Landscape of Intelligent Cores: An Autonomic Multi-agent Approach for Space Applications

BLESSON VARGHESE AND GERARD MCKEE
School of Systems Engineering
University of Reading
UNITED KINGDOM
{hx019035, g.t.mckee}@reading.ac.uk

Abstract: - Space applications are challenged by the reliability of parallel computing systems (FPGAs) employed in space crafts due to Single-Event Upsets. The work reported in this paper aims to achieve self-managing systems which are reliable for space applications by applying autonomic computing constructs to parallel computing systems. A novel technique, ‘Swarm-Array Computing’ inspired by swarm robotics, and built on the foundations of autonomic and parallel computing is proposed as a path to achieve autonomy. The constitution of swarm-array computing comprising four constituents, namely the computing system, the problem / task, the swarm and the landscape is considered. Three approaches that bind these constituents together are proposed. The feasibility of one among the three proposed approaches is validated on the SeSAm multi-agent simulator and landscapes representing the computing space and problem are generated using the MATLAB.

Key-Words: - swarm-array computing, autonomic computing, parallel computing, multi-agent systems.

1 Introduction
Managing networked computing systems has become a very cumbersome task with their increasing complexity. From a business point of view the need for self-managing systems has risen due to the current crisis of complexity. To address the need for self-managing systems, autonomic computing was proposed for better management of increasingly complex computing systems and reduce the total cost of ownership of systems today [1] [2]. Autonomic computing solutions aims to reallocate management responsibilities from administrators to the computing system itself based on high-level policies [3] [4].

From the research perspective, autonomic computing is defined by self-management [5], and is characterized by four objectives and four attributes. The objectives and attributes that contribute to self-management are not independent functions. The objectives considered are (a) Self-configuration, (b) Self-healing, (c) Self-optimizing and (d) Self-protecting [6]-[8]. The attributes considered are (a) Self-awareness, (b) Self-situated, (c) Self-monitoring and (d) Self-adjusting [6]-[8].

The benefits of autonomy in computing systems are also apparent for parallel computing systems (e.g. multi-core processors, clusters, grids, field programmable gate arrays (FPGA), general purpose graphics processing units (GPGPU), application-specific integrated circuit (ASIC) and vector processors), namely reducing cost of ownership and reallocating management responsibilities to the system itself. The parallel computing paradigm employs the concurrent utilization of multiple processing elements to solve a problem [9]. Wide ranges of problems have found quicker solutions by utilizing parallel computational power, since the processor-memory bottleneck is addressed. For example, parallel computing is useful for developing problem solvers that engage in computationally intensive operations and voluminous data processing requiring high processing rate.

One such area that can be influenced by the merging autonomic computing and parallel computing are in space applications. Space crafts employ FPGAs, a special purpose parallel computing system that are subject to malfunctioning or failures of hardware due to ‘Single Event Upsets’ (SEUs), caused by radiation on moving out of the protection of the atmosphere [10]-[12]. One solution to overcome this problem is to employ reconfigurable FPGAs. However, there are many overheads in using such technology and hardware reconfiguration is challenging in space environments. In other words, replacement or servicing of hardware is an extremely limited option in space environments. On the other hand software changes can be accomplished.

However, the question which path should be adopted to achieve autonomy in parallel computing systems for space applications persists. In this paper, we propose a novel synergy, with the aim of applying autonomic computing constructs to parallel computing systems and the advantage that software changes can be accomplished in FPGAs employed for space applications. Parallel computing and swarm robotics are closely linked together to offer a new computing paradigm, ‘Swarm-Array computing’ that can harness autonomic
computing for parallel computing systems useful for building reliable space systems.

Swarm-array computing is biomimetically inspired by the theory of autonomous agents in natural swarms, abstracted and implemented in swarm robotics. In this paper, FPGA cores are considered as the parallel computing system and an approach to implement swarm-array computing is proposed. The cores of the FPGA are considered to be autonomous agents with a high degree of self-managing capabilities. Subtasks to be executed reside upon a landscape of intelligent cores. The feasibility of the proposed approach is validated using SeSAm [13] [14] simulator and two dimensional landscapes are generated on the MATLAB toolkit.

The remainder of the paper is organized as follows. Section 2 introduces and describes the constituents of swarm-array computing. Section 3 proposes three approaches in swarm-array computing to achieve autonomy in parallel computing systems. Simulation studies for one among the three proposed approaches are presented in Section 4. The paper is concluded in Section 5.

