RESEARCH REPORT

VTT-R-00177-13

Multi-Core Processing from NPP I&C Perspective

Authors: Jukka Ranta

Confidentiality: Public

Version 1.0/1.11.2012

Report's title Multi-Core Processing from NPP I&C Perspective	
Customer, contact person, address VYR	**Order reference** 4/2012SAF
Project name Coverage and Rationality of the Software I&C Safety Assurance	**Project number/Short name** 73831 CORSICA
Author(s) Jukka Ranta	**Pages** 12/
Keywords Multi-core processor, Nuclear power, Instrumentation and control	**Report identification code** VTT-R-00177-13

Summary

This report provides an introduction to multi-core processors from the reliability point of view. An overview of the technology and differences from single-core processors is given with descriptions of some of the methods used for improving computing performance. The impact on reliability is considered along with potential ways to use the technology.

Parallel processing using multiple processor cores provides advantages in computing power with smaller space, weight and power requirements for the hardware. These advantages drive the trend and goals of technological development. Given this trend, eventually these types of processors will be used also for nuclear power plant instrumentation and control applications. Possibly because there simply are no alternatives on the market.

Increased complexity of these devices and their functionality is one of the main concerns when developing safety critical applications. The effort and thereby costs of verification and validation activities can become prohibitively high.

Confidentiality	Public

Espoo 21.1.2013

Written by	**Reviewed by**	**Accepted by**
Jukka Ranta	Jussi Lahtinen	Jari Hämäläinen

VTT's contact address
VTT Technical Research Centre of Finland
P.O. Box 1000, FI-02044 VTT, Finland
Phone internat. +358 20 722 4520
Fax +358 20 722 4374

Distribution (customer and VTT)

SAFIR2014 Reference group 2

Preface

This report has been written as a part of the CORSICA-project (Coverage and rationality of the software I&C safety assurance). One of the tasks in the project is to survey possible future technologies that may become relevant from nuclear power plant instrumentation and control systems. Earlier, FPGA-technology was studied. The overall objective of the project is to improve the efficiency of safety evaluation of software based I&C systems.

CORSICA-project is a part of the SAFIR2014-research programme (Finnish Research Programme on Nuclear Power Plant Safety 2011–2014) which funds projects related to nuclear power plant safety.

Espoo, January 2013

Jukka Ranta

Contents

1. Introduction

This report is prepared as a part of the CORSICA project of the SAFIR2014 research programme. The SAFIR programs (The Finnish Research Programme on Nuclear Power Plant Safety) consider nuclear power plant (NPP) safety from multiple aspects ranging from reactor physics to cognitive aspects of control room design. SAFIR2014 runs from 2011 to 2014. The main focus of CORSICA (Coverage and Rationality of the Software I&C Safety Assurance) is on evaluation and assessment of the reliability and safety of software for instrumentation and control (I&C) systems.

The aim of this report is to give an overview of multi-core processors and parallel processing and their characteristics relevant to use in safety related applications. The main terminology is presented along with the main concepts and technological aspects. The most relevant topics mentioned often in the literature are covered. This should give a basis and a starting point for the reader to follow the developments in the field.

Though it is unlikely that advanced multi-core processors will be used for safety systems in the near future, the trend suggests that eventually the processor market offers few alternatives. Also, even modern single-core processors have features that make them much more complex than their predecessors. These features, such as speculative instruction execution, are also introduced in this report. Multi-core processors are mentioned in the 2009 NRC (U.S. Nuclear Regulatory Commission) report [NRC, 2009] on emerging technologies.

Sections 2 and 3 introduce the background and basics of multi-core processing technology and the drivers of current technological and market development. Section 4 introduces task structure of software execution (processes, threads). Sections 5 and 6 cover performance enhancing features used also in single-core processors and potential problems and faults from poor design. Sections 7 and 8 describe symmetric and asymmetric multi-core processing and virtualization. Section 9 discusses methods to improve robustness of systems using redundancy and diversity on multiple cores and abundant computing resources. Section 10 introduces use of multiple cores and emulation of regular processors on ASIC and FPGA devices. Section 11 briefly considers the standards currently relevant form multi-core perspective. Sections 12 and 13 further consider nuclear specific topics and give a summary.

