
**High-Performance and
Time-Predictable
Embedded Computing**

RIVER PUBLISHERS SERIES IN INFORMATION SCIENCE AND TECHNOLOGY

Series Editors

K. C. CHEN

*National Taiwan University
Taipei, Taiwan*

and

University of South Florida, USA

SANDEEP SHUKLA

*Virginia Tech
USA*

and

Indian Institute of Technology Kanpur, India

Indexing: All books published in this series are submitted to the Web of Science Book Citation Index (BkCI), to CrossRef and to Google Scholar.

The “River Publishers Series in Information Science and Technology” covers research which ushers the 21st Century into an Internet and multimedia era. Multimedia means the theory and application of filtering, coding, estimating, analyzing, detecting and recognizing, synthesizing, classifying, recording, and reproducing signals by digital and/or analog devices or techniques, while the scope of “signal” includes audio, video, speech, image, musical, multimedia, data/content, geophysical, sonar/radar, bio/medical, sensation, etc. Networking suggests transportation of such multimedia contents among nodes in communication and/or computer networks, to facilitate the ultimate Internet.

Theory, technologies, protocols and standards, applications/services, practice and implementation of wired/wireless networking are all within the scope of this series. Based on network and communication science, we further extend the scope for 21st Century life through the knowledge in robotics, machine learning, embedded systems, cognitive science, pattern recognition, quantum/biological/molecular computation and information processing, biology, ecology, social science and economics, user behaviors and interface, and applications to health and society advance.

Books published in the series include research monographs, edited volumes, handbooks and textbooks. The books provide professionals, researchers, educators, and advanced students in the field with an invaluable insight into the latest research and developments.

Topics covered in the series include, but are by no means restricted to the following:

- Communication/Computer Networking Technologies and Applications
- Queuing Theory
- Optimization
- Operation Research
- Stochastic Processes
- Information Theory
- Multimedia/Speech/Video Processing
- Computation and Information Processing
- Machine Intelligence
- Cognitive Science and Brain Science
- Embedded Systems
- Computer Architectures
- Reconfigurable Computing
- Cyber Security

For a list of other books in this series, visit www.riverpublishers.com

High-Performance and Time-Predictable Embedded Computing

Editors

Luís Miguel Pinho

CISTER Research Centre, Polytechnic Institute of Porto, Portugal

Eduardo Quiñones

Barcelona Supercomputing Center, Spain

Marko Bertogna

University of Modena and Reggio Emilia, Italy

Andrea Marongiu

Swiss Federal Institute of Technology Zurich, Switzerland

Vincent Nélis

CISTER Research Centre, Polytechnic Institute of Porto, Portugal

Paolo Gai

Evidence Srl, Italy

Juan Sancho

ATOS, Spain



River Publishers

Published, sold and distributed by:

River Publishers
Alsbjergvej 10
9260 Gistrup
Denmark

River Publishers
Lange Geer 44
2611 PW Delft
The Netherlands

Tel.: +45369953197
www.riverpublishers.com

ISBN: 978-87-93609-69-3 (Hardback)
978-87-93609-62-4 (Ebook)

©The Editor(s) (if applicable) and The Author(s) 2018. This book is published open access.

Open Access

This book is distributed under the terms of the Creative Commons Attribution-Non-Commercial 4.0 International License, CC-BY-NC 4.0) (<http://creativecommons.org/licenses/by/4.0/>), which permits use, duplication, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, a link is provided to the Creative Commons license and any changes made are indicated. The images or other third party material in this book are included in the work's Creative Commons license, unless indicated otherwise in the credit line; if such material is not included in the work's Creative Commons license and the respective action is not permitted by statutory regulation, users will need to obtain permission from the license holder to duplicate, adapt, or reproduce the material.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper.

