Java Environment for Parallel Real-Time Development

Project Number 216682

D8.1– Whitepaper

Version 0.4
September 8, 2008
Draft

Public Distribution

aicas with contribution from all partners

Project Partners: **aicas**, **EADS Deutschland**, **FZI**, **RadioLabs**, **SkySoft Portugal**, **SYSGO**, **Technical University Cluj-Napoca**, **The Open Group**, **Technical University of Vienna**, **University of York**

Project Partner Contact Information

aicas Fridtjof Siebert Haid-und-Neu-Strasse 18 76131 Karlsruhe, Germany Tel:+49 721 663 968 23 Fax:+49 721 663 968 93 E-mail:siebert@aicas.com	**EADS Deutschland** Thomas Mahr Woerthstrasse 85 89077 Ulm, Germany Tel: +49 731 392 7469 Fax: +49 731 392 2074 69 E-mail:thomas.mahr@eads.com
FZI Gábor Szeder Haid-und-Neu-Strasse 10-14 76131 Karlsruhe, Germany Tel: +49 721 965 4266 Fax: +49 721 965 4259 E-mail:szeder@fzi.de	**RadioLabs** Filippo Corsini via Cavaglieri 26 00173 Rome, Italy Tel: +39 069 727 8250 Fax: +39 069 727 8268 E-mail:filippo.corsini@radiolabs.it
SkySoft Portugal José Neves Av. D. João II, Torre Fernão Magalhães, 7º 1998-025 Lisbon, Portugal Tel: +351 21 382 9366 Fax: +351 21 386 6493 E-mail:jose.neves@skysoft.pt	**SYSGO** Jacques Brygier 5, Rue Hans List, Batiment F 78290 Croissy-sur-Seine, France Tel: +33 1 300 912 63 Fax: +33 1 301 504 48 E-mail:jacques.brygier@sysgo.fr
Technical University Cluj-Napoca Gheorghe Sebestyen-Pal G. Baritiu 26-28 40027 Cluj-Napoca, Romania Tel:+40 264 401 476 Fax:+40 264 594 491 E-mail:gheorghe.sebestyen@cs.utcluj.ro	**Technical University of Vienna** Martin Schoeberl Treitlstrasse 3 1040 Vienna, Austria Tel: +43 15 880 118 207 Fax: +43 15 869 149 E-mail:mschoebe@mail.tuwien.ac.at
The Open Group Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 675 7721 E-mail:s.hansen@opengroup.org	**University of York** Andrew Wellings Heslington Hall York YO10 5DD, United Kingdom Tel: +44 1904 432 742 Fax: +44 1904 432 767 E-mail:andy@cs.york.ac.uk

Document Control

Version	Status	Date
0.1	Ingo Prötel: First Draft	2008-06-27
0.2	Fridtjof Siebert: Added contents, revised structure.	2008-08-06
0.3	Fridtjof Siebert: Fixed layout, added Figures on WPs and toolchain.	2008-08-14
0.4	Fridtjof Siebert: Added corrections suggested by Martin&Andy	2008-08-18

This page was intentionally left blank.

Abstract

Multicore systems have become standard for desktop computers today and current operating systems and software development tools provide straightforward means to use the additional computing power. However, a more fundamental change in the design and development of software is required to fully exploit the power of multicore systems. Furthermore, the fast growing market of embedded systems is currently largely unaffected by the introduction of multicore systems. This will change quickly in the future, which will mean that there will be a demand on efficient development of reliable embedded software that can give real-time guarantees and exploit the available power on multicore systems.

The JEOPARD project addresses this demand by developing software tools to exploit multicore power while ensuring correctness and predictable timing.

Introduction – Going for Multicore

It is recognised that there is a new software crisis facing the industry. To continue the increasing growth of computer power requires a transition to multicore architectures, yet our software development models are rooted in sequential programming. Add into this mix the growing use of embedded computers and their ever increasing demand for better performance within constrained power consumption (possibly battery supplied). The result is an urgent requirement for tools to support multicore embedded software development.

