_from_current_frame_2()
         Check_Bound_Condition
         read_from_previous_frame_2()
         distance_criterion_check()
      }
   }
/*** Level 0 ***/
for (i=-1 …..) /*ME at original frame */
for (j=-1 …..)
   {
   for (m=0 …..)
   for(n=0 …..)
      {
         read_from_current_frame_0()
         Check_Bound_Condition
         read_from_previous_frame_0()
         distance_criterion_check()
      }
   }
```

D. Pickling

Pickling is a definition that comes from a Python module named Pickle that needs to be used in order to pass the arguments to a process [9]. In terms of Multiprocessing, the arguments to the Multiprocessing queue need to be picklable. There are certain types of names which are picklable and certain other types which are not picklable. For example the video sequence file which the algorithm HS processes is unpicklable. This creates a problem with large proportions. Let us assume that we have a file opened in the parent process. If we attempt to call a child process in order for it to be processed or to extract something from it (such as a frame), then it is almost certain that the child process will not be able to process the file and it will raise a Value Error exception. One solution to the aforementioned problem is the use of a flag which indicates whether the file is opened by a process or not.

V. EXPERIMENTAL RESULTS

In this section we present the experimental results in order to prove the robustness of our proposed methodology for the parallelization of the HS algorithm, using the Multiprocessing module. We have to note that no other authors have presented parallelization methodologies of block matching algorithms on high performance embedded systems and thus, we cannot provide any comparisons.

For the integration of the experimental measurements we used the Sony Xperia P (LT22i) smartphone with a 2x ARM Cortex-A9 processor at 1.00 GHz and with Android version 4.1.2 and Doogee Voyager2 DG310 smartphone with MTK6582 Quad Core processor at 1.3GHz with Android version 4.4.4. Additionally, the BUS and Foreman video sequences were used to evaluate the performance of the parallelized HS algorithm. The BUS video sequence consists of 150 frames with a resolution of 352 × 288 pixels and the Foreman video sequence consists of 300 frames with a resolution of 3840×2160 pixels. The adopted block size was 16 × 16 pixels as the basic unit for block matching and the search area considered in these experiments, ranged from 8 × 8 pixels to 16 × 16 pixels. Additionally, we concluded that the optimal number of processes for the parallelized HS was eleven and as a result, we had the maximum reduction of the execution time.

In order to assess the performance of the parallel implementation, we measured the practical execution time and we calculated the speedup. The practical execution time (or wall clock) is the total time that a parallel implementation needs to complete the computation. The execution time is obtained by calling the time() function of the module time. This function returns the current system time in ticks since 12:00am, January 1, 1970 (epoch). Note that the execution time in this work is referred on the motion estimation of all frames. Furthermore, it is known that the speedup is defined as the serial runtime of the sequential program, when executed on a single processor, divided by the execution time of the parallel implementation. All versions executed recursively 30 times and the average clock duration was computed.

As we can see in Fig. 1, the serial and parallel executions for the BUS video sequence take about 245.330 and 218.453 seconds and 175.0000 and 155.4600 respectively. In Fig.2, the speedups are shown for the same video. Additionally, after many experiments we have ascertained that the use of the Cython can optimize the speedups approximately 47%. Those results seem very satisfying, because as mentioned earlier, HS cannot be fully parallelized.

Finally, an important metric that can evince the possibilities of the parallelization is the energy consumption of the battery life of the Android device. Specifically in our experiments, each motion estimation level performs in worst case 4548, 20280 and 46158 arithmetic operations respectively. Therefore, the total number of the arithmetic operations is 70986. The Sony Xperia P smart phone has the Li-Ion 1305 mAh battery and the watt-hours of this battery are: 1305 mAh x 3.72 V / 1000 = 4.854 Wh. We executed the serial program 10 times and we ascertained that the 4.854 Wh of the battery was consumed 0.3% which is 0.01456 Wh. In addition, the energy efficiency of the serial program is: 70986 / 0.01456 = 4895586.20 operations/Wh (Fig. 2). On the other hand, the ten executions of the parallel program consumed 0.2% of the 4.854 Wh which is 0.0097 Wh and energy efficiency is 70986 / 0.00969 = 7318144.33 operations/Wh (Fig. 2). As a result, after the parallelization, the consumption of the Wh of the battery was meliorated by 0.00486 Wh and energy efficiency was improved by 2422558.13 operations/Wh. The same experimental results were confirmed and with the Doogee Voyager2 DG310 smart phone (Fig.2).

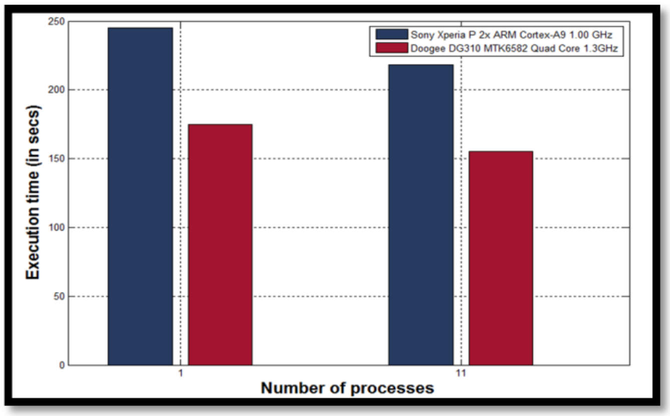

Fig1: Serial and parallel execution time of the HS Algorithm for the video sequence BUS.

