



Designing Tomorrow's Microprocessors

Antonio González
Director, Intel Barcelona Research Center
Professor, Computer Architecture Department, UPC

Lecturers:
Josep M Codina, Ayose Falcón, Antonio González, Enric Herrero, Marc Lupon, Pedro Marcuello,
Raúl Martínez, Tanausu Ramírez, Kyriakos Stavrou, Ferad Zyulkyarov

Aula Empresa, Facultat d'Informàtica de Barcelona, January 29-31, 2013

© Intel Corporation, 2013

Overview of Today's Microprocessors and Future Trends

- Computing Evolution
- Technology Scaling
- Microprocessor Families
- Future Challenges
- Some Research Projects at Intel Labs

3

Designing Tomorrow's Microprocessors

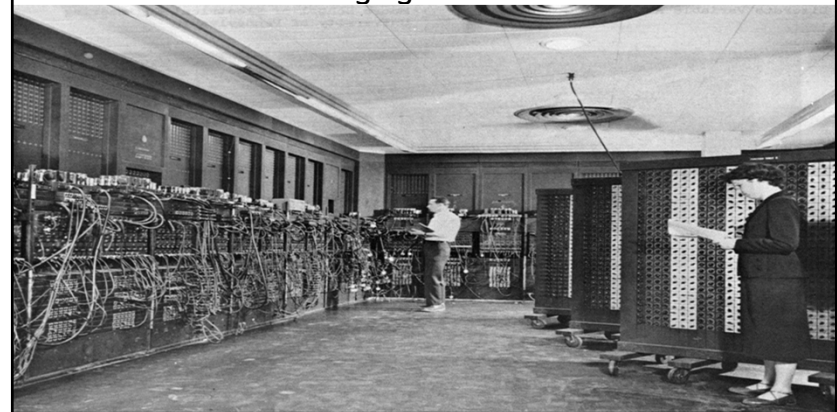
Agenda

- Overview of Today's Microprocessors and Future Trends
Antonio González
- Microarchitecture of Current Microprocessors
Marc Lupon
- Design Cycle for Microprocessors
Raúl Martínez
- Methodology for Research in Microprocessors
Pedro Marcuello
- Profiling and Performance Evaluation
Ferad Zyulkyarov
- Multi-core Processor Architectures
Ayose Falcón
- Parallel Programming
Josep M Codina
- Reliability
Enric Herrero and Tanausu Ramírez
- Hw/Sw Co-designed Microprocessors
Kyriakos Stavrou

2

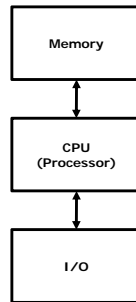
Designing Tomorrow's Microprocessors

Changing the World



Initial Developments

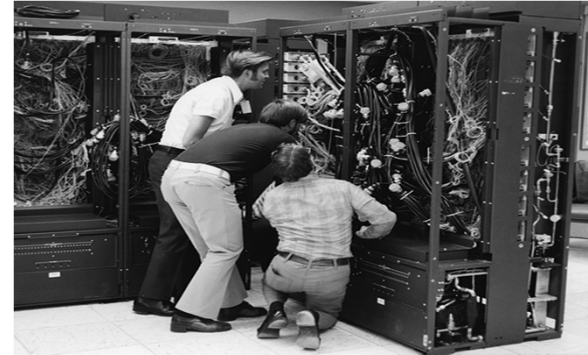
- 1946: ENIAC by J.P. Eckert and J. Mauchly
- 1945: Stored program by J.V. Neuman
- 1949: EDSAC by M. Wilkes
- 1952: UNIVAC I and IBM 701



5

Designing Tomorrow's Microprocessors

Computing Evolution



The beginning
(ENIAC I - 1946)

- Thousands Ops/sec
- Tens of data

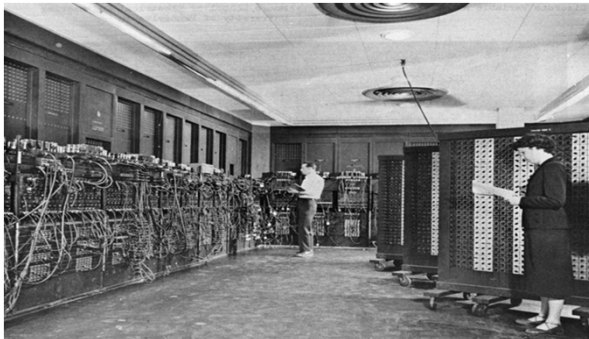
Mainframes and
Supercomputers
(CDC 7600 - 1969)

- Millions Ops/sec
- Millions of data

7

Designing Tomorrow's Microprocessors

Computing Evolution



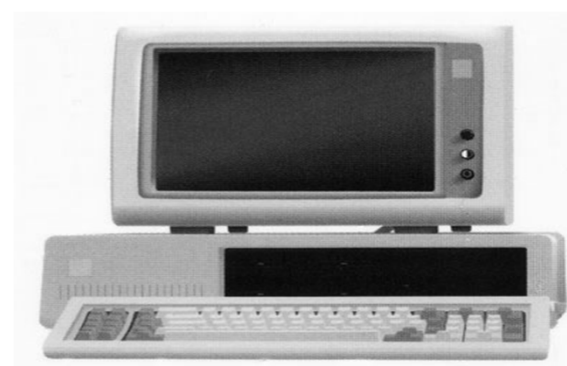
The beginning
(ENIAC I - 1946)

- Thousands Ops/sec
- Tens of data

6

Designing Tomorrow's Microprocessors

Computing Evolution



The beginning
(ENIAC I - 1946)

- Thousands Ops/sec
- Tens of data

Mainframes and
Supercomputers
(CDC 7600 - 1969)

- Millions Ops/sec
- Millions of data

Mega-scale
Computing
(IBM PC - 1980)

- Millions Ops/sec
- Millions of data

8

Designing Tomorrow's Microprocessors

Computing Evolution



The beginning (ENIAC I - 1946)

- Thousands Ops/sec
- Tens of data

Mainframes and Supercomputers (CDC 7600 - 1969)

- Millions Ops/sec
- Millions of data

Mega-scale Computing (IBM PC - 1980)

- Millions Ops/sec
- Millions of data

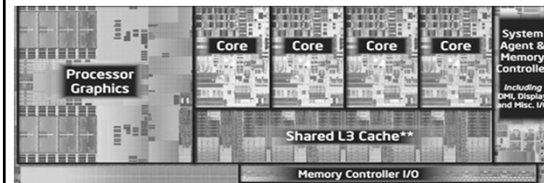
Giga-scale Computing (Mobile - 2000's)

- Billions of Ops/sec
- Billions of data

9

Designing Tomorrow's Microprocessors

At the Heart of This Evolution: The Microprocessor



The Microprocessor (Intel 4004 - 1971)

- 2300 transistors
- Tens of thousand ops/sec

High Performance (Intel Core i7 - 2012)

- Over 1 Billion transistors
- Hundreds of Billions ops/sec

11

Designing Tomorrow's Microprocessors

At the Heart of This Evolution: The Microprocessor



The Microprocessor (Intel 4004 - 1971)

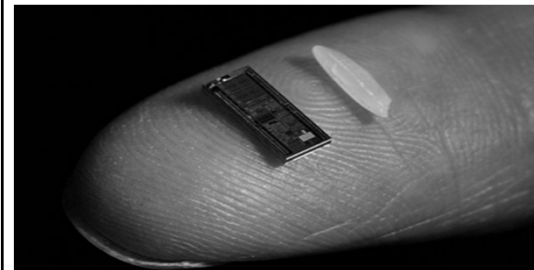
- 2300 transistors
- Tens of thousand ops/sec

www.MyNikko.com 2006

10

Designing Tomorrow's Microprocessors

At the Heart of This Evolution: The Microprocessor



The Microprocessor (Intel 4004 - 1971)

- 2300 transistors
- Tens of thousand ops/sec

High Performance (Intel Core i7)

- Around 1 Billion transistors
- Hundred of Billions ops/sec

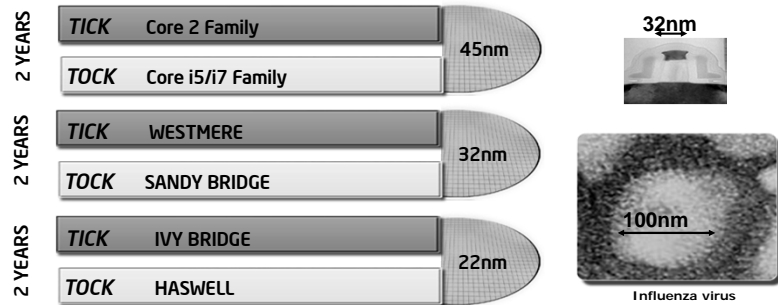
Ultra Low Power (Intel Atom)

- Around 50M transistors
- Billions of ops/sec

12

Designing Tomorrow's Microprocessors

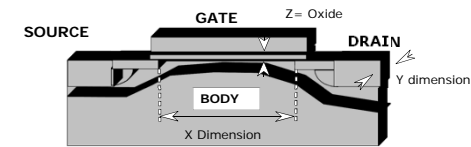
Intel's Tick Tock Model



13

Designing Tomorrow's Microprocessors

Technology Scaling Theory



X & Y Dimensions scale down by 30%	Doubles transistor density
Z-Oxide thickness scales down	Faster transistor, higher performance
Vcc & Vt scaling	Lower active power

Technology scaling is a great thing

15

Designing Tomorrow's Microprocessors

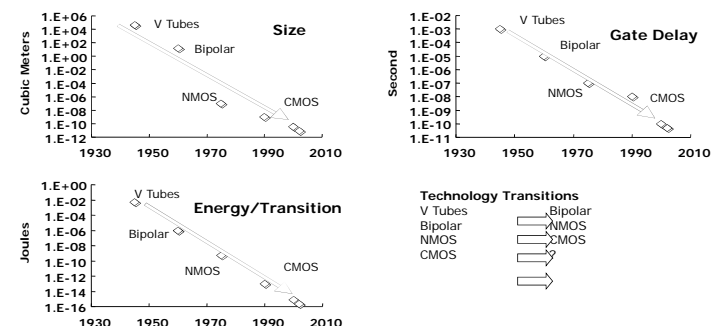
Gordon Moore's Law

- The number of transistor in a chip doubles every 2 years
 - Based on 4 points (year/transistor count) (1959, 1), (1962, 8), (1964, 32), (1965, 64)
 - Established on April 19, 1965 in the Electronics Magazine, predicting 65000 transistors in 1975
- Revised in 1975 in IEEE International Electron Device Meeting

14

Designing Tomorrow's Microprocessors

Technology Scaling Trends



Scaling Will Continue

16

Designing Tomorrow's Microprocessors

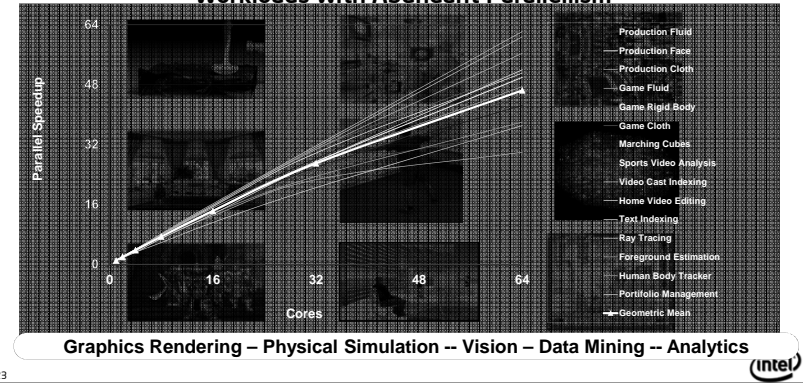
The Present: A Compute Continuum



21



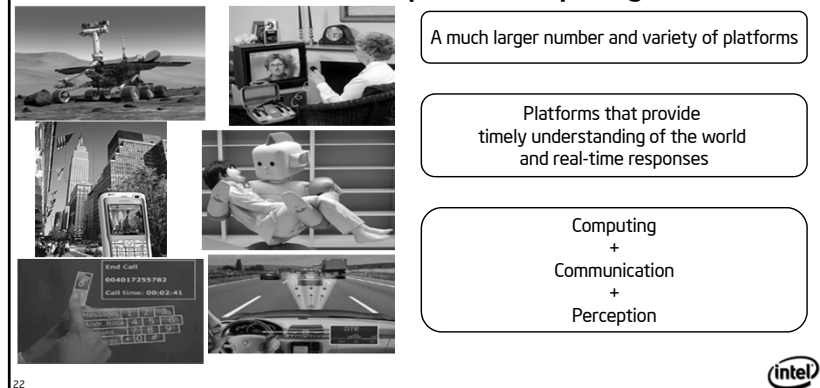
Good News Workloads with Abundant Parallelism



23



The Future: Ubiquitous Computing



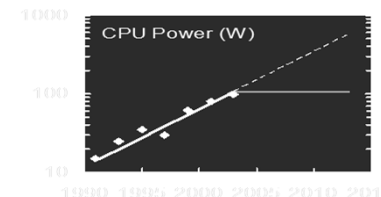
22



Challenge #1:

Power budget to remain flat in every segment

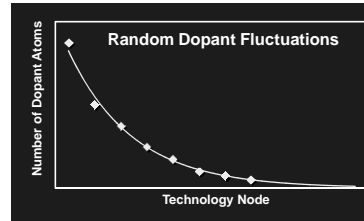
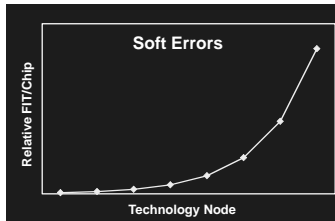
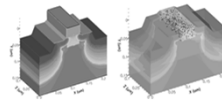
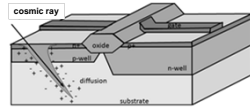
- Autonomy in a wireless world
- Desire for very small form factors
- Cost of cooling solution
- Noise of cooling solution



24



Challenge #2: Growing vulnerability and variability



25

Intel Labs Barcelona Mission

Develop novel processor microarchitectures that provide dramatic improvements in performance with no increase in power and without compromising reliability for the increasing diversity of computing systems



27

Intel Labs Mission and Structure

Delivering Breakthrough Technologies to Fuel Intel's Growth

Collaborative Research

INDUSTRY



GOVERNMENT



UNIVERSITIES



Visioning, Inventing, & Validating

ENVISIONING NEW USER EXPERIENCES



INVENTING AND VALIDATING NEW TECHNOLOGIES



Pathfinding & Technology Transfer

NEW PRODUCT INNOVATION



NEW BUSINESS VENTURING



SOFTWARE OPEN SOURCING



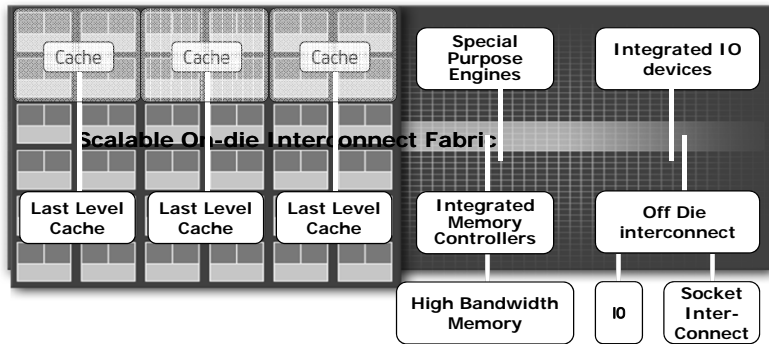
26

MAIN RESEARCH AVENUES



28

Research Focus: Future Microprocessors

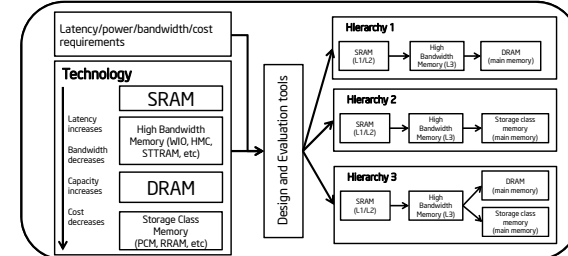


29



Future Memory

- Performance evaluation methodologies
- Memory architectures based on emerging technologies
- Multi-level memory architectures
- Novel usage models of persistent memory

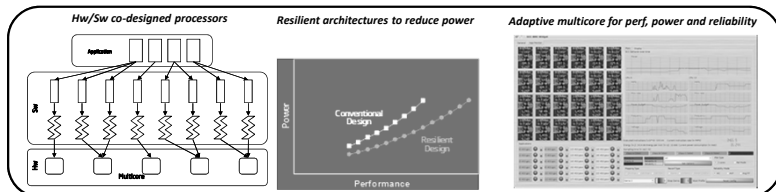


31



Future Cores

- Energy-efficient performance
- Reliable and adaptive architectures
- Heterogeneous architectures
- Architectures for emerging technologies

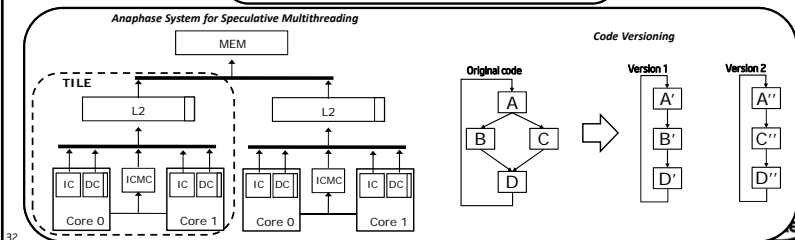


30



Future Programming

- Languages & programming abstractions
 - Transactional memory
 - Lightweight threading
- Compilers
 - Dynamic compilation
 - Parallelization



32



Microarchitecture of Current Microprocessors

Marc Lupon
marc.lupon@intel.com

Computing Domains

Personal Computer (PC)

- Affordable
- Personal (one/few users)
- In-place (office/work)
- General-purpose applications
- **Example:** Desktops



High-Performance Computing (HPC)

- Complex
- Really expensive
- Shared (owned by company or institution)
- Special place (data center)
- Scientific, data-intensive applications
- **Example:** Supercomputer



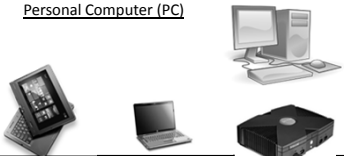
Embedded devices

- Cheap
- Simple
- Personal (one/few users)
- Mobile (most of them)
- Unique-purpose applications
- **Example:** Discman



Computing Domains

Personal Computer (PC)



High-Performance Computing (HPC)

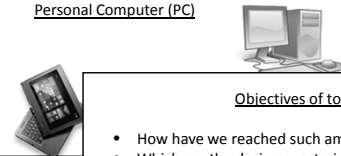


Embedded devices



Computing Domains

Personal Computer (PC)



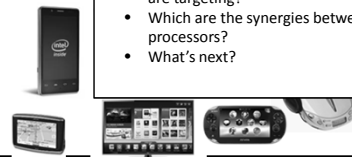
High-Performance Computing (HPC)



Objectives of today's lecture:

- How have we reached such amount of devices?
- Which are the design constraints that each device must satisfy to be successful?
- How are the processors built according to the domain they are targeting?
- Which are the synergies between different domain processors?
- What's next?

Embedded devices



Outline

- Evolution of computing market segments
 - **The old days:** Performance at whatever cost
 - **Yesterday:** Climbing walls
 - **Today:** Let's move!
- Requirements and micro-architectural techniques for distinct market segments
 - Desktops and laptops (PC domain)
 - Embedded and Mobile (Embedded domain)
 - Servers and Supercomputers (HPC domain)
- Characteristics of products in the market
 - Processors for gaming consoles (XBOX vs PS3)
 - Processors for mobile devices ()
 - Processors for high-computing domains (Atom vs Xeon)
- What's next?
 - User experience and novel devices
 - Design trends for products to come

The old days (<2005)

Embedded devices

First goal: Real-time
Applications: Sequential
Performance: Medium
Power: Efficient
Cost: Cheap
Challenge: Be cheap
Example: TI TMS family

Personal Computer

First goal: Performance
Applications: Sequential
Performance: High
Power: Don't care
Cost: Medium
Challenge: ILP
Example: Pentium family

HPC World

First goal: Performance
Applications: Parallel
Performance: High
Power: Don't care
Cost: High
Challenge: Communication
Example: Cray family

Cheap and efficient!

- Microcontrollers
- Accelerators
- Small area and width
- Easy to validate
- Programmable devices
- Few instructions

Performance at whatever cost!

- Big cores running at high-frequency
- Superscalar, deeply pipelined processors
- Aggressive Out-of-order pipelines
- Huge, very accurate branch predictors
- Objective: Extract ILP (CISC-like Processors)
Extract DLP (Vector Processors)

Yesterday (<2010)

Embedded devices

First goal: Ubiquitous
Applications: Sequential
Performance: Medium
Power: Efficient
Cost: Cheap/Medium
Challenge: General PP
Example: ARM9 family

Personal Computer

First goal: Low-power
Applications: Parallel
Performance: Medium
Power: Efficient
Cost: Medium
Challenge: TLP
Example: Core family

HPC World

First goal: Fast response
Applications: Parallel
Performance: High
Power: Efficient
Cost: High
Challenge: I/O
Example: Itanium family

Simple and General

- General Purpose
- Small area and width
- More instructions
- Objective: Extract ILP
- RISC-like Processors
- In-order
- Simple designs
- SOC with accelerators

Climbing walls

- Techniques to climb the power and memory wall
 - SMT
 - Memory Hierarchy
- Big area, huge amount of transistors
- Chip multiprocessor
- Medium-frequency
- Power-efficiency

Keep running

- Huge cores running at high-frequency
- Wider architectures
 - IA64b
- VLIW
- Predication

Today (2010-2015)

Embedded devices

First goal: Performance
Applications: Parallel
Performance: High
Power: Efficient
Cost: Medium
Challenge: Perf/Power
Example: Tegra family

Personal Computer

First goal: Survival
Applications: Parallel
Performance: Variable
Power: Efficient
Cost: Medium
Challenge: Perf/Power
Example: HSW family

HPC World

First goal: Reduce cost
Applications: Parallel
Performance: High
Power: Efficient
Cost: Low
Challenge: Perf/Power
Example: Atom family

Performance

- General Purpose
- Multiprocessors
- Out-of-order
- Objective: Extract TLP
- Heterogeneity
- Minimal power
- Always connected

Let's move!