2. Technology Fundamentals

The essence of multi-core processing is in executing multiple program instructions in parallel and independent of each other. Traditional processors have one core and execute one instruction at a time. The novelty of the technology is actually not in having multiple cores processing in parallel but having those cores on the same processor device. If the cores are on the same chip it is referred to as chip multiprocessors (CMP). The processor cores can also be on multiple chips in a single package. Computers with multiple processors on separate circuit boards and several processors per circuit board have been the basis for “supercomputing” since the 1960s.

Two to eight core general purpose processors are commonplace in the consumer market while processors with hundreds of cores are being designed and already found in more specialised applications, such as, graphics and signal processing. Special purpose devices commonly have a variety of processing cores of different designs for specific tasks, whereas, general purpose processors for the consumer market have a number of identical cores which are often same designs as older single-core processors. These are referred to as heterogeneous and homogeneous cores, respectively. Heterogeneous and special purpose processors have a longer history of multi-core computing with large numbers of cores but the general purpose processors in laboratories are catching up.

From the efficiency point of view, a current issue is redesigning software originally designed to be run on a single-core processor into a structure suitable for multi-core processing. Typically programs are designed to be run as a single stream of instructions being processed in a particular order with the results of earlier computations readily available. With parallelism (and dynamic scheduling of instructions onto a number of cores) it is not always clear what data is up to date. Some computations simply must be run sequentially but, for example, vector operations such as sum of two vectors is easy to process element wise in parallel. The vector sum would be characterised as single instruction multiple data, SIMD, and is a typical feature of modern processors with history reaching back to 1970s and vector supercomputers. Computer systems running multiple separate programs are more suited to take advantage of parallel processing than systems running a single computationally heavy program. The less the computations are dependent on each other and the less there is data transfer, the easier it is to run them in parallel. Due to less than perfect design and structure of software, the full benefits of using multiple cores are not achieved and the efficiency of increasing the number of cores falls short of linearly increasing computing power.
Processors are CMOS technology; for the physical aspects and reliability considerations, see e.g. the earlier CORSICA report on FPGA technology [Ranta, 2012].

3. Development Drivers

The main driver to move to multi-core is the aim to have more computing power with less electric power consumption. Also, reducing the amount of hardware needed for a system is an advantage. Increasing the clock speed of a processor causes a significant increase in the power consumption. This causes heating problems requiring more efficient cooling and also more and more systems run on batteries. The energy consumption of a single-core processor increases approximately as the cube of operating frequency (increasing clock speed requires also using higher voltages). Approaches such as pipelining require more transistors for control and memory, speculative execution does things twice if the guess went wrong, and increasing the number and size of cache memories requires more transistors on the chip and controllers to handle them. Increasing the number of cores causes an approximately linear increase in power consumption (assuming the work can be efficiently distributed onto the cores).
Splitting the computing effort onto multiple processors provides a better scalable approach for increasing computing power. Having the processor cores on the same chip provides significant advantages in transferring data between the cores because accessing RAM can be a significant bottle neck for program execution. On-chip cache memories and high speed intercore communication (via architectures such as bus, crossbar, ring, or network-on-chip) enhance benefits of parallel processing when the processes or threads need to exchange data. If there is no need to transfer any data at all between the cores, the programs could run on entirely separate computers (requiring more space for the equipment).
Forerunners of the use of multi-core processors in the safety area in the future are likely to be the automotive and aviation/aerospace industries. Both have an interest in packing the systems into smaller space with less weight. Also, both have an interest in the power consumption and cooling requirements. There are advantages from the reduced space, weight and power (SWaP) needs also in industrial automation systems.

4. Programs, Processes, Tasks and Threads

A *program* (application) is a set of instructions (an executable file or a set of files) residing on a disk. When it is loaded into memory and being executed it is a *process*, that is, a process is an instance of a program and there can be multiple instances of a specific program. On the other hand, a program can consist of multiple processes. Processes do not communicate or exchange data except through system level methods, for example, accessing the same files. The operating system sees to it that processes do not interfere with each other by, for

example, allocating separate memory areas and other resources for each. A *thread* is like a process but on a lower level and exists within a process. A process can have multiple threads. Threads do not have resources allocated to them like processes do and are often called light-weight processes. Threads share their process' resources, exchange data and need to be designed carefully to avoid race conditions and other potential problems of parallel execution. A *task* is usually defined as a set of instructions loaded into memory and often means a part of a process or a thread.

