Opportunity Knocks: Disruption in Computer Systems

Michael O'Boyle

Institute of Computer Systems Architecture

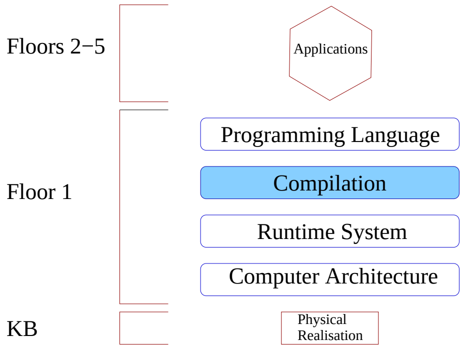
School of Informatics
University of Edinburgh
UK

April, 2010

Talk Structure

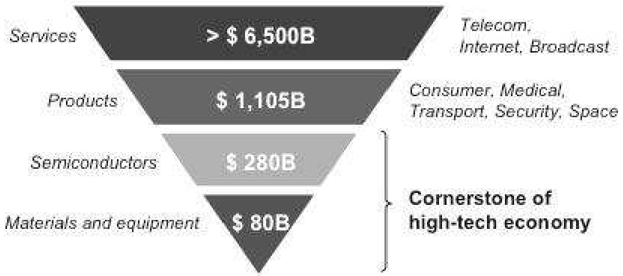
- Computing Systems Architecture
  - What is it?
  - Why is it important
- Changing Landscape
  - Past
  - Present
  - Future
- Challenges and Innovation areas
  - Open area
  - Work at Edinburgh
- Reflection on Hamming

Computing Systems Architecture: What is it?



- Mapping applications to hardware
- Compiler: 'C' $\mapsto$ x86

Computing Systems Architecture: Is it important?



Services
> $ 6,500B
Telecom, Internet, Broadcast
Products
$ 1,105B
Consumer, Medical, Transport, Security, Space
Semiconductors
$ 280B
Materials and equipment
$ 80B
Cornerstone of high-tech economy

Semiconductors underpin over 16% of the global economy

- Economically Significant hence high EPSRC/EC ICT budget
- Disruption in foundations is a cause for concern

Changing Landscape Technology Past

- Moore's Law
  - An observation in 1965
  - Growth in transistors
- Still holds for 40+ years

Moore's Law graph, 1965

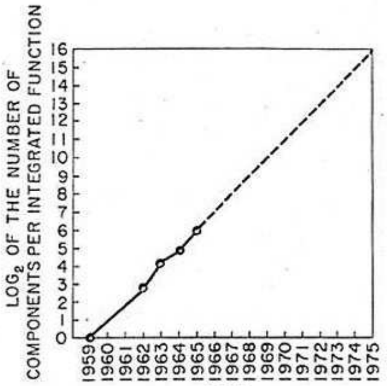
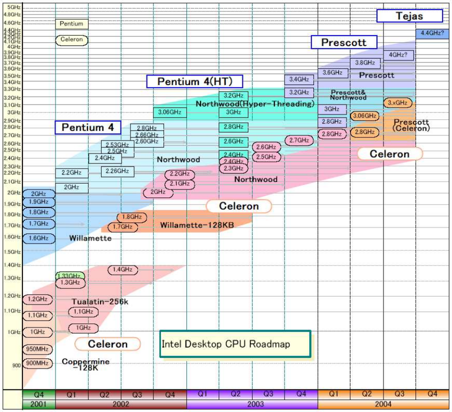


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

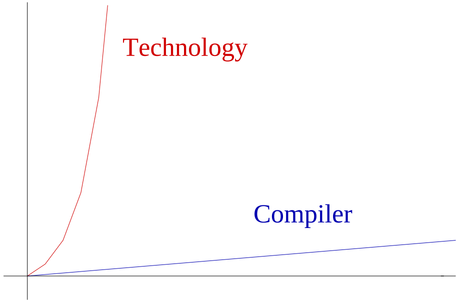
Architecture Past

- Scaling of transistors
- Translated into clock speed
- Engine for new products
- Onward and upward!
- (Hard to find old graphs!)



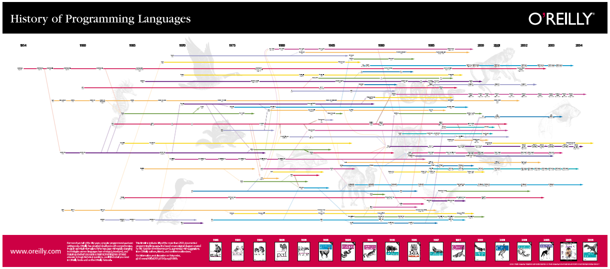
Copyright (c) 2003 Hiroshige Goto All rights reserved.