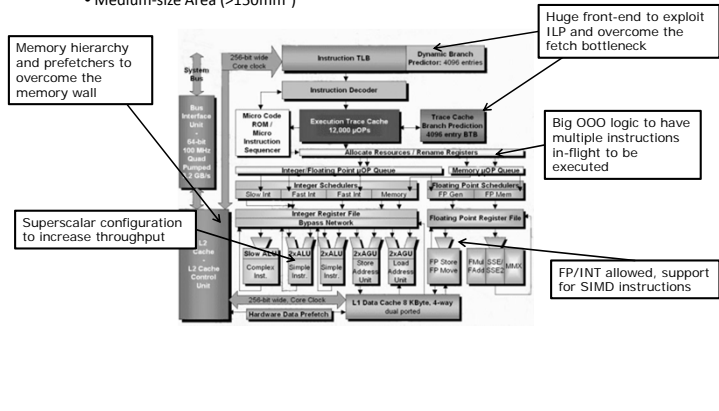
- Go to Low-power
- Win all domains
- Reconfigurable
- Adaptable
- Reliable
- Simplify Parallel programming

Go cheap!

- Reliable
- Responsive
- Redundancy
- Simple cores...
- ...but lot of cores
- Objective:
 - Increase Instr/Watt
 - Enable parallelism

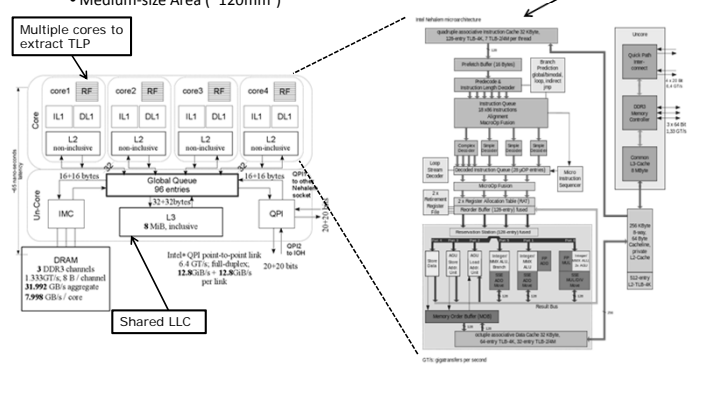
The Personal Computing Domain

- Requirements for the PC domain (<2005)
 - Performance at whatever cost! (from 1Ghz to 4Ghz)
 - High power budget (60W to 115W)
 - Medium-size Area (>150mm²)



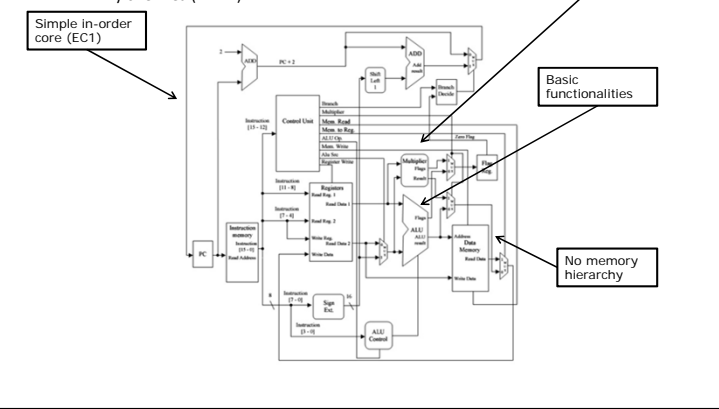
The Personal Computing Domain

- Requirements for the PC domain (Today)
 - Performance in any kind of application (sequential, parallel)
 - Fixed power budget (10W tablets, 20W-ultrabooks, 100W-desktops)
 - Medium-size Area (~120mm²)

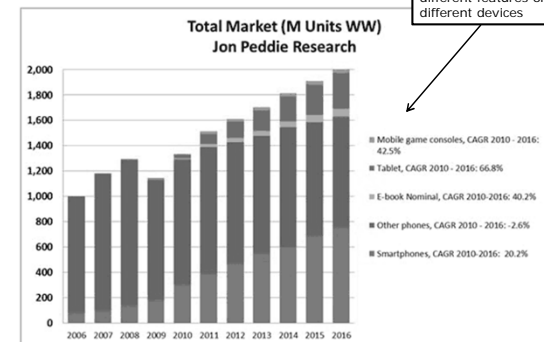


The Embedded Domain

- Requirements for the Embedded domain (<2005)
 - Performance in specific applications (no General purpose)
 - Small power budget (1W at most)
 - Tiny-size Area (1mm²)



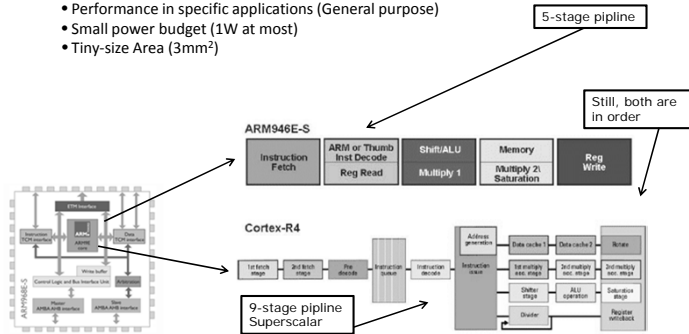
The Embedded Domain



Embedded devices become ubiquitous, require different features on different devices

The Embedded Domain

- Requirements for the Embedded domain (Yesterday)
 - Performance in specific applications (General purpose)
 - Small power budget (1W at most)
 - Tiny-size Area (3mm²)

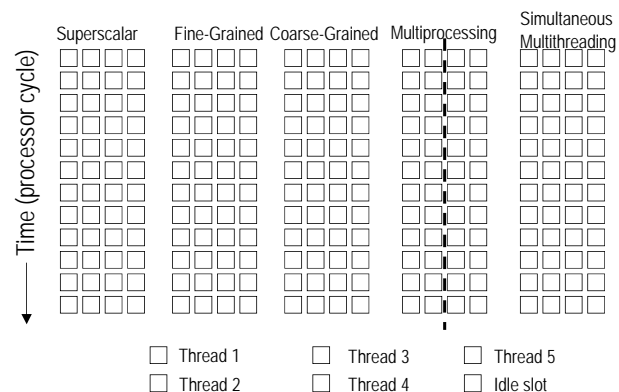


Low-Power HPC Domain

- Requirements for the PC domain (<2005)
 - Extract maximum parallelism (20 Threads)
 - High power budget (130W)
 - Big-size Area (500mm²)
- Techniques to implement a low-power HPC processor
 - **Voltage/frequency scaling**: reduce supply voltage and/or frequency when processor is idle
 - **Clock gating**: disable clocks to inactive components
 - Reduce power consumption of **memory components**
 - **Pipeline gating**: reduce mis-speculated instruction execution
 - **Pipeline balancing**: adjust effective pipeline ways for available IPC
 - **Efficient issue logic**: cluster structure, adjust effective issue queue size, no matching for ready entries, reducing tag matching entries

14

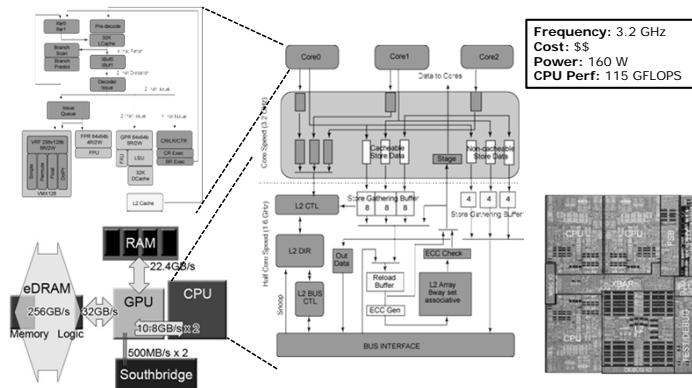
Low-Power HPC Domain



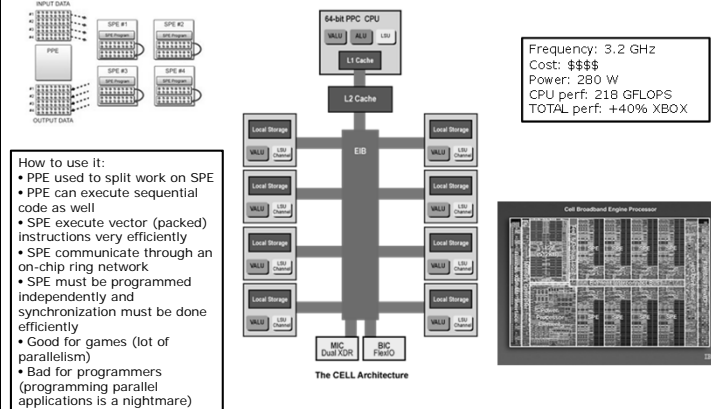
Outline

- Evolution of computing market segments
 - **The old days:** Performance at whatever cost
 - **Yesterday:** Climbing walls
 - **Today:** Let's move!
 -
- Requirements and micro-architectural techniques for distinct market segments
 - **Desktops and laptops (PC domain)**
 - **Embedded and Mobile (Embedded domain)**
 - **Servers and Supercomputers (HPC domain)**
- Characteristics of products in the market
 - Processors for gaming consoles (XBOX vs PS3)
 - Processor for graphics mobile devices (Tegra4)
- What's next?
 - User experience and novel devices
 - Design trends for products to come

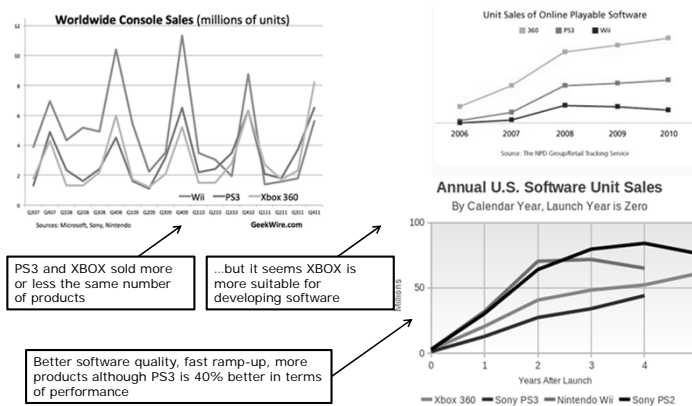
Gaming Processors (XBOX 360)



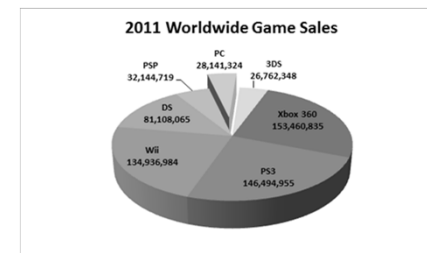
Gaming Processors (Cell/PS3)



Gaming Processors: HW/SW Sales



Gaming Processors: HW/SW Sales



Even with the most powerful hardware, the same volume of game consoles sold and the advantage of having a successful predecessor (PS2), XBOX 360 still sells more games.

Conclusion: Sometimes, productivity is more important than FLOPS. Better games can be developed in less time if the CPU is general purpose.

Mobile processor (Tegra3)

12 Graphic cores, able to execute CUDA

NVIDIA Tegra 3 GPU



Saver power-optimized core (ARM Cortex A9 with lower frequency, 500MHz)

Throughput and parallelism extracted through 4 general purpose cores (ARM Cortex A9, 1.4 GHz single core or 1.3 GHz quad-core)

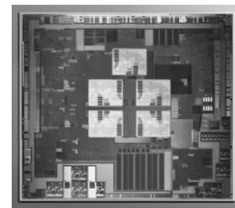
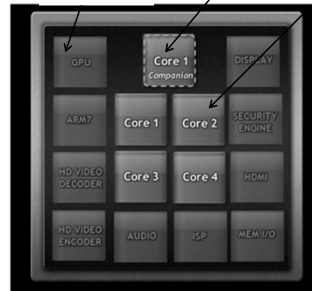
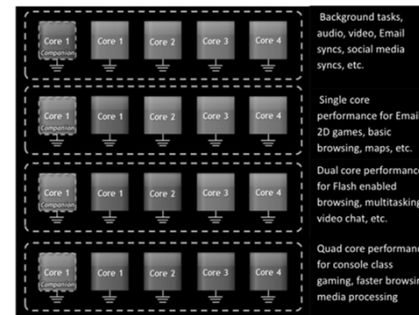


Figure 3 Low Power Companion CPU on Kai-EI

Mobile processor (Tegra3)



Mobile processor (Tegra4)

72 Graphic cores, able to execute CUDA, 520 MHz

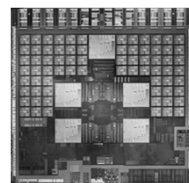
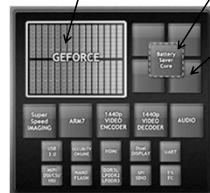
NVIDIA Tegra 4 GPU



Saver core for low power (ARM Cortex A15 with lower frequency, 700MHz)

Throughput and parallelism extracted through 4 general purpose cores (ARM Cortex A15, 1.9 GHz)

Soft programmable modem processor



nVidia Tegra 4 mobile processor

What's next?

- More devices to embrace more user experiences
 - Mobile → Small devices → Small area
 - Portable → Long battery life → Low power
 - Manageable → Human interaction → Low temperature
 - Multi-use → Different applications → General purpose
 - Connected → Fast access to cloud → System-on-chip
 - Interactive → Different uses → Heterogeneity
 - Powerful → new experiences → Performance!
 - Synchronized → all-in-all devices → Common architecture

• Solutions?

- Each device has a new core or
- A processor/chip to rule them all!



Conclusions

- The world moves fast, creating new devices (computing domains)
- Computing domains used to be independent...
- ...but now they face the same challenges
 - **Low power**
 - **Parallel computing**
 - **Cheap (cooling, area, validation)**
 - **General-purpose applications**
 - **Performance (when needed)**
- We have seen how different micro-architectures attempt to approach the same problems, with their strengths and weaknesses
- **Processors must help to increase productivity (profiler, debugger), not the other way around (complex ISA, explicit synchronization, etc)!**
- We don't know what is going to be in the future, but the door is open to any bright idea (and this is)

Design Cycle for Microprocessors

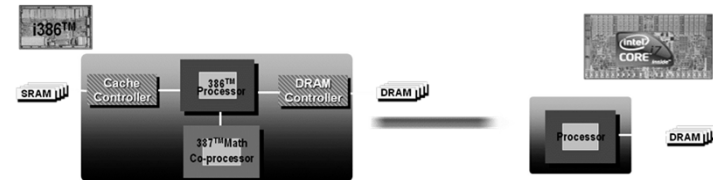
Raül Martínez

Intel Barcelona Research Center

Aula Empresa, Facultat d'Informàtica de Barcelona

© Intel Corporation, 2013

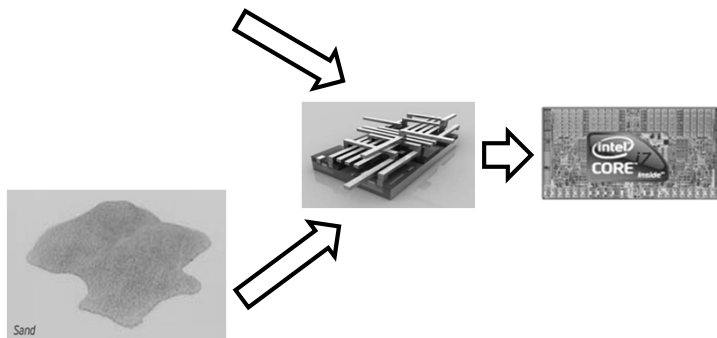
Introduction



2

Designing Tomorrow's Microprocessors

Introduction



3

Designing Tomorrow's Microprocessors

Agenda

- Introduction
- General Phases
- Design Steps
- Validation
- Design Types and Intel Tick-Tock model
- Conclusions

4

Designing Tomorrow's Microprocessors

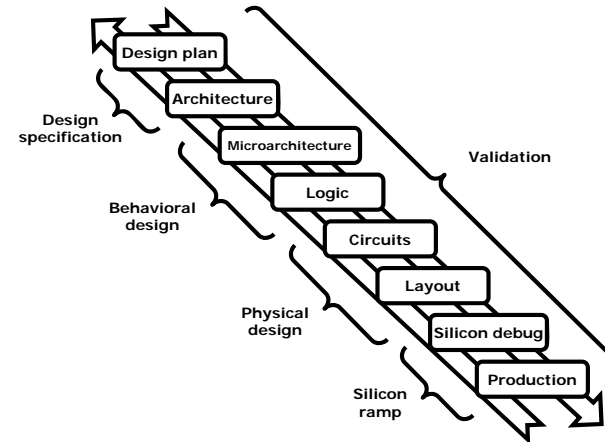
General Phases



5

Designing Tomorrow's Microprocessors

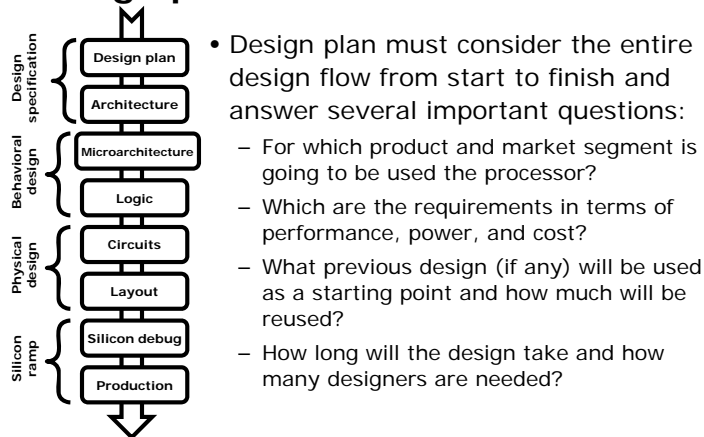
Design Steps



6

Designing Tomorrow's Microprocessors

Design plan



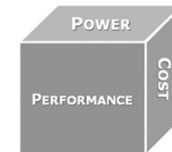
7

Designing Tomorrow's Microprocessors

Design plan

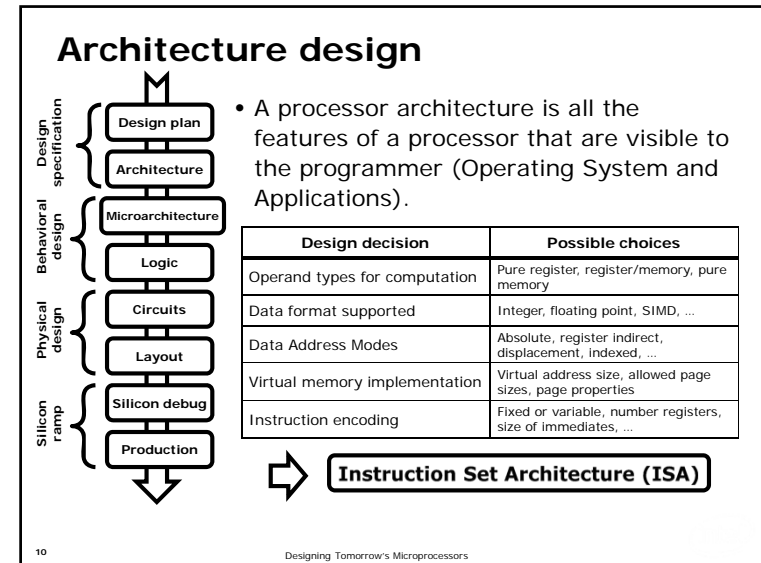
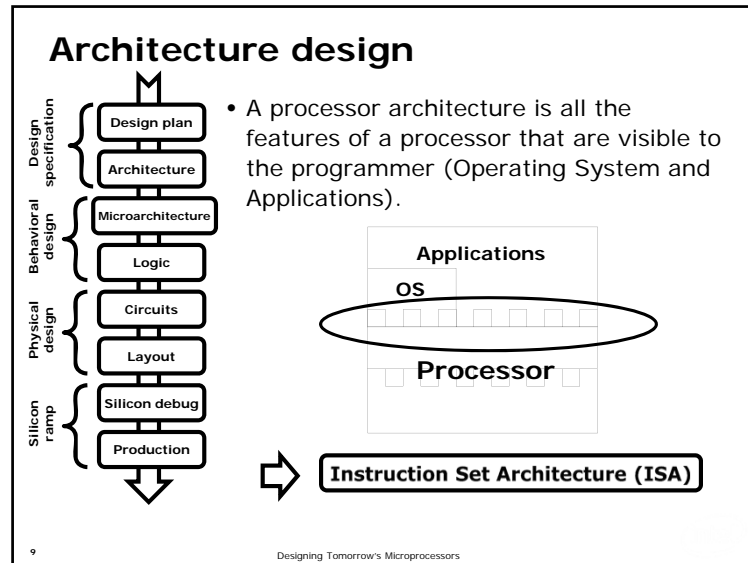
Market	Product	Priorities
Server	High-end server	Performance, reliability, and multiprocessing
	Farm/Blade server	Performance, reliability, and multiprocessing within power limit
Desktop	High-end desktop	Performance
	Mainstream desktop	Balanced performance and cost
	Value desktop	Lowest cost at required performance
Mobile	Mobile desktop replacement	Performance within power limits
	Mobile battery optimized	Power and performance
Embedded	Mobile handheld	Ultralow power
	Consumer electronics and appliances	Lowest cost at required performance

Microprocessor Products and Market segments



8

Designing Tomorrow's Microprocessors

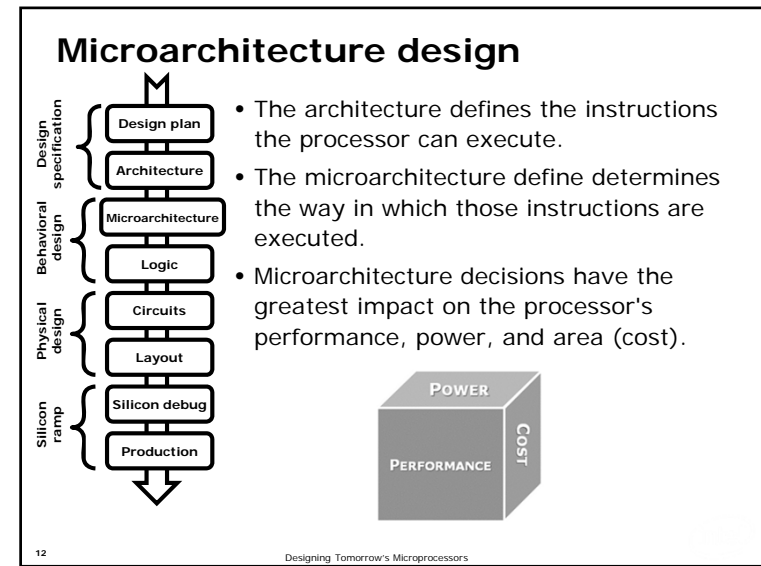


Architecture design

Instruction Set Architecture (ISA) Category		
CISC	Complex Instruction Set Computers	Complex but compact instructions
RISC	Reduce Instruction Set Computers	Simple instructions
VLIW	Very Long Instruction Word	An instruction is a set of operations grouped together by the compiler

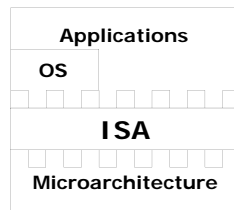
Category	Architecture	Processor	Manufacturer
CISC	VAX	MicroVax 78032	DEC
	X86	Pentium 4, Athlon XP	Intel, AMD
RISC	SPARC	UltraSPARC IV	Sun
	PA-RISC	PA 8800	Hewlett Packard
	PowerPC	PPC 970 (G5)	IBM
VLIW	EPIC	Itanium 2	Intel

11 Designing Tomorrow's Microprocessors



Microarchitecture design

- Microarchitecture changes are not visible to the programmer and can improve performance without software changes.
- Because microarchitectural changes maintain software compatibility, processor microarchitecture have changed much more quickly than architectures.
- Today's higher integration capacity allows more complex techniques to be implemented.



