EFTOS: A Software Framework for More Dependable Embedded HPC Applications.

G. Deconinck[1], V. De Florio[1], R. Lauwereins[1], T. Varvarigou[2]

[1] K.U.Leuven, Dept. Elektrotechniek, Kard. Mercierlaan 94, B-3001 Leuven, Belgium
[2] N.T.U.A., Dept. Elect. Comp. Eng., I. Putechniou 9, GR-15733 Zographou, Greece

Abstract. Within the ESPRIT project EFTOS (Embedded Fault-Tolerant Supercomputing), a framework is developed to integrate fault tolerance flexibly and easily into distributed embedded HPC applications. This framework consists of a variety of reusable fault tolerance modules acting at different levels. The cost and performance overhead of generic Operating System and Hardware level fault tolerance mechanisms are avoided, while at the same time the burden of *ad hoc* fault tolerance programming is removed from the application developers. Integration of this functionality in real embedded applications validates this approach, and provides promising results.

Keywords. fault tolerance, embedded system, HPC

1 Introduction

Current industrial embedded applications require high computing performance that may be only provided by parallel computing systems. However they are prone to data-induced software faults and operate in industrial environments. Hence, these applications require fault tolerance to cope with the problems that will occur. Application developers tend to deal with these needs of availability and reliability of their applications by developing their own fault tolerance code as part of their application. This increases the cost of application development, maintenance and upgrade considerably.

Within the ESPRIT project EFTOS (**E**mbedded **F**ault-**T**olerant **S**upercomputing), several of these embedded applications have been investigated which require both high performance and fault tolerance [2]. We saw that different applications have many fault tolerance requirements in common, and incorporated similar ad hoc solutions. The goal of EFTOS is to streamline these requirements into a reusable fault tolerance framework that can be flexibly and easily integrated into the target applications. As such, application developers can enjoy the benefits of a variety of fault tolerance functions from which they can pick and choose what is necessary for their environment. The cost and performance overhead of generic Operating System and Hardware level fault tolerance mechanisms are avoided, while removing the burden of fault tolerance programming from the application developers at the same time.

Although the project aims at general embedded applications running on parallel systems, two representative applications have been analysed in more detail:

an image processing module in an automatic mail processing system [3] and a remote controller in a high voltage substation.

The distributed target system on which this framework is being implemented is a Parsytec *CC* System [1] which combines powerful processing nodes, I/O modules and routers into a parallel system (MIMD architecture). The parallel operating system, *EPX* (*Embedded Parallel extensions to uniX*), provides the functionality to operate the parallel environment via a message-passing API.

The EFTOS project is coordinated by Parsytec, who specialises in embedded HPC systems. AEG and ENEL represent the market of embedded applications. The knowledge on fault tolerance is supplied by K.U.Leuven, N.T.U.A. and D.L.R. The project started in April 1996 and runs for 2 years.

Several research projects investigated fault tolerance for embedded applications on distributed systems.

- The Delta-4 project (Esprit 818/2252) proposes a *generic architecture for dependable distributed computing*, based on multiplication of modules. Two variants of the architecture focus on portability and on performance of the approach. Both however rely on fail-silent hardware and on an atomic multicast protocol [6].
- The projects PDCS/PDCS2 (Esprit 3092/6362) consider *predictably dependable computing systems* aiming at making the design, development and production of dependable computing systems more predictable and cost effective (fault prevention, tolerance, removal and forecasting) [7]. Focus is on safety and security issues, as well as on quantitative assessments of dependability of the system. The MARS system (partly developed in this project) addresses embedded real time systems. It meets hard real time deadlines and tolerates interconnection and node faults, but requires dedicated hardware and a specific operating system (suited for time-triggered applications) [5].
- The FTMPS project (Esprit 6731) applies software fault tolerance solutions to number- crunching applications in massively parallel systems [4]. However, real-time aspects are not taken into account.

In fact, none of these projects considered standard hardware and operating systems for soft real time applications, to provide a powerful fault tolerance framework. It is clear however that there is an emerging need to provide such a framework to allow the application developer to incorporate *adaptable fault tolerance solutions into the embedded HPC applications*, based on standard hardware and software products. The availability of such a framework could decrease the current tendency of industrial application developers who frequently develop (similar) *ad hoc* solutions to improve the fault tolerance capabilities of their embedded applications.

