

The Future of Reconfigurable Computing

A “Small Matter of Programming”

Duncan A. Buell

Department of Computer Science and Engineering
University of South Carolina

11 July 2005

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

Credits

- ▶ USC Faculty: Jim Davis, Gang Quan
- ▶ GWU Faculty: Tarek El-Ghazawy
- ▶ GMU Faculty: Kris Gaj
- ▶ USC Students: S. Akella, C. Cathey, L. Cordova, S. Devarkal, P. Kancharla, S. Rangunathan, H. Wake
- ▶ SRC Computers: David Caliga, Jeff Hammes
- ▶ (the other) SRC/IDA: Jeff Arnold and about 25 others

Credits(2)

- ▶ This represents a lot of work on the part of many people.
- ▶ I have myself put fingers to keyboard, so some of these experiences are my own.
- ▶ But a lot of the experience is that of others, to whom I am truly grateful.
- ▶ The opinions are my own, unless they are incorrect, in which case they were provided to me by others.

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

A Story from 1979

- ▶ I was contracted to moonlight writing code for a new “personal computer”
- ▶ I discovered the “sort” routine was a disk-to-disk bubble sort, and in executing one single sort it scratched a fully-loaded disk so badly that the disk was unreadable.
- ▶ I rewrote the sort.
- ▶ I wrote a short article for *80 Microcomputing*, naming the company.

A Story from 1979

- ▶ I was contracted to moonlight writing code for a new “personal computer”
- ▶ I discovered the “sort” routine was a disk-to-disk bubble sort, and in executing one single sort it scratched a fully-loaded disk so badly that the disk was unreadable.
- ▶ I rewrote the sort.
- ▶ I wrote a short article for *80 Microcomputing*, naming the company.
- ▶ Then I got a letter from the company.

A Story from 1979(2)

Radio Shack

A Division of Tandy Corporation

TANDY SYSTEMS DESIGN

817-390-3011

1100 ONE TANDY CENTER, FORT WORTH, TEXAS 76102

March 24, 1980

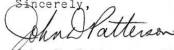
Dr. Duncan A. Buell
Assistant Professor, Dept. of Computer Sciences
102 Nicholson
Louisiana State University
Baton Rouge, LA 70803

Dear Dr. Buell:

I saw your article in 80 Microcomputing. You know we need help. Would you be interested in coming to work for Tandy Corporation? We think that personal computers is one of the most exciting and fastest growing fields in electronics. We need outstanding people that can meet the challenge and grow with us.

Please send me a resume or call me if you are interested. Tell your friends and students to contact us as well.

Sincerely,



John D. Patterson, PhD.
Director, Tandy Systems Design

JDP/mc

A Story from 2005

From: openfpga-bounces@osc.edu on behalf of "A"
Sent: Thu 4/14/2005 8:43 AM
To: OpenFPGA
Subject: [Openfpga] "B" artical in Journal "C"

All-

Just want to pass along a nice summary of the
"D" in Journal "C".

http://www.journal_address.html

Openfpga mailing list

Openfpga@osc.edu

<http://email.osc.edu/mailman/listinfo/openfpga>

A Story from 2005(2)

From: BUELL, DUNCAN A

Sent: Saturday, April 16, 2005 9:23 AM

To: openfpga@osc.edu

Cc: comments@journalc.com; buell@sc.edu

Subject: FW: "D" artical in Journal "C"

FLAME ON

Do not get me wrong. I am a big fan of "D" ...

(some reasoned commentary and sage opinion)

FLAME OFF

Duncan A. Buell

A Story from 2005(3)

From: (article author)

Sent: Saturday, April 16, 2005 5:29 PM

To: BUELL, DUNCAN A; openfpga@osc.edu

Cc: buell@sc.edu; someone_else

Subject: (the article)

KNEE-JERK DEFENSIVENESS ON.

(text)

DEFENSIVENESS OFF. THOUGHTFUL RESPONSE ON.

(text)

Any and all replies are welcome!

Send to comments@journalc.com

(article author)

A Story from 2005(4)

From: David Pointer [pointer@ncsa.uiuc.edu]
Sent: Wednesday, May 11, 2005 3:42 PM
To: BUELL, DUNCAN A
Subject: Challenge the RC industry?

Hello Duncan,

As I mentioned in an earlier email, ...

I remember your openfgpa email ...

Would you be interested in opening the summer institute on July 11 by challenging the RC industry? ...

Regards,
Dave

An Excerpt from My Flame

“We made do with what we had in 1992-1994 because at that time we had no choice. It was successful but certainly not without its detractors. It is disappointing that we are still apparently stuck at the same place we were over a decade ago.

The problem in 1992 was not hardware but programming. The problem in 2005 is not hardware but programming. The problem is that use of the hardware is too much like “design”, a word I tried (unsuccessfully) to purge from the lexicon of the project people I worked with. The fact that in this past week’s phone call someone said that the default standard for these machines was VHDL says that people are still not thinking seriously enough about getting applications to work. Yes, the default standard is VHDL and yes, that is precisely the problem!”