Whilst it is currently beyond the state of the art to productively develop software for the full power provided by multicore architectures [1], the JEOPARD project believes that there is a migration path that can be plotted from the current state of embedded systems practice to a multicore future.

Multicore Symmetric Multiprocessor Systems (SMP) have become standard for desktop computers today and current operating systems and software development tools provide means to use this additional computing power. Increasingly, embedded systems are being proposed and built using multicore systems, e.g., mobile phones. Future embedded systems may well exploit more complex multicore architectures such as NUMA (Non-Uniform Memory Architecture) and NoC (Network-on-Chip).

In the near future, more demands will be made on these embedded systems and greater reliance will be placed on the delivery of their services. More and more of these systems will become high-integrity systems whose failure can cause loss of life, environmental harm, or significant financial loss.

The increasing demand on the processing power for the more and more complex tasks realised by these systems can only be met by the use of multicore systems. In addition to desktop systems, embedded systems have additional requirements on predictable timing behaviour and safety-criticality.

Hence, the JEOPARD project believes that the transition to full multicore platforms can be achieved in two stages. The first stage is to provide an environment within which embedded software can be developed for SMP platforms. Embedded systems use a large variety of different architectures, such

that portability via uniform interfaces to the systems resources is required to support the versatility in terms of performance, power and coping with devices that range from low-end controller systems to powerful high-end embedded systems. Whilst many of these systems can be reflected in a SMP-type of architecture, others require a NUMA. Integrating access to these architectures within a software development environment now requires study and the development of prototype tools. These tools need to be evaluated to determine to what extend they can be used for future industrialisation. This is the second stage of the transition. The study and prototyping can be done in parallel with the first stage.

The main strategic objective of the JEOPARD project is to provide the tools for platform-independent development of predictable systems that make use of SMP multicore platforms. These tools will enhance the software productivity and reusability by extending technology that is established on desktop system by the specific needs of multicore embedded systems. The project will actively contribute to standards required for the development of portable software in this domain.

Whilst current platforms may consists of between 4 and 8 homogeneous RISC processors with an SMP architecture style, this approach does not scale to future platforms where a NUMA architecture is more appropriate. Consequently, the JEOPARD project will undertake research into how such systems can be programmed and analysed for their real-time properties. Where appropriate proof-of-concept prototypes will be developed.

Hence, the JEOPARD project attempts to strike a balance between meeting the need for industrial strength tools to ameliorate the current crisis, and plotting a migration path to future exploitation of the full power and generality of multicore platforms.

Next Generation Coding for Multicore Systems

For embedded real-time systems to profit from the additional computation power means a change in the way software is created. Traditionally, there is one control task on a processing node, consequently, performance cannot be improved by simply distributing tasks among the cores. But rather the task itself must be parallelised. This requires a fundamental change in how the programmers create real-time control software. Time-critical tasks need to be restructured to enable parallel execution, and the resulting parallel actions have to be mapped to the set processors.

In real-time software development on single core systems it is always clear that the highest priority thread runs and has full and immediate access to all resources in the system. On a multicore system, however, a second lower priority thread may run and competes for other resources in the system, such that actual throughput might go down due to lock contention. New classes of failures will appear and some of these, like data race-conditions, will move from a mere theoretical possibility to actual faults that crash systems at runtime.

Another opportunity provided by multicore systems is to distribute functionalities that used to run on a single processor over the available cores. Previously having one node exclusively execute one functionality guaranteed quality of service at a fundamental level. The integration of several functional task on one multicore system must ensure at least the same quality of service. The user must be able to model the requirements for functionalities and enforce these under all circumstances.