Contents

Preface	xiii
List of Contributors	xv
List of Figures	xvii
List of Tables	xxi
List of Abbreviations	xxiii
1 Introduction	1
<i>Luís Miguel Pinho, Eduardo Quiñones, Marko Bertogna, Andrea Marongiu, Vincent Nélis, Paolo Gai and Juan Sancho</i>	
1.1 Introduction	1
1.1.1 The Convergence of High-performance and Embedded Computing Domains	3
1.1.2 Parallelization Challenge	5
1.2 The P-SOCRATES Project	6
1.3 Challenges Addressed in This Book	8
1.3.1 Compiler Analysis of Parallel Programs	8
1.3.2 Predictable Scheduling of Parallel Tasks on Many-core Systems	9
1.3.3 Methodology for Measurement-based Timing Analysis	9
1.3.4 Optimized OpenMP Tasking Runtime System	9
1.3.5 Real-time Operating Systems	10
1.4 The UpScale SDK	10
1.5 Summary	11
References	12

2	Manycore Platforms	15
	<i>Andrea Marongiu, Vincent Nélis and Patrick Meumeu Yomsi</i>	
2.1	Introduction	15
2.2	Manycore Architectures	17
2.2.1	Xeon Phi	17
2.2.2	Pezy SC	18
2.2.3	NVIDIA Tegra X1	19
2.2.4	Tilera Tile	21
2.2.5	STMicroelectronics STHORM	22
2.2.6	Epiphany-V	23
2.2.7	TI Keystone II	24
2.2.8	Kalray MPPA-256	25
	2.2.8.1 The I/O subsystem	26
	2.2.8.2 The Network-on-Chip (NoC)	26
	2.2.8.3 The Host-to-IOs communication protocol	28
	2.2.8.4 Internal architecture of the compute clusters	28
	2.2.8.5 The shared memory	29
2.3	Summary	30
	References	31
3	Predictable Parallel Programming with OpenMP	33
	<i>Maria A. Serrano, Sara Royuela, Andrea Marongiu and Eduardo Quiñones</i>	
3.1	Introduction	33
3.1.1	Introduction to Parallel Programming Models	34
	3.1.1.1 POSIX threads	35
	3.1.1.2 OpenCL™	35
	3.1.1.3 NVIDIA® CUDA	36
	3.1.1.4 Intel® Cilk™ Plus	36
	3.1.1.5 Intel® TBB	36
	3.1.1.6 OpenMP	37
3.2	The OpenMP Parallel Programming Model	37
3.2.1	Introduction and Evolution of OpenMP	37
3.2.2	Parallel Model of OpenMP	39
	3.2.2.1 Execution model	39
	3.2.2.2 Acceleration model	40
	3.2.2.3 Memory model	41

3.2.3	An OpenMP Example	42
3.3	Timing Properties of OpenMP Tasking Model	43
3.3.1	Sporadic DAG Scheduling Model of Parallel Applications	43
3.3.2	Understanding the OpenMP Tasking Model	44
3.3.3	OpenMP and Timing Predictability	46
3.3.3.1	Extracting the DAG of an OpenMP program	47
3.3.3.2	WCET analysis is applied to <i>tasks</i> and <i>tasks parts</i>	48
3.3.3.3	DAG-based scheduling must not violate the TSCs	49
3.4	Extracting the Timing Information of an OpenMP Program	51
3.4.1	Parallel Structure Stage	52
3.4.1.1	Parallel control flow analysis	53
3.4.1.2	Induction variables analysis	53
3.4.1.3	Reaching definitions and range analysis	53
3.4.1.4	Putting all together: The wave-front example	53
3.4.2	Task Expansion Stage	54
3.4.2.1	Control flow expansion and synchronization predicate resolution	54
3.4.2.2	t_{id} : A unique task instance identifier	56
3.4.2.3	Missing information when deriving the DAG	57
3.4.3	Compiler Complexity	58
3.5	Summary	58
	References	59

4 Mapping, Scheduling, and Schedulability Analysis 63

Paolo Burgio, Marko Bertogna, Alessandra Melani, Eduardo Quiñones and Maria A. Serrano

