Managing power amongst a group of networked embedded FPGAs using dynamic reconfiguration and task migration

*David Kearney & Mark Jasiuna*s
{david.kearney, mark.jasiunas]@unisa.edu.au
Reconfigurable Computing laboratory
School of Computer and Information Science
University of South Australia

Abstract

Small unpiloted aircraft (UAVs) each have limited power budgets. If a group (swarm) of small UAVs is organised to perform a common task such as geo-location, then it is possible to share the total power across the swarm by introducing task mobility through an ad hoc wireless network where the communication encoding and decoding is done on FPGAs. This paper describes the construction of a distributed operating system where partial dynamic reconfiguration and network mobility are combined so that FPGA tasks can be moved to make the best use of the total power available in the swarm.

1. Introduction

The term swarm is usually identified with a group of living organisms who arrange themselves to cooperate to achieve a common task that no one of them could complete as an individual. For example a swarm of birds may fly in a slipstream formation to save on energy or a swarm of ants will construct a shortest spanning tree path between a food source and their nest[1]. UAVs that cooperate to achieve a common task (such as geo-location) in an autonomous way (using agents) have been given by analogy the title of swarm in this paper.

Small UAVs (of weight less than 25kg and wingspan less than 3m) are often limited by their resources as compared with larger manned and unmanned planes. However combining the capabilities of several small UAVs can produce a useful capability. A key contributor to this capability is to fit each UAV with an embedded reconfigurable computer consisting of a small microprocessor and an FPGA. This allows many of the tasks of the UAV to be performed on board whilst using only a modest amount of electrical power. Typical tasks that a small UAV may be required to perform include image processing and tracking, feature identification, encryption and compression, and coding for data transmission. Many of these algorithms can be performed on an FPGA more effectively than a microprocessor.

In this paper we introduce the concepts of sharing a single FPGA among different tasks that may not need to execute at the same time and allowing such tasks to migrate between members of the swarm either to share power across the swarm or provide for the replacement of members of the swarm who may need refuelling without stopping the execution of tasks critical to the swarms mission.

Dagstuhl Seminar Proceedings 06141
Dynamically Reconfigurable Architectures
http://drops.dagstuhl.de/opus/volltexte/2006/740

The paper is organized as follows. In the first section we review the literature on capabilities and applications of small UAVs and the compute platforms they might use. We examine publications that report the benefits swarms of UAVs. We show that whilst there have been many publications of swarm applications there has been less attention to the resource sharing possibilities of swarms especially extensions to compute sharing and power sharing. In section 2 we introduce a typical scenario where power and FPGA computer resource sharing could be beneficial in a swarm of UAVs performing a surveillance function.

Sections 3 gives a typical scenario for mobility across a swarm. Section 4 introduces infrastructure for mobility of applications between UAVs. We explain why we have opted for agent based decentralized control of mobility and fuzzy rules for the decision making. We describe check pointing of applications.

2. Reconfigurable computing in swarms of UAVs

In this section we first discusses the capabilities of and applications to which small UAVs have been applied. We describe the computing requirements for a small UAV performing these applications. We show from the literature that scarce resources for small UAVs include electrical power and high performance computing capability. We give examples form the literature that show how power can be minimized and computing capability maximized on a single UAV by the use of FPGAs on UAVs in preference to more traditional software only embedded systems. Next we investigate the advantages that a swarm of UAVs has over single platforms in overcoming small UAV limitations. We give examples of how a swarm can improve application performance in geo-location by using the diversity of sensor locations. We highlight that there is no literature of the use of a swarm to share the scarce resources that support these types of applications. In particular there has been no investigation of the sharing of power and high performance embedded computing resources across the swarm. The final section reviews the literature on the sharing of the types of embedded FPGA compute resources that are used on small UAVs. Using our definition of partial dynamic run time reconfiguration we show how published operating systems for reconfigurable computing might allow the sharing of FPGA resources among many applications in UAVs applications. We note that the literature does not contain specific work on the extension of FPGA application sharing in a distributed sense across several FPGAs. These topics are the subject of this paper.