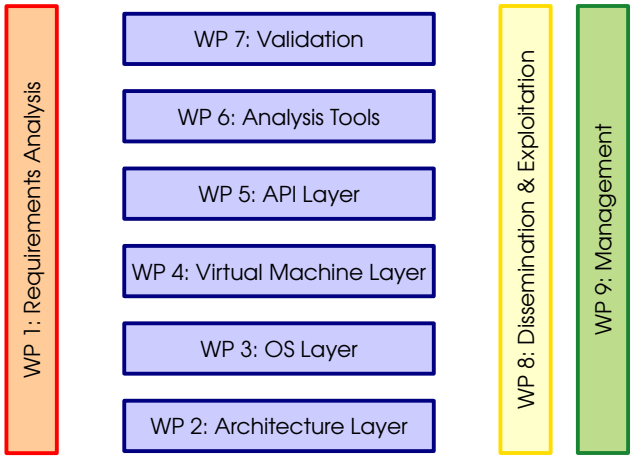


Figure 1: *The JEOPARD project work package structure shows the layered structure of the project. Orthogonal activities are the requirements analysis, validation and project management.*

Technologies for Multicore Systems

The JEOPARD project has partners that work on all layers involved in a multi core system, from the bare hardware within a multicore processor, via a multicore real-time OS, a multicore real-time Java VM, Java APIs for multicore systems up to different critical applications and tools for the analysis of the correctness of these applications. An overview of the layers and the corresponding work packages in the JEOPARD project is shown in Figure 1. The following sections will explain these individual layers in more detail.

An overview of the different building blocks for the JEOPARD toolchain is shown in Figure 2. Central to the toolchain is the Java application. The developer can use specifications developed by the JEOPARD project partners to make use of multiple processors. The application is run on two possible execution environments, a VM running on an off-the-shelf multicore processors, or a parallel Java bytecode processor. For the correctness validation, a static analysis tools and a concurrent unit testing tool are developed'

CPU Architecture Layer

The basis of a multicore architecture is a Chip Multiprocessor (CMP). The JEOPARD project will consider three basic variants of a CMP to support Java programs targeted at multicore platforms:

1. SMP – a symmetric arrangement of conventional processors, as typified by Pentium class multicore CPUs.

2. Multi-JOP – a number of hardware Java processors with some shared memory (based upon the Java processor JOP [6, 7]).

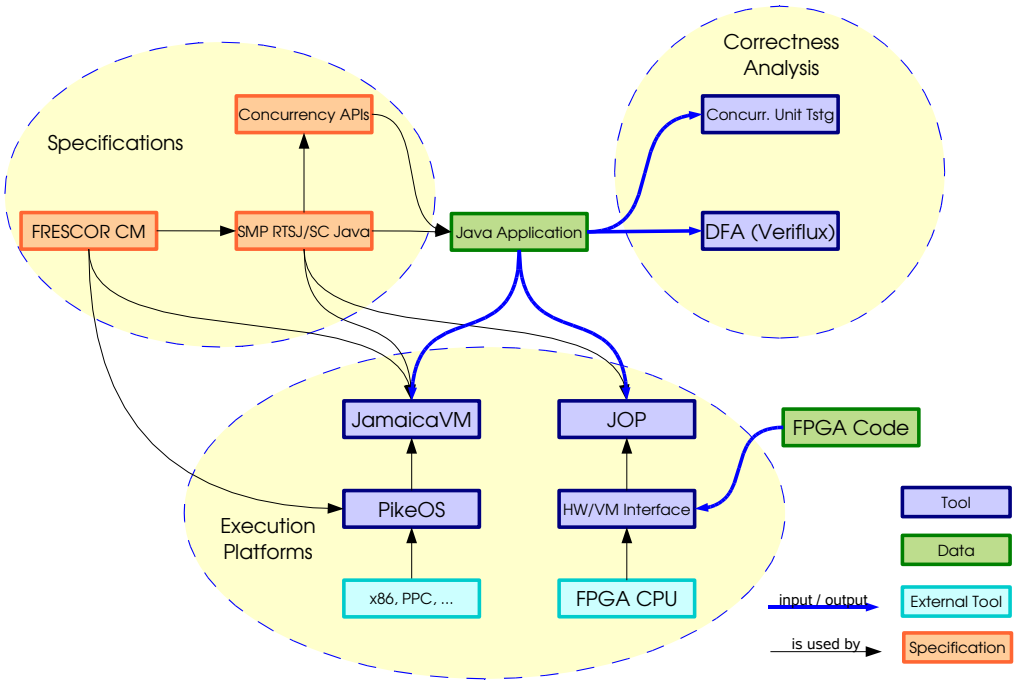


Figure 2: *The JEOPARD toolchain consists of three main aspects: specifications, correctness analysis and execution platforms.*

3. NUMA – to incorporate future architectures, including those with substantial field-programmable elements.

For all architectures, the project will develop efficient support for higher level OS and Java based services. In particular, specific services for synchronisation, scheduling support and deterministic automatic memory management – these will improve overall system performance.

