Heterogeneous Multicores?

Why?

David May
Bristol University and XMOS

The past

Heterogeneous architectures have been around for a long time

... to provide big processors with agile input-output processors

... to add specialised accelerators

... to build general purpose parallel computers

Often using lots of processors

Input-output processors

IBM 360 and its channels

CDC 6600 and its input-output processors

... and now, ARM

Is a chip with several different ARMs heterogeneous?

Note: the IBM 360 was *one* architecture with multiple implementations

Specialised accelerators

Illiac 3

Attached numeric processors

Evans and Sutherland

Pixel planes

REYES / Renderman

... and a lot of processor/co-processor combinations for DSP and graphics

General purpose (?)

Illiac 4

Vector processors - CDC and Cray

DAP

SIMD within GP (VIS, MMX, Chameleon)

Scalar-Vector nodes for HPC (and now Scalar-SIMD nodes)

REYES / Renderman again

The last 20 years

We have known how to do general purpose parallel computing since 1990

But the explosive growth of PCs has taken us in a different direction

Implementing Moore's 'law' is only possible with exponential market growth

But there has been exponential market growth so we have spent 20 years improving on 1980s technologies

... most superscalar techniques date from the 1960s (IBM 360/91)!

The Present

Efficiency of ‘sequential’ processors is static or declining

Most systems have multiple processors

The superscalar applications processors can’t do input-output fast enough

So we need to add input-output processors

The only way to add more performance is more processors

... but what kind of processors?

Graphics

Graphics is an established and well-understood parallel application

It has become a technology driver and can exploit a lot of processors

It makes (some) sense to design application-specific graphics hardware

... especially if the hardware is replaced every year

Graphics processors are applicable in *some* other areas

... such as High Performance Computing

High Performance Computing

In High Performance Computing, aim to maximise peak FLOPS

peak = "the performance we guarantee you can't exceed"

It's not important that the FLOPS are useful or not - it's the Top 500 list that matters

More seriously, there are quite a few data-parallel HPC applications

But the *general purpose* processing - and the interconnect - is what really affects performance and efficiency

... along with the languages, tools and algorithms

Parallel Processing

Need general purpose architectures and languages

... not a variety of special purpose architectures and languages

It's always possible to execute parallel programs on sequential processors

... but the converse isn't true

Want to write portable, re-usable, parallel software

But - as in 1990 - there is a lack of standardisation

Parallel Program Patterns

Parallel Random Access Machines (PRAMs/BSP)

Data Parallelism / Process Arrays

Directed Dataflow Graphs

Task Farms and Server Farms

Event handlers

Recursive use of any of the above

The SIMD - MIMD issue

... an old debate!

The cost of instruction handling in a simple processor is very low

Local execution and synchronisation avoids global communication

A MIMD array will run faster and at lower energy than a SIMD array

MIMD can emulate SIMD, but the converse doesn't work

We've already shown how MIMD can unify computation, communication, synchronisation and event-handling (XMOS)

Heterogeneous architectures

Programming is made complex by multiple tool chains and languages - and arbitrary architectural restrictions

If we want heterogeneity, we want it in the *implementation*, not the *architecture*

... one programming model, several optimisations

But unless you're optimising a chip for a specific application program you don't know what the ratio between the heterogeneous components should be

... potentially need a range of chips with varying ratios

Heterogeneous architectures

The distribution of physical resources on a heterogeneous chip is fixed

... the chip is (almost) an application specific processor

If a new algorithm or optimisation requires a different distribution

... there will be a loss of efficiency

... or a different chip will have to be used

But algorithms have by far the greatest impact on performance and efficiency

What's gone wrong?

A general purpose parallel computer should be re-programmable - by normal people

Software was supposed to be portable - so why do we install a specific core to run it?

Huge market volumes have supported Moore's Law investments

Moore's Law investments have enabled growth of everything - especially complexity

Persistent ideas about cache-coherent shared memory are inhibiting simple ways of using parallelism

Design by aggregation

... it started in software design and it has spread into hardware design

Take big components from libraries and put them together

There is confusion about ‘abstraction’ - its supposed to be about managing complexity, not hiding it

You can’t risk redesigning complex sub-components - or porting complex software

The ultimate lock-in! Too-complex-to-understand! Too-big-to-fail!

... but it will

Is there an alternative?

We have known how to do *homogeneous* universal parallel computing for 20 years

The issue is the interconnect - we don't want heterogeneity here

Build everything directly on top of scalable message passing

The processors just have to keep up with the interconnect and they don't have to be complicated

It probably doesn't matter if there are different processor architectures

... but - apart from input-output processing - it's probably not worth it

General Purpose Parallel Processing

Think in terms of *Processes and Communication Patterns*, not *Algorithms and Data Structures*

Scalable interconnect to support *any* communication patterns (eg Clos networks, not 2-D meshes)

Low latency communication enables high parallelism and rapid spread of computation; bounded latency enables latency hiding by the processors

Collections of processors used to execute large sequential programs

Design the interconnect; then design the processors to keep it busy

Universality and Efficiency

Universal parallel machine = Universal processors + Universal interconnect

The processors can be optimised for efficiency, not speed

Clos networks implement *permutations* on their inputs

Compiler can implement known permutation patterns optimally

Efficient hashing techniques can be used for unknown patterns

Potential $log(p)$ emulation overhead but can be hidden by $log(p)$ *excess parallelism*

Commodity parallel processing

Ideally, we want to build processing like memory

Choose an economical chip size ($70mm^2$ for DRAM), $100mm^2$ for logic

... this will hold hundreds of processors

Stack them up in 3D using through-silicon-vias.

Connect them using silicon photonics and wavelength division multiplexing

General purpose components with behaviour defined by software

The future - an optimist's view

The new generation is getting bored with more and more graphics and VR. Reality is more interesting - 'Things'

Soon we will realise we shouldn't be throwing away a billion phones every year - where are they?

General purpose parallelism will emerge. It will surprise you and it will enable new applications and products

There will probably be heterogeneity at the (many) interfaces - now we're doing Reality and Robotics, not VR

But there may also be interesting new places for heterogeneity such as in classical + quantum computing

The open computing project

How simple can we make a parallel processor?

Simple deterministic processing node

Non-blocking network

Concurrent programming language and operating system

Implementable in anything - including emerging technologies

Nothing hidden by abstraction layers

And finally ...

2014 will be the 30^{th} anniversary of the launch of the Transputer and Occam

It would be good to re-construct it

It would be good to build a 2014 version

It might be very efficient - and a 2014 chip could hold 1000s

If you'd like to help, get in touch ...

And at the very least, this a a good excuse for a party!