13

Designing Tomorrow's Microprocessors

Microarchitecture design

The microarchitecture defines the different functional units on the processor as well as the interactions and division of work between them.

14

Designing Tomorrow's Microprocessors

Microarchitecture design

- Designing a processor microarchitecture involves trade-offs of IPC, frequency, die area, power, and design complexity.
 - Number of stages of the pipeline.
 - Instruction issue width.

15

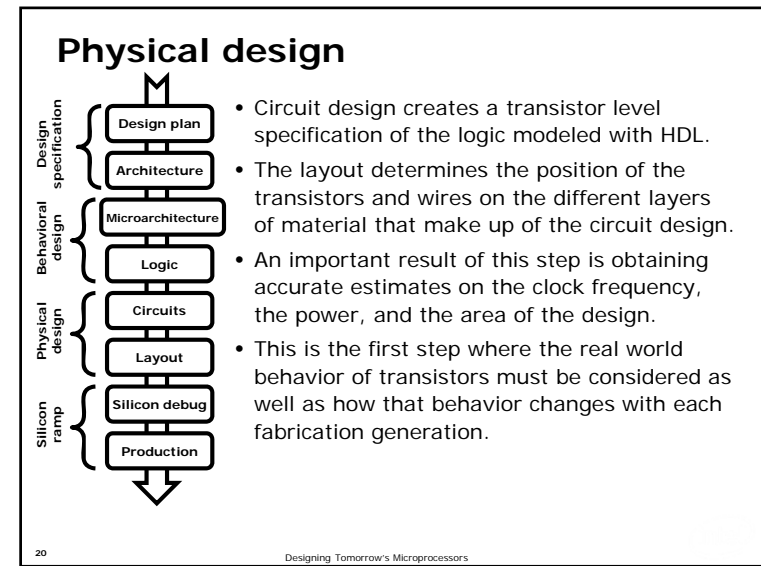
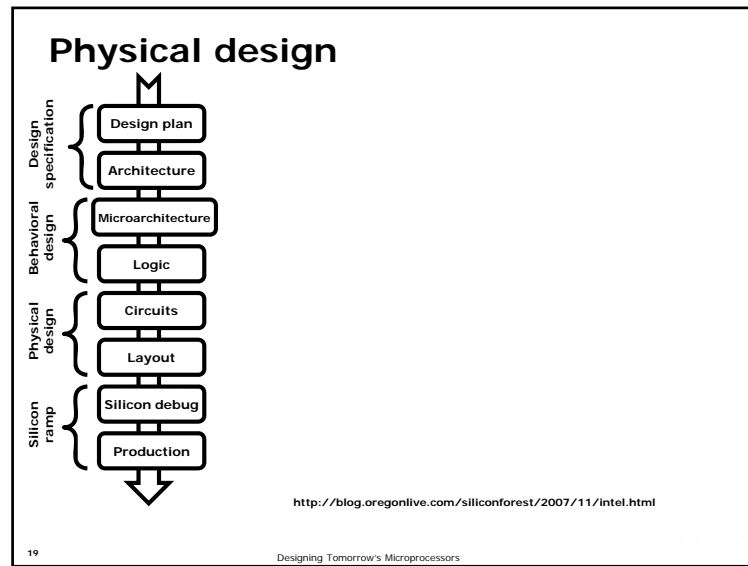
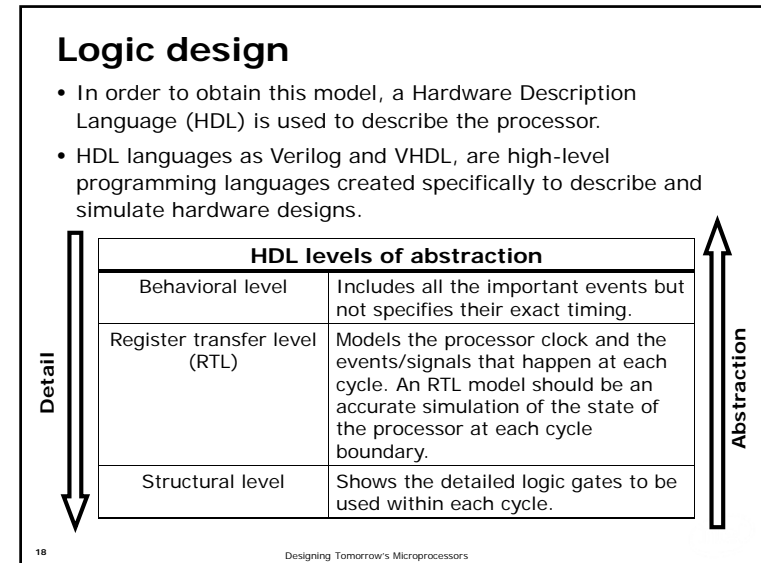
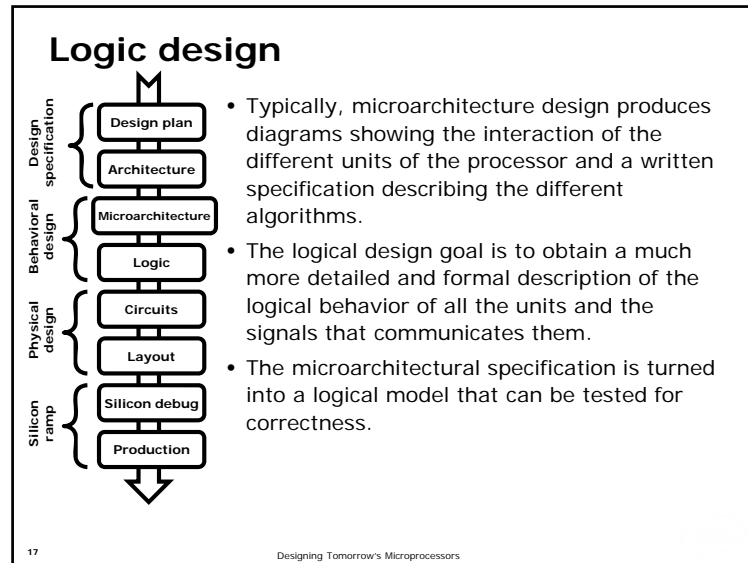
Designing Tomorrow's Microprocessors

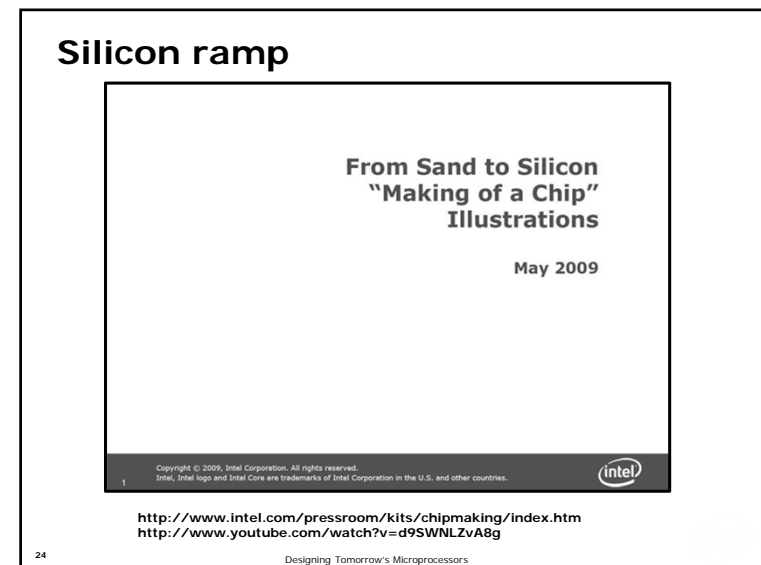
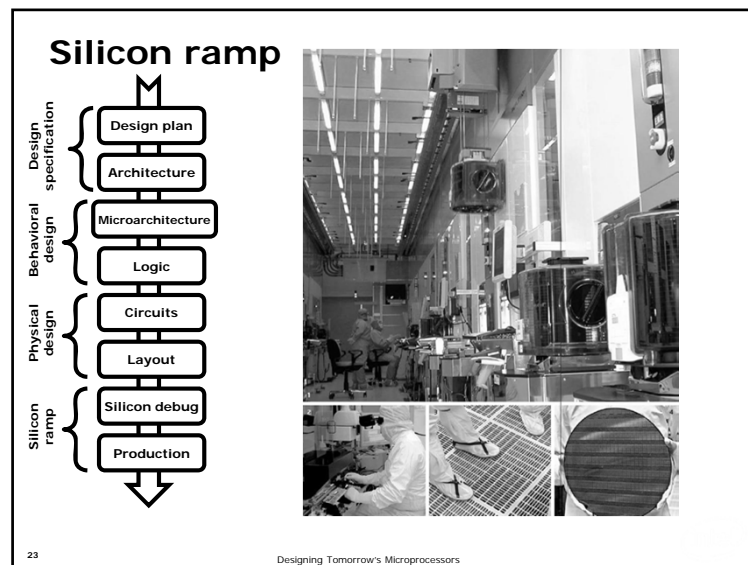
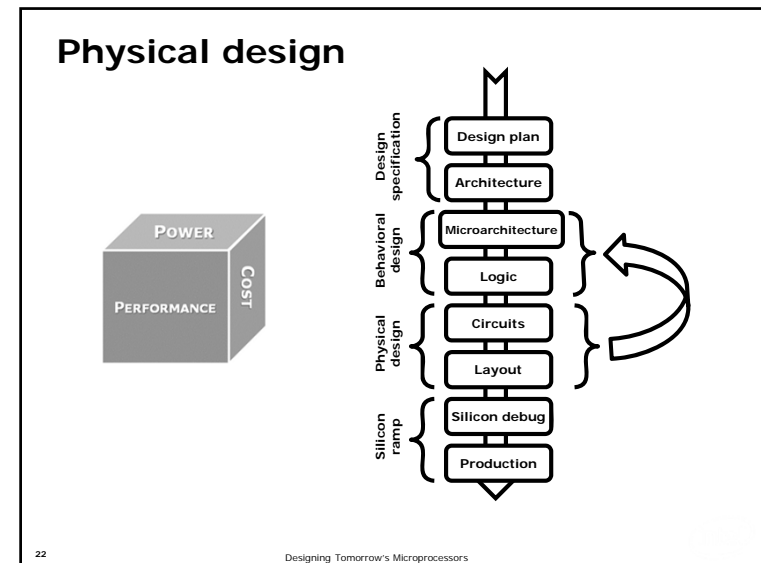
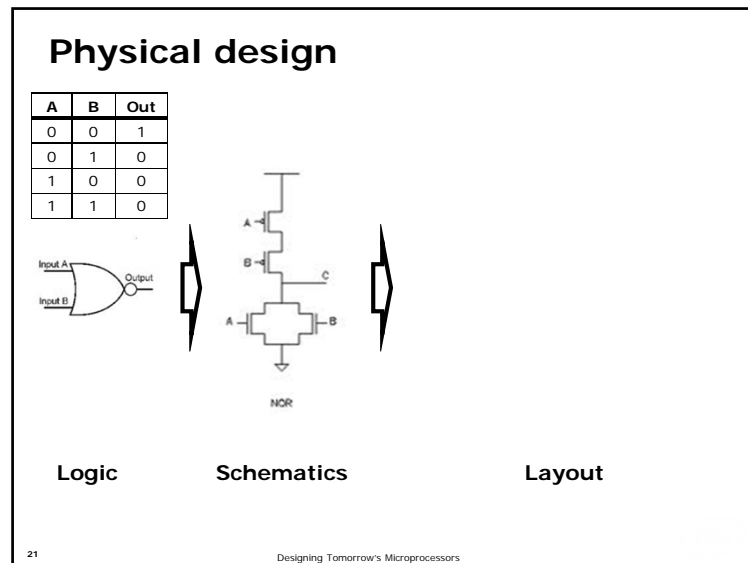
Microarchitecture design

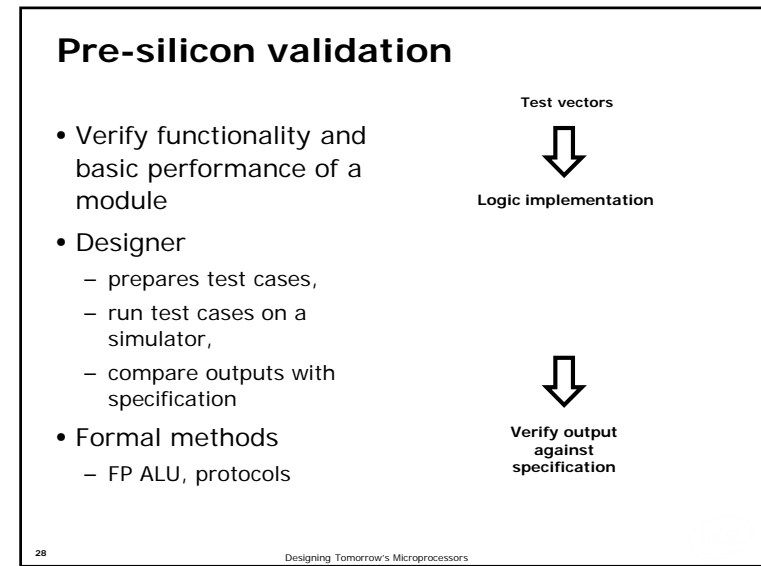
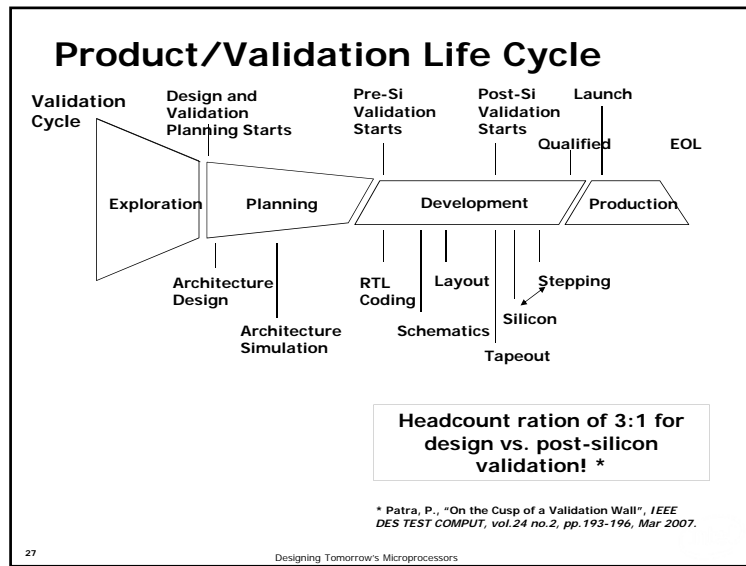
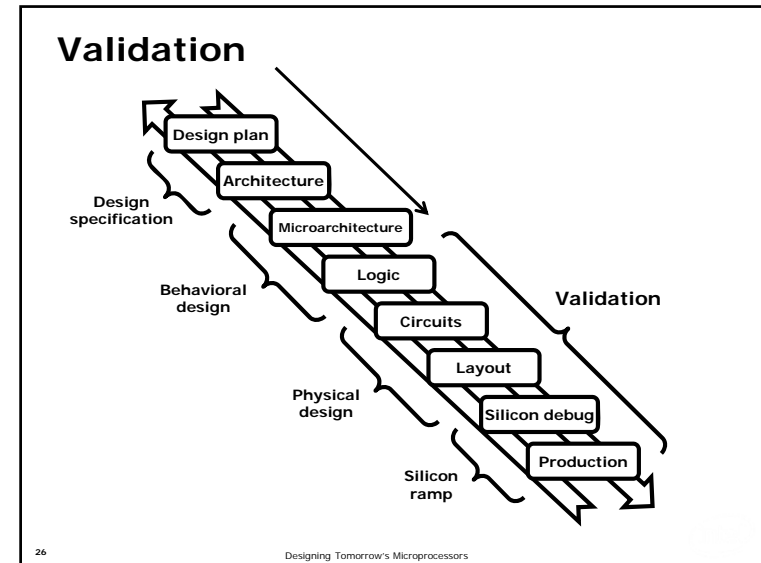
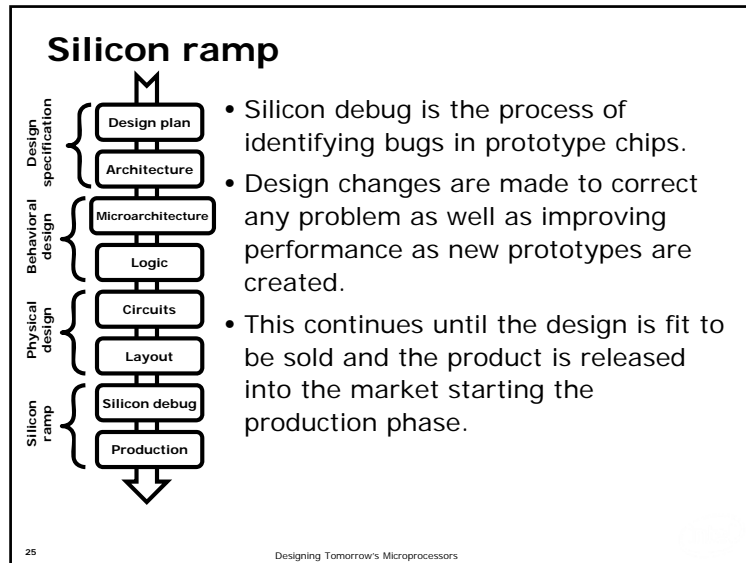
- Designing a processor microarchitecture involves trade-offs of IPC, frequency, die area, power, and design complexity.
 - Number of stages of the pipeline.
 - Instruction issue width.
 - Methods to resolve control dependencies.
 - Methods to resolve data dependencies.
 - Memory hierarchy.
 - In-order / out-of-order execution
 - Multi threading
 - Branch prediction
 - Number and type of functional units

16

Designing Tomorrow's Microprocessors







Post-silicon validation methods

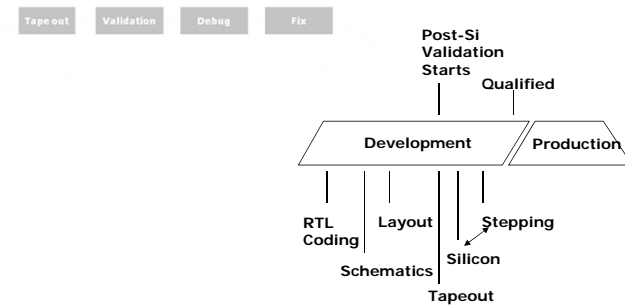
- Random Instruction Testing (RIT)
- Real software testing
- Output checked against architectural simulators

29

Designing Tomorrow's Microprocessors

Post-silicon validation

- Several iterations or "steppings"



30

Designing Tomorrow's Microprocessors

Platform Validation

Extensive operating system, network and application testing ensures compatibility in today's environment

31

Designing Tomorrow's Microprocessors

DFT/DFV features

- Design for Test/ Design for Validation
 - Specific HW features to ease testing/debug
- Scan logic
 - Logic to load and extract fine grain data
 - Allow reachability
 - Allow observability

32

Designing Tomorrow's Microprocessors

Cost of Silicon Bug

"Finding bugs in model testing is the least expensive approach, but the cost of a bug goes up 10x if it's detected in component test, 10x more if it's discovered in system test, and 10x more if it's discovered in the field, leading to a failure, a recall, or damage to a customer's reputation."