The scientific aims of the CPU architecture part of JEOPARD are:

- to achieve a better understanding of the tradeoffs between software and hardware implementations of Java core functionality;
- to examine the bottlenecks inherent in current implementations of the Java VM within real-time systems;
- to examine and understand the implications of future multicore and CPU architecture trends upon software and hardware implementations of the Java VM, including presence of field-programmable elements within the chip.

OS Layer

One programming model for a multicore CPU is that of an SMP, as used in many applications from supercomputers to high-performance server systems for many decades. Within the JEOPARD

project, the OS layer is considered within an SMP context only (ie. the OS is not considered for Multi-JOP, which has no need for an OS, nor within NUMA). In order to exploit the power of the platform, efficient and effective use of all CPUs must be made. This can be achieved in many ways, from explicit use of all CPUs directly, perhaps via a separate OS on each CPU; through to virtualising the platform so that higher levels are not aware of the underlying multi-CPU platform.

The computational load of server systems typically consists of a large number of mostly independent processes. Moreover, the performance of a server system is usually measured as average throughput: If there are any deadlines to be met by processes at all, they are "soft" at best, i.e. missing a deadline is not considered to be a fatal error. With such requirements, a "load balancing" approach is entirely suitable: all processes are treated alike at the operating system level. They are selected for execution on a "first come, first serve" basis as processor cores become available. Today, most multiprocessor operating systems work by such a strategy. In some cases, heuristics are used in addition, e.g. to improve cache utilisation by avoiding excessive process migrations. Obviously, such techniques are not suited for scheduling processes with hard timing requirements.

For a real-time system, these techniques are not sufficient, more important than high average throughput are worst-case execution times of each single task to ensure that all deadlines will be met and that priorities are respected.

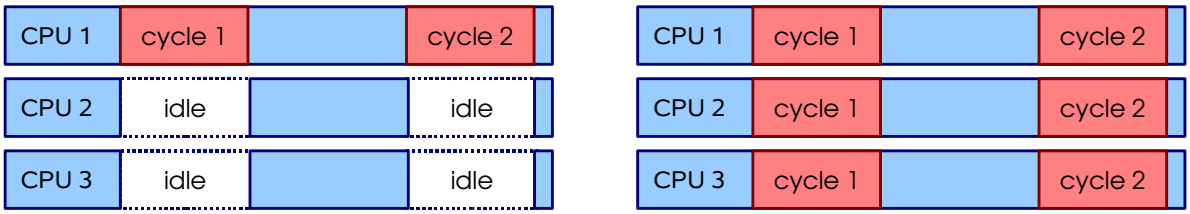
Although POSIX (and its real-time extensions) currently does not provide specific support for SMP systems, the issue has been raised [4]. POSIX.1 defines the "Scheduling Allocation Domain" as the set of processors on which an individual thread can be scheduled at any given time. POSIX states that [5]:

- "For application threads with scheduling allocation domains of size equal to one, the scheduling rules defined for SCHED_FIFO and SCHED_RR shall be used;"
- "For application threads with scheduling allocation domains of size greater than one, the rules defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an implementation-defined manner."
- "The choice of scheduling allocation domain size and the level of application control over scheduling allocation domains is implementation-defined. Conforming implementations may change the size of scheduling allocation domains and the binding of threads to scheduling allocation domains at any time."

With this approach, it is only possible to write strictly conforming applications with real-time scheduling requirements for single-processor systems. If an SMP platform is used, there is no portable way to specify a partitioning between threads and processors.

