UNIVERSITY OF
CAMBRIDGE

Programming Language Design and Analysis motivated by Hardware Evolution

Alan Mycroft

Computer Laboratory, University of Cambridge

`http://www.cl.cam.ac.uk/users/am/`

24 August 2007

What's the issue?

Hardware keeps changing (as ever) but change to multi-core processing require a major rethink (i.e. opportunities for us!) to:

- Programming Languages
- Analysis Techniques
- Opportunities for Static Analysis

Aim of this talk

- Tutorial on why and how hardware is changing (stress points)
- Neat ideas for research (completed/starting/to be done) for this community.

What's the issue (2)?

Head in the sand:

Of course, you can always write and use small programs and never use the additional parallelism (editors, latex, browsers ...), and never write papers on the new implications. Only then can you ignore these technology changes.

Plan of the talk

- Changes in Technology
- Programming Implications (for Programmers and for Languages)
- Opportunities for Static Analysis/Type Systems

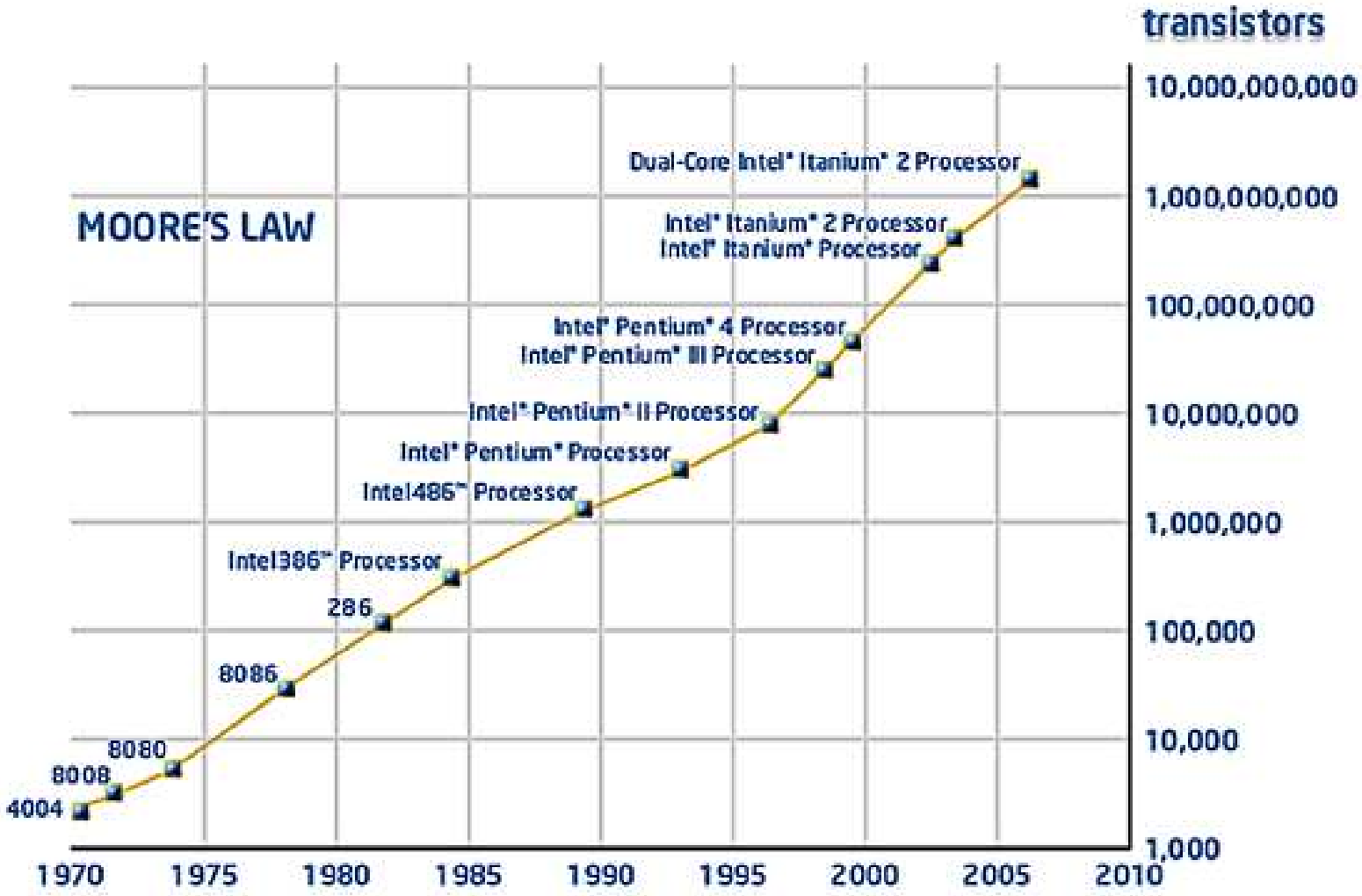
What has *suddenly* changed?