John Bourgojn, MIPS CEO
at a DesignCon 2006 panel

33

Designing Tomorrow's Microprocessors

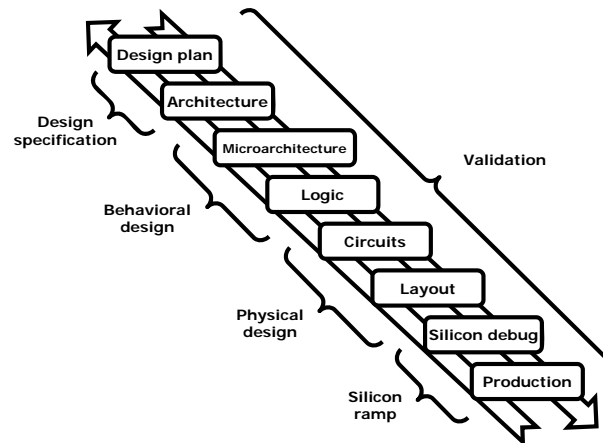
Validation vs. Test

	Validation	Test
Goal:	Does the design meet the intent? → Check design correctness.	Does the product work as designed? → Check manufacturing correctness.
Time scale	Weeks / Months	Seconds
Medium	Platform / Tester	Tester
Means	Test patterns, software, logic analyzers, oscilloscopes, DFT, DFV, etc.	Test patterns, DFT
Volume	Small production sample.	Every manufactured die.

34

Designing Tomorrow's Microprocessors

Design Steps



35

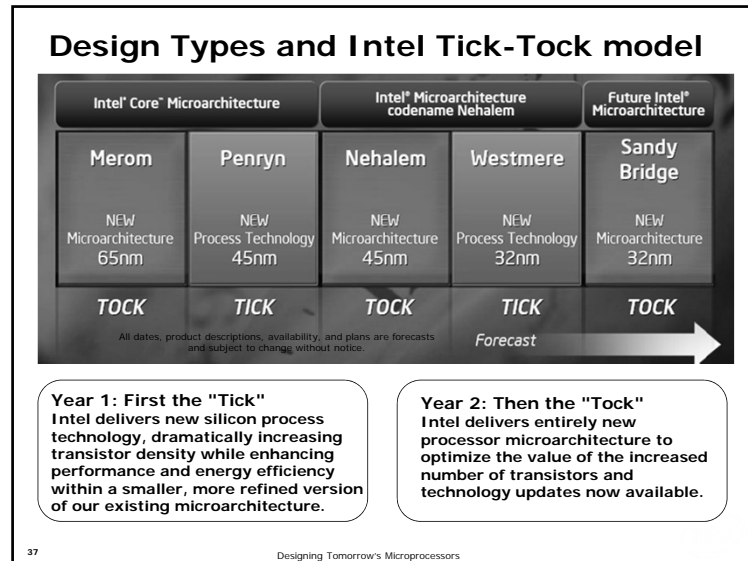
Designing Tomorrow's Microprocessors

Design Types and Intel Tick-Tock model

Design Type	Reuse
Lead	Little to no reuse
Proliferation	Significant logic changes and new manufacturing process
Compaction	Little or no logic changes, but new manufacturing process
Variation	Some logic changes on same manufacturing process
Repackage	Identical die in different package

36

Designing Tomorrow's Microprocessors



Conclusions

- Moore's Law predicts the increase in transistor density.
- Transistor scaling and growing transistor budgets have allowed microprocessors performance to increase at a dramatic pace, but they have also increased the effort of microprocessor design.
- The production of new fabrication generations is inevitably more complex than previous generations.
- This implies a higher effort in validation at all the design levels.
- There is a need for new and better methodologies and tools to help in the different tasks.
- A sustained research at all the steps but specially at the fields of microarchitecture, process technology, and validation is required.

38

Designing Tomorrow's Microprocessors

Design Cycle for Microprocessors

Raúl Martínez
(raul.martinez@intel.com)

Intel Barcelona Research Center

Aula Empresa, Facultat d'Informàtica de Barcelona

© Intel Corporation, 2013



Metodología de Investigación en Microprocesadores

Pedro Marcuello

Intel Barcelona Research Center

Aula Empresa, Facultat d'Informàtica de Barcelona, February 2010

© Intel Corporation, 2010

Nota del presentador

Esta presentación solo muestra opiniones personales y no las de Intel Corporation

2

Methodology for Research in Microprocessors

Agenda del curso

- Introducción y futuras tendencias en microarquitectura
- Ciclo de vida en el diseño de microprocesadores
- Investigación en microprocesadores
- Arquitecturas multi-core
- Programación paralela
- Systems-on-Chip
- Reducción de consumo
- Fiabilidad
- Máquinas virtuales co-diseñadas

Qué se hace ahora?

Qué se puede hacer?

3

Methodology for Research in Microprocessors

Objetivo de este bloque

- Mostrar cómo es el trabajo de un investigador
 - Qué tareas se realizan
 - De dónde vienen las ideas
 - Cómo se evalúan las ideas
 - Qué se hace con estas ideas
- Convenceros de que puede ser un buen trabajo

4

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del Entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Disseminación de resultados

5

Methodology for Research in Microprocessors

Definiciones

- Investigación:
 - RAE: Actividad que tiene por fin ampliar el conocimiento científico, sin perseguir, en principio, ninguna aplicación práctica
 - Wikipedia: La **investigación científica** es la búsqueda intencionada de conocimientos o de soluciones a problemas de carácter científico
- Investigador:
 - RAE: Persona que realiza investigación
 - Persona que tiene el firme convencimiento que sus conocimientos pueden mejorar un proceso o solucionar un problema

6

Methodology for Research in Microprocessors

Qué necesito para ser investigador?

- Infinitas ganas de aprender
- Mucha paciencia
- Mucha dedicación
- Mucho espíritu crítico
- Mucha iniciativa/inventiva

7

Methodology for Research in Microprocessors

Qué obtengo por ser investigador?

- Reconocimiento
- Realización personal
- Convertirte en un experto a nivel mundial en un tema concreto

8

Methodology for Research in Microprocessors

Reconocimiento

- Video "Intel's Rock Star"

9

Methodology for Research in Microprocessors

Reconocimiento?

- Todo el mundo sabe en que trabajas ...
 - 'No se, pero te estás muchas horas allí'
 - 'Tu trabajo consiste en acelerar ordenadores'
 - 'Hace que los ordenadores sean más pequeños y se calienten más'
- ... y lo valoran
 - Investigar en Intel era considerado profesión de riesgo en un banco para otorgarte una hipoteca porque los detectives no tenían ingresos fijos
- Por no hablar de los medios de comunicación

10

Methodology for Research in Microprocessors

Mi lugar de trabajo



11

Methodology for Research in Microprocessors

Y entonces?

- Si a un investigador le haces las siguiente encuesta
 - Te gusta tu trabajo?
 - Lo cambiarías por otro?
 - Lo cambiarías por otro trabajo razonable relacionado con la informática?
- Las respuestas serían por amplia mayoría
 - Sí → 95%
 - Sí → 70%
 - No → 90%

12

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del Entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Disseminación de resultados

13

Methodology for Research in Microprocessors

Fuentes de conocimiento

- Azar:
- El método científico

14

Methodology for Research in Microprocessors

Grandes descubrimientos por azar

- Gastronomía
 - Queso
 - Coñac
 - "LSD"
 - Patatas Chips
- Otros campos
 - Caucho vulcanizado

Y en la ciencia ... ???

15

Methodology for Research in Microprocessors

El azar en la ciencia

- Alexander Fleming: Penicilina
- Wilhem Roentgen: Rayos X
- Percy Spencer: Microondas

16

Methodology for Research in Microprocessors

Pregunta: En qué se parecen?

- a) Los tres recibieron el Premio Nobel por sus arduas investigaciones y descubrimientos
- b) Los tres eran estadounidenses
- c) Los tres estaban investigando cuando la suerte les sonrió
- d) Los tres anteriores son correctas

17

Methodology for Research in Microprocessors

El método científico

Conjunto de reglas aceptadas por la comunidad científica para guiar la investigación

1. Observación del medio
2. Planteamiento de hipótesis
3. Probación de las hipótesis mediante experimentación
4. Tesis o teoría científica (conclusiones)

18

Methodology for Research in Microprocessors

Investigación aplicada/tecnológica

- Investigación que se realiza en ingeniería y que genera conocimientos para el sector productivo
 - Farmacéuticas
 - Nuevas vacunas
 - Automóvil
 - Seguridad
 - Prestaciones
 - Consumo
 - Tecnología
 - Seguridad
 - Prestaciones / Velocidad
 - Consumo

19

Methodology for Research in Microprocessors

El método científico en el estudio de microprocesadores

1. Observación del medio
 - Estudio del entorno
 - Detectar qué se ha de mejorar
2. Planteamiento de hipótesis
 - Plantear cómo se puede mejorar
3. Probación de hipótesis
 - Preparación de experimentos
4. Teoría científica
 - Patentes/Prototipos
 - Artículos

20

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del Entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Diseminación de resultados

21

Methodology for Research in Microprocessors

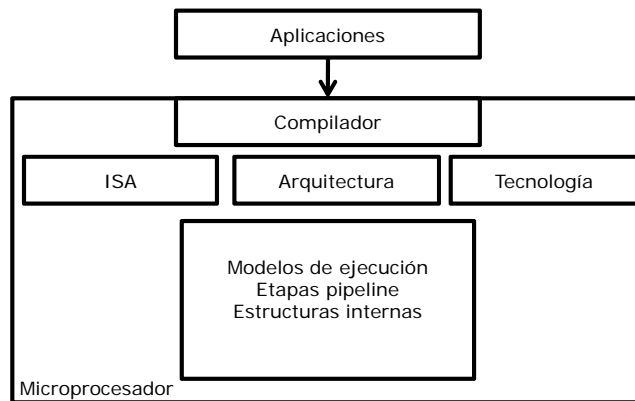
Estudio del entorno / Observación

- Observación del fenómeno a fin de poder formular una hipótesis
- Incluye
 - Tomar medidas sobre el proceso
 - Comparar diferentes realizaciones del suceso

22

Methodology for Research in Microprocessors

El “entorno” microprocesador



23

Methodology for Research in Microprocessors

Posibles mejoras

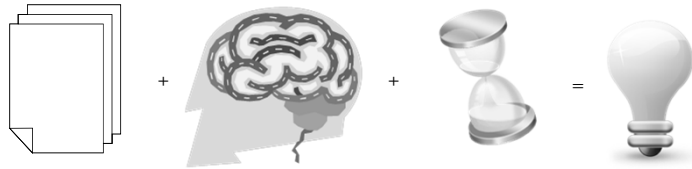
- Reducir el tiempo de ejecución de las aplicaciones
 - $T_{\text{ejec}} = \# \text{inst} * \text{CPI} * \text{Frec}$
- Consumir menos (más tiempo de batería)
 - $P = \text{Frec} * V^2 * C * \text{Act}$

Arquitectura

24

Methodology for Research in Microprocessors

Propuesta de ideas



25

Methodology for Research in Microprocessors

Utilidad

- La idea debe ser nueva
 - Hacer un estudio exhaustivo del tema
- La idea debe mejorar una situación real / actual
 - El problema detectado no debe ser artificial

26

Methodology for Research in Microprocessors

Qué se puede hacer?

- Procesadores actuales IPC ~ 1
 - Con un Ancho de Banda de Issue de 4/6
 - Rendimiento de memoria $\sim 70/80\%$ tasa aciertos
 - Explotar TLP / MLP además de ILP
- Pentium-4 consumía cerca de 80W a 3.6GHz
 - Actuales Core 2 Duo está entorno 35W a 2.2GHz

27

Methodology for Research in Microprocessors

Ejemplos

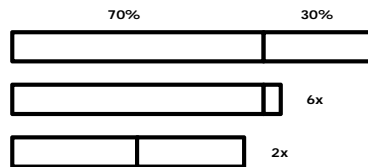
- Problema: el coste de SQRT de ~ 40 ciclos
 - Buscar un algoritmo que tarde en media 10 ciclos
- Problema: La tasa de fallos en cache es muy elevada
 - Buscar un prefetcher mejor basado en agrupar instrucciones dependientes
- Problema: el ROB se llena en caso de fallos en L2
 - Graduar las instrucciones especulativamente y para las instrucciones dependientes predecir el valor que vendrá de L2

28

Methodology for Research in Microprocessors

Impacto de la idea. Ley de Amdahl

$$A = \frac{1}{(1 - F_m) + F_m/A_m}$$



29

Methodology for Research in Microprocessors

Tienen impacto nuestros ejemplos?

- SQR?
 - Representan en media menos del 0.01% de las instrucciones
- Fallos de memoria?
 - ~30% instrucciones son loads
 - ~10% de los accesos a memoria son fallos
- ROB?
 - Cada vez la latencia de L2 es mayor
 - Cada vez se hace fetch de mayor número de instrucciones
 - ROB no aumenta significativamente en cada generación

30

Methodology for Research in Microprocessors

El factor tiempo en el estudio de microprocesadores

- La investigación en microprocesadores se hace a 5-7 años vista
 - 4 últimos años son para la fabricación del chip, pero el diseño está ya congelado
 - Tecnología de fabricación todavía no existe
- Esto influye en el entorno
 - Qué querrán los consumidores de aquí 6 años?
 - Cómo serán las aplicaciones de entonces?

31

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Diseminación de resultados

32

Methodology for Research in Microprocessors

Pero antes ...

- Detallar la propuesta
 - Identificar los bloques afectados
 - Encajar la técnica en el pipeline
 - Asignar cuándo y durante cuánto va a realizarse cada paso de la técnica
 - Detectar algún posible caso patológico
 - Rollbacks
- Establecer nuestra microarquitectura base
 - Actual
 - Justa

33

Methodology for Research in Microprocessors

Nuestros ejemplos

- SQRD:
 - Sólo afecta a la ALU donde está el divisor/SQRD
- Memoria:
 - Tabla adicional para aparejar stores/loads
 - IP de la operación de memoria
 - @ accedida por las operaciones de memoria
 - Casos patológicos
 - Si en una pareja store/load, hay un store entre ellos que escribe en la misma @?
 - Timing
 - Actualizaciones en graduación o en ejecución

34

Methodology for Research in Microprocessors

Nuestros ejemplos II

- ROB:
 - Predictor de valor para loads
 - Hardware para rollback
 - Guardar todo el estado arquitectónico
 - Timing
 - Actualizaciones del predictor
 - Acceso del predictor

35

Methodology for Research in Microprocessors

Estudio de potencial

- Simulador sencillo que nos pueda dar una indicación del potencial
 - Técnica ha de estar detallada
 - Las demás partes del procesador según el impacto

Si el potencial es muy inferior al esperado

→

Volver a la casilla de salida

36

Methodology for Research in Microprocessors

Cómo experimentar?

Tipo	Desarrol.	Depurac.	Adaptab.	Error	O(t)
Diseño chip	Altísima	Muy costosa	Nula	0	Igual
FPGA	Muy Alto	Muy costosa	Si	0	Medio
Simulador Detallado	Alto	Costosa	Si	Poco	Muy lento
Simulador Esquemático	Medio/Alto	Medio/Costosa	Si	Medio/Poco	Lento

37

Methodology for Research in Microprocessors

Con qué experimentar?

- Programas de prueba (benchmarks)
 - Coherentes con la propuesta
 - Reflejen el problema que queremos tratar
 - Disponibles para todo el mundo
 - Programas reales / sintéticos
- Suites reconocidas
 - SPEC.org
 - MediaBench, PhysicBench, etc.

38

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del Entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Disseminación de resultados

39

Methodology for Research in Microprocessors

Qué he de mirar?

- Fundamentalmente, aquello que estamos intentando mejorar
 - Velocidad → Tiempo de ejecución
 - Consumo → Potencia consumida
- Pero se ha de intentar justificar el por qué de la mejora / empeoramiento
 - Tratamiento de outliers
 - Números extraños
 - Fuera de rango
 - Comportamientos anómalos

40

Methodology for Research in Microprocessors

Métricas

- Métricas reales
 - Tiempo de ejecución (s)
 - Potencia consumida (W)
 - Fallos por año
- Otras métricas
 - IPC
 - Fallos de cache / Fallos predicción
 - ED / ED²

41

Methodology for Research in Microprocessors

Resumir todo en un número

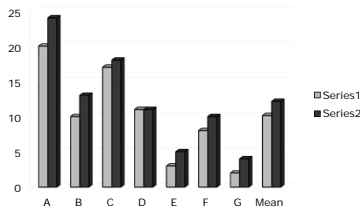
- Medias
 - Aritmética
 - Distribuciones normales
 - Tiempo, Tasas de fallos
 - Sensible a outliers
 - Geométrica
 - Distribuciones no normales
 - Insensible a outliers
 - Rendimiento
 - Harmónica
 - Insensible a outliers
 - Rendimiento, velocidad
- NO HACER MAL USO DE ELLAS !!!

42

Methodology for Research in Microprocessors

Ingeniería de números I

- Relatividad
 - Porcentaje de mejora de las técnicas 1 y 2



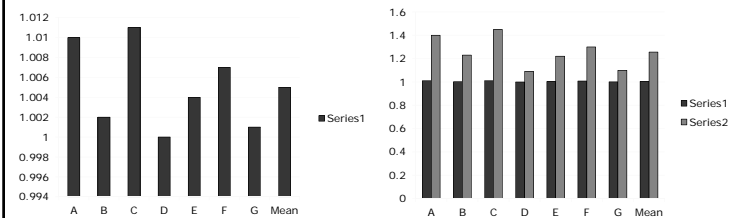
- "La serie 2 mejora la serie 1 en un 20%"

43

Methodology for Research in Microprocessors

Ingeniería de números II

- Parcialidad
 - ... la técnica A obtiene una mejora de 0,5%



- ... la técnica A obtiene una mejora del 26%

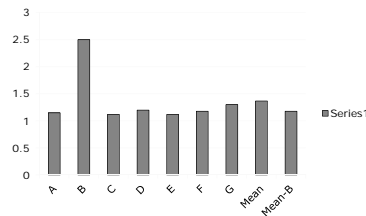
44

Methodology for Research in Microprocessors

Ingeniería de números

- Outliers

- ... la técnica A obtiene una mejora en rendimiento del 38%



- ... la técnica A (sin B) obtiene una mejora del 17%

45

Methodology for Research in Microprocessors

Ingeniería de números IV

- Imaginación

- Nuestra técnica obtiene una mejora del 15% en el ratio fallos de cache por cada 1000 fallos de saltos

46

Methodology for Research in Microprocessors

Agenda

- La investigación científica
- El método científico
- Estudio del Entorno. Propuesta de ideas
- Preparación y validación de experimentos
- Interpretación de resultados
- Diseminación de resultados

47

Methodology for Research in Microprocessors

Diseminación de resultados

- Compartir tus ideas con los demás investigadores del mundo
 - Escribir un artículo
 - Presentarlo en una conferencia
 - Publicarlo en una revista
 - Escribir una patente
 - Escribir una tesis

48

Methodology for Research in Microprocessors

Y luego ...

- Fama, reconocimiento, trabajo y...

Vuelta a empezar

49

Methodology for Research in Microprocessors

Agradecimientos

- Ramon Canal
- Josep-Llorenç Cruz
- Pepe González
- Fernando Latorre
- Javier Lira
- Grigorios Magklis
- Raúl Martínez

50

Methodology for Research in Microprocessors

Application Profiling and Performance Evaluation

Ferad Zyulkyarov
Intel Labs Barcelona
January 29, 2013

Outline

- Introduction to profiling
- Platform independent profiling
- Platform dependent profiling
- Profiling parallel programs
- Power profiling

Outline

- ⇒ • Introduction to profiling
 - Platform independent profiling
 - Platform dependent profiling
 - Profiling parallel programs
 - Power profiling

Introduction to Profiling

- What is profiling?
- Why profiling is important?