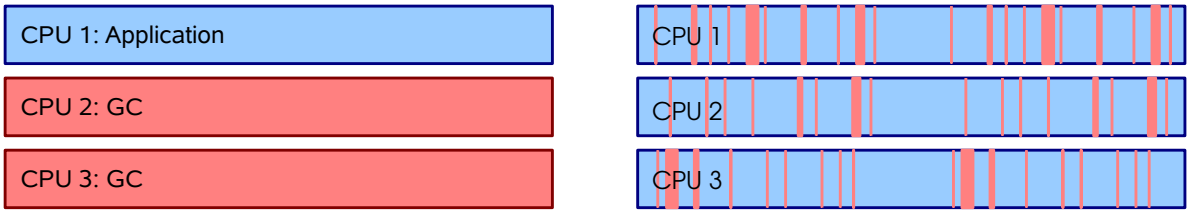
Additional APIs have been proposed but currently these have not been standardised. The approach has been to set the initial allocation domain of a thread as part of its thread-creation attributes. The proposal is only a draft and so no decision has been taken on whether to support dynamically changing the allocation domain.

Since Kernel version 2.5.8, Linux has provided support for SMP systems [3] via the notion of CPU affinity. Each process in the system can have its CPU affinity set according to a CPU affinity mask. A process's CPU affinity mask determines the set of CPUs on which it is eligible to run.



Blocking GC Parallel blocking GC

Figure 3: *A blocking garbage collector cannot be used in real-time systems, independent of whether it is a single processor or parallel garbage collector.*



Concurrent parallel GC Incremental parallel GC

Figure 4: *A concurrent parallel garbage collector can use use a subset of the available processors and run in parallel of the application. However, most flexible is a parallel GC that can be fully interleaved with the application execution, i.e, a fine-grain incremental parallel GC.*

VM Layer

For real-time Java applications to run on multicore embedded systems, a multicore real-time Java VM implementation is required. The runtime services of the VM such as the interpreter, thread synchronisation, memory management, etc. must be implemented to exploit the available processors while executing predictably and efficiently.

The biggest challenge for such a Java VM is to provide real-time garbage collection that can execute in parallel and without preemption of the Java application. A non-parallel blocking GC cannot be used in a real-time system, even a standard parallel GC is still blocking, see Figure 3.

What is needed is at least a parallel concurrent GC that uses a set of CPUs reserved for garbage collection work. But the highest flexibility can be reached only with an incremental parallel GC as shown in Figure 4.

Such a real-time garbage collector must ensure that sufficient free memory is available even though many processors may be performing memory allocations and garbage collection in parallel. Fine grain synchronisation techniques need to be developed to ensure that this parallel execution does not suffer from contention. In addition to the implementation of a parallel real-time garbage collector, a theoretical analysis of the level of parallelism that can be achieved is needed [8].

The parallel execution of thread synchronisation via Java monitors requires a careful implementation of the primitive monitor operations such as entering, exiting and notify. Also, the Java thread scheduler must know about multiple processors, parallel Java interpretation and JIT compilation.

The result will be the first deterministic Java implementation for parallel systems. The users will be able to develop deterministic parallel applications while profiting from the safety, flexibility and productivity of a Java environment.

API Layer

For a standard desktop computing environment, the application programmer is content to allow the OS to manage the access to the computing resources. In an embedded environment this is generally not the case. Better real-time performance can often be obtained by partitioning the applications threads between the core processors. Currently, there is no easy way to do this using Java or the RTSJ API. This is an accepted limitation of the current real-time Java technology. Although some consideration has been given to an API to do this within the context of JSR 282, the proposal is not based on practical experience.

The work on the API layer is concerned with extending the Java programming model to allow access to the resources in a machine-independent way. The focus will be on providing support for static partitioning but dynamic aspects will also be considered.

The implementation of this extended functionality will build on appropriate support to be provided by the underlying OS layer. Specifically, the OS will have to enable the Java virtual machine to fully control parallel vs. time-multiplexed execution of its threads as well as to control which threads are executed on which processor core (i.e. processor affinity) and even to migrate threads between cores. Current state-of-the art RTOSes fail to provide this kind of functionality. Also, OS API standards do not yet specify such functionalities. Therefore, the JEOPARD project will develop new definitions and specifications of such new OS API functionalities.

