reiner@hartenstein.de
15 July 2010

Seminar 10281
11. 7. - 16. 7. 2010

Dynamically
Reconfigurable Architectures

Schloss Dagstuhl

Where Computer Scientists Meet

Reiner Hartenstein

TU Kaiserslautern

Why we need to Reinvent Programmer Education

Amid the Clamor

Michael Wrinn, (keynote at SIGCSE2010):
Suddenly, All Computing Is Parallel: Seizing Opportunity Amid the Clamor
http://www.sigcse.org/sigcse2010/attendees/keynotes.php

the proud era of
von Neumann architecture
passes into history.

Foundational change of this magnitude
will disrupt traditional habits
throughout the discipline ....

... especially how students are to be introduced ....

RC is the silver bullet

„Multicore computers shift the burden of software performance from chip designers to programmers."

[J. Larus: Spending Moore's Dividend; C_ACM, May 2009]

To avoid future unaffordability of our cyber infrastructure we need a massive **software to configware migration**

The migration of the huge supply of legacy software saves much more energy than most proposals from the climate protection scene

... impossible without reinventing programmer education (embedded system scene has not sufficient manpower)

 3 http://hartenstein.de

New Programmer Education

What existing environments are a good starting point, or, could be included?

UML ? Access to existing libraries ?

New mix of skills needed, currently not available

Skills to detect overhead and bottlenecks

Fully understanding the architecture, its nodes, and its communication and memory architecture

Essential: intuitive awareness of locality

 4 http://hartenstein.de

Memory Mapping Issues

Must focus primarily on memory mapping issues

µP and co-processors share memory on the node,

and interconnect many nodes with
NoC, GigE, Infiniband, or a custom interconnection

configuring the nodes in a distributed memory layout

Which parallelism model

The domain scientist, or RC-aware domain scientist, should be equipped with a sufficiently transparent parallelism model

practicing time to space mapping:

- to learn more locality awareness,
- efficient solutions from the beginning, avoiding a later re-design..

Parallelism Levels

Instruction-sequential to concurrent-only: **not** the silver bullet

- ILP instruction level parallelism
- I-PLP instruction process level parallelism

Twin-paradigm sequential to concurrent: **the way to go!**

- DLP data object level parallelism
- DSLP data stream level parallelism
- ILP instruction level parallelism
- I-PLP instruction process level parallelism

7

Visible architecture

The programming model:
hardware view presented to the programmer:
Which hardware architecture parts are visible
and which are under the programmer's direct control.

RC programming model:
whether (and how) the programmer can control :
- data transfers between FPGA and onboard memory,
- between FPGA and µP memory, as well as
- between FPGA and µP itself.

8

Cray-XD1 Architecture

Cray-XD1 allows Opteron µP to access FPGA internal registers, internal and external memory.

The FPGA can access the µP memory.

However, HLL use disables some of these features.

several transfer modes between µP and the FPGA (depending on its initiator).

The µP can read from / write to the FPGA local memory space (i.e. internal registers, internal BRAMS, and external memory)

FPGA can read from / write to the µP local memory space.

Cray-XD1 Architecture (2)

Cray-XD1 allows Opteron µP to access FPGA internal registers, internal and external memory.

most bandwidth-efficient transfer mode:
write-only mode (producer initiates the transfer):
burst (for large amount of data) or non-burst.

How datastreams are processed

The graphic interface should transparently illustrate to its user, how data streams are running through the structures.

to the Domain scientist this is much more informative, than abstract text-only notations like languages, which do not provide physical locality awareness. The ease of understanding should be the main objective.

Here the user should not need to learn a textual language. Instead he should know the semantics of the types of available boxes.
A simulator should enable "running" the structures sticked together.

Graphic Representation

We need a modeling level less abstract than by textual languages

We need a graphic box interconnect scheme like from Mathworks.

Its supply of boxes should include (transport-triggered) data paths, memory blocks, asMs, queues, stacks, (de)multiplexers, interconnect resources, etc.

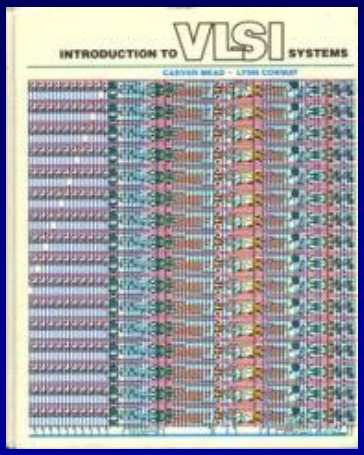
Users should be able to select and interconnect such "boxes" during a session at the graphical user interface.

The user determines the box dimension (path width, pipe length, etc.)

Qualified Programmer Population missing

Missing programmer population, tools and methodology:

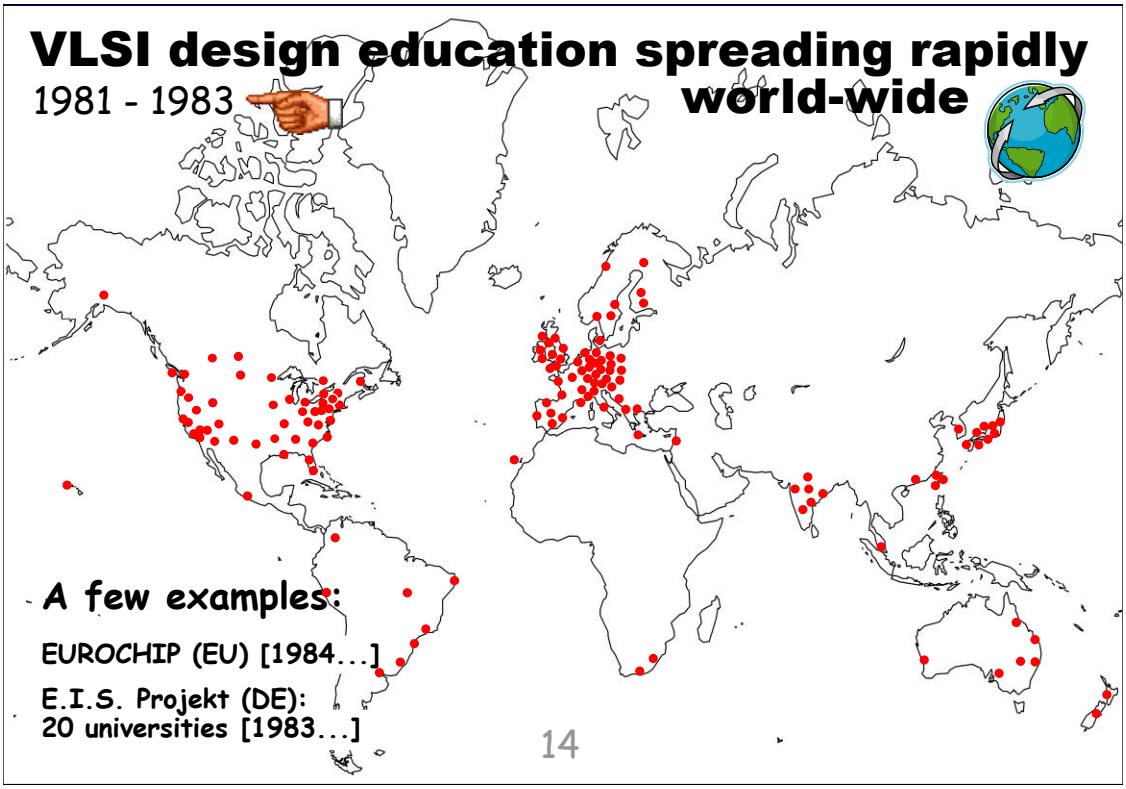
a scenario like before the Mead-&-Conway revolution



 *) RC = Reconfigurable Computing 13 http://hartenstein.de

VLSI design education spreading rapidly world-wide

1981 - 1983



A few examples:

EUROCHIP (EU) [1984...]

E.I.S. Projekt (DE):
20 universities [1983...]

14

New Course Material

We need a new undergraduate text book

We need a graphic representation method and its semantics

We need a model-based undergraduate lab course running under this interface

New CS Curricula

CS curricula must represent today's computing reality by parallel, **reconfigurable**, and distributed computing.

We need to change Computer Architecture courses by introducing parallel, **reconfigurable**, and distributed computing in early undergraduate CS.

not yet here: complicated problems and in-depth analysis.

from the start, we expect students to consider the world of computing as filled with **hetero** systems.

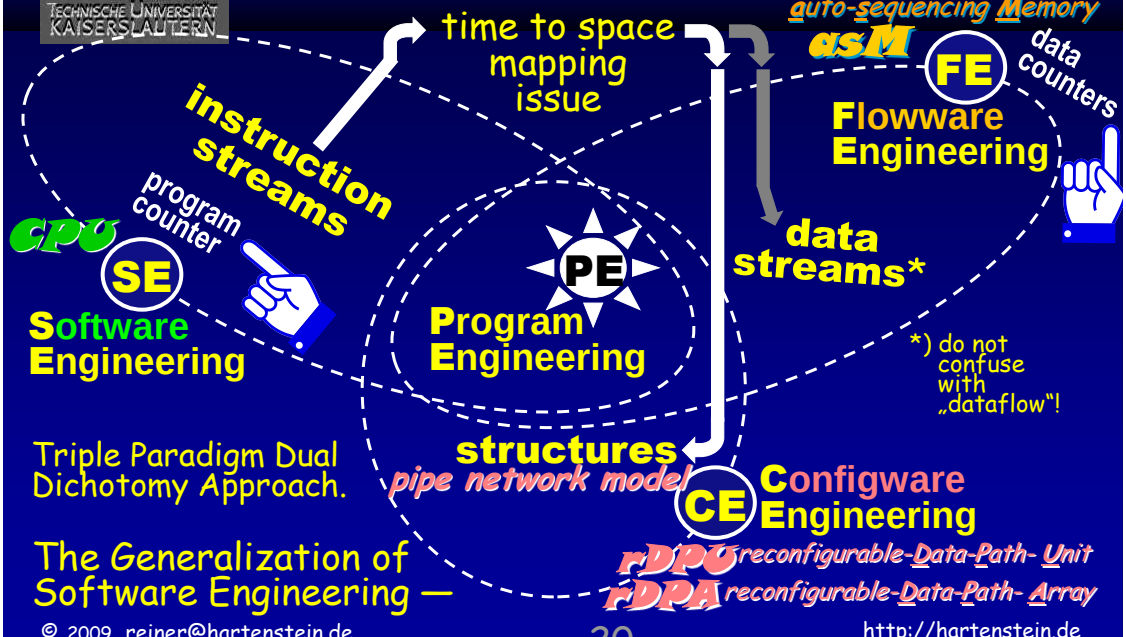
thank you for your patience

END

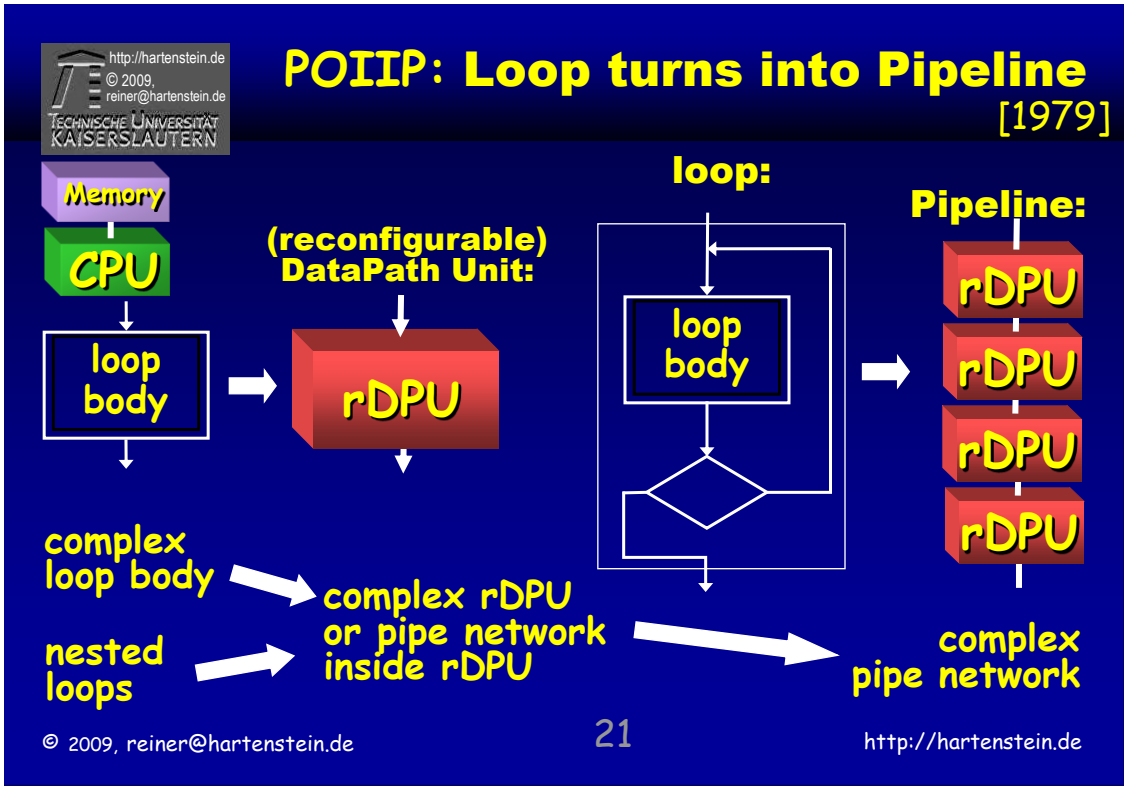
backup for discussion

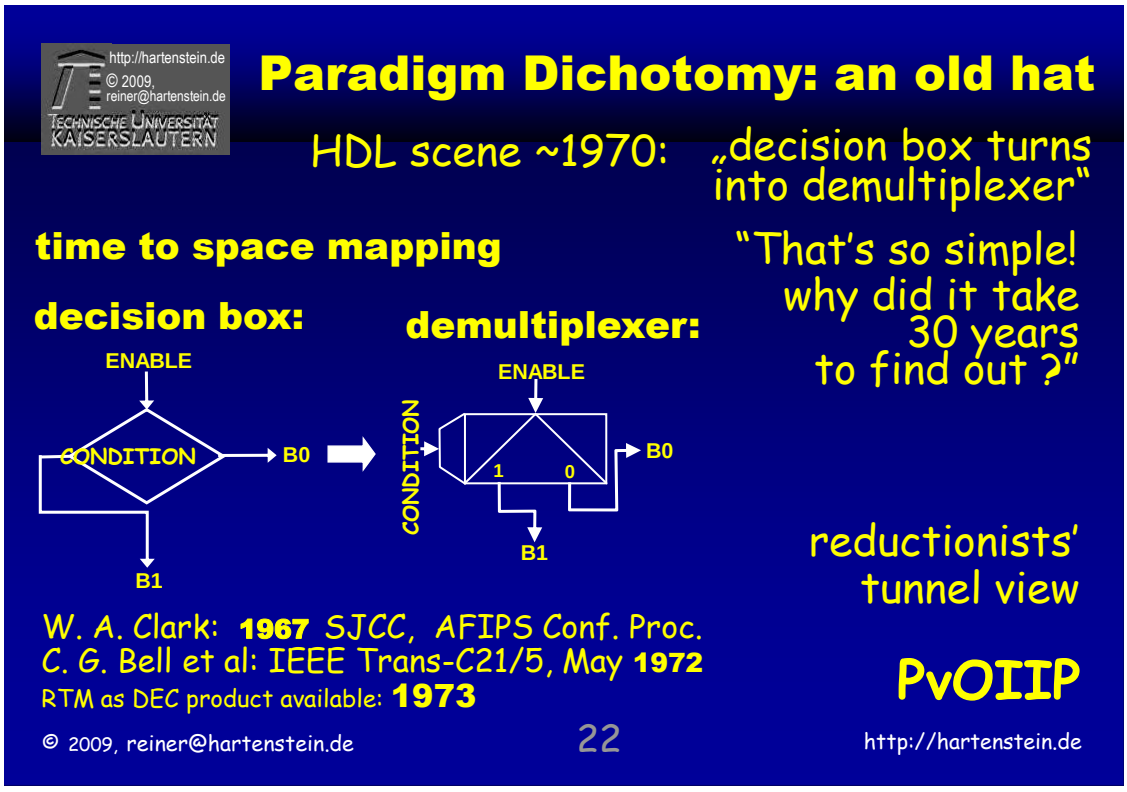
19

A Heliocentric CS Model needed



20


http://hartenstein.de
© 2009, reiner@hartenstein.de
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
POIIP: Loop turns into Pipeline
[1979]
Memory
CPU
loop body
(reconfigurable) DataPath Unit:
rDPU
loop:
loop body
Pipeline:
rDPU
rDPU
rDPU
rDPU
complex loop body
nested loops
complex rDPU or pipe network inside rDPU
complex pipe network
© 2009, reiner@hartenstein.de
21
http://hartenstein.de


http://hartenstein.de
© 2009, reiner@hartenstein.de
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
Paradigm Dichotomy: an old hat
HDL scene ~1970: „decision box turns into demultiplexer"
time to space mapping
"That's so simple! why did it take 30 years to find out ?"
decision box:
demultiplexer:
ENABLE
CONDITION
B0
B1
ENABLE
CONDITION
B0
1
0
B1
reductionists' tunnel view
W. A. Clark: 1967 SJCC, AFIPS Conf. Proc.
C. G. Bell et al: IEEE Trans-C21/5, May 1972
RTM as DEC product available: 1973
PvOIIP
© 2009, reiner@hartenstein.de
22
http://hartenstein.de

Double Dichotomy

1) Paradigm Dichotomy

von Neumann Machine
instruction stream
(Software-Domain)

⟷

Datastream Machine
data stream
(Flowware-Domain)

2) Relativity Dichotomy

time:
-Procedure
(Software-Domain)

⟷

space:
-Structure
(Configware-Domain)

What is a Dichotomy ?

Dichotomy = mutual allocation to two opposed domains such, that a third domain is excluded.

The dichotomy model as a didactic orientation guide to overcome the software/configware chasm

Dichotomy of paradigms (von Neumann /Anti machine): the „Twin Paradigm"

time-iterative to space-iterative

n time steps, 1 CPU

a time to space mapping

1 time step, n DPUs

Often the space dimension is limited

n*k time steps, 1 CPU

a time to space/time mapping

n time steps, k DPUs

POTTP

Strip mining

[D. Loveman, J-ACM, 1977]

loop transformation methodogy: 70ies and later [survey: Diss. Karin Schmidt, 1994, Shaker Verlag]

e. g. example: bubble sort migration

25

Relativity Dichotomy

time domain:

procedure domain
(Machine Dichotomy)

time

2 phasees:
1) programming instruction streams
2) run time

von Neumann Machine

space domain:

structure domain

space

3 phases:
1) reconfiguration of structures
2) programming data streams
3) run time

Anti Machine

26

time to space mapping

time domain:
procedure domain

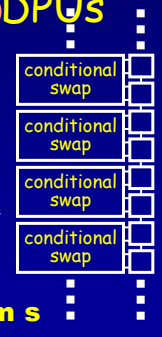
space domain:
structure domain

time algorithm → space algorithm

complexity: O(n) only

program loop
n time steps, 1 CPU

→ *pipeline*
1 time step, n DPUs



complexity: $O(n^2)$

Bubble Sort
n x k time steps,
1 „conditional swap" unit

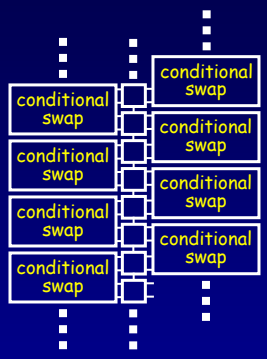
→ *Shuffle Sort*
k time steps,
n „conditional swap" units

time algorithm → space/time algorithm s

27

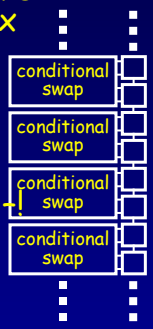
Architecture instead of synchro

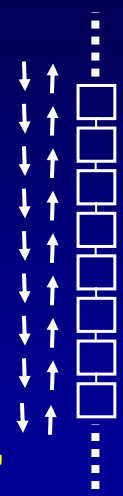
Example



direct time to space mapping
accessing conflicts

Better Architecture instead of complex synchronisation:
half he number of Blocks
+
up / down of data (shuffle function)





complexity: O(n) only

modification: with shuffle-function
„Shuffle Sort"

28

CUDA: a programming language

Simple, elegant, code easy to maintain, mature compilers

The CUDA programming model forces the developer to identify the irreducible level of parallelism in his problem.

seems to be a better, more intuitive and extensible way to think about our problems.

the CUDA code was cleaner and more readable language for expressing parallelism

reducing the programming model from three levels to one level for a simpler more elegant solution

29

ISA Extensions

create instructions tailored to applications.

merge many simple operations into mega ops

automatic ISA extensions can be effective

manually created ISA extensions gives larger gains:
Tensilica reports speedups of 40x to 300x for FFT, AES and DES encryption

avoid store / communicate intermediate results

30

Too many HDLs

Table 1. Surveyed HLLs

1	Impulse-C	14	System-Studio	27	CoreFire
2	Handel-C	15	ConvergenSC	28	DSS
3	Mitrion-C	16	Transmogrifier-C	29	Forge
4	Dime-C	17	SPARK	30	CASH
5	System-C	18	Brass	31	C2Verilog
6	Catapult-C	19	DeFacto	32	Bach C
7	Carte-C	20	MATCH	33	SpecC
8	Streams-C	21	JHDL	34	Ocapi
9	AccelChip	22	Galadriel & Nenya	35	HardwareC
10	NAPA-C	23	Viva	36	Cones
11	SA-C	24	Ptolemy II	37	BDL
12	Trident Compiler	25	SysGen	38	Cocentric
13	CHiMPS	26	RC Toolbox	39	C2H

E. El-Araby et al.: Comparative Analysis of High Level Programming for Reconfigurable Compu
Methodology And Empirical Study; Proc. SPL2007, Mar del Plata, Argentina, Febr. 2007

EDA the main bottleneck

The different HLL paradigms/approaches:

- imperative programming (Impulse-C)
- functional programming (Mitrion-C)
- schematic/graphical programming (DSPLogic)