The 36th International Symposium on Remote Sensing of Environment,
11 – 15 May 2015, Berlin, Germany, ISRSE36-354-1

VIRTUALIZING SUPER-COMPUTATION ON-BOARD UAS

E. Salamí[a,*], J. A. Soler[a], R. Cuadrado[a], C. Barrado[a], E. Pastor[a]

[a] Technical University of Catalonia (UPC), Computer Architecture Department, 08860 Castelldefels, Spain - esalami@ac.upc.edu

KEY WORDS: UAS, remote sensing, super-computing, parallelization, image processing, benchmarking, virtualization

ABSTRACT:

Unmanned aerial systems (UAS, also known as UAV, RPAS or drones) have a great potential to support a wide variety of aerial remote sensing applications. Most UAS work by acquiring data using on-board sensors for later post-processing. Some require the data gathered to be downlinked to the ground in real-time. However, depending on the volume of data and the cost of the communications, this later option is not sustainable in the long term. This paper develops the concept of virtualizing super-computation on-board UAS, as a method to ease the operation by facilitating the downlink of high-level information products instead of raw data. Exploiting recent developments in miniaturized multi-core devices is the way to speed-up on-board computation. This hardware shall satisfy size, power and weight constraints. Several technologies are appearing with promising results for high performance computing on unmanned platforms, such as the 36 cores of the TILE-Gx36 by Tilera (now EZchip) or the 64 cores of the Epiphany-IV by Adapteva. The strategy for virtualizing super-computation on-board includes the benchmarking for hardware selection, the software architecture and the communications aware design. A parallelization strategy is given for the 36-core TILE-Gx36 for a UAS in a fire mission or in similar target-detection applications. The results are obtained for payload image processing algorithms and determine in real-time the data snapshot to gather and transfer to ground according to the needs of the mission, the processing time, and consumed watts.

1. INTRODUCTION

Unmanned aerial systems (UAS, also known as UAV, RPAS or drones) have a great potential to support a wide variety of aerial remote sensing applications. UAS may allow addressing new remote sensing scenarios in which manned aircraft have never been introduced due to accessibility and/or risk reasons. Very low level UAS are today a reality in most developed countries for visual line of sight tasks such as electric power tower maintenance, precise agriculture or commercial publicity. Main advantages in front of manned aviation are the fast deployment, high precision data acquisition and low cost. In contrast UAS have more limitations on payload weight, flight endurance and power consumption.

Today most aerial works acquire data using on-board sensors for later post-processing. In a manned aircraft the payload operators can check the correct acquisition during flight. In a UAS the data gathered has to be downlinked to the ground to do similar verification. However, depending on the volume of data and the cost of the communications, this later option is not sustainable in the long term. This is not the case for the command and control link. For safety reasons the command and control link is mandatory. But the characteristics of this link are different than the payload link: high frequency (usually 20 Hz for downlink telemetry and 5 Hz for upload commands) and low bandwidth. For the payload transmission UAS usually employ radio-modems or WiFi for visual line of sight, and satellites for beyond visual line of sight. Transmission rates of such channels is limited to 56-128 Kbps for radio modems, 54 Mbps for typical WiFi and up to 150 Mbps for last generation WiFi families. Given the payload capabilities (fast acquisition rates, high precision sensors, hyper-spectral data, etc.) it is easy to realize that real-time transmission is not possible in most of the applications.

In this paper we focus in two UAS applications: hotspot and jellyfish detection. In a hotspot detection application the UAS operates in a survey pattern to cover the area of a forest looking for high temperatures. Although this application can be applied for the detection of forest fires, it has more benefits in an immediate post-fire situation. The UAS should obtain fast and useful information about the location of most critical areas. In this sense, the payload will need to have at least a thermal sensor, and the data to retrieve for the firefighters is the location of these areas as soon as possible. The execution of the on-board hotspot algorithm can help to provide the fast reaction required. The minimum information to download can be just the detected hotspots characteristics; this is, their geographic coordinates, magnitude and temperature. But it could help the end users to also obtain a visual image of the area augmented with the hotspot position. Considering that only a subset of the pictures are to be transmitted, the situation can be technically feasible, as we will show in the results section.

The other application we address is the detection of jellyfish shoals close to the shoreline. The application should respond very fast since the jellyfish are in movement and any delay on processing the data makes the information obsolete and useless. A payload with a high resolution camera, algorithms for image segmentation and pattern matching, and a fast data transfer containing only text alerts is the solution we propose for this application.

For both applications we develop the concept of virtualizing super-computation on-board the UAS. This is an automatic method to facilitate the speed-up of the payload algorithm execution and to ease the downlink of high-level information products. Rather than transferring the bulky raw data, an on-board high level computation capability will obtain information closer to the end-user. It becomes clear that recent developments in miniaturized multi-core devices could pave the way to on-board computation. This hardware shall satisfy size, power and weight constraints. Achieving high performance processing also requires hardware able to sustain high MIPS/MFLOPS per watt. Several technologies are appearing with promising results for high performance computing on unmanned platforms, such as the 36 cores of the TILE-Gx36 by Tilera (now EZchip) or the 64 cores of the Epiphany-IV by Adapteva.

The strategy for on-board super-computing is presented for the

*Corresponding author

two applications described above, with different variants and using the execution times of 8 algorithms in the TILE-Gx36 architecture. The paper also includes proposals on hardware selection, software architecture and communications aware design. The results are obtained using a service oriented software architecture where the payload processing services determine in real-time the data snapshot to gather according to the actual position of the UAS. Processing time and consumed watts are the main parameters measured to present the benefits of the virtualizing super-computation concept.