4.1	Introduction	63
4.2	System Model	64
4.3	Partitioned Scheduler	66
4.3.1	The Optimality of EDF on Preemptive Uniprocessors	66
4.3.2	FP-scheduling Algorithms	67
4.3.3	Limited Preemption Scheduling	68

4.3.4	Limited Preemption Schedulability Analysis	69
4.4	Global Scheduler with Migration Support	70
4.4.1	Migration-based Scheduler	70
4.4.2	Putting All Together	72
4.4.3	Implementation of a Limited Preemption Scheduler	73
4.5	Overall Schedulability Analysis	75
4.5.1	Model Formalization	75
4.5.2	Critical Interference of cp-tasks	78
4.5.3	Response Time Analysis	80
4.5.3.1	Inter-task interference	80
4.5.3.2	Intra-task interference	82
4.5.3.3	Computation of cp-task parameters	84
4.5.4	Non-conditional DAG Tasks	86
4.5.5	Series-Parallel Conditional DAG Tasks	86
4.5.6	Schedulability Condition	86
4.6	Specializing Analysis for Limited Pre-emption Global/ Dynamic Approach	87
4.6.1	Blocking Impact of the Largest NPRs (LP-max)	88
4.6.2	Blocking Impact of the Largest Parallel NPRs (LP-ILP)	88
4.6.2.1	LP worst-case workload of a task executing on c cores	89
4.6.2.2	Overall LP worst-case workload	90
4.6.2.3	Lower-priority interference	91
4.6.3	Computation of Response Time Factors of LP-ILP	92
4.6.3.1	Worst-case workload of τ_i executing on c cores: $\mu_i[c]$	92
4.6.3.2	Overall LP worst-case workload of $lp(k)$ per execution scenario s_1 : $\rho_k[s_1]$	94
4.6.4	Complexity	95
4.7	Specializing Analysis for the Partitioned/Static Approach	96
4.7.1	ILP Formulation	96
4.7.1.1	Tied tasks	97
4.7.1.2	Untied tasks	99
4.7.1.3	Complexity	100
4.7.2	Heuristic Approaches	100
4.7.2.1	Tied tasks	101

4.7.2.2	Untied tasks	103
4.7.3	Integrating Interference from Additional RT Tasks	103
4.7.4	Critical Instant	104
4.7.5	Response-time Upper Bound	105
4.8	Scheduling for I/O Cores	107
4.9	Summary	107
	References	109

5 Timing Analysis Methodology 113

Vincent Nélis, Patrick Meumeu Yomsis and Luís Miguel Pinho

5.1	Introduction	113
5.1.1	Static WCET Analysis Techniques	115
5.1.2	Measurement-based WCET Analysis Techniques	118
5.1.3	Hybrid WCET Techniques	119
5.1.4	Measurement-based Probabilistic Techniques	120
5.2	Our Choice of Methodology for WCET Estimation	121
5.2.1	Why Not Use Static Approaches?	122
5.2.2	Why Use Measurement-based Techniques?	124
5.3	Description of Our Timing Analysis Methodology	127
5.3.1	Intrinsic vs. Extrinsic Execution Times	127
5.3.2	The Concept of Safety Margins	128
5.3.3	Our Proposed Timing Methodology at a Glance	130
5.3.4	Overview of the Application Structure	131
5.3.5	Automatic Insertion and Removal of the Trace-points	133
5.3.5.1	How to insert the trace-points	133
5.3.5.2	How to remove the trace-points	135
5.3.6	Extract the Intrinsic Execution Time: The Isolation Mode	136
5.3.7	Extract the Extrinsic Execution Time: The Contention Mode	137
5.3.8	Extract the Execution Time in Real Situation: The Deployment Mode	141
5.3.9	Derive WCET Estimates	141
5.4	Summary	143
	References	143

