Computing in the age of parallelism: Challenges and opportunities

Jorn W. Janneck

jwj@cs.lth.se

Computer Science Dept.
Lund University

Multicore Day, 23 Sep 2013
why we are here

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
Dotted line extrapolations by C. Moore.
C. Moore, Data Processing in ExaScale-Class Computer Systems, April 2011
Circuits are still getting larger at a steady pace...

… but speed-up is no longer the result of a faster clock.

Instead, we get more parallel devices.
Physics of computing - efficiency

Performance = parallelism.
Efficiency = locality.
It really is that simple.

Bill Dally

Approximate power in 28nm processor

Bill Dally
Pentium IV ("Northwood")

Core i7 980X ("Gulftown")

Epiphany-IV
programming parallel machines
programming parallel machines

C w/ “iLib”

SOPM ("ajava")

C + VHDL

CUDA

Virtex

HDL

VENTUREFORTH

CUDA

VENTUREFORTH

Virtex

HDL
portability

then

- instruction set
- number/kind of registers
- address space (size, structure)
- system software
  - OS
  - GUI
  - ...

now

- all of the above, plus...
  - number and kind of processing elements
  - … and possibly other 'stuff', like memories
  - topology of interconnect

unicore

FPGA

multicore

processor array

GPU
programming sequential machines
C, Pascal, Forth, Java, Lisp, Fortran, Haskell, ...

* for the purpose of this talk “von Neumann machine” ≈ “single stored-program computer”
a pivotal model of computing

programmers

compiler writers

language designers

computer architects
programming parallel machines

- Processor 1
- Multi-core 2 ~ 10s
- MPPA, GPU 10s ~ 100s ...
- FPGA 100s ~ 1000s

- Pentium IV
- Penryn quadcore
- Am2045
- Nvidia GT200
- Virtex
Implement streaming applications efficiently on a wide range of computing platforms.

language, methodology
VHDL/Verilog synthesis
software synthesis, IDE
profiling, hardware synth., STHORM
profiling, tools, software synth.
software synthesis, applications
software synthesis, applications

UC Berkeley
Xilinx
Ericsson
EPFL
Lund University
INSA Rennes
Halmstad Högskolan

our work

Pentium IV
Penryn quadcore
Am2045
Nvidia GT200
Virtex

processor
1
multi-core
2 ~ 10s
MPPA, GPU
10s ~ 100s ...

FPGA
100s ~ 1000s
streaming applications

how a video expert explains an MPEG decoding algorithm

digital signal processing
media processing
video/audio coding
image processing / analytics
networking / packet processing
automatic control
cryptography
gene sequencing
...

**dataflow**

- **computational kernels** *(actors)*
  - provide explicit concurrency
  - execute in discrete atomic steps

- **directed point-to-point connections**
  - lossless, order-preserving
  - conceptually unbounded
  - asynchrony, abstraction from time

- **communicating streams of discrete data packets** *(tokens)*
  - no shared state

Some things **not** represented in this model:

- **time** (incl. speed of actors and communication channels)
- **execution policy** of actors
- communication protocol and communication **medium**
- **location** and implementation technology of actors

MPEG/ISO standardized RVC-CAL, a subset of our actor language, in 2010.
implementation

parsing, translation

composition, scheduling

code generation

partitioning

architecture

interconnect
questions we try to answer

about stream programs

Is it determinate?
Does it do what it is supposed to do?
Is it bounded?
Does it deadlock?
Can it be implemented efficiently?

about their (more or less) parallel implementations

Does it still work?
Does it deadlock now?
Is it sufficiently fast/efficient/small ...?
What/how many resources does it use?
dataflow profiling

A trace is a DAG
- vertices are steps
- edges are dependencies

dependencies mediated by
- tokens
- state variables

trace: the structure of a computation
analysis: post-mortem scheduling

What if we...
... changed latency somewhere?
... multiplexed resources?
... did so dynamically?
... used different communication fabric?
...
trace analysis
trace metrics (examples)

input (bytes)

power (∑ power)

latency (incremental)
Critical path:

the **longest** time-weighted sequence of events from the **start** of the program to its **termination**.
Impact Analysis: identifying potential for improvement
**Objective:** Evaluate a trade-off between the buffer size and the critical path length

- Minimum buffer size configuration for a given throughput
- Optimize only critical buffers (*i.e.* buffers that block steps in the current critical path)
critical path & buffer optimization

- Resonable optimal buffer size configuration
- The throughput is not significantly improved
a pivotal model of computing

programmers

language designers

teachers, too!

compiler writers

computer architects

Figure 1.12
Organization of the CPU and main memory of the IAS computer.
There Is No More Sequential Programming. Why Are We Still Teaching It?
panel title at the Intel Academic Forum, Supercomputing 08

Parallel Programming
- preemption
- tasks
- synchronization
- communication
- determinacy
- deadlock
- threads
- semaphores

Programming
- objects
- programming languages
- types
- design patterns
- variables
- data structures
- complexity
- design-by-contract
- algorithms
- loops
- processors
- memory management
an alternative?

My take

- Modeling, simulating, and programming parallel and concurrent systems is a more fundamental problem than how to make use of multicores efficiently.
- Freshman teaching should focus on composing parallel programs; sequential programming should be taught (perhaps) as a way of writing the modules to be composed.

Within a few years multicores will be viewed as a transparent way of simplifying and speeding up parallel programs (not very different than the way we used to view computers with faster clocks).
These are exciting times.

Computing is losing its North Star.

Let's imagine that most computers have scores or hundreds of general-purpose cores. (And let's say we do not know exactly how many there are, or how they are connected.)

Or programmable logic, or combinations of both.

How would that affect...

– our idea of an algorithm?

– the languages and abstractions we use to program such a machine?

– the concept of a module?

– the things we need to do to validate programs?

– the techniques for making programs efficient?

– the concepts and the implementation of an operating system?

– the way we teach programming?
Thanks!


**opendf.org**
Open Dataflow portal

**opendf.sf.net**
open source tool suite

**orcc.sf.net**
Open RVC-CAL Compiler

**www.actors-project.eu**
ACTORS Project

**mpeg.chiariglione.org**
MPEG

jwj@cs.lth.se