Small or medium-scale focused research project (STREP)

ALMA

Architecture oriented paraLlelization for high performance embedded Multicore systems using scilAb

FP7-ICT-2011-7

Project Number: 287733

Deliverable D3.1

D3.1 ALMA specific state-of-the-art analysis

Editors:	Panayiotis Alefragis, George Goulas/TMES
Authors:	TMES: G. Goulas, P. Alefragis, N. Voros, S. Vasilopoulos, S. Konstantopoulos A. Kakarountas, V. Maratou, K. Antonopoulos, K. Petropoulos, A. Triantafyllopoulou, K. Christopoulos, C. Gogos, C. Valouxis KIT: M. Hübner, T. Stripf, T. Bruckschlögl UR1: S. Derrien, D. Ménard, O. Sentieys UOP: G. Dimitroulakos, K. Masselos, N. Kavvadias Recore: G. Rauwerda, K. Sunesen, J. Potman, P. Klok IOSB: D. Göhringer, T. Perschke ICOM: N. Mitas, D. Kritharidis
Version:	v2.00
Status:	FINAL
Dissemination level:	PUBLIC (PU)
Filename:	FP7-ICT-2011-7-287733-WP3-D3.1-v2.0.docx

ALMA Consortium

Karlsruhe Institute of Technology (Coordinator)	Germany
University of Rennes I	France
Recore Systems B.V.	Netherlands
University of Peloponnese	Greece
Technological Education Institute of Messolongi	Greece
Intracom SA Telecom Solutions	Greece
Fraunhofer Insitute of Optronics, System Technologies and Image Exploitation	Germany

Document revision history

Version	Based on	Date	Author	Comments / Changes
v1.00	-	30/12/11	Panayiotis Alefragis	Final release version
v2.00	v1.00	15/10/13	George Goulas	2nd year revision

Abstract

This document presents the ALMA specific state of the art. The ALMA project aims to perform research in the area of architecture-oriented parallelization for high performance embedded multi-core systems using Scilab and to provide a toolset to demonstrate the above research goals.

The ALMA specific state-of-the-art deliverable provides an overview of the area for techniques for generating parallel code targeting multicore architectures. In the deliverable, a discussion about the ALMA project in the context of the ARTEMIS and HIPEAC roadmaps is followed by an overview of existing design methodologies for multi-core embedded systems is presented. Following, existing multicore architectures, their characteristics and programmability is discussed next; in particular, architecture characteristics of multi-core and many-core architectures, as well as existing many-core architectures, the programmability of many-core processor architectures, compilation techniques and compiler development and parallel code generation issues are discussed. Existing approaches to parallelism extraction are presented next; in particular, optimization techniques for intermediate code mainly with respect to data transfers are presented, as well as coarse and fine-grained parallelism extraction techniques.

List of Figures

Table of contents

Glossary of Terms

AADL	Architecture Analysis and Design Language
ADL	Architecture Description Language
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric MultiProcessing
API	Application Programming Interface
ASG	Abstract Syntax Graph
AST	Abstract Syntax Tree
AVX	Advanced Vector Extensions
BUG	Bottom-Up-Greedy
CDFG	Control and Data Flow Graph
CFG	Control Flow Graph
CIL	Common Intermediate Language
CPU	Central Processing Unit
CRISP	Cutting edge Reconfigurable ICs for Stream Processing
DSL	Domain-Specific Language
DTSE	Data Transfer and Storage Methodology
GCC	GNU Compiler Collection
GPP	General Purpose Processor
GSP	General Streaming Processor
DSP	Digital Signal Processor
ELF	Executable and Linking Format
EULA	End User Licence Agreement
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
HDL	Hardware Description Language

HIR	High Level Intermediate Representation
HLA	High Level Synthesis
HPC	High Performance Computing
IMS	Integrated Modulo Scheduling
ILP	Instruction Level Pipelining
IR	Intermediate Representation
ISA	Instruction set architecture
JIT	Just In Time
KAHRISMA	KArlsruhe's Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array Processor
LIR	Low Level Intermediate Representation
LISA	Language for Instruction Set Architecture
LISP	Is a family of computer programming languages
LLVM	Low-Level Virtual Machine
LTI	Linear Time-Invariant
MCA	Multicore Association
MMX	Multi Media Extension
MPSoC	Multiprocessor System on Chip
MPPB	Massively Parallel Processor Breadboarding
NLP	Nested Loop Programs
NP	Non-Polynomial
NoC	Network on Chip
NUMA	Non-Uniform Memory Access
OpenCL	Open Computing Language
PCCA	Partial Component Cluster Assignment
PIS	Pragmatic Integrated Scheduling
PIP	Parametric Integer Programming

PEMBIC	PEloponnese ALMA Scilab/MATLAB BITcode
PTX	Parallel Thread eXecution
RHOP	Region-based Hierarchical Operation Partitioning
RTL	Register Transfer Level
RFD	Reconfigurable Fabric Device
SCoP	Static Control Part
SIMD	Single Instruction Multiple Data
SMP	Symmetric MultiProcessing
SoC	System-on-Chip
SRA	Strategic Research Agenda
SSA	Static Single Assignment
SUIF	Stanford University Intermediate Format
SWP	Sub-Word Parallelism
UAS	Unified Assign and Schedule
UMA	Uniform Memory Access
VHDL	Very high speed integrated circuits Hardware Description Language
VLIW	Very Long Instruction Word

1. Introduction

The ALMA project aims to research automated parallelism extraction for Scilab code, targeted for embedded multi-core platforms and to present the research results as a toolset. The ALMA toolset inputs are a program in a language subset of Scilab, instrumented with specific ALMA annotations, as well as a target multi-core embedded architecture description. Based on those inputs, the ALMA toolset will generate automatically an optimized executable for the target multi-core embedded architecture. The toolset design will be generic in the sense that it will take into account possible future extensions for (heterogeneous) multi-core SoCs. The initial multi-core target architectures to be used with the ALMA toolset are the Recore Xentium architecture and the KIT Kahrisma architecture.

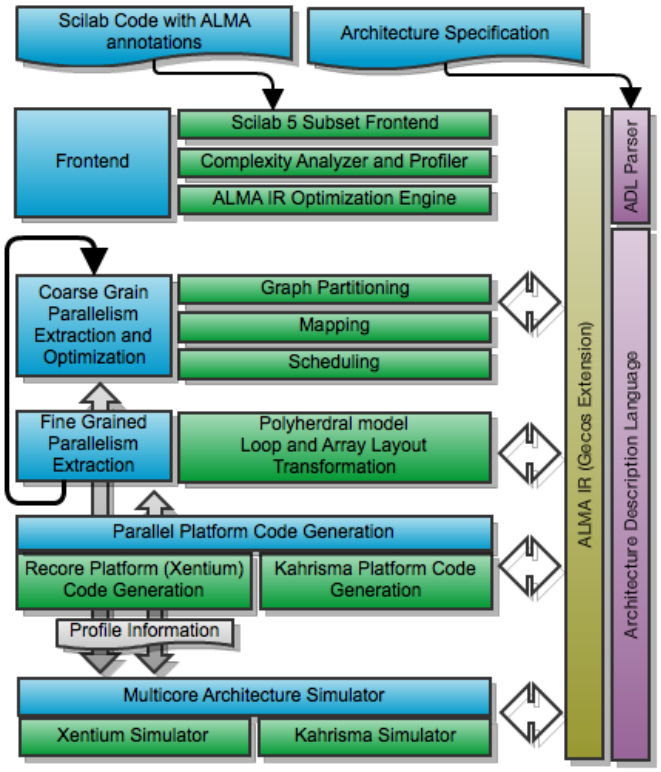

Figure 1: ALMA toolset overview

This document presents the state of the art related to the ALMA toolset specification as presented in . More specifically, a high level state of the art of multi-core processors and their characteristics are presented as well as compiler state of the art, relevant to embedded multi-core code generation. The main focus of this document is the state of art for parallelism extraction methodologies. It should be mentioned that the usual granularity to define the limit between fine and coarse grained parallelism extraction is that fine grained refers to concepts close to the hardware, like instruction level parallelism and pipelining, while coarse grained parallelism refers to loop constructs parallelism extraction. For ALMA, fine grained parallelism extraction is extended to loops parallelization, while the coarse grained parallelism extraction faces a larger picture of the application, viewing even loops as single tasks. This document also presents intermediate code optimizations, with emphasis on data transfers. Finally, existing parallelization design methodologies are presented.

2. ALMA in the context of ARTEMIS and HIPEAC Roadmaps

One of the main goals of ALMA, apart from producing state-of-the-art technology for strengthening the European industry position world wide, is to share a common vision with other European fora and envision a common goal.

As members of ARTEMIS and HIPEAC, the ALMA consortium members have strong relationships with both organizations and participate actively in order to establish a viable, competitive roadmap until 2030.

ARTEMIS Strategic Research Agenda (SRA)[1] for the next generation of Embedded systems is summarized in Figure 2. As described in the figure, ARTEMIS highlights two parallel sets of industrially driven research objectives:

- Research into technology that will offer completely new solutions to the technical barriers that hinder progress towards the application context's goals.
- Technical solutions that form the basis of developing the pre-competitive industrial goals, by combating the complexity of new systems through **improved designs** and **implementation processes** and **tools**, a major Innovation Accelerator.

In more detail, the ARTEMIS SRA distinguishes three main areas of research:

- Reference designs and architectures.
- Seamless connectivity and interoperability.
- Design methods and tools.

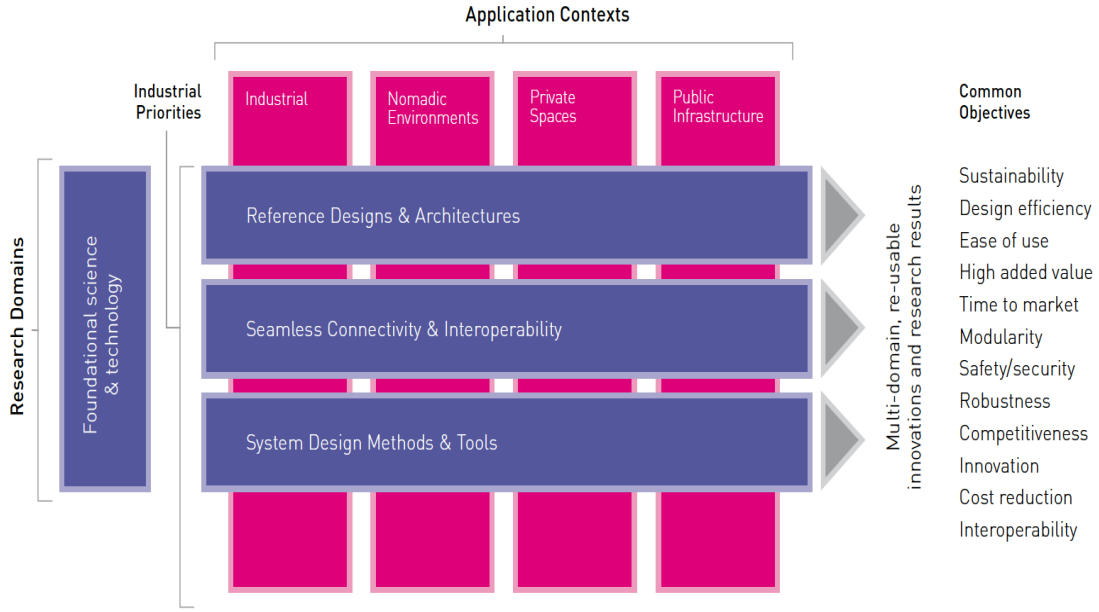

Figure 2: Relationship between the application and research dimensions of the ARTEMIS Strategy

All these topics are elements that will be extended in the ARTEMIS successor to embrace the technology challenges for 2030, with more complex embedded systems that are:

- Derived from architectural models and principles allowing reusability, composability.

- Safe and secure by design, based on interoperability standards for systems and design tools.
- Situation aware for distributed real-time and highly certified operations.
- Interconnected to enable the development of new and smart applications and to create solutions to the areas of major change.
- Dynamic, autonomous, adaptive and self-organised.
- Seamlessly interacting with their environment.
- Designed using optimised and consistent processes and tools.

In parallel, the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HIPEAC) shares and extends most of the ARTEMIS goals. According to the HIPEAC Roadmap [2]: "*Designing embedded systems often requires hardware-software co-design for a product that will sell only tens of thousands of units. This small volume makes design too expensive with today's tools. To overcome this,* ***more powerful design tools*** *are needed that can* ***reduce the time to adapt*** *or* ***create new designs****, particularly through* ***integration*** *and* ***cross-unit optimization***".

According to the HIPEAC Roadmap, the major key factors for embedded computing will be:

- Power efficiency and massive parallelism (e.g, GPU, many-core) since it will be required from modern application and systems.
- Reduced resource utilization.
- Improved reliability.
- Increased network integration.
- Certified programming languages and design paradigms.