6 OpenMP Runtime 145

Andrea Marongiu, Giuseppe Tagliavini and Eduardo Quiñones

6.1	Introduction	145
6.2	Offloading Library Design	146
6.3	Tasking Runtime	148
6.3.1	Task Dependency Management	155
6.4	Experimental Results	158
6.4.1	Offloading Library	159
6.4.2	Tasking Runtime	160
6.4.2.1	Applications with a linear generation pattern	160
6.4.2.2	Applications with a recursive generation pattern	162
6.4.2.3	Applications with mixed patterns	163
6.4.2.4	Impact of cutoff on LINEAR and RECURSIVE applications	165
6.4.2.5	Real applications	166
6.4.3	Evaluation of the Task Dependency Mechanism	167
6.4.3.1	Performance speedup and memory usage	168
6.4.3.2	The task dependency mechanism on the MPPA	170
6.5	Summary	171
	References	171

7 Embedded Operating Systems 173

*Claudio Scordino, Errico Guidieri, Bruno Morelli,
Andrea Marongiu, Giuseppe Tagliavini and Paolo Gai*

7.1	Introduction	173
7.2	State of The Art	175
7.2.1	Real-time Support in Linux	175
7.2.1.1	Hard real-time support	176
7.2.1.2	Latency reduction	178
7.2.1.3	Real-time CPU scheduling	180
7.2.2	Survey of Existing Embedded RTOSs	180
7.2.3	Classification of Embedded RTOSs	186
7.3	Requirements for The Choice of The Run Time System	187
7.3.1	Programming Model	187
7.3.2	Preemption Support	187

7.3.3	Migration Support	188
7.3.4	Scheduling Characteristics	188
7.3.5	Timing Analysis	188
7.4	RTOS Selection	190
7.4.1	Host Processor	190
7.4.2	Manycore Processor	190
7.5	Operating System Support	191
7.5.1	Linux	191
7.5.2	ERIKA Enterprise Support	191
7.5.2.1	Exokernel support	191
7.5.2.2	Single-ELF multicore ERIKA Enterprise	192
7.5.2.3	Support for limited preemption, job, and global scheduling	192
7.5.2.4	New ERIKA Enterprise primitives	193
7.5.2.5	New data structures	194
7.5.2.6	Dynamic task creation	196
7.5.2.7	IRQ handlers as tasks	196
7.5.2.8	File hierarchy	197
7.5.2.9	Early performance estimation	197
7.6	Summary	200
	References	200

Index **203**

About the Editors **205**

Preface

Nowadays, the prevalence of electronic and computing systems in our lives is so ubiquitous that it would not be far-fetched to state that we live in a cyber-physical world dominated by computer systems. Examples include pacemakers implanted within the human body to regulate and monitor heartbeats, cars and airplanes transporting us, smart grids, and traffic management.

All these systems demand more and more computational performance to process large amounts of data from multiple data sources, and some of them with guaranteed processing response times; in other words, systems required to deliver their results within pre-defined (and sometimes extremely short) time bounds. This timing aspect is vital for systems like planes, cars, business monitoring, e-trading, etc. Examples can be found in intelligent transportation systems for fuel consumption reduction in cities or railways, or autonomous driving of vehicles. All these systems require processing and actuation based on large amounts of data coming from real-time sensor information.

As a result, the computer electronic devices which these systems depend on are constantly required to become more and more powerful and reliable, while remaining affordable. In order to cope with such performance requirements, chip designers have recently started producing chips containing multiple processing units, the so-called multi-core processors, effectively integrating multiple computers within a single chip, and more recently the many-core processors, with dozens or hundreds of cores, interconnected with complex networks on chip. This radical shift in the chip design paved the way for parallel computing: rather than processing the data sequentially, the cooperation of multiple processing elements within the same chip allows systems to be executed concurrently, in parallel.

Unfortunately, the parallelization of the computing activities brought up many challenges, because it affects the timing behavior of the systems as well as the entire way people think and design computers: from the design of the hardware architecture, through the operating system up to the conceptualization of the end-user application. Therefore, although many-core processors are promising candidates to improve the responsiveness of these systems,

the interactions that the different computing elements may have within the chip can seriously affect the performance opportunities brought by parallel execution. Moreover, providing timing guarantees becomes harder, because the timing behavior of the system running within a many-core processor depends on interactions that are most of the time not known by the system designer. This makes system analysts struggle in trying to provide timing guarantees for such platforms. Finally, most of the optimization mechanisms buried deep inside the chip are geared only to increase performance and execution speed rather than providing predictable time behavior.