2 Swarm-Array Computing

As discussed in the section above, parallel computing also benefits from the application of the autonomic computing paradigm. However, which path should be adopted to achieve this autonomy in parallel computing systems? In this context, swarm-array computing, a swarm robotics inspired approach is proposed as a path to achieve autonomy. The development of the swarm-array computing approach from the foundations of parallel and autonomic computing is shown in Figure 1. The constitution of the swarm-array computing approach can be separated into four different constituents, namely the computing system, the problem / task, the swarms and the landscape as shown in Figure 1. Each constituent is considered in the following sub sections.

2.1 The Computing System

FPGAs are a technology under investigation in which the cores of the computing system are not geographically distributed. The cores in close proximity can be configured to achieve a regular grid or a two dimensional lattice structure. Another reason of choice to look into FPGAs is its flexibility for implementing reconfigurable computing.

The cores of a parallel computing system can be considered as a set of autonomous agents, interacting with each other and coordinating the execution of tasks. In this case, a processing core is similar to an organism whose function is to execute a task. The focus towards autonomy is laid on the parallel computing cores abstracted onto intelligent cores. The set of intelligent cores hence transform the parallel computing system into an intelligent swarm. The intelligent cores hence form a swarm-array. A parallel task to be executed resides within a queue and is scheduled onto different cores by the scheduler. The swarm of cores collectively executes the task.

The intelligent cores described above are an abstract view of the hardware cores. But then the question on what intelligence can be achieved on the set of cores needs to be addressed. Intelligence of the cores is achieved in two different ways. Firstly, by monitoring local neighbours. Independent of what the cores are executing, the cores can monitor each other. Each core can ask the question of ‘are you alive’ to its neighbours and gain information. Secondly, by adjusting to core failures. If a core fails, the process which was executed on the core needs to be shifted to another core where resources previously accessed can be utilized. Once a process has been shifted, all data dependencies need to be re-established.

To shift a process from one core to another, there is a requirement of storing data associated and state of the executing process, referred to as checkpointing. This can be achieved by a process monitoring each core or by swarm carrier agents that can store the state of an executing process. The checkpointing method suggested is decentralized and distributed across the computing system. Hence, though a core failure may occur, a process can seamlessly be transferred onto another core. In effect, awareness and optimizing features of the self-ware properties are achieved.
2.2 The Problem/Task
The task to be executed on the parallel computing cores can be considered as a swarm of autonomous agents. To achieve this, a single task needs to be decomposed and the sub tasks need to be mapped onto swarm agents. The agent and the sub-problems are independent of each other or in other words, the swarm agents are only carriers of the sub-tasks or are a wrapper around the sub-tasks.

The swarm displaces itself to a goal across the parallel computing cores or the environment. The goal would be to find an area accessible to resources required for executing the sub tasks within the environment. In this case, a swarm agent is similar to an organism whose function is to execute on a core. The focus towards autonomy is laid on the executing task abstracted onto intelligent agents. The intelligent agents hence form a swarm-array.

The intelligent agents described above are an abstract view of the sub-tasks to be executed on the hardware cores. Intelligence of the carrier agents is demonstrated in two ways. Firstly, the capabilities of the carrier swarm agents to identify and move to the right location to execute a task. In this case, the agents need to be aware of their environments and which cores can execute the task. Secondly, the prediction of some type of core failures can be inferred by consistent monitoring of power consumption and heat dissipation of the cores. If the core on which a sub-task being executed is predicted to fail, then the carrier agents shift from one core to another gracefully without causing an interruption to execution, hence making the system more fault-tolerant and reliable. An agent can shift from one core to another by being aware of which cores in the nearest vicinity of the currently executing core are available.

2.3 The Swarms
A combination of the intelligent cores and intelligent swarm agents leads to intelligent swarms. The intelligent cores and intelligent agents form a multi-dimensional swarm-array. The arena in which the swarms interact with each other is considered in the next sub-section.

2.4 The Landscape
The landscape is a representation of the arena of cores and agents that are interacting with each other in the parallel computing system. At any given instance, the landscape can define the current state of the computing system. Computing cores that have failed and are predicted to fail are holes in the environment and obstacles to be avoided by the swarms.