UNIVERSITY OF
CAMBRIDGE

1. Nothing
2. We have hit a design dead-end

Which is true? Both?

Moore's Law and Scaling



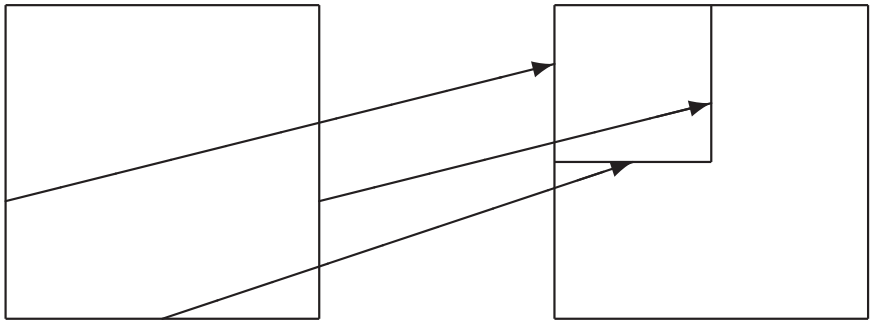
Moore's Law and Scaling (2)

- Moore's Law: the empirical observation the number of transistors per unit area on a chip doubles every 18 months. Originally formulated in 1965 as doubling every 2 years, but 18 months gives a better fit to reality.
- Originally backward looking, but industry adopted it as a design objective and the ITRS (The International Technology Roadmap for Semiconductors) envisages its continuation to 2020!
- This means that a 2020 (say 12 years time) chip will have 256 times as many transistors as today's chips.

Moore's Law and Scaling (3)

UNIVERSITY OF CAMBRIDGE

So, every 3 years linear dimensions shrink by a linear factor of 2.



Chip dimensions (chosen for largely economic reasons) have tended to stay pretty constant, e.g. 1cm x 1cm. (Bigger, more functional, chips can be sold for more but the probability of a defect on them tends to 1 with increasing size.)

What does this scaling do electrically?

Moore's Law and Scaling (4)

Scaling: electrically each smaller component has (to first order)

- capacitance C reduced by 4
- resistance R unchanged
- hence switching speed ($f = 1/RC$) 4 times faster
- power per component ($f.CV^2$) unchanged at same voltage
- power per chip increased by 4 [HEAT!]

But, in practice we reduce voltage swings (which reduces the speed increase) so that we can still go (say) twice as fast, but with the new chip generating not much more power than the old one.

Moore's Law and Scaling (5)

But: while feature size goes down uniformly down in all designs, speeds do not scale as well as suggested:

- in 1985 off-chip RAM and CPUs went at the same speed, now accessing RAM can take 200 cycles.

So much of Moore's Law gain in transistor budget has been spent on caches to try to hide this.

Reduced voltage swings and smaller feature sizes mean less reliable components (e.g. cosmic rays and statistical thermal noise).

[NEW RESEARCH AREA: programming with unreliable components] Design and analysis for unreliability (e.g. lambda-zap [Walker et al.]). Redundant computation and voting logic. ECC. Do you want every last video pixel to be accurate?

What has *suddenly* changed (reprise)?

1. Nothing – Moore's Law of exponentially increasing (over time) transistor densities (i.e. exponentially decreasing 'feature size') continues unchanged. And individual components continue to switch faster.