These challenges need to be addressed by introducing predictability in the vertical stack from high-level programming models to operating systems, together with new offline analysis techniques. This book covers the main techniques to enable performance and predictability of embedded applications. The book starts with an overview of some of the current many-core embedded platforms, and then addresses how to support predictability and performance in different aspects of computation: a predictable parallel programming model, the mapping and scheduling of real-time parallel computation, the timing analysis of parallel code, as well as the techniques to support predictability in parallel runtimes and operating systems.

The work reflected in this book was done in the scope of the European project P-SOCRATES, funded under the FP7 framework program of the European Commission. The project website (www.p-socrates.eu), provides further detailed information on the techniques presented here. Moreover, a reference implementation of the methodologies and tools was released as the UpScale Software Development Kit (<http://www.upscale-sdk.com>).

Luís Miguel Pinho
Eduardo Quiñones
Marko Bertogna
Andrea Marongiu
Vincent Nélis
Paolo Gai
Juan Sancho

February 2018

List of Contributors

Alessandra Melani, *University of Modena and Reggio Emilia, Italy*

Andrea Marongiu, *Swiss Federal Institute of Technology in Zürich (ETHZ), Switzerland; and University of Bologna, Italy*

Bruno Morelli, *Evidence SRL, Italy*

Claudio Scordino, *Evidence SRL, Italy*

Eduardo Quiñones, *Barcelona Supercomputing Center (BSC), Spain*

Errico Guidieri, *Evidence SRL, Italy*

Giuseppe Tagliavini, *University of Bologna, Italy*

Juan Sancho, *ATOS, Spain*

Luís Miguel Pinho, *CISTER Research Centre, Polytechnic Institute of Porto, Portugal*

Maria A. Serrano, *Barcelona Supercomputing Center (BSC), Spain*

Marko Bertogna, *University of Modena and Reggio Emilia, Italy*

Paolo Burgio, *University of Modena and Reggio Emilia, Italy*

Paolo Gai, *Evidence SRL, Italy*

Patrick Meumeu Yomsi, *CISTER Research Centre, Polytechnic Institute of Porto, Portugal*

Sara Royuela, *Barcelona Supercomputing Center (BSC), Spain*

Vincent Nélis, *CISTER Research Centre, Polytechnic Institute of Porto, Portugal*

List of Figures

Figure 1.1	P-SOCRATES Global perspective.	7
Figure 1.2	P-SOCRATES combines high-performance parallel programming models, high-end embedded many-core platforms and real-time systems technology.	7
Figure 1.3	Vertical stack of application decomposition.	8
Figure 1.4	The UpScale SDK.	11
Figure 2.1	Knights Landing (KNL) block diagram: (a) the CPU, (b) an example tile, and (c) KNL with Omni-Path Fabric integrated on the CPU package.	17
Figure 2.2	PEZY-SC architecture block diagram.	19
Figure 2.3	NVIDIA Tegra X1 block diagram.	20
Figure 2.4	Tilera <i>Tile</i> architectural template.	21
Figure 2.5	STMicroelectronics STHORM heterogeneous system.	22
Figure 2.6	Block diagram of the Epiphany-V chip from Adapteva.	23
Figure 2.7	Texas Instrument Keystone II heterogeneous system.	24
Figure 2.8	High-level view of the Kalray MPPA-256 processor.	25
Figure 2.9	MPPA-256 NoC architecture.	27
Figure 2.10	A master task runs on an RM of an I/O subsystem.	28
Figure 2.11	Internal architecture of a compute cluster.	29
Figure 2.12	Memory accesses distributed across memory banks (interleaved).	30
Figure 2.13	Memory accesses targeting a same memory bank (contiguous).	31
Figure 3.1	OpenMP components stack.	38