5. Scalar and Superscalar Processors and Other Performance Enhancements

The following describes a number of approaches to improve efficiency of a single-core processor (or a single core on a multi-core processor). Many multi-core processors are based on single-core designs and each of the cores uses these technologies.

A *scalar processor* executes one instruction at a time on each clock cycle with none of the special features of a superscalar processor. It takes one instruction and related data and processes them and then starts fresh with the next instruction. *Superscalar processors* can execute more than one instruction per clock cycle and break the simple scalar progress in program execution. A superscalar processor contains several functional units (e.g., Arithmetic Logic Units (ALU) and Floating-Point Units (FPU)) within the processor and they are not all needed for each instruction. Allocating the instructions suitably onto the functional units multiple instructions can be executed in parallel. The efficiency of this approach depends on how accurately the instructions are dispatched onto the functional units to keep them occupied.

Pipelining is another approach to make best use of the resources on a processor. Executing one instruction consists of multiple steps from fetching the instruction from memory to writing the output. These steps use different parts of the processor which can process different instructions. Therefore it is possible to "pipeline" instructions to execute at the same time but at different steps. For example, on first cycle instruction A is at step 2 and instruction B at step 1, then on the second cycle instruction A has moved to step 3, B to step 2 and a new instruction C starts at step 1. Pipelining is instruction level parallelism, whereas multiple cores implement thread level parallelism.

Speculative execution starts executing a conditional branch (based on an "if"-statement) before the needed comparison (result of the "if") has finished executing. The branch instructions are placed into the pipeline before the comparison results are available. The choice is based on a prediction or a guess of which branch is more likely and the associated instructions are processed while waiting for the comparison results. If the guess is wrong, the results are scrapped and the correct instructions are then processed.

If the particular data needed by the next instruction is not available in the cache and access to memory is slow, *out of order execution* can be used to keep the processor busy doing something (hopefully) useful while waiting for the data. Some other instruction which has its data available is executed instead of the one that should be next. Hence, the instructions are not executed in the original order.

Because the access to RAM is slow compared data transfer on chip and execution of instructions, *cache memories* are used to hold some of the data on the chip. The efficiency can be improved by using larger caches in multiple levels (named L1, L2, etc.). Typically program instructions and data have separate caches. The efficiency also depends on cache management (there are multiple alternative replacement policies), that is, what data is dumped from the cache when more is retrieved from RAM. Therefore, the specifics of the data in the cache and order of instructions affects the final efficiency as a *cache miss* (the needed data was not found in a cache) can cause a significant delay of possibly hundreds of cycles while waiting for data to arrive from RAM (when a processor stops because it has nothing to do while waiting, is called a *stall*). Further, having copies of data in multiple caches requires measures to be taken to maintain coherence of the data [Blake et al., 2009].

6. Symptoms of Problems of Poorly Designed Scheduling

Several different types of problems may arise when the design of the software is a mismatch for the operating system and hardware platform. Mainly these problems are due to allocation of shared resources and scheduling of processes and threads onto cores. The symptoms are seen as the worsening of the following characteristics: *Latency* is the time delay or response time of the system; *Jitter* is the variation in response time; and *Throughput* is the total data flow. A problem can also manifest as: *Lockout* happens when one core prevents other cores from accessing a shared resource; *Deadlock* is a situation in which no one can access a resource because (at least) two requests wait for and block each other. Using deterministic scheduling (static schedules) of instructions helps with these problems but also the access to all shared resources should have a deterministic schedule. Generation of static schedules and other scheduling issues are discussed in [Hilbrich, Goltz, 2011]. Worst case execution time (WCET) analyses, real-time performance requirements and related design guidelines are discussed in [Kästner et al., 2012]. Timing anomalies is the term used for counter intuitive scheduling problems arising from seemingly small changes.