Uninhabited airborne vehicles are projected to become a major segment of the aviation industry over the next 20 years [2], primarily enabled by developments in computing, communications and sensor technologies. An area where UAVs will likely make a major impact is in surveillance and remote data collection. Examples of applications include fire ground (active bushfire) surveillance [3] crop and vegetation surveying [4], emergency data communications and maintaining the security of people and assets against terrorist related threats [5]. Small UAVs (of gross mass less than 25kg) will most likely perform these tasks, working together in closely co-located teams called swarms. This is because swarms can carry a range of sensors, and their diversity overcomes the limited field of view of a single small UAV flying at a relatively low altitude. Swarms also provide increased reliability through redundancy.

The sensors used on small UAVs have in the past been confined to very light-weight devices. For example video cameras and small RF sensors are quite practical on small UAVs. However, it is clear from studies conducted on large UAVs [6] and satellites that more complex sensors such as infrared imagers could provides a major improvement in the quality of information that can be gathered [7].

A reconfigurable computer is a processing platform consisting of a general purpose processor interfaced to memory and a programmable logic device PLD [8]. The most widely used PLD is a field programmable gate array (FPGA) [9]. An FPGA is an array of logic cells connected via programmable routing. Each logic cell can be configured to perform logic functions allowing complex circuits to be constructed. FPGA's are ideal for implementing common types of algorithms on UAVs [10][11][12][13] [14].

Small inexpensive UAVs have been found usefullin military roles. They can be considered somewhat expendable, allowing swarms to operate in closer proximity to threats where sensors and effectors are more effective and operate using less power[15]. Once such area of research is electronic warfare where the goal is to gather information and suppress the enemy's information gathering using electronic sensors and effectors (jamming). For example, several UAVs can be used to geo-locate the position of radar emitters for suppression [16]. A UAV cn fly much closer to a radar emitter making jamming possible at very low power. While the prospect of armed UAVs in combat roles has been explored, the current focus remains on intelligence, surveillance and reconnaissance missions[17].

Geo-location is a good example of the benefits of swarms. It requires the cooperation and exchange of information between several UAVs. Geo-location works by taking a directional bearing of an object from a number of different locations and combining them to determine the objects exact position. Finn et al. describes how a group of 6 sensors can reduce the location error by more than 80% (figure 1).

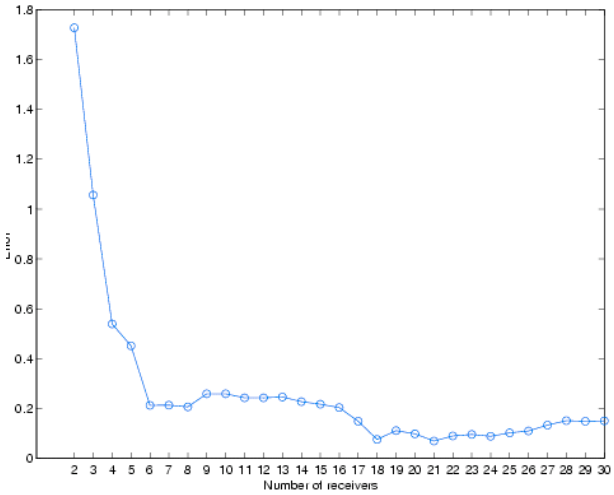

Figure 1: Reduction in the location error margin (Y axis) with the number of sensors (X axis) used to determine the location. [22]

3. Sharing resources in a swarm a typical scenario

The missions of UAV swarms can be divided into two classes. In the single mission we have a swarm requiring N planes each with different capabilities to perform the swarm function. We have just N planes available. We deploy these planes and attempt to arrange their computing tasks so that all planes run out of fuel at the same time. Allowing for fuel to return to base (assumed the same for each plane) we end the deployment when each plane has just this much fuel left. The aim is to maximize the time that the swarm is deployed over the target area doing useful work.

In the continuous mission scenario N planes are required to form the swarm but we assume that we have more than N planes available. Thus it is possible maintain a continuous mission by retiring planes from the swarm that are running low on fuel and replacing them with other planes with a full fuel load provided the planes are reconfigurable. The objective in this case is for example to maintain continuous surveillance over the targe area. Task mobility can improve the time a swarm is deployed and is essential in the continuous mission scenario. In the following we describe why this is the case.