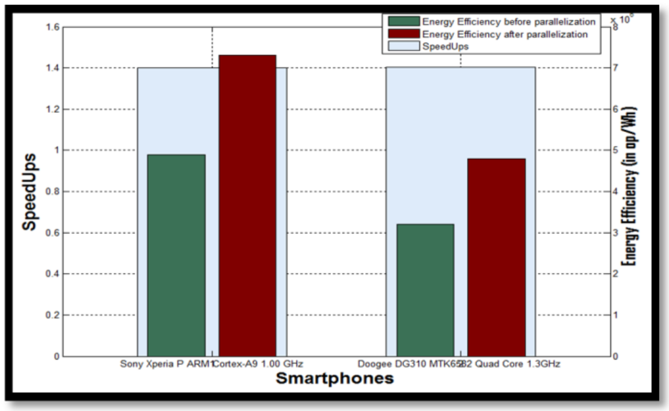

Fig 2: SpeedUps / Energy efficiency results for the BUS video sequence.

VI. CONCLUSIONS

Multimedia applications play an important role in our daily life. Up to now, nobody has presented a parallelization methodology of a key multimedia component, the hierarchical block matching algorithm, on high performance embedded systems like a smart mobile phone. In this paper, we present a unique approach in parallelizing this component, using the Python programming language and particularly the Multiprocessing module. The experimental results on multicore smart phones prove that speedup gains and energy improvements are possible with Python. Python is a well-supported language in many desktops and embedded systems, and thus our gains are not only limited on the embedded domain.

VII. REFERENCES

[1] P. Baglietto, M. Maresca, A. Migliaro, and M. Migliardi,"Parallel implementation of the full search block matching algorithm for motion estimation," In Proceedings of the 1995 International Conference on Application Specific Array Processors, pp. 182–192, 1995.

[2] V. Bhaskaran and K. Konstantinides,"Image and Video Compression Standards," Kluwer Academic Publishers, 1998.

[3] Kroupis, N., Dasigenis, M., Argyriou, A., Tatas, K., Soudris, D., Thanailakis, A., ... & Goutis, C. E. (2001). Power, performance and area exploration of block matching algorithms mapped on programmable processors. In *Image Processing, 2001. Proceedings. 2001 International Conference on* (Vol. 3, pp. 728-731). IEEE.

[4] Tatas, K., Siozios, K., Soudris, D., Masselos, K., Potamianos, K., Blionas, S., & Thanailakis, A. (2003). Power optimization methodology for multimedia applications implementation on reconfigurable platforms. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*(pp. 430-439). Springer Berlin Heidelberg.

[5] Dasygenis, Minas, and Panagiotis Michailidis,"Evaluating modern parallelization techniques on block matching algorithms," *Proceedings of the 18th Panhellenic Conference on Informatics*. ACM, 2014.

[6] Jagiwala, Darshna D., and Mrs SN Shah,"Analysis of Block Matching Algorithms for Motion Estimation in H. 264 Video CODEC," *Analysis* 2.6,2012, pp. 1396-1401.

[7] Koduri, N. R., Dlodlo, M. E., De Jager, G., & Ferguson, K. L. (2011, June). Fast implementation of block motion Estimation Algorithms in Video Encoders. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on* (pp. 103-107). IEEE.

[8] Palach, Jan., Parallel Programming with Python. Packt Publishing Ltd, 2014.

[9] Gaddis, Tony, Starting out with Python. Pearson Addison Wesley, 2009.

[10] J. Jain and A. Jain,"Displacement measurement and its applications in intraframe image coding," IEEE Transactions on Communications, 29(12), 1981, pp.1799–1808.

[11] H.-M. Jong, Liang-Gee, and T.-D. Chiueh,"Parallel architectures for 3-step hierarchical search block-matching algorithm," IEEE Transactions on Circuits and Systems for Video Technology, 4(4),Aug 1994, pp. 407–416.

[12] C. Konstantopoulos, A. I. Svolos, and C. Kaklamamis,"Efficient parallel algorithm for hierarchical block-matching motion estimation. In Proc," SPIE Visual Communications and Image Processing, 3653,1998, p.p. 481–490.

[13] Beazley, David.,"Understanding the python gil.", PyCON Python Conference. Atlanta, Georgia,2010.

[14] HaubleinK., Reichenbach M., Fey D., "Fast and generic hardware architecture for stereo block matching applications on embedded systems," inReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on gurable Computing and FPGAs,Cancun, 2014, p.p. 1-6.

[15] Siou-Shen Lin, Po-Chih Tseng, Liang-Gee Chen,"Low-power parallel tree architecture for full search block-matching motion estimation" Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium onYear: 2004,2004 Volume: 2, Pages: II - 313-16.

[16] Monteiro E., Vizzotto B., Diniz C., Zatt, B., Bampi S.,"Applying CUDA Architecture to Accelerate Full Search Block Matching Algorithm for High Performance Motion Estimation in Video Encoding," Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium onYear: 2011.,Vitoria, Espirito Santo, 2011, p.p. 128 - 135.