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

A Recap of Reconfigurable Computing History

(Serious Reconfigurable Computing Platforms)

- ▶ 1987-1990 SPLASH, Supercomputing Research Center
 - ▶ programmed in LDG or designed with schematic capture
 - ▶ excellent hardware, significant speedup,
 - ▶ but hard to program, so the number of apps was limited

- ▶ 1992-1994 Splash 2, Supercomputing Research Center
 - ▶ programmed in VHDL
 - ▶ excellent hardware
 - ▶ programming nonstandard, but it was in fact programming

- ▶ 1986-1995? Programmable Active Memories (PAM, PAMette), DEC Paris
 - ▶ programmed in C++ (allegedly)
 - ▶ great hardware, but limited apps

A Recap of Reconfigurable Computing History(2)

- ▶ Many, many small boards for teaching digital logic
- ▶ Many smallish boards for embedded systems, niche applications
- ▶ No new large “computing” platforms until the new SRC/SGI/(Octiga Bay=Cray) systems

The Must-Have Issues of Reconfigurable Computing

- ▶ Enough hardware to do real, useful, full-scale applications
 - ▶ Enough bandwidth to keep the logic fed with data
 - ▶ Auxiliary memory (to function like a cache)
- ▶ Several, different, applications demonstrating that this is not a special purpose device masquerading as a computer
- ▶ A factor of 50 to 100 speedup so that a business case can be made that a “different” computer is an OK strategy
- ▶ “Code development” must be sufficiently easy that one would *choose* to use this hardware to do “the next” application

Issues and Answers

Four conclusions from Splash 2, 1994; all are still true today

- ▶ Memory is necessary
- ▶ I/O bandwidth is necessary
- ▶ Programming is possible
- ▶ The programming environment is crucial

An Analogy with Parallel Computing

- ▶ In the 1980s there were lots of parallel computers for sale
- ▶ High performance required tuning specifically for the architecture
- ▶ Code was not portable
- ▶ Performance results were hard to compare

An Analogy with Parallel Computing

- ▶ In the 1980s there were lots of parallel computers for sale
- ▶ High performance required tuning specifically for the architecture
- ▶ Code was not portable
- ▶ Performance results were hard to compare
- ▶ All the companies died

An Analogy with Parallel Computing

- ▶ In the 1980s there were lots of parallel computers for sale
- ▶ High performance required tuning specifically for the architecture
- ▶ Code was not portable
- ▶ Performance results were hard to compare
- ▶ All the companies died
- ▶ MPI is much less efficient, but standard and portable
- ▶ Beowulfs exist in vast numbers

An Analogy with Parallel Computing

- ▶ In the 1980s there were lots of parallel computers for sale
- ▶ High performance required tuning specifically for the architecture
- ▶ Code was not portable
- ▶ Performance results were hard to compare
- ▶ All the companies died
- ▶ MPI is much less efficient, but standard and portable
- ▶ Beowulfs exist in vast numbers
- ▶ There's a lesson here

An Analogy with Parallel Computing(2)

- ▶ Practically every application is programmed on a different hardware platform and in a different “programming” mode
- ▶ The platforms are small and have resource constraints
- ▶ The code is not portable
- ▶ Good results are highly dependent on tuning for the resource constraints

An Analogy with Parallel Computing(2)

- ▶ Practically every application is programmed on a different hardware platform and in a different “programming” mode
- ▶ The platforms are small and have resource constraints
- ▶ The code is not portable
- ▶ Good results are highly dependent on tuning for the resource constraints
- ▶ Haven't we been here before?

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

New Hope for the Future

- ▶ Three serious platform vendors (SRC, SGI, Cray)
("serious" \iff a complete computer, not just a board)
- ▶ Significant motion in response to vendors' offerings
- ▶ An existence proof (SRC) that programming is possible

New Hope for the Future

- ▶ Three serious platform vendors (SRC, SGI, Cray)
("serious" \iff a complete computer, not just a board)
- ▶ Significant motion in response to vendors' offerings
- ▶ An existence proof (SRC) that programming is possible
- ▶ **SRC**: Other programming approaches may be following the SRC lead

New Hope for the Future

- ▶ Three serious platform vendors (SRC, SGI, Cray)
("serious" \iff a complete computer, not just a board)
- ▶ Significant motion in response to vendors' offerings
- ▶ An existence proof (SRC) that programming is possible
- ▶ **SRC**: Other programming approaches may be following the SRC lead
- ▶ **Others**: Other programming approaches may be paralleling SRC

New Hope for the Future

- ▶ Three serious platform vendors (SRC, SGI, Cray)
("serious" \iff a complete computer, not just a board)
- ▶ Significant motion in response to vendors' offerings
- ▶ An existence proof (SRC) that programming is possible
- ▶ **SRC**: Other programming approaches may be following the SRC lead
- ▶ **Others**: Other programming approaches may be paralleling SRC
- ▶ Motion towards standardization

Programming

- ▶ To be a “computer” it must be programmable
- ▶ Reconfigurable computing can't be just for electrical engineers
- ▶ “Design” → “hardware” must be a process familiar to programmers
- ▶ “Debugging” must look and feel like debugging

SMOP

SMOP: /S*M*O*P/, n.

[Simple (or Small) Matter of Programming]

1. A piece of code, not yet written, whose anticipated length is significantly greater than its complexity. Used to refer to a program that could obviously be written, but is not worth the trouble. Also used ironically to imply that a difficult problem can be easily solved because a program can be written to do it; the irony is that it is very clear that writing such a program will be a great deal of work. "It's easy to enhance a FORTRAN compiler to compile COBOL as well; it's just a SMOP"

2. Often used ironically by the intended victim when a suggestion for a program is made which seems easy to the suggester, but is obviously (to the victim) a lot of work. Compare "minor detail".

The Jargon File, <http://catb.org/~esr/jargon/html/S/SMOP.html>

SMOP

All we really need is a basic programming environment,
...portable across (or at least comparable across) several
platforms,
...so we can get a number of applications programmed with
a factor of 100 speedup on a number of platforms,
...and with enough wiggle room in the implementations that
we we can tweak the parameters and the method of
implementation.
...and all written in C (okay, maybe also Fortran).

Who Needs to Be Convinced?

- ▶ Put yourself in the shoes of middle management running a software shop.
- ▶ You look good if this quarter's work is done early or cheaply.
- ▶ Will you invest in extra hardware and extra training?
- ▶ What are the risk and cost? What's the likelihood of success?
- ▶ Is there a good enough reason not to take the safe route and just buy more hardware?

Why Not Just VHDL/Verilog/Whatever?

- ▶ Yet another programming language, debugging style
- ▶ Ten times more programmers than hardware designers in the world
- ▶ HDL tools aren't really intended to be “compilers”
- ▶ Synchronizing vendor releases, licenses, patches, ...
- ▶ The **programming environment** is critical

“Programming” Is Not “Hardware Design”?

- ▶ It is my experience, from watching hardware designers, that the way they attack a design problem is different from the way in which programmers flesh out the code for an application.
- ▶ The programming process is top down, stubs and templates turning into subprograms, and a lot of bottleneck-ology.
- ▶ Hardware design seems to be done more “in the large” and less by stepwise refinement.
- ▶ If this is true, it is not surprising that HDL tools support an environment different from what programmers are familiar with.

What Is This “Programming Environment?”

- ▶ The ability to do functional correctness testing without the hardware
- ▶ Hints from the compiler on the efficiency of synthesized code
- ▶ The ability to interact with the memory, communication paths, etc., [in software](#)
- ▶ The ability to go from software emulation/simulation to execution on the hardware with no effect except increased speed
- ▶ Dropping below HLL implementation only when the HLL syntax is incapable of being synthesized efficiently

We Do Not Need Optimal Performance

- ▶ C to FPGAs will not be as efficient as detailed hardware design
- ▶ Various comparisons come up with 2X loss in speed and area
- ▶ We will win, in spite of the 2X/4X loss, with programmer productivity
- ▶ Ten apps running 50X faster is better than two apps running 100X faster
- ▶ Ride the technology curve with portable code

Outline

Credits

How We Got to This Talk

A Recap of History

A Small Matter of Programming

The Path Forward

Some Things That SRC Has Done Right

SRC Computers has an integrated hardware/software solution that actually works. What did they do?

- ▶ MAPC code is a `whatever.mc` module to go along with the host computer `othercode.c` modules
- ▶ Seamless compilation/synthesis for either simulation or hardware execution
- ▶ Debugging is possible entirely in software mode
- ▶ Excellent compiler messages on compiler-inserted ticks in loops
- ▶ Print statements, timing calls work only when they are supposed to
- ▶ Vendor macros for when compiling would insert ticks
- ▶ OpenMP-like parallelism plus vector-like loop optimization

Some Hardware Issues That Should Be Ignorable

- ▶ Timing below the increment of “one tick”
- ▶ Combinational versus sequential objects
- ▶ “Optimal” use of the hardware, instead of optimal use of the people
- ▶ General purpose synthesis—ignorable because programs are highly structured

Some Hardware Issues That Cannot Be Ignored

- ▶ Finiteness of resources (memory, silicon)
- ▶ Memory bank accesses and conflicts
- ▶ Loop dependencies that make one tick per iteration impossible
- ▶ KISS—not everything can easily be synthesized
- ▶ Explicit parallelism of functional units
- ▶ Tick-by-tick execution and flow of data

Isn't this “ordinary” high-performance computing?

This is **hardware-aware**, but not necessarily **hardware-savvy**, computing.

Is SRC Special?

Yes, maybe

- ▶ Cray and SGI don't have a software solution
- ▶ The SRC compiler is constantly improving
- ▶ Maybe it is required that the compiler be tailored for the hardware

No, maybe not

- ▶ Mitrion is about to have a similar product
- ▶ ImpulseC, Celoxica, CatapultC ... ??
- ▶ Maybe one HLL can result in compilers for different hardware configurations
- ▶ Maybe the generic compiler approach is the analogue to MPI and will be more successful