(Depressing) Compiler Past: Probesting's Law



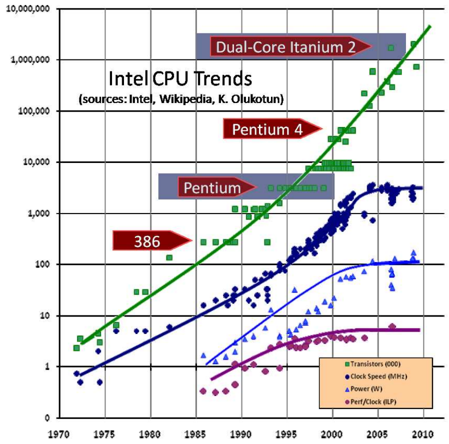
- Compiler double computing power every 18 Years!
- Only marginal contributions.
- Should focus on programmer productivity instead!

Programming Language Past



- Performance delivered by technology
  - Languages built for comfort not speed
- Rise of scripting languages
  - Just too late compilation
  - Focus on reducing programming effort

Changing Landscape Present: Technology Disruption



- Power stops clock scaling
- Parallelism only way to use the transistors
- Not the way we thought parallelism would become mainstream

Programming Panic

"The industry is in a little bit of a panic about how to program multicore processors, especially heterogeneous ones," said Moore.

“To make effective use of multicore hardware today you need a PhD in computer science. That can’t continue if we want to enable heterogeneous CPUs,” he said.

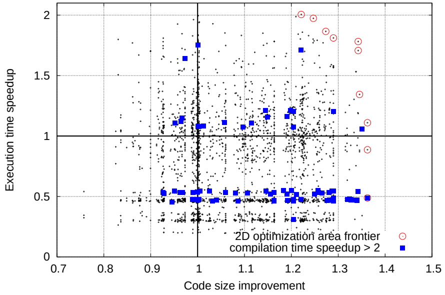
Chuck Moore, an AMD senior fellow

Changing Landscape Present: Programming Languages

Outfitting C++ for Multi-core Processor Parallelism
Intel Threading Building Blocks
O'REILLY
Beginning Scala
Stre
UPC
8th Annual Workshop on Charm++ and its Applications

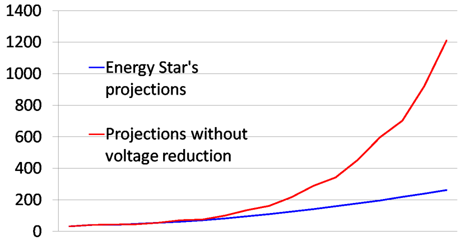
- Wide range of programming paradigms
- No clear winners

Compilers +Architectures: Overwhelming Complexity



- Greater computing speeds and tools allow greater exploration
- Overwhelming complexity

Technology Future: Energy wall ahead



- Billions of Kilowatt hour/year y-axis 0-1400
  - Data centres till 2017 (Source: Babak Falsafi)
- Technology trend: transistors for free, energy is the cost
  - You can have your cake but not eat it!
- Heterogeneity and parallelism twin solutions

Programming Language Future $64k question

C++1xx

SPIRAL
Software/Hardware Generation for DSP Algorithms

- Good vs Bad vs Something else
- The world is waiting for the next programming language
- Convergence of HPC vs General vs Embedded
  - Redraw the programming language boundaries

Energy, Customisation, Parallelisation, Complexity and Change

- These are the critical issues in Systems Research
- Energy efficiency: overarching driver
- Parallelism + Customisation
  - Each new core is slower
  - GPUs just the start
- Design space of new systems massive
  - Deciding what to do with 10^{10+} transistors
  - Different for different domains
- Complexity and Change
  - Faster generations
- Examine issues in reverse order
  - Start with Compilation
  - A new methodology

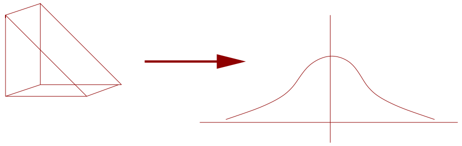
Why do compilers fail to find the best optimisation?

