Backward Error Recovery in the APEmille Parallel Computer

Piero Maestrini
Istituto di Elaborazione della Informazione
Consiglio Nazionale delle Ricerche
maestrini@iei.pi.cnr.it

Tamás Bartha
Computer and Automation Research Institute
Hungarian Academy of Sciences
bartha@sztaki.hu

Introduction

This paper describes the fault tolerance concept (and in particular the error recovery scheme) used in the APEmille computer. The APEmille development project is a physicists-driven effort to produce a versatile parallel processor optimized for heavy duty (e.g. Lattice Gauge Theory) numerical applications.
From the user's view APEmille is a parallel computer consisting of processing elements optimized for floating-point arithmetic. The processors are arranged in a three-dimensional mesh topology with wrapped-around toroidal connections. Each processor executes the same instruction stream but has a private memory bank, so different nodes can operate on arbitrary data. In other words, APEmille has a Single Instruction Multiple Data (SIMD) architecture [1].
On the hardware level the architecture is more complicated. It has three main components: in addition to the already mentioned application processors (also called *Janes*), there are central processing units called *Tarzan* performing flow-control and coordinating tasks, and communication controllers called *Cheetah* functioning as interfaces between Janes, Tarzan, and the rest of the system. These devices are implemented as custom designed integrated circuits.
The APEmille architecture is hierarchically structured. The smallest functional unit is a *Processing Board* (PB) or *cluster*, containing eight Jane processors, one Tarzan control processor and one Cheetah commuter. The eight Jane processors are logically placed in the vertices of a cube. Tarzan supplies each Jane with a uniform instruction flow and global addresses. Data exchange and control information is delivered by the Cheetah commuter. A synchronous network provides the interconnection among PBs.
The next hierarchy level is called *APE Unit*. It is built up of four PBs connected to a stand-alone computer called *Local Host* (LH). The LH supervises the attached PBs and provides local disk storage. It can access the memory and registers of the Tarzan and Jane processors over a dedicated PCI bus. On the other hand, Tarzan units can trigger interrupts in LH to signify exceptions, service requests, or local conditions. The four boards and their host are connected to a custom backplane.
A configuration of four APE Units housed in a standard rack is called a *crate* (see Figure 1). The hosts of a crate are cooperating over a general-purpose, high performance Control Network. Also attached is a special computer acting as the *Global Host* (GH) of the entire APEmille machine. The GH manages the global disk storage and collects global signals, like halt requests, exceptions, if conditions, etc. Further crates can be added until the system reaches its maximum configuration of 2048 (24×8×8) processing elements.

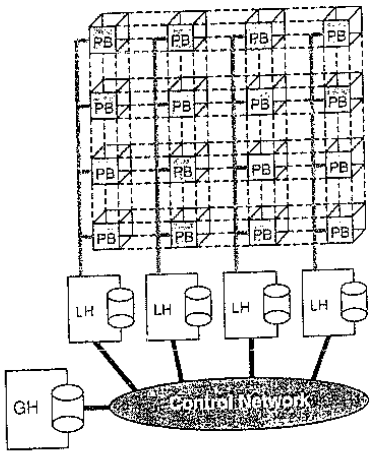

Figure 1. The structure of an APEmille crate

Motivation

APEmille is the third generation of the APE family of supercomputers (the previous machines were called *APE* and *APE100*), but the first one to include built-in support for fault tolerance. The reasons that led to the consideration of reliability issues from the early stages of system design come in a large part from the experiences with the previous generations. The APE supercomputers are number-crunching engines aimed at solving complex scientific problems. The typical duration of such a computation ranges from several days to several weeks. Due to the large number of components built into an APEmille configuration, the expected MTTF may fall *into the same range* without fault tolerance. Thus, a long-running computation can be invalidated by a system failure, and there is no guarantee that the results of a successfully terminated program are not corrupted by an undetected error.
To improve on this situation the designers of APEmille decided to incorporate fault tolerance into the system. They chose the *error removal* based ap-

proach, composed of the following main steps: *error detection*, system-level *fault diagnosis*, *system repair* (or *reconfiguration*), and *backward error recovery*. APEmille includes several additions to the SIMD paradigm, like local addressing and flexible message routing that make reconfiguration theoretically possible, although its implementation is not planned yet.