If the computing tasks that the swarm must execute are statefull applications like tracking [18] the continuous mission is only feasible if task state can be migrated from the members of the swarm that are running low on fuel to those that are replacing them. Thus task mobility is required for this type of mission to be feasible. In the single mission case task mobility is not strictly necessary for feasibility. Tasks can be loaded on each member of the swarm. The swarm will then remain aloft till the first plane in the swarm losses power. Then the whole swarm must return to base. It might seem possible therefore to plan so that each plane has exactly the fuel loaded for the tasks that is needs to perform if you know in advance the workload the swarm will encounter. However we don't know in advance the workload of the swarm in many practical situations. For example imagine that the task of the swarm is to perform surveillance. This applications consist of a continuous task of scanning the seas below UAV1 looking for an object using a low power visible CMOS camera. When the object is identified then a high power periodic task is invoked to gain an alternative image of the object using a IR sensor on UAV2. The relative power consumed depends on the frequency the IR sensor is used. Because we cannot predict how many objects will be detected on the mission we cannot predict the relative power consumption between the UAV1 and UAV2 due to the difference in the power required to operate the sensors. Thus in the absence of task mobility it could be expected that one UAV would run low on power sooner that the other. If we have task mobility then we can equalize the power between the UAVs.

The question now arises as to how we can arrange for this mobility to happen. We have decided to use the agent paradigm to express and control this mobility. It is generally accepted that an agent must posses at a minimum the properties of ***autonomy***, ***social interaction***, ***reactivity***, and ***proactiveness*** [19]. ***Mobile agents*** are a special class of agents that are able to migrate between host computer systems while executing [20]. Mobile agents are not able to function without the support of an ***agent environment*** that

executes on host systems and aids in the migration process. In the remainder of this section, the key properties of agents is examined in greater detail.

The autonomous operation of agents in dynamic systems are one of there most attractive features. An autonomously agent is entrusted to act and decide on courses of action without being specifically directed by the user [21]. This ability of agents is especially useful in dynamic environments where deterministic processes or agents would require constant instruction from the user. Milojicic defines the transfer of authority to act on a user behalf as the defining attribute of mobile agents when compared to other forms of mobile code and execution [22].

The agent paradigm implies a degree of interaction between agents and external entities. Social interactions are implemented by exchanging messages formatted in an agent communication language [23]. The messages can contain information or coordination of activities where agents are collaborating to achieve common goals. Through teambuilding, individual agents have the ability to increase there effectiveness by cooperative coordination in order to achieve common goals [24]. In agent environments with restricted resources, selective teambuilding and coordination can maximize the usage of resources.

4. Sharing FPGA computing and power resources across a swarm of UAVs

Sharing a single FPGA among many embedded tasks, allowing them to be loaded at any time, is a necessary first step to making these tasks mobile across a swarm of UAVs each of which is fitted with an FPGA[25]. In this section we explain how the operating system is extended to support this mobility. In the next section the autonomous agent based design of the distributed operating system and the fuzzy rule base that controls task migration are described. An agent based environment has been chosen for the swarm because it allows members of the swarm to be consider as disposable in a way that does not place the whole swarm in jeopardy. The behaviour of each agent in an autonomous agent based environments is usually governed by rules which are specific to each agent. We describe how we have adopted a fuzzy rule base for our agents.

A swarm of UAVs is a collection of a many different types of resources ranging from platforms, to sensors and effectors, to processing units. To best make use of these, they must be interconnected in such a way as to enable them to not only share the resource, but manage it responsible. This requires coordination in resource allocation which involves balancing the needs of applications with other resources such as power and bandwidth. Although there are many ways in which this can be implemented, the nature of a swarm makes any form of centralized control undesirable as it introduces a single point of failure in a system prone to unreliability.

Computing agents are a distributed computing paradigm that suits such environments. Agents are a subclass of computer programs that exhibit the properties of autonomy, social ability, reactivity and pro-activeness. The agents can be further categorized as mobile or static agents. A static agent may represent a resource such as a camera which is

fixed to a platform whereas mobile agents represent applications that may move there execution between platforms. Unlike many other distributed computing paradigms, mobile agents allow the transfer of state, not just execution, between nodes. This is done under the agents own control, which allows applications to customise migration rules which can further enhance the advantages of the distributed system by taking advantage of application specific knowledge. The behaviour of an agent is specified as a set of basic rules that govern its behaviour. A static sensor for example might have rules which specify under what conditions it should share data with an application. The agent may rank connected applications in order of mission priority and throttle the bandwidth of trivial applications in favour of mission critical tasks.

We now describe the infrastructure that supports the agent environment.
In order for these agents to be useful they must exist in a networked environment that supports their basic requirements, which are:

- Discovery of other agents
- Communication with other agents
- Providing information about other nodes
- Migration between nodes

Further requirements of the environment are:

- Transaction type migration – all or nothing
- Message routing and forwarding

If agents are to communicate and exchange information they must first be able to find the location of other agents of interest. To facilitate this, each node maintains a list of agents currently at its locale. Agents are defined on the network by their location and abilities. These are used at the time of creation to create a unique tuple which identifies that agent on the node. The tuple is defined as {*sequence number*, *home node*, *class*, *current node*, *ability list*}, where '*sequence number*' is a unique identification number with respect to the '*home node*', which is the node that the agent was created on. '*class*' is the type of agent, '*current node*' is the node that the agent currently executes on, and finally the '*ability list*' describes the agents abilities. Nodes periodically exchange or update peers so that a global snapshot of agents is available at each node. When an agent wishes to communicate with another, sends the message to the host on which it is executing along with the *sequence number/home node* key that identifies the host on the network. The host then passes the message onto the host where the recipient is executing, which is then collected by the target. Should a node receive a message for a recipient that no longer exists on the network it must update its peers and handle the undelivered message. If the recipient has simply ended its execution, the message can be deleted and an update of the current agents exchanged. If the recipient has migrated, the message must be forwarded to the agents new location and the agent table updated.

Mobile agents require the most support for the migration process. Performing a migration uses resources of bandwidth and computation (due to downtime). Because of this agents need as much information as possible available to make the best decisions possible. Information that a mobile agent may be interesting includes the CPU and memory usage

on the target host platform, its physical location, the availability of reconfigurable computing resources, power usage, bandwidth to various other nodes, and the availability of other local resources. All this information is made available on request to mobile agents using messages passed directly to the target node.

If a mobile agent wishes to migrate to another node, a sequence of transactions take place between itself and the target node to transfer its state and execution to the new location. First, each agent is written within a framework that requires developers to write methods to extract its state, and allow itself to resume a state. When an application is to migrate, its sends a request to the target host, which then invokes a new instance of that agents class in a sleep state (so it is not performing any processing). The new instance is given a temporary identifier which is returned to the migrating node. When this is received, the agent stops performing its task, captures its state, and sends it to the sleeping node which restores itself to this state. At this point, the both nodes are notified, and the original instance of the mobile agent end its execution. The new instance is then free resume its task in the new location. The thing to note here is that the developer of the mobile agent has not written code for migration, just recover and restore methods. The actual migration is performed upon a request to the agent environment.

While an application is performing its processing, the agent component is constantly examining the network for opportunities to increase its effectiveness through migration. The search for possibly advantageous migrations is implemented in a separate thread so as not to directly affect the application. The objective for the application developer is to define a set of conditions where a migration is desirable. Fuzzy logic has been used thus far as the basis of expressing the desired behaviour, although the framework allows the developer to use other means for expressing these conditions as rules.

An example of fuzzy rules, consider the case of applications searching for targets within a sub-region of the operating area of a swarm of UAV's. If there are many applications executing within this swarm, it may not be possible to uarentee that a particular uav can be move to a particular location. The applications rules must express its 'desire' to migrate to planes that are already in the region. A high level descriptions of a rule that will exhibit the behaviour is:

```
If (the visibility of sensors on this platform is LOW) AND
(the visibility on another platform is HIGH) then (desire
                    to migrate is HIGH)
```

This rule may be combined with others that compare the power and bandwidth availability, the types of sensors and utilization, as well as the cost of migrations and produce the desired behaviour.

5. Conclusion

Swarms of small UAVs can benefit from the use of embedded reconfigurable computers. By extending a operating system for reconfigurable computing, we have constructed a distributed operating system that allows mobile agent enabled applications to migrate

their execution between networked platforms based on fuzzy logic rules. This allows applications to not only migrate to increase performance or move closer to sources of data, but also allows power to be managed across a swarm to increase its overall mission time.