2 Fault tolerance requirements

The errors that occur in systems in the scope of the EFTOS project can be summarised in broad terms in the following categories:

- software errors triggered by untested or unforeseen input values;
- hardware faults caused by electromagnetic interference;
- the propagation of errors through communication channels from one part of the system to the others;
- memory corruption by errors propagating from one process to another, as processes are running concurrently in the same memory space;
- the loss of a number of subsequent inputs due to a faulty input item;
- meeting time constraints and deadlines.

	detection	isolation and reconfiguration	recovery	associated mechanisms
thread level	watchdogs	termination of faulty thread	re-initialisation of a thread	exception handling
		re-mapping of threads	restoring saved state in application	checkpointing, atomic actions
memory level	memory access checking	freeing memory resources	restoring memory	stable memory
node level	I'm alive	disconnection of a faulty node	re-initialisation or fast rebooting of a node	
		reconfiguration of process graph / communication topology	integration of a recovered node into the system	
system level	monitoring system parameters		fast reboot fast restart	
message level	communication time-out	discard message	retransmission	reliable communication
	checking of message ordering		re-ordering	
link level	checking of communication channel/partner status before communication	termination of a faulty communication channel	re-initialisation of a communication channel	

Table 1. Fault tolerance requirements.

Based on this categorisation we have developed fault tolerance modules that deal with the different types of errors. Application developers were asked to prioritise these modules according to the relevance to the needs of their application, their impact on fault tolerance and the feasibility to integrate them into the target application. This lead to the following orthogonal classification of requirements, according to the *location* where fault tolerance is required (processing and networking modules), according to the *steps* to achieve fault tolerance (detection, isolation and recovery mechanisms). Table 1 shows the list of the set of fault tolerance functionality that our library contains and the corresponding classification. We also added associated mechanisms.

3 EFTOS framework with reusable software solutions

Based on these requirements, the EFTOS project is providing a framework for fault tolerance, elaborated on a Parsytec CC system, which acts at different levels. (This results in the architecture of figure 1.)

- At the lowest level, it contains *elements* for error detection (D tools) and for error recovery (R tools). These are parametrisable functions that can be

used in stand-alone functionality, or in combination with the next levels to apply fault tolerance to processing or communication modules.
- At the middle level, the *DIR net* (detection, isolation and recovery network) combines these elements to tolerate faults. It ensures consistent decisions in the distributed system and serves as a backbone to pass information among the fault tolerance elements.
- At the highest level, these elements can be combined into mechanisms, such as to provide fault-tolerant communication or voting on results.

The **D tools** are meant to detect errors. They are dynamically started by the user during the execution of the application. As such, the programmer can supply them with the correct parameters. When an error is detected, the D tool passes the necessary information via a standardised interface to the DIR net. This includes the type of error that occurred, the location where the error is detected and the D tool that detected it. Examples of D tools that are being integrated include:

- watchdog timers for communication and computation, detecting if a message is delivered in time or if a task produces its results before a certain deadline;
- assertions that check invariant relationships in values of variables;
- support to detect wrong memory accesses, e.g. to unused or non-existing memory or write access to read-only areas;
- monitoring of system and environmental parameters to detect deviations from the normal behaviour;
- trap handlers to catch signals;
- application-specific D tools, etc.

The **R tools** provide recovery mechanisms. They are started by the DIR net and have to bring the application back into a consistent state. Possibly the system will have less functionality than before (graceful degradation). They include the following:

- restarting a single node or a set of nodes;
- disabling communication channels temporarily or permanently; thereafter they have to be reset, reinitialised, and brought back into a consistent state. New communication channels must be established if a communication channel was completely removed or if the reconfiguration steps require another application topology;

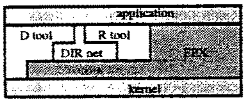

Fig. 1. EFTOS architecture. The application makes use of the message-passing environment EPX, running on top of the operating system kernel. The elements for detection (D tools) and recovery (R tools) interact with the application and are interconnected via the DIR net.