In addition, the APIs to allow hardware-implemented FPGA components to be integrated within the overall Java framework will be addressed. Furthermore, the impact at the API-level of a more general NUMA architecture will be considered.

Tools Layer

Independent of the implementation and run-time aspects, reliable applications running on parallel systems require in-depth analysis of the correctness of the implementation and of time-related aspects such as schedulability, lack of deadlocks, race conditions, etc. With the use of multicore systems, parallel execution becomes the norm such that errors that do not manifest as faults on single CPU systems are much more likely to cause system failure.

For the correctness analysis of parallel applications, existing analysis tools need to be enhanced to find errors related to parallel execution through static analysis or concurrent unit tests. Such tools will be developed in the JEOPARD project.

Real-Time for Multicore and Standards

The Real-Time Specification for Java (RTSJ) is an established standard for real-time software development in Java [2]. The JEOPARD project therefore works closely together with the RTSJ-related Java Specification Requests (JSR 282) to ensure that future versions of the RTSJs are conforming to the requirements that arise on multicore systems [9].

RTSJ and SMPs

In order to make the RTSJ fully defined for SMP multiprocessor systems, the following issues need to be addressed.

1. The dispatching model – the current specification has a conceptual model which assumes a single run queue per priority level.
2. The allocation model – the current specification provides no mechanisms to support processor affinity,
3. The synchronisation model – the current specification does not distinguish between synchronised methods that suspend holding their locks and those that do not,
4. The cost enforcement model – the current specification does not consider the fact that processing groups can contain scheduling objects which might be simultaneously executing,
5. The affinity of interrupts (happenings) – the current specification provides no mechanism to tie interrupts (happenings) and their handlers to particular processors, and
6. The failure model – the current specification makes no statements about partial failures of the underlying platform.

The JEOPARD project considers each of the above issues in turn and proposes and implements corresponding APIs.

Use Cases

Industrial partners within the JEOPARD consortium will apply the tools and technologies of the JEOPARD project for the development of real-world applications. The three industrial applications that will be developed are from the following domains:

1. **Multicore Radar Processor**: The tool chain will be used to develop an example radar processor based on a hybrid microprocessor-FPGA-system.

2. **Software Radio**: The multicore embedded processors enables the rapid reconfiguration of high-performance radio systems. The target is the validation of a system that can be rapidly reconfigured changing quickly the channel coding in order to adapt the device dynamically to the wireless communication systems features by selection among several advanced techniques such as Convolutional codes, Turbo-Codes, LDPC codes.

3. **Airline Operation Communication Solution**: On-board airplane component that handles communication related to flight plan and other on-board services. Applications have different safety levels and run in separate OS partitions.

JEOPARD partners

The JEOPARD consortium consist of the following industrial, academic, technology and standardisation organisations.

aicas

EADS Deutschland

FZI

RadioLabs

SkySoft Portugal

SYSGO

Technical University Cluj-Napoca

Technical University of Vienna

The Open Group

University of York

Conclusion

The technologies developed by JEOPARD address the problems that arise when multicore systems will become widespread in embedded applications.

JEOPARD will have its main strategic impact on the future of real-time embedded systems. The ever increasing ubiquity of embedded systems is resulting in the automation of ever more critical

tasks and their increasing importance to society. The move towards parallel, multicore systems that has already happened for desktop computers is also becoming the standard for embedded systems. More demands will be made on them and reliance will be placed on the delivery of their services. Increasingly, their failure could cause loss of life, environmental harm, or significant financial loss. These additional requirements for embedded systems require specific support for the development such that these systems will remain safe and secure despite their ever increasing complexity and interconnectedness.

The JEOPARD project works on advancing multicore systems implementation technologies so that the additional demands for embedded systems, safety, robustness and deterministic timing can be attained at a much lower cost. The envisaged impact is extremely important, since the safety and security of society as a whole is becoming ever more dependent on powerful embedded applications.

With the results of JEOPARD, multicore embedded real-time applications can be developed in Java. They could benefit from the same type of platform independence that is found in today's enterprise applications. These application will also benefit from a wealth of tools that is nearly unsurpassed. A majority of industries in the area have declared interest. Some of them have taken pro-active actions to make this happen (e.g., Boeing).