Introduction to Profiling

- What is profiling?
 - Recording and analyzing the behavior and the characteristics of a program at runtime
- Why profiling is important?
 - Profiling is important for optimization



Introduction to Profiling

- What is profiling?
 - Recording and analyzing the behavior and the characteristics of a program at runtime
 - Examples:
 - Functions which are called most
 - The type of instructions executed
 - Cache misses
 - Etc.
- Why profiling is important?
 - Profiling is important for optimization
 - Optimization: identify and resolve bottlenecks
 - Identify the parts within the program which cause the program to run slow and improve these program parts



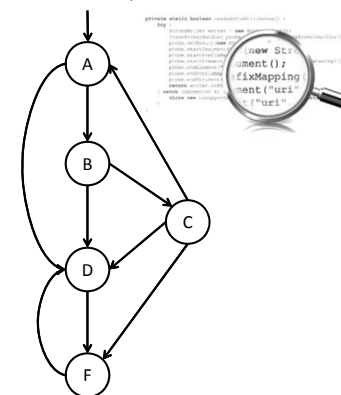
Outline

- Introduction to profiling
- ➡ • Platform independent profiling
- Platform dependent profiling
- Profiling parallel programs
- Power profiling

Platform Independent Profiling

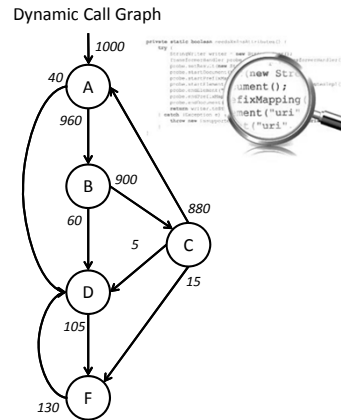
- Platform independent results
 - Obtained by analyzing the program itself without considering the platform
 - Expressed with platform independent metrics
- Application level optimization
 - Optimizations will improve the performance of the program for all platforms

Static Call Graph



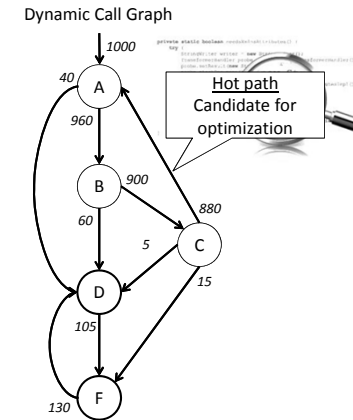
Platform Independent Profiling

- Platform independent results
 - Obtained by analyzing the program itself without considering the platform
 - Expressed with platform independent metrics
- Application level optimization
 - Optimizations will improve the performance of the program for all platforms



Platform Independent Profiling

- Platform independent results
 - Obtained by analyzing the program itself without considering the platform
 - Expressed with platform independent metrics
- Application level optimization
 - Optimizations will improve the performance of the program for all platforms

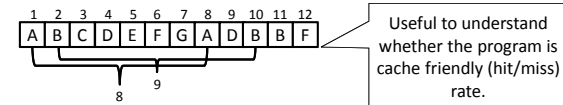


Other Platform Independent Metrics

- Same memory reference distance
- Number of instructions
 - Static
 - Dynamic
- Working data set

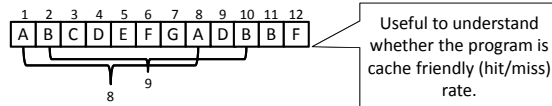
Other Platform Independent Metrics

- Same memory reference distance



Other Platform Independent Metrics

- Same memory reference distance



Cache with size 10



Hit



Miss



Cache with size 5

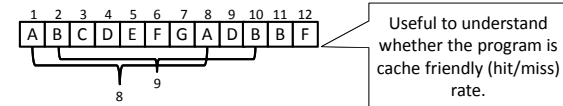


2



Other Platform Independent Metrics

- Same memory reference distance



Cache with size 10



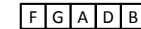
Hit

5

Miss

7

Cache with size 5



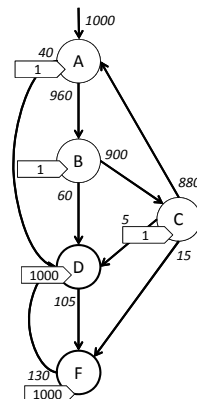
3

9

Other Platform Independent Metrics

- Number of instructions

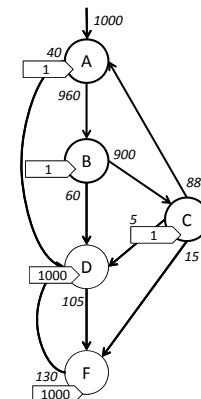
- Static
- Dynamic



Other Platform Independent Metrics

- Number of instructions

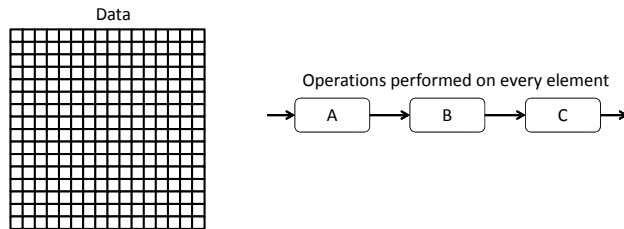
- Static
- Dynamic



For performing an effective optimization is necessary to find the part of the program where most of the instructions are executed.

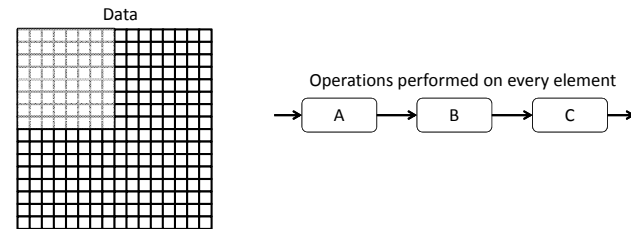
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



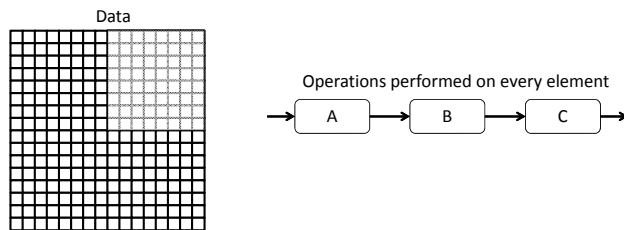
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



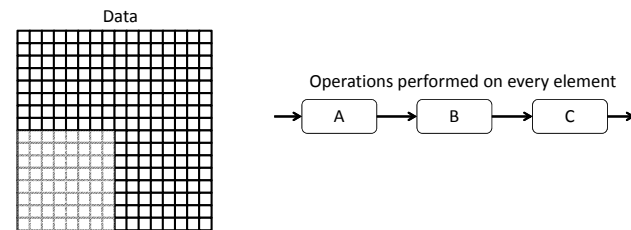
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



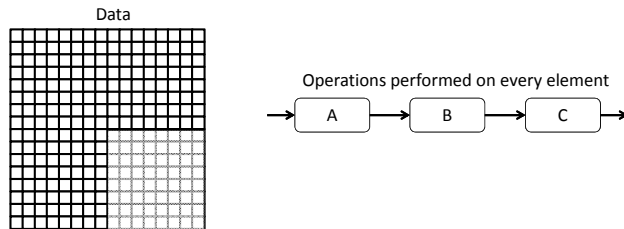
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



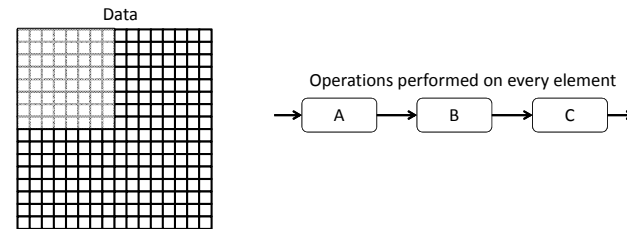
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



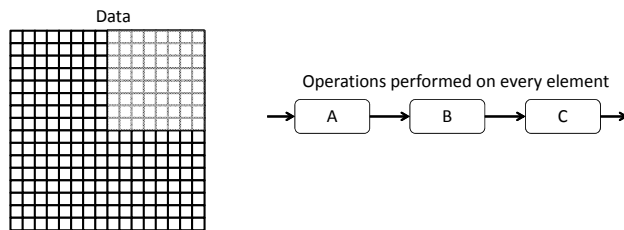
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



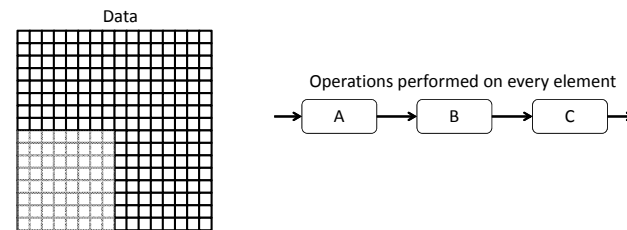
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



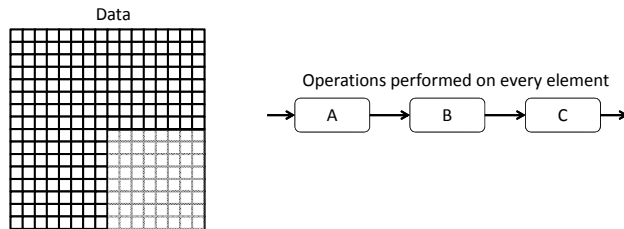
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



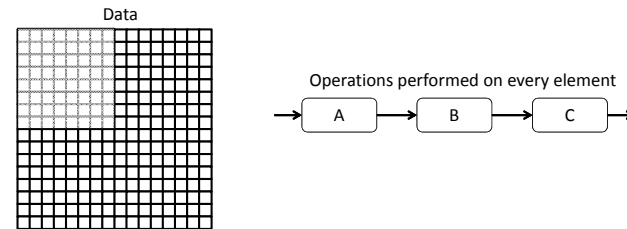
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



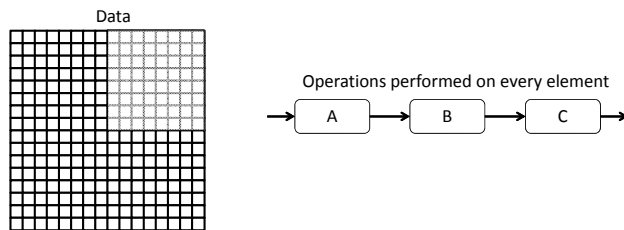
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



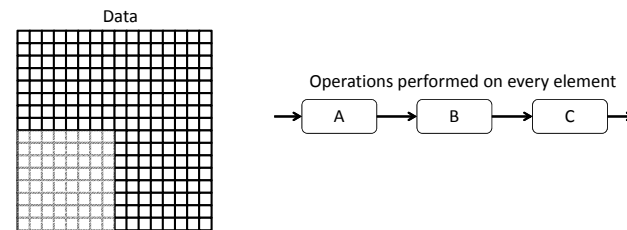
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



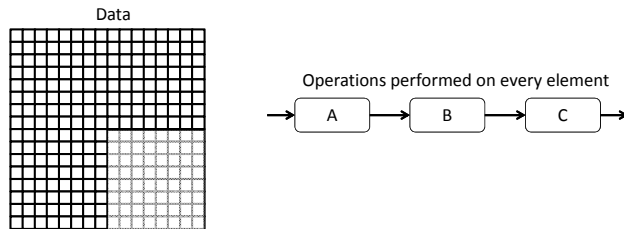
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



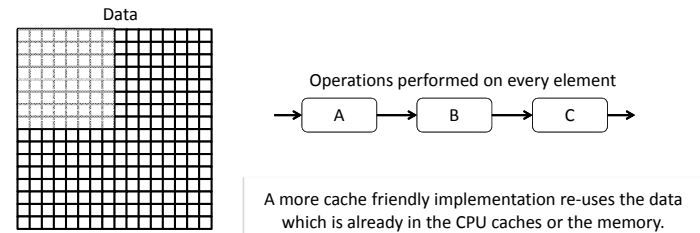
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



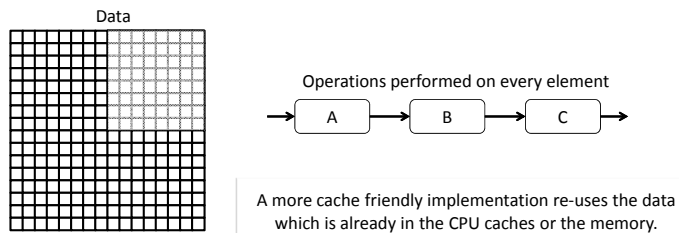
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



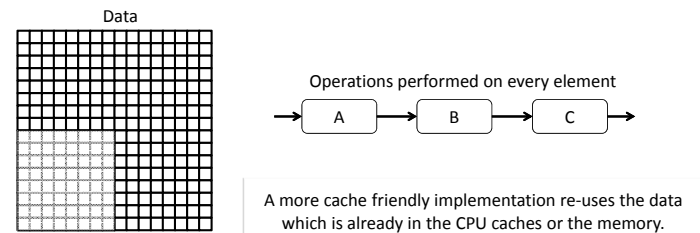
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



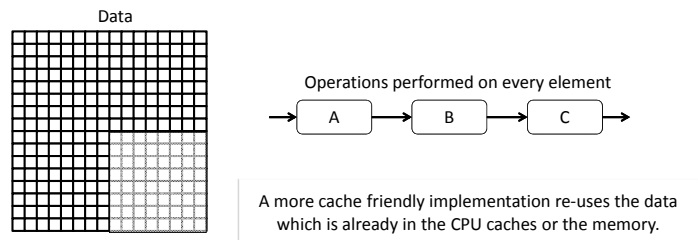
Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches



Other Platform Independent Metrics

- Working data set
 - The amount of memory the application operates on
 - Important to know and organize with respect to the available system memory and CPU caches

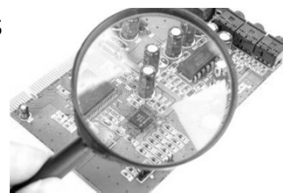


Outline

- Introduction to profiling
- Platform independent profiling
- ➡ • Platform dependent profiling
- Profiling parallel programs
- Power profiling

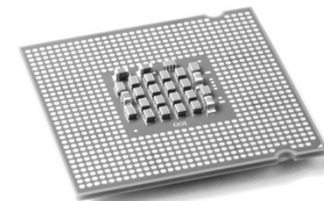
Platform Dependent Profiling

- Platform dependent results
 - Obtained by analyzing the underlying system
 - Expressed through platform dependent metrics
- Platform specific optimizations
 - Optimizations will improve the performance of the program for that specific (or similar) platform



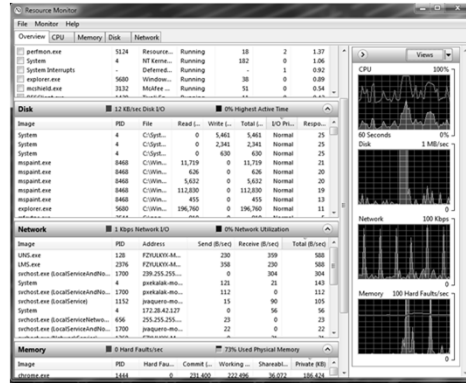
CPU Performance Monitoring

- Execution unit
 - Instructions executed
 - Type of instructions
 - Instruction per cycle
- Cache
 - Hits/misses
- Branch predictor
 - Mispredicts
- TLB
 - Hits/misses
 - Page faults
- Others



System Level Performance Monitoring

- Operating System
 - Context switches
 - User space execution
 - Kernel space execution
 - Others
- Network
 - Data send/received
 - Packet loss
 - Latency
 - Bandwidth
 - Jitter
 - Others
- Disk
 - Data read/written
 - Bandwidth
 - Others
- Memory
- Graphics

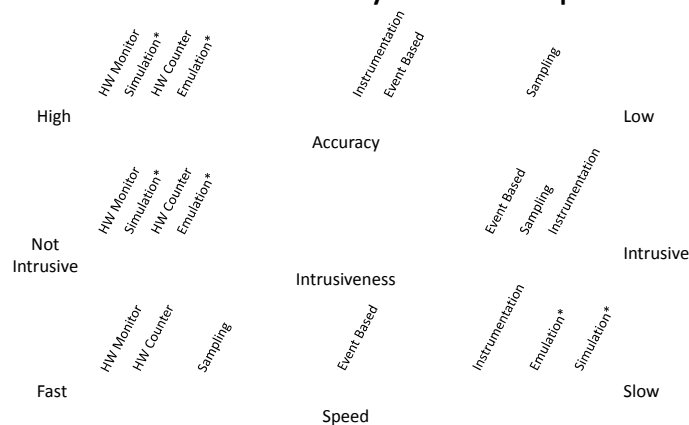


Performance Analysis Techniques

- Hardware monitors
- Hardware counters
- Event based
- Statistical sampling
- Instrumentation
- Emulation and simulation
- Hybrid
 - Combination of some of the above



Pros and Cons for the Different Performance Analysis Techniques



Example Profiling Tools

- VTune
- CodeAnalyst
- Gprof
- Visual Studio Profiler
- PAPI
- Pin
- Valgrind
- Netbeans Profiler
- JProfiler
- And many others

Identifying Problematic Program Code

- Code where most of the
 - CPU cycles are burnt
 - Cache misses happen
 - TLB misses happen
 - Branch mispredictions happen

Outline

- Introduction to profiling
- Platform independent profiling
- Platform dependent profiling
- ⇒ • Profiling parallel programs
- Power profiling

Profiling Parallel Programs

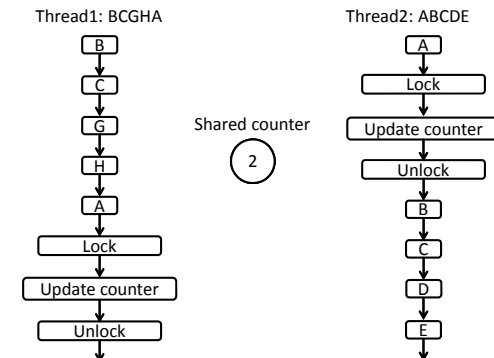
- What do we want to find when profiling parallel programs?

We want to find when, why and how much of the program does not execute in parallel?

- Find lock contention
 - which causes a program execution to serialize
- Find critical path
- Find bottlenecks

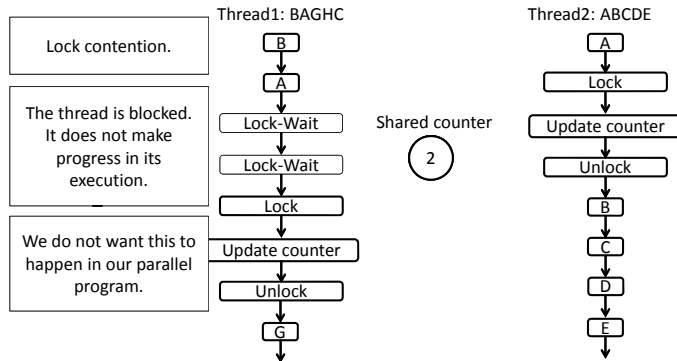
Example Parallel Program

Suppose we have two threads. Each thread iterates over a stream of letters and increments a shared counter when it encounters "A".



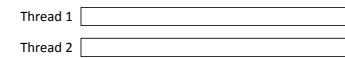
Lock Contention

Suppose we have two threads. Each thread iterates over a stream of letters and increments a shared counter when it encounters "A".

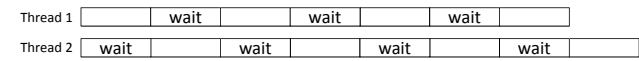


Parallel Execution

How the execution of a parallel program should look like

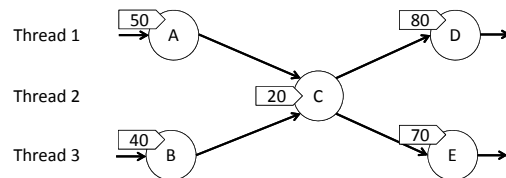


How the execution of a parallel program should not look like



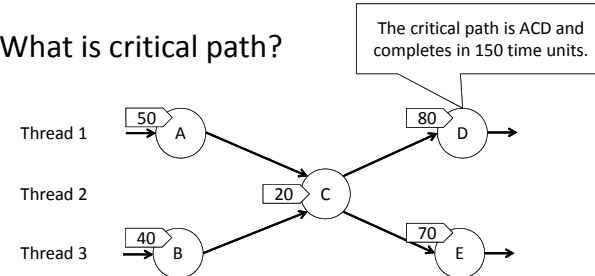
Critical Path

- What is critical path?



Critical Path

- What is critical path?

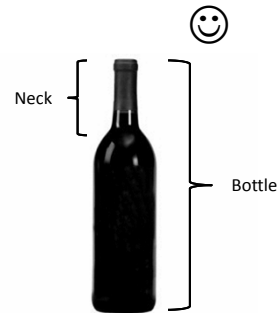


The execution time of a program depends on the critical path. The program will not complete before the critical path completes execution.

Optimizations should target reducing the execution time of the critical path.

Bottleneck

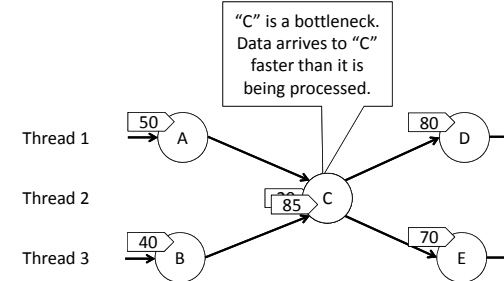
- What is bottleneck?



Application Bottleneck

A bottleneck is a part of the program (or system component) which lies on the critical path and has unacceptably low performance.

Bottlenecks severely limit the performance of the entire program (or system) and should be located and eliminated.

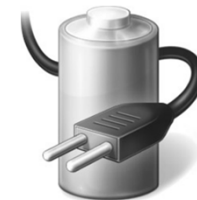


Outline

- Introduction to profiling
- Platform independent profiling
- Platform dependent profiling
- Profiling parallel programs
- ➡ Power profiling

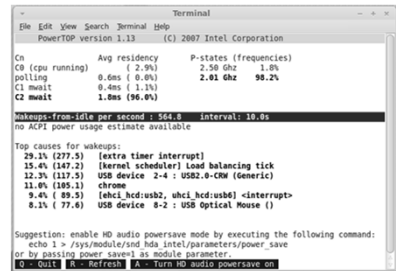
Power Profiling

- Power consumption is very important in the world of battery powered (mobile) devices
- There exist guidelines for writing power-aware programs which consume less power




System Level Power Profiling

- Identify the applications which consume most power
- Used for manual or automated power management
- Example:
 - PowerTop



Application Level Power Profiling

- Identify parts in the program which cause excessive power consumption
 - Functions which excessively use GPS, WiFi, 3G, graphics
 - Functions which prevent falling to sleep mode
 - Functions which use power-expensive instructions
 - Compare the power consumption of alternative implementations
 - The optimizations target re-writing the program code in more power-efficient manner
- 



Power Optimization Strategies


Application Level

- Use events instead of continuous polling
- Update graphics less often
- Redraw only the window components that change
- Operate in small data that fits in the caches
- Avoid or reduce background activity

Power Efficiency: Developing Power Aware Apps
<http://software.intel.com/en-us/articles/energy-efficient-software-developing-power-aware-apps/>
 Optimizing Software Applications for Power
<http://software.intel.com/en-us/blogs/2010/11/29/optimizing-software-applications-for-power-part-1-of-13>


Questions?

Contact: feradz@gmail.com



Multi-core Processor Architectures

Ayose Falcón
Senior Research Scientist
Intel Barcelona Research Center, Intel Labs

 Seminari d'Empresa 2013
Facultat d'Informàtica de Barcelona, Jan/Feb 2013

Agenda

- Motivation
- Parallel architectures
- Cache coherency and consistency
- Interconnecting the cores
- Multithreading
- Concluding remarks

Ayose Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

2



Why parallel computing?

Simple task:
Eat a pizza with 6 slices as fast as possible!!




Ayose Falcón
Intel Barcelona Research Center


Multi-core Processor Architectures
Seminari d'Empresa 2013

3



Why parallel computing?

1x  = 6 Time Units

6x  = 1 Time Unit

Faster ... but hungrier !!