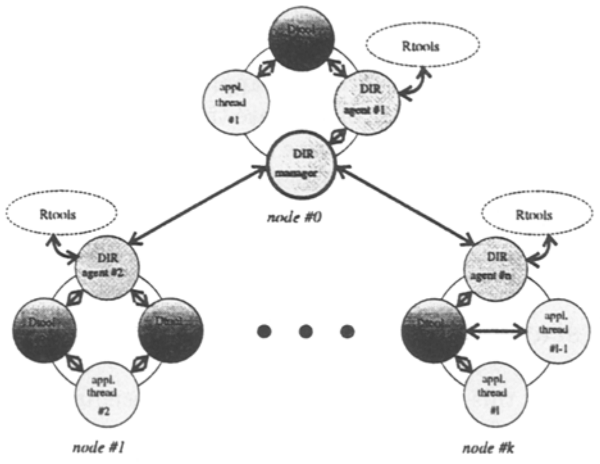

Fig. 2. Architecture of the DIR net.

- (virtually) disconnecting unusable threads or nodes from the rest of the application; releasing memory and other resources, that were assigned to threads or links that failed or had to be removed;
- validate output before releasing it, or selecting the correct answer from a set to mask faults;
- application-specific or user-specified recovery actions, etc.

The detection-isolation-recovery net, in short **DIR net**, is the backbone of the fault tolerance framework. It provides an interface to which all D tools and R tools should conform, while leaving enough room for flexibility within the tools. The DIR net allows distributed actions to take place. Therefore, it consists of a hierarchical network with a central manager and several distributed agents. This DIR net structure is shown in figure 2.

- The main module in the DIR net is the manager, which keeps a global view of the system. This view includes the status of each node used in the partition, the type and location of the D tools, the type and location of errors that occurred, and the status of R tools that are being executed. The manager also has the possibility to connect to an operator module, which forms a two-way interface between the operator and the DIR net to perform manual recovery actions.
- The DIR manager is assisted by multiple DIR agents on the different nodes, that do the field work. These agents start up and initialise D tools. They are warned when an error occurs, and forward this information to the manager. These agents take local recovery actions, or for coordinated actions, they start R tools upon request of the DIR net manager. Note that different agents are not interconnected, but can only communicate via the manager.

A hierarchical/centralised structure of the DIR net has been implemented so far. We are also evaluating a distributed (peer to peer) architecture for the DIR net to avoid the single point failure problems of the centralised version. However, performance reasons make a distributed version of DIR net quite unattractive, especially in the area of high performance applications that we are considering. Besides, the DIR net is a lean piece of software that has little probability of failure compared to the data-dependent software faults affecting the applications, which makes failures introduced by the DIR net quite unlikely. Hence, we are leaning towards building support mechanisms for the hierarchical DIR net structure (backups, duplications, message checking, heartbeat mechanisms, etc.).

4 Current status and outlook

A first prototype of this EFTOS framework has been implemented. It comprises basic elements of each of the fault tolerance steps. In a test application, this resulted in successful experiences when software fault injection was applied. While the implementation work continues on different levels, the developed basic elements, structures, and techniques will be tested, integrated into the target applications. These industrial application developers will validate the framework and provide feedback to the developers.

As such, the developed framework will become a flexible, and standardised set of solutions to improve the dependability of a large set of embedded applications at a low cost.

Acknowledgements. This project has partly been supported by ESPRIT project 21012 EFTOS, COF/96/11 and an FWO Krediet aan Navorsers. Geert Deconinck has a grant from IWT. Rudy Lauwereins is research associate of the FWO - Flanders.

5 References

[1] Anon., "Parsytec CC Series", Parsytec GmbH, Aachen (D), Jul. 1995.

[2] Anon., "EFTOS: Embedded Fault-Tolerant Supercomputing", Technical Annex ESPRIT- project 21012, Feb. 1996.

[3] G. Deconinck et al., "Fault Tolerance Requirements in Postal Automation: a Case Study", 4th *IFAC Alg. and Arch. for Real-Time Control*, Vilamoura (P), Apr. 1997.

[4] G. Deconinck et al., "Fault Tolerance in Massively Parallel Systems", *Transputer Communications*, Vol. 2(4), Dec. 1994, pp. 241–257.

[5] H. Kopetz et al., "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", *IEEE Micro*, Feb. 1989, pp. 25–40.

[6] D. Powell (Ed.), "Delta-4: A Generic Architecture for Dependable Distributed Computing", *Springer-Verlag*, Berlin Heidelberg New York, 1991.

[7] B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood (Eds.), "ESPRIT Basic Research Series: Predictably Dependable Computing Systems" *Springer-Verlag*, Berlin Heidelberg New York, 1995.