University of Western Macedonia
Department of Informatics and Telecommunication Engineering
Kozani, 50100, Greece

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

Radoglou Grammatikis Panagiotis, Manganopoulou Evdoxia and Dasygenis Minas
{radg.pan@ieee.org ,manganopoulou_evi@hotmail.com ,mdasyg@ieee.org}

Summary

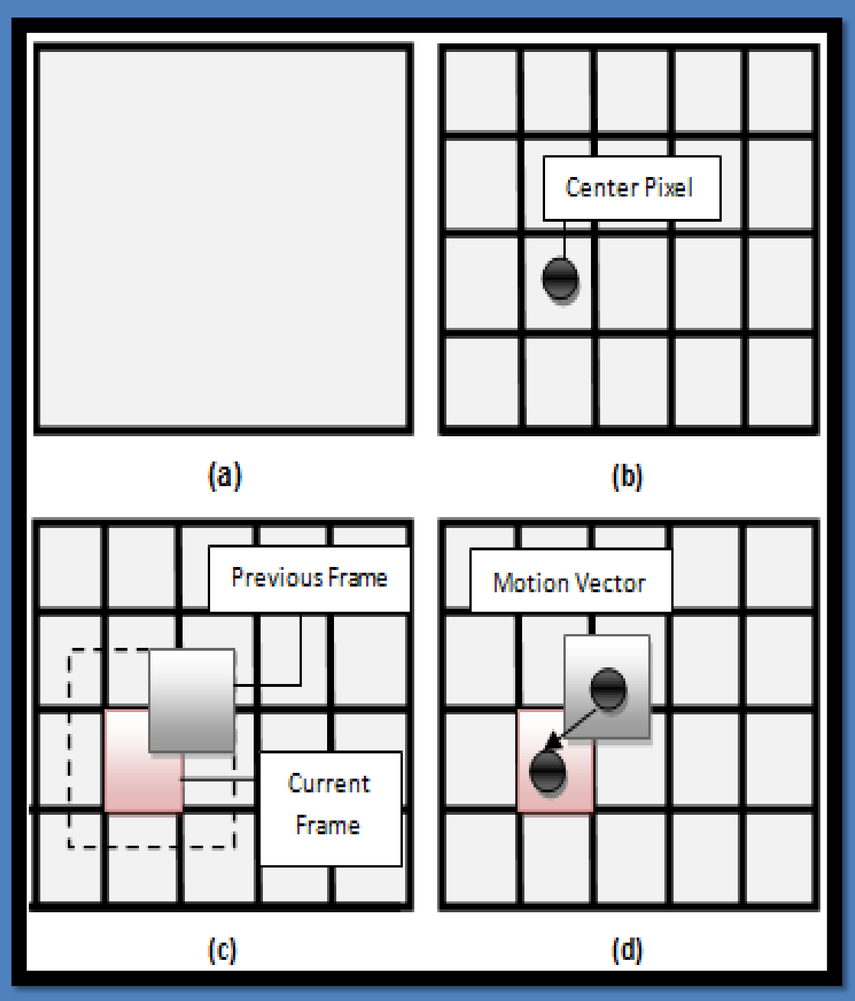
•We have parallelized Hierarchical Search (HS) block matching algorithm using Python programming language and specifically Multiprocessing library.
•The main reason we used the Python framework is that it provides portability between different embedded platforms.
•The experimental results on parallelization of the HS algorithm on a high performance multi core Android systems, illustrate the usefulness of our methodology, with speedup up to 1.4.

Motivation

•Multimedia applications play an important role in our daily life.
•Hierarchical search is a fundamental algorithm in modern multimedia encoders.
•Investigate the flexibility of Python.
•Common parallelization methodologies, such as OpenMP, OpenMPI, GPGPU cannot be applied to a high-performance embedded systems.

Block Matching

A Block Matching Algorithm is a way of locating matching macroblocks in a sequence of digital video frames for the purposes of motion estimation.



HS Algorithm

•The motion vector search begins at the lowest resolution picture (level 2).
•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.

Why Python?

•Simple
•Interactive
•Cross platform
•Large number of libraries
•Fast prototyping
•High productivity

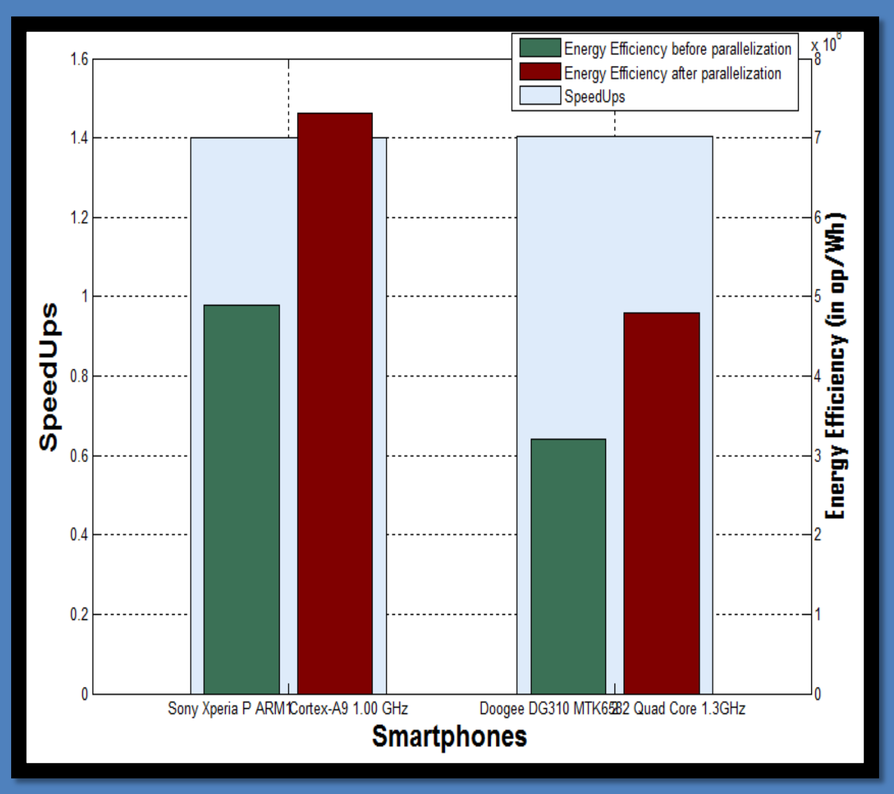
Why Multiprocessing?

•Python threads cannot work concurrently due to GIL.
•Multiprocessing enables true concurrent (parallel) work across multiple cores.
•As modern systems increase core counts, effective utilization of these cores is critical to performance.
•If data is restricted to each process significant performance gains are possible.
• Spawning processes and sharing data can have significant overhead.

Experimental Results

a) Smartphone Sony Xperia P LT22i
•Execution time before parallelization: 245.330 secs.
•Execution time after parallelization: 218.453 secs.
•Energy efficiency before parallelization: 4895586.20 operations/Wh.
•Energy efficiency after parallelization: 7318144.33 operations/Wh.

b) Smartphone Doogee DG310 MTK6582
•Execution time before parallelization: 175.0000 secs.
•Execution time after parallelization: 155.4600 secs.
•Energy efficiency before and after parallelization respectively: 3321853.64 and 4876582.23 operations/Wh.



Conclusions

•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.
•Parallel computing in Python is recent. There are many complications and limitations in the Multiprocessing library that needn't be there and will probably disappear in the future.

IEEE MOCAST 2016