2. But these improvements do not translate into faster x86-style processors – basically we don't know what to do with all the additional transistors on a chip. Whole systems do not run faster.

To misquote somewhat (Moore's law is size not speed):

> "Moore's Law is dead. [for ever-faster x86 architectures]
> Long live Moore's Law! [ever more and faster components]"

The big problem: long wires

Scaling 4 disjoint copies of a chip onto one with smaller feature size in principle is not a problem—apart from getting four-times as many pins on the new chip!

However, that's no fun—we want these chips to communicate.

The problem is that long thin wires are very resistive (hence slow).

- OK if wire length co-scales with features, but:
- What about a 1mm copper wire at feature-size width?
  ITRS: 111ps in 2006 rising to 1ns in 2013!

Sounds small, but 1ns/mm delay means 75 clock ticks for a cross-chip round-trip on a 2.5GHz 15mm wide chip!!

Aside: synchronous design

In elementary digital electronics courses, we are told to arrange our circuit with all the clock wires connected to a common clock (the synchronous assumption).

- this design style breaks down if it takes more than one clock cycle to get across the chip
- so we go slowly, or use huge (millions of transistor size + heat) networks for clock-distribution

The big problem: long wires (2)

Birds eye view: technology scaling favours computation over communication.

Wires are no longer free, and local re-computation is far better than sharing computation with a distant place.

NEW RESEARCH AREA:
E.g. EPSRC grant "C3D: Communication-Centric Computer Design" (Moore Greaves Mullins Mycroft).

The big problem: long wires (3)

Of course, while we have a slow long wire across a chip, we can put gates on it for free (in fact we may need buffers to amplify the signal anyway).

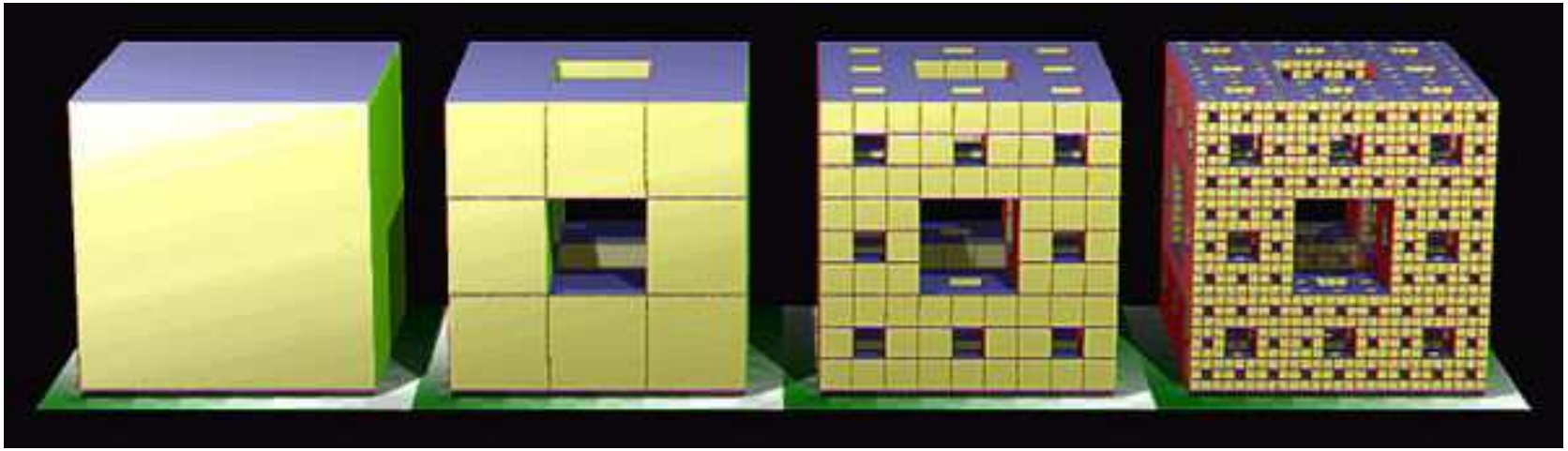
This gives the idea of Network-of-Chip (NoC). [Not this talk]

NB: moving data across a chip will have large *latency* but may still have high *bandwidth* so that bulk transmission of a data structure to another CPU may be possible.