A landscape is modelled from three different perspectives which is the basis for the swarm-array computing approaches discussed in the next section. Firstly, a landscape comprising dynamic cores (are autonomous) and static agents (are not autonomous) can be considered. In this case, the landscape is affected by the intelligent cores. Secondly, a landscape comprising of static cores and dynamic agents can be considered. In this case, the landscape is affected by the mobility of the intelligent agents. Thirdly, a landscape comprising of dynamic cores and dynamic agents can be considered. In this case, the landscape is affected by the intelligent cores and mobility of the carrier agents.

3 Computing Approaches
At this point one might cogitate on how the swarm-array computing constituents can be extended towards applying autonomic computing constructs to parallel computing systems. There are at least three approaches based on intelligent cores, intelligent agents, and intelligent swarms. However, in this paper only the first approach based on intelligent cores are considered.

In the first approach, only the intelligent cores are considered to be autonomous swarm agents and form the landscape. A parallel task to be executed resides within a queue and is scheduled onto the cores by a scheduler. The intelligent cores interact with each other to transfer tasks from one core to another at the event of a hardware failure. Figure 2 is a diagrammatic illustration of the first approach. The simulation methodology of the first proposed approach is described in Section 4.
In the second approach, only the intelligent swarm agents are considered to be autonomous and form the landscape. A parallel task to be executed resides in a queue, which is mapped onto carrier swarm agents by the scheduler. The carrier swarm displace through the cores to find an appropriate area to cluster and execute the task. The intelligent agents interact with each other as considered in Section 2.2 to achieve mobility and successful execution of a task.

In the third approach, both the intelligent cores and intelligent agents are considered to form the landscape. Hence, the approach is called a combinative approach. A parallel task to be executed resides in a queue, which is mapped onto swarm agents by a scheduler. The swarm agents can shift through the landscape utilizing their own intelligence, or the swarm of cores could transfer tasks from core to core in the landscape. The landscape is affected by the mobility of intelligent agents on the cores and intelligent cores collectively executing a task by accommodating the intelligent agent.

The second and third approaches not only consider the computational resource as a swarm of resources, but also the task to be executed as a swarm of sub-tasks. Hence, the approach considers complex interactions between swarms of sub-tasks and swarms of resources. The interactions between swarm agents bring about the notion of intelligent agents or swarm agents carrying the sub-tasks and intelligent cores or swarm of cores executing the sub-task. In other words, the approaches can be viewed as a computational approach emerging from the interaction of multi-dimensional arrays of swarm agents. However, in this paper only the first approach is considered.

4 Experimental Studies
Simulation studies were pursued to validate and visualize the proposed approach in Swarm-Array Computing. Various simulation platforms were considered, namely network simulators, which could predict behaviours of data packets in networks, and multi-agent simulators, that could model agents and their behaviours in an environment. Since FPGA cores are considered in this paper, network simulators were not an appropriate choice. The first approach proposed in this paper considers executing cores as agents; hence a multi-agent simulator is employed.

The feasibility of the proposed swarm-array computing approach was validated on the SeSAm (Shell for Simulated Agent Systems) simulator. The SeSAm simulator environment supports the modelling of complex agent-based models and their visualization [13] [14].

The environment has provisions for modelling agents, the world and simulation runs. Agents are characterized by a reasoning engine and a set of state variables. The reasoning engine defines the behaviour of the agent, and is implemented in the form of an activity diagram, similar to a UML-based activity diagram. The state variables of the agent specify the state of an agent. Rules that define activities and conditions can be visually modelled without the knowledge of a programming language. The building block of such rules are primitives that are pre-defined. Complex constructs such as functions and data-types can be user-defined.

The world provides knowledge about the surroundings the agent is thriving. A world is also characterized by variables and behaviours. The modelling of the world defines the external influences that can affect the agent. Hence, variables associated with a world class can be used as parameters that define global behaviours. This in turn leads to the control over agent generation, distribution and destruction.

Simulation runs are defined by simulation elements that contribute to the agent-based model being constructed. The simulation elements include situations, analysis lists, simulations and experiments. Situations are configurations of the world with pre-positioned agents to start a simulation run. Analysis lists define means to study agents and their behaviour with respect to time. Simulations are combinations of a situation, a set of analysis items and a simulation run; or in other words a complete definition of a single simulation run. Experiments are used when a combination of single simulation runs are required to be defined.

