Computer Architecture and Design Methodologies

Andreas Weichslgartner
Stefan Wildermann
Michael Glaß
Jürgen Teich

Invasive Computing for Mapping Parallel Programs to Many-Core Architectures

Springer

Computer Architecture and Design Methodologies

Twilight zone of Moore's law is affecting computer architecture design like never before. The strongest impact on computer architecture is perhaps the move from unicore to multicore architectures, represented by commodity architectures like general purpose graphics processing units (gpgpus). Besides that, deep impact of application-specific constraints from emerging embedded applications is presenting designers with new, energy-efficient architectures like heterogeneous multi-core, accelerator-rich System-on-Chip (SoC). These effects together with the security, reliability, thermal and manufacturability challenges of nanoscale technologies are forcing computing platforms to move towards innovative solutions. Finally, the emergence of technologies beyond conventional charge-based computing has led to a series of radical new architectures and design methodologies.

The aim of this book series is to capture these diverse, emerging architectural innovations as well as the corresponding design methodologies. The scope will cover the following.

Heterogeneous multi-core SoC and their design methodology

Domain-specific Architectures and their design methodology

Novel Technology constraints, such as security, fault-tolerance and their impact on architecture design

Novel technologies, such as resistive memory, and their impact on architecture design

Extremely parallel architectures

More information about this series at http://www.springer.com/series/15213

Andreas Weichslgartner · Stefan Wildermann
Michael Glaß · Jürgen Teich

Invasive Computing for Mapping Parallel Programs to Many-Core Architectures

Springer

Andreas Weichslgartner
Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Erlangen, Bayern
Germany

Stefan Wildermann
Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Erlangen, Bayern
Germany

Michael Glaß
Embedded Systems/Real-Time Systems
University of Ulm
Ulm, Baden-Württemberg
Germany

Jürgen Teich
Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Erlangen, Bayern
Germany

ISSN 2367-3478 ISSN 2367-3486 (electronic)
Computer Architecture and Design Methodologies
ISBN 978-981-10-7355-7 ISBN 978-981-10-7356-4 (eBook)
https://doi.org/10.1007/978-981-10-7356-4

Library of Congress Control Number: 2017958628

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Acknowledgements

This work originated from within the Transregional Collaborative Research Center 89 “Invasive Computing” (abbr. InvasIC) in which a novel paradigm for the design and resource-aware programming of future parallel computing systems is investigated. For systems with 1000 and more cores on a chip, resource-aware programming is of utmost importance to obtain high utilization as well as high computational and energy efficiency, but also in order to achieve predictable qualities of execution of parallel programs. The basic principle of invasive computing and innovation is to give a programmer explicit handles to specify and argue about resource requirements desired or required in different phases of execution.

InvasIC is funded by the Deutsche Forschungsgemeinschaft (DFG), aggregating researchers from Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Karlsruher Institut für Technologie (KIT), and Technische Universität München (TUM). Its scientific team includes specialists in parallel algorithm design, hardware architects for reconfigurable MPSoC development as well as language, tool, application, and operating system designers.

At this point, we like to thank all participating scientists of InvasIC who enabled and jointly contributed to the achievements of InvasIC in general and to the results summarized in this book in particular. Our particular thanks go to the DFG for funding InvasIC.

Contents

Abbreviations

2D	Two dimensional, 25, 28, 47, 137, 138, 141, 158
AG	Address generator, 26
AHB	Advanced high-performance bus, 31
AMBA	Advanced microcontroller bus architecture, 31
API	Application programming interface, 38
BCET	Best-case execution time, 46, 47, 51, 52
BE	Best effort, 28, 29, 30, 32
BN	Best neighbor, 65, 67, 76
bps	Bits per second, 17
BRAM	Block random access memory, 79
CA	Cluster agent, 80
CAP	Communication-aware programming, 13, 14, 19, 57, 157
CDF	Cumulative distribution function, 123
C*i*C	Dynamic many-core *i*-let controller, 27, 34
CPU	Computer processing unit, 20, 23, 35, 51
CSP	Constraint satisfaction problem, 105
DAARM	Design-time application analysis and run-time mapping, 4, 86, 88, 132, 158
DMA	Direct memory access, 13, 31
DOP	Degree of parallelism, 11, 20, 21, 22, 23
DSE	Design space exploration, 4, 18, 35, 39, 68, 85
E3S	Embedded system synthesis benchmarks suite, 75, 111, 114, 120, 148
EA	Evolutionary algorithm, 96, 120, 127
FCFS	First-come, first-served, 50
FF	First free, 50
FIFO	First in, first out, 31, 32, 72, 79
flit	Flow control digit, 29, 31, 32, 49
FPGA	Field-programmable gate array, 27, 35, 79
fps	Frames per second, 16

FSM	Final state machine, 17, 33
GA	Global agent, 80
GC	Global controller, 26
GPU	Graphics processing unit, 35
GS	Guaranteed service, 28, 29, 30, 31, 49
HAL	Hardware abstraction layer, 34
HAM	Hybrid application mapping, 39, 45, 49, 59, 85, 86, 87, 88, 127, 128, 129, 132, 141, 148
HPC	High-performance computing, 1, 24
HW	Hardware, 35
I/O	Input/Output, 25, 26, 139
ID	Identifier, 33, 70, 72
ILP	Integer linear programming, 127
IM	Invasion manager, 26
i-Core	Invasive core, 24, 27, 39
i-NA	Invasive network adapter, 13, 25, 26, 28, 29, 30, 31, 32, 33, 79, 81
i-NoC	Invasive network on chip, 3, 5, 9, 19, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33 34, 35, 36, 37, 38, 39, 47, 59, 68, 70, 73
*i*RTSS	Invasive run-time system, 18, 34
L1	Level 1, 24, 51
L2	Level 2, 24
LRU	Least recently used, 51, 55
LUT	Lookup table, 79, 154
MAC	Minimal average channel load, 64, 80
MIT	Massachusetts Institute of Technology, 52
MMC	Minimum maximum channel load, 64
MMKP	Multi-dimensional multiple choice knapsack problem, 107, 109, 110, 111, 114, 115, 117, 119
MPSoC	Multiprocessor system-on-chip, 17, 21, 22, 23, 24, 52, 138, 154
MTTF	Mean time to failure, 160
NA	Network adapter, 47, 48, 49
NN	Nearest neighbor, 80
NoC	Network on chip, 3, 4, 5, 9, 14, 28, 33, 47, 48, 49, 57, 58, 59, 64
OP	Operating point, 19, 89, 101, 112, 113, 114, 120, 158
OpenMP	Open multi-processing, 11
OS	Operating system, 9, 12, 26, 27, 28, 31, 34, 59, 95
OSAL	Operating system abstraction layer, 34
P2P	Point to point, 86, 88, 117, 129
PE	Processing element, 26
PFH	Probability of failure per hour, 16, 19
PGAS	Partitioned global address space, 11, 17, 40
PiP	Picture in picture, 14

PL	Path load, 64, 65, 76, 81
QoS	Quality-of-service, 28, 31, 48, 89, 92, 128, 131
QSDPCM	Quad-tree structured differential pulse code modulation, 75
RAM	Random access memory, 141
RANSAC	Random sample consensus, 125
RISC	Reduced instruction set computing, 14, 19, 27, 57, 58
RM	Run-time management, 5, 32, 85, 86, 87, 90, 101, 102, 106, 112, 113, 114, 122, 127, 128, 133, 141, 145, 147,158
RR	Round robin, 50
RTC	Real-time calculus, 103
RW	Random walk, 65, 77, 78
RWW	Random walk weighted, 65, 66, 67, 77
SA	Simulated annealing, 127
SAT	Boolean satisfiability problem, 137, 138, 147, 148, 149, 150, 151, 154
SEM	Self-embedding module, 72, 73, 74, 79, 80
SIFT	Scale-invariant feature transform, 125, 126
SIL	Safety integrity level, 216
SL	Service level, 19, 28, 31
TCB	Trusted computing base, 141, 154
TCPA	Tightly coupled processor array, 25, 26, 125
TDM	Time division multiplexing, 50, 52, 131
TDMA	Time division multiple access, 55
TGFF	Task graphs for free, 55, 92, 128, 129, 131, 132, 139
TLM	Tile local memory, 13, 14, 24, 31, 50
VC	Virtual channel, 28, 30, 31, 32, 101, 102, 103, 141
VLIW	Very large instruction word, 25
WCET	Worst-case execution time, 47, 51, 52, 94, 99, 111
WCRT	Worst-case response time, 50
WRR	Weighted round robin, 28, 52, 92, 129, 132
XML	Extensible markup language, 36

Symbols

A	Attacker, 17
$\mathcal{A}$	Variable assignment in CSP, 105
α_i	Underutilization factor, 21
avg_{net}	Average network load, 75
B	Communication channel, 102, 105, 141, 148
B_{E}	Best-case execution time, 46
β	Map task, 55, 59, 60, 64, 65, 67, 91, 92, 95, 100, 102, 104, 107, 111, 113, 120, 142
β_{CG}	Map task cluster of constraint graph function, 103, 105
β_{DSE}	Map task function in DSE, 102
bw	Minimum required message bandwidth, 46, 47, 49, 59, 64, 71, 98
C	Task cluster, 101, 102, 103, 104, 105, 112, 141
c	Cost function for self-embedding algorithm, 60, 62
cap	Link capacity, 49, 59, 66, 98
CL	Worst-case communication latency, 93
$\mathcal{C}onf$	Confidentiality, 17
D	Domain in CSP, 106
δ	Deadline, 46, 91
Δc	Number of cores to invade, 23
E	Set of edges of an application graph, 46, 47, 49, 50
e	Edge of an application graph, 46
E_{CPU}	Overall maximal processor energy consumption of a mapping, 99
E_{inc}	Energy consumption of all mapped operating points by incremental RM, 114
embAlg	Embedding algorithm, 60
E_{MMKP}	Energy consumption of all mapped operating points by MMKP RM, 114
E_{NOC}	Overall maximal NoC energy consumption of a mapping, 99

Env	Environment, 17
E_{OV}	Overall maximal energy consumption of a mapping, 90, 99, 107, 109
$\in Conf$	$\in$-confidentiality, 17
equaltype	Checks if the resource type of a tile matches a certain resource type, 101
E_{rel}	Relative energy consumption of MMKP RM and incremental RM, 114
E_{Lbit}	Energy consumption of one bit in a NoC link, 99, 112
E_{Sbit}	Energy consumption of one bit in a NoC router, 99, 112
E_{NoCbit}	Energy consumption of routing one bit over a NoC router, 99
η_R	NoC router delay, 93
f	Frequency, 49, 69, 125
$G_{APP'}(V,E)$	Example application graph, 46, 47, 50
G_{Arch}	Short notation of architecture graph, 106
$G_{Arch'}(U,L)$	Example architecture graph, 48
$G_{\mathrm{Arch}}(U,L)$	Architecture graph, 47, 50, 91, 102, 141
$G_{\mathrm{App}}(V,E)$	Application graph, 46, 49, 71, 91, 103, 141
$G_{\mathrm{App}}(V = T \cup M, E)$	Application graph, 70, 71
G_C	Short notation of constraint graph, 106
$G_C(V_C,E_C)$	Constraint graph, 101, 102, 141
gettype	Determines the resource type of a tile: $U \rightarrow R$, 47, 49, 50, 94, 96, 98, 99, 101, 144
hop	Hop constraint in the constraint graph, 102, 103
H^+	Hop distance, 48, 49, 93, 102, 104
H	Manhattan distance, 48, 49, 50, 66, 67, 93, 99, 102, 104
h	Max hop distance in self-embedding algorithm, 60, 61, 62, 64, 65, 66, 76
H_ρ^+	Hop distance of a route, 93, 99
H_ρ	Manhattan distance of a route, 50, 93, 99
I	Input space, 53
i	Running variable, 18, 19, 20, 66
$\Im\mathfrak{E}$	Invasive efficiency, 22, 23
INF	Infinum, 53
$INF_{\mathcal{L}}$	Best-case end-to-end latency, 125,
INF_{LComp}	Best-case tile latency, 125
INF_{LNoC}	Best-case NoC latency, 125
INF_{TrNoC}	Best-case NoC throughput, 126
$\Im\mathfrak{P}$	Average number of processors utilized, 22, 23
$\Im\mathfrak{S}$	Invasive speedup, 22, 23
isrouted	Function evaluates whether a message is routed over the NoC or utilizes local tile communication, 100
$\Im\mathfrak{T}$	Invasive execution time, 22, 23

j	Running variable, 114
k	Number of bits of a head flit of a serialized task graph, 71
$K_{\max}$	Maximum number of tasks for schedule, 94, 95, 96, 102, 104, 105, 120
$\mathfrak{Q}$	Worst-case end-to-end latency of a path, 92
$\mathcal{L}$	Worst-case end-to-end latency of an application, 90, 91, 98, 125, 126
L	Set of NoC links, 48, 49, 50, 64, 75, 102, 105
l	NoC link, 48, 49, 50, 59, 60, 64, 75, 98, 143
λ	Lagrangian multipliers, 109, 111
load	Load induced of a task onto a tile, 59, 66, 70, 102, 104
LW	Link width, 49, 72, 125
M	Set of messages of an application graph, 46, 48, 60, 61, 63, 70, 71, 93, 97, 98, 100, 147
$\mathfrak{m}$	Degree of parallelism, 20, 21
m	Message of an application graph, 46, 47, 48, 60, 63, 64, 65, 70, 76,82, 92, 100, 122, 124, 147, 148
$\mathcal{M}_C$	Communication channels, 102, 103, 106, 141, 148
n	Number of cores, 201, 21, 22
n	Number of applications, 106, 107, 108, 111, 118
n_f	Number of flits, 93
o	Objective, 53, 54, 99, 100,138, 139, 143, 144, 149
obj	Number of detected objective in Harris Corner algorithm, 125
O_υ	Invasive overhead function, 21
$O_{\upsilon+}$	Invasive overhead function for invade and infect, 22
$O_{\upsilon-}$	Invasive overhead function for retreat and infect, 22
O_{infect}	Invasive overhead function for infect, 23
O_{invade}	Invasive overhead function for invade, 23
O_{retreat}	Invasive overhead function for retreat, 23
P	Period of the application, 46, 49, 98, 102
p	Program, 52, 53
$path$	Path in mapping, 91
pathLoad	Cost function for self-embedding which evaluates a NoC path, 83
$paths$	Set of paths, 91
power	Power function returns the power consumption of a processing core, 99
pr	Priority of a task, 94, 95, 96, 101
pred	Function which returns the predecessor vertex in a graph, 62
Pred_T	Function which returns the predecessor tasks, 95, 106
$\langle \text{pr(t)}, \forall t \in C \rangle$	List of priorities, 102
$\mathcal{P}_{S_i}$	Set of incarnations of a shape of application i, 147

$\mathcal{P}_S$	Set of incarnations of a shape, 144, 146, 148
P_S	Incarnation of a shape, 145, 147
$\mathbf{P}_S$	SAT variable for shape, 147
P_S^1	Incarnation of shape, 142
P_S^2	Incarnation of shape, 142
Ψ	Mapping step in self-embedding algorithm, 62
Q	State space, 53, 54
R	Set of resource types, 47, 48, 101, 102, 108, 109, 110, 102, 125
r	Resource type, 47, 48, 49, 50, 89, 90, 98, 99, 100, 102, 125, 142, 180
$\text{rate}_{\text{link}}$	Link utilization, 64, 75
req	Task memory requirements, 59, 70
res	Tile memory resources, 57
ρ	Routing of a message, 52, 59, 64, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 106, 107, 108, 111, 143, 145, 150
ρ_{CG}	Routing of a message cluster of a constraint graph, 102, 104, 105
$\mathfrak{S}$	Speedup, 20
$\mathcal{S}$	Set of shapes, 142, 144, 145, 147
S	Shape, 142, 143, 144, 145, 146, 147, 148
SI	Scheduling interval, 94, 95, 96, 98, 102, 120
SI_{os}	Scheduling interval os overhead, 94, 95, 96, 98, 102, 120
size	Message size 46, 71, 99
SL	Service level in the NoC, 49, 52, 70, 76, 84, 96, 110, 116, 120, 128, 129, 130, 131, 132, 140, 155, 180, 190
SL_{max}	Number of scheduling intervals on the NoC, 52, 43, 44, 67, 68, 94, 97, 80, 115, 119, 120, 124, 125, 126, 128, 129, 130, 131, 154, 155, 175
Soft	Software, 17
succ	Successor function returns all successors of a vertex, 60, 71, 106
SUP	Supremum, 53
$SUP_{\mathcal{L}}$	Worst-case end-to-end latency, 114
SUP_{LComp}	Worst-case tile latency, 125
SUP_{LNoC}	Worst-case NoC latency, 125
SUP_{TrNoC}	Worst-case NoC throughput, 125
$\mathfrak{T}$	Execution time, 20, 21, 23
T	Set of tasks of an application graph, 46, 47, 50, 59, 60,62, 66, 71, 79, 91, 98, 99, 100, 101, 102, 143, 17
t	Task of an application graph, 64, 65, 68, 69, 81, 82, 83, 84, 86, 87, 89, 98, 93, 94, 96, 97, 115, 116, 117, 118, 121, 122, 123, 124, 125, 126, 128, 129, 132, 154, 177, 179

τ	Cycle length, 50, 94
$\mathcal{T}_C$	Task clusters, 102, 103, 104, 105, 141
θ_r	Instances of resource type available, 108, 109, 110
TL	Worst-case computing latency, 91, 94, 95, 96
tr	Throughput, 49
type	Resource type function: $T \longrightarrow R$, 47, 71, 91
type_{CG}	Type constraint of a constraint graph, 102, 116
U	Set of tiles of an architecture graph, 48, 49, 59, 64, 65, 66, 98, 103, 144, 145, 147
u	SAT variable for the tile u, 146
u	Tile, 46, 47, 48, 59, 60, 62, 65, 66, 67, 73, 75, 88, 89, 94, 97, 98, 101, 105, 107, 144, 145, 146, 149
V	Set of application graph vertices, 45
v	Number bits of a serialized task, 69, 67
random	Function which returns a normal distributed random value in the specified bounds, 66, 67
valid	Functions which determines if a u is valid, i.e. has valid coordinates, 66, 67
W	Amount of work, 20, 21
W_E	Worst-case execution time, 47, 49, 50, 60, 95, 96, 97, 98, 99
W_R	Worst-case response time, 51, 52
χ	NoC width, 48, 49, 60, 64, 66, 76, 144
x	X-coordinate, 47, 66, 144, 145
$\mathfrak{X}$	Set operating point, 106, 110, 111
$\mathfrak{x}$	Operating point, 106, 107
Ξ	Temporal interference, 50
$\mathcal{Y}$	NoC height, 48, 50, 59, 64, 66, 76, 144
y	Y-coordinate, 48, 66, 144, 146
z	Number of bits of a serialized task, 71

Abstract

In recent years, heterogeneous multi- and many-core systems have emerged as architectures of choice to harness the performance potential of the ever increasing transistor density of modern semiconductor chips. As traditional communication infrastructures such as shared buses or crossbars do not scale for these architectures, NoC have been proposed as novel communication paradigm. However, these NoC-based many-core architectures require a different way of programming, OS, compilers, etc. Invasive computing addresses these issues and combines research on a holistic solution from software to hardware, for current and future many-core systems. Using the three invasive primitives invade, infect, and retreat, the application developer can exclusively claim resources, use them for parallel execution, and make them available for other applications after the computation is finished.

In the realm of invasive computing, this book proposes methodologies to map applications, i.e., invading computing and network resources for a static application graph, to NoC-based many-core architectures.

The first method is called self-embedding and utilizes the inherent task-level parallelism of applications, modeled by application graphs, by distributing the mapping process to the different tasks. Each task is responsible for mapping its direct succeeding task in the application graph and the communication towards there. Exploring the status and resource availability of the mapping task's local neighborhood only instead of a global search makes this application mapping highly scalable while offering competitive quality in terms of NoC utilization.

The second contribution of this book targets guarantees on non-functional execution properties of applications. While self-embedding maps applications in a distributed, scalable, and fast manner, it is strictly performed during run time and does not conduct any analysis which is inevitable for a predictable execution. As a remedy, we propose a novel HAM methodology which combines compute-intensive analysis at design time with run-time decision making. The design-time analysis is performed during a DSE which aims to find Pareto-optimal mappings regarding the multi-objective optimization criteria such as timing, energy consumption, or resource usage. With the concept of composability, applications can be analyzed individually and then combined during run time to arbitrary