- Fundamental reason for failure is complexity, undecidability and change
  - Minimise execution time over space all equivalent programs
  - Solve halting problem over an unbounded space!
- The processor architecture behaviour is so complex that it is almost impossible to determine the best code sequence *a priori*.
  - Although individual components are simple, together impossible to derive realistic model
  - O-O execution and cache have non-deterministic behaviour!
- Q: If your cache changed from LRU to random replacement how would you rewrite your code.

Complexity and Change: Down with Logic Up with Data

- Hardware moves faster than applications
  - System software always playing catch-up
- Complexity of design issues and design space
  - Beyond our abilities
  - Hand-crafted approaches no longer viable
- Industrial revolution of design methodology

Complexity and Change: Down with Logic Up with Data



$$\wedge, \vee, \forall, \exists \mapsto \sigma, \mu, E[x]$$

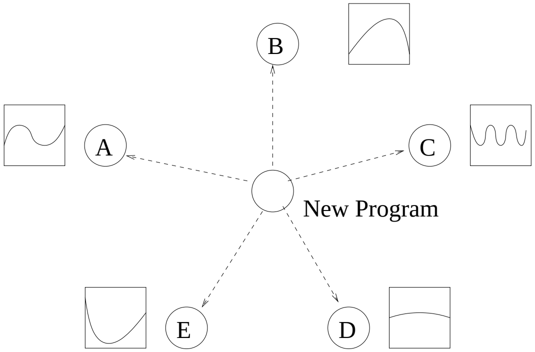
Current compiler approaches:

- Model optimisation as several smaller problems that are at least NP hard and then develop approximate solutions.
- Evaluation based on how good it solves a model - not based on evidence!
- Model always wrong and out of date - throw away the model

Start of fundamental change in how we approach design

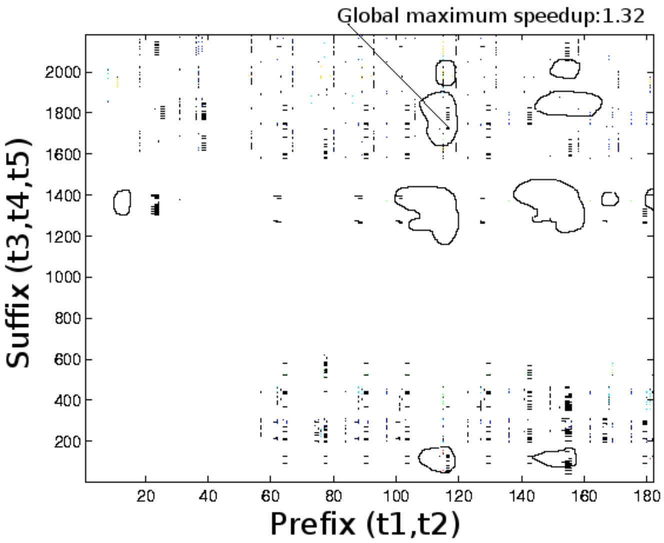
- Will transform how we do our research
- Based on evidence not weak comparison. Automation.

Learning optimisations using nearest neighbour classification



- Features of program define neighbour
- Use neighbours distribution to select best opt
- Best opt depends on evidence.
  - Changes with time
  - Automatically updated

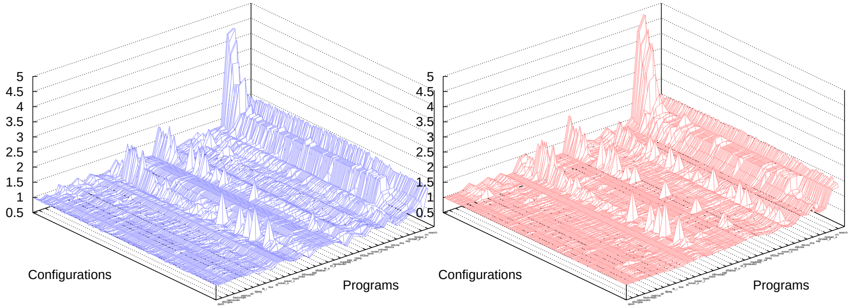
Finding the needle : Adpcm on TI C6713



Change and Complexity: Challenges

- Statistical and Evidence based techniques rapidly taking over
- Guaranteeing authentic data -
  - Data providence in systems
- Modelling Challenges
  - What are the right features?
  - Managing unbounded structures
  - What are the best models?
- Evidence of Generalisation
  - Did you just get lucky?
  - Task transference
- Finding the right question to ask
- Going beyond correlation to discovering structure
  - Explaining why something works/fails
  - Driving innovation