As considered in Section 2, the swarm-array computing approach needs to consider the computing platform, the problem/task and the landscapes. The parallel computing platform considered in this paper is FPGAs. The cores of the FPGA are modelled as agents in SeSAm, in accordance with the swarm-array computing approach reported in this paper. The intelligent cores are an abstraction of the hardware cores arranged in a 5 X 5 regular grid structure. The model assumes serial bus connectivity between individual cores. Hence, a task scheduled on a core can be transferred onto any other core in the regular grid abstraction.

The breakdown of any given task to subtasks is not considered within the problem domain of swarm-array computing. The simulation is initialized with sub-tasks scheduled to a few cores in the grid. Each core maintains a record of the subtasks it is executing and can monitor cores in the regular grid to which the subtasks can be assigned in the event of a predicted failure. The behaviour of the individual cores varies randomly in the simulation. For example, the temperature of the FPGA core changes during simulation. If the temperature of a core exceeds a predefined threshold, the subtask executed on the core...
requires reassignment to another available core that is not predicted to fail. During the event of a transfer or reassignment, the record of the subtask maintained by the core is also transferred to the new core. If more than one sub-task is executed on a core predicted to fail, each sub-task may be reassigned to different cores.

Figure 3 is a series of screenshots of a random simulation run developed on SeSAm for eight consecutive time steps from initialization. The figure shows the executing cores as rectangular blocks in pale yellow. When a core is predicted to fail, i.e., temperature increases beyond a threshold, the core is displayed in red. The subtasks are shown as blue filled circles that occupy a random position on a core. As discussed above, when a core is predicted to fail, the subtask executing on the core predicted to fail gets seamlessly transferred to a core capable of processing at that instant.

A log was recorded of the temperature of individual cores of the FPGA during a simulation. A 3D plot of the multi-core landscape during an arbitrary time step, where the z-axis represents the temperature was generated on the MATLAB toolkit. Figure 4 (top) shows the 3D representation of the landscape. Since the landscape is uneven, a contour plot was also generated for better visualization. Figure 4 (bottom) shows the contour plot of the landscape. Since figure 6 is the representation of the landscape of a 5 X 5 regular grid, the landscape has less resolution. Hence higher resolution contour plots were generated. Figure 5 (top) shows the contour plot for a 25X 25 regular grid during an arbitrary time step of the simulation run and figure 5 (bottom) shows the contour plot for a 100X100 regular grid during an arbitrary time step of the simulation run.

The generated landscape provides a pictorial view of the computing space and the problem. According to the colour bar provided in the legend of the plots, a light shade on the contour plot represents cores of higher temperature while the dark shaded regions represent safe cores of the FPGA. From the 3D plot it is understood that the peaks of the landscape are cores predicted to fail while the valleys and lower elevations on the landscape represent safe cores of the FPGA. In the approach considered in this paper, the sub task always resides on the safe cores of the FPGA. In other words, within the landscape sub tasks thrive in valleys and lower elevations.

From a different perspective, the peaks in the landscape can be considered as obstacles for the executing sub tasks. Hence, the sub tasks avoid the peaks by being autonomously and seamlessly transferred to safer regions within the landscape. The notion of obstacles in a landscape is also seen within the domain of swarm robotics, where a swarm of mobile autonomous agents on a landscape avoids obstacles to reach a goal. The landscape in the approach presented in this paper is dynamic (subject to change every time step) and hence makes the computing space and problem much more complex than a static landscape usually considered in swarm robotics. Eventually, the problem of applying autonomic computing constructs to parallel computing platforms is funnelled to a classic, yet a complex swarm robotics problem of obstacle avoidance.

The simulation studies are in accordance with the expectation and hence are a preliminary confirmation of the feasibility of the proposed approach in swarm-array computing. Though some assumptions and minor
approximations are made, the approach is an opening for applying autonomic constructs to parallel computing platforms.

5 Conclusion
In this paper, swarm-array computing, a novel technique to apply autonomic computing constructs in parallel computing is proposed. The foundation and inspiration of the approach is introduced. The constitution of swarm-array computing and how the constituents can be utilized to achieve autonomic properties are described. Three approaches that bind the constitution of swarm-array computing are proposed. Space applications employing FPGAs are identified as an area that can be influenced by swarm-array computing. The feasibility of one among the three proposed approaches is presented in this paper.

Future work will include identifying the major challenges that need to be addressed in swarm-array computing. Emphasis will be laid on how the swarm-computing approaches can be steered into real time implementation. Further effort will be made to explore the concepts of abstraction, decentralized checkpointing, intelligent cores and intelligent agents.

References:


