A Decade of Reconfigurable Computing: a Visionary Retrospective

Reiner Hartenstein (embedded tutorial)

CS Dept. (Informatik), University of Kaiserslautern, Germany
http://www.fpl.uni-kl.de   hartenst@rhrk.uni-kl.de

Abstract. *The paper surveys a decade of R&D on coarse grain reconfigurable hardware and related CAD, points out why this emerging discipline is heading toward a dichotomy of computing science, and adocates the introduction of a new soft machine paradigm to replace CAD by compilation.*

1. Introduction

Rapidly increasing attendance [1] of conferences on reconfigurable computing and the adoption of this topic area by congresses like ASP-DAC, DAC, DATE, ISCAS, SPIE, and others indicate, that reconfigurable platforms are heading from niche to mainstream, bridging the gap between ASICs and microprocessors (fig. 2). It's time to revisit R&D results: the goal of this paper. (On some projects mentioned we have only incomplete or no information on the status of implementation.)

2. Coarse-Grained Reconfigurable Architectures

Using FPGAs as accelerator platforms is not subject of this paper. In contrast to FPGA use (fine grain reconfigurable) the area of *Reconfigurable Computing* mostly stresses the use of coarse grain reconfigurable arrays (RAs) with pathwidths greater than 1 bit, because fine-grained architectures are much less efficient because of a huge routing area overhead and poor routability [2]. Since computational datapaths have regular structure, full custom designs of *reconfigurable datapath units* (rDPUs) can be drastically more area-efficient, than by assembling the FPGA way from single-bit CLBs. Coarse-grained architectures provide operator level CFBs, word level datapaths, and powerful and very area-efficient datapath routing switches.

A major benefit is the massive reduction of configuration memory and configuration time, as well as drastic complexity reduction of the P&R (placement and routing) problem. Several architectures will be briefly outlined (also see figure 1). Some of them introduce *multi-granular* solutions, where more coarse granularity can be achieved by bundling of resources, such as e. g. 4 ALUs of 4 bits each to obtain a 16 bit ALU.

2.1 Primarily Mesh-Based Architectures

Mesh-based architectures arrange their PEs in a rectangular 2-D array with horizontal and vertical connections which supports rich communication resources for efficient parallelism. and encourages nearest neighbour (NN) links between adjacent PEs (NN or 4NN: links to 4 sides {east. west. north, south}, or, 8NN: NN-links to 8 sides {east, north-east, north, north-west, west, south-west, south, south-east}, like in CHESS array). Typically, longer lines are added with different lengths for connections over distances larger than 1.

DP-FPGA (Datapath FPGA) [3] has been introduced to implement regularly structured datapaths. It is a FPGA-like mixed fine and coarse grained architecture with 1 and 4 bits. Its fabric includes 3 component types: control logic, the datapath, and memory. The datapath block consists of 4 bit-slices: each bit-slice with a lookup table, a carry chain and a four-bit register. DP-FPGA provides separate routing resources for data (horizontal, 4 bits wide) and control signals (vertical, single bit). A third resource is the shift block to support single-bit or multi bit shifts and irregularities.

The KressArray is primarily a mesh of rDPUs physically connected through wiring by abutment: no extra routing areas needed. In 1995 it has been published [4] as "rDPA" (reconfigurable DataPath Array). "KressArray" has been coined later. The KressArray is a super-systolic array (generalization of the systolic array: fig. 4) which is achieved by DPSS (see § „DPSS"). Its interconnect fabric distinguishes 3 physical levels: multiple unidirectional and/or bidirectional NN links (fig. 3), full length or segmented column or row backbuses, a single global bus reaching all rDPUs (also for configuration). Each rDPU can serve for routing only, as an operator, or, an operator with extra routing paths. The 2nd and 3rd level is layouted over the cell: wiring by abutment capability is not affected.

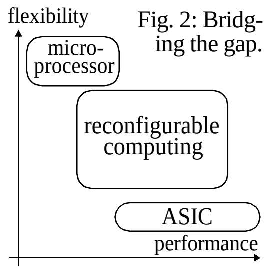


Fig. 2: Bridging the gap.

Project	first publ.	Source	Architecture	Granularity	Fabrics	Mapping	intended target application
PADDI	1990	[29]	crossbar	16 bit	central crossbar	routing	DSP
PADDI-2	1993	[31]	crossbar	16 bit	multiple crossbar	routing	DSP and others
DP-FPGA	1994	[3]	2-D array	1 & 4 bit, multi-granular	inhomogenous routing channels	switchbox routing	regular datapaths
KressArray	1995	[4] [10]	2-D mesh	family: select pathwidth	multiple NN & bus segments	(co-)compilation	(adaptable)
Colt	1996	[11]	2-D array	1 & 16 bit inhomogenous	(sophisticated)	run time reconfiguration	highly dynamic reconfig.
RaPID	1996	[26]	1-D array	16 bit	segmented buses	channel routing	pipelining
Matrix	1996	[13]	2-D mesh	8 bit, multi-granular	8NN, length 4 & global lines	multi-length	general purpose
RAW	1997	[15]	2-D mesh	8 bit, multi-granular	8NN switched connections	switchbox rout	experimental
Garp	1997	[14]	2-D mesh	2 bit	global & semi-global lines	heuristic routing	loop acceleration
Pleiades	1997	[32]	mesh / crossbar	multi-granular	multiple segmented crossbar	switchbox routing	multimedia
PipeRench	1998	[28]	1-D array	128 bit	(sophisticated)	scheduling	pipelining
REMARC	1998	[16]	2-D mesh	16 bit	NN & full length buses	(information not available)	multimedia
MorphoSys	1999	[17]	2-D mesh	16 bit	NN, length 2 & 3 global lines	manual P&R	(not disclosed)
CHESS	1999	[18]	hexagon mesh	4 bit, multi-granular	8NN and buses	JHDL compilation	multimedia
DReAM	2000	[19]	2-D array	8 &16 bit	NN, segmented buses	co-compilation	next generation wireless
CS2000 family	2000	[21]	2-D array	16 & 32 bit	inhomogenous array	co-compilation	communication
MECA family	2000	[22]	2-D array		(not disclosed)		tele- & datacommunication
CALISTO	2000	[23]	2-D array		(not disclosed)		tele- & datacommunication
FIPSOC	2000	[24]	8x12, 8x16 array		(not disclosed)		tele- & datacommunication
flexible array	2000	[25]	2-D array	(not disclosed)	(not disclosed)	(not disclosed)	software radio

Fig. 1: Summary of the technical details of the different coarse-grained reconfigurable architectures; Note: NN stands for "nearest neighbour".

A first 32 bit KressArray included an additional control unit for the MoM-3 [5] Xputer [6] [7] [8] [9] with rDPUs supporting all C language operators. With the new Xplorer environment [10] rDPUs also support any other operator repertoires including branching, while loops and do-while loops. I/O data streams from and to the array can be transferred by global bus, array edge ports, or ports of other rDPUs (addressed individually by an address generator). Although KressArrays are dynamically partially configurable, applications tried out so far did not make use of it.

The KressArray Family. Supported by an application and development tool and platform architecture space explorer (PSE) environment the basic principles of the KressArray define an entire family of KressArrays covering a wide but generic variety of interconnect resources and functional resources. A later version of this PDE environment, called (see paragraph “Xplorer,”), supports the rapid creation of RA and rPDU architectures optimized for a particular application domain (like e. g. image processing, multimedia, or others), and rapid mapping of applications onto any RA of this family.

Colt [11] combines concepts from FPGAs and data flow computing. It’s a 16 bit pipenet [12] and relies highly on runtime reconfiguration using wormhole routing. Hereby, the data stream headers hold configuration data for routing and the functionality of all PEs encountered. Colt has a mesh of 16 bit IFUs (Interconnected Functional Units), a crossbar switch, an integer multiplier, and six data ports. Each IFU features an ALU, a barrel shifter to support multiplication and floating point, a decision unit for flow branching, and optional delay units for pipeline synchronization.

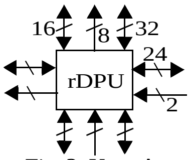


Fig. 3: KressArray NN ports examples.

MATRIX [13] (Multiple Alu architecture with Reconfigurable Interconnect eXperiment) is a multi-granular array of 8-bit BFUs (Basic Functional Units) with procedurally programmable microprocessor core including ALU, multiplier, 256 word data and instruction memory and a controller which can generate local control signals from ALU output by a pattern matcher, a reduction network, or, half a NOR PLA. With these features, a BFU can serve as instruction memory, data memory, register-file and ALU, or independent ALU function. Instructions may be routed over the array to several ALUs. The routing fabric provides 3 levels of 8-bit buses: 8 nearest neighbour (8NN) and 4 second-nearest neighbour connections, bypass connections of length 4, and global lines.

The Garp Architecture. [14] resembles an FPGA and comes with a MIPS-II-like host and, for acceleration of specific loops or subroutines, a 32 by 24 RA of LUT-based 2 bit PEs. Basic unit of its primarily mesh-based architecture is a row of 32 PEs, a reconfigurable ALU. The host has instruction set extensions to configure and control the RA. Array execution is initialized by the number of clock cycles to do. Host and RA share the same memory hierarchy. Memory accesses can be initiated by the RA, but only through the central 16 columns. The blocks in the leftmost column are dedicated controllers for interfacing. For fast reconfigurations, the RA features a distributed cache with depth 4, which stores the least recently used configurations. The routing architecture includes 2 bit horizontal and vertical lines of different length, segmented in a non-uniform way: short horizontal segments spanning 11 blocks, long horizontals spanning the whole array, and different length vertical segments.

RAW: Reconfigurable Architecture Workstation. [15] pro-vides a RISC multi processor architecture composed of NN-connected 32-bit modified MIPS R2000 microprocessor tiles with ALU, 6-stage pipeline, floating point unit, controller, register file of 32 general purpose and 16 floating point registers, program counter, and local cached data memory and 32 Kilobyte SRAM instruction memory. The prototype chip features 16 tiles arranged in a 4 by 4 array. Early concepts [15] having been abandoned included also configurable logic to allow customized instructions. RAW provides both a static (determined at compile-time) and a dynamic network (determined at run-time: wormhole routing for the data forwarding). Since the processors lack hardware for register renaming, dynamic instruction issuing or caching (like in superscalar processors), statically scheduled instruction streams are generated by the compiler, thus moving the responsibility for all dynamic issues to the development software. However, RAW provides possible flow control as a backup dynamic support, if the compiler should fail to find a static schedule.

array	applications	pipeline properties		mapping	scheduling (data stream formation)
		shape	resources		
systolic array	regular data dependencies	linear only	uniform only	linear projection or algebraic synthesis	
super-systolic RA	no restrictions			simulated annealing, genetic morphing, or other P&R algorithm	(e.g. force-directed) scheduling algorithm

Fig. 4: Pipelined datapath arrays (pipe networks).

REMARC. (Reconfigurable Multimedia Array Coprocessor) [16], a reconfigurable accelerator tightly coupled to a MIPS-II RISC processor, consists of an 8 by 8 array of 16 bit “nanoprocessors” with memory, attached to a global control unit. The communication resources consist of nanoprocessors NN connections and additional 32 bit horizontal and vertical buses which also allow broadcast to processors in the same row or column respectively, or, to broadcast a global program counter value each cycle to all nanoprocessors, also to support SIMD operations.

MorphoSys. (Morphoing System [17]) has a MIPS-like “TinyRISC” processor with extended instruction set, a mesh-connected 8 by 8 RA, a frame buffer for intermediate data, context memory, and DMA controller. The RA is divided into four quadrants of 4 by 4 16 bit RCs each, featuring ALU, multiplier, shifter, register file, and a 32 bit context register for storing the configuration word. The interconnect network features 3 layers: 4 NN ports, links of distance 2, and, inter-quadrant buses spanning the whole array. TinyRISC extra DMA instructions initiate data transfers between the main memory and the “frame buffer” internal data memory for blocks of intermediate results, 128 by 16 bytes in total.

The CHESS Array. The CHESS hexagonal array [18] features a chessboard-like floorplan with interleaved rows of alternating ALU / switchbox sequence (figure 5). Embedded RAM areas support high memory requirements. Switchboxes can be converted to 16 word by 4 bit RAMs if needed. RAMs within switchboxes can also be used as a 4-input, 4-output LUT. The interconnect fabrics of CHESS has segmented four-bit buses of different length. There are 16 buses in each row and column, 4 buses for local connections spanning one switchbox, 4 buses of length 2, and 2 buses of length 4, 8 and 16 respectively. To avoid routing congestion, the array features also embedded 256 word by 8 bit block RAMs. An ALU data output may feed the configuration input of another ALU, so that its functionality can be changed on a cycle-per-cycle basis at runtime without uploading. However, partial configuration by uploading is not possible.

The DReAM Array. (Dynamically Reconfigurable Architecture for Mobile Systems [19]) for next generation wireless

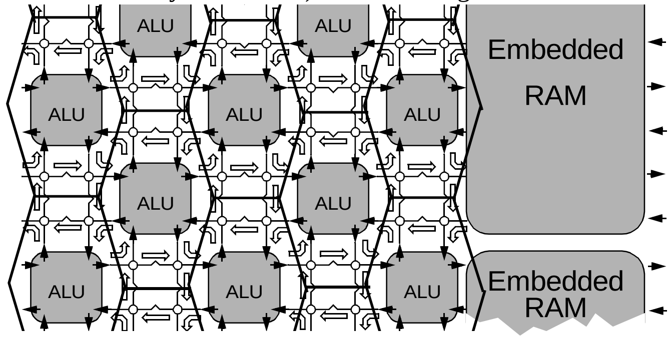


Fig. 5: CHESS array hexagon floor plan.

communication, is a 0.35 µm CMOS standard cell design fabricated by Mietec/Alcatel. Each RPU consists of: 2 dynamically reconfigurable 8-bit Reconfigurable Arithmetic Processing (RAP) units, 2 barrel shifters, a controller, two 16 by 8-bit dual port RAMs (used as LUT or FIFO), and, a Communication Protocol Controller. The RPU array fabric uses NN ports and global buses segmentable by switching boxes.

CS2000 family. Chameleon Systems [20] offers the CS2000 family multi-protocol multi-application reconfigurable platform RCP (reconfigurable communication processor) for telecommunications and data communications [21], with a 32 bit RISC core as a host, licensed from ARC Cores, UK, with full memory controller and PCI controller, connected to a RA of 6, 9, or 12 reconfigurable tiles, where each tile has 7 32-bit rDPUs (each including an 8 word instruction memory), 4 local memory blocks of 128 x 32 bits, 2 16x24-bit multipliers. Every 3 tiles are grouped as a "slice" also including 8 kBytes of local memory. The RA allows multiple independent data streams and its reconfigurable fabric can be changed within a single clock cycle. The CS1200 family aims at initial markets in communication infrastructure and is intended to cope with the chaotic world of evolving standards, protocols and algorithms with application areas as 2nd and 3rd generation wireless basestations, fixed point wireless local loop (WLL), smart antennas, voice over IP (VoIP), very high speed digital subscriber loop (DSL), and, for instance, supports 50 channels of CDMA2000.

The MECA family by Malleable [22] is intended to be a family of DSPs (digital signal processors) optimized for VoIP, by compressing voice into ATM or IP packets etc., aims at next generation VoIP and VoATM. The MECA family claims - compared to conventional DSPs- a speed-up factor of 10.

CALISTO (Configurable ALgorithm-adaptive Instruction Set TOpology) by Silicon-Spice [23] intends to be an innovative adaptive instruction set architecture for internet protocols (IP) and ATM packet.-based networks with flexibility for Any-Service-Any-Port (ASAP) to deliver voice and data simultaneously over a unified data network. CALISTO is a single-chip communications processor for carrier-class voice gateways, soft switches, and remote access concentrators/remote access servers (RAC/RAS), which aims at applications like echo cancellation, voice/fax/data modems, packetization, cellification, delay equalization.

FIPSOC (Field-programmable System-on-Chip) by SIDSA [24] has an 8051 controller, a RA and a RAA (reconfigurable analog array). The 8x12 (8x16 or 16x16) RA is an array of "digital macro cells" (DMC) including a 4-input LUT and 4 latches, programmable to be a 4 bit up/down counter, shift register, 4 bit adder, 16x4 bit RAM etc. The RAA has "configurable analog blocks" (CAB) usable as differential amplifiers, comparators, converters etc. The RA is a multi-context RA featuring 2 extra configuration memories. Device booting can be done from external parallel ROM (as the 8051 controller does), external serial PROM, or RS 232 serial port. FIPSOC is intended to be used as an ASIC emulator for rapid prototyping.

MorphICs mentions its RA, but without details [25].

2.2 Architectures Based on Linear Arrays

Some RAs are based on one or several linear arrays, typically also with NN connect, aiming at mapping pipelines onto it. If the pipes have forks, which otherwise would require a 2-D realization, additional routing resources are needed, like longer lines spanning the whole or a part of the array, often being segmented. Two RAs have linear array structure. RaPiD [26] provides different computing resources, like ALUs, RAMs, multipliers, and registers, but irregularly distributed. While RaPiD uses mostly static reconfiguration, PipeRench relies on dynamic reconfiguration, allowing the reconfiguration of a PE in each execution cycle. Besides the mostly unidirectional NN connects, it provides also a global bus.

RaPiD: The Reconfigurable Pipelined Datapath (RaPiD) [26] aims at speed-up of highly regular, computation-intensive tasks by deep pipelines on its 1-D RA. RaPiD-1 features 15 DPUs of 8 bit with integer multiplier (32 bit output), 3 integer ALUs, 6 general-purpose datapath registers and 3 local 32 word memories, all 16 bits wide. ALUs can be chained. Each memory has a special datapath register with an incrementing feedback path. To implement I/O streams RaPiD includes a stream generator with address generators, optimized for nested loop structures, associated with FIFOs. The address sequences for the generators are determined at compile-time. RaPiD's routing and configuration architecture consists of several parallel segmented 16 bit buses, which span the whole array. The length of the bus segments varies by tracks. In some tracks, adjacent bus segments can be merged. The sophisticated interconnect fabric cannot be detailed here.

Paradigm	Platform	Programming source
"von Neumann"	*Hardware*	*Software*
Xputer	coarse grain *Flexware*	high level *Configware*
RL (FPGA etc.)	fine grain *Flexware*	netlist level *Configware*

Fig. 6: About terminology.

PipeRench [28], an accelerator for pipelined applications, provides several reconfigurable pipeline stages ("stripes") and relies on fast partial dynamic pipeline reconfiguration and run time scheduling of configuration streams and data streams. It has a 256 by 1024 bit configuration memory, a state memory (used to save the current register contents of a stripe), an address translation table (ATT), four data controllers, a memory bus controller and a configuration controller. The reconfigurable fabric of the PipeRench allows the configuration of a pipeline stage in every cycle, while concurrently executing all other stages. The fabric consists of several (horizontal) stripes composed of interconnect and PEs with registers and ALUs, implemented as 3-input lookup tables. The ALU includes a barrel shifter, carry chain circuitry, etc. A stripe provides 32 ALUs with 4 bits each. The whole fabric has 28 stripes. The interconnect scheme of PipeRench features local interconnect inside a stripe as well as local and global interconnect between stripes and four global buses.

2.3 Crossbar-Based Architectures

A full crossbar switch, a most powerful communication network is easily to rout. But the 2 RAs of this category use only reduced crossbars. PADDI for the fast prototyping of DSP datapaths features eight PEs, all connected by a multilayer crossbar. PADDI-2 has 48 PEs, but saves area by restricted crossbar with a hierarchical interconnect structure for linear arrays of PEs forming clusters. This fabrics sophistication has again an impact on routing.

PADDI-1 (Programmable Arithmetic Device for DSP) [29] [30], for rapid prototyping of computation-intensive DSP data paths, consists of clusters of 8 arithmetic execution units (EXUs) 16 bits wide, including 8 word SRAM (which may be concatenated for 32 bits) and connected to a central crossbar switch box. Interconnect is organized in 2 levels: a 3 bit global bus to broadcast global instructions to each EXU's controller CTL, decoded locally at second level into a 53 bit instruction word.

The PADDI-2 Architecture [31] features a data-driven execution mechanism. Although the basic architectural principles of the original PADDI were kept, the PADDI-2 has several differences. It has 48 EXUs. Each PE features a 16 bit ALU also including booth multiply and select, multiplication instructions taking 8 cycles on a single processor. Alternatively, eight EXUs can be pipelined, resulting in a single cycle multiplication. PADDI-2 EXUs are packed in 12 clusters of four elements each two level interconnect: six 16 bit intra-cluster data buses (plus one bit control buses, and, inter-cluster 16 data buses, which can be broken up into shorter segments.

The Pleiades Architecture [32] is a generalized low power "PADDI-3" with programmable microprocessor and heterogeneous RA of EXUs, which allows to integrate both fine and coarse grained EXUs, and, memories in place of EXUs. For each algorithm domain (communication, speech coding, video coding), an architecture instance can be created (with known EXU types and numbers). Communication between EXUs is dataflow driven. The control means available to the programmer are basic EXU configurations to specify its operation, and interconnect configurations to build EXU clusters. All configuration registers are part of the processor's memory map and configuration codes are processor's memory writes.

2.4 Memory Bandwidth Problems

RAM cycle time as throughput bottleneck of von-Neumann-style computing, known as "memory bandwidth problem", a communication gap which spans up to 2 orders of magnitude, can be even wider in RA use as accelerator in data-intensive applications - a challenge to RA architectures and mappers (see § "DTSE"). Interfaces like RAMbus etc. may reduce the gap a little bit for both, host and accelerator. But the use of caches, mainly based on instruction loops (write once - multiple read) does not help in RA usage: compare fig. 15. That's why RA architectures should be capable to interface multiple memory banks, only known from the KressArray (example: fig. 7, where 8 ports communicate with 4 memory banks [33]). Also see chapter 4 and 5.

2.5 Future Reconfigurable Architectures

A universal RA obviously is an illusion. The way to go is toward sufficiently flexible RAs, optimized for a particular application domain like e. g. wireless communication, image processing or multimedia etc. There is a need for tools supporting such dedicated RA architecture development. But architectures have an immense impact on implementability of good mapping tools. "Clever" fabrics are too sophisticated to find good tools. The best solution are simple generic fabrics architecture principles, or, a mapping tool which generically creates by itself the architectures it can manage easily [10], or, a combination of both approaches like the platform space exploration (s. „The KressArray Family" and "Xplorer,"). See chapter 5.

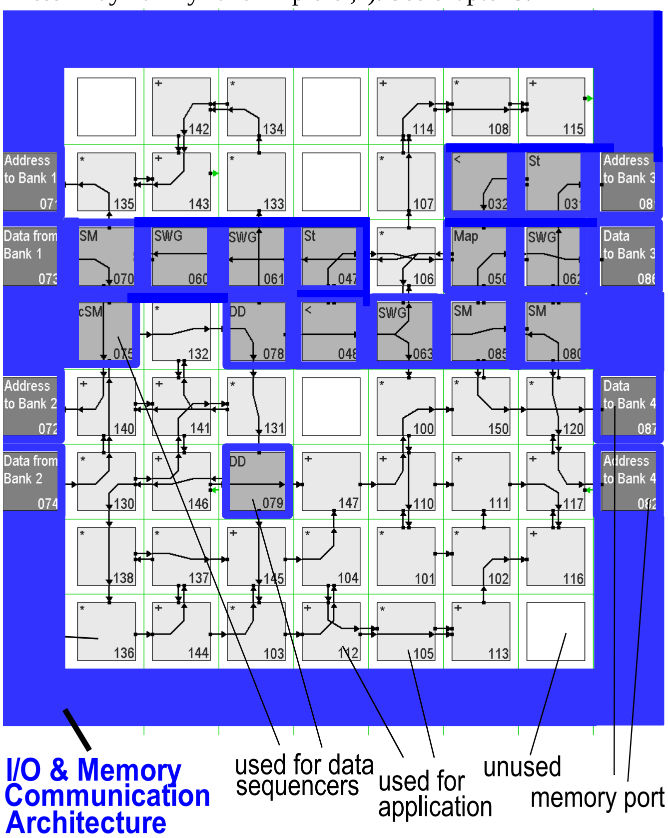


Fig. 7: Mapping application (linear filter) and memory communication architecture (dark background) onto the same KressArray, including the address ports and the data ports to 4 different memory banks (5 of 8 memory port connects are routed through application DPUs).

3. Programming Coarse Grain RAs

Programming frameworks for RAs (also see fig. 1) are highly dependent on structure and granularity, and differ by language level. For MorphoSys, MATRIX, PADDI-2 and REMARC it's assembler level. Some support the designer by a graphical tool for manual P&R. Others feature automatic design flow from HDL or high-level programming language. Environments differ by the approach used for technology mapping, placement, routing. Using only a simple script for technology mapping [34] DP-FPGA [3] is not considered.

Technology mapping is mostly simpler for coarse grain architectures than for FPGAs. Approaches are: direct mapping, where the operators are mapped straight forward onto PEs, with one PE for one operator, or, using an additional library of functions not directly implementable by one PE, or, more sophisticated tree matching also capable to merge several operators into one PE by a modified FPGA tool kit. An exception is the RAW compiler doing partitioning instead of technology mapping, since RAW has RISC cores as PEs accepting blocks from program input.

For operator placement, the architecture has an impact. An approach often used for FPGAs synthesis is placement by simulated annealing or a genetic algorithm. Garp uses a tree matching algorithm instead, where placement is done together with technology mapping. The use of greedy algorithms is feasible only for linear arrays (PipeRench), or with a high level communication network (RAW). PADDI is an exception by using a scheduling algorithm for resource allocation.

AS	application-specific	MIPS	million instructions / second
ASAP	any service any port	MMACS	million MAC per second
ASPP	AS programmable product	MOPS	million operations/ second
CAB	configurable analog block	NN	nearest neighbour
CDFG	control / data flow graph	NNP	NN port
CFB	configurable function block	OS	operating system
CLB	configurable logic block	P&R	placement & routing
CP	communication processor	PA	processor array
DP	data path	PSE	platform space explorer
DPSS	DP synthesis system	RA	reconfigurable array
DRL	dynamically RL	RAA	reconfigurable analog array
DSE	design space explorer	RCP	reconfigurable CP
DSL	digital subscriber loop	rDPU	reconfigurable DP unit
DSP	digital signal processor	RL	reconfigurable logic
DTSE	data transfer and storage explorer	RTOS	real time OS
		SAL	single assignment language
FPL	field-programmable logic		
GCC	GNU C compiler	SAP	single assignment program
HDL	hardware descr. language	SUIF	Stanford intermediate Form
IP	internet protocol	VoIP	voice over IP
JHDL	Java HDL	WLL	wireless local loop
MAC	multiply and accumulate		

Fig. 8: Glossary.

Routing also features quite different approaches. In two cases, the routing is not done in an extra phase but integrated into the placement and done on the fly. One approach (KressArray) uses a simple algorithm restricted to connects with neighbours and targets with at most the distance of one. The other (RaPiD) employs the pathfinder algorithm [35], which has been developed for FPGA routing. Greedy routing would be not satisfying. General exceptions to the routing approaches is the RAW architecture, which features only one high-level communication resource, so no selection of routing resources is needed, and the PADDI architecture, which features a crossbar switch having the same effect. Greedy routing algorithms are only used for 1-D RAs, or architectures capable to cure routing congestion by other mechanisms, like Colt with wormhole run-time reconfiguration.

3.1 Assembler Programming

Assembler level code for coarse grain architectures can be compared to configuration code for FPGAs. In the case of systems comprising a microprocessor / RA symbiosis, only the reconfigurable part is considered for the classification.

mapping	Kress DPSS	CHESS	RaPiD	Colt
placement	simulated annealing	simulated annealing		genetic algorithm
routing		Pathfinder		greedy algorithm

Fig. 9: FPGA-Style Mapping for coarse grain reconfigurable arrays.

Programming is done mainly at a kind of assembler level for PADDI-2, MATRIX, and, RAs of REMARC and MorphoSys. PADDI-2 is crossbar-based whereas the rest is mesh-based.

Programming PADDI-2. For programming PADDI-2 [31], a tool box has been developed which includes software libraries, a graphical interface for signal flow graphs, routing tools, simulation tools, compilation tools and tools for board access and board debugging. Major parts of this process are done manually. The input specifies assembly code for each function in the signal flow graph. The programmer manually partitions the signal flow graph with a graphical tool, which also aids in manual placement and routing. As an alternative to manual placement and routing, an automated tool is provided, which guarantees to find a mapping, if one exists, by exhaustive methods which need much computation time.

For Programming MATRIX. [13] an assembly level macro language has been developed. Some work on P&R pointed out the original MATRIX's weak points [36].

REMARC tools. [16] allow concurrent programming of the RISC processor (by C using the GCC compiler) and the RA by adding REMARC assembler instructions. The compiler then generates assembly code for the RISC processor with the REMARC assembler instructions embedded which are further processed by a special REMARC assembler generating binary code for the REMARC instructions. Finally, the GCC compiler is used again to generate RISC instruction code to invoke REMARC and its instructions embedded as binary data.

Programming MorphoSys. is supported by a SUIF-based compiler [17] for host, and development tools for RA. Host / RA partitioning is done manually by adding a prefix to functions to be mapped onto RA. The compiler generates TinyRISC code for RA activation. Configuration code is generated via a graphical user interface or manually from an assembler level source also usable to simulate the architecture from VHDL.

3.2 Frameworks with FPGA-Style Mapping

Computation-intensive algorithms for mapping onto FPGAs are well-known and can often be used directly for coarse grain architectures like for CHESS, Colt, KressArray, RaPiD (see fig. 9). All four use simulated annealing or other genetics for placement, and two use pathfinder for routing [35]. The KressArray DPSS (Datapath Synthesis System) accepts a C-like language source. The compilation framework for the RaPiD system works similar, but relies on relatively complex algorithms. Colt tools use a structural description of the dataflow. CHESS has been programmed from a hardware description language (JHDL) source. P&R quality has a massive impact on application performance. But, due to the low number of PEs, P&R is much less complex than for FPGAs and computational requirements are drastically reduced.

DPSS (DataPath Synthesis System) [4] generates configuration code for KressArrays from ALE-X high-level language sources [4] [37] supporting datapaths with assignments, local variables and loops. After classical optimizations it generates an expression tree. Next processing steps include a front end, logic optimization, technology mapping creating a netlist, simultaneous P&R by simulated annealing, and I/O scheduling (incl. loop folding, memory cycle optimization, register file usage). The result is the application's KressArray mapping and array I/O schedule. Finally configuration binaries are assembled. Routing is restricted to direct NN connect and rout-through of length 1. Other connect is routed to buses or segmented buses. DPSS has also been part of the MoM-3 Xputer compiler accepting and partitioning a subset of C subset into sequential MoM code and structural KressArray code. The more general CoDe-X approach [38] uses this MoM compiler as part of a partitioning co-compiler accepting a C language superset and partitioning the application onto the host and one or several Xputer-based accelerators.

Tools for Colt [11] accept a dataflow description (below C level) for placement by a genetic algorithm and routing by a greedy algorithm (routing congestion is cured at run-time by wormhole reconfiguration). Data stream headers hold configuration data for routing and the functionality of all PEs encountered.

Programming RaPiD [26] is done in RaPiD-C, a C-like language with extensions (like synchronization mechanisms and conditionals to identify first or last loop iteration) to explicitly specify parallelism, data movement and partitioning, RaPiD-C programs may consist of several nested loops describing pipelines. Outer loops are transformed into sequential code for address generators, inner loops into structural code for the RA.

machine category		Computer ("v. Neumann")	Xputer [7] (no transputer)
machine paradigm		procedural sequencing: deterministic, no "data flow"	
driven by		control flow	data stream(s)
RA support		no	yes
engine principles		instruction sequencing	data sequencing
state register		program counter	(multiple) data counter(s)
communication path set-up		at run time	at load time
data path	resource	single ALU	array of ALUs
	operation	sequential	parallel

Fig. 10: Machine Paradigms.

The compilation steps are: netlist generation from structural code, extraction of dynamic control, generation of controller code instruction streams for dynamic control and generation of I/O configuration data for the stream units. The netlist is mapped onto RaPiD by pipelining, retiming, and P&R. Placement is done by simulated annealing, with routing (by pathfinder [35]) done on the fly to measure placement quality [39].

For Programming the CHESS array [18] a compiler [40] has been implemented accepting JHDL [41] sources and generating CHESS netlists. Placement is done by simulated annealing and routing by Pathfinder's negotiated congestion algorithm [35]. Part of the work is not disclosed.

3.3 Other mapping approaches

Greedy algorithms are poor in mapping to FPGAs. But, although Garp is mesh-based, mapping treats it like a linear array which allows mapping in one step by a simple greedy routing algorithm. RAW features only one communication resource, removing the wire selection problem from routing. Instead, the compiler schedules time multiplexed NN connections. CPU cores inside RAW PEs simplify mapping by loading entire source code blocks. PipeRench resembling a linear array and interconnect fabrics restrictions keep placement simple for a greedy algorithm. PADDI uses a standard P&R approach.

Garp tools. [14] use a SUIF-based C compiler [42] to generate code for the MIPS host with embedded RA configuration code to accelerate (only non-nested) loops. At next basic blocks are generated and converted into hyperblocks containing a contiguous group of basic blocks, also from alternative control paths. Control flow inside a hyperblock is converted for predicated execution. Blocks, which cannot be mapped, are removed from hyperblocks. The resulting reduced hyperblock is then the basis for mapping. The next step generates interfacing instructions for the host, and transforms the hyperblock into a DFG (data flow graph). The proprietary Gamma tool [43] maps the DFG onto Garp using a tree covering algorithm which preserves the datapath structure, supports features like the carry chains. Gamma first splits the DFG into

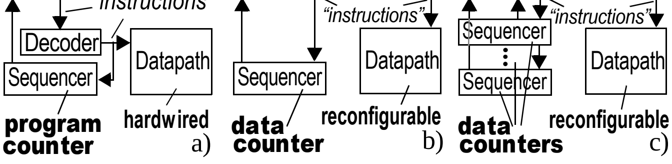


Fig. 11: Machine paradigms: a) v.Neumann, b) Xputer, c) parallel Xputer

Explorer System	year	source	inter-active	status evaluation	status generation
DPE	1991	[75]	no	abstract models	rule-based
Clio	1992	[76]	yes	prediction models	advice generator
DIA	1998	[77]	yes	prediction fr. library	rule-based
DSE for RAW	1998	[48]	no	analytical models	analytical
ICOS	1998	[85]	no	fuzzy logic	greedy search
DSE f. Multimedia	1999	[86]	no	simulation	branch and bound
Xplorer	1999	[10][49]	yes	fuzzy rule-based	simulated annealing

Fig. 12: Design Space Exploration Systems.

subtrees and then matches subtrees a with module patterns which fit in one Garp row. During tree covering, the modules are also placed in the array. After some optimizations the configuration code is generated (incl. outing [45]), assembled into binary form, and, linked with the hosts C object code.

RAW tools [46] [47] include a SUIF-based C compiler and a run-time system managing dynamic mechanisms like branch prediction, data caching [48], speculative execution, dynamic code scheduling. The compiler handles resource allocation, parallelism exploitation, communication scheduling, code generation (for host and switch processors in each tile), and, divides execution into coarse-grain parallel regions internally communicating by static network, whereas intra-region communication uses messages. The phases are: pointer analysis [46] with data dependency analysis and memory access generation (if known at compile-time), partitioning application data for distributed memory; space-time scheduling [47] for parallelization; address translation for caching by software. RAW binary is generated by the MIPS compiler back end. The RAW project aims more at parallel processing rather than reconfigurable computing and failed in finding a good automatic mapping algorithm [49].

PipeRench tools. [28] [50] use the DIL single-assignment language (SAL) for design entry and as an intermediate form. First, the compiler inlines all modules, unrolls loops and generates a straight-line SA program (SAP). After optimizations and breaking the SAP into pieces fitting on one stripe, a greedy P&R algorithm is run which tries to add nodes to stripes. Once placed, a node is routed and never moved again. P&R is fast by crossbar switch usage, coarse granularity, and, restriction to unidirectional pipelines.

CADDI. [51], assembler and simulator, has been implemented for PADDI. First a silage [52] specification is compiled into a CDFG (control /data flow graph), used for estimations of critical path, minimum and maximum bounds for hardware for a given time allocation, minimum bounds of execution time, and for transformations like pipelining, retiming, algebraic transformations, loop unrolling and operation chaining. CDFG to architecture technology mapping is straight-forward since all components are fixed and routing through crossbars is efficient. If several PADDI clusters are involved, a partitioning step comes before resource allocation, assignment, and scheduling. The assignment phase maps operations to EXUs by a rejectionless antivoter algorithm [53].

Compilation for Pleiades. Because of the complex design space created by the heterogeneous architecture, the application mapping problem is not yet solved completely.

C~SIDE. development tools for the Chameleon CS2000 family include a GNU C compiler for the RISC host, a HDL synthesizer for the reconfigurable fabric, a simulator, a C-style debugger and a verifier. eBIOS (eConfigurable Basic I/O Services), a kind of operating system, interfaces the RISC processor with the reconfigurable fabric. C~SIDE is a tool box, rather than a co-compiler. About mapping or compilation for the MECA family [22], FIPSOC [24] and MorphICs RA [25] no information is available.

IDE. (Integrated Development Environment) with a C-compiler, debugger, simulator and "Evaluation Module" (EVM) serves for CALISTO from Silicon Spice [23], where a Real-time operating system (RTOS) supports "Any Service Any Port" (ASAP) configurations for up to 240 channels of carrier class G.711 VoIP (voice over IP). IDE is a tool box, but no co-compiler.

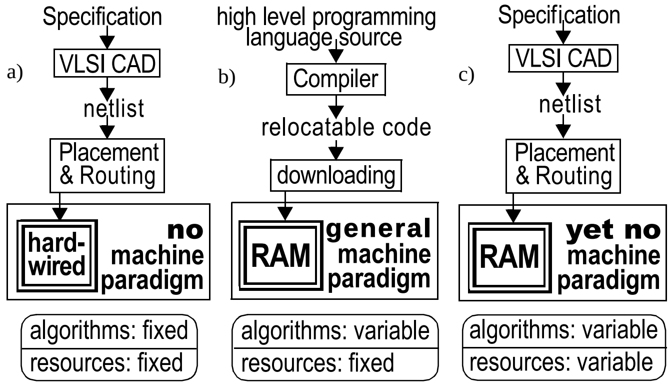


Fig. 13: Synthesis a) hardwired, b) "von Neumann", c) reconfigurable.

3.4 Run-time Mapping

The VIRTEX FPGA family from Xilinx, the RAs being part of the CS2000 series systems from Chameleon and others are run-time reconfigurable. Programming a host/RA combination is a kind of H/S Co-design. However using such devices changes many of the basic assumptions in the HW/SW co-design process: host / RL interaction is dynamic and needs a kind of tiny operating system like eBIOS, also to organize RL reconfiguration under host control. A typical goal is mimization of reconfiguration latency (especially important in communication processors), to hide configuration loading latency, and, list scheduling to find 'best' schedule for a series of eBIOS calls (also see § "C~SIDE"). For more about typical aspects of run-time reconfiguration see [54] [55].

3.5 Retrospective

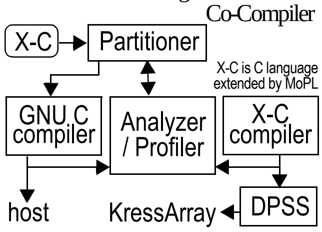
The history of silicon synthesis and application distinguishes three phases [56] (or, 3 Makimoto waves [57]): hardware design (fig. 13 a), introduction of the microcontroller (fig. 13 b), and RA usage (fig. 13 c). The transition from phase 1 (structural synthesis) to phase 2 brought a shift from net-list-based CAD (fixed algorithms, no machine paradigm, infinite design space) to RAM-based (procedural) synthesis by compilation, based on a machine paradigm, which reduces the design space by guidance. Note: RAM-based means flexibility and fast turn-around. Accelerator synthesis (fig. 16 b) still uses phase 1 methods (CAD). The third phase introduces reconfigurable hardware, i. e. RAM-based structural synthesis: resources have become variable. But still phase 1 synthesis tools are mainly used: versions of CAD tools. Since structural synthesis has become RAM-based it is time to switch to real compilation techniques, based on a soft machine paradigm. But the R&D scene ignore, that we now have a dichotomy of RAM-based programming: procedural programming versus structural programming, integrating two worlds of computing (fig. 17)

4. Compilation Techniques

The classical "von Neumann" scheme is obsolete (fig. 16 a). Today, microprocessor / accelerator(s) symbiosis is the dominant computer application (fig. 16 b), where sequential code is downloaded into the host's RAM, whereas the accelerator is implemented by CAD. From this view also the classical compiler is a kind of obsolete. Using RAs as accelerators again changes this scenario: now implementations onto both, host and RA(s) are RAM-based and compilation techniques for both sides would be desirable: *co-compilation* (fig. 16 c). But until recently only hardware/software co-design environments have been implemented, where a C compiler is only an isolated minor part to program the host and RAs are still programmed by CAD [43] [50] [58] [59] [62].

Since the design flow in mapping applications onto RAs is still managed by hand, using isolated tools for particular steps for particular platforms, currently massive hardware expertise is needed to implement accelerators on RA platforms. Programming of both, host and RA, being RAM-based allows turn-around times of minutes - instead of months needed for hardwired ASIC accelerators. This means a change of market structure by migration of accelerator implementation from IC vendor to customer who mainly has no hardware designers available. So there is a strong need for automatic compilation from high level programming language sources onto RAs.

Fig. 14: CoDe-X Co-Compiler



Most of the platforms summarized above make use of a host/RA symbiosis. Newer commercial platforms include all on a single chip. E. g. Altera's EXCALIBUR combines a core processor (ARM, or MIPS, etc.), embedded memory and RL. Compilers mentioned above (except CoDe-X, see below), like e. g. a GNU C compiler, are used just for programming the host. However, software / configware partitioning and the creation of host/RA interfaces are still mainly done manually.

4.1 Software / Configware Partitioning

An important goal to be met by innovative compilers is to do the partitioning automatically which has to be controlled several criteria: how far an algorithm is parallelizable at all, to consider the capacity and capabilities of available flexware to find out what and how much workload fits onto a given RA part (with a compiler which is retargettable by parametrization [38]), to optimize the hardware/configware trade-off, and, by PSE to optimize the performance/hardware/flexware trade-off: A compiler doing all this from a high level language source we call a software / configware *Co-compiler* (see next section).

4.2 Co-Compilation

Real software / configware co-compilation means a consequent transition from a CAD / compilation mix to real machine-paradigm-based compilation onto both, host and RA including automatic partitioning. (This is the only way to avoid, that designers hate their tools [60]). CoDe-X is the first such environment having been implemented [38]. also using parts of earlier work [61].

CoDe-X. The availability of reconfigurable universal accelerator machines (like the Xputer [62]) creates a demand for innovative compilation techniques, such as e. g. a partitioning compiler accepting input from programming language sources (fig. 14). This is needed, because two different kinds of code have to be downloaded (fig. 16 c): procedural code into the RAM of the host, and, structural code into the (hidden) RAM of the reconfigurable accelerator. We need a partitioning compiler, which we call *Co-Compiler*. An example is the *CoDe-X* partitioning co-compiler [38] (fig. 14), accepting C programs and partitioning an application into a task for the host and tasks for the accelerator machine.

Loop transformations. Host/accelerator partitioning in CoDe-X is mainly carried out by identifying loops suitable for parallelizing transformation into code downloadable to the MoM accelerator machine (MoM is an Xputer architecture). The CoDe-X implementation includes 5 different loop transformation methods: strip mining [63] [64], loop fusion [63] [65], loop splitting [63] [66]), loop interchanging [67], and, loop unrolling [64]. Within CoDe-X these loop transformations are controlled by resource parameters for optimum adaptation to the amount of KressArray resources available, like in hardware / software co-design.

C~SIDE etc. Chameleon Systems reports for its CS2000 series co-compilation [21] techniques, combining compiler optimization, multithreading to hide configuration loading latency, and, list scheduling to find 'best' schedule for eBIOS calls (see § "C~SIDE"). Whether automatic partitioning is used is undisclosed.

4.3 Why we need a Soft Machine Paradigm

Exploding ASIC design cost and shrinking ASIC product life cycles are a motivation to replace at least some of the ASICs by RAs for product longevity by upgrading [68] [69] [70]. Performance is only one part of the story. The design community is far away from fully exploiting the flexibility of RAs [41], supporting novel powerful techniques -directly in system jointly with all other components in real time, dramatically faster than simulation- for debugging, run-time profiling, verification, system-wide tuning, run-time API, field-maintenance, field-upgrades (also via internet) flexible self-monitoring functions by configuring unused parts of the RA.

This potential is largely unrealized although having been technically possible and demonstrated already for about a decade [41]. Using RAs instead of ASICs creates new market structures by transferring synthesis from vendor to customer, who does not have the hardware experts required, and, thus needs compilers to replace CAD. A machine paradigm makes compilers much easier to develop and machines easier to program - the success story of software industry.

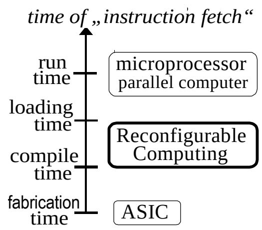


Fig. 15: "Instruction Fetch".

Soft Machine Paradigm needed. We need a new machine paradigm, since "v. Neumann" does not support soft datapaths because "instruction fetch" is not done at run time (fig. 15). Instead of a program counter like the "von Neumann machine" (fig. 11 a) we need a data counter (fig. 11 b) [62] provided by the Xputer paradigm. Figure 10 clarifies he fundamental differences between both machine paradigms. The Xputer (deterministic) is not a classical "data flow machine" operating inderterministically since being driven by arbitration. An Xputer (non-von-Neumann) is not a transputer (von Neumann). Equipped with multiple data sequencers (fig. 11 c) a single such Xputer machine may even handle several parallel data streams, like illustrated by the example in fig. 7 (see section 2.4).

The Xputer Machine Paradigm. Xputer data sequencers (see fig. 11 b and c) have already been implemented [72]. This Xputer machine paradigm -the diametral counterpart of the von Neumann paradigm- has been published a decade ago: the new "Xputer" machine paradigm for soft hardware (fig. 10) [6] [7] [8] [9]. Instead of a "control flow" sublanguage a "data stream" sublanguage recursively defines *data goto*, *data jumps*, *data loops*, *nested data loops*, and *parallel data loops* (using multiple data counters) like by the MoPL language [73] - easy to learn by its similarity to control flow.

5. Design Space Explorers (DSEs)

Some development environments aim beyond compiling. DSEs (fig. 12 [49]) select one of many alternative solutions to meet a design goal meeting constraints or desired properties to optimize a *design* or a (by PSE) *programmable platform*. Guidance systems or design assistants are interactive DSEs giving advice during the design flow. Some DSEs avoid the status generation and provide only predictions etc. from a knowledge data base. Advanced DSEs provide status generation, e g. by expert system, and present advice like a choice of proposals. Non-interactive DSEs automatically generate a solution status from rule-based knowledge or fuzzy learning.

5.1 Design Space Exploration

Interactive design assistants are DPE and Clio (both for VLSI) and DIA. Including effect predictors and proposal generators DPE (Design Planning Environment) [75] (using an expert system), Clio [76] (using a hierarchy of templates) and DIA (Datapath-Intensive ASICs) [77] (targeting semi-custom ASIC behavioural level and based on encapsulated expert knowledge), generate a design flow by creating a schematic, a data flow graph, or a layout from a specification and area, cycle time, power, e.a. constraints and to improve area, power, throughput etc.

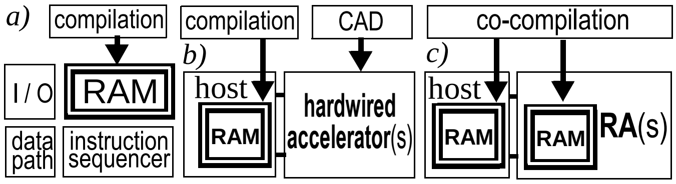


Fig. 16: Computing Platforms: a) “v. Neumann”, b) current, c) emerging.

DTSE. For data-dominated systems the usual local optimization techniques lead to performance-degrading runtime solution of access conflicts. A cost overhead of 10 - 100% (in power) for hardware and around 35% (in clock rate) for software has been estimated [78] [79]. For global exploration the use of conflict-directed ordering (CDO) [80] as an extension of force-directed scheduling (FDS) [81] has been proposed [82]. Instead of working on a signal access flow graph (SAFG) [80] a multi-dimensional conflict graph (MD-CG) is used for a generalized CDO (G-CDO) algorithm for the ATOMIUM / ACROPOLIS data transfer and storage exploration (DTSE) system [83] [84].

5.2 Platform Space Explorers (PSEs)

A PSE serves to find an optimum RA or processor array (PA) platform for an application domain by optimizing array size, path width, processor's MIPS, number of ALUs and branch units, local SRAM size, data and instruction cache sizes, local bandwidth, interconnect latency etc. from requirements like chip area, total computation, memory size, buffer size, communication etc. Software or configware application programming is finally not part of exploration, but may serve platform evaluation. All three being non-interactive, the *DSE [48] for RAW* [15] featuring an analytical model, *ICOS* (Intelligent Concurrent Object-oriented Synthesis) [85] featuring object-oriented fuzzy techniques, and *"DSE for Multimedia Processors"* [86] (DSEMMP) aim at automatic synthesis of a multiprocessor platform from system descriptions, performance constraints, and a cost bound and generate an architecture.. DSEMMP aims at shared memory with intel Strong-ARM SA-110 as a starting point.

5.3 Compiler / PSE symbiosis

Since to map an application onto a coarse grain RA may take only minutes, retargettable mappers or compilers may be also used for platform exploration. By profiling the results of the same application or benchmark on different platforms may be compared. Such a compiler / PSE symbiosis like in *Xplorer* provides direct verification and yields more realistic and more precise results than explorers using abstract models for estimation and gives better support for manual tuning.

Xplorer, an interactive PSE framework [10] [49] has been implemented around the DPSS mapper [4]. This universal design space exploration environment supports both, optimum architecture selection (e. g. domain-specific) and application development onto it and includes several tools: *architecture editor* (to edit communication resources and annealing parameters), *mapping editor* (to change I/O port type, freeze locations of edge port, cell or cell group etc.), *instruction mapper* to change the operator repertoire, *architecture suggestion generator* [87], *HDL generator* for cell simulation, *retargettable cell layout generator* (planned, similar to [88]), *power estimator* (planned [89], using methods from [91]). A cycle through an exploration loop usually takes only minutes, so that a number of alternative architectures may be checked in a reasonable time. By mapping the application onto it verification is provided directly.

The memory bandwidth problem. The Xplorer also yields efficient solutions to the memory bandwidth problem [33] by supporting mixed rDPU types in an array, so that both, data sequencers and rDPUs dedicated to the application can be mapped onto the same KressArray what is illustrated by the example in fig. 7 (also see section 2.4). These Xplorer capabilities provide a straight-forward approach to support architectural implementations of the Xputer soft machine paradigm (also see section 4.3).

5.4 Parallel Computing vs. Reconfigurable

RISC core IP cells are available (e.g. from ARM) so small, that 32 or more (soon 64) of them would fit onto a single chip to form a massively parallel computing system. But this is not a general remedy for the parallel computing crisis [92], indicated by rapidly shrinking supercomputing conferences. For many application areas process level parallelism yields only poor speed-up improvement per processor added. Amdahls law explains just one of several reasons of inefficient resource utilization. A dominating problem is the instruction-driven late binding of communication paths (fig. 15), which often leads to massive communication switching overhead at run-time. R&D in the past has largely ignored, that the so-called "von Neumann" paradigm is not a communication paradigm. However, some methods from parallel computing and parallelizing compiler R&D scenes may be adapted to be used for lower level parallelism on RA platforms (compare § "CoDe-X" ff.).

6. Conclusions

Exploding design cost and shrinking product life cycles of ASICs create a demand on RA usage for product longevity. Performance is only one part of the story. The time has come fully exploit their flexibility to support turn-around times of minutes instead of months for real time in-system debugging, profiling, verification, tuning, field-maintenance, and field-upgrades. The new machine paradigm and language framework is available for novel compilation techniques to cope with the new market structures transferring synthesis from vendor to customer.

Reconfigurable platforms and their applications are heading from niche to main-stream, bridging the gap between ASICs and micro-processors. Many system-level integrated future products without reconfigurability will not be competitive. Instead of tech-nology progress better archi-tectures by RA usage will be the key to keep up the current innovation speed beyond the limits of silicon. It's time to revisit past decade R&D results to derive commercial solutions: at least one promising approach is available. It is time for you to get involved. Theory and backgrounds are ready for creation of a dichotomy of computing science (fig. 17) for curricular innovations.

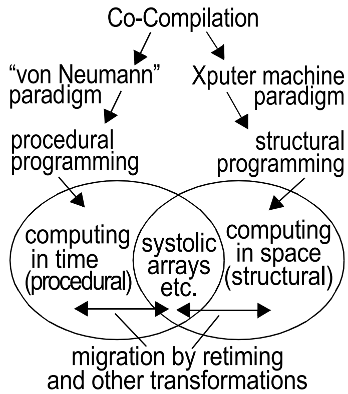


Fig. 17: Computing Science Dichotomy.

7. Literature

1. R. Hartenstein, H. Grünbacher (Editors): The Roadmap to Reconfigurable computing - Proc. FPL2000, Aug. 27-30, 2000; LNCS, Springer-Verlag 2000
2. R. Hartenstein: The Microprocessor is no more General Purpose (invited paper), Proc. ISIS'97, Austin, Texas, USA, Oct. 8-10, 1997.
3. D. Cherepacha and D. Lewis: A Datapath Oriented Architecture for FPGAs; Proc. FPGA'94, Monterey, CA, USA, February 1994.
4. R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Data-path Architecture; ASP-DAC'95, Chiba, Japan, Aug. 29 - Sept. 1, 1995
5. H. Reinig: A Scalable Architecture for Custom Computing; Ph.D. Thesis, Univ. of Kaiserslautern, Germany, July 1999.
6. R. Hartenstein, A. Hirschbiel, M. Weber: MoM - a partly custom-design architecture compared to standard hardware; IEEE CompEuro 1989
7. R. Hartenstein et al.: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware; InfoJapan'90, 30th Anniversary o' Computer Society of Japan, Tokyo, Japan, 1990.
8. R. Hartenstein et. al.: A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE J.SSC, Volume 26, No. 7, July 1991.
9. (invited reprint of [7]) R. Hartenstein, A. Hirschbiel, K. Schmidt, M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance-HW; Future Generation Com-puter Systems 7 91/92, p. 181-198, (Elsevier Scientific).
10. U. Nageldinger et al.: KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures; ASP-

DAC, Yokohama, Japan, Jan. 25-28, 2000.
11. R. A. Bittner et al.: Colt: An Experiment in Wormhole Run-time Reconfiguration; SPIE Photonics East `96, Boston, MA, USA, Nov. 1996.
12. K. Hwang: Advanced Computer Architecture; McGraw-Hill, 1993.
13. E. Mirsky, A. DeHon: MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources; Proc. IEEE FCCM‘96, Napa, CA, USA, April 17-19, 1996.
14. J. Hauser and J. Wawrzynek: Garp: A MIPS Processor with a Reconfigurable Coprocessor; Proc. IEEE FCCM‘97, Napa, April 16-18, 1997.
15. E. Waingold et al.: Baring it all to Software: RAW Machines; IEEE Computer, September 1997, pp. 86-93.
16. T. Miyamori and K. Olukotun: REMARC: Reconfigurable Multimedia Array Coprocessor; Proc. ACM/SIGDA FPGA‘98, Monterey, Feb. 1998.
17. H. Singh, et al.: MorphoSys: An Integrated Re-configurable Architecture; Proceedings of the NATO RTO Symp. on System Concepts and Integration, Monterey, CA, USA, April 20-22, 1998.
18. A. Marshall et al.: A Reconfigurable Arithmetic Array for Multimedia Applications; Proc. ACM/SIGDA FPGA‘99, Monterey, Feb. 21-23, 1999
19. J. Becker et al.: Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems; Proc. FCCM’00, Napa, CA, USA, April 17-19, 2000.
20. http://www.chameleonsystems.com
21. X.Tang, et al.: A Compiler Directed Approach to Hiding Configuration Loading Latency in Chameleon Reconfigurable Chips; in [1]
22. http://www.malleable.com
23. http://www.silicon-spice.com
24. http://www.sidsa.com
25. http://www.MorphICs.com
26. C. Ebeling et al.: „RaPiD: Reconfigurable Pipelined Datapath“, in [27]
27. M. Glesner, R. Hartenstein (Editors): Proc. FPL’96, Darmstadt, Germany, Sept. 23-25, 1996, LNCS 1142, Springer Verlag 1996
28. S. C. Goldstein et al.: PipeRench: A Coprocessor for Streaming Multimedia Acceleration; Proc. ISCA‘99, Atlanta, May 2-4, 1999
29. D. Chen and J. Rabaey: PADDI: Programmable arithmetic devices for digital signal processing; VLSI Signal Processing IV, IEEE Press 1990.
30. D. C. Chen, J. M. Rabaey: A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths; IEEE J. Solid-State Circuits, Vol. 27, No. 12, Dec. 1992.
31. A. K. W. Yeung, J.M. Rabaey: A Reconfigurable Data-driven Multiprocessor Architecture for Rapid Prototyping of High Throughput DSP Algorithms; Proc. HICSS-26, Kauai, Hawaii, Jan. 1993.
32. J. Rabaey: Reconfigurable Computing: The Solution to Low Power Programmable DSP; Proc. ICASSP’97 Munich, Germany, April 1997.
33. M. Herz: High Performance Memory Communication Architectures for Coarse-Grained Reconfigurable Computing Architectures; Ph. D. Dissertation, Universitaet Kaiserslautern, January 2001
34. D. Lewis: Personal Communication, April 2000.
35. C. Ebeling et al.: Placement and Routing Tools for the Tryptich FPGA; IEEE Trans VLSI Systems 3, No. 4, December 1995.
36. A. DeHon: Personal Communication, February 2000.
37. T. Molketin: Analyse, Transformation und Verteilung arithmetischer und logischer Ausdruecke auf die rekonfigurierbare Datenpfadarchitektur; Diplomarbeit, Universitaet Kaiserslautern, 1995
38. J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators; Ph. D. dissertation, Kaiserslautern University, 1997.
39. C. Ebeling: Personal Communication, March 2000.
40. A. Marshall: Personal Communication; February 2000.
41. B. Hutchings, B. Nelson: Using General-Purpose Programming Languages for FPGA Design; Proc. DAC 2000, Los Angeles, June 2000
42. M. W. Hall et al.: Maximizing Multiprocessor Performance with the SUIF Compiler; IEEE Computer, Dec. 1996
43. T. J. Callahan and J. Wawrzynek: Instruction-Level Parallelism for Reconfigurable Computing; in [44] pp. 248-257.
44. R. Hartenstein, A. Keevallik (Editors): Proc. FPL’98, Tallinn, Estonia, Aug. 31- Sept. 3, 1998, LNCS, Springer Verlag, 1998
45. J. Hauser: Personal Communication, March 2000.
46. R. Barua et al.: Maps: A Compiler-Managed Memory System for RAW Machines; Proc. ISCA‘99, Atlanta, USA, June, 1999.
47. W. Lee et al.: Space-Time Scheduling of Instruction-Level Parallelism on a RAW Machine; Proc. ASPLOS‘98, San Jose, Oct. 4-7, 1998.
48. C. A. Moritz et al.: Hot Pages: Software Caching for RAW Microprocessors; MIT, LCS-TM-599, Cambridge, MA, Aug. 1999.
49. U. Nageldinger: Design-Space Exploration for Coarse Grained Reconfigurable Architectures; Dissertation, Universitaet Kaiserslautern, Febr. 2001
50. M. Budiu and S. C. Goldstein: Fast Compilation for Pipelined Reconfigurable Fabrics; Proc. FPGA‘99, Monterey, Feb. 1999, pp. 135-143.
51. D. Chen at al.: An Integrated System for Rapid Prototyping of High Performance Data Paths; Proc. ASAP‘92, Los Alamitos, Aug. 4-7, 1992
52. P. H. Hilfinger: A High-Level Language and Silicon Compiler for Digital Signal Processing; Proc. 1985 IEEE CICC., Portland, May 20-23, 1985.
53. M. Potkonjak, J. Rabaey: A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs; Proc. DAC‘89, Las Vegas, June 1989
54. J. Noguera, R. Badia: A HW/SW Partitioning Algorithm for Dynamically Reconfigurable Architectures; Proc. DATE 2001, Munich, Germany March 12 - 15, 2001
55. J. Noguera, R. Badia: Run-time HW/SW Codesign for Discrete Event Systems using Dynamically Reconfigurable Architectures; Proc. ISSS 2000 (Int’l Symp. System Synthesis), Madrid, Spain, Sept. 20 - 22, 2000
56. N. Tredennick: Technology and Business: Forces Driving Microprocessor Evolution; Proc. IEEE 83, 12 (Dec. 1995)
57. T. Makimoto: The Rising Wave of Field-Programmability; in: [1]
58. M. Weinhardt, W. Luk: Pipeline Vectorization for Reconfigurable Systems; Proc. IEEE FCCM, April 1999
59. M. Gokhale, J. Stone: NAPA C: Compiling for a hybrid RISC / FPGA architecture; Proc. IEEE FCCM April 1998
60. N. N.: FCCM’98 Top 10 Predictions...; http://www.fccm.org/top10_98raw.html
61. K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, University of Kaiserslautern 1994
62. J. Becker et al.: A General Approach in System Design Integrating Reconfigurable Accelerators; Proc. IEEE ISIS’96; Austin, TX, Oct. 9-11, 1996
63. L. Lamport: The Parallel Execution of Do-Loops; C. ACM, Febr. 1974
64. D. B. Loveman: Program Improvement by Source-to-Source Transformation; J. ACM 24,1, p. 121-145, Jan. 1977
65. W. A. Abu-Sufah, D. J. Kuck, D. H. Lawrie: On the Performance Enhancement of Paging Systems Through Program Analysis and Transformations; IEEE-Trans. C-30(5), p. 341-356, May 1981
66. U. Banerjee: Speed-up of Ordinary Programs; Ph.D. Thesis, Univ. of Illinois at Urbana-Champaign, DCS Rep. UIUCDCS-R-79-989, Oct. 1979.
67. J. Allen, K. Kennedy: “Automatic Loop Interchange”; Proc. ACM SIGPLAN’84 Symp. on Compiler Construction, Montreal, Ca, June 1984
68. T. Kean: It‘s FPL, Jim - but not as we know it! - Market Opportunities fro the new Commercial Architectures; in [1]
69. H.Fallside, M.Smith: Internet Connected FPL; in [1]
70. R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger: An Internet Based Development Framework for Reconfigurable Computing; in [71]
71. P. Lysaght, J. Irvine, R. Hartenstein (Eds.): Proc. FPL'99, Glasgow, UK, Aug./Sept. 1999, LNCS Vol. 1673, Springer-Verlag, 1999
72. M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; Proc. ASAP‘97, Zurich, Switzerland, July 14-16, 1997
73. A. Ast, J. Becker, R. Hartenstein, R. Kress, H. Reinig, K. Schmidt: Data-procedural Languages for FPL-based Machines; in [74]
74. R. Hartenstein, M. Servit (Editors): Proc. FPL’94, Prague, Czech Republic, Sept. 7-10, 1994, LNCS, Springer Verlag, 1994
75. D. Knapp & al.: The ADAM Design Planning Engine, IEEE Trans CAD, July 1991
76. J. Lopez et al.: Design Assistance for CAD Frameworks; Proc. EURO-DAC’62, Hamburg, Germany, Sept. 7-10, 1992
77. L. Guerra et al.: A Methodology for Guided Behavioural Level Optimization; Proc. DAC’98, San Francisco, June 15-19, 1998
78. A. Vandecapelle et al.: Global Multimedia Design Exploration using Accurate Memory organization Feedback; Proc. DAC 1999
79. T. Omnès et al: Dynamic Algorithms for Minimizing Memory Bandwidth in High throughput Telecom and Multimedia; in: Techniques de Parallelisation Automatique, TSI, Éditions Hermès, 1999
80. S. Wuytack et al: Minimizing the required Memory Bandwidth in VLSI System Realizations; IEEE Trans. VLSI Systems, Dec. 1999
81. P. Paulin et al.: Algorithms for High-Level Synthesis; IEEE Design and Test, Dec. 1989
82. F. Cathoor et al: Interactive Co-design of High Throughput Embedded Multimedia; Proc. DAC 2000
83. L. Nachtergaele et al.: Optimization of Memory Organization and Partitioning for Decreased Size and Power in Video and Image Processing Systems; Proc. IEEE Workshop on Memory Technology, Aug 1995
84. F. Cathoor et al.: Custom Memory Management Methodology - Exploration of Memory Organization f. Embedded Multimedia Systems; Kluwer 1998
85. P.-A. Hsiung et al.: PSM: An Object-oriented Synthesis Approach to Multiprocessor Design; IEEE Trans VLSI Systems 4/1, March 1999
86. J. Kin et al.: Power Efficient Media Processor Design Space Exploration; Proc. DAC’99, New Orleans, June 21-25, 1999
87. R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger: Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures; in [1]
88. V. Moshnyaga, H. Yasuura: A Data-Path Modules Design from Algorithmic Representations; IFIP WG 10.5 Worksh. on Synthesis, Generation and Portability of Library Blocks for ASIC Design, Grenoble, France, Mar 1992
89. U. Nageldinger et al.: Design-Space Exploration of Low Power Coarse Grained Reconfigurable Datapath Array Architectures; in [90]
90. D. Soudris, P. Pirsch, E. Barke (Editors): Proc. PATMOS 2000; Göttingen, Germany Sept. 13 - 15, 2000; LNCS, Springer Verlag, 2000
91. L. Kruse et al.: Lower Bounds on the Power Consumption in Scheduled Data Flow Graphs with Resource Constraints; Proc. DATE, Mrch 2000.
92. R. Hartenstein (invited paper): High-Performance Computing: über Szenen und Krisen; GI/ITG Worksh. Custom Computing, Dagstuhl, June 1996