The structure of the paper is as follows: Section 2 presents four processing boards and their performance in terms of execution speed. Section 3 details our catalog of algorithms for the payload, describing eight image processing functionalities and showing their parallelization alternatives and performances. Section 4 explains the strategy for the on-board super-computation and how to incorporate it into our service oriented software architecture for UAS applications. The strategy uses the queuing theory to derive the best level of parallelism for a given set of application requirements. Finally section 5 concludes the paper and highlights some future work.

2. MULTICORES

The miniaturization of electronic devices is allowing the integration of multiple processing units in a single chip. Dual core, quad core and even hex and oct core chips are appearing in the market at very affordable prices. This increase of computer power allows to speed-up execution of running programs and/or the simultaneous execution of different programs. But the price to be paid for the computing power increase is an increase in power consumption.

The market of microprocessors has switched from computer to portable devices. According to (Barr and Massa, 2006) the market sells over 6 billion new microprocessors each year. Less than 2 percent of these microprocessors are used in general-purpose computers.The rest are for cell phone, tablets, smart watches, music players, cars, TVs, etc. For most of these portable devices the energy consumption becomes an important issue to consider and has to be balanced with the execution speed.

Previous work proposed models to evaluate this balance. In (Choi et al., 2013) a roof-line model of energy is proposed according to Amdahl's Law (Amdahl, 1967). The authors present a mathematical model to evaluate the speed-up and energy-balance points for processor instructions and also for memory operations with the aim to provide the best strategy for code allocation in terms of speed and energy. The best strategy has to be adequate to the hardware balance. Hardware balance depends on CPU and memory relative speeds and energy consumption, and on the size of the memory.

A similar approach is provided in (Cho and Melhem, 2010). This work derives simple formulas to describe the interplay between application speed-up and energy consumption. The authors apply an optimization strategy to decide the allocation of serial and parallel regions of an application. This optimization can be tuned to either minimize the total energy consumption or to obtain the best balance between time and energy.

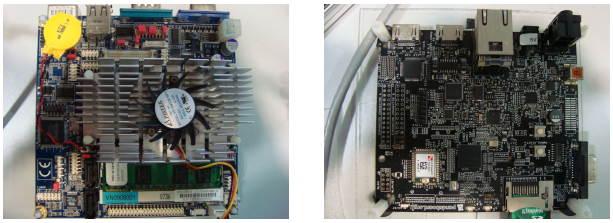
Some microprocessor can work at different clock frequencies, dissipating less energy at lower frequencies. They are known as power-aware processors, and for them a different strategy is presented in (Ge and Cameron, 2007). Using an analytical model for evaluating and predicting the performance and scalability of parallel applications, the authors propose strategies for decreasing the peak processor throughput. Results show that this strategy can save up to 30% of the energy at the cost of small performance losses.

Our approach will be similar to that presented in (Ge et al., 2009). With the aim to find an energy-performance efficient resource allocation for computing a given workload the paper first evaluates the energy performance and the efficiency of parallelization to decide about the best CPU selection.

2.1 Multicore processors boards description

In this research we will not work with frequency scalable processors. Our approach is to evaluate the speed and energy performances of 4 multi core boards in order to select the best configuration for the design of a payload architecture responsible for the UAS mission management and the on-board execution of the required image processing algorithms.

For this evaluation, the following processors/boards have been selected as on-board processing units in our UAS:



(a) Via C7 (b) Panda

(c) iT7500 (d) TILE-Gx36

Figure 1: Boards/processors used in the benchmarking

- EPIA N700-15 VIA C7-1.5 (**Via C7**): The VIA C7 is a single core, x86 architecture processor with a clock frequency of 1.5 GHz. Despite of being a general purpose processor, the integrated technologies such as CoolStream and StepAhead improve its power efficiency. The size of the board in which is integrated makes this processor suitable for embedded systems.

- Pandaboard with OMAP 4430 integrated CPU (**Panda**): The Pandaboard is a low-power development board based on OMAP 4430 system on a chip. OMAP system is built around Cortex-A9 ARM architecture microprocessor that runs up to 1.2 GHz clock frequency. In addition to the ARM microprocessor, OMAP 4430 integrates a 3D graphics accelerator and a programmable DPS for video encoding/decoding. ARM is a RISC architecture that, in contrast with x86 processors, require significantly fewer transistors, and thus, reduces costs, heat and power used. Pandaboard integrates all the systems required in a computer such as RAM memory, hard drive, and wireless communications, fact that makes it more power efficient but less hardware flexible.

- INTEL T7500 on GENE-9655 motherboard (**iT7500**): The iT7500 is a general purpose dual core x86 64-bits microprocessor at 2.2 GHz target to notebook computers. iT7500 uses advanced technologies such as C-States and Speed-Step to enhance power efficiency and allows systems to address more than 4 GB of both virtual and physical memory. iT7500 has been tested mounted in GENE-9665 embedded motherboard which offers high hardware configuration capabilities to adapt systems to mission requirements.

- TilenCore-Gx board with TILE-Gx36 (**TILE-Gx36**): The TilenCore-Gx board integrates a TILE-Gx36 CPU which is a 36 cores processor of 64-bits, at 1.2 GHz. The TILE-Gx36 is a RISC architecture optimized for networking and multimedia applications. The multi-core design includes a communication network mesh architecture able to scale to hundreds of cores on a single chip. It is a general purpose CPU oriented to good power efficiency. Principal uses of TILE-Gx36 are networking equipment, including intelligent routers and firewalls, and cloud computing applications such as web indexing, data-mining and multimedia applications as for example broadcast video servers.