7. Symmetric and Asymmetric Multiprocessing

Symmetric multiprocessing (SMP) and Asymmetric multiprocessing (AMP) refer to the platform and the way processes are handled. When using symmetric multiprocessing, there is one operating system and all processes run under it and are scheduled by it. The reliability, especially real time performance, of SMP is greatly influenced by the load balancing algorithms used by the OS to schedule tasks onto the cores and allocate other resources. Asymmetric multiprocessing approach considers each core as a separate processing element and there is little or no intercore communication. Different cores can run different processes or even different operating systems. The presence and need to use shared resources creates additional complexity to an AMP system (see Virtualization below).

8. Virtualization, Supervisors, and Hypervisors

Efficient computing resources allow virtualization of computing environments. The overall system can run on one set of hardware but contain multiple systems which can have their own operating systems and tasks which know nothing about each other. The common resources are allocated by a higher level controller, a hypervisor, which ensures that the operating systems all have the resources they need and do not interfere with each other. A single operating system acts as a supervisor to control the access that individual processes have to hardware. A hypervisor is a step up in the ladder as it controls multiple operating systems, see Figure 1.

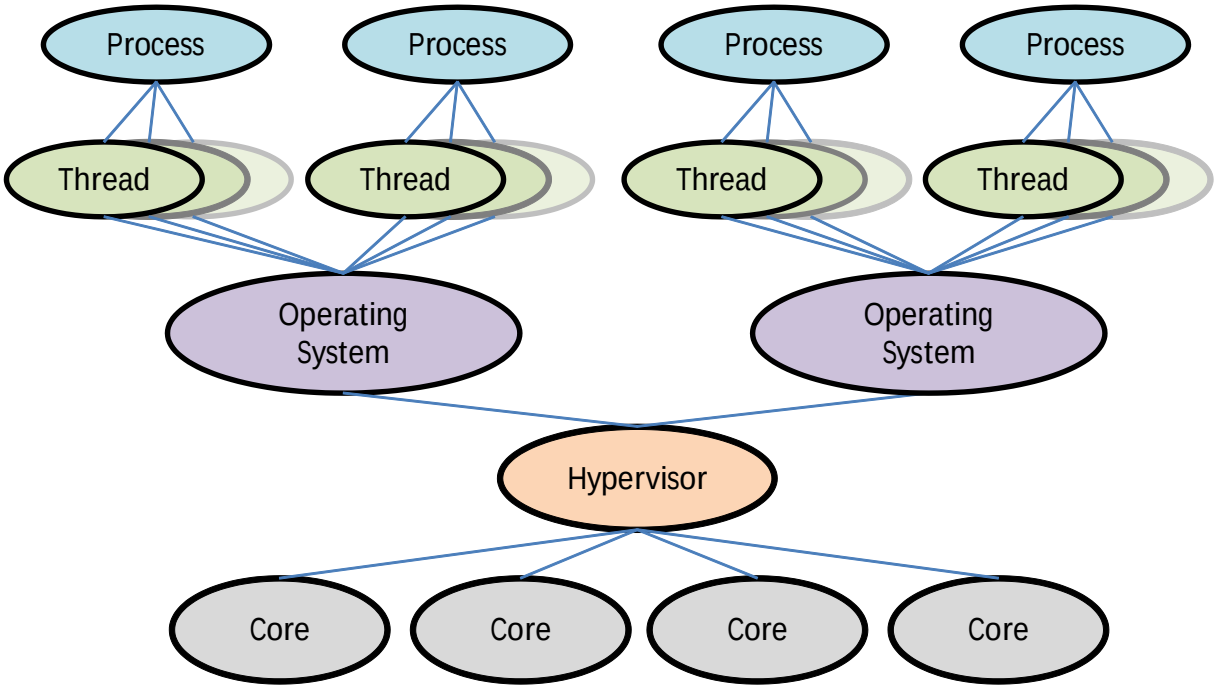

Figure 1. Structure of operating systems controlled by a hypervisor in a virtualized environment. The operating systems do not know about each other or about the underlying hardware.

