Software or Configware? 
About the Digital Divide of Parallel Computing

The “havenots”
Configware methodology to move data around more efficiently:
“havenots” are found in the HPC community
Configware engineering as a qualification for programming embedded systems:
The “havenots” are our typical CS graduates
Reconfigurable HPC is torpedaded by deficits in education:
curricular revisions are overdue

Software to Configware Migration
Software to Configware Migration is the most important source of speed-up
Hardware is just frozen Configware
this talk will illustrate the performance benfit which may be obtained from Reconfigurable Computing (RC), point of view, this talk hardly mentions FPGAs
(But coarse grain may be always mapped onto FPGAs)
data are moved around by software

i.e. by memory-cycle-hungry instruction streams which fully hit the memory wall

(slower than CPU clock by 2 orders of magnitude)

... understand only this parallelism solution:

the instruction-stream-based approach

the data-stream-based approach

has no von Neumann bottle neck

... continue to bang their heads against the memory wall

extremely unbalanced

... path of least resistance:

avoiding a paradigm shift

Many researchers seem never to stop working on sophisticated solutions for marginal improvements ...

... continously ignoring methodologies promising speed-ups by orders of magnitude ....

more offending statements to come

>> Embedded Computing <<

• HPC
• Embedded Computing
• The wrong Roadmap
• Configware Engineering
• Dual Machine Paradigms
• Speed-up Examples
• Final Remarks

History of Machine Models

・mainframe age
・computer age (PC age)

example: MD GRAPE-2 PCI board [1999] 4 chips for N-body simulation converts a PC to 64 GFlops

transform computers into supercomputers

(procedural mind set)

instruction-stream-based

(coordinates by Malmix wave)

mainframe

compile

µProc - accel
History of Machine Models

1957: Mainframe age
- Compile
- Mainframe
- Instruction-stream-based
- Procedural mind set (coordinated by Makintosh wave)

1967: Computer age (PC age)
- Compile
- Design
- Structural mind set
- Data-stream-based

1977: Accelerators
- µProc. accel.

1987: Computer age (PC age)
- Structural mind set
- Data-stream-based
- By hardware guys

1997: Computer age (PC age)
- Structural mind set
- Data-stream-based
- By hardware guys
- Design

2007: Computer age (PC age)
- Structural mind set
- Data-stream-based

the hardware / Software Chasm:

Typical programmers don’t understand function evaluation without machine mechanisms (counters, state registers)

Growth Rate of Embedded Software

Already today, more than 98% of all microprocessors are used within embedded systems.

+10 times more programmers will write embedded applications than computer software by 2010

Software to Configware Migration is the most important source of speed-up

Hardware is just frozen Configware

typical CS graduates: the „havenots“

To-day, typical CS graduates are unqualified for this labor market

... cannot cope with Hardware / Configware / Software partitioning issues

... cannot implement Configware

the current CS mind set is based on the Submarine Model

This model does not support Hardware / Configware / Software partitioning

Software invisible: under the surface

Hardware invisible: under the surface

Algorithm

Procedural high level Programming Language

Assembly Language

Hardware / Configware / Software Partitioning skills urgently needed

Hardware is just frozen Configware

or: to cope with any combination of co-design

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
By the way...
...the oldest and largest conference in the field:

International Conference on
Field-Programmable Logic
and Applications (FPL)
http://fpl.org
Aug. 20 - Sept 1, 2004, Antwerp, Belgium

μProc-accels
...going into every type of application
288 submissions!
they all work on high performance

CS Education
You cannot teach Hardware
to a Programmer
But to a Hardware Guy
you always can teach Programming

© 2004, reiner@hartenstein.de
http://hartenstein.de

Completely wrong roadmap
beef up old architectural
principles by new technology?