Recovery in APEmille

Before presenting the error recovery procedure itself, we take a look at the decisions behind it. These decisions were influenced by three main factors: system architecture, applied testing methods, and component failure semantics. The architecture has already been introduced, Table 1 gives an overview of the testing and failure characteristics of the system components:

	Component	Testing method	Failure semantic	Not detected	Recovery method
Run mode	Jane processor	Comparison	Value	Common mode	Checkpointing
	Tarzan processor	Comparison	Value	Common mode	Checkpointing
	Cheetah commuter	Comparison	Omission/Value	Common mode	Checkpointing
	Memory	EDAC	Omission/Value	Multiple ($n > 2$)	Error correction
	Network (inter-PB)	EDAC	Performance/Value	Multiple ($n > 2$)	Retransmission
System mode	Host processor	Watchdog timer	Crash (fail-stop)	Value	Message logging
	Host memory	Parity	Omission/Value	Multiple ($n > 1$)	Retry
	Host disk	EDAC	Omission/Value	Multiple	Duplication
	Control Network	Multi-checksum	Performance/Value	Multiple	Retransmission

Table 1. Failure model of APEmille components

The most notable peculiarity of APEmille is its "two-sided" nature. The hardware is a combination of a SIMD *processing* part formed of PBs, and a loosely-coupled *supervisor* part consisting of host computers. Furthermore, system operation can also be divided into two distinct stages. In *system mode* only the supervisor part functions (fulfills service requests or performs diagnosis), the processing part is frozen. In *run mode* the processing part is active, and while it is running the application program the supervisor part waits for local/global signals.

The processing elements of a PB are tested by comparators built-in the Cheetah commuter. The memory and network devices are protected by error detecting and correcting codes (EDAC). A detected error raises an exception in the LH, which switches APEmille to system mode and starts a special diagnostic session to locate the occurred faults [2].

The Global and Local Hosts are PC compatible computers complying the Compact PCI standard. They run a custom version of the Linux operating system. The hosts include only the usual PC-specific error detection techniques: memory is tested using parity, disk and network devices are guarded by EDAC and checksums. Clean shutdown is ensured by a (software) watchdog timer. Furthermore, the hosts employ a distributed system-level diagnosis procedure.

We propose different recovery mechanisms for the processing and supervisor parts [3]. The processing part is best suited for *checkpointing*. For this purpose, APEmille is periodically switched to system mode, every LH performs diagnosis on the corresponding four PBs, and if the whole APE Unit is fault-free it successively saves the local states of each processing element. Checkpoints are stored on the local disk of the LH. In order to ensure data integrity in the presence of host failures, checkpoint files are replicated on multiple hosts. (Replication does not decrease performance, as it may take place in the background while APEmille is in run mode.) Checkpoint consistency is not an issue due to the "all or nothing" property of the communication: simple point-to-point transfer is performed by Cheetah with high speed, whereas flexible routing takes place in system mode. The only concern is checkpoint latency: a large amount of data must be transferred and saved, therefore checkpoints cannot be taken arbitrarily oft.

Beyond supervision the main purpose of host computers is serving requests, handling exceptions, and executing RPC calls of the OS. These are a relatively few, well-defined operations, initiated by messages and similar events. Reliably logging the occurred events at the process interfaces ensures the possibility of restoring the complete local state of a host easier, than taking a checkpoint of each process and enforcing the consistency of the resulting checkpoint set. Consequently, for the supervising part *message logging* is proposed. The logging and replication techniques are determined by the reliability of the host computers. We assume that multiple simultaneous host failures have a small probability. If this holds, *sender-based message logging* and *available copy replication* are the most efficient methods. Otherwise, a *family-based logging* protocol is suggested.

References

[1] F. Aglietti et al., "Self-Diagnosis of APEmille," *Proc. EDCC-2 Workshop on Dependable Computing*, Gliwice Poland, May 1996.

[2] S. Chessa, B. Sallay, P. Maestrini, "Diagnostic Model and Diagnosis Algorithm of a SIMD Computer", To appear in *Proc. EDCC-3 Conference on Dependable Computing*, Prague, 1999.

[3] T. Bartha, "A Proposal for the Recovery Subsystem of the APEmille Parallel Computer," *Tech. Report*, IEI-CNR, under preparation.