9. Error Detection and Recovery, Redundancy and Diversity on Chip

Multiple processors provide resources for redundancy and diversity (see [Reichenbach, Wold, 2010 and Villalpando et al., 2011]). Unused resources can provide redundancy to compensate for permanent faults, for example, by switching computations from a failing core onto an unused core which can have its own cache memory. In addition to multiple cores, chips can have multiple power pins, I/O ports and other resources. Running the same computations on multiple cores in parallel allows an easier switch to using the results of a redundant core but also the comparison of the results to check correctness. For example, three parallel processors provide a triple modular redundancy (TMR) structure. Different versions (by different design teams) of the software provide diversity that could allow a TMR scheme to discover design errors in addition to hardware failures.

The term *lockstep* execution or lockstep mode refers to running multiple copies of the same code parallel on multiple processor cores. With some separate and some common caches for cores, shared access to memory and other shared resources together with dynamic scheduling means that clock cycle by clock cycle lockstep execution does not necessarily happen automatically when executing multiple identical processes or threads. Hence, the platform needs to be designed to facilitate lockstep mode and some chips on the market feature special lockstep cores.

As an internal protection against errors many processors and platforms implement error checking for data transfer. Error detection and correction codes and redundant data (parity data) are used to verify that data has not changed during transmission (detection) or can be recreated (correction) if it does. These approaches provide protection against, for example, errors caused by radiation or power fluctuation. As more and more transistors are packed onto the chips, their susceptibility to outside effects increases (smaller transistors are less robust and more of them means more potential error sources).

A drawback of having redundancy and diversity on the same chip is the susceptibility to common cause failures affecting the circuit board the chip is on or originating from it, such as, power fluctuation or error in the clock signal. Also, elaborate methods to improve reliability are likely to increase complexity and the overall benefits may be lost as verification and validation become more laborious.

10. IP Cores for FPGAs and ASICs

Multiple cores can be present also in devices such as FPGAs (Field Programmable Gate Array) or ASICs (Application Specific Integrated Circuit) which are electronic circuits designed for specific purposes. As such they can implement parallel processing without the software - operating system - processor structure but with the logic built into the hardware circuitry. The design process of the circuitry allows the use of predesigned blocks and components that can be combined with application specific designs rather similar to subroutine libraries for software. These are known as IP (Intellectual Property) cores and can implement actual processor cores that execute software instructions. This approach allows emulation of no-longer existing processors types and production of custom multi-core processors by including multiple IP cores (copies of the same or different cores) onto a single device.

11. Standards and Certification

The standards that are often mentioned in literature on multi-core processors in safety related applications reflect the application areas which have interest in the SWaP advantages, namely automotive (IEC 26262) and aerospace (DO178B for software and DO-254 for hardware). The higher generic IEC 61508 is also often referenced. The nuclear standard 61513 has been mentioned in the literature in the multi-core context but for now nuclear applications are far from the focus of interest.
Certified multi-core computing platforms are beginning to appear on the market, again, with aerospace and automotive leading the way. In the aviation field Integrated Modular Avionics (IMA) concept and ARINC 600-series standards are used. The automotive industry has AUTOSAR (AUTomotive Open System ARchitecture) partnership (http://www.autosar.org/). On the other hand, an example on the research side is the RECOMP-project which aims to find approaches to achieve more cost efficient certification. In [Reichenbach, Wold, 2010] some examples of certified products and current certification activities are mentioned and [Fuchsen, 2010] considers numerous aspects of reliability of multi-core platforms for aviation. Thus, with time there will be a selection of platforms certified according to various standards along with experience on developing them. However, nuclear won't be a forerunner.

12. Nuclear Perspective

For a safety system intended to be used in a nuclear power plant, simplicity is a key advantage. Verification and validation of the design are simpler and can be performed with less effort. It also helps with licensing, maintenance and in the future replacing the system with something more up-to-date. Multi-core processing is not simple. In particular, when compared to old scalar processors the difference is significant.
The overall trend is moving to multi-core processing. In a decade or two we may be in a situation where all available "modern" processors are based on some type of multi-core approach with various other performance enhancing features included. For a software based system designed at that time and intended to have a lifetime of decades, the alternatives are limited. On the other hand, other areas with safety requirements will push the development and reliable platforms will be available and potentially adaptable to the needs and requirements of nuclear applications.