Many business critical applications (home applications, telematics applications, consumer applications) are already developed in Java. Actually most applications which involve service delivery are probably already based on Java because of the need to deliver the same applications to millions of different consumer systems (write once, run anywhere). Major frameworks and profiles have been defined for such applications: DVB-MHP for content based applications, OSGi for control based applications, MIDP for mobile devices. But as the services and applications to be deployed become more demanding on the computational power, it will be necessary to provide multicore real-time virtual machines that will enable the operation of these applications in a dependable manner. JEOPARD's contribution is a requirement to enable the mass deployment of such real-time embedded applications.

Acknowledgement

This work was partially funded by the European Commission's 7th framework program's JEOPARD project, #216682. Contributions came from all project partners, in particular from Jose Almeida, Neil Audsley, Jacques Bruygier, Annalisa Durantini, Scott Hansen, Thomas Mahr, Vlad Olaru, Ingo Prötel, Martin Schoeberl, Tobias Schoofs, Gabor Szeder, and Andy Wellings.

Glossary

API or Application Programming Interface is the external documented interface of a software library.

GC, Garbage Collection is a memory management system that automatically deletes objects from the heap when they are no longer reachable, so that the memory can be reused. A real-time Garbage

Collector is only invoked at a defined time, can be interrupted by higher priority threads with very low latency, and can guarantee an upper bound of execution time.

Java An Object Oriented programming language defined by SUN Microsystems. Unlike C++, Java is not a hybrid language, but enforces the development of Object-Oriented programs. Java is robust, type-safe language with an integrated Garbage Collector. It comes with a very large standard library, such that developers can focus on the main purpose of their applications. All this contribute to Java's popular for desktop and server applications.

Java VM The runtime environment used to execute *Java* applications. Typically, the Java VM contains a Java class loader, a Java bytecode interpreter, a memory management system using *garbage collection* and specific interfaces to I/O services or to other programming languages.

Multicore A multicore is one that combines two or more independent processors into a single package, often a single integrated circuit (IC).

NUMA or Non-Uniform Memory Architecture is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors. NUMA has a single address space at the native instruction level, shared memory, and non uniform access time to main physical memory.

RTSJ or Real-time Specification for Java is an interface defined by the Real-Time Java Experts Group (RTJEG) under the Java Community Process. It extends the specification of the Java language and library specification with features required for the development of real-time systems using Java. RTSJ does not require the presence of a real-time Garbage Collector, but the presence of such a Garbage Collector simplifies the development of real-time systems significantly.

SMP Symmetric Multiprocessor. SMP architectures have a single address space at the native instruction level, shared memory, and uniform access time to main physical memory.

Bibliography

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 18 2006.

[2] Greg Bollela. *Real-Time Specification for Java*. Addison-Wesley, 2001.

[3] Linux Manual Page. sched_setaffinity(), 2006. http://www.die.net/ doc/ linux/ man/ man2/ sched_setaffinity.2.html.

[4] Michael Gonzalez Harbour. Supporting SMPs in POSIX, private communication, 2006.

[5] Open Group/IEEE. The open group base specifications issue 6, ieee std 1003.1, 2004 edition. IEEE/1003.1 2004 Edition, The Open Group, 2004.

[6] Christof Pitter and Martin Schoeberl. Towards a Java multiprocessor. In *Proceedings of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2007)*, pages 144–151, Vienna, Austria, September 2007. ACM Press.

[7] Martin Schoeberl. *JOP: A Java Optimized Processor for Embedded Real-Time Systems*. PhD thesis, Vienna University of Technology, 2005.

[8] Fridtjof Siebert. Limits of parallel marking garbage collection. In *ISMM '08: Proceedings of the 7th international symposium on Memory management*, pages 21–29, New York, NY, USA, 2008. ACM.

[9] A.J. Wellings. Multiprocessors and the real-time specification for java. In *Proceedings of the 11th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing ISORC-2008*, pages 255–261. Computer Society, IEEE, IEEE, May 2008.