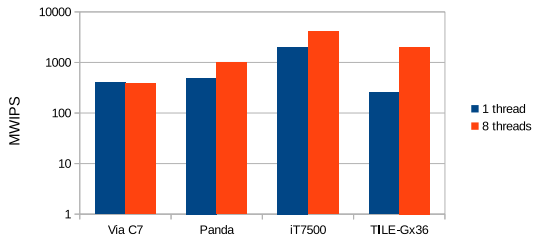
2.2 Benchmarking results

We have applied three test benchmarks to all the above boards. All of them are synthetic loops aimed at the benchmarking of parallel boards. The first two are classic well-known codes: Whetstone (Curnow and Wichmann, 1976) and Dhrystone (Weicker, 1984). Whetstone was adapted from its former version in Algol 60 and Fortran by the Technical Support Unit of the Central Computer and Telecommunications Agency (TSU/CCTA). Whetstone aims to measure the performance of scientific applications and thus is composed by a number of loops with floating points operations (add, mult, sin, log, etc.) In contrast Dhrystone, created as a replica of Whetstone for non-numerical applications, contains a number of loops with arithmetic operations, conditionals, pointer references and so on. Both invent a shelf measuring unit, MWIPS and DMIPS, which provides the number of millions of instructions per second for each type of benchmark.

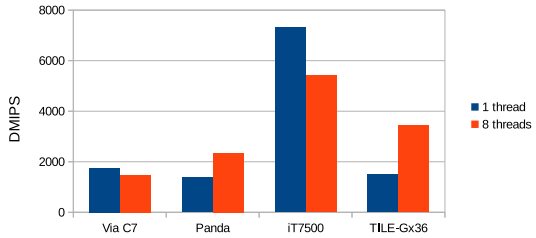
The third benchmark is a mix of both that we have constructed with a simple structure. We execute 500 tasks, each containing 5 operations (2 floating point and 3 integer operations) that execute in a ten millions of iterations loop. We provide the results in MIPS (million of instructions per second) by dividing the fix number of instructions (25,000 million) by the seconds needed to execute them. The loop kernel has no data dependency and thus is fully parallelizable.

As Figure 2 shows, the maximum instructions per second in all three benchmarks in a sequential execution (1 thread) is obtained by the iT7500 processor, the processor with the highest clock frequency. On the other side the TILE-Gx36 is the processor with less throughput, but similar to the VIA C7 and Panda. In contrast, TILE-Gx36 is the one able to obtain more speed-up when executed over 8 threads, which is reasonable since the other boards have only one or two cores while TILE-Gx36 has thirty six. Note that MWIPS in Whetstone are shown in logarithm scale. Note also that some examples with 8 threads (Via C7 and iT7500 for Dhrystone) execute poorless compared with sequential.

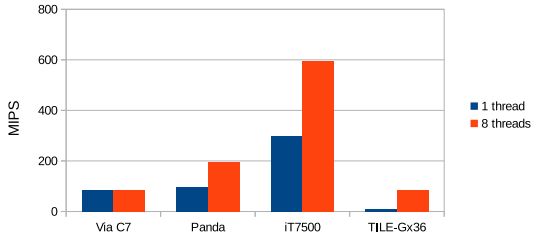
We have used the ad-hoc loops to measure the power consumption of each board. Figure 3 shows the execution times (3a), the power consumption (3b) and the efficiency (3c) for 1 to 36 threads executions. Figure 3a extends what we could observe in previous MIPS plots, basically the effect of the number of processors of



(a) Whetstone



(b) Dhrystone



(c) Ad-hoc 500 tasks with a 50 M instructions loop

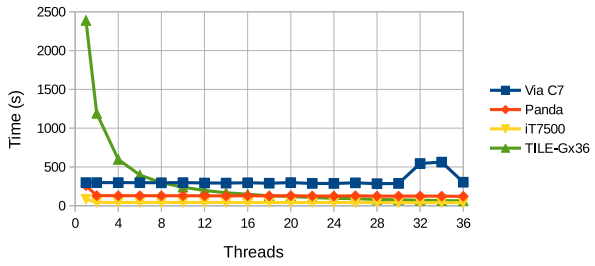
Figure 2: Benchmarking results in MIPS

each board: Via C7 with one processor executes always with the same time except for 2 anomalies at 32 and 33 threads, Panda and iT7500 notice some speed-up only with 2 threads, and TILE-Gx36 time execution is a inverse exponential plot that crosses the rest of the plots at 8, 18 and 36 threads. This means that although TILE-Gx36 is not a fast processor, when scaling with 8 threads it is comparable to Via C7, with 18 threads to a Panda and with all 36 threads to iT7500.

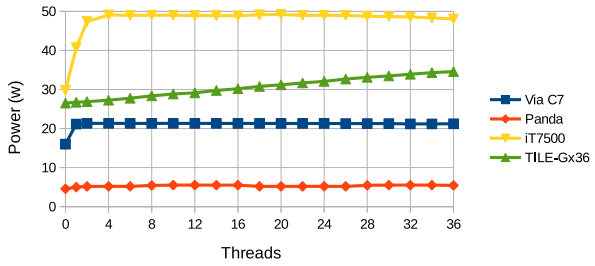
During such executions we have measured the power consumption using a programmable uninterrupted power supply (UPS) used in automobile industry. This system is used as part of the on board electronics to switch from batteries into an on-board alternator. The results, given in Watts, show that the price to pay for iT7500 being the fastest processor is a higher consumption compared to the rest of boards. The Panda board is the one with the lowest consumption, with a very small difference when executing with one or two cores. In the middle we found the Via C7 and TILE-Gx36, the second with a linear increment of 0.3 Watt per core.