Pollack's Law
(area efficiency)
...the CPU is a methusela,
the steam engine
of the silicon age
http://hartenstein.de

Completely wrong mind set
The key problem, the memory wall,
cannot be solved by new CPU technology
The vN paradigm is not a communication paradigm
Its monopoly creates a completely wrong mind set
We need a 2nd machine paradigm (a 2nd mind set ...)
We need an architectural communication paradigm
But we need both paradigms: a dichotomy

© 2004, reiner@hartenstein.de
http://hartenstein.de

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
3rd machine model became mainstream

De facto Duality of RAM-based platforms

We now have 2 types of programmable platforms

From Software to Configware Industry

benefit from RAM-based & 2nd paradigm

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
Nick Tredennick’s Paradigm Shifts explain the differences

Software Engineering
- CPU
  - resources: fixed
  - algorithm: variable
  - 1 programming source needed

Configware Engineering
- configware
  - resources: variable
  - algorithm: variable
  - 2 programming sources needed

Flowware programs data streams

Flowware: not new

data streams*: not new

1980: data streams (Kung, Leiserson: systolic arrays)
1989: data-stream-based Xputer architecture
1990: rDPU (Rabaey)
1994: Flowware Language MoPL (Becker et al.)
1995: super systolic array (rDPA) + DPSS tool (Kress)
1996+: Stream-C language, SCCC (Los Alamos), SCORE, ASPRC, Bee (UC Berkeley), ...
1996+: streaming languages (Stanford et al.)
1996+: configware / software partitioning compiler (Becker)

Compilation: Software vs. Configware

Flowware defines: which data item at which time at which port

Flowware: data stream* ...

Dual Machine Paradigms

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
Why a new machine paradigm???

The anti machine as the 2nd paradigm is the key to curricular innovation

... a Trojan horse to introduce the structural domain to the procedural-only mind set of programmers

Programming by flowware instead of software is very easy to learn (... some language primitives)

Flowware education: no fully fledged hardware expert needed to program embedded systems

Counters: the same micro architecture?

instruction stream machine: (von Neumann etc.)
data stream machine: (anti machine)

yes, is possible, but for data counters ...

... a much better AGU methodology is available

*) for history of AGUs see Herz et al.: Proc. ICECS 2002, Dubrovnik, Croatia

commercial rDPA example: PACT XPP - XPU128

• Full 32 or 24 Bit Design
• 2 Configuration Hierarchies
• Evaluation Board available, and
• XDS Development Tool with Simulator

mapping algorithms efficiently onto rDPA by DPSS: based on simulated annealing

symbiosis of machine models

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
The image contains a slide from a presentation titled "Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004" by Reiner Hartenstein, University of Kaiserslautern, Germany.

The slide content is as follows:

**Software / Configware Co-Compilation**
- "High level PL source"
- "W" machine paradigm
- SW code
- Partitioner
- Analyzer/Profiler
- CW compiler
- Resource Parameters
- Supporting different platforms

**Better solutions by Configware instead of Software**
- Methodologies not new: high level synthesis (1980+)
- Loop transformations (1970+)
- Many other areas
- Memory cycles minimized
- E.g.: no instruction fetch at run time & other effects
- No cache misses!
- Memory access for data: caches do not help anyhow
- Loop xforms: no intra-stream data memory cycles
- Complex address computation: no memory cycles

**Speed-up Examples**
- HPC
- Embedded Computing
- The wrong Roadmap
- Configware Engineering
- Dual Machine Paradigms
- Speed-up Examples
- Final Remarks

**Conditional operation example**
- \[ S = R + (\text{if } C \text{ then } A \text{ else } B \text{ endif}) \]
- If C then load A
- Instruction decoding
- Instruction execution
- Store result

**Software to Configware Migration**
- Ulrich Nageldinger's talk about KressArray Xplorer:
- Question by a highly respected industrial senior researcher:
- "But you can't implement decisions!"
- (Symptom of ...)

This slide contains technical information and examples related to software and hardware compilation, demonstrating the benefits of using Configware over traditional software methods, particularly in the areas of memory cycle minimization and speed-up examples for platforms like CPU, FPGA, and others.
rDPA (coarse grain) vs. FPGA (fine grain)

<table>
<thead>
<tr>
<th></th>
<th>rDPA</th>
<th>FPGA</th>
<th>µProc</th>
<th>DSP</th>
<th>FPGA</th>
<th>rDPA</th>
<th>hardwired</th>
</tr>
</thead>
<tbody>
<tr>
<td>performance (MOPS/mW, o'o' magnitude)</td>
<td>hardwired 3</td>
<td>rDPA 4</td>
<td>rDPA 3</td>
<td>FPGA 2</td>
<td>FPGA 2</td>
<td>µProc 0</td>
<td>0</td>
</tr>
<tr>
<td>area efficiency (trans/chip, o'o' magnitude)</td>
<td>hardwired 4</td>
<td>rDPA 4</td>
<td>rDPA 3</td>
<td>FPGA 2</td>
<td>FPGA 2</td>
<td>µProc 0</td>
<td>0</td>
</tr>
</tbody>
</table>

Why the speed-up ...

... although FPGA is clock slower by \( x \leq 3 \) or even more (most know-how from high level synthesis discipline)

- support operations: no clock nor memory cycle
- decisions operations: moving operator to the data stream (before run time)
- most „data fetch“ without memory cycle

Why the speed-up ...

... although FPGA is clock slower by \( x \leq 3 \) or even more (most know-how from high level synthesis discipline)

- support operations: no clock nor memory cycle
- decisions operations: moving operator to the data stream (before run time)
- most „data fetch“ without memory cycle

First Indications of Change

- PARS & Speed-up, Basel, Switzerland, March 2003: keynote
- IPDPS, Nice, France, April 2003: after a decade of non-overlap: first IPDPS people coming
- PDP'04, La Coruna, Spain, Feb. 2004: keynote
- IPDPS, Santa Fe, NM, USA, April 2004: keynote
- HPC Asia 2004 - 7th Intl Conference on High Performance Computing, July 20-22, 2004 Omiya Sonic City, Tokyo Area, Japan: Workshop on Reconfigurable Systems for HPC (RHPC) -> keynote

Conclusions

- RC has become mainstream in all kinds of applications
- CS education deficits: a curricular revision is overdue ...
- by a merger with the embedded systems mind set
  We need an academic grass roots movement, for ....
  ... free material & tools for undergraduate lab courses
to program and emulate small SW/CW/HW examples
all know-how needed readily available:

get involved!

HPC experts coming ...

- Simulation of Star Clusters: x10 speed-up by supercomputer-to-morphware migration (also molecular biology et al.)
- N-body problem went configware
- Reiner Spurzem, University of Heidelberg
  - Access to: ARL, Astronomisches Rechen Institut, founded 1700 in Berlin, moved 1945 to Heidelberg by August Kopff
thank you for your patience

MD-GRAPE-2 PCI board [1999]
4 chips MD-GRAPE-2 for N-body simulation
Converts a PC to 64 GFlops

...a curricular revision is overdue

Very high throughput on low power slow FPGAs may be obtained only by algorithmic cleverness* -- under the mind set of CW

*) still mainly ignored by our CS curricula

not your job?

next winner of the „Not My Job“ Award?
**Mega-rGAs**

```
System gates per rGA chip
```

- **1984**: 10000
- **1986**: 100000
- **1988**: 1000000
- **1990**: 1000000
- **1992**: 1000000
- **1994**: 1000000
- **1996**: 1000000
- **1998**: 1000000
- **2000**: 1000000
- **2002**: 1000000
- **2004**: planned

---

**Loop Transformation Examples**

- **sequential processes**:
  - loop 1-16
  - body
  - loop 1-8
  - body
  - endloop
  - fork
  - loop 1-8
  - body
  - loop 9-16
  - body
  - endloop
  - join
  - loop 1-2
  - trigger
  - endloop

- **resource parameter driven Co-Compilation**:
  - host: loop 1-8
  - trigger
  - endloop
  - resource: loop 1-4
  - trigger
  - endloop
  - loop 1-2
  - trigger
  - endloop

---

**Speedup by Xputers**

- **MoM architecture**: 2-D memory space, adj. scan window
- **grid-based design rule check example**
- **speed-up**: >1000
- **complex boolean expressions in 1 clock cycle**
- **address computation overhead**: 94%
Reiner Hartenstein, University of Kaiserslautern, Germany
http://hartenstein.de

Reconfigurable Computing:
a second programming domain

Migration of programming to the structural domain
The structural domain has become RAM-based
The opportunity to introduce the structural domain to programmers ...
... to bridge the gap by clever abstraction mechanisms using a simple new machine paradigm

The dichotomy of models

• Note for von Neumann:
  state register is with the CPU

• Note for the anti machine:
  state register is with memory bank /
  state registers are within memory banks

The structural domain is primarily data-stream-based:

Flowware ..... mostly not yet modelled that way:
  most flowware is hidden by its indirect
  instruction-stream-based implementation

Flowware converts „procedural vs. structural“ into „control-procedural vs. data-procedural“ ...

Traditional Environment

<table>
<thead>
<tr>
<th>machine paradigm</th>
<th>programmable</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction-stream-based</td>
<td>Software</td>
</tr>
<tr>
<td>data-stream-based</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Importance of binding time

not all switching is done by Configware

Configuration: like a kind of pre-packed frozen-in „super instruction fetch“

old CS lab course philosophy:
given an application: implement it by a program –/
new CS freshman lab course environment:
Given an application:
a) implement it by writing a program
b) implement it as a morphware prototype
c) Partition it into P and Q
c.1) implement P by software
c.2) implement Q by morphware
c.3) implement P / Q communication interface

All enabling technologies are available

• literature from last 30 years
• languages & (co-)compilation techniques
• anti machine and all its architectural resources
• parallel memory IP cores and generators
• morphware vendors like PACT ....
• anything else needed

„EDA industry shifts into CS mentality“ [Wojciech Maly]

• Microprogramming to replace FSM design
• Hardware languages replace EE-type schematics
• EDA Software and its interfacing languages
• Newer system level languages like systemC etc.
• Small and large module re-use
• Hierarchical organization of designs, EDA, et al.
• ....................

Reconfigurable Computing: a second programming domain

Migration of programming to the structural domain
The structural domain has become RAM-based
The opportunity to introduce the structural domain to programmers ...
... to bridge the gap by clever abstraction mechanisms using a simple new machine paradigm

Reconfigurable Computing

Using coarse grain morphware platforms leads to Reconfigurable Computing, which is really Computing,

whereas physical use of fine grain morphware (FPGAs etc.) means kind of Logic Design on a strange platform.
Software is the source for programming traditional hardwired processors (instruction-stream-driven: von Neumann machine paradigm and its derivatives).

For Configware and Flowware we prefer the anti machine paradigm - counterpart of von Neumann.

Flowware provides a (data-)procedural abstraction from the (data-stream-based) structural domain... a Trojan horse to introduce the structural domain to the procedural-only mind set of programmers.

Flowware education: no fully fledged hardware expert needed to program embedded systems.

Flowware heading toward mainstream

- Data-stream-based Computing is heading for mainstream
  - 1946: (von Neumann machine paradigm)
  - 1980: data streams (Kung, Leiserson)
  - 1989: anti machine paradigm
  - 1990: rDPA (Rabaey: coarse grain reconfigurable array)
  - 1994: anti machine high level programming language
  - 1995: super systolic array (rDPA)
  - 1996+: SCCC (LANL), SCORE, ASPRC, Bee (UCB), ...
  - 1997+: discipline of distributed memory architecture
  - 1997+: configware / software partitioning compiler

Flowware-based paradigms, methodologies

1946: (von Neumann machine paradigm)
1980: data streams (Kung, Leiserson)
1989: anti machine paradigm
1990: rDPA (Rabaey: coarse grain reconfigurable array)
1994: anti machine high level programming language
1995: super systolic array (rDPA)
1996+: SCCC (LANL), SCORE, ASPRC, Bee (UCB), ...
1997+: discipline of distributed memory architecture
1997+: configware / software partitioning compiler

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
Gokhale: Streaming Languages: a different mind set

"After a few years of looking at the problem, I realized that most of the application space could be described well with a stream-oriented communicating sequential processes model." Gokhale said, the compiler makes FPGA design available to software engineers, but they still should have an "abstract notion" of hardware: "One way to get performance is to tile application-specific arithmetic units across a chip," she noted. "Telling the compiler to unroll inner loops is a way to do that."

What’s not needed, she said, is a knowledge of the hardware at a clock cycle level.

In contrast to SystemC, which provides both behavioral and structural views, Streams-C is purely behavioral. It assigns operations to clock cycles, thus providing behavioral synthesis.

Stanford Streaming Languages: DSP-C

Set of language extensions to ISO C programming language allows application programmers describe key features of DSPs that enable efficient source code compilation:
- Fixed point data types,
- Divided memory spaces,
- Circular arrays and pointers

DSP-C uses arrays

Sections of the arrays are selected for use in calculations using array indices or array range specifications

www.dsp-c.org

Stanford Streaming Languages: Brook

Brook defines much more abstract streams
- Dynamic length streams
- Could be multidimensional (but only fixed length??)
- Entire stream is a monolithic object
- Stream programming language for modern graphics hardware

Flowware Languages

Specialized:
- **Brook**: for modern graphics hardware

**Streams-C**: defines 1-D streams; generates VHDL

**DSP-C**: allows to describe key features of DSPs

General purpose:
- **MoPL**: fully supporting the anti machine paradigm - the counterpart of the von Neumann paradigm
Streaming Languages: Parallelism

- Independent thread parallelism
  - Stick with pthreads or other high-level definition
- Loop iteration, data-division parallelism
  - Divide-up loop iterations among functional units
  - Loop iterations must be data-independent (no critical dependencies)
- Pipelining of segments of “serial” code
  - Find places to overlap non-dependent portions of serial code
  - Ex 1: Start a later loop before earlier one finishes
  - Ex 2: Start different functions on different processors
  - Harder than loop iteration parallelism because of load balancing
- Pipelining between time-steps
  - Run multiple time-steps in parallel, using a pipeline

- Ex. 1: Start a later loop before earlier one finishes
- Ex. 2: Start different functions on different processors
  - Harder than loop iteration parallelism because of load balancing
- Pipelining between time-steps
  - Run multiple time-steps in parallel, using a pipeline

AGUs

Speed-up Enablers

Hier eine Liste

DRC 4 orders of magnitude

Address computation overhead

Translate into super-systolic rather than into instruction streams

Determine interconnect fabrics by compilation, but not before fabrication

Determine memory architecture by compilation, but not before fabrication

Application-specific distributed memory

- Application-specific memory: rapidly growing markets:
  - IP cores
  - Module generators
  - EDA environments
- Optimization of memory bandwidth for application-specific distributed memory
- Power and area optimization as a further benefit
- Key issues of address generators will be discussed

Acceleration Mechanisms

- Parallelism by multi bank memory architecture
- Auxiliary hardware for address calculation
- Address calculation before run time
- Avoiding multiple accesses to the same data.
- Avoiding memory cycles for address computation
- Improve parallelism by storage scheme transformations
- Improve parallelism by memory architecture transformations
- Alleviate interconnect overhead (delay, power and area)

Significance of Address Generators

- Address generators have the potential to reduce computation time significantly.
- In a grid-based design rule check a speed-up of more than 2000 has been achieved, compared to a VAX-11/750
- Dedicated address generators contributed a factor of 10 - avoiding memory cycles for address computation overhead
### Smart Address Generators

- **1983** The Structured Memory Access (SMA) Machine
- **1984** The GAG (generic address generator)
- **1989** Application-specific Address Generator (ASAG)
- **1990** The slider method: GAG of the MoM-2 machine
- **1991** The AGU
- **1994** The GAG of the MoM-3 machine
- **1997** The Texas Instruments TMS320C54x DSP
- **1997** Intersil HSP45240 Address Sequencer
- **1999** Adopt (IMEC)

### Adopt (from IMEC)

- cMMU synthesis environment:
  - application-specific ACUs for array index reference
  - ACU as a counter modified by multi-level logic filter
  - ACU with ASUs from a Cathedral-3 library
  - distributed ACU alleviates interconnect overhead (delay, power, area)
  - nested loop minimization by algebraic transformations
  - AE splitting/clustering
  - AE multiplexing to obtain interleaved ASs
  - other features

### Linear Filter Application

- Parallelized Merged Buffer Linear Filter Application with example image of x=22 by y=11 pixel

### super systolic

- KressArray Family generic Fabrics:
  - a few examples
  - select nearest neighbour (NN) Interconnect: an example
  - more NNports: rich Rout Resources

---

Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004
Super Pipe Networks

The key is mapping, rather than architecture

<table>
<thead>
<tr>
<th>Super Pipe Networks</th>
<th>Pipeline Properties</th>
<th>Array Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KressArray principles

- take systolic array principles
- replace classical synthesis by simulated annealing
- yields the super systolic array
- a generalization of the systolic array
- no more restricted to regular data dependencies
- now reconfigurability makes sense

Hardware / Configware / Software Partitioning

- instruction-stream-based procedural
- data-stream-based structural

Algorithm

- mapping

Brain Usage: both Hemispheres

Hardware & Software Co-Design
Keynote address, IPDPS 2004, Santa Fe, NM, USA, April 26 - 30, 2004