Change and Complexity:Our approach



- Automatically building a compiler across configurations
- Deliver a compiler that achieves 67% of maximnum
  - Across all progs/archs With just one profile run
- Automating compiler construction - within a small space

Parallelism: A massive challenge

- How to program a parallel machine has been around a long while
  - Ex-WCS occam programmer 1986
- In past - just wait 18 months
  - True for Supercomputing too. Write MPI once - wait
  - Now - will get SLOWER each generation!
- Debate has often broken down into who has control
  - Going beyond expert programmer vs auto-paralleliser
- Realisation that
  - Programmers cost !
  - Static analysis of pointer rich C code is doomed
  - Parallelism going mainstream
- Changes everything

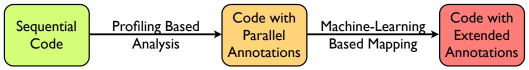
Parallelisation Challenge : Programming Languages

- Major hurdle to innovation is APIs: C and x86
- Crying out for the new Java for parallelism
- Languages are critical but hardest to influence
- Fashion not science
  - Sisal in the 90s
- Convergence of Embedded, HPC and general-purpose
  - Redraw Architecture and Language map
  - No longer Fortran+MPI (HPC) vs C (embedded) vs Java (general)
  - Allows creative thinking for new domains
- A new language should describe parallelism not mapping
  - Work with emerging new systems software approaches
- Domain specific

Parallelisation Challenges Compiler Perspective

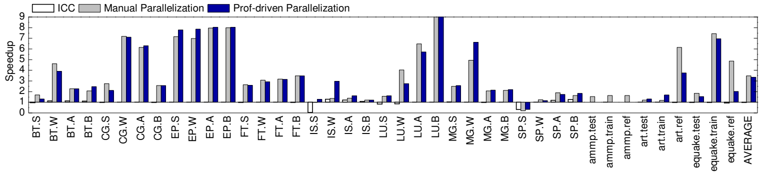
- A hot area for years to come!
- New transformations for parallelism
  - ML only good at predicting within a space - we must define space
- Dynamic analysis in its infancy
  - Need to make cheaper
  - Combine with static analysis
  - Use in on-line context - drive on-line parallelisation
- Dynamic parallelism
  - Pre-canned mappings from off-line knowledge
- Data dependent parallelisation
  - Combine speculation support in hardware and runtime systems

Parallelism: Dynamic Analysis +ML


Sequential Code
Profiling Based Analysis
Code with Parallel Annotations
Machine-Learning Based Mapping
Code with Extended Annotations

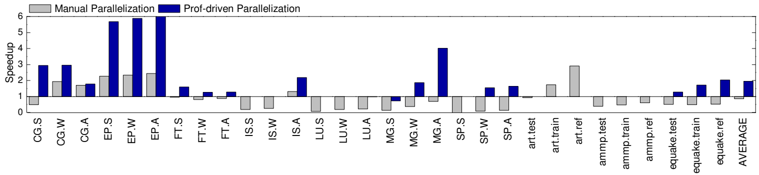
- Split into 2 fundamental problems
  - Determining/Discovering parallelism
  - Mapping that parallelism to an arbitrary machine
- Use profile information to discover parallelism
  - Inherently unsafe! However in our (limited) experience it works
  - Associate memory usages with source code
  - Currently limited to loop parallelism
- Select and predict best loops to parallelise
  - Based on off-line learnt model
  - Focus on thread number and schedule

Performance vs ICC and OpenMP on Intel Xeon 2x4 cores



- Took sequential NAS and Spec 'C' code - upto 7k lines long
  - Auto-parallelised and generated OpenMP code
  - Compared against ICI on Intel 8 core and hand-parallelised NAS-PB and Spec OMP2001 also in OpenMP
- Achieve 96% of hand parallelised version
  - Even when Spec OMP2001 have been *retuned*
  - Spec OMP sequential performance double standard SPEC

Performance vs OpenMP on Cell

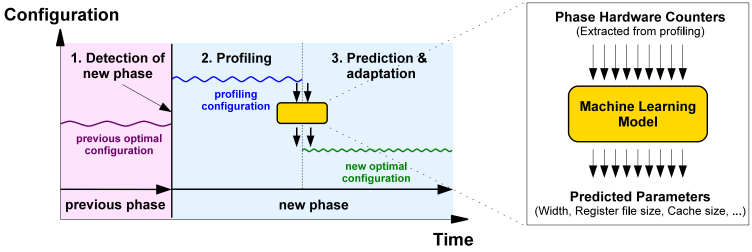


- Similar comparison
  - No auto-paralleliser available on Cell
  - Not all OpenMP programs passed through xlc compiler
- On average Hand-parallelised code gives a slow down
  - We achieve a speedup of 2
- Shows that hand parallelised code ill-fitted for Cell
  - Room for improvement for Cell OpenMP library
- ML works around this

Energy and Customisation: Challenges

- The gap between a general-purpose processor and an ASIC is 1000x in terms of energy
  - Specialised hardware such as GPUs driven by this
- One direction, specialise hardware at design/compiler/runtime
- Compiler challenge
  - Pick the best ISA: PASTA project
  - Select the right hardware components for the job
  - Transform the code to fit
- Prior knowledge of compiled code
  - Allows ahead of time powerup /down
  - Just turn on what you'll use
- Alternatively focus on critical path
  - and slow down the rest

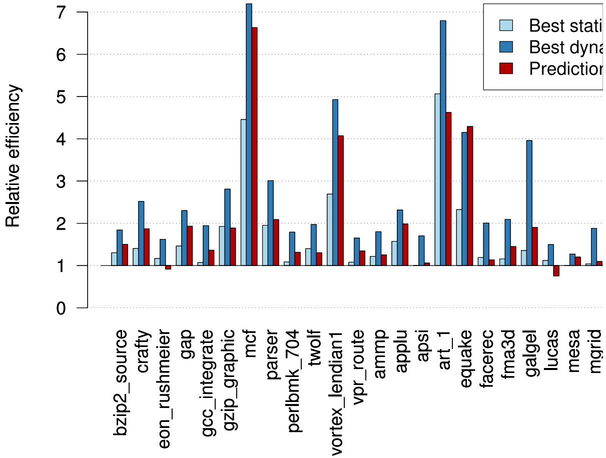
Challenges and Innovation areas: Hardware Specialisation


Configuration
1. Detection of new phase
2. Profiling
3. Prediction & adaptation
profiling configuration
previous optimal configuration
new optimal configuration
previous phase
new phase
Time
Phase Hardware Counters
(Extracted from profiling)
Machine Learning Model
Predicted Parameters
(Width, Register file size, Cache size, ...)

Gather features on hardware behaviour via hardware counters

- Send to a trained hardware model
- Predict and change hardware configuration on the fly

Challenges and Innovation areas: Hardware Specialisation



- Achieves 70% of the max energy/perfom available in space

Challenges and Innovation areas: Beyond Silicon

- All this work assumes a (new) business as usual
  - What happens when technology scaling stops?
- Quantum computing
  - Looks unlikely to be general purpose
- Carbon nanotubes
  - Don't know
- Biologically Inspired
  - Vague at best
- Designing and programming such systems is a major challenge

Challenges and Innovation areas: Summary

- Compilers
  - Golden times ahead
  - Silver bullet ?
- Language
  - Most important: High Risk though!
- Hardware
  - Customisation and Replication
- Runtime
  - Symbiosis with compiler blurring with JIT
- Beyond Silicon
  - Wide open

Reflection on Hamming: Effort

"I spent a good deal more of my time for some years trying to work a bit harder and I found, in fact, I could get more work done. I don't like to say it in front of my wife, but I did sort of neglect her sometimes; I needed to study. You have to neglect things if you intend to get what you want done. There's no question about this."

BUT

"drive, misapplied, doesn't get you anywhere"

- Be smart with effort
- More to life than work!

Reflection on Hamming: Selling

"I have now come down to a topic which is very distasteful; it is not sufficient to do a job, you have to sell it. 'Selling' to a scientist is an awkward thing to do. It's very ugly; you shouldn't have to do it. The world is supposed to be waiting, and when you do something great, they should rush out and welcome it. But the fact is everyone is busy with their own work. You must present it so well that they will set aside what they are doing, look at what you've done, read it, and come back and say, "Yes, that was good."

- As true as it ever was
- Means to an end. Not an end in itself

Reflection on Hamming: Vision thing

“ When you are famous it is hard to work on small problems. This is what did Shannon in. After information theory, what do you do for an encore? The great scientists often make this error. They fail to continue to plant the little acorns from which the mighty oak trees grow. They try to get the big thing right off. And that isn't the way things go. “

- Important is not the same as grandiose!
- Beware of the vision thing as you age!

Opportunity knocks!