To fuse both data, time and power, we defined the efficiency as the millions of instructions per second that each board can execute with 1 Watt. Figure 3c shows these results. We can observe that best efficiency is given for Panda. Only TILE-Gx36 has a positive behavior with the increase of parallelism, although the final efficiency stays always below the second most efficient pro-

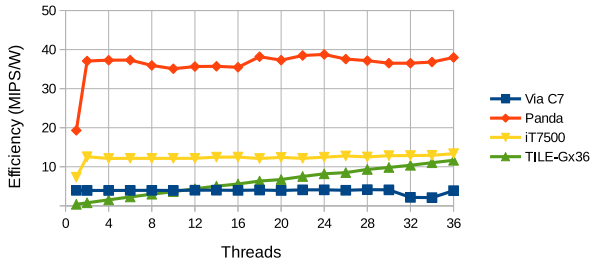
cessor, the iT7500.



(a) Execution time



(b) Power consumption



(c) Efficiency

Figure 3: Benchmarking scalability

As a conclusion of the benchmarking done we decided first to select the iT7500 board for executing real-time flight-related programs, at the cost of more consumption. Then for the non real-time programs, basically those related to payload processing, we decided to board the TILE-Gx36 given its better scalability.

3. ALGORITHMS CATALOGUE

For the evaluation, we have selected a set of data processing algorithms that may be of interest in a fire mission or in similar target-detection missions. All programs have been developed using C++ and OpenCV (Bradski, 2000), an open source computer vision framework released under a BSD License.

3.1 Algorithms description

Our catalogue of image processing tools accounts for a total of eight algorithms that can be classified into four application areas: *detection* (hotspots and jellyfish), *georeferencing* (georef and geotiff), *panoramas* (fusion, mosaic, and stitch) and *selection* (quality).

- **Hotspots** is a simple segmentation algorithm to detect hot spots in thermal images (Salamí et al., 2009). Pixels over a given threshold temperature are grouped into hotspots. For each hotspot, the algorithm annotates information about its center of mass, bounding box, number of pixels, and temperature. The result is a text file with the information of the detected hotspots. Bounding boxes are also marked as green rectangles on the output image.

- **Jellyfish** is a more complex segmentation algorithm to detect large jellyfish shoals on coastlines using aerial high resolution images (Barrado et al., 2014). For each image, an initial phase is used to intensify the pixel differences. This is done by running first a decorrelation stretch for adjusting pixel intensity values; and second, a contrast stretching transformation. Then, color segmentation is applied in order to find the regions of interest of the captured image. Grouping is done by applying first an opening morphological process and afterwards by using connected component labeling. The program computes the different properties of the groups, such as the area, number of pixels and bounding box. Bounding boxes are also marked on the output image.

- **Georef** determines the geographic coordinates of a given pixel using direct georeferencing as described in (Salamí et al., 2013). The input data is the position and attitude of the aerial platform at the acquisition time, the external and internal parameters of the camera, and the terrain elevation. Note that this algorithm performs floating point computation, rather than image processing tasks.

- **Geotiff** writes the georeferenced TIFF image. The algorithm first compensates the camera lens distortion, and then rectifies the image using the geolocation of the four corners to compute the homography. Finally, a TIFF file with georeferencing information is written using the geotiff library (GeoTIFF WWW Homepage, n.d.). The input data are the image, the position and attitude of the aerial platform at the shot time, the external and internal parameters of the camera, and the terrain elevation.

- **Fusion** creates a new image overlapping thermal information over the visual image. Both, thermal and visual images are first rectified and properly scaled. Georeferencing information is then used to compute the position of the thermal image over the visual one. Weighted addition is performed in the overlapping area. Finally, the resulting georeferenced TIFF image is written. Input data include the images, the position and attitude of the aerial platform, the external and internal parameters of both cameras, and the terrain elevation.

- **Mosaic** composes a panorama using georeferencing. The images are added to the resulting mosaic one by one. Each new image is rectified and scaled. Georeferencing information is then used to compute the position of the image in the mosaic. A weighted addition is performed in the overlapping area. Finally, the resulting georeferenced TIFF image is written. As in previous algorithms, input data include the position and attitude of the aerial platform for each shot, the external and internal parameters of the camera, and the terrain elevation.

- **Stitch** composes a panorama using the stitching algorithm in OpenCV, which is based on the method proposed by M. Brown and D. Lowe (Brown and Lowe, 2007). It uses invariant local features to find matches between all of the images. Note that the objective is not to compose the panorama

of the entire overflown area, but the stitching of small patches of visual or thermal images, such as the images where a hotspot is visible.

- **Quality** measures the quality of an image based on blur and entropy metrics. A blurred image is one whose edges and shapes are not clearly defined. The blur metric computes a sharpness grade based on the number of edge pixels detected by the Canny function. The image is first blurred to reduce the amount of noise present in the image, eliminating many spurious edges. The entropy of the image is used to detect over or underexposure. If the average image brightness is too high or too low, it becomes a homogeneous image (low entropy). The normalized HSV histogram is used to obtain the probability density function of the colors in the image.

3.2 Parallelization

Different approaches exist for transforming a sequential program to run in a highly parallel mode. The basic requirements for an algorithm to run in parallel are the Bernstein's conditions (Bernstein, 1966), the first and most critical of these conditions is that 2 programs cannot run in parallel, if the input variables of one program depend on the output variables of the other program. When considering a complex algorithm, like the stitching method, there exists a critical path of tasks that need to be finished for the algorithm to continue. The fraction of the code that cannot be parallelized will limit the potential speed-up of the parallelization (Amdahl, 1967).

Another important aspect to consider when applying parallelization is granularity. Granularity refers to the size of the partitioned tasks. Fine granularity means the algorithm is separated into small tasks, while coarse granularity represents the opposite. There is a design trade-off when deciding this factor: coarse granularity increases work imbalance, since individual tasks take longer to compute it can happen that some cores are kept waiting. Fine granularity solves this problem by reducing the time of individual tasks, but introduces an overhead in creating threads and communicating between them.

In this initial evaluation we have not spent additional efforts to exploit the parallelism of the applications. We have only taken advantage of the possibilities that OpenCV offered us. OpenCV implements different parallelization frameworks. We choose Intel TBB (Reinders, 2007), since it was recommended by the hardware manufacturer. TBB is a framework that implements dynamic parallelization, this means that the work is balanced during execution time instead of being fixed in the code. This reduces work imbalance, since idle cores can *steal* tasks from other threads. For the chosen architecture, this is beneficial, since manually balancing the load between 36 cores is complicated. However, the overhead introduced by TBB grows with the number of cores (Contreras and Martonosi, 2008).

For the analysis of the performance, it is important to consider how the OpenCV libraries implement TBB. When a program calls an OpenCV function, TBB is not immediately used, the function will run in a sequential manner. At some point, if the function needs to perform a CPU intensive computation, the function `cv::parallel_for` is called, which then initializes TBB to start distributing the workload. As discussed before, this introduces a limitation in the performance increase of the parallelization.

As a case of use, we analyze the code of the stitching algorithm to see which portions actually use the TBB framework. Table 1 identifies the different processing kernels and shows the time necessary to compute each step on a single thread, averaged over 100 samples. Last column specifies which parts of the algorithm really made use of TBB parallelization. The test was made with sets of 4 images on an Intel cpu.

Code section	Time (s)	Time (%)	TBB
Finding features	11.304323	58.78%	Yes
Pairwise Matching	3.757168	19.53%	Yes
Estimating Rotations	0.000088	0.00%	No
Warping Images	0.031560	0.16%	No
Exposure Compensation	0.048385	0.25%	No
Finding Seams	0.255811	1.33%	No
Compositing	3.831968	19.92%	Yes

Table 1: Coverage of the different stitching algorithm sections

We can see that not all the steps in the pipeline use TBB, but the most significant do. *Finding features, Pairwise Matching and Compositing* account for a 98.25% of the total computing time in this example. However, the TBB only comes active in the CPU intensive parts of this steps and a portion of this steps still execute concurrently. In ideal conditions, according to Ahmdal's Law, the maximum possible increase in performance in this situation is about 57X.

3.3 Performance evaluation

This section presents preliminary results on the use of TILE-Gx36 for parallelizing the data processing algorithms. Table 2 shows the execution time of the algorithms when only one core of TILE-Gx36 is used. Each algorithm was run over one hundred images, with resolutions of 320×240 pixels for thermal images, 5 MP for high resolution images, and 16 MP for very high resolution ones. Sets of eight images were used to compose panoramas in Mosaic and Stitch. For each algorithm, the average execution time per image and standard deviation in seconds is given. Note that, for Mosaic and Stitch, the total time to compose the panorama is eight times the value given in the table.

Algorithm	Input image resolution	Execution time	
		Average	Std. dev.
Hotspots	0.08 MP	0.026 s	0.002 s
Jellyfish	16 MP	116.761 s	1.655 s
Georef	5 MP	0.002 s	0.000 s
Geotiff	5 MP	7.035 s	980 s
Fusion	5 MP, 0.08 MP	11.771 s	1.725 s
Mosaic	8×5 MP	8.966 s	2.473 s
Stitch	8×5 MP	105.777 s	106.123 s
Quality	5 MP	2.104 s	0.009 s

Table 2: Input data sets resolution and sequential execution time per image in TILE-Gx36

The graphical representation in Figure 4 highlights the great difference in execution times (note that the time axis is in logarithmic scale). Hotspots and Georef are the only ones that exhibit low enough execution times to be used in real-time at the selected acquisition frequency. The total execution time of Quality could be reduced by analyzing only a limited zone in the image. Jellyfish and Stitch are the most time consuming benchmarks, but they are the most complex too. Mosaic is about twelve times faster than Stitch. The quality of Mosaic composition is lower (see Figure 5), but may be sufficient for many applications. The main drawback of the Stitch algorithm, apart from the time cost, is its high variability (from 12 seconds to 300 seconds for different data input sets).