Acknowledgements

The authors would like to acknowledge the support of the Sir Ross and Sir Keith Smith Fund.

References

1. M. Dorigo, V. Maniezzo and A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics-Part B 26(1) ,1996
2. S. Shammai, European UAV market is expanding [Online, accessed 1/5/2004]. URL: http://www.frost.com, 2004
3. Use of Unmanned aircraft for fire detection [Online, accessed 1/5/2004]. URL: http://fire.feric.ca/36152002/WorkshopProgram.htm#2, 2004
4. S. R. Herwitz, L. F. Johnson, J. F. Arvesen, R. G. Higgins, J. G. Leung, & S. E. Dunagan, 'Precision Agriculture as a Commercial Application for Solar-Powered Unmanned Aerial Vehicles', 1st AIAA UAV Conf, Portsmouth VA, 2002
5. US Homeland Security ponders prospective UAV test centre [Online, accessed 9/5/2004]. URL: http://www.uavworld.com/_disc1/000000e9.htm, 2003
6. V. G. Ambrosia, S. S. Wegener, D. V. Sullivan, S. W. Buechel, J. A. Brass, S. E. Dunagan, R. G. Higgins, E. A. Hildum, & S. M. Schoenung, 'Demonstrating UAV-Acquired Real-Time Thermal Data over Fires', Photogrammetric Engineering and Remote Sensing, 2002
7. S. A. Barbulescu, 'Iterative decoding of turbo codes and other concatenated codes', University of South Australia, Adelaide, Australia, 1996
8. Compton, K. and Hauck, S., Reconfigurable computing: a survey of systems and software, ACM Computing surveys (CSUR), Vol. 34, pp. 171-210, 2002
9. Brown, S., Francis, R., Rose, J., and Vranesic, z., Fiel Programmable Gate Arrays., Boston, USA: Kluwer and Acad. Publishers, 1992.
10. Sanderson, "FPGA Computing Provides Superior Performance Density for UAV Applications," COTS Journal, 2003
11. Robinson, "UAV multi-mission payloads demand a flexible common processors," COTS journal, 2003.
12. Yamada, H., Tomenaga, T., Ichikawa, M., An Autonomous Flying Object Navigated by Real-Time Optical Flow and Visual Target Detection.
13. Scalera, J., Jones, C., Soni, M., Bucciero, M., Athanas, P., Abbot, K., Mishra, A., Reconfigurable Object detection in FLIR Image Sequences
14. Petronio, Bambha, Caswell, and Berleson, "An FPGA-Based Data Acquisition System for a 96 GHZ W-Band Radar."
15. Finn, A., Brown, K., Lendsay, T., Miniature UAV's & future electronic warfare, EW & Radar division DSTO
16. Ledger, D., Electronic warfare capabilities of Mini UAVs, Aerosonde (online: http://www.aerosonde.com/drawarticle/73#finn)

17. Donnelly, H., Swarming UAVs, [online accessed 5/1/06], url: http://www.military-aerospace-technology.com/article.cfm?DocID=686
18. Wong. S., Jasiunas, M., Kearney, D., Towards a Reconfigurable Tracking system, 15th International Conference on Field Programmable Logic and Applications, AUG 2005, IEEE
19. Wooldridge, Jennings, Intelligent Agents: Theory and Practice
20. Kotz, Gray, Mobile Agents and the Future of the Internet
21. Wieb, G., Revatsos, M., Nickles, M., Capturing Agent Autonomy in roles and XML
22. 2000, Milojicic, Douglis, Paindaveine, Wheeler, Zhou, Process Migration, ACM Computing Surveys, Vol 32, No 3, pp. 241-299
23. Genesereth, M.R. and Ketchpel, S.P. (1994). Software agents, Communications of the ACM 37(7); 48-53
24. Tambe, M. (1997), Implementing Agent teams in Dynamic Multi-agent Environments
25. Wigley, G, Kearney, D. The Development of an Operating System for Reconfigurable Computing. In Proc. IEEE Symp. FCCM'01, April 2001, IEEE Press.