Ayose Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

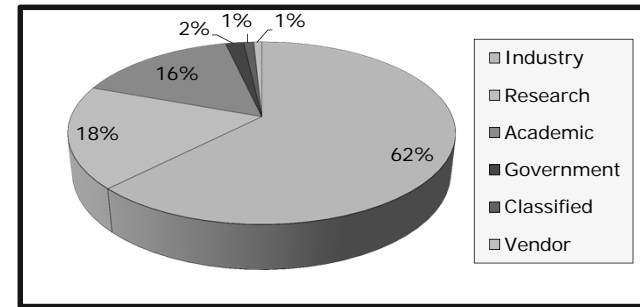
4



Parallel computing promises

- Increase speed and concurrency (big datacenters)
 - Solve larger problems (Human genome deciphering)
 - Process huge amount of data
 - Solve problems in real time and in due time
- More economic (Google – cluster computing)
- Power efficiency (Embedded computing)

Who is using parallel computing



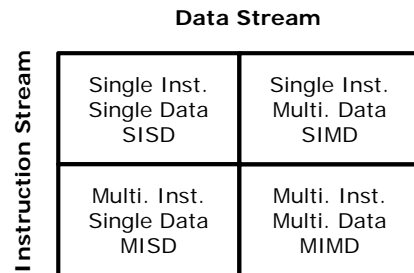
Agenda

- Motivation
- **Parallel architectures**
- Cache coherency and consistency
- Interconnecting the cores
- Multithreading
- Concluding remarks

Defining a parallel architecture

- **Sequential** architecture:
 - Single processor
 - Single control flow path
- **Parallel** architecture:
 - Multiple processors with interconnection network
 - Multiple control flow paths
 - Communication and synchronization
- **A parallel computer can be defined as a collection of processing elements that communicate and cooperate to solve large problems fast**

Parallel architectures



Categorization by Michael FLYNN (1972)

Ayosé Falcón
Intel Barcelona Research Center

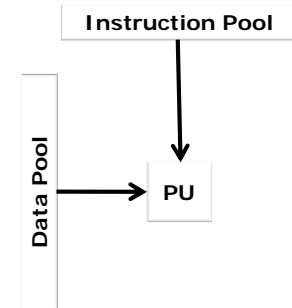
Multi-core Processor Architectures
Seminariis d'Empresa 2013

9



1) Single Instruction Single Data (SISD)

- First machines to be built
- Conventional sequential machines; no parallelism
 - Von Neumann architecture
- Deterministic execution
- Most common even today



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminariis d'Empresa 2013

10



SISD example: Univac I (1951)



Ayosé Falcón
Intel Barcelona Research Center

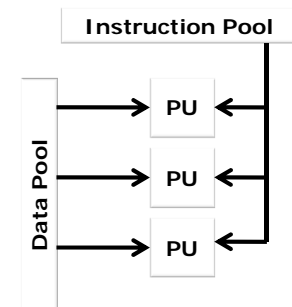
Multi-core Processor Architectures
Seminariis d'Empresa 2013

11



2) Single Instruction Multiple Data (SIMD)

- Exploit data-level parallelism (DLP)
- Best suited for "regular" applications
 - Graphics/image processing
- Simple SIMD units in uniprocessors
 - MMX/3DNow! (x86)
 - SSE/AVX/Altivec/...



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminariis d'Empresa 2013

12



SIMD example: ILLIAC IV (1976)



Ayosé Falcón
Intel Barcelona Research Center

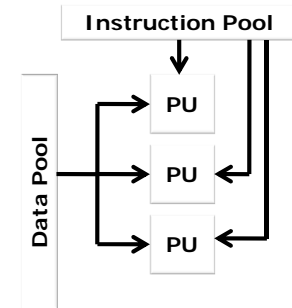
Multi-core Processor Architectures
Seminari d'Empresa 2013

13



3) Multiple Instructions Single Data (MISD)

- Aka **systolic arrays**
- Same data flowing through a linear array of processors
- Not many examples of this
- Hard to program
- Used for fault-tolerance



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

14



MISD example: Space shuttle flight control



Ayosé Falcón
Intel Barcelona Research Center

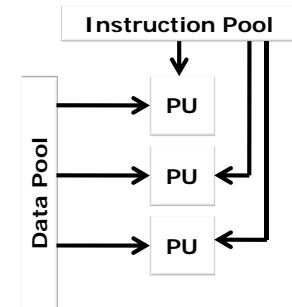
Multi-core Processor Architectures
Seminari d'Empresa 2013

15



4) Multiple Instructions Multiple Data (MIMD)

- Different processors working asynchronously and independently
- Exploit Thread-Level Parallelism (TLP)
- Chip multiprocessors and parallel supercomputers are MIMD



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

16



MIMD example: IA cluster



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

17



Bell's taxonomy of MIMD computers

Multicomputers

- Multiple address spaces
- System consists of multiple computers, called nodes
- Nodes are interconnected by a message-passing network
- Each node has its own processor, memory, NIC, and I/O devices

Multiprocessors

- Shared address space
- Further classified based on how memory is accessed:
 - Uniform Memory Access (UMA)
 - Non-Uniform Memory Access (NUMA)
 - Cache-Only Memory Access (COMA)
 - Cache-Coherent Non-Uniform Memory Access (cc-NUMA)

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

18



Shift from traditional parallel computing to multi-cores



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

19



Multi-core processor is a special kind of multiprocessor

- All processors are on the same chip
- Multi-core processors are **MIMD**:
 - Different cores execute different threads (Multiple Instructions)
 - Operate on different parts of memory (Multiple Data)
- Multi-core is a **shared memory multiprocessor**:
 - All cores share the same address space

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

20

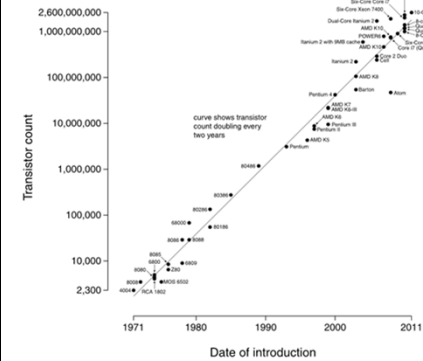


Why multi-core processors?

- For **individuals**:
 - Solve larger problems → divide & conquer
 - Concurrency
 - Power efficiency
- For **companies**:
 - Time-to-market → simply replicate cores
 - Easier to extract performance → but programmers have to do it
 - Better power and thermal management → die layout

Moore's law for multi-cores

Microprocessor Transistor Counts 1971-2011 & Moore's Law



- More and more transistors available
- Reaching power limit for single processors
- More on-chip RAM not really useful
- Hard to extract more ILP for uniprocessors
- Focus on TLP!

Multi-core programming

- Programmers must use threads or processes
 - Threads are relevant for business/desktop apps
 - Multiple processes with complex systems or SPMD based parallel applications
- Spread the workload across multiple cores
 - OS maps threads/processes to cores
 - Transparent to the programmer
- Write parallel algorithms
 - True for scientific/engineering applications
 - Programmer needs to define the mapping to cores

Agenda

- Motivation
- Parallel architectures
- **Cache coherency and consistency**
- Interconnecting the cores
- Multithreading
- Concluding remarks

Caches

Memory can be fast or vast !!

Gap between Memory and Processor grows ..



Caches come to the rescue ☺

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

25



Caches for uniprocessors and CMPs

- For single core systems caches are nice:
 - Reduce average access time
 - Bigger → performance
 - Generally power efficient
- For multi-core still nice, but shared memory requires:
 - Cache Coherency
 - Memory Consistency

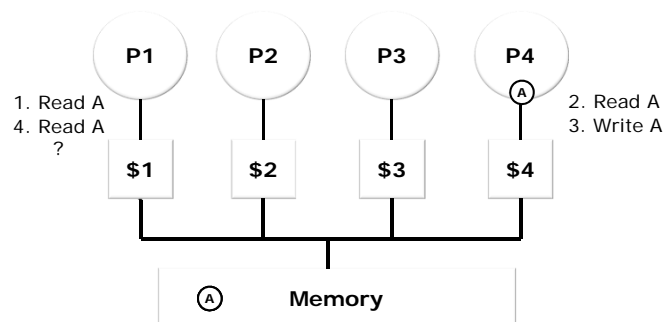
Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

26



Cache coherency (example)



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

27



Cache coherency protocols

Snoopy protocols

- Data requests are sent to all processors
- Processors “snoop” bus constantly
 - If they have a copy, they act accordingly
 - Invalidate or update own content
- Requires broadcast, since caching information is at processors
 - Perfect for global buses → But, does not scale well
- Many snoopy algorithms: MSI, MESI, MESIF, MOESI, etc.

Ayosé Falcón
Intel Barcelona Research Center

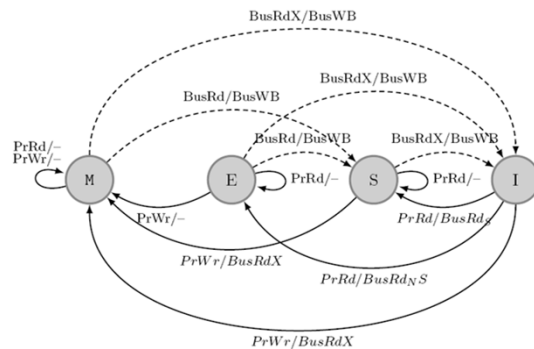
Multi-core Processor Architectures
Seminari d'Empresa 2013

28



Cache coherency protocols

Example: MESI protocol



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

29



Cache coherency protocols

Directory-based protocols

- Common (logically) table that stores data ownership
 - May be physically distributed
- Invalidation/Update sent point-to-point
- Scale much better than bus-based
 - But, can become a performance bottleneck as well for many cores (unless distributed)

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

30



Memory consistency: the dark side of shared memory

- Coherency guarantees that writes are visible to all the processors
- But, in which order do they appear?
 - Affects ease-of-programming and performance

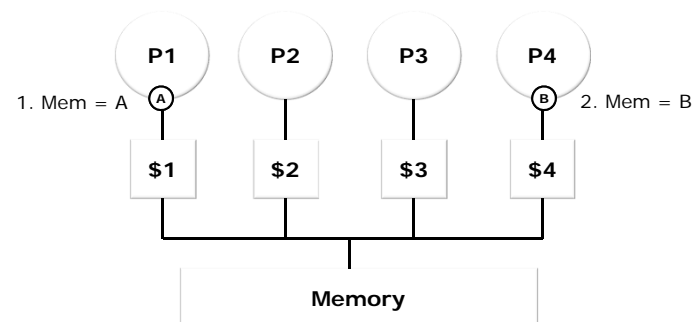
Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

31



Memory consistency (example)



Case 1

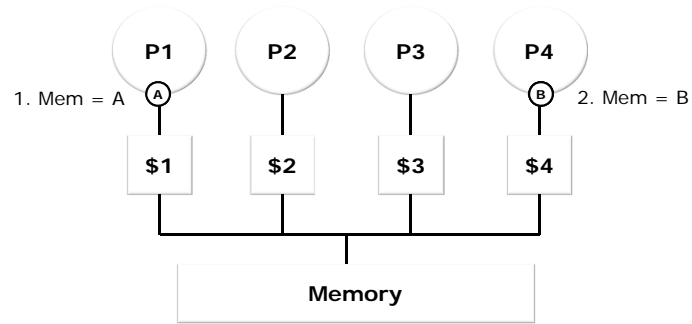
Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

32



Memory consistency (example)



Case 2

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

33



Memory consistency model

- Specifies constraints on the order in which memory operations (from any process) can appear to execute with respect to one another
- Contract between programmer and system
- Implications for both programmer and system designer
 - Programmer uses to reason about correctness and possible results
 - System designer can use to constrain how much accesses can be reordered by compiler or hardware

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

34



Agenda

- Motivation
- Parallel architectures
- Cache coherency and consistency
- **Interconnecting the cores**
- Multithreading
- Concluding remarks

Ayosé Falcón
Intel Barcelona Research Center

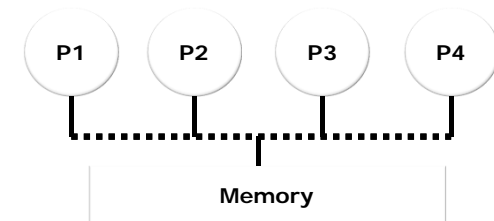
Multi-core Processor Architectures
Seminari d'Empresa 2013

35



Simple bus-based systems

- Fairly simple collection of "wires"
- Common medium that everyone can *snoop*
- Common for current CMPs
- Does not scale ☹



Ayosé Falcón
Intel Barcelona Research Center

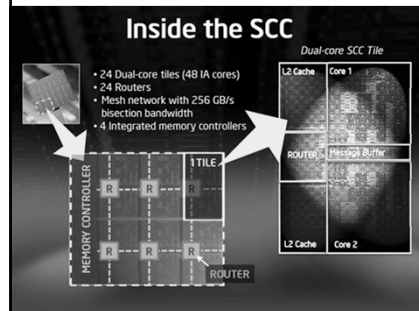
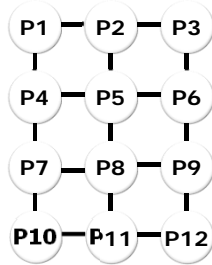
Multi-core Processor Architectures
Seminari d'Empresa 2013

36



Mesh networks

- Each core functions as a router
- Longer latencies than bus
 - Multiple hops
- More suitable for many-core



2. Intel's experimental system uses a 2D mesh to connect 48 independent x86 processor cores. The software cache coherency simplifies the design of the system interconnect.

37



Networks on chip

- Emerging design paradigm for interconnection within the chip
- Based on techniques used widely in networking
 - More suitable for large-scale CMPs
- NoCs provide:
 - QoS
 - Separability between computation and communication
 - Reusability

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminariis d'Empresa 2013

38



Agenda

- Motivation
- Parallel architectures
- Cache coherency and consistency
- Interconnecting the cores
- **Multithreading**
- Concluding remarks

Ayosé Falcón
Intel Barcelona Research Center

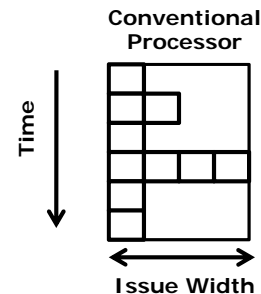
Multi-core Processor Architectures
Seminariis d'Empresa 2013

39



Multithreading within a core

Cores usually underutilized ...



Ayosé Falcón
Intel Barcelona Research Center

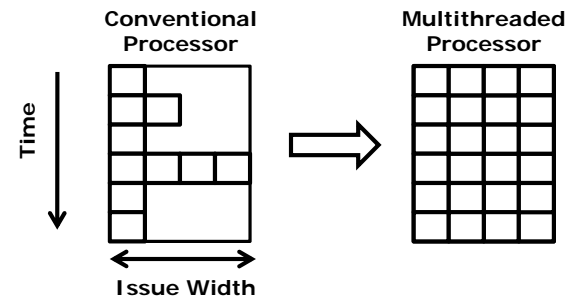
Multi-core Processor Architectures
Seminariis d'Empresa 2013

40



Simultaneous multithreading (SMT)

Fill it app with instructions from other threads



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

41



Speculative multithreading

- Speculatively parallelize a serial application
 - Use speculation to overcome ambiguous dependences
 - Hardware support to recover from mis-speculations
- Different implementations
 - Thread Level Speculation (extract parallelism)
 - Helper threads (prefetch to memory)
 - Multi-path (follow all paths on a branch)
 - Etc ..

Ayosé Falcón
Intel Barcelona Research Center

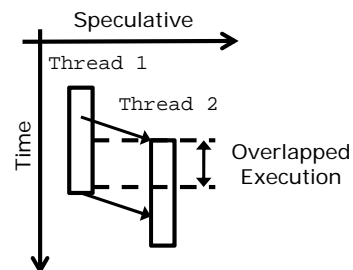
Multi-core Processor Architectures
Seminari d'Empresa 2013

42

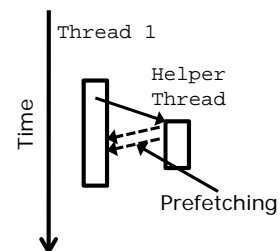


Examples

Thread-level Speculation



Helper Threads



Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

43



Agenda

- Motivation
- Parallel architectures
- Cache coherency and consistency
- Interconnecting the cores
- Multithreading
- **Concluding remarks**

Ayosé Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

44



Summary—Conclusions

- Parallel computing is here to stay
- Today we discussed about:
 - Why we need parallelism
 - What are the types of parallel architectures
 - How we keep memory and caches coherent and consistent
 - Multi-core interconnects
 - Multithreading
- Top of the iceberg: Loads of exciting research happening!!

Ayose Falcón
Intel Barcelona Research Center

Multi-core Processor Architectures
Seminari d'Empresa 2013

45



Questions?

ayose.falcon@intel.com

Ayose Falcón
Intel Barcelona Research Center

Multicore Processor Architectures
Aula Empresa 2012



Multi-core Processor Architectures

Ayose Falcón
Senior Research Scientist
Intel Barcelona Research Center, Intel Labs



Seminari d'Empresa 2013
Facultat d'Informàtica de Barcelona, Jan/Feb 2013



Parallel Programming

Josep M. Codina

Intel Barcelona Research Center, Intel Labs

Aula Empresa, Facultat d'Informàtica de Barcelona, 2013

© Intel Corporation, 2013

Parallel Programming?

I want to divide the effort of eating pizza to eat all slices as fast as possible!!

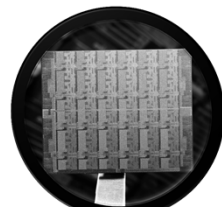
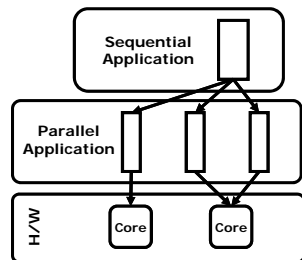


But, I wonder...

How we split this hard task ☺

Why Parallel Programming?

- Applications are naturally parallel
- Multi-core are out there
- We need to think in parallel!!!!



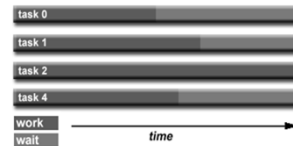
48- IA cores in the Intel "Single-chip Cloud Computer"

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Models
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

Design of Parallel Applications

- Understand the Problem
 - Identify hotspots
 - Profile application
 - Identify bottlenecks to the parallelization
 - Communication
 - Synchronization
 - Consider alternative versions of the application
- Split Application in Parts
 - Maximum parallelism
 - Minimum imbalance
 - Minimum waiting time

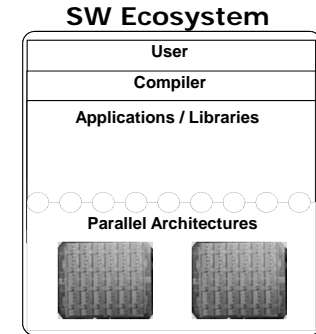


5

Designing Tomorrow's Microprocessors

Who Creates Parallel Applications?

- User/ Programmer
 - Parallel languages and libraries
- Compilers
 - Automatic Parallelization
- Parallel Architectures
 - Multi-core in a single chip
 - Many multi-core chips



6

Designing Tomorrow's Microprocessors

What Do We Need?

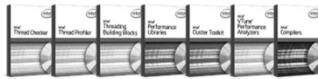
Parallel Programming Languages



Intel Ct

Provide Tools Today

Architect, Thread, Debug & Tune



www.intel.com/software/products

Research Tomorrow's Techniques



Transactional Memory



Speculative Multi-threading



Data Parallel Languages



Educate Tomorrow's Experts



University Outreach
Intel @ Press
Intel® Software College

- Helped 45 universities add parallel programming courses
- 7500 students took them
- 2007 Goal: 400+ universities



7

Designing Tomorrow's Microprocessors

When to Parallelize an Application?

- **Allways!!!!**
- Stop thinking in sequential applications
- Multicore era is here and we have to leverage the computing capabilities that we have out there

8

Designing Tomorrow's Microprocessors

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Models
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

9

Designing Tomorrow's Microprocessors

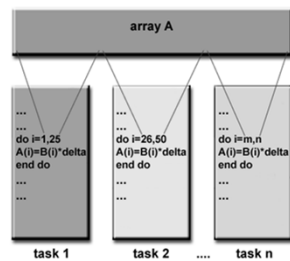
Parallel Programming Models

- Data Parallel
- Message Passing
- Shared Memory
- Distributed Shared Memory Model
- NOTE: These models are orthogonal to the actual hardware!!!

10

Designing Tomorrow's Microprocessors

Data Parallel Model



- Set of cooperating tasks on the same data structure
- Tasks perform the same operation over on different data items

Advantage

- Easy to write and comprehend
- No synchronization

Disadvantage

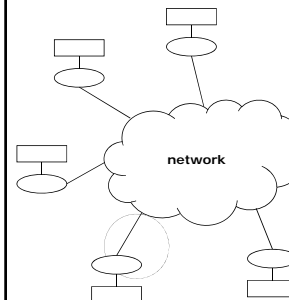
- No independent code

Example: HPF (High Performance Fortran)

11

Designing Tomorrow's Microprocessors

Message Passing Model



Address space
Process

- A set of cooperating sequential processes
- Each with own local address space
- Processes interact with explicit transaction (send, receive,...)

Advantage

- Programmer controls data and work distribution

Disadvantage

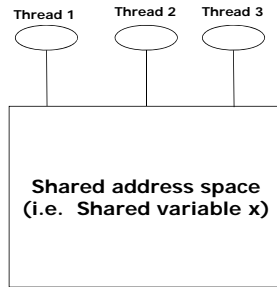
- Communication overhead for small transactions
- Hard to program!