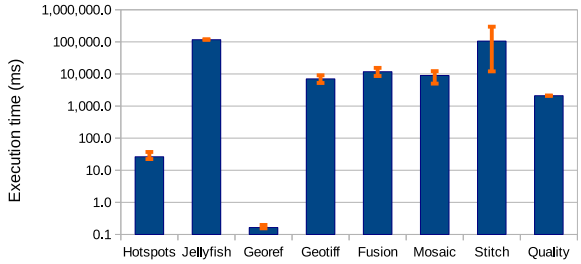
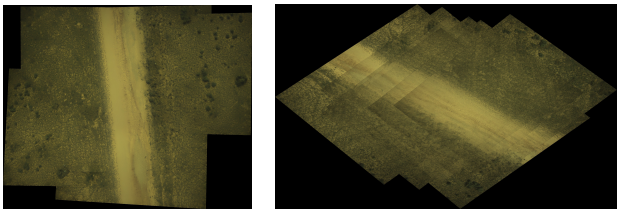


Figure 4: Algorithms sequential execution time in TILE-Gx36 (orange lines represent minimum and maximum values)



(a) Stitch (b) Mosaic

Figure 5: Stitch and mosaic compositions

Figure 6 shows the performance speed-up when the number of threads is increased to 2, 4, and 8 with respect to the sequential execution with one thread. The maximum improvement is achieved by Geotiff algorithm, with a speed-up of 2.3X for the 8-threads execution, followed by Fusion and Mosaic, with speed-ups of 1.9X and 1.7X respectively. All three algorithms use the OpenCV *warpPerspective* function, which is parallelized with the TBB library. No improvement is observed for Jellyfish, despite working on very high resolution images. This is because it does not make use of TBB optimized functions. Additional hand effort is required if we want to exploit parallel execution. The Stitch algorithm exhibits a highly variable execution time, with individual 8-threads runs with speed-ups ranging from 0.91X to 4.51X. Furthermore, a significant drop in performance when executing with 4 threads can be observed.

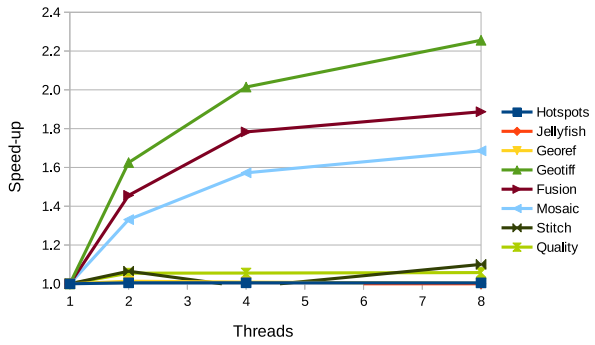


Figure 6: Algorithm performance speed-up in TILE-Gx36

Another approach is presented in Figure 7. This plot shows the performance speed-up when the number of threads is increased to 2, 4, and 8, but the parallelization approach is now at the application level. We obtain almost perfect speed-ups when we assign a different image to each core, in the figure shown for the Jellyfish detection algorithm. The same will happen for different applications running in separate cores.

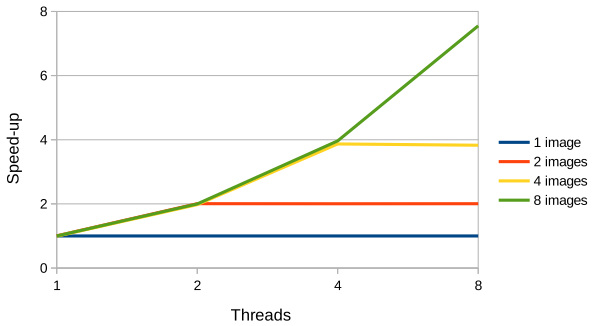


Figure 7: Speed-up in the TILE-Gx36 for application-level parallelism

From the above results, it can be concluded that coarser grain parallelism, such as application level, is better exploited in this parallel board than the fine grain parallelism, such as OpenCV TBB multi-threading. The Fusion algorithm, for example, takes about 12 seconds to process one image in one core; but if a new request for Fusion arrives before the previous one is over, it can be started in a different core. Assuming that the 36 cores of TILE-Gx36 were fully dedicated to Fusion, this means that one request can be attended every 0.33 seconds. For the Jellyfish algorithm, sequential execution time is one order of magnitude higher, but still with application level parallelization over 36 cores, we are able to process one new image every 3.33 seconds.

4. VIRTUALIZATION

This section presents the virtualizing super-computing approach. As an example, the strategy for on-board super-computing is described for the hotspots and jellyfish missions.

4.1 Software Architecture

The software architecture of our UAS follows the distributed architecture paradigm. Each functionality is isolated in a software agent or service which executes independently over a communication bus or middleware. Figure 8 shows a generalist view of the approach. The most important services are the Mission Manager, the Flight Plan Manager and the Payload Processing. The Mission Manager is responsible for the interaction between the flight route and the payload. The Flight Plan Manager feeds the UAS autopilot with the sequence of waypoints required to flight the area of interest with the adequate parameters of speed, altitude and surveillance pattern. The Payload processing manages the on-board cameras and drives the image storage.

To manage the virtualizing super-computing we have defined a new software agent connected with the Payload processing. Using a configuration file with the algorithm benchmarking data, the service dispatches the images to the most efficient computation system. Availability of the hardware, power consumption and real-time requirements are also considered before dispatching an image processing algorithm. For the results presented in this paper we have configured the TILE-Gx36 as the virtual super-computing board, while the iT7500 board is in charge of executing the rest of software services of the UAS.

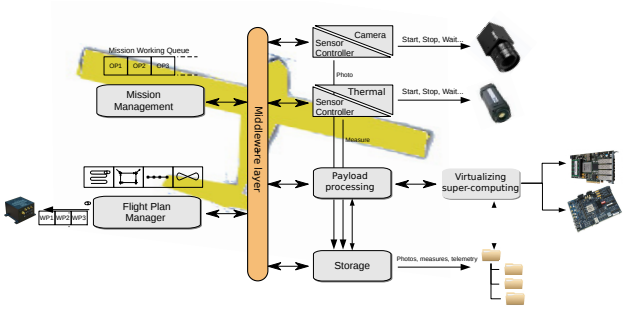


Figure 8: Virtualizing super-computing architecture

4.2 Super-computing strategy

Section 3 showed that not all algorithms scale properly. While three of the algorithms have some scalability properties (Geotiff, Fusion, and Mosaic), other ones exhibit a sequential behavior (like Jellyfish). Having a number of 36 cores of TILE-Gx36 available and with a low penalty in power for each additional core in use, the question is which is the best parallelization strategy. We base our choice on the results given by an M/M/C model of queuing (Gautam, 2012).

We have a system of queues on which images act as incoming clients who need a service. The services are provided by the execution of the algorithms in the processing cores. The queuing theory allows us to obtain information about the level of service of such system. Output values of interest are the average time in the system (W), average waiting time (W_q), average clients in the system (L), average customers in queue (L_q), and server utilization (ρ).

Let's focus on Fusion, although the following arguments apply to any of them. Suppose the frequency of arrival of the images is a Markov entry process with arrival rate (λ) equal to 1 image per second. The service rate (μ) is also a Markov process defined by the execution time of the algorithm, that is the inverse of the number of seconds of the execution. The scheduling policy is first-in first-out. Finally the number of cores is set to 32. We applied the following M/M/C queuing models:

- Sequential: arrival rate $\lambda = 1$ Hz,service rate $\mu = 1/11.8s = 0,085$ Hz, $C = 32$ servers (cores).
- Parallel 2T: arrival rate $\lambda = 1$ Hz, service rate $\mu = 1/8.1s = 0.123$ Hz, $C = 32/2 = 16$ servers of 2 cores.
- Parallel 4T: arrival rate $\lambda = 1$ Hz, service rate $\mu = 1/6.6s = 0.152$ Hz, $C = 32/4 = 8$ servers of 4 cores.
- Parallel 8T: arrival rate $\lambda = 1$ Hz, service rate $\mu = 1/6.2s = 0.161$ Hz, $C = 32/8 = 4$ servers of 8 cores.

Table 3 shows the obtained values for the four models. First conclusion is that the parallel version with 8 threads is not an option, as the queues will tend to infinity. In general, this happens when λ is greater or equal than C times μ. Second, the best option in terms of execution time is clearly the parallel with 2 threads, with an average time in the system of 8.11 seconds. Power consumption increase compared to the sequential model is minimal. Furthermore, the average number of customers in queue (0.01) indicates that small sized buffering structures are required. The parallel with 4 threads is worse in terms of both time and power. Based on the results, the virtualizing agent would dispatch the 2 threads release.

Model	L	L_q	W	W_q	ρ
Sequential	11.80	0.00	11.80	0.00	0.369
Parallel 2T	8.11	0.01	8.11	0.01	0.506
Parallel 4T	9.02	2.42	9.02	2.42	0.825
Parallel 8T	∞	∞	∞	∞	∞

Table 3: Parallelization strategies of Fusion with 32 cores

4.3 The hotspots mission

As a case of use, we describe the parallelization strategy for the hotspots mission. The UAV scans an area taking thermal and visual images. Low-quality visual images are discarded. Thermal images are processed on-board in real-time. If a hotspot is detected, a notification including the geographical position and a estimation of the magnitude of the hot spot is sent to the ground segment. If the corresponding visual image is available, the fusion of the thermal and visual images is also sent. Four of the evaluated algorithms are involved in the process: Hotspots, Georef, Quality, and Fusion. From now, we will suppose that georeferencing is integrated into Hotspots.

Assuming a realistic frequency of one pair of photos every second, Hotspots execution time (28 ms including georeferencing) is low enough to run in one dedicated core maintaining the input rate. Quality, however, needs at least three cores to be able to process visual images as they arrive, as it takes 2.1 seconds per image. There are still 32 cores available for Fusion. According to Table 3, the 2 threads parallel execution appears as the candidate of choice for Fusion.

4.4 The jellyfish mission

The jellyfish mission focuses on the detection of jellyfish shoals close to the shoreline. The primarily objective is to be able to daily inform the coast guards and the citizens about the proximity of jellyfish shoals. It also allows to obtain relevant statistics about the jellyfish proliferation and their movements. Unlike the hotspots mission, there is a unique algorithm considerably slower (117 seconds), and with none intrinsic parallelism. The question here is whether there is a feasible input rate (λ) at which the system is not saturated. As stated previously, the queues will tend to infinity when λ is greater or equal than C times μ. Thus, the upper limit for λ is

$$\lambda < C \times \mu = 36 \times 117^{-1} = 0.308 \text{ images/s}$$

Applying the queuing theory for arrival rates under the computed value, the results on Table 4 are obtained. Obviously, as the arrival rate is close to the limit, it produces too high values for the queue occupancy and service time. However, arrival rates of one image every 4 or 5 seconds leads to an appropriate behavior of the service.

λ	L	L_q	W	W_q	ρ
0.30	67.46	32.36	224.87	107.87	0.975
0.25	29.96	0.71	119.84	2.84	0.813
0.20	23.42	0.02	117.10	0.10	0.650

Table 4: Jellyfish M/M/C model with 36 cores for different image frequencies

We believe this is an acceptable rate for this particular mission. Note that the mission has no special constraint on image overlapping. Furthermore, flight speed and altitude, together with photo shot frequency can be dynamically adjusted by the mission manager depending on the results of the detection.

5. CONCLUSIONS

In this paper we have evaluated four processing boards used separately in some of our UAS experiments. The size and weight characteristics of any electronics to board, but also the power consumption and computing performance are crucial characteristics to consider. With the obtained performance figures we decided to design a payload bay in which our distributed software architecture can execute. The faster processor iT7500 was devoted to the mission and flight management services, while the TILE-Gx36 multi-core was specifically dedicated to execute the payload algorithms.

Eight image processing algorithms were tested as payload algorithms for target detection, selection, georeferencing and creation of panoramas. All eight algorithms use the OpenCV libraries compiled for parallel execution. Results showed that, in general, the obtained parallelism was very limited. Five out of eight algorithm showed almost no speed-up. The other three algorithms (geotiff, fusion and mosaic) obtained some performance when devoting 2, 4 or 8 cores, but only in one case we obtain an speed-up greater than 2. This results do not mean that better parallelization is not possible, they just demonstrate that the task to speed-up an application is not an easy task. Parallelization requires a deep knowledge of the code and of the hardware architecture in which it will run. Moreover, it requires development time. Only if the algorithm will execute many times, over the same hardware, the parallelization time is worthwhile.

The parallelization strategy we present in this paper is suitable for dynamic missions, with alternative algorithms being selected at flight time. Using queue theory we develop a super-computing strategy where fine- and coarse-grain parallelism can coexist in benefit of the final user-oriented necessities. At the price of larger individual latencies, we obtain a system able to attend an incoming set of arrival images without collapsing the application. Key parameters are arrival frequency, processing times and number of resources. Understanding the processing time and the maximum number of available resources, a simple arithmetic operation can derive the best option for parallelization. The virtualizing super-computing service is the new agent of the UAS software architecture devoted to this task.

In future work we will introduce more alternatives for hardware selection. New boards shall be evaluated such as the 64-core Epiphany-IV, or the small, low cost and low power Raspberry Pi. New payload bay designs, with several co-processor boards on-board, shall result from this evaluation. Also, the virtualizing super-computing service must be developed to increase the focus on power consumption as metrics for the selection of the best strategy for the payload processing. This will also include real-time restrictions, which could be also incorporated into the service. In parallel to this, efforts to improve the fine-grain parallelism will progress, together with the incorporation into our catalog new payload processing algorithms required in future UAS applications.

ACKNOWLEDGEMENTS

This work has been funded partially by Ministry of Education of Spain under contract number TRA2013-45119-R and partially by NASA's "IPM: A Multicore Intelligent Payload Module for High Performance, Onboard Data Processing" project under the NNH11ZDA001N-AIST Advanced Information Systems Technology Program.

REFERENCES

Amdahl, G. M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings AFIPS 1967 Spring Joint Computer Conference, pp. 483–485.

Barr, M. and Massa, A., 2006. Programming embedded systems, 2nd edition. " O'Reilly Media, Inc.".

Barrado, C., Fuentes, J. a., Salamí, E., Royo, P., Olariaga, a. D., López, J., Fuentes, V. L., Gili, J. M. and Pastor, E., 2014. Jellyfish monitoring on coastlines using remote piloted aircraft. IOP Conference Series: Earth and Environmental Science 17, pp. 012195.

Bernstein, a. J., 1966. Analysis of Programs for Parallel Processing. IEEE Transactions on Electronic Computers EC-15(5), pp. 306–307.

Bradski, G., 2000. The opencv library. Dr. Dobb's Journal of Software Tools.

Brown, M. and Lowe, D. G., 2007. Automatic panoramic image stitching using invariant features. International Journal of Computer Vision 74(1), pp. 59–73.

Cho, S. and Melhem, R., 2010. On the interplay of parallelization, program performance, and energy consumption. Parallel and Distributed Systems, IEEE Transactions on 21(3), pp. 342–353.

Choi, J. W., Bedard, D., Fowler, R. and Vuduc, R., 2013. A roofline model of energy. In: Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 661–672.

Contreras, G. and Martonosi, M., 2008. Characterizing and improving the performance of Intel Threading Building Blocks. 2008 IEEE International Symposium on Workload Characterization, IISWC'08 pp. 57–66.

Curnow, H. J. and Wichmann, B. A., 1976. A synthetic benchmark. The Computer Journal 19(1), pp. 43–49.

Gautam, N., 2012. Analysis of Queues: Methods and Applications. CRC Press.

Ge, R. and Cameron, K. W., 2007. Power-aware speedup. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, IEEE, pp. 1–10.

Ge, R., Feng, X. and Cameron, K., 2009. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In: Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pp. 1–8.

GeoTIFF WWW Homepage, n.d. http://www.remotesensing.org /geotiff/geotiff.html. Accessed March 2015.

Reinders, J., 2007. Intel Threading Building Blocks. First edn, O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Salamí, E., Barrado, C., Pastor, E., Royo, P. and Santamaria, E., 2013. Real-time data processing for the airborne detection of hot spots. Journal of Aerospace Information Systems pp. 444–451.

Salamí, E., Pedre, S., Borenzstejn, P., Barrado, C., Stoliar, A. and Pastor, E., 2009. Decision support system for hot spot detection. In: Intelligent Environments 2009, Ambient Intelligence and Smart Environments, Vol. 2, IOS Press, pp. 277–284.

Weicker, R. P., 1984. Dhrystone: A synthetic systems programming benchmark. Commun. ACM 27(10), pp. 1013–1030.