	ARTEMIS Roadmap Objectives		
	Reference designs and architectures	*Seamless connectivity and interoperability*	*Design methods and tools*
ALMA Contribution	• Two state of the art parallel architectures will be developed as part of ALMA • Architecture Description Language (ADL) for describing the underlying architectures	• Interoperability among ALMA optimization tools through the GeCoS environment	• ALMA front end tools for efficient profiling • Fine grain and coarse grain parallelization tools • Efficient Parallel code generation • Multi core simulators with profiling and tracing capabilities

Table 1: ALMA relation to ARTEMIS Roadmap

	HIPEAC Roadmap Objectives

	More powerful design tools	*Reduce the time to adapt*	*Create new designs through cross unit optimization*
ALMA Contribution	• ALMA front end tools for efficient profiling • Architecture Description Language (ADL) for describing the underlying architectures • Multi core simulators with profiling and tracing capabilities	• Fast design space exploration through coarse grain parallelization algorithms • Architectural adaptations are easier through the use of ADL • Reconfigurable parallel architectures from KIT and Recore	• Fine grain and coarse grain parallelization tools • Efficient parallel code generation optimized for the underlying architectures

Table 2 ALMA relation to HIPEAC Roadmap

As revealed from the above analysis, the ALMA project objectives are fully in line with the vision described in ARTEMIS and HIPEAC Roadmap since it intends to deliver an innovative methodology and associated tools for developing cost effectively complex embedded systems that will take advantage of the underlying multicore architectures developed by KIT and Recore.

According to its roadmap, ALMA contributes to several goals of ARTEMIS and HIPEAC Roadmap as depicted in Table 1 and Table 2.

3. Overview of existing design methodologies

Figure 3 shows the flow chart of the typical embedded systems design methodology, used in many companies and research institutes. As can be seen, the development process is divided into two parts; the first part, which is done by the application programmer and the second part, which is done by the embedded system developer. This separation is necessary, as most application programmers are experts in their application domain but they are not experts in parallelization and hardware architectures. Therefore, embedded system developers are needed for the second part. To achieve a good solution, efficient communication between application developers and embedded system developers is essential.

The general design methodology starts with a specification description. The programmer then develops the application using high level languages, such as MATLAB or Scilab. The advantages of these languages are that different algorithms can be quickly explored. If high performance is already important at this stage, parallel simulation can be employed (e.g. Parallel Computing Toolbox by MathWorks) or (parts of) algorithms may be programmed using C/C++. Although high performant commercial C/C++ libraries for e.g. image processing exist, containing many of the basic algorithms, their main disadvantage is that often the source code is not available and therefore these libraries cannot be used for embedded systems. After the application programmer has successfully developed the application, he/she forwards the high level description together with code documentation to the embedded system developer.

The embedded system developer analyzes the application by considering the target platform (e.g. embedded multi-core, DSP, FPGA) and the requirements of the user (e.g. real-time performance, power). Hereby static and dynamic analysis concepts are used. For example profiling is used to identify possible hotspots and bottlenecks in the current implementation. Also the communication and memory access patterns of the application need to be identified using tool-based or manual analysis. All these analysis techniques need to be done in respect to the target platform, as e.g. other platforms have different memory and communication infrastructures. Also different processors have different instruction sets and varying execution times for the instructions. To execute these analysis techniques for the target platform, the embedded system developer, manually adapts the application for the target platform based on his/her experience to get a first running draft version. Common adaptations are transformation from floating-point to fixed-point arithmetic and transformation aiming to optimally implement parts of the algorithm on the target platform. After the first draft has been successfully generated, it is analyzed using the techniques mentioned above. For some of these analysis methods there exist tools such as gprof[3,42] for profiling, for others these steps are done manually. Based on the results of the analysis, the application is manually modified, e.g. by choosing a different parallelization. For these modifications a strong interaction between the embedded system developer and the application programmer is extremely important to exploit the different expertise and to prevent long redesign cycles.

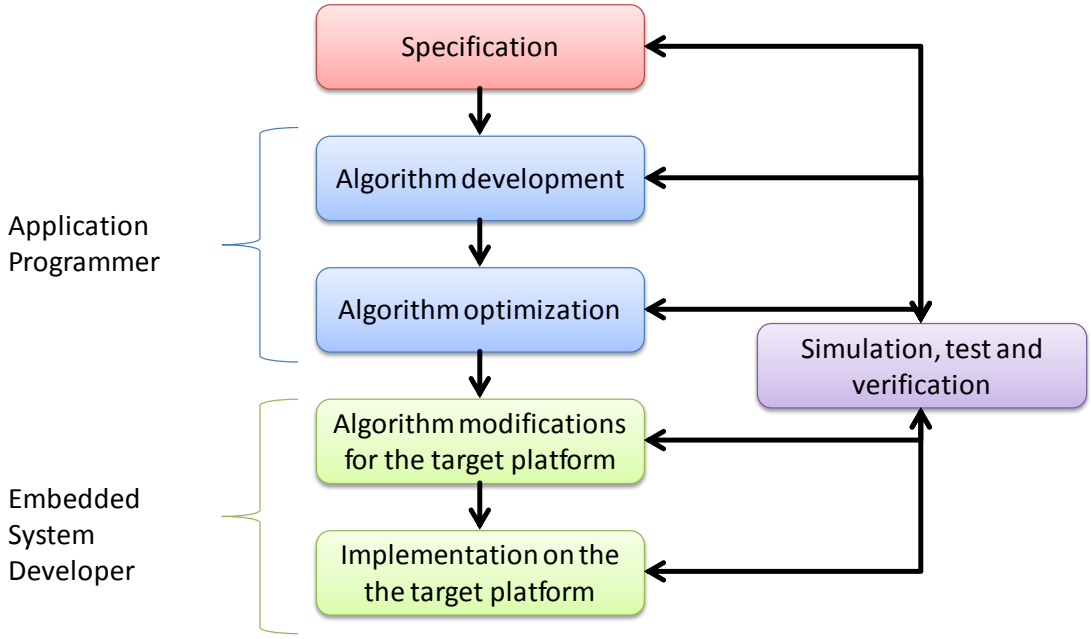

Figure 3: Flow chart of the typical embedded systems design methodology

In addition, the embedded system developer can use tools for automatically adapting the application to the target platform. For example, to program Digital Signal Processors (DSPs) or microprocessors, MATLAB provides the Real-Time Workshop[4] which can generate C code from embedded MATLAB code. However, the transformation from MATLAB code to embedded MATLAB code has to be done manually by the programmer. Furthermore, the generated code is not as efficient as the hand-written code from an experienced programmer. In addition, not all target platforms are supported, for example there is so far no support for embedded multi-core platforms.

The support of other architectures such as FPGAs using MATLAB HDL-coder or C-to-FPGA tools, such as ImpulseC[5] or CatapultC[6] is very limited and can only be used for accelerators. Therefore, for FPGAs mostly proprietary frameworks and libraries of frequently used Intellectual Property (IP) cores exist, such as memory and communication interfaces or optimized algorithms. New algorithms can then be integrated into the existing framework using such C-to-FPGA tools. On this front Xilinx has lately introduced the Vivado HLS tool which works on the same principles as all other C-to-FPGA tools.

The latest trend for embedded and high performance computing is the OpenCL framework. Originally it was developed as a standard to express parallelism in a more coherent way than current solutions (which are based on complex APIs) while retaining low level features for synchronization and memory management and still be re-targetable.

OpenCL is now supported on various platforms ranging from CPUs (Intel Core series, ARM), GPU/GPGPU (e.g. Nvidia Kepler, AMD FirePro, Intel Xeon Phi, ARM Mali) as well as FPGAs from Altera. Currently, OpenCL support for Texas Instruments DSP processors is in active development

With OpenCL, the abstraction layer is raised in terms of expressing parallelism on homogeneous or heterogeneous platforms. However, the framework provides little support on problems where the ALMA project has planned solutions (e.g. the application developer still has to deal with the more strict type system of C).

4. Existing multicore architectures, characteristics and programmability

Application demands are increasing for embedded systems requiring e.g. more features, performance and flexibility. As a consequence, the complexity of the hardware that implements the demanding applications increases as well, resulting in an increasing challenge on system architecture, power consumption management and programmability.[7] [8] [9,10,11, 12]

It is becoming difficult to predict which applications are going to be successful in the future and what kind of hardware is required to support these. For this reason, application developers are interested in programmable multi-core architectures, which enable the anticipation on expected market trends and offers flexibility, if the market develops differently [13] . Moreover, technology trends show that systems develop towards heterogeneous multi/many-core systems with different cores or hardware accelerators specialized for certain tasks, where power consumption and dissipation plays an important role for embedded systems [12,14].

4.1 Multi-Core and Many-Core Architecture Characteristics

Multi-core CPU architectures do commonly fall under the multi-core paradigm, i.e. parallel “fat” cores that are independent computing units. These can be programmed independently with communication capabilities between them or by using an Operating System, which combines multiple cores into one seamless processing environment and thereby offering Symmetric Multiprocessing (SMP) capabilities. The application does not need to be aware of the individual cores, but only defines tasks/threads to process. This simplifies software development and software portability, as an application is implemented without knowing the exact system topology. Heterogeneous elements within these systems are being handled by specific software stacks which need to be integrated inside the processor’s operating system tasks[12]. Multi-core Operating Systems often rely on a Unified Memory Architecture (UMA), where all processors share the same memory and I/O, and access times are nearly identical across the whole system which often relies on a traditional “interconnection bus”[15].

There are also SoC architectures that follow the many-core paradigm. In a many-core architecture the processing units are often smaller and the number of processing units larger. There is no exact definition for the number of cores in a many-core; tens, hundreds or more cores, where a workload is distributed over these cores. The complexity of new multi-core processors as well as many-core architectures and the growing requirements in terms of computational power, led to novel paradigms of interconnection among the various hardware components within a chip (and within a system). Mainly driven by the fact the capacity of a shared-bus interconnect does not scale with large hardware architectures interconnection solutions, like Network-on- Chip (NoC) come into play. In a NoC-based interconnect hardware elements communicate in parallel over dedicated communication links by means of packets or messages. Abandoning the traditional “interconnection bus”, UMA cannot be guaranteed anymore, rendering many-core architectures into Non-Uniform Memory Access (NUMA) architectures.

4.2 Existing Many-Core Architectures

Many-core architectures which fit within the many-core paradigm are the MPPA 256 many-core from Kalray[16], the TILE-Gx™ Processor from Tilera[17], the Epiphany-IV 64-core from Adapteva[18] and the Cutting edge Reconfigurable ICs for Stream Processing (CRISP)[19].

The FP7 CRISP project developed a scalable and dependable reconfigurable many-core system concept that can be used for a wide range of streaming applications in consumer, automotive, medical and defence markets. The envisioned platform solution includes a massive multi-core processing architecture[20], called the General Streaming Processor (GSP). The GSP fuses the many-core concept with NoC interconnection solutions.

The architecture is based on the use of the Reconfigurable Fabric Device (RFD) chips, which contains nine Xentium[21] cores and two memory tiles connected with a high bandwidth packet switched reconfigurable NoC and interfaces for external communication. A general-purpose processor (GPP), e.g. ARM, is used for the platform supervision in order to execute e.g. the reconfiguration of the platform.

The CRISP architecture concept has been further evolved within the "Massively Parallel Processor Breadboarding" project (MPPB)[22] where instead of an ARM integrated into a fully NoC oriented architecture, a Leon 2 GPP / AMBA subsystem is integrated with a NoC interconnect system. Hardware devices on both systems are memory mapped and are accessible by both the AMBA subsystem and the NoC system, which makes seamless interoperability possible.

4.3 Programmability of Many-Core Processor Architectures

For MPPB and other many-core architectures, traditional SMP UMA based multi-core programming methods shift to the many-core programming paradigm. Designing for 2, 4 or 8 homogeneous cores would not work anymore. Software development needs to take into account the architectural aspects like interconnect and heterogeneous elements in order to utilize these architectures efficiently. Thereby software development is forced to move from the existing ways of programming a multi-core and to look for alternatives to cope with the increasing complexity of heterogeneous hardware architectures.

One way to cope with this paradigm is to partition applications into separate processing parts, which will be referred to as tasks. These tasks are executed on the individual processors, which use communication functionality to synchronize the tasks and to transport data for processing between the processors. So, the tasks need to be identified and partitioned in such a way that they fit efficiently into the intended target architecture. The Therefore, the many-core programming paradigm includes inter-processor communication between tasks as well as tasks deployment.

The implementation level of these those many-core software solutions implementations can range from bare metal implementation on the intended hardware without any software support to having Operating System support on each individual element. The advantage of bare metal implementations is that the targeted hardware can be utilized efficiently without much overhead; however, the downside is that it requires a significant development amount of effort to implement the application on the many-core architecture. On the other hand, Operating Systems require also processing resources and processing time, which decreases efficiency; but it reduces the implementation effort for an application to run on the many-core architecture.