Example: MPI

12

Designing Tomorrow's Microprocessors

Shared Memory Model



- Different simultaneous execution threads (processes)
- Read / Write to one shared memory space and invalidate if necessary

Advantage

- Read remote memory via an expression
- Write remote memory through assignment

Disadvantage

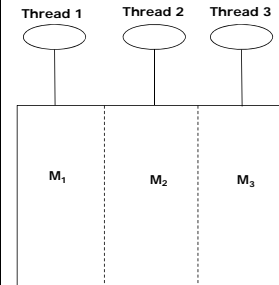
- Manipulating shared data leads to synchronization requirements
- Does not allow locality exploitation

Example : OpenMP

13

Designing Tomorrow's Microprocessors

Distributed Shared Memory Model



Partitioned shared address space

(with each partition having affinity to corresponding thread)

- Similar to the shared memory paradigm
- Memory M_i has *affinity* to Thread i
- At the same time each thread has global view of memory

Advantage

- Helps exploiting locality of references
- Simple statements as SM

Disadvantage

- Synchronization still necessary

Example: UPC, Titanium, Co-Array, Global Arrays

14

Designing Tomorrow's Microprocessors

Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Languages
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

15

Designing Tomorrow's Microprocessors

Traditional Auto Parallelization

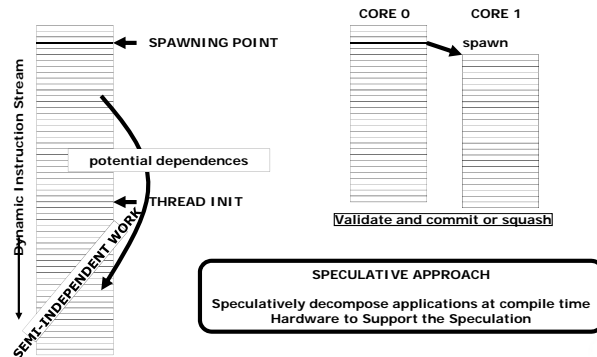
- Automatic decomposition of applications into threads
- No need for inserting directives or pragmas
- Compiler identifies suitable parts of the application for parallelization
 - Typically simple loops
 - Considering simple memory disambiguation schemes
- This approach is typically limited to simple applications
 - Dependences limit its applicability to large scale applications

16

Designing Tomorrow's Microprocessors

Speculative Multithreading

- Automatic decomposition of applications into speculative threads
 - No need to be conservative on the memory disambiguation



17

Designing Tomorrow's Microprocessors

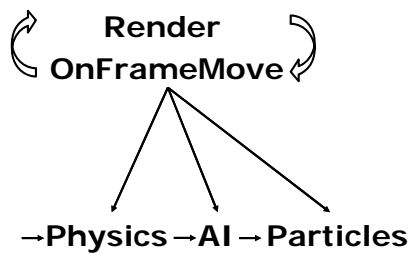
Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Languages
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

18

Designing Tomorrow's Microprocessors

Usual Game Structure



Comparative Analysis of Game Parallelization, Dmitry Eremin, GDC'08

19

Designing Tomorrow's Microprocessors

Usual Game Structure



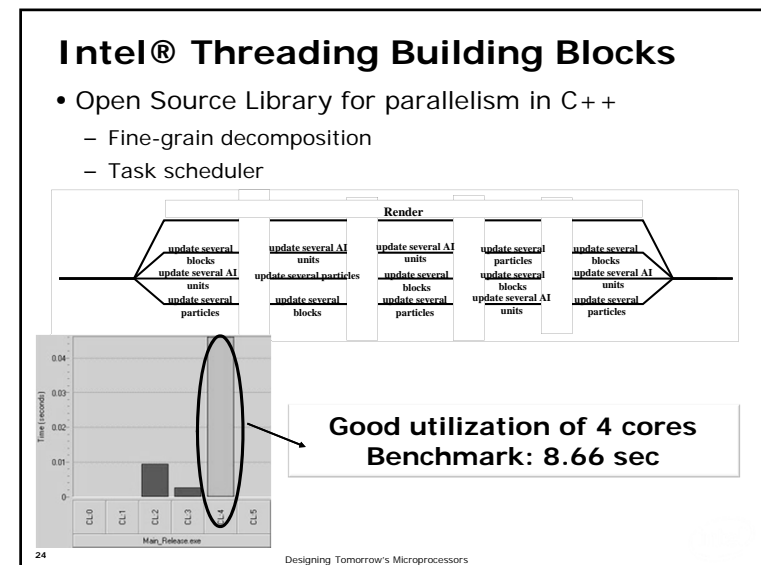
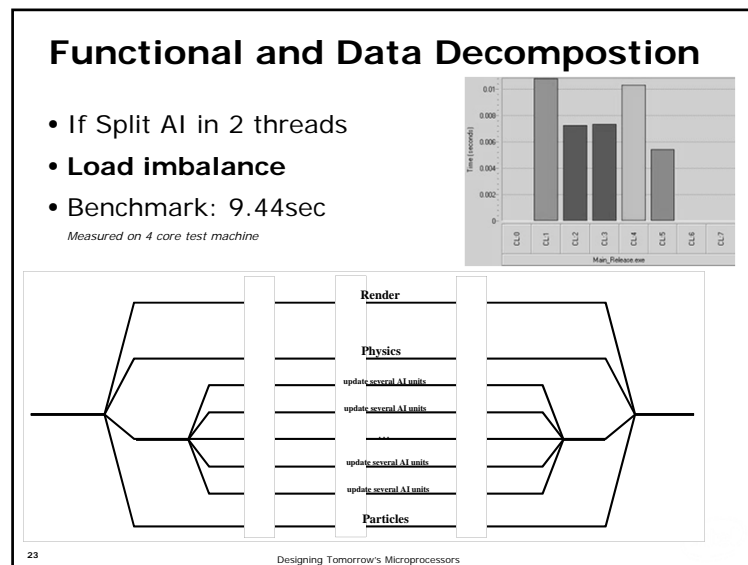
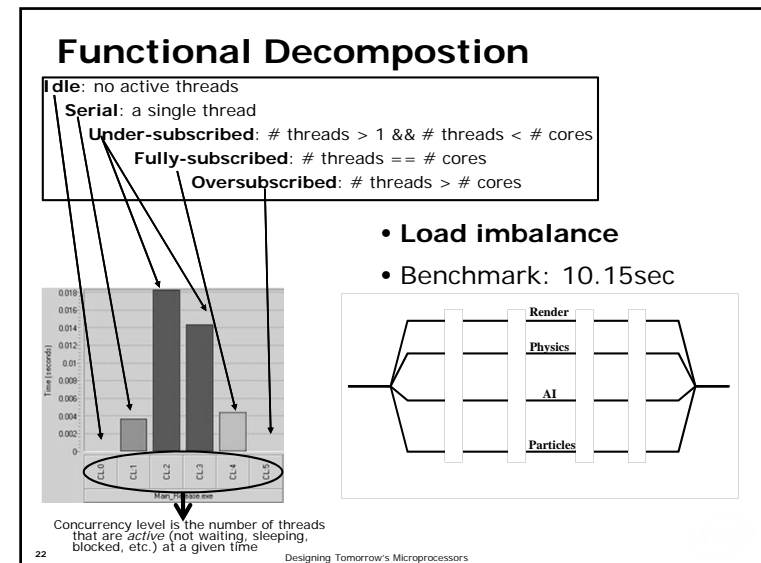
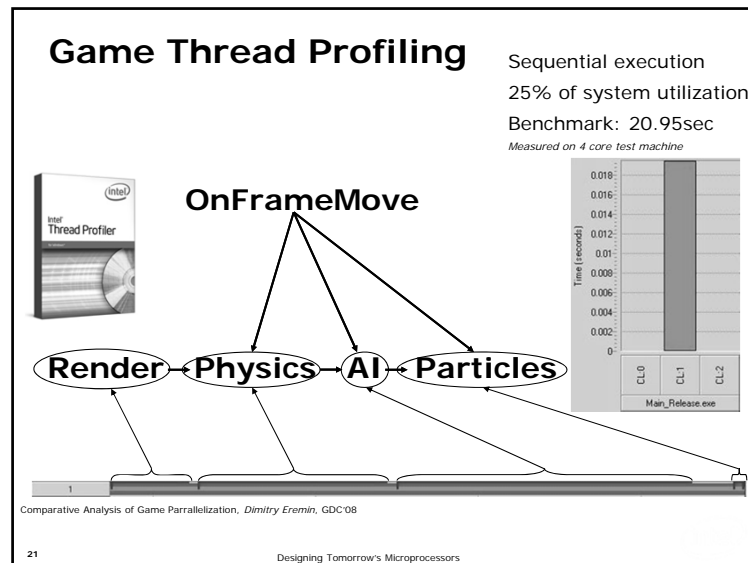
Render → Physics → AI → Particles

<http://softwarecommunity.intel.com/articles/eng/1363.htm>

Comparative Analysis of Game Parallelization, Dmitry Eremin, GDC'08

20

Designing Tomorrow's Microprocessors



Agenda

- Motivation
- Designing Parallel Applications
- Parallel Programming Languages
- Automatic Parallelization
- Parallel Programming for Games
- Concluding Remarks

25

Designing Tomorrow's Microprocessors

Summary - Conclusions

- Parallel Applications are required to leverage Parallel Architectures
- Today we discussed about:
 - Why we need parallel applications
 - Considerations When Designing parallel applications
 - Parallel Programming Models
 - Auto Parallelization and Speculative Multithreading
 - The importance of threading in gaming
- We need to think in parallel to create parallel applications!!!

26

Designing Tomorrow's Microprocessors

Parallel Programming

Josep M. Codina

Intel Barcelona Research Center, Intel Labs

Aula Empresa, Facultat d'Informàtica de Barcelona, 2013

© Intel Corporation, 2013

Designing Tomorrow's Microprocessors

Tanausu Ramírez and Enric Herrero

Intel Barcelona Research Center

Aula Empresa, Facultat d'Informàtica de Barcelona, January 2013

© Intel Corporation, 2013

Course Agenda

- Overview of Today's Microprocessors and Future Trends
- Microarchitecture of Current Microprocessors
- Design Cycle for Microprocessors
- Methodology for Research in Microprocessors
- Profiling and Performance Evaluation
- Multi-core Processor Architectures
- Parallel Programming
- **Reliability**
- Hardware/Software Co-designed Microprocessors

2

Designing Tomorrow's Microprocessors

Acknowledgment

- **Raimat/TRAMS Project**
 - Xavi Vera
 - Nicholas Axelos, Javier Carretero, Daniel Sánchez, Matteo Monchiero
- **Phoenix Project**
 - Jaume Abella, Pedro Chaparro, Osman Unsal, Oguz Ergin
- **Intel**
 - J. Tschanz, S. Mitra, S. Iacobovici, K. Bowman, C. Wilkerson, and many others!

3

Designing Tomorrow's Microprocessors

• Reliability Challenges

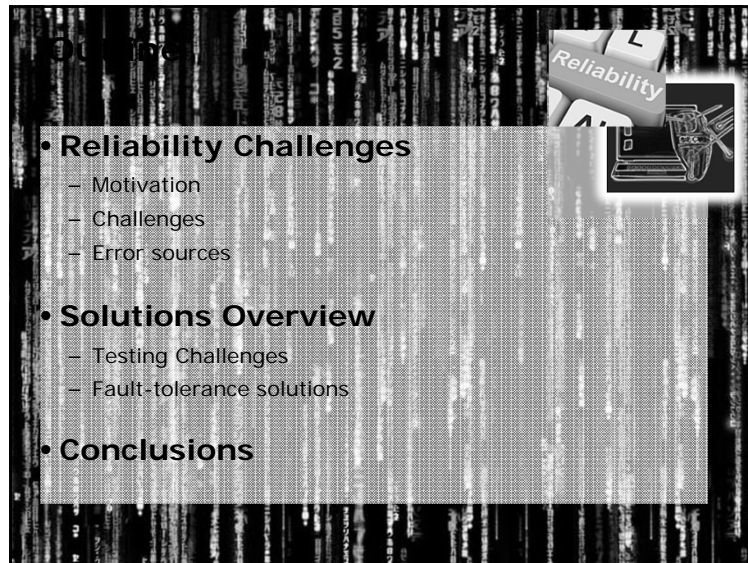
- Motivation
- Challenges
- Error sources

• Solutions Overview

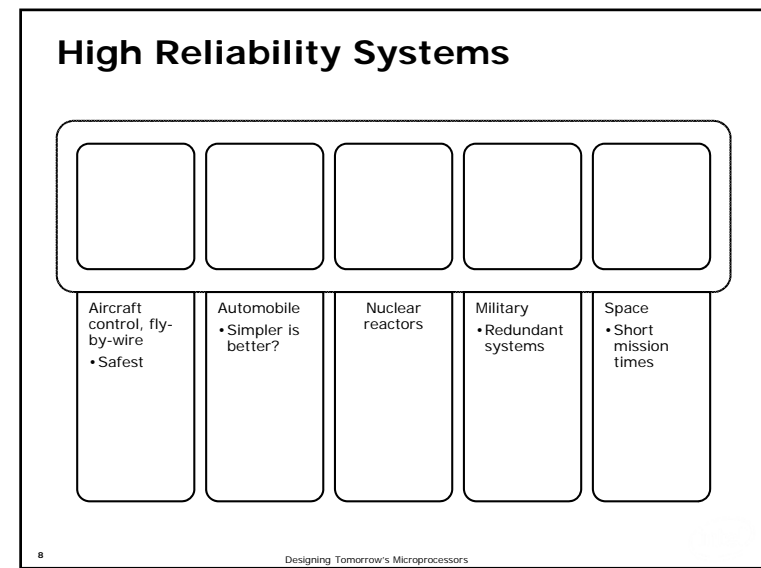
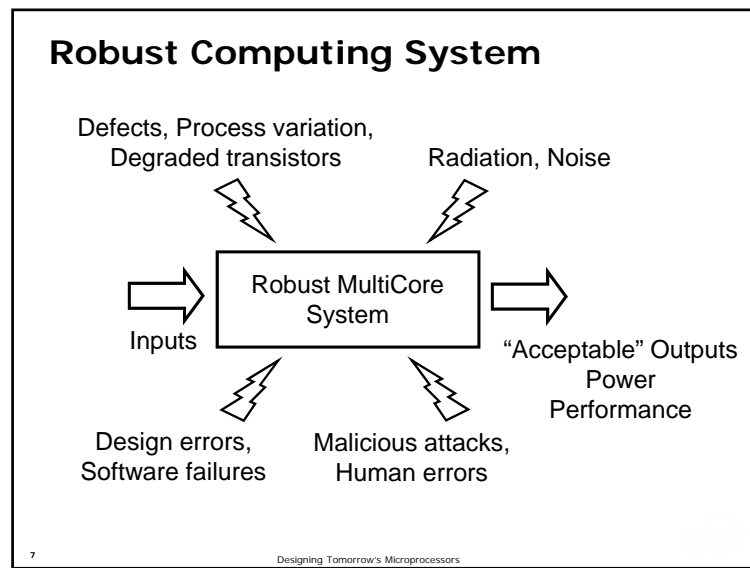
- Testing Challenges
- Fault-tolerance solutions

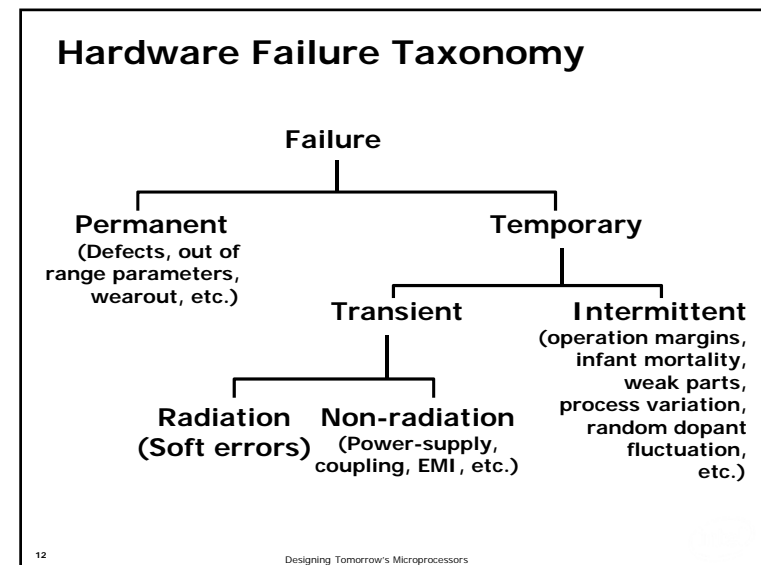
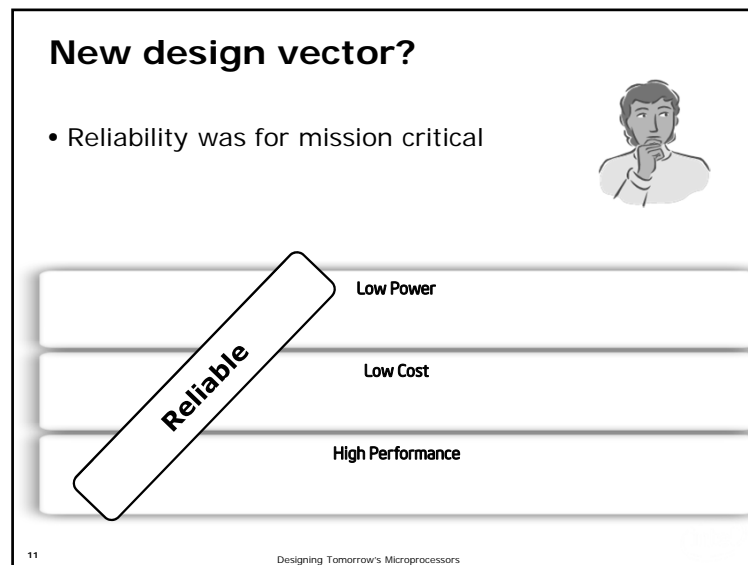
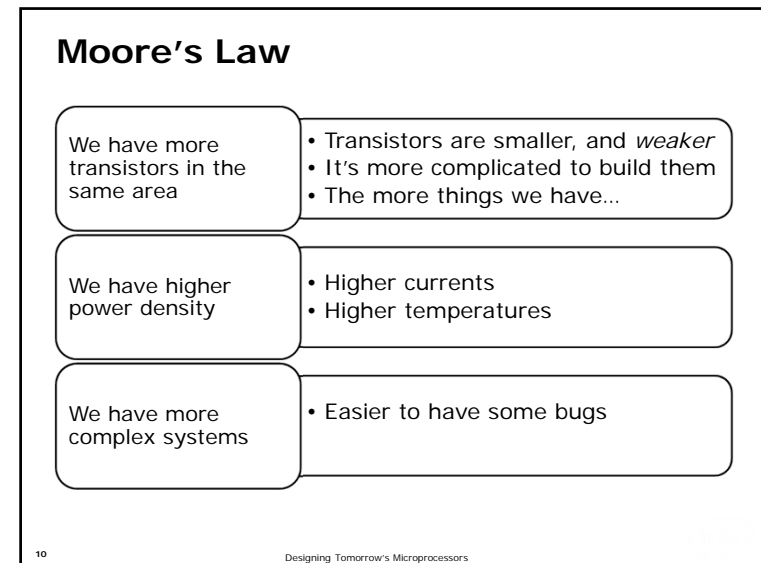
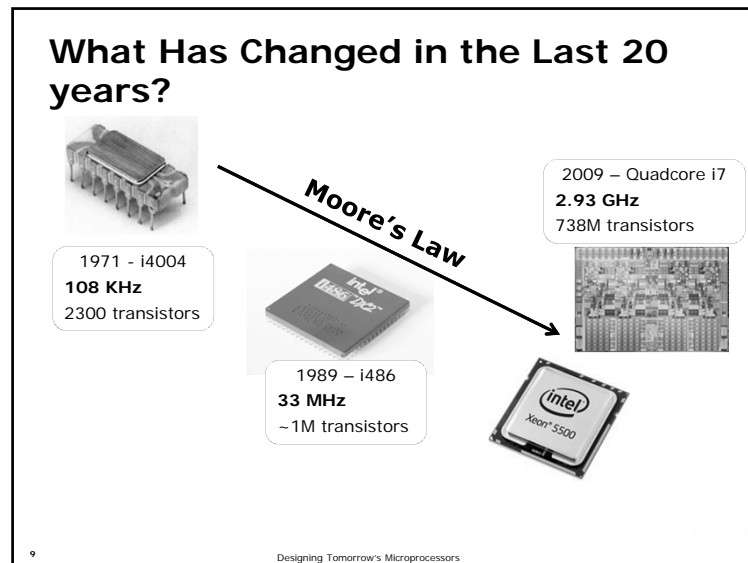
• Conclusions



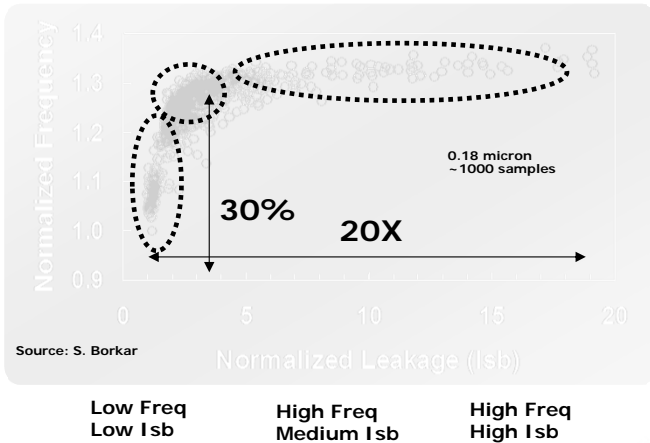


- **Reliability Challenges**
 - Motivation
 - Challenges
 - Error sources
- **Solutions Overview**
 - Testing Challenges
 - Fault-tolerance solutions
- **Conclusions**





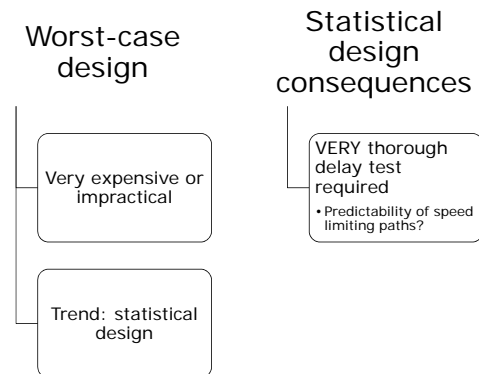
Process Variations



13

Designing Tomorrow's Microprocessors

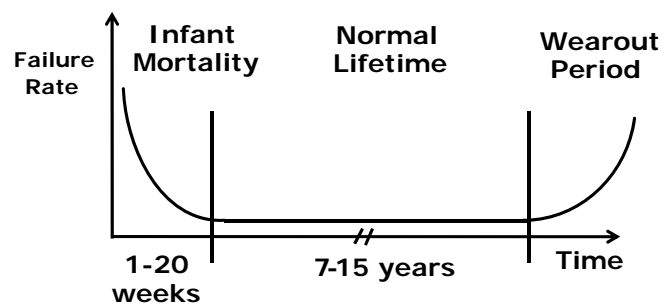
Process Variations (i2)