Figure 3.2	OpenMP releases time-line.	39
Figure 3.3	Structured parallelism.	40
Figure 3.4	Unstructured parallelism.	40
Figure 3.5	OpenMP-DAG composed of <i>task parts</i> based on the code.	48
Figure 3.6	DAG composed on task region parts.	50
Figure 3.7	aDAG of the OpenMP program.	55
Figure 3.8	The DAG of the OpenMP program	56
Figure 4.1	An application is composed of multiple real-time tasks.	65
Figure 4.2	RT tasks are mapped to OS threads, which are scheduled on the processing elements.	65
Figure 4.3	Fully preemptive vs. non-preemptive scheduling: preemption overhead and blocking delay may cause deadline misses.	69
Figure 4.4	A sample cp-task. Each vertex is labeled with the WCET of the corresponding sub-task.	76
Figure 4.5	Work-case scenario to maximize the workload of an task τ_i , in the sequential case.	81
Figure 4.6	Worst-case scenario to maximize the workload of an interfering cp-task τ_i	82
Figure 4.7	DAGs of lp(k) tasks; the $C_{i,j}$ of each node $v_{i,j}$ is presented in parenthesis.	89
Figure 4.8	Tasks example.	104
Figure 4.9	Different release patterns for the example of Figure 4.8. (a) represents the most optimistic case, while (c) the most pessimistic, i.e., yielding to the highest WCET. (b) represents an intermediate case.	105
Figure 5.1	Example distribution of execution time.	115
Figure 5.2	Extended task dependency graph (eTDG) of an example application.	133
Figure 5.3	Impact of an unused variable on the execution time of an example application.	135
Figure 6.1	Timing diagram of an offloading procedure.	147
Figure 6.2	Task suspension in the baseline implementation (considering tied tasks and WFS).	151
Figure 6.3	Untied task suspension with task contexts and per-task stacks.	152

Figure 6.4	On the left (a), the DAG of an OpenMP program. On the right (b), the sparse matrix data structure implementing DAG shown on the left.	157
Figure 6.5	Costs of offload initialization.	159
Figure 6.6	Speedup of the LINEAR benchmark (no cutoff). . .	161
Figure 6.7	Speedup of the RECURSIVE benchmark (no cutoff).	163
Figure 6.8	Structure of the MIXED microbenchmark.	164
Figure 6.9	Speedup of the MIXED benchmark.	164
Figure 6.10	Speedup of the LINEAR benchmark (with cutoff).	165
Figure 6.11	Speedup of the RECURSIVE benchmark (with cutoff).	166
Figure 6.12	Speedups for the BOTS benchmarks.	167
Figure 6.13	Performance speedup of the Cholesky (a) and r3DPP (b) running with <i>lightweight omp4</i> , <i>omp4</i> , and <i>omp 3.1</i> , and varying the number of tasks. . . .	168
Figure 6.14	Memory usage (in KB) of the Cholesky (a) r3DPP (b) running with <i>lightweight omp4</i> , <i>omp4</i> , and <i>omp 3.1</i> , and varying the number of tasks.	169
Figure 6.15	Performance speedup of the Cholesky (a) and r3DPP (b) running on the MPPA with <i>lightweight omp4</i> , <i>omp4</i> , and <i>omp 3.1</i> , and varying the number of tasks.	170
Figure 7.1	Number of Linux-based supercomputers in the TOP500 list.	174
Figure 7.2	Structure of the multicore images in the original ERIKA Enterprise structure.	195
Figure 7.3	Structure of the Single-ELF image produced by ERIKA Enterprise.	196

List of Tables

Table 3.1	Parallel programming models comparison	35
Table 4.1	Worst-case workloads of tasks in Figure 4.7	90
Table 4.2	Five possible scenarios of taskset in Figure 4.7, assuming a four core system	91
Table 4.3	Computed worst-case workload for each of the scenarios in Table 4.2	91
Table 6.1	Memory usage of the sparse matrix (in KB), varying the number of tasks instantiated	170
Table 7.1	Classification of RTOSs	186
Table 7.2	ERIKA Enterprise footprint (expressed in bytes) . .	198
Table 7.3	Timings (expressed in clock ticks)	199
Table 7.4	Footprint comparison between ERIKA and NodeOS for a 16-core cluster (expressed in bytes)	199
Table 7.5	Thread creation/activation times (expressed in clock ticks)	199

List of Abbreviations

ADEOS	Adaptive Domain Environment for Operating Systems, a patch used by RTAI and Xenomai
ALU	Arithmetic logic unit
API	Application Programming Interface
APIC	Programmable timer on x86 machines
AUTOSAR	International consortium that defines automotive API
BCET	Best-Case Execution Time
BFS	Breadth First Scheduling
BOTS	Barcelona OpenMP task suite
BSD	Berkeley software license
CBS	Constant Bandwidth Server, a scheduling algorithm
CFG	Control Flow Graph
CMP	chip multi-processor
COTS	Commercial Off-The-Shelf
CPU	central processing unit
CUDA	Compute Unified Device Architecture
DAG	Direct Acyclic Graph
DDR	double data rate
DMA	direct memory access (engine)
DRAM	dynamic random-access memory
DSP	digital signal processor
DSU	debug system unit
EDF	Earliest Deadline First Scheduler, a scheduling algorithm
ELF	binary format that contain executables
eTDC	Extended Task Dependency Graph
EVT	Extreme Value Theory
FIFO	First-In First-Out
FLOPS	floating-point operations per second
FPGA	Field-Programmable Gate Array
GPGPU	General Purpose Graphics Processing Unit
GPL	General Public License

GPOS	General purpose Operating System
GPU	Graphics Processing Unit
GRUB	Greedy Reclamation of Unused Bandwidth, a scheduling algorithm
HAL	Hardware Abstraction Layer
HMI	Human Machine Interface
HPC	High Performance Computing
HRT	High Resolution Timers
ID	Identifier
IEC	International Electrotechnical Commission
IG	Interference Generator
IID	Independent and Identically Distributed
ILP	instruction-level parallelism
IRQ	Hardware Interrupt
ISA	Instruction Set Architecture
LHC	Large Hadron Collider
LIFO	Last-In First-Out
LL-RTE	Low-level runtime environment
LRU	Least recently used
MBPTA	Measurement-Based Probabilistic Timing Analysis
MCU	Microcontroller Unit
MEET	Maximum Extrinsic Execution Time
MIET	Maximum Intrinsic Execution Time
MPPA	Multi Purpose Processor Array
NOC	network-on-chip
NOP	No Operation
NUCA	non-uniform cache architecture
OpenCL	Open Computing Language
OpenMP	Open multi processing (programming model)
OpenMP DAG	OpenMP Direct Acyclic Graph
OS	Operating system
PCFG	Parallel Control Flow Graph
PCIe	peripheral component interconnect express
PE	processing element
PLRU	Pseudo-LRU
POSIX	Portable Operating System Interface for UNIX
PPC	PowerPC
P-SOCRATES	Parallel Software Framework for Time-Critical Many-core Systems

pWCET	Probabilistic Worst-Case Execution Time
RCU	Linux Read-Copy-Update technique
RM	Resource manager
RT	Real time
RT task	Real-Time task
RTE	Runtime environment
RTOS	Real time operative system
SDK	Software Development Kit
SMP	Symmetric Multi Processor
SMT	simultaneous multi-threading
SoC	System-on-Chip
SOM	system-on-module
SP	Stack pointer
TBB	Thread Building Blocks
TC	Task context
TDG	Task Dependency Graph
TDMA	Time Division Multiple Access
TLB	translation lookaside buffer
TLP	Thread-Level Parallelism
TSC	Task Scheduling Constraint
TSP	Task Scheduling Point
VLIW	Very Large Instruction Word
VPU	vector processing unit
WCET	Worst Case Execution Time
WFS	Work First Scheduling