Rent's Rule – Fractals

Fractals: self-similar structures – look the same (or similar) at all magnifications.

- Coastline
- Mandelbrot set, Julia set, Menger set:



Hausdorff (fractional) dimension [Menger is 2.73].
Power law: $T = T_0 g^d$. Ordinary cube $8 = 2^3$.

Rent's Rule – circuits

Another empirical law:

- Rent (1960): external pins T on a chip are $T_0 g^p$ where g is number of internal components.
- Donath: applies also to wire-length distribution.

Designs which minimise the exponent offer promise [fewer long wires].

- System Level Interconnect Prediction Workshop.

Intriguing connection to software: top-down design is statically fractal. Many algorithms are dynamically fractal except for global memory. NEW RESEARCH DIRECTION: static analysis and optimisation for dimension?

Summary: Why don't we have 5GHz or 10GHz Pentiums?

UNIVERSITY OF CAMBRIDGE

Because 3GHz seems a natural economic limit for this style of processor.

Indeed, we now buy two 2.5GHz processors ('dual core') instead, and next year quad-core.

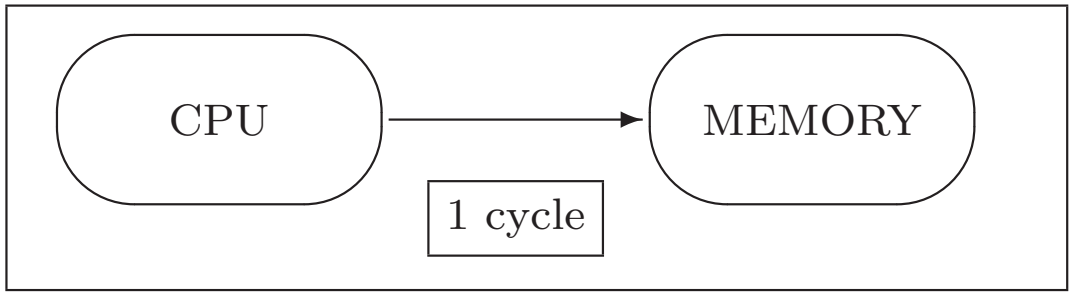
This isn't just marketing—we've seen the technological forces above.

As an aside note that merely distributing the clock on fast Pentium processors could take 25% and more of the total power (=heat output) of the chip.

Plan of the talk

- Changes in Technology — why now?
- Programming Implications (for Programmers and for Languages)
- Opportunities for Static Analysis/Type Systems

A programmer's view of memory

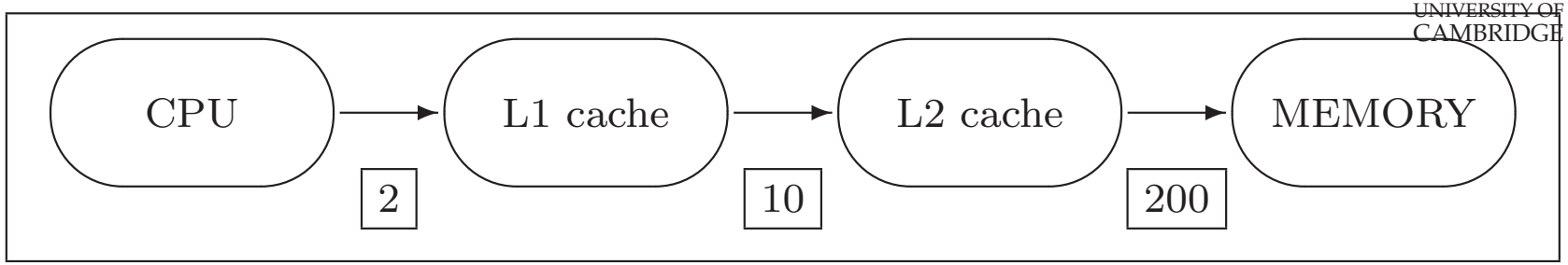


This model was pretty accurate in 1985. Processors (386, ARM, MIPS, SPARC) all ran at 1–10MHz clock speed and could access external memory in 1 cycle; and most instructions took 1 cycle.