14

Designing Tomorrow's Microprocessors

Bathtub Curve



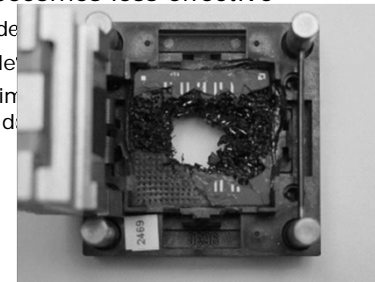
15

Designing Tomorrow's Microprocessors

Infant Mortality

• Burn-in becomes less effective

- Latent defects
- Higher leakage
- Power limits remain

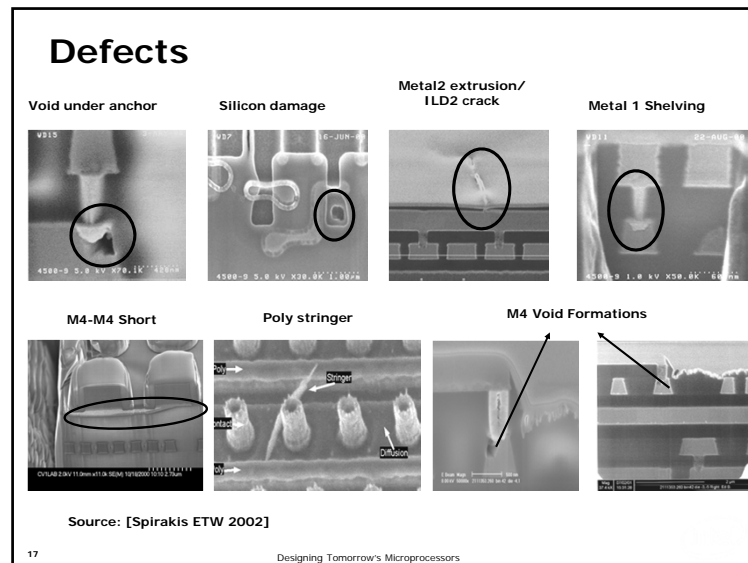


timing errors
problem
by defects

Next Generation Burn-in & Test System for Athlon Microprocessors : Hybrid Burn-in, Mark Miller, Burn-in & Test Socket Workshop, 2001

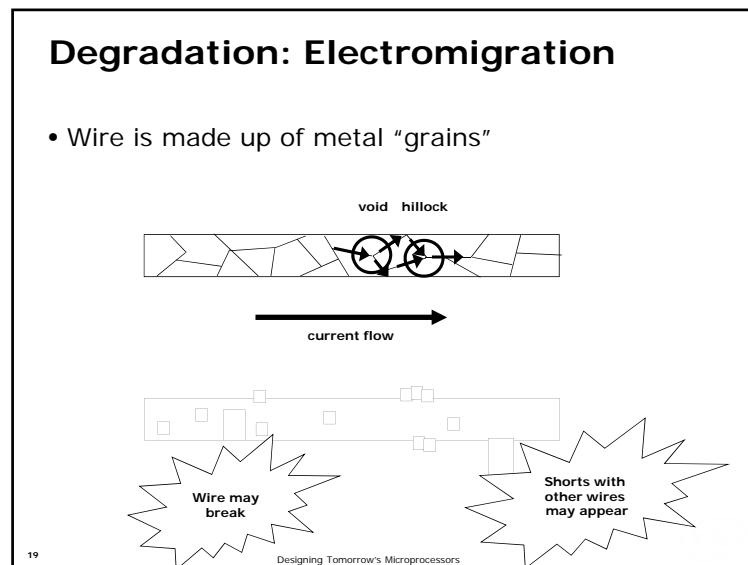
16

Designing Tomorrow's Microprocessors



Degradation

	Affects	Weakest	Worst input
Electromigration	Connections	Longest	"1"
Stress migration	Connections	Longest	Any
Time-dependent dielectric breakdown (TDDB)	Gate oxide	Widest	"0" PMOS "1" NMOS
Negative Bias Temperature Instability (NBTI)	Gate oxide PMOS	Widest	"0"
Large thermal cycling	Package	---	---
Short thermal cycling	Unknown	---	---

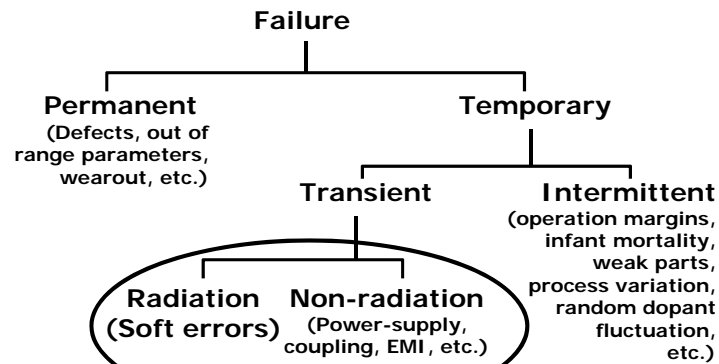


Degradation: Current practice

- Speed guardbands
 - Very expensive
- Summary
 - If we add temperature variations we open the door to...

OVERCLOCKING!

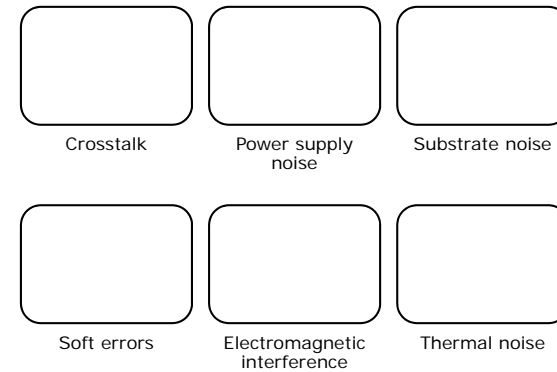
Hardware Failure Taxonomy



21

Designing Tomorrow's Microprocessors

Source of Noise

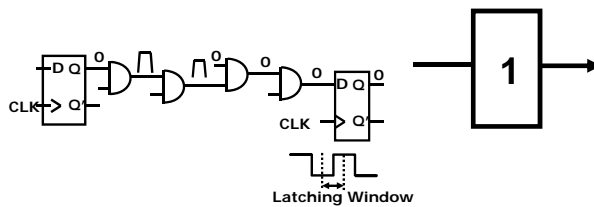
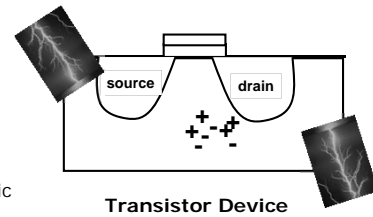


22

Designing Tomorrow's Microprocessors

Soft Errors

- Neutrons strike on Si device
- Alpha particles from packaging
- They impact latches
- They impact combinational logic



23

Designing Tomorrow's Microprocessors

Soft Errors (2)

- Soft Errors can cause problems in different ways
 - Change the data value in the Caches and Memory
 - Corrupt the execution of instruction due the flip of data in the pipeline registers.
 - Change the character of a SRAM-Based FPGA circuit. (Firm Error)
 - Datapath logic SET (Single Event Transient) caught by registers/memory

24

Designing Tomorrow's Microprocessors

Soft Errors (3): Evidence

- Error logs of large servers
 - Normand, IEEE T. Nuclear Science, Dec. 1996.
- Sun Microsystems, 2000 (from Baumann, IRPS 2002)
 - Cosmic ray strikes on L2 cache
 - Mysterious crashes of Sun flagship servers
 - Companies affected
 - Baby Bell (Atlanta), America Online, Ebay, & dozens others
 - Verisign moved to IBM Unix servers (for the most part)



25

Designing Tomorrow's Microprocessors

Soft Errors (i4): Doomsday



Altitude of 30,000 feet on a route crossing the north pole both cause increase in neutron flux.

Four 1M 130nm SRAM-based FPGAs, it would be subject to 0.074 upsets per day = 324 hours between upsets.

Assume one such system on-board each commercial aircraft, 4,000 civilian flights per day, 3 hours average flight time.

Nearly 37 aircraft will experience a neutron-induced SRAM-based FPGA configuration failure during the duration of their flight.

26

Designing Tomorrow's Microprocessors

Summary: VLSI Trends & Reliability

Reliability Problems	VLSI Trends
Power supply, Signal integrity problems	High speed, low voltage, large current
Process variation (die-to-die, intra-die)	Many transistors, nano-fabrication, high speed
Manufacturing defects	Many transistors, new material, burn-in issues
Degradation over time	Large current, increased electric field, new material, thin oxide, stress void, High-K
SEUs due to radiation	Many transistors, low-voltage, high speed

27

Designing Tomorrow's Microprocessors

• Reliability Challenges

- Motivation
- Challenges
- Error sources

• Solutions Overview

- Testing Challenges
- Fault-tolerance solutions

• Conclusions



General Solutions for Faults

- Design systems that minimize the presence of faults

• Fault avoidance / removal

- Rigorous design and verification: Design a system with minimal faults
- Comprehensive testing: Validate/test a system to remove the presence of faults

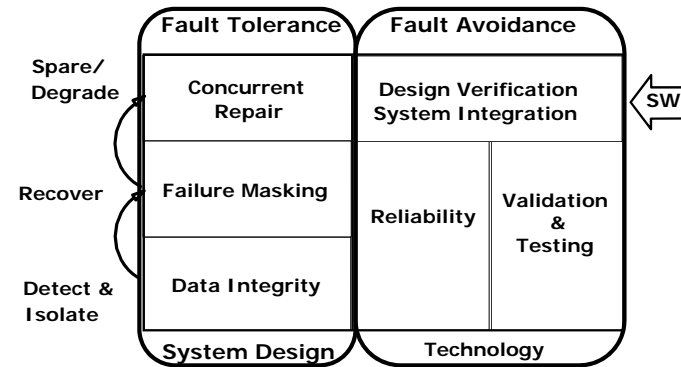
• Fault tolerance

- Living and deal with faults!
- Built-in error detection
 - eg. Redundancy

29

Designing Tomorrow's Microprocessors

High Availability Building Blocks



30

Designing Tomorrow's Microprocessors

Fault avoidance → Testing

Validation

- Find out if the processor does what it has to do (functional, compatibility, etc)

Testing

- One mechanism/process to validate the system

Coverage

- How many errors we catch...
- % of the area covered
- % of the transistors covered
- % of inputs covered

31

Designing Tomorrow's Microprocessors

Testing Levels

Application level:

- Programs are run and their outputs verified. Easy to do, but the coverage is very low

Functional level:

- Each individual block is tested with different inputs. Coverage increases, but many timing/state related faults and block interactions are missed

Structural level:

- Inputs are injected to force failures to show up in the different nodes of the block. The coverage is much higher, but it is complex producing proper inputs for high coverage

32

Designing Tomorrow's Microprocessors

Testing Challenges

- Technology issues/integration consequences
 - Testing time (vs. time to market)
 - Yield vs Debug
 - Low Operating voltages (V_{cc}): decrease margins, more noise
- “New” on-die complex structures
 - Interaction of multiple cores
 - Interconnection network, validate the coherence protocol
 - DVS/DVFS

33

Designing Tomorrow's Microprocessors

Testing Cost

- Circuit issues becoming much larger fraction of overall Post-Si bugs
 - Functional issues losing weight
- Test costs sharply up from generation to generation
 - Regular pattern quantity goes down every generation to keep costs
 - More corner cases
 - More complexity to test
- Circuit bugs take much longer to root
 - Long latency between failure and syndrome

34

Designing Tomorrow's Microprocessors

Design For Testing (DFT)

- DFT consists in designing with “testability features” to speed testing and reduce costs
 - Increasing the observability and controllability of circuits
 - Putting hardware in place to allow testing
 - e.g. scan chain, test pins, Built-In Self Test (BIST), etc.
 - Space & time redundancy
 - Self-checking blocks: some properties of the outputs are checked for correctness (e.g. residue, parity, ECC)
 - Full hardware replication: operations are repeated in replicated hardware
 - Outputs are latched twice at different times to detect timing errors

35

Designing Tomorrow's Microprocessors

Fault-tolerant design

- Providing fault-tolerant design for every component is normally not an option

- **How critical is the component?**
- **How likely is the component to fail?**
- **How expensive is it to make the component fault-tolerant?**

fault-tolerant *f(ɔːl-təl(-ə)-rənt/*
adj : able to function in the
 absence of a major component



36

Designing Tomorrow's Microprocessors

Error Detection

- Most important factor because a processor cannot tolerate a problem which is not aware
- Key to error detection is redundancy
 - without redundancy a processor cannot detect any errors.

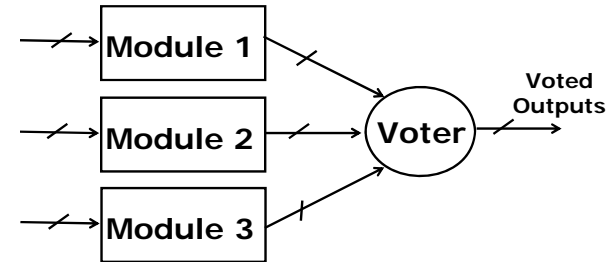
Type of Redundancy	Basic Idea	Example
Physical/spatial	Add redundant hardware	Replicate modules and have replicas to compare the results
Temporal	Perform redundant operations	Run a program twice on the same HW and compare the results
Information	Add redundant bits	Add a parity bit to a word in memory

37

Designing Tomorrow's Microprocessors

Physical redundancy

- No error on outputs → failure masking
 - Triple Modular Redundancy (TMR)
 - N-Modular Redundancy (NMR)



38

Designing Tomorrow's Microprocessors

Temporal Replication

RMT: Redundant threads execute at the same time to check for errors

Hardware
implementation
on SMT

Multi-core based RMT using fingerprint

Compare
signatures of
instruction
sequences only

Fewer
compares are
needed ->
reduce thread
communication

Optimized
versions:
selective/partial
replication

39

Designing Tomorrow's Microprocessors

Information Redundancy

- Coding Like – Error detecting codes (EDC)
 - add redundant bits to a datum to detect when it has been affected by an error
- Parity
 - Simple and cheap solution (dataword + parity bit)
- ECC
 - More sophisticated and can also correct errors (SECDEC)
 - Used in large caches and memory
- Extended protection against multiple upsets
 - Interleaving (spatial), scrubbing (temporal)

40

Designing Tomorrow's Microprocessors

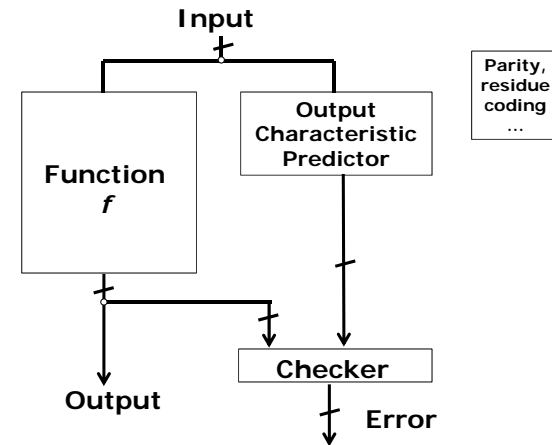
Coding Like Logic – FUs - ALUs

- Residue/arithmetic codes
 - Uses residues with coverage close to the one achieved by FUs replication at a fraction of the area & power
- Mod D residue of number N: remainder of N divided by D
 - For arithmetic ops: $X \text{ op } Y = Z \rightarrow (X \text{ op } Y) \bmod D = ((X \bmod D) \text{ op } (Y \bmod D)) \bmod D$
- Example:
 - $X = 46238$; $46238 \bmod 3 = 2$
 - $Y = 56788$; $56788 \bmod 3 = 1$
 - $X + Y = 103206$; $103206 \bmod 3 = 0$ ($2 + 1 \bmod 3$ is 0!)

41

Designing Tomorrow's Microprocessors

Concurrent Error Detection (CED)



42

Designing Tomorrow's Microprocessors

Software Assisted Error Detection

- Software-implemented hardware fault-tolerance
 - It introduces the redundant operations and checks/assertions
 - Hardened values and subroutines
 - Programmer may choose to check only critical code
- EDDI, SWIFT approaches

<pre>ld r12=[GLOBAL] add r11=r12,r13 st m[r11]=r12</pre> <p>(a) Original Code</p>	<pre>ld r12=[GLOBAL] 1: ld r22=[GLOBAL+offset] add r11=r12,r13 2: add r21=r22,r23 3: cmp.neq.unc p1,p0=r11,r21 4: cmp.neq.or p1,p0=r12,r22 5: (p1) br faultDetected st m[r11]=r12 6: st m[r21+offset]=r22</pre> <p>(b) Duplicated Instr. Code</p>
---	---

43

Designing Tomorrow's Microprocessors

Reliability Challenges

- Motivation
- Challenges
- Error sources

Solutions Overview

- Testing Challenges
- Fault-tolerance solutions

Conclusions

Fault Tolerant System

Post-SI Validation

Detects Degradation Errors

Defects/Bugs/Hard Faults

Soft Errors

Pre-SI Validation, Functional Validation

Quad-Use Technology

Core-Independent

Online/Offline Mode

Combine with SW/HW approaches

45

Designing Tomorrow's Microprocessors

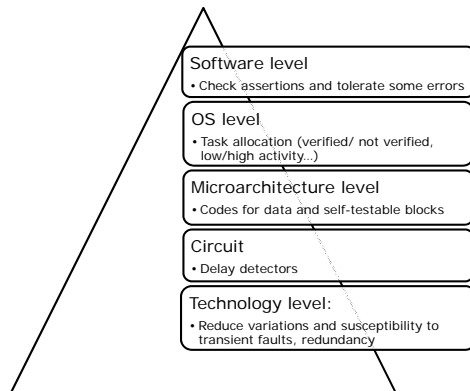
Multi-layer Approach

- A lot has been said about solutions that combine circuit, parch, and SW approaches
 - Few to none seen
 - Each layer tries to do its best... probably paying a high price
- We need to design solutions bottom-up *considering* all different layers
 - Clearly identify error detection and recovery requirements for each level
 - Each layer contributes with its own detection and recovery capabilities
 - Co-design CKT/HW/SW solutions!

46

Designing Tomorrow's Microprocessors

Error Detection levels



47

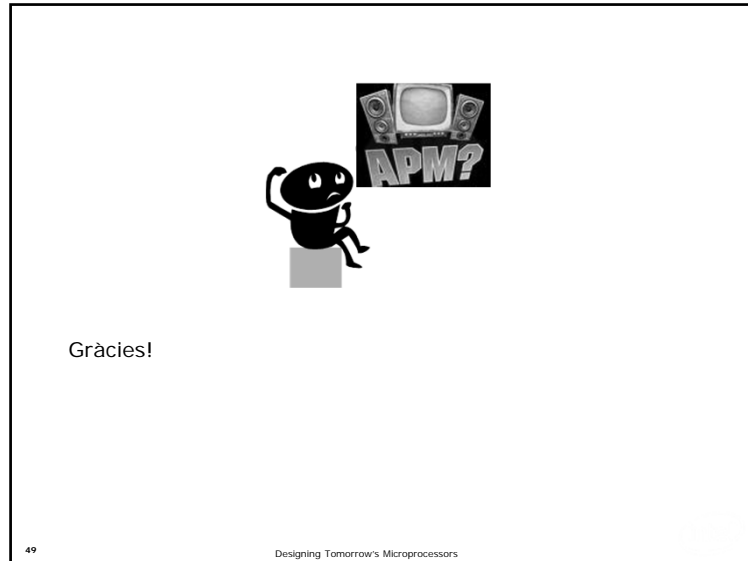
Designing Tomorrow's Microprocessors

Adapt the System

- Reconfiguration
 - Transistor level (manufacturing time)
 - Spare transistors
 - Circuit level
 - Spare cache lines
 - Microarchitecture
 - Memory, processor, routers, networks
 - Software
 - Components that can be virtualized (e.g., bypass levels in inorder cores)
 - OS
 - Task allocation (heterogeneity)

48

Designing Tomorrow's Microprocessors





Hardware / Software co-designed processors

Kyriakos Stavrou

Jan 31st, 2013

1


What is this presentation about

- CPU design today faces important challenges
 - Power, performance, complexity, validation, cost, ...
- This presentation is about a radical approach

Co-designed CPUs
CPU = HW + SW

2

Designing Tomorrow's Microprocessors




Agenda

- Overview of the technology
 - What are the HW/SW co-designed processors
- Key Ideas and Advantages
 - From the HW-only CPU to the co-design paradigm
- Research Projects / Market examples
 - Academic Research
 - Products
- Potential and Open Issues
 - A glance to the huge potentials

3

Designing Tomorrow's Microprocessors



HW/SW co-designed processors

- Traditionally CPUs have been HW only

HW

- Alternative: co-designed processors
 - CPU = HW + SW

HW

SW

App 1

App 2

App 3


Operating System

CPU

– The system is unaware of the CPU internals

4

Designing Tomorrow's Microprocessors



HW/SW co-designed processors

- HW/SW co-design targets the major challenges

- Power Consumption
 - Through using simpler Hardware
 - Through using less Hardware
 - By optimizing code once / using many times
- Design Complexity
 - Co-designed processors have simpler HW
 - Simpler HW is much easier to validate
- Performance
 - Synergy between the HW and SW
 - Exploit dynamic information

5

Designing Tomorrow's Microprocessors



Agenda

- Overview of the technology
 - What are the HW/SW co-designed processors
- Key Ideas and Advantages
 - From the HW-only CPU to the co-design paradigm
- Research Projects / Market examples
 - Academic Research
 - Products
- Potential and Open Issues
 - A glