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

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.“

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)

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
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

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.
Cray-XD1 Architecture

Cray-XD1 allows Opteron \( \mu P \) to access FPGA internal registers, internal and external memory.

The FPGA can access the \( \mu P \) memory.

However, HLL use disables some of these features.

several transfer modes between \( \mu P \) and the FPGA (depending on its initiator).

The \( \mu P \) 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 \( \mu P \) local memory space.

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 stucked 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

VLSI design education spreading rapidly world-wide

A few examples:
EUROCHIP (EU) [1984...]
E.I.S. Projekt (DE): 20 universities [1983...]
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

A Heliocentric CS Model needed

triple paradigm dual dichotomy approach.

The generalization of software engineering —

configured data path unit

Dynamically Reconfigurable Architectures;
Dagstuhl Seminar 10281, July 11 – 16, 2010
**POIIP: Loop turns into Pipeline**

[1979]

- Loop body
- (reconfigurable) DataPath Unit: rDPU
- complex loop body
- nested loops
- complex rDPU
- or pipe network inside rDPU
- Pipeline: complex pipe network

**Paradigm Dichotomy: an old hat**

HDL scene ~1970: decision box turns into demultiplexer

"That's so simple! why did it take 30 years to find out?"

C. G. Bell et al: IEEE Trans-C21/5, May 1972
RTM as DEC product available: 1973

Reductionists' tunnel view

PvOIIP

http://hartenstein.de
Double Dichotomy

1) Paradigm Dichotomy

von Neumann Machine
instruction stream
(Software-Domain)

Datastream Machine
data stream
(Flowware-Domain)

2) Relativity Dichotomy

-Procedure
(Software-Domain)

-Structure
(Configware-Domain)

time:

space:

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“
**Dynamically Reconfigurable Architectures;**

Dagstuhl Seminar 10281, July 11 – 16, 2010

---

**time-iterative to space-iterative**

Often the space dimension is limited

---

loop transformation methodology:
70's and later

- e.g. example: bubble sort migration

---

Relativity Dichotomy

**time domain:**

- procedure domain
- (Machine Dichotomy)

- 2 phases:
  - 1) programming instruction streams
  - 2) run time

von Neumann Machine

---

**space domain:**

- structure domain

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

Anti Machine

---

© 2009, reiner@hartenstein.de

---

25

Strip mining

[D. Loveman, J-ACM, 1977]

© 2009, reiner@hartenstein.de

---

26

© 2009, reiner@hartenstein.de

---

http://hartenstein.de
### Time to Space Mapping

**Time Domain:**
- Procedure domain
- Program loop: \( n \) time steps, 1 CPU
- Complexity: \( O(n^2) \)

**Space Domain:**
- Structure domain
- Pipeline: 1 time step, \( n \) DPsUs

#### Examples

- **Bubble Sort**
  - \( n \times k \) time steps, 1 "conditional swap" unit
  - Complexity: \( O(n^2) \)

- **Shuffle Sort**
  - \( k \) time steps, \( n \) "conditional swap" units

**Modification:**
- Direct time to space mapping
- Accessing conflicts
- Complexity: \( O(n) \) only

### Architecture Instead of Synchro

**Better Architecture instead of complex synchronisation:**
- Half the number of Blocks
- Up / down of data (shuffle function)

**Example:**
- "Shuffle Sort"
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. It 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

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
Too many HDLs

Table 1. Surveyed HLLs

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Impulse-C</td>
<td>14</td>
<td>System-Studio</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>Handel-C</td>
<td>15</td>
<td>ConvergenSC</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Mitrion-C</td>
<td>16</td>
<td>Transmogrifier-C</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>Dime-C</td>
<td>17</td>
<td>SPARK</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>System-C</td>
<td>18</td>
<td>Brass</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Capult-C</td>
<td>19</td>
<td>DeFacto</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>Carte-C</td>
<td>20</td>
<td>MATCH</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Streams-C</td>
<td>21</td>
<td>JHDL</td>
<td>34</td>
</tr>
<tr>
<td>9</td>
<td>AccelChip</td>
<td>22</td>
<td>Galadriel &amp; Nenya</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>NAPA-C</td>
<td>23</td>
<td>Viva</td>
<td>36</td>
</tr>
<tr>
<td>11</td>
<td>SA-C</td>
<td>24</td>
<td>Ptolemy II</td>
<td>37</td>
</tr>
<tr>
<td>12</td>
<td>Trident Compiler</td>
<td>25</td>
<td>SysGen</td>
<td>38</td>
</tr>
<tr>
<td>13</td>
<td>CHIMPS</td>
<td>26</td>
<td>RC Toolbox</td>
<td>39</td>
</tr>
</tbody>
</table>

© 2009, reiner@hartenstein.de

EDA the main bottleneck

The different HLL paradigms/approaches:

- imperative programming (Impulse-C)
- functional programming (Mitrion-C)
- schematic/graphical programming (DSPL#[graphical representation of schematic programming])

“Simulator-of-The-Year” Phenomenon
[courtesy Richard Newton]