There are solutions available to facilitate this way of implementation by providing communication libraries; some examples for communication are MPI[23], Linx[24], MCAPI[25]. The

communication libraries provide a means to communicate from processing elements/processors in a standardized abstract way, which makes it suitable for heterogeneous (many-core) systems.

The pain of “manual deployment” can be reduced by using tools like Poly-Platform™ [26] or MP Designer™[27]. Using tools to partition an application into separate processing tasks, eases software development development on multi-/many-core architectures. However, manual effort to partition an application and take care of the deployment of the individual processing elements is still required.

Changes in hardware or software may lead to other partitioning of the application in order to efficiently utilize the system hardware; the application partioning becomes hardware architecture dependent and hard to maintain, expand or port. Maintaining the deployment of the application on a many-core architecture is becoming complex as the amount of individual processors inside a many-core system is increasing.

At this moment there are solutions available to offer a higher level of implementation comfort on AMP/NUMA heterogeneous architectures. A small overview of solutions is given:

- OpenCL[28,29]
  Master Slave architecture cross platform parallel programming framework which is able to submit work in the form of architecture specific kernels from a host (Master) to a processor, mostly used in GPU acceleration for GPP.
- Multi-core Association (MCA) API’s[30]
  A set of C based application programming interfaces for embedded systems, supporting communication (MCAPI), resource management (MRAPI), task management(MTAPI).
- UPC[31]
  Unified Parallel C which is an extension on C99 programming language, it offers C extension on parallel programming, shared address space, synchronisation and memory management in the form of compiler directives.
- OpenACC[32]
  Extension on OpenMP[33] to include hardware acceleration support. The extension is in the form of compiler directives.
- OpenHMPP[34]
  Master Slave architecture based parallel programming model using compiler directives for which it is possible to specify computations, which can be processed by a slave processor/hardware accelerator.

All of these solutions require a runtime to be implemented supporting the intended target architecture. Observing the current solutions, the majority is targeted to perform massive heterogeneous computing. In case of OpenCL, OpenACC and OpenHMPP the runtimes strongly rely on a host/accelerator or master/slave architecture, where one core is responsible for hosting, running and controlling the application. These standards have a strong background in High Performance Computing (HPC) where heterogeneous support is often done in form of GPUs. . The solutions from the HPC domain are less suited for embedded systems, because of large overheads in terms of performance, memory and power. The MCA APIs seem to focus more on the embedded area. All mentioned APIs require runtime implementations where the directive-based solutions require this to be combined with a matching compiler.

Additionally, initial research is being performed on many-core Operating Systems which target NUMA based heterogeneous many-core architectures; Barrelfish[35] is a first “Proof of concept” open source research Operating System targeting NUMA architectures.

While embedded systems are changing to many-core including heterogeneous architectures with complex interconnect structures, the application demands are growing in features, real-time performance and power consumption. Programmability of these systems with the same comfort as a single core does not seem to exist anymore. The challenge now, is to provide ease of programming on this kind of many-core systems to application developers.

4.4 Architecture Description Languages

The main goal of an architecture description is a formal description of components of the system or the system as a whole, the structure and behavior of system components and their interconnections.

To achieve this description a wide range of languages can be used. Whereas natural languages tend to be inaccurate and are hardly amenable to machine processing, multiple languages have emerged with a formal syntax for system descriptions.

Languages with a formal syntax are, for example, programming languages, scripting languages or Hardware Description Languages (HDL). Most of these languages can be used for the description of software, hardware, mechanical or other systems and could be used as an Architecture Description Language (ADL). However, not all of them are suitable for describing large system-architectures as they are designed for a very special purpose, e.g. programming languages are hardly usable for the description of mechanical systems.

Typically ADLs provide a formal syntax for system description and are mainly designed for processor or system description and support hardware development, system verification, simulation or compiler generation. Depending on the purpose of an ADL, either structural or behavioral information or a mixture of both is covered.

Traditional HDLs like VHDL or Verilog offer possibilities for hardware development and verification on a low level of abstraction. They provide structural and behavioral information and enable testing and verification of hardware components, but require very high effort to describe, test and verify large and complex many-core SoC architectures.

LISA (Language for Instruction Set Architecture) is an ADL that sets its focus on behavioral description of processors, instruction sets and on compiler generation. The description is divided into hardware resources and executable operations. This enables the generation of a cycle-accurate simulation environment as well as the generation of a compiler supporting special functions described in LISA. For multi-core architecture design LISA has to be coupled to another multi-core framework, as the LISA language does not support such architectures directly. For multi-core architecture design LISA has to be coupled to another multi-core framework as the LISA language does not support such architectures directly.[36]

The Architecture Analysis and Design Language (AADL), standardized by the Society of Automobile Engineers (SAE)[37], provides a formal syntax to model embedded hardware and software systems. The main goal is to support model-based engineering for embedded systems in the automotive and aerospace domain. The language supports the description of hardware architecture models based on processors, memory and communication busses as well as the description of software models composed of processes, threads and data on system level.

With its capabilities, the AADL allows to model embedded systems and their behavior, software mapping to hardware modules and communication between threads and processes. Additionally, AADL supports formal analysis and simulation for performance, safety, security and reliability requirements validation through component properties. The language focuses on the system level view and therefore does not allow for a low level modeling of processor

architectures, instruction sets, cache hierarchies or detailed communication systems like Network-on-Chip architectures as required by the coarse- and fine-grain parallelization steps.

One of the most prominent languages used for system simulation is SystemC[38]. SystemC is a C++ library that extends C++ with basic structures, modules, ports and interfaces for structural hardware description. The behavior of a module or interface can be added with threads and processes. The range of SystemC covers hardware description, very similar to traditional HDLs like VHDL, as well as system and software description. Additional libraries may also extend the functionality of SystemC. The most interesting SystemC library for system description is TLM – Transaction Level Modeling. TLM provides structures, interfaces and functions to efficiently model the communication of system components. However, as SystemC is a C++ library, the handling of SystemC is more one of a programming language. Therefore SystemC does not allow extracting behavioral information, which is a crucial step for application parallelization.

An extension to SystemC is ArchC[39]. The purpose of ArchC is to provide standardized keywords, macros and templates to simplify architecture description. The direction of ArchC is more into typical ADLs to provide an overview of the whole system and use a special syntax - here functions - to describe the architecture. Despite this, the main focus of ArchC lies in the description of processor architectures and not in system description.

The SoClib[40] platform aims to provide a range of system components, called modules, described in SystemC and in a special meta-language. This meta-language can be used to create and connect larger systems in an easy way. Modules can be available at two levels of abstraction, either cycle accurate or at a transactional level. Therefore the SoCLib meta-language can be accounted as a typical architecture description language. However the description is mainly structural and offers only limited possibilities to describe module behavior on system level.

Even though many different Architecture Description Languages exist, most of them follow a special purpose that limits their usage in different areas. For the ALMA project, the requirements are a structural description on a high level of abstraction for system simulation combined with an analyzable behavioral module description for the parallelization of applications.

4.5 Compilation Techniques and Compiler Development

Compiler development has for decades been at the heart of computer science, and has remained an extremely active field of research. While the importance of compilers and language technology was appreciated to a varying degree by computer scientists over the last decades, facts are that a never-ending pipeline of new hardware and processor architectures, as well as new demands such as reduced energy consumption, always implied new peaks in compiler research. There is a gigantic amount of literature and textbooks; however the best recent survey of the field was compiled by Srikant and Shankar[41], who published an excellent collection of articles on advanced topics in language technology, focusing on optimisation and code generation.

Nowadays, modern compiler frameworks (e.g. GCC[42], LLVM[43], Trimaran[44], Open64[45]) are separated into three parts: the front-end, the middle-end and the back-end. The front-end checks whether the program is correctly written in terms of the programming language syntax and semantics. Here legal and illegal programs are recognized. Errors are reported, if any, in a useful way. Type checking is also performed by collecting type information. The front-end then generates an intermediate representation (IR) of the source code for processing by the middle-end. The middle-end is where most optimizations takes place. Typical transformations for optimization are removal of useless or unreachable code,

discovery and propagation of constant values, relocation of computation to a less frequently executed place (e.g., out of a loop), or specialization of computation based on the context. The middle-end generates another IR for the following back-end. The back-end (also referred to as code generation) is responsible for translating the IR from the middle-end into assembly code. The target instruction(s) are chosen for each IR instruction. Register allocation assigns processor registers for the program variables. The back-end utilizes the hardware by figuring out how to keep parallel execution units busy, filling delay slots, and so on. Although most algorithms for optimization are nondeterministic polynomial (NP), heuristic techniques are well-developed.

Besides the instruction selection and register allocation, the compilation back-end is responsible for scheduling the instructions on the target architectures. For RISC processors the scheduling assigns an order to the instructions and places each instruction into one time slot. Within the ALMA project both multi-core architectures rely on VLIW processors as processor cores. For VLIW processors the scheduling must additionally perform a fine-grained parallelism extraction and utilize the available instruction level parallelism in order to decide which instructions can be executed in parallel. In the past, several techniques for improving fine grained parallelism instruction for VLIW processors have been researched. The techniques includes superblock scheduling [46], trace scheduling [47], hyperblock scheduling[48] and modulo scheduling[49].

Compilers targeting VLIW architectures with clustered or distributed register files must decide – additionally to compilers for single-cluster VLIW architectures – on which cluster each operation will be executed (cluster assignment) as well as generate inter-cluster communication (ICC) to copy register values between clusters. The cluster assignment algorithm and its integration into the compilation back-end influence the quality of the fine grained parallelism extraction. The first compiler that supported clustered-VLIW architectures was the Bulldog compiler [50] described in a Ph.D. thesis by John Ellis in 1985. Within this compiler also the first cluster assignment algorithm Bottom-Up-Greedy (BUG) for performing the cluster assignment was introduced. After that, several cluster assignment algorithms and integrations into compiler back-ends have been published. The integration into the compiler back-end is mainly influenced by the phase ordering. The clustering can take place as a separate and isolated phase before scheduling. Cluster assignment algorithms using this phase ordering are BUG, Partial Component Cluster Assignment (PCCA)[51], and Region-based Hierarchical Operation Partitioning (RHOP)[52]. Alternatively, cluster assignment and scheduling can be coupled and take place within one phase. The coupled phase ordering is addressed by Unified Assign and Schedule (UAS) [53], Pragmatic Integrated Scheduling (PIS)[54], and Integrated Modulo Scheduling (IMS)[55] cluster assignment algorithms. In contrast, Capitanio[56] performs the cluster assignment after scheduling.

4.6 Parallel Code Generation

Parallel code generation for multi-core architectures relies on the code generation for single processors. So for each processor of a multi-core architecture the code generation is individually performed. The synchronization between the processor cores of distributed memory architectures is realized with dedicated communication primitives and the communication on multiprocessor systems can present a huge impact on the whole system performance[57]. There has been a lot of research for optimizing the message communication overhead for parallel code generation.

The start-up cost of communication for distributed-memory architectures is typically greater than the per-byte transmission cost. That offers optimization potential that is used by the three following communication optimizations by combining messages in various ways to reduce the total amount of communication overhead [58]. Message coalescing [59] detects

redundant communications and coalesces them into a single message, allowing the data to be reused rather than communicated for every reference. Separate communications for different references to the same data are avoided if the data has not been modified between uses. Message aggregation[60] reduces the number of messages between the same source and destination by aggregating them into a single larger message at the expense of copying them to a single contiguous buffer. Recently, message aggregation was used also to reduce the energy consumption by the reduction of the number of channels in a multi-processor system-on-chip with a Network-on-Chip[61]. Message vectorization optimizes array element accesses indexed within nested loops that can be vectorized into a single larger message. Dependence analysis is used to determine the outermost loop at which combining can be applied.

A number of optimizations seek to hide communication overhead by overlapping message with communication. Message pipelining attempts to hide message transfer time by separating send and receive primitives for element messages. Message pipelining is supported by various architectures by non-blocking messages that hide the message copy time.

5. Existing approaches for parallelism extraction

5.1 Optimization of intermediate code mainly with respect to data transfers

An important issue in contemporary compilers is the careful design of the compiler intermediate representation (IR)[62] which is a necessity, due to its dual purpose: it both serves as an abstract target machine and as input to an ideal retargetable machine back-end. Another increasing trend covers the application of powerful machine-independent optimizations at the abstraction of high-level intermediate code. Then, the resulting IR can be correspondingly mapped for JIT, GPU or GPP back-end compilation.

The IR must be able to bridge the large semantic gap between a high-level language and machine-dependent code, and at the same time to provide the means for developing easy to write, efficient and correct analyses and optimization transformations[63].

5.1.1 IR code optimizers of popular compilers

Recent compilation frameworks provide and expose a linearized IR form for applying textbook analyses and optimizations[64, 65, 66].

GCC[42] (version 4.0 and onwards) supports the GIMPLE mid-level IR[67] which can be expressed in Static Single Assignment (SSA) form[68, 69]. Many GCC optimizations traditionally applicable to the machine-dependent RTL (Register Transfer Language) representation (a kind of list-processing macro language based on LISP) have been rewritten for GIMPLE. Compiling a source file with **`-fdump-tree-all-raw`** generates GIMPLE dumps for all intermediate stages prior entering RTL form. Although these dumps provide incomplete information, it is possible to devise GCC plugins that emit a faithful reconstruction of the internally used GIMPLE, sufficient for targeting back-end architectures. The “GIMPLE front-end” and “GIMPLE back-end” projects are longer term aims for allowing direct GIMPLE import and export to and from GCC, thus rendering GCC to an all-purpose machine-independent optimizer.

GRAPHITE[70,71] introduced the polyhedral model in GCC for designing efficient, fine grained, source-level compiler analyses and optimizations. Typical supported transformations are loop blocking, interchange, strip-mining and parallelization of innermost loops. GRAPHITE is considered near-ready for production use. Further, there exist early attempts to extending GRAPHITE for non-static or irregular control parts of programs.

LLVM[43] is a modern framework for developing compilers that draws growing interest. LLVM uses a register-based VM/compiler IR named LLVM bitcode that can be targeted by the clang C/C++ frontend[72]. High-level profiling is readily accessible since LLVM bitcode can be interpretively executed. LLVM supports a number of sophisticated optimizations such as SSA-based loop-oriented intraprocedural optimizations and certain interprocedural / link-time optimizations.

Polly[73] is an advanced data-locality optimizer and automatic parallelizer. It uses the polyhedral model to analyse and optimize the memory access pattern of a program. It aims in speeding up sequential code by improving memory locality and consequently the cache use. Furthermore, Polly is able to expose different kinds of parallelism, which it exploits by introducing OpenMP[74] and SIMD code. Future work includes vectorization for GPU targets via the LLVM PTX[75] back-end. Polly is not yet considered ready for production use.

COINS[76] is a compiler infrastructure written in Java, that supports two IRs: the high-level IR (HIR) and the low-level IR (LIR) based on S-expressions. COINS features a powerful SSA-based optimizer and extensive support of optimizations devoted to exploiting parallelism: SIMD, VLIW as well as symmetric multiprocessing (SMP).

Trimaran/IMPACT is a unified compiler/simulator environment[44] for a parameterized VLIW architecture. Trimaran supports optimizations for local ILP and software pipelining. It supports the feedback of analysis results for code improvement. A recent version added architectural support for multicluster architectures, without the corresponding additions in the compiler side. A popular descendant of IMPACT is Open64[45] that has evolved independently of Trimaran.

Open64[45] is an open-source research compiler targeting a 64-bit architecture. It uses the WHIRL IR, a kind of register-based IR with 5 distinct layers; this characteristic renders its use cumbersome. Open64 has C/C++ and Fortran 90/95 front-ends, however, it is considered rather difficult to extend or port to newer architectures. Open64 includes advanced interprocedural optimizations, loop nest optimizations, global scalar optimizations, and code generation with advanced global register allocation and software pipelining.

LANCE[77,78], an interesting academic compiler, has introduced an executable IR form (IR-C), which combines the simplicity of three-address code with the executability of ANSI C code. LANCE compilation passes, accepts and emits IR-C, which eases the integration of LANCE into third-party environments. However, the available optimization engine focuses on scalar optimizations.

Machine-SUIF[79] is an extensible infrastructure for building back-end compilers and comes with a set of analysis and optimization passes. It supports the SUIFvm linear IR which can be organized into a CFG or SSA form. Additional optimization passes (strength reduction, lazy code motion, if-conversion) and back-ends (ARM) have been added to Machine-SUIF. It is not suitable for loop-oriented optimizations.

SUIF[80] (version 1) is a dormant compiler project that provides significant parallelization capabilities. It supports classic compiler optimizations (e.g. forward propagation, array access scalarization) as well as parallelism and locality optimizations (loop interchange, reversal and skewing) using a data-dependence library and a prototype parallel code generator for shared-address space multiprocessors. SUIF was used extensively in research in the late 90's and early 00's. The aforementioned parallelization transformations were never ported to the second and final version of SUIF (SUIF2).

Phoenix[81] is an initiative of Microsoft Inc.[82] for the development of a modern .NET compiler also supporting static compilation. It appears that Phoenix is a rewrite and extension of the Machine-SUIF API in C# albeit with a different IR[83]. As an IR, the CIL (Common Intermediate Language) is used which is entirely stack-based, a feature that hinders the application of modern optimization techniques. Its licensing scheme (EULA) prevents third parties from commercializing their technologies incorporating Phoenix-based front-ends and code generators.

CoSy[84] is the prevalent commercial retargetable compiler supporting among others C and C++. It uses the CCMIR intermediate language whose specification is confidential. To optimize the IR, it provides a number of optimization passes (such as algebraic simplification, constant propagation, dead code elimination, common subexpression elimination, zero-overhead loop optimizations) while it is extensible towards new analysis and transformation passes.

5.1.2 Source-to-source transformation tools

Another take on IR optimizations involves automated code and Abstract Syntax Tree (AST) restructurings by grammatical transformation tools. Often source-to-source transformation tools[85] are involved and applied on C/C++ code obtained by IR back-translation. Such tools provide the means for performing semantic transformations on the source code/IR; a graph rewriting, tree rewriting, or semantic (pattern) matching engine is typically involved. The optimization range that can be targeted includes scalar/algebraic optimizations, array optimizations (e.g. matrix flattening), data layout optimizations, high-level optimizations (loop transformations, software pipelining), interprocedural optimizations, code obfuscation and other restructurings.

The most popular tools in this category include DMS[86], TXL[87], POET[88], ROSE[89], and Cetus[90].

DMS[86] is a commercial transformation system being in production for several years. The supported grammars span across the entire spectrum of modern high-level languages, more importantly: C, C++, C#, Java, FORTRAN, Pascal, Ada, Matlab, Verilog, VHDL, and SystemC. DMS supports both high-level (source-to-source) and procedural transformations; the latter applied to internal compiler data structures, such as a program IR in memory. The analysis engines that can be built using DMS include attribute grammar recognizers and control/data-flow analysers. One of the main strengths of DMS appears to be the scale of user input that it accepts; it is possible to analyse software systems consisting of tens of thousands of files.

ROSE[89] is a compiler infrastructure written in C++ for building source-to-source translators with complete language support for C90/99, C++, and Fortran 77-90-95-2003. ROSE has been successfully employed in auto-tuning program optimization and claims robust OpenMP 3.0 support. Up to recently, ROSE depended on an external C/C++ frontend by EDG[91]; the latest versions are distributed without this frontend, since the ROSE can be plugged-in to other compilers via the OpenAnalysis API. It features predefined loop optimizations (loop interchange, fusion, fission, splitting, unrolling). The transformation mechanisms are based on source code pattern matching and substitution at a high-level, while at the IR-level, classical procedural transforms can be applied. ROSE is a complex software system (source distribution around 50MB) that requires significant time to elaborate.

TXL[87] is a program transformation system based on the TXL functional rule-based programming language, specifically designed for software analysis, program transformation and domain-specific language (DSL) development. Available language grammars for experimentation include C, C++, Java, Ada, Eiffel, Fortran and Python. TXL liberates the programmer from most programming burden (data structure design and extension, garbage collection, development of a matching and substitution engine). It is also possible to develop various control/data-flow analyzers in TXL, as a set of transformations, applied in a sequence of stages. TXL is typically used for program normalization (reduction of a program to a subset adhering to certain syntactic complexity), program optimization (improving the runtime and/or space performance of a program), refactoring/restructuring (altering the structure of a program to make it easier to understand) and program renovation. Although TXL is very efficient for tree based representations and/or transformations, it is not well suited for graph based intermediate representations (e.g., dataflow and/or control flow graph) which are mandatory in parallelizing compilers.

POET[88] is an open-source transformation scripting language. It is applicable to extensive parameterization of compiler optimizations for automatic performance tuning, empirical/interactive tuning of applications and domain-specific code generation. Recent publications employ POET in optimizing scientific codes[92,93] through source-level program transformations, examples of which include loop optimizations such as automatic parallelization, blocking, interchange, and fusion/fission, redundancy elimination

optimizations such as strength reduction of array address calculations, and data layout optimizations such as array copying and scalar replacement.

Cetus[90,94] is a source-to-source C compiler written in Java with an extensive set of compiler passes working on a high-level IR. It supports parallelization techniques, analysis and transformations: data dependence analysis, loop parallelizer, source program canonicalization and procedural abstraction of loops. Regarding its disadvantages, Cetus depends heavily on third-party open-source tools.

The Gecos (Generic Compiler Suite)[95] project is a source to source C compiler infrastructure leveraging Model Driven Software Design technologies. Gecos is a software component based framework that relies on the Eclipse/OSGI plugin infrastructure to be easily extensible. Among other features, Gecos provides most standard optimizing compiler passes, advanced tree matching/term rewriting facilities, and an easily customizable source code generation back-end. The main strength of Gecos however lies in its state of the art loop analysis and transformation/optimization toolbox. This toolbox is based on a polyhedral representation of loop nests, and offers a unified framework for expressing (and checking the legality of) very complex combinations of loop and data layout transformations. Gecos is open source software and is hosted on Inria GForge.

5.1.3 Data transfer and storage methodology / DTSE

When source-to-source transformations target parallelism, the ultimate goal[64] is to produce loops with iterations free of data and control dependencies in order to be easily assigned to different processing elements. On the other hand, optimizing towards the locality of reference involves transforming loop nests so that the consecutive memory references are moved close to each other in time (temporal locality) and memory address space (spatial locality). Traditionally, parallelism exposition comes first in existing methodologies in respect to the optimization of the memory locality.

Two possible ways exist for enhancing parallelism. The first regards physically exposing parallel segments of application code, a process requiring dynamic application statistics. The second refers to the automatic application of parallelized code transformations[64, 96]. This method requires exact dependence analysis and is not always possible in applications with dynamically allocated data structures, data dependent behaviour or non-linear array indexing.

Locality is enhanced by data layout transformations, i.e. by properly mapping data to the memory address space. A formalized methodology for optimizing locality is the Data Transfer and Storage Methodology (DTSE)[97]. The first step of the methodology is the application of loop transformations to increase locality irrespective of the underlying architecture. These types of transformations belong to the category of platform-independent optimizations. The second step includes platform-dependent transformations that either adapt code to be efficient for a specific memory architecture or produce an efficient custom memory hierarchy for the given application, the latter applicable when reconfigurability is an option. Currently, this flow is semi-automated with restrictions in respect to the form of mathematical expressions utilized for the indexing of data arrays. These restrictions can be compensated by the application of loop transformations manually by the designer. However, manual optimization requires an in-depth analysis of the source code in order to take efficient decisions.

5.1.4 Front-end optimization issues in numerical analysis language compilers

Scilab[98] as well as MATLAB[99] and GNU Octave[100] are dynamic numerical analysis languages that are usually accessible from within interpretive environments. For this reason, there was no initial need for developing strong optimizations at the intermediate code level. Recent trends show that there is increasing interest on native compilation of MATLAB-like languages, a task where both machine-independent and machine-dependent optimizations have an important role. Recurring themes of analysis and optimization aspects in MATLAB-like compilers include: type estimation/inference, array-aware data flow analyses, array storage layout, computation precision (selection of proper data types, bitwidth analysis), intraprocedural parallelization, and partial evaluation[101]. Relevant research projects include FALCON[102], MatParser[103, 104], TANOR[105], MATCH[106], MAT2C[107], OMPC[108] and McLab[109].

McLab[110] is a toolset aiming in the translation of a subset of MATLAB to interpretable bytecode named McVM[111]. McLab is an academic project regarding the development of an extensible compiler toolkit for MATLAB source code. The McLab approach applies JIT (Just-In Time) compilation to McLab IR, which can be either interpreted (McVM) or compiled to native code. The same project attempts to bridge the semantic differences between MATLAB and the traditional workhorse for scientific computation, Fortran (McFor), possibly the initial aim of the McLab team. Generally, McLab is not considered to involve strong optimizations; the authors state that their effort aims in delivering a complete MATLAB subset infrastructure for other research groups to experiment.

The McFLAT analysis and transformation engine[112] does provide some optimization facilities. The McFLAT engine computes dependence for loop statements and applies various transformations to improve program runtime. In MATLAB, loop bounds and array sizes are mostly determined at runtime, so the profiler component of McFLAT generates data for different runs of the program. Then it attempts to move most of the required computations (such as range estimations rather than range evaluations) at compile-time. Other challenging aspects of MATLAB compilation involve dynamic loading and typing, safe updates, and array copy semantics.

Internally, the McLab front-end first translates MATLAB to a clean subset of MATLAB called Natlab. The subset is then converted into a tree-based intermediate representation called McIR on which McFLAT operates. This is equivalent to the internal IR used by McVM, a simplified and transformed version of the original AST. This IR supports a high-level textual linearized form as well, resembling well-known dynamic garbage-collected imperative languages such as Perl and Ruby with dominant 3-address statements.

MatParser[103,104] automatically converts a specific class of MATLAB programs featuring complete nested loop programs (NLP) into single-assignment programs. Statement parallelism within nested loops is detected by solving a Parametric Integer Programming (PIP) problem for every data dependency, which should be very slow and unsuitable for large programs. The supported NLPs present certain limitations such as purely static control and affine linear expressions. Thus, MatParser is not able to parse and compile realistic MATLAB-like programs.

FALCON[102] is a relevant work dealing with MATLAB code restructuring. FALCON performs static, dynamic and interactive (user-driven) analyses on a convenient IR in order to generate Fortran 90 programs with directives for parallelism. It includes capabilities for interactive and automatic transformations at both the operation-level and the functional- or algorithm-level such as algebraic restructurings, textbook loop optimizations (e.g. loop interchange). Due to the availability of a limited data dependence analyzer, FALCON uses annotations to express even intra-task parallelism.

TANOR[105] is an automated framework for designing hardware accelerators for numerical computation on reconfigurable platforms and especially for partially evaluated versions of N-body simulation codes. It incorporates an optimization engine termed as “Algorithm Optimizer”. The code parser and analyzer phase transforms the input specifications into an IR termed Abstract Syntax Graph (ASG). The algorithm optimizer transforms the ASG using function evaluation, interaction matrix tiling or clustering, and intra- and inter-cluster data traversal. Function evaluation regards the optimization of transcendental function evaluation via Taylor or minimax polynomials. Interaction matrix tiling refers to matrix partitioning schemes for enhancing cache performance. Data traversal regards data layout reorderings for addressing optimization.

One of the first open-source MATLAB frontends was MATCH[106]. Interestingly enough, the corresponding grammar (equivalent to a MATLAB 5 subset) has been originally derived from an old version of Octave rather than directly reverse engineering the MATLAB specification. The MATCH IR is an AST that can be mapped to a collection of CFGs. MATCH is used as the frontend of the MAT2C[107] compiler.

MAT2C[107, 113] is an optimizing source-to-source translator system that converts MATLAB programs to stand-alone C code. The system consists of four main parts: (1) a custom frontend that parses MATLAB M-files into an intermediate representation (IR); (2) a type inference engine called MAGICA[114] that takes an encoding of the IR, infers the intrinsic types, array shapes and value ranges of the various represented MATLAB expressions, and decorates the IR with the inferred information; (3) scalar optimization phases and (4) a C back-end code generator.

OMPC[108, 115] is an open-source MATLAB-to-Python compiler that uses syntactic transformations and functionality emulation to transform MATLAB into Python programs. The resulting Python programs are not self-contained, but rather carry a number of dependencies. OMPC implements numerical objects emulating the dynamic behaviour of their MATLAB counterparts. The dynamic features of the MATLAB engine differ from those of Python as well as most other general-purpose interpreters: array slicing, on-demand updating of the variable namespace, element-wise operations, and implied returns are not present in Python in the same form.

The authors of OMPC have developed an approach to dealing with modelling and exploiting memory locality of applications to guide compiler optimizations. A source-level metric named static reuse distance has been introduced, to model the memory behaviour of applications written in MATLAB. Then, an efficient algorithm is used to compute static reuse distances using an extended version of dependence graphs. The authors state that their approach works on sizable programs.

5.2 Coarse grained parallelism extraction and optimization

The coarse grain parallelism extraction and optimization phase assumes that chunks of the program large enough relative to the size of the entire program are identified as tasks. For the problem setting discussed here an intermediate representation of the serial program in the form of a Control and Data Flow Graph (CDFG) is provided as input. The scheduling problem that should be solved is the spatial and temporal assignment of tasks to processors. The spatial assignment also known as mapping is the allocation of tasks to the processors, while the temporal assignment is the attribution of a start time to each task. Since control dependencies can be transformed to data dependencies, CDFG is actually a Directed Acyclic Graph (DAG) where each node is a task and each edge represents a data dependency between tasks, meaning that the execution of the task at the source of the edge should be completed before the task at the target of the edge starts executing. Moreover, when a task runs on a certain processor and the dependent task is scheduled to run on a

different processor then a communication cost should be applied postponing the start of execution of the dependent task until the arrival of the required data. Consequently, task scheduling can be seen as a trade-off between minimizing interprocessor communication costs and maximizing the concurrency of the task execution. Before analyzing works on coarse grained parallelism, it is important to understand that limits on automatic parallelism do exist and that various types of dependencies can affect the parallelism extraction process[116, 117].

Two fundamental heuristics exist for the problem of task scheduling: list scheduling and clustering[118]. List scheduling sorts the nodes of the task graph by giving a priority value to each node while respecting the topological order implied by the graph. Then, in turn each node of the sorted list is scheduled to a processor chosen based on a strategy trying to minimize the make-span (time needed for completion) of the task graph. On the other hand, clustering groups together nodes that should be executed on the same processor. The goal of clustering is to provide a set of clusters with minimal data dependencies between them. Tasks belonging to different clusters can run concurrently, thus increasing the average degree of concurrency. Three steps can be identified in clustering based algorithms: (1) clustering to an unlimited number of clusters, (2) mapping of clusters to the processors and (3) scheduling of the nodes. A relevant work by Beg[119] presents a clustering heuristic that partitions the task graph in order to assign computational workload on cores of a multi-core system. The system is based on the Trimaran compiler framework[44] and its main feature is to identify the critical path for the code, which is the largest instruction sequence that needs to be assigned to a single core. The length of the critical path is actually the limit defined by Amdahl's law[120].

In combination with the above mentioned heuristics two techniques can be used: the insertion technique and node duplication. With insertion technique a task can be scheduled earlier than already scheduled tasks on the same processor provided that a sufficient gap exists and no dependency is violated. On the other hand, with node duplication one node can be scheduled to run to more than one processor so as to save communication costs.

Although the goal of the ALMA project is to find an optimal partition and scheduling of the task graph, it is apparent that if suboptimal solutions can be acquired very fast they can also be very useful from the end user perspective as they can be used to get a very good estimate of the impact of the changes the end user makes to an evolving code during the development cycle. Heuristic and Meta-heuristic randomized algorithms can be used to get solutions in a timely manner. These solutions can be used as a rough estimate or as initial solutions to an exact algorithm that creates an upper bound to the solution quality.

Various single-path meta-heuristic methods as well as population-based meta-heuristics are going to be investigated to find a combination that will produce near-optimal solution to the tasks graph partitioning, mapping and scheduling to specific cores. Single path meta-heuristics are techniques to search the feasible solution space and use one heuristic method called neighbourhood to define the next move and another heuristic method to accept the next move or not. Interesting single-path meta-heuristics include Simulated Annealing[121], Tabu Search[122, 123, 124], Great Deluge[125, 126] and Late Acceptance Hill Climbing[127, 128]. Population-based meta-heuristics maintain a population of candidate solutions and perform various solution combination heuristics on subsets of the population in order to create a new generation of solutions. A comparison for the use of various heuristic methods, including single path and population based meta-heuristics, for solving the tasks planning problem to heterogeneous multiprocessors is presented in Braun et al[129]. Population based meta-heuristics include Genetic Algorithms[130] and Scatter Search[131,132]. Bio-inspired algorithms, like Ant Colony Optimization[133,134] and Artificial Bee Colony[135] try to imitate biological colonies and leverage the collective cooperative intelligence they demonstrate. Meta-heuristics and Bio-inspired algorithms are good at finding a good solution, but they usually are unable to

find the optimal one. Also, they can suffer from getting trapped in local optima in terms of the solution space shape. For population-based meta-heuristics this trap is expressed by a reduced diversity between the members of the population, fact that traps the algorithm within a narrow area of the feasible solution space. In ALMA, we intend to employ the above search methods in a hyper-heuristics like framework, where the various techniques are going to be activated according to the solution progress.

Ferrandi et al.[136] propose an Ant Colony Optimization to optimize Hierarchical Task Graphs for MPSoC parallel applications. Li et al[137] propose a population-based meta-heuristics approach for the mapping and scheduling problems. More specifically, they employ a Genetic Algorithm for the task assignment problem. In order to solve the task mapping problem, they use Ant Colony and Cooperative Ant Colony approaches.

On the other hand, Mixed Integer Linear Programming formulation of problems provides a deterministic and provably optimum solution identification method to solve optimization problems and has been utilized to solve the problem of automated parallelization of embedded software[138]. Since it was not feasible to solve the whole problem to optimality, authors of this work have used the Hierarchical Task Graph[139] in order to effectively partition the problem. Also, relevant to the ALMA coarse grained parallelism extraction and optimization is the work of Luis et al[140]. They use a Data Flow Graph and a Control Flow Graph to extract a Petri-net representation of the code, they use Girkar's Algorithm[141,142] to identify essential dependences and eliminate unnecessary synchronization. The behaviour of the optimization technique allows for simple partitioning schemes to be employed in order to increase the granularity of the parallelism extraction. Liu et al.[143] present a graph partitioning heuristic algorithm for parallel task assignment. Their partitioning algorithm is based on the Kernighan and Lin partitioning algorithm[144]. Yi et al.[145] present an ILP formulation for task mapping and scheduling. Their technique incorporates profiling driven loop level partitioning, task transformations, functional pipelining and memory architecture aware data mapping.

An important aspect that should also be considered in the ALMA coarse grained optimization phase, is the fact that even a very detailed architecture description that provides an accurate temporal representation of individual instructions execution and network communication time implications can not really represent the real behaviour of a parallel program. Tournavitis et al.[146] identify that traditional target-specific mapping heuristics are inflexible and propose a machine learning based prediction mechanism for the mapping decisions. Also, they identify that static analysis proves inadequate and use profile-driven parallelism detection. Rul et al.[147,148] present a framework to extract parallelism from sequential code based on profile information, more specifically the interprocedural data flow graph and the data-sharing graph. After parallelism detection, they employ a set of parallel constructs as templates for the generated code and their detection mechanism is able to identify and use the appropriate construct. We intend to integrate the feedback information from the architecture simulator in the partitioning and scheduling algorithms in order to incrementally optimize the actual parallel program performance. Hendrickson et al. [149] observe that the standard graph partitioning algorithms actually minimizes the wrong metric and lack expressibility. In their work, they survey several proposed alternatives.

Some recent contributions to the problem are presented bellow. Huang et al.[150] approaches the mapping of application task graphs onto reconfigurable hardware by incorporating multiple architectural variants for each task reflecting tradeoffs between the resources consumed and the task execution throughput. Then, the mapping process is completed employing a genetic algorithm. Singh and Singh [151] also propose a genetic algorithm approach that simultaneously tries to optimize mapping of the tasks, optimize sequence of execution and also search for an optimal configuration of the parallel system. A work that is highly relevant to the ALMA project is the paper by Prasad et al.[152]. In this contribution the design and implementation of the MEGHA compiler is described. MEGHA automatically

compiles MATLAB program to enable synergistic execution on heterogeneous processors. Data parallel regions of the program are identified and composed into kernels by solving a constrained graph clustering problem. Then heuristics map the identified kernels to CPU or GPU. Ferrandi et al.[153] propose an approach based on Ant Colony Optimization that explores different alternative designs to determine an efficient hardware-software partitioning, to perform the mapping and to establish an execution order of the tasks. Authors claim that their approach can be applied to any parallel C application represented through Hierarchical Task Graphs.

5.3 Fine grained parallelism extraction

The approach for fine-grained parallelization in the ALMA project combines techniques from loop transformation and parallelization with operator word length optimization techniques. In the following we detail the most recent research works on the topic and place them into perspective with respect to the ALMA project goals.

5.3.1 Loop vectorization for SIMD parallelization

Regular and repetitive computation patterns (such as those found in vector processing languages such as scilab) are known to be interesting candidates for fine grained parallelism extraction (and in particular short width vectorization), as their regular structure makes them amenable to advanced analyses and transformations, such as those offered by the polyhedral model.

These analysis and transformation techniques leverage on theoretical foundations that date back to the early 80s. However, this model only recently found its way within optimizing and parallelizing compilers. This increasing adoption is mainly due to a recent renewing of interest for automatic parallelization techniques, mainly motivated by the outbreak of multi-core architectures. This revival led to many breakthroughs (both practical and theoretical) over the last few years[154, 155, 156, 157, 158].

The core idea behind the polyhedral model is to provide a compact iteration wise representation of static control loop nests using integer polyhedrons. The approach supports imperfectly nested loops with affine array accesses and affine guards (Static Control Parts). The approach enables powerful analyses and transformations ranging from data locality optimizations to automatic parallelization.

In particular, it is widely accepted that optimizing a program for parallelism without considering data locality leads to an inefficient execution scheme. A combined approach is therefore mandatory to actually improve performance. In addition, parallelizing for short vector with SIMD extension poses several other challenges, which are due to the tight architectural constraint surrounding these instruction sets, and for which standard code selection technique (e.g., those based on dynamic programming) cannot be applied.

Taking advantage of such instruction sets hence requires combining loop transformations with lower level optimizations. Such an approach was followed by Trifunovic et al.[159], where they examined the interactions between loop transformations of the polyhedral framework and subsequent vectorization back-ends. In particular they modelled the performance impact of various loop transformations and vectorization strategies, and showed that this model could be integrated within a polyhedral compilation framework.

More recently, Kong et al.[160] introduced what are called “vectorizable codelets” to better interface the back-end vectorizer and the high-level loop transformation framework. These codelets are defined to have good properties to enable efficient vectorization, and serves as the interface between the two components. The polyhedral framework is used to transform

the loops such that codelets are exposed. This is one step towards tighter integration between low level optimization and loop transformations.

An important point is that these vector instructions operate on a specific vector register file, whose word length can vary from 64 bits (MMX) to 256 bits (Intel AVX). These vector registers only support a restricted subset of memory accesses. For example, a vector register can only read/write from a contiguous set of memory location. Many SIMD ISA (Instruction Set Architecture), even further constraint the addressing by only enabling addresses whose value is aligned with the size of the SIMD vectors. To overcome these limitations, SIMD ISA extension offers numerous shuffling/packing instructions that can be used to reorganize the data within vector registers. While extending the usability of the vector extensions, these extra instructions cause performance overhead, and can make SIMD less efficient than a pure scalar based code.

Since contiguity in memory accesses has significant impact on performance, vectorizers often seek to find groups of operations with contiguous memory accesses. Liu et al.[161] observed that such grouping may lead to poor reuse of the vector registers. They proposed to first group the operations by maximizing the reuse of vector registers, and then using data layout transformation to make array accesses contiguous.

The work by Henretty et al.[162] has studied how complex non-linear memory layout optimizations could be used to arrange data in memory so as to reduce and/or eliminate non-unit stride memory accesses for stencil like computation patterns. While targeted at a very specific application context, the ideas proposed in this work could be easily extended to a wider class of kernels.

A somewhat similar approach was followed by Nuzman et al.[163] and Franchetti et al.[164], who propose to extend the support of SIMD to non-unit stride memory access patterns (their approach only considers power of two constant stride values) by a combination of elementary shuffling/packing operations, and data layout reorganization. Their results show that small non-unit stride access pattern can still lead to an efficient SIMD vectorization.

While being very effective in practice, the applicability of this kind of optimizations is very limited, as in many imperative languages (e.g., C/C++) the layout of the data in memory is considered as being part of the semantic of the program. Hence, the compiler is not allowed to modify it. This is not the case in scilab, where no memory layout is imposed a priori. Such flexibility opens many interesting layout optimization opportunities.

5.3.2 Data encoding exploration and accuracy evaluation

Because scilab operates in floating-point arithmetic, mapping a scilab program onto an embedded platform supporting only integer arithmetic involves a floating-point to integer or fixed-point conversion stage. The choice of the encoding for each variable is very important as it will enable (or hinder) the possibility of using sub-word parallelism offered by the SIMD units. For example, the ability to encode on 8-bit fixed-point arithmetic can lead to twice as much performance over a 16-bit encoding.

The limited choice of word length (only power of two of bytes) in most SIMD processors somewhat reduces the encoding feasible space and may limit the impact of the accuracy exploration stage. To tackle this problem, several approaches have been proposed. For example, Lambrechts et al.[165] propose to resort to subtler encoding by using what is known as Software SIMD. The idea is to emulate through software a custom SIMD operation, where the size of each vector element can be chosen to suit the designer needs. Using this scheme it is possible to efficiently implement constant coefficient multiplications and additions. The main idea consists in adding a so-called guard bit into the encoding so as to prevent carry propagation and to isolate vector elements from each other during the arithmetic operation.

While initially targeted at processors without SIMD extension, software and hardware SIMD can be combined together to provide a more fine grained fixed-point arithmetic support.

A flexible and configurable SIMD operator supporting more data types than classical SIMD processors is proposed by Khan et al.[166,167] This operator can execute different operations often used in multimedia applications. Thus, the supported wordlength match the bit size of low precision pixel data. This operator is able to execute five operations on 8-bit data, four operations on 10-bit data, three operations on 12 bit data, two operations on 16-bit data and one operation on 40-bit data.

Although many encodings are possible, the most critical part in such an approach lies in the numerical accuracy evaluation, mainly because this evaluation is involved in the iterative word-length optimization process. This numerical accuracy can be evaluated using analytical or simulation-based approaches.

In fixed-point simulation-based approaches, the application performance and the numerical accuracy are directly determined from fixed-point (bit-true) simulations. To obtain accurate results, the simulations must use a high number of samples. To reduce word-length optimization time, approaches based on fixed-point simulations attempt to reduce the time to emulate fixed-point arithmetic by using efficient data types. Most of them use object oriented language and operator overloading[168] to implement fixed-point arithmetic. However, the fixed-point execution time of an algorithm is significantly longer than the one of a floating-point simulation[169].

The general idea of analytical approaches is to determine a mathematical estimation of the output quantization noise power. The computation of this analytical expression could be time-consuming, but it is executed only once. Then, the numerical accuracy is determined very quickly by evaluating the mathematical expression for a given word-length of each data. Existing analytical approaches for numerical accuracy evaluation are based on perturbation theory[170,171]. The challenge is to compute the gain between each noise source and the output. Approaches based on hybrid techniques[172,173,174] have been proposed to compute the gains from a set of fixed-point simulations. In the approaches based on affine arithmetic[175], the gains are computed with an affine arithmetic based simulation. It was proposed for Linear Time-Invariant (LTI) systems in [176] and recently, for non-LTI systems in [177]. In [178,179] the gains are obtained from the impulse response of the system between the output and each noise source due to bit elimination.

6. Conclusions

This document provides the ALMA project specific state-of-the-art. First, the ALMA project in the context of the ARTEMIS and HIPEAC roadmap is discussed. Following, an overview of the existing design methodologies for multi-core embedded systems is presented. An overview of the area for techniques for generating parallel code for multi-core architectures is presented. Optimization techniques for intermediate code mainly with respect to data transfers are discussed, as well as coarse and fine grained parallelism extraction techniques and optimization. The investigation in the state of the art underlined that ALMA targets a crucial topic which is currently not solved. The consortium is very confident, that the envisioned approach in ALMA will be a step ahead for the programming of complex multiprocessor systems and due to this, a highly improved exploitation of these novel hardware architectures can exist for many users.

7. References

[1] High level vision ITEA-ARTEMIS 2030. Available at: http://www.artemis-ju.eu/publications

[2] The HiPEAC vision for Advanced Computing in Horizon 2020. Available at: http://www.hipeac.net/roadmap

[3] Graham, S.L., Kessler, P.B., McKusick, M.K. “gprof: A Call Graph Execution Profiler”, Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

[4] MathWorks, “Real-Time Workshop Embedded Coder”. Available at:

http://www.mathworks.de

[5] Pellerin D., S. Thibault,“Practical FPGA Programming in C”, Prentice Hall,Professional Technical Reference, 2005.

[6] Fingeroff M.,“High-Level Synthesis Blue Book”; Xlibris Corp, May 21st, 2010.

[7] Datta, Kushal, An efficient design space exploration framework to optimize power-efficient heterogeneous many-core multi-threading embedded processor architectures, Dissertation,

University of North Carolina at Charlotte, 2011

[8] Trish Messiter, Complexities of embedded systems, whitepaper, Clarinox Technologies Pty Ltd, 2011

[9] F. Vahid and T. Givargis. Timing is Everything -- Embedded Systems Demand Teaching of Structured Time-Oriented Programming . Int. Wkshp. on Embedded Systems Education, (WESE), Oct 2008

[10] Sousa, L., Alegria, F., Germano, J., Developing and Integrating Lab Projects as Important Learning Components in an Embedded Systems Course, Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on June 2007

[11] Crnkovic, I., Component-based software engineering for embedded systems, ICSE '05 Proceedings of the 27th international conference on Software engineering, New York, Malardalen Univ., Vasteras, Sweden, 2005

[12] SMP, Asymmetric Multi- processing And The HSA Foundation, HSA Foundation , 2012, www.chipdesignmag.com/sld/shuler/2012/09/27/smp-asymmetric-multiprocessing-and-the-hsa-foundation

[13] International Technology Roadmap for Semiconductors (ITRS), www.itrs.net

[14] Berkel, C.H. van; Multi-core for mobile phones; Design, Automation and Test in Europe Conference & Exhibition 2009, Nice, France, April 20-24, 2009. (pp. 1260-1265). IEEE. Invited paper; April, 2009

[15] Blake, G., Univ. of Michigan, Ann Arbor, MI, USA, Dreslinski, R.G., Mudge, T., A survey of multicore processors, Whitepaper, Signal Processing Magazine, IEEE (Volume:26 , Issue: 6), 2009

[16] MPPA 256, Kalray, www.kalray.eu

[17] TILE-Gx™ processor family, Tilera, http://www.tilera.com

[18] Epiphany-IV Microprocessor, Adapteva, www.adapteva.com

[19] CRISP: Cutting edge Reconfigurable ICs for Stream Processing project, www.crisp-project.eu

[20] Cardoso, João M. P., Hübner, Michael (Eds.), “Reconfigurable Computing - From FPGAs to Hardware/Software Codesign”, Springer, 2011.

[21] Recore Systems, 2011, www.recoresystems.com/technology/xentium-technology/

[22] Karel H.G. Walters, S.H. Gerez, GJ.M. Smit, S. Baillou, G.K. Rauwerda, R. Trautner Multicore SoC for On-Board Payload Signal Processing; Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference, San Diego CA, United States, June 6-9, 2011.

[23] MPI: A Message-Passing Interface, www.mpi-forum.org

[24] LINX Interprocess Communication (IPC), Enea, Sweden, www.enea.com/solutions/middleware/interprocess-communication

[25] MCAPI, Multicore Communications API, www.multicore-association.org/workgroup/mcapi.php

[26] Poly-Platform™, PolyCore Software, California, United states, polycoresoftware.com/solutions/products/poly-platform

[27] MP Designer™, Target Compiler Technologies, Leuven, Belgium, www.retarget.com/products/mpdesigner.php

[28] OpenCL, Khronos group, www.khronos.org/opencl

[29] Maxim Shevtsov, OpenCL: the advantages of heterogeneous approach, whitepaper, Intel, 2013

[30] Multicore Association, California, United states, www.multicore-association.org

[31] UPC - Unified Parallel C, www.upc.gwu.edu/

[32] OpenACC Application Program Interface, www.openacc-standard.org

[33] OpenMP, www.openmp.org

[34] Open HMPP, Open Hybrid Multicore Parallel Programming, CAPS, Rennes, France, www.caps-entreprise.com/openhmpp-directives

[35] The Barrelfish Operating System, http://www.barrelfish.org/

[36] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri and L. Benini, “An Integrated Open Framework for Heterogenous MPSoC Design Space Exploration”, in Proceedings of the Design, Automation and Test in Europe (DATE) conference. Munich, 2006

[37] SAE International, The Architecture Analysis and Design Language Standard, http://standards.sae.org/as5506b/

[38] Accellera Systems Initiative, http://www.accellera.org

[39] Sandro Rigo, Guido Araujo, Marcus Bartholomeu, and Rodolfo Azevedo, “ ArchC: A SystemC-Based Architecture Description Language.”, In 16th Symposium on Computer Architecture and High Performance, Computing (SBAC-PAD 2004), 27-29 October 2004, Foz do Iguacu, Brazil, pages 66–73. IEEE Computer Society, 2004.

[40] SoClib, http://www.soclib.fr/trac/dev

[41] Srikant Y. and P. Shankar, “The compiler design handbook – optimizations and machine code generation”, CRC Press, ISBN 978-1-4200-4382-2,2008.

[42] GCC, the GNU Compiler Collection, http://gcc.gnu.org/

[43] The LLVM Compiler Infrastructure, http://www.llvm.org/

[44] Trimaran Consortium. An Infrastructure for Research in Backend Compilation and Architecture Exploration. http://www.trimaran.org/

[45] Open64 compiler, http://www.open64.net/

[46] Wen-mei W. Hwu , Scott A. Mahlke , William Y. Chen , Pohua P. Chang , Nancy J. Warter , Roger A. Bringmann , Roland G. Ouellette , Richard E. Hank , Tokuzo Kiyohara , Grant E. Haab , John G. Holm , Daniel M. Lavery, “The Superblock: An effective technique for VLIW and superscalar compilation”, THE JOURNAL OF SUPERCOMPUTING, 1993.

[47] Fisher, J.A., “Trace Scheduling: A Technique for Global Microcode Compaction”, Computers, IEEE Transactions on , vol.C-30, no.7, pp.478-490, July 1981.

[48] Mahlke, S.A.; Lin, D.C.; Chen, W.Y.; Hank, R.E.; Bringmann, R.A., "Effective Compiler Support For Predicated Execution Using The Hyperblock", Microarchitecture, 1992. MICRO 25., Proceedings of the 25th Annual International Symposium on , vol., no., pp.45-54, 1-4 Dec 1992.

[49] Ramakrishna Rau. B. “Iterative modulo scheduling: an algorithm for software pipelining loops”, In Proceedings of the 27th annual international symposium on Microarchitecture (MICRO 27). ACM, New York, NY, USA, 63-74. DOI=10.1145/192724.192731, 1994.

[50] Ellis J. R., “Bulldog: a compiler for vliw architectures”, Ph.D. dissertation, Yale University, New Haven, CT, USA, aAI8600982, 1985.

[51] Desoli G., “Instruction assignment for clustered vliw dsp compilers: A new approach,” HP Laboratories Cambridge, Tech. Rep., 1998.

[52] Chu M., K. Fan, and S. Mahlke, “Region-based hierarchical operation partitioning for multicluster processors,” in Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, ser. PLDI ’03. New York, NY, USA: ACM, pp. 300–311, 2003.

[53] Oezer E., S. Banerjia, and T. M. Conte, “Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures”, in Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture, ser. MICRO 31. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 308–315, 1998.

[54] Nagpal R. and Y. N. Srikant, “Pragmatic integrated scheduling for clustered vliw architectures”, Software: Practice & Experiance, vol. 38, no. 3, pp. 227–257, 2008.

[55] Eriksson M. V. and C. W. Kessler, “Integrated modulo scheduling for clustered vliw architectures,” in Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers, ser. HiPEAC ’09. Berlin, Heidelberg: Springer-Verlag, pp. 65–79, 2009.

[56] Capitanio A., N. Dutt, and A. Nicolau, “Partitioned register files for vliws: a preliminary analysis of tradeoffs”, in Proceedings of the 25th annual international symposium on Microarchitecture, ser. MICRO 25. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 292–300, 1992.

[57] Brisolara Lisane, Sang-il Han, Xavier Guerin, Luigi Carro, Ricardo Reis, Soo-Ik Chae, and Ahmed Jerraya., “Reducing fine-grain communication overhead in multithread code generation for heterogeneous MPSoC”, In Proceedingsof the 10th international workshop on Software & compilers for embedded systems (SCOPES '07), Heiko Falk and Peter Marwedel (Eds.). ACM, New York, NY, USA, 81-89. DOI=10.1145/1269843.1269855, 2007.

[58] Palermo, D.J., Su, E., Chandy, J.A., Banerjee, P. "Communication Optimizations Used in the Paradigm Compiler for Distributed-Memory Multicomputers", Parallel Processing, ICPP 1994, International Conference on , vol.2, no., pp.1-10, 15-19 Aug. 1994, 1994.

[59] Koop, M.J.; Jones, T.; Panda, D.K., "Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach", *Cluster Computing and the Grid, CCGRID 2007. Seventh IEEE International Symposium on* , vol., no., pp.495-504, 14-17 May 2007, 2007.

[60] Hiranandani S., K. Kennedy, C. Tseng, “Compiling Fortran D for MIMD Distributed Memory Machines”, Commun. ACM v. 35, n.8, pp. 66-80. 1992.

[61] Guangyu Chen, Feihui Li, and Mahmut Kandemir, “Compiler-directed channel allocation for saving power in on-chip networks”, SIGPLAN Not. 41, 1 (January 2006), pp. 194-205, 2006.

[62] Conway M.E., “Proposal for an UNCOL”, Communications of the ACM, Vol. 1, No. 10, pp. 5-8, 1958.

[63] Click Cliff and Michael Paleczny, “A simple graph-based intermediate representation”, ACM SIGPLAN workshop on Intermediate representations, pp. 35-49, San Francisco, California, United States, 1995.

[64] Allen Randy and Ken Kennedy, “Optimizing Compilers for Modern Architectures”, Morgan Kauffman, 2001.

[65] Aho A. V., M.S. Lam, R. Sethi and J. D. Ullman, “Compilers: Principles, Techniques, and Tools”, 2nd edition, Pearson Education, Boston, MA, USA, 2006.

[66] Bacon David F., Susan L. Graham, and Oliver J. Sharp, “Compiler Transformations for High-Performance Computing”, ACM Computing Surveys, Vol. 26, No. 4, pp. 345-420, 1994.

[67] Merrill Jason, “GENERIC and GIMPLE: A New Tree Representation for Entire Functions”, GCC Developers Summit, pp. 171-180, 2003.

[68] Appel A. W., “SSA is functional programming”, ACM SIGPLAN Notices, Vol. 33, No. 4, pp. 17–20, Apr 1998.

[69] Cytron R., J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck, “Efficiently computing static single assignment form and the control dependence graph”, ACM Transactions on Programming Languages and Systems, Vol. 13, No. 4, pp. 451-490, October 1991.

[70] GCC GRAPHITE: GIMPLE represented as polyhedra. http://gcc.gnu.org/wiki/Graphite

[71] Pop S., A. Cogen, C. Bastoul, S. Girbal, G.-A. Silber, and N. Vasilache, “GRAPHITE: Polyhedral Analyses and Optimizations for GCC”, GCC Summit 2006, Ottawa, Canada, 2006.

[72] clang LLVM C/C++ frontend. http://clang.llvm.org

[73] Grosser Tobias, Hongbin Zheng, Ragesh Aloor, Andreas Simbuerger, Armin Groesslinger, Louis-Noël Pouchet, “Polly – Polyhedral Compilation in LLVM”, First International Workshop on Polyhedral Compilation Techniques (IMPACT 2011), Chamonix, France, April 3, 2011.

[74] OpenMP. http://www.openmp.org

[75] Nvidia PTX 2.3 specification,

http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/ptx_isa_2.3.pdf

[76] COINS: A COmpiler Infra-Structure home page. http://www.coins-project.org

[77] Leupers R., O. Wahlen, M. Hohenauer, T. Kogel, and P. Marwedel, “An Executable Intermediate Representation for Retargetable Compilation and High-Level Code Optimization,” in Int. Conf. on Inf. Comm. Tech. in Education, June 2003.

[78] LANCE C compiler. http://www.lancecompiler.com

[79] Machine-SUIF research compiler. http://www.eecs.harvard.edu/hube/research/machsuif.html

[80] SUIF compiler. http://suif.stanford.edu

[81] Phoenix compiler infrastructure. https://connect.microsoft.com/Phoenix

[82] Microsoft home page. http://www.microsoft.com

[83] Safonov Vladimir I, “Trustworthy Compilers”, John Wiley and Sons, 2010.

[84] Associated Compiler Exprerts, CoSy Compiler Development System. http://www.ace.nl

[85] The Program Transformation website. http://www.program-transformation.org

[86] DMS. http://www.semdesigns.com/products/DMS/DMSToolkit.html

[87] The Tree Transformation Language (TXL) homepage. http://www.txl.ca

[88] POET transformation tool. http://bigbend.cs.utsa.edu/poet/index.php

[89] ROSE Compiler. http://www.rosecompiler.org

[90] The Cetus compiler infrastructure. http://cetus.ecn.purdue.edu/

[91] Edison Design Group. http://www.edg.com

[92] Qing Yi, “Optimizing and tuning scientific codes”, in Scalable computing and communications: Theory and Practice, Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya (eds.), John Wiley&Sons. 2011.

[93] Qing Yi, “POET: A Scripting Language For Applying Parameterized Source-to-source Program Transformations”, Software Practice & Experience, accepted for publications in 2011.

[94] Dave C., H. Bae, S.-J. Min, S. Lee, R. Eigenmann, and S. Midki, “Cetus: A source-to-source compiler infrastructure for multicores”, IEEE Computer, Vol. 42, No. 36, 2009.

[95] Gecos, http://gecos.gforge.inria.fr/doku.php

[96] Banerjee Utpal, “Loop Parallelization”, Springer, 1994.

[97] Catthoor F., K. Danckaert, K.K. Kulkarni, E. Brockmeyer, P.G. Kjeldsberg, T. van Achteren, and T. Omnes, “Data Access and Storage Management for Embedded Programmable Processors”, Springer Academic Publishers, 2002.

[98] Scilab. http://www.scilab.org

[99] Mathworks Corporation. http://www.mathworks.com

[100] GNU Octave. http://www.gnu.org/software/octave/

[101] DeRose Luiz A., “Compiler techniques for MATLAB programs”, Ph.D thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1996.

[102] DeRose L., K. Gallivan, E. Gallopoulos, B. Marsolf, and D. Padua, “FALCON: A MATLAB interactive restructuring compiler”, 8th International Workshop on Languages and Compilers for Parallel Computing (LCPC’95), pp.269-288, Columbus, OH, USA, Aug. 1995.

[103] MatParser. http://www.liacs.nl/~cserc/compaan/matparser.htm

[104] Kienhuis Bart, “MatParser: An array dataflow analysis compiler”, Technical report UCB/ERL M00/9, Department EECS, University of California at Berkeley, Berkeley, CA, USA, Feb. 2000.

[105] Jung Kim Sub, Deng Lanping, P. Mangalagiri, K. Irick, K. Sobti, M. Kandemir, V. Narayanan, C. Chakrabarti, N. Pitsianis, and Sun Xiaobai, “An Automated Framework for Accelerating Numerical Algorithms on Reconfigurable Platforms Using Algorithmic/Architectural Optimization”, IEEE Transactions on Computers, Vol. 58, No. 12, pp. 1654-1667, Dec. 2009.

[106] Joisha Pramod G., Abhay Kanhere, Prithviraj Banerjee, U. Nagaraj Shenoy and Alok Choudhary, “The Design and Implementation of a Parser and Scanner for the MATLAB Language in the MATCH Compiler”, Technical Report CPDC-TR-9909-017, Department of Electrical and Computer Engineering, Northwestern University, September 1999.

[107] The MAT2C homepage.

http://www.ece.northwestern.edu/cpdc/pjoisha/MAT2C/

[108] OMPC open-source MATLAB-to-Python translator. http://ompc.juricap.com/

[109] McLAB project. http://www.sable.mcgill.ca/mclab/

[110] Casey Andrew, Jun Li, Jesse Doherty, Maxime Chevalier-Boisvert, Toheed Aslam, Anton Dubrau, Nurudeen Lameed, Amina Aslam, Rahul Garg, Soroush Radpour, Olivier Savary Belanger, Laurie Hendren, and Clark Verbrugge, “McLab: An extensible compiler toolkit for MATLAB and related languages”, C3S2E '10, Montreal, Canada, May 2010.

[111] Chevalier-Boisvert Maxime, “McVM: an Optimizing Virtual Machine for the MATLAB Programming Language”, Master’s thesis, McGill University, Montreal, Canada, Aug. 2009.

[112] Aslam Amina and Laurie Hendren, “McFLAT: A Profile-based Framework for MATLAB Loop Analysis and Transformations”, LCPC 2010, Houston, Texas, USA, October 2010.

[113] Joisha Pramod G. and Prithviraj Banerjee, “MAGICA: A Software Tool for Inferring Types in MATLAB”, Technical Report CPDC-TR-2002-10-004, Department of Electrical and Computer Engineering, Northwestern University, October 2002.

[114] MAGICA: A type inference engine for MATLAB. http://www.ece.northwestern.edu/cpdc/pjoisha/MAGICA/

[115] Jurica Peter and Cees van Leeuwen, “OMPC: an open-source MATLAB-to-Python compiler”, Frontiers in Neuroinformatics, Feb. 2009.

[116] Wall D. W.. “Limits of instruction-level parallelism”, In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), volume 26, pages 176–189, New York, NY, ACM Press, 1991.

[117] Mak Jonathan and Alan Mycroft, “Limits of Parallelism Using Dynamic Dependency Graphs”, WODA ’09, July 20, Chicago, Illinois, USA, 2009.

[118] O. Sinnen, *Task Scheduling for Parallel Systems*. John Wiley & Sons, 2007

[119] Beg Mirza, “Critical Path Heuristic for Automatic Parallelization”, Technical Report CS-2008-16, David R. Cheriton School of Computer Science, University of Waterloo, August 2008

[120] Amdahl Gene M., “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”, pages 483–485, Reston, VA, USA, 1967.

[121] Kirkpatrick, S.; Gelatt, C. D. Jr. and Vecchi, M. P. “Optimization by simulated annealing”, Science, Vol. 200, No. 4598, pp. 671-680, 1983.

[122] Glover Fred, "Tabu Search - Part 1", ORSA Journal on Computing 1 (2): 190-206, 1989.

[123] Glover Fred, "Tabu Search - Part 2". ORSA Journal on Computing 2 (1): 4-32, 1990.

[124] Fred Glover (1990). "Tabu Search: A Tutorial", Interfaces, vol. 20, no. 4, pp. 74-94, doi: 10.1287/inte.20.4.74, 1990.

[125] Dueck Gunter, "New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel", Technical report, IBM Germany, Heidelberg Scientific Center, 1990.

[126] Dueck Gunter, "New Optimization Heuristics The Great Deluge Algorithm and the Record-to-Record Travel", Journal of Computational Physics, Volume 104, Issue 1, pp. 86-92, 1993.

[127] Burke E. K. and Y. Bykov, "A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems", (extended abstract), in procceedings of PATAT 2008 conference. Montreal, Canada, August 2008.

[128] Ozcan E., Y. Bykov, M. Birben, E. K. Burke, "Examination timetabling using late acceptance hyper-heuristics", in proceedings of the 11th conference on Congress of Evolutionary Computation. Trondheim, Norway, May 2009.

[129] Braun, T. D., Siegel, H. J. and Beck, N. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Systems", IEEE Journal of Parallel and Distributed Computing, Vol. 61, pp. 810-837, 2001.

[130] Goldberg, David E. “Genetic Algorithms in Search Optimization and Machine Learning”, Addison Wesley. p. 41. ISBN 0201157675, 1989.

[131] Glover F., M. Laguna and R. Martí, “Scatter Search. Advances in Evolutionary Computation: Theory and Applications”, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag, New York, pp. 519-537, 2003.

[132] Glover F., M. Laguna and R. Martí, “Fundamentals of Scatter Search and Path Relinking”,Control and Cybernetics, vol. 39, no. 3 pp. 653-684, 2000.

[133] Colorni A., M. Dorigo et V. Maniezzo, “Distributed Optimization by Ant Colonies”, Actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.

[134] Dorigo M., “Optimization, Learning and Natural Algorithms”, PhD thesis, Politecnico di Milano, Italie, 1992.

[135] Karaboga D., “An Idea Based On Honey Bee Swarm for Numerical Optimization”, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[136] Ferrandi Fabrizio, Christian Pilato, Donatella Sciuto, Antonino Tumeo, “Mapping and Scheduling of Parallel C Applications with Ant Colony Optimization onto Heterogeneous Reconfigurable MPSoCs”, *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific* , vol., no., pp.799-804, doi:10.1109/ASPDAC.2010.5419782, 2010.

[137] Li Min, Hui Wang, Ping Li, "Tasks mapping in multi-core based system: hybrid ACO&GA approach", *ASIC, 2003. Proceedings. 5th International Conference on* , vol.1, no., pp. 335- 340 Vol.1, doi: 10.1109/ICASIC.2003.1277556, 2003

[138] Cordes Daniel, Peter Marwedel, Arindam Mallik, “Automatic Parallelization of Embedded Software Using Hierarchical Task Graphs and Integer Linear Programming”, CODES+ISSS’10, ACM, Scottsdale, Arizona, USA, October 24–29, 2010.

[139] Girkar M. and C. D. Polychronopoulos, “The hierarchical task graph as a universal intermediate representation”, International Journal of Parallel Programming, 22(5):519–551, 1994.

[140] Luis J.P., C.G. Carvalho, J.C. Delgado, "Parallelism Extraction in Acyclic Code", 4th Euromicro Workshop on Parallel and Distributed Processing (PDP '96), pp.0437,1996.

[141] Girkar Milind Baburao. “*Functional Parallelism: Theoretical Foundations and Implementation*”,Ph.D. Dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA. UMI Order No. GAX92-15814, 1992.

[142] Girkar M. and C. D. Polychronopoulos, “Automatic Extraction of Functional Parallelism from Ordinary Programs”, *IEEE Trans. Parallel Distrib. Syst.* 3, 2 (March 1992), 166-178. DOI=10.1109/71.127258 http://dx.doi.org/10.1109/71.127258, 1992.

[143] Liu Shan Fan and Mary Lou Soffa. “Parallel Task Assignment by Graph Partitioning”, In *Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe* (PARLE '92), Daniel Etiemble and Jean-Claude Syre (Eds), Springer-Verlag, London, UK, pp. 965-966, 1992.

[144] Beg Mirza, “Critical Path Heuristic for Automatic Parallelization”, University of Waterloo, David R. Cheriton School of Computer Science, Technical Report CS-2008-16, August 2008.

[145] Yi Ying, Wei Han, Xin Zhao, Erdogan, A.T., Arslan, T., "An ILP formulation for task mapping and scheduling on multi-core architectures", *Design, Automation & Test in Europe Conference & Exhibition, DATE '09.*, pp.33-38, 20-24 April 2009.

[146] Tournavitis Georgios, Zheng Wang, Björn Franke and Michael F.P. O'Boyle, “Towards a Holistic Approach to Auto-Parallelization: Integrating Profile-Driven Parallelism Detection and Machine-Learning Based Mapping”, ACM SIGPLAN 2009 Conference on Programming Language Design and Implementation (PLDI'09), 2009.

[147] Rul Sean, Hans Vandierendonck and Koen De Bosschere, “A Dynamic Analysis Tool for Finding Coarse-Grain Parallelism”, 5th HiPEAC Industrial Workshop, pp. 2, 2008.

[148] Rul Sean, Hans Vandierendonck, and Koen De Bosschere, “Function level parallelism driven by data dependencies”, *SIGARCH Comput. Archit. News* 35, 1, pp. 55-62,http://doi.acm.org/10.1145/1241601.1241612, 2007

[149] Hendrickson Bruce and Tamara G. Kolda.“Graph partitioning models for parallel computing”, *Parallel Comput.* 26, 12, pp. 1519-1534,http://dx.doi.org/10.1016/S0167-8191(00)00048-X, 2000.

[150] M. Huang, V. K. Narayana, M. Bakhouya, J. Gaber, and T. El-Ghazawi, “Efficient Mapping of Task Graphs onto Reconfigurable Hardware Using Architectural Variants,” *IEEE Transactions on Computers*, vol. 61, no. 9, pp. 1354–1360, 2012.

[151] J. Singh and G. Singh, “Improved Task Scheduling on Parallel System using Genetic Algorithm,” *International Journal of Computer Applications*, vol. 39, no. 17, pp. 17–22, Feb. 2012.

[152] A. Prasad, J. Anantpur, and R. Govindarajan, “Automatic compilation of MATLAB programs for synergistic execution on heterogeneous processors,” in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, New York, NY, USA, 2011, pp. 152–163.

[153] F. Ferrandi, C. Pilato, D. Sciuto, and A. Tumeo, “Mapping and scheduling of parallel C applications with Ant Colony Optimization onto heterogeneous reconfigurable MPSoCs,” in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, 2010, pp. 799–804.

[154] Bastoul C., “*Efficient code generation for* automatic *parallelization and optimization”*, In ISPDC’03 IEEE International Symposium on Parallel and Distributed Computing, pp. 23-30, Ljubljana, 2003.

[155] Bondhugula U., A. Hartono, J. Ramanujam, and P. Sadayappan, “*PLuTo: A practical and fully automatic polyhedral program optimization system*”, In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, Tucson, AZ, ACM Press, June 2008.

[156] L.N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos. “*Iterative optimization in the polyhedral model: Part II, multidimensional time*”*,* In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’08), pp. 90–100, Tucson, Arizona, ACM Press, June 2008..

[157] Pouchet L.N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, and N. Vasilache, “*Loop Transformations: Convexity, Pruning and Optimization*”, In Proceedings of the 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL’11), pp. 549–562, Austin, TX, ACM Press, January 2011.

[158] Alias C., F. Baray, and A. Darte. “Bee+cl@k: an implementation of lattice-based array contraction in the source-to-source translator rose”, In Proceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES’07), San Diego, California, USA, June 13-15, 2007, pp. 73–82, 2007.

[159] Trifunovic K., D. Nuzman, A. Cohen, A. Zaks, and I.Rosen, “Polyhedral-model guided loop-nest auto-vectorization”, In International Conference on Parallel Architectures and Compilation Techniques, volume 0, pp. 327–337, Los Alamitos, CA, USA, IEEE Computer Society, 2009.

[160] Kong, M., Veras, R., Stock, K., Franchetti, F., Pouchet, L. N., and Sadayappan, “When polyhedral transformations meet SIMD code generation”, in *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation* (PLDI), June 2013.

[161] Liu, J., Zhang, Y., Jang, O., Ding, W., and Kandemir, M., “A compiler framework for extracting superword level parallelism”, in *Proceedings of the 33rd ACM SIGPLAN conference on Programming language design and implementation* (PLDI), June 2012.

[162] Henretty T., K. Stock, L.N. Pouchet, F. Franchetti, J. Ramanujam, and P. Sadayappan. “Data layout transformation for stencil computations on short-vector simd architectures”, In Proceedings of the 20th international conference on Compiler construction: part of the joint European conferences on theory and practice of software, CC’11/ETAPS’11, pp. 225–245, Berlin, Heidelberg, Springer-Verlag, 2011.

[163] Nuzman D., I. Rosen, and A. Zaks, “Auto-vectorization of interleaved data for simd”, In Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, PLDI ’06, pp. 132– 143, New York, NY, USA, ACM, 2006..

[164] Franchetti F. and M. Püschel, “Generating SIMD vectorized permutations”, In International Conference on Compiler Construction (CC), volume 4959 of Lecture Notes in Computer Science, pp. 116–131. Springer, 2008.

[165] Lambrechts A., P. Raghavan, D. Novo, E. R. Ramos, M. Jayapala, F. Catthoor, and D. Verkest, “Enabling wordwidth aware energy and performance optimizations for embedded processors”, in Workshop on Optimizations for DSP and Embedded Systems, Mar 2007.

[166] Khan S., E. Casseau, D. Menard, “High speed reconfigurable SWP operator for multimedia processing using redundant data representation”, Int. Journal of Information Sciences and Computer Engineering (IJISCE), vol. 1, p. 45-52, May 2010.

[167] Khan S., E. Casseau, D. Menard, “Reconfigurable SWP Operator for Multimedia Processing”, Proc. of the 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, Boston, MA, USA, IEEE Computer Society, pp. 199-202, 2009.

[168] Kim S. and W. Sung, “*Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures*”, IEEE Transactions on Circuits and Systems for Video Technology, 8(8):935–940, December 1998.

[169] Coors M., H. Keding, O. Luthje, and H. Meyr, “Integer Code Generation For the TI TMS320C62x”, In IEEE International Conference on Acoustics, Speech, and Signal Processing 2001 (ICASSP’01), pp. 1133–1136, Sate Lake City, US, May 2001.

[170] Constantinides G.. “Perturbation Analysis for Word-length Optimization”, In IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’03), pp. 81–90, Napa, California, April 2003.

[171] Wadekar S. and A. Parker, “Accuracy Sensitive Word-Length Selection for Algorithm Optimization”, IEEE/ACM International Conference on Computer Design (ICCAD’98), pp. 54–61, 1998.

[172] Shi C. and R.W. Bodersen, “A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary inputs“, In IEEE International Symposium on Circuits and Systems (ISCAS’04), pp. 373–376, Vancouver, Canada, May 2004.

[173] Constantinides G.A., “Word-length optimization for differentiable nonlinear systems“, ACM Transactions on Design Automation of Electronic Systems, 26(1):26–43, 2006.

[174] Fiore P.D., “Efficient approximate wordlength optimization”, IEEE Transactions on Computers, 57(11):1561 –1570, 2008.

[175] Caffarena G., J.A. Lopez C. Carreras, and A. Fernandez, “SQNR Estimation of Fixed-Point DSP Algorithms”, EURASIP Journal on Advances in Signal Processing, 2010.

[176] Lopez J.A., C. Carreras, and O. Nieto-Taladriz, “Improved Interval-Based Characterisation of Fixed-Point LTI Systems With Feedbacks Loops”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26:1923–1933, 2007.

[177] Caffarena G., J.A. Lopez C. Carreras, and A. Fernandez, “SQNR Estimation of Fixed-Point DSP Algorithms”, EURASIP Journal on Advances in Signal Processing, 2010.

[178] Menard D., R. Rocher, and O. Sentieys, “Analytical fixed-point accuracy evaluation in linear time-invariant systems”, IEEE Transactions on Circuits and Systems I, 55 (10):3197–3208, 2008.

[179] Rocher R., D. Menard, O. Sentieys, P. Scalart, “Analytical accuracy evaluation of Fixed-Point Systems”, in Proceedings of the 15th European Signal Processing Conference (EUSIPCO), Poznan, Poland, September 2007.