The increase in computing power will have an impact on design of systems. For example, testing by simulation can be more comprehensive given a certain amount of time to run the simulations and the same improvements also allow running more static analyses during design.

13. Conclusions

Transition from simple scalar processors to superscalar processors and the use of pipelining, speculative execution, and other performance enhancing features makes it less transparent and predictable what is going on inside a processor in different situations. With the addition of multiple processing cores running in parallel, the situation becomes even less clear as the order in which instructions of different threads and processes are executed may vary and the cores compete for shared resources, such as, access to the main memory.

Determinism in performance is a key property to strive for. This objective has multiple aspects. Scheduling of processes running on cores should be based on static schedules instead of dynamically scheduling them at runtime. The use of shared resources should also be deterministic. This means, for example, intercore communication, common caches, and memory controllers. For systems that have strict reliability requirements this is an important property and in particular important to achieving reliable real-time performance.

One approach to improve reliability is to carefully divide the available caches. Some areas of caches can be reserved for instruction while others contain only data. Caches used by multiple cores can be segmented for those cores and similarly different processes can have their own predesignated cache areas. Specific cache areas can be allocated for intercore communication so that only one core, process or other entity has write permission. These approaches reduce risk of conflicts on resource use and scheduling problems resulting from cache misses. Also, with such a strict approach the risk of memory corruption is reduced. The memory controller providing access to main memory is also a shared resource and it can become a bottleneck for maintaining availability of needed data in the caches and worst case performance should be evaluated.

Asymmetric multi-core processing has less interaction between cores and is the recommended approach over symmetric multi-core processing where an operating system schedules the use of the cores and other resources. Downside is the increased complexity due to the need for a hypervisor level in the architecture, that is, the result is more but simpler layers.

If the computational load is sufficiently small, using just one of the cores can be an option. This simplifies the structure of computations significantly. If the future trends go towards processors with a large number of simple cores instead of just a few (or a few dozen) more versatile ones, using just one processor core that behaves like a scalar processor may be a very good choice in safety related NPP systems. The SWaP advantages are not as important as in fields like aviation and automotive industry. Instead, laborious design and V&V activities needed to produce and licence a system or platform have greater importance. Furthermore, the available resources and computing power make it easier and more tempting to run several functions on the same hardware but this makes the system more susceptible to common cause failures.

Running multiple cores in lockstep is another way to keep the situation more orderly and transparent. This approach also facilitates detection of hardware malfunctions by running multiple copies of the same process on different cores and comparing the results, for example, triple modular redundancy. Design errors can be detected by using different versions or designs in parallel.

References

Blake, G., Dreslinski, R., Mudge, T., 2009. A Survey of Multicore Processors, IEEE Signal Processing Magazine, pp. 26-37.

Fuchsen, R., 2010. How to Address Certification for Multi-core Based IMA Platforms: Current Status and Potential Solutions, 29th Digital Avionics Systems Conference.

Hilbrich, R., Goltz, H.-J., 2011. Model-based Generation of Static Schedules for Safety Critical Multi-core Systems in the Avionics Domain, International Workshop on Multicore Software Engineering, IWMSE'11, pp. 9-16.

Kästner, D., et al., 2012. Meeting Real-Time Requirements with Multi-core Processors, SAFECOMP 2012 workshops, pp. 117-131.

NRC, 2009. Instrumentation and Controls in Nuclear Power Plants: An Emerging Technologies Update, NUREG/CR-6992.

Ranta, J., 2012. The Current State of FPGA Technology in the Nuclear Domain, VTT Technology: 10.

Reichenbach, F., Wold, A., 2010. Multi-core Technology - Next Evolution in Safety Critical Systems for Industrial Applications, 13th Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools, pp. 339-346.

Villalpando, C., Rennels, D., Some, R., Cabanas-Holmen, M., 2011. Reliable Multicore Processors for NASA Space Missions, Aerospace Conference, 2011 IEEE, pp. 1-12.