Indeed the C language was a expressively time-accurate as a language: almost all C operators took one or two cycles.

But this model is no longer accurate!

A modern view of memory timings



So what happened? On-chip computation (clock-speed) sped up faster (1985–2005) than off-chip communication (with memory) as feature sizes shrank.

The gap was filled by spending transistor budget on caches which (statistically) filled the mismatch until 2005 or so.

Techniques like caches, deep pipelining with bypasses, and superscalar instruction issue burned power to preserve our illusions. 2005 or so was crunch point as faster, hotter, single-CPU Pentiums were scrapped. These techniques had delayed the inevitable.

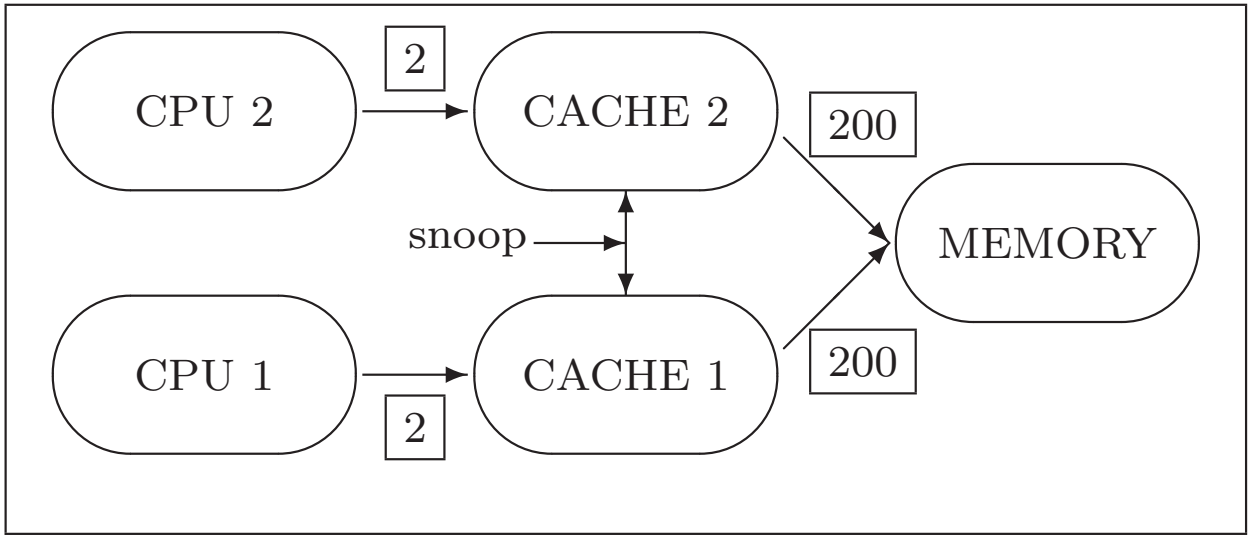
Multi-core chips or Chip Multi-Processors (CMP)

For much of the past 20 years chip designs (as per previous slide) were distorted to support faster and faster single-core processors. (10 years of Moore's Law's additional transistors had been spent on caches, pipelines, superscalar processing without being visible at the user-level.)

Now the dam is broken, more innovative designs are possible. We can spend the vast transistor budget on many and varied things; what would you like to spend it on?

[VAST NEW RESEARCH AREA!]

Multi-core-chip memory models



Today's model:

Will scale to 2, 4 maybe 8 processors. But ultimately shared memory becomes the bottleneck (1024 processors?!?).

NEW [or ancient] RESEARCH AREA: Programming languages without a uniform memory model?

Multi-core-chip memory models (2)

- Can program analysis help design memory models?
- Note temporal order of writes may even vary according to different observers (research on "Java Memory Model" problems)

```
int a=7, b=6, x;
par { {a=11; b=10; }   |   { x = a-b; } }
```

Can `x` be negative?

[NEW-ish RESEARCH AREA:] shared memory algorithms parameterised by memory model. Analysis? Refactoring?

Multi-core state of the art

- Intel's Terascale chip: 80 cores (8x10 array) each running at 3.1GHz all on a 2.75cm^2 chip. On-chip network to connect to other chips and memory. Plus a 20MB SRAM bonded *on top* of the chip of CPUs.
- MIT and Berkeley: RAW and SCALE tiled processors; RAMP FPGA-based emulator for a range of new architectures.
- Stop press at Hotchips 20/8/2007 Tilera announced: chip with 64 RISC-like cores running at up to 1.0GHz plus switches linking cores in an 8x8 grid ("to 32 terabits per second of data bandwidth across the whole chip").

Plan of the talk

- Changes in Technology
- Programming Implications (for Programmers and for Languages)
- Opportunities for Static Analysis/Type Systems

Machine-oriented languages

These are languages in which machine operations all have high-level language correspondents, and high-level language constructs take a cycle or two.

- C for x86, MIPS, SPARC in 1985
- Occam for transputers (1980's computer on a chip)
- More?

Of course, we also need higher-level languages, but thinking about what is a good machine-level language helps clarify thoughts.

Machine-oriented languages can be efficiently compiled without pervasive whole-program optimisations.

Pointers considered harmful

OK, I really only mean *unrestricted* pointers, the sort you see in C or Java or ML. Let's explore:

- Pointer copy is more expensive that integer copy (even if both are 32 or 64 bits). Why?
- Because pointer copy creates an alias.
- Aliases stop data being moved between processors.
- Alias analysis (at least whole program) is undecidable in theory and intractable in practice.

So, want languages to support non-uniform memory—indeed we wish to de-emphasise shared RAM (serialised access).

OO-Languages (C++ and Java) considered harmful

History: C and similar languages were fine for mid-1980's. Programs got bigger—needed better languages: object-orientation.

But OO programming co-grew with the spread of the ever-faster single processor (no pressure to evolve).

Current OO languages have real problems with non-uniform memory. Let's look at a simple C++ declaration (chosen on day zero).

```
class Q { ... };
void f(Q x) { ... };     /* by value */
void g(Q *p) { ... };    /* by reference (sort of) */
void h(Q &p) { ... };    /* by reference (sort of) */
```

OO-Languages considered harmful (2)

What's wrong with this? The very first thing we do in C++ is to freeze in constraints about whether a function shares a data address space with its caller!

- Call-by-reference means caller and callee share data addresses
- Call-by-value means value can be transmitted to another processor (and potentially [Java Futures] code after the call executed in parallel).

This is early binding of physical distribution. [BAD!]

OO-Languages considered harmful (3)

Surely late binding of physical distribution is what we want? How about call-by-either-when-equivalent

```
void f(Q @p) { ... };   /* either CBV or CBR? */
```

meaning "use either CBV or CBR but reject the body of `f` if it does anything which can tell the difference"?

Functions *needing* CBR or CBV can have it, but CBEWE allows late binding of physical distribution.

OO-Languages considered harmful (4)

Why can't we do this automatically? Alias analysis?

- Whole program alias analysis is too imprecise/too discontinuous
- Suggestion is more like a type system – gives anchors to local analysis.

Ah, but Java doesn't have all this C++ nonsense?

- Java just fixes on CBR – inhibiting distribution

What's wrong with RMI (remote method invocation)?

- user still has very different syntax for CBV/CBR (= early binding)
- late change is still too hard semantically.

Taming pointers

"Structured programming for pointers".

Possibilities:

- '`Q @p`' notation above is essentially linear or quasi-linear types.
- Ownership types from OO community
- Extend pointers to know which memory region they apply to:
  `extern void f(int addressspace(A) *p);`
  These are essentially 'regions' (Talpin, Tofte).

Lightweight enough for ordinary programmers?

NEW RESEARCH NEEDED!

Quasi-linear types

Linear types are too much of a pain for many programmers (all data threading is explicit).

PacLang looked at providing quasi-linear types for general programmer use:

```
extern void g(packet p);          /* consumes p */
extern bool h(!packet p);         /* 'borrows' p */
packet f(packet p)
{ if h(p) return p;
  /* needs    "else kill(p)"
        or    "else g(p)"              here */
  return new(packet)
}
```

Quasi-linear types (2)

Microsoft's Singularity OS project further develops this idea to a message-passing operating system—linear data buffers (allocated in `ExHeap`) can be transferred from process to process.

What does linearity buy?

Implementation either by copy or by reference – perfect for multi-core with non-uniform memory.

Locks considered harmful

Locking mechanisms represent shared memory. This needs to be discouraged along with programming mechanisms which stress it without user awareness. E.g. `synchronized` objects in Java.

Reminder: Needham-Lauer "Duality of operating structures" showed shared memory and message-passing are duals.

Encourage message-passing programming models. (Actor model has disjoint memory spaces per process.)

Interesting observation: message passing and shared memory are directly equivalent with linear uses of buffers.

Locks considered harmful (2)

More lightweight alternatives (but still need shared memory)

- lock-free data structures?
- software transactional memory?
- speculative memory?

NEW RESEARCH AREAS: (i) refactoring between locks and these (e.g. Ennals' 'LockBend'); (ii) making any of these work with non-standard memory models.

What if memory is distributed across an on-chip network?

Locks considered harmful (3)

Separation logic as a basis for reasoning?

Coarser-grain notion of ‘separation’ (e.g. between memory spaces)?

I use ML or Haskell, aren't I safe?

Well, in some sense, because the compiler can choose to copy or alias data in pure functional languages. But

- ML has references (can't be copied)
- How does laziness distribute? Copy = rework!
- Even pure data: when do we distribute and when not?

Wall's work

Wall had an interesting paper "Limits of Instruction Level Parallelism" in 1991/1993.

- Compiled SPEC test suite
- Looked at instruction traces and assumed various models including hardware with perfect oracle (e.g. for branch prediction).
- Oriented to classical CPU, instructions executed more-or-less in order.
- How about re-doing this work using additional cores for speculation? (see next).

Wall's work (2)

E.g. compiling

```
if e0 then e1 else e2
```

into

```
par {x = e0 | x1 = e1 | x2 = e2 };
if x0 then x1 else x2
```

What are the lower bound limits on existing computations then?

RESEARCH: Are there then 'obvious' (to a compiler) spurious sequentialities in SPEC? Can analysis tell the user what inhibits parallelisation?

What about the heat?

- Compiling to optimise energy. Philips Digital Audio Cassette.
- As circuits get smaller, voltage scaling means higher leakage current. Must turn parts of chip on and off.
- Even if we're doing sequential computation, might wish to migrate the computation (heat source).

NEW RESEARCH: not totally new areas, but new models/analyses needed.

What's a bad interface?


```
module two_stage (in, out, clock);  <<=======
    input [7:0] in;
    output [7:0] out;
    input clock;

    reg [7:0] state1;
    reg [7:0] state2;
    assign out = state2;              // or out = state1
    always @(posedge clock)
          begin
             state1 <= in;
             state2 <= state1;
          end
endmodule
```

What's a bad interface (2)?

Good interface languages:

- are easy for ordinary programmers to understand
- fault programs after partial edits are made
- provide anchors for static analysis

E.g. C's function types (ISO not K&R).

NEW-ish RESEARCH AREA: interfaces for multi-core?
For memory system assumptions?

Can we characterise program forms

Berkeley dwarfs – archetypical large programs.

Do different dwarfs need different optimisation analyses/anchors?

Plan of the talk

UNIVERSITY OF
CAMBRIDGE

- Changes in Technology
- Programming Implications (for Programmers and for Languages)
- Opportunities for Static Analysis/Type Systems

Research I'd like to see done

- programming language designs to better reflect new hardware (e.g. aliasing controls, physical distribution, actors)
- more programming support/analysis for (un)reliability
- fundamental limits to algorithms (extend Wall's work), fractal analysis.
- innovative hardware ways to use all these extra transistors

SAS Heresy

- Why is this talk about language design not static analysis

Because I no longer believe in whole program analysis;

- Why?

The discontinuity effect: big program program P might have property Q but a small change $P + \Delta P$ might no longer have Q. What if Q is needed to parallelise P?

Local inference is good (and generally more continuous), but needs user-specified anchor points – interface specifications.

Conclusions

- "what is a computer" is now changing quite quickly;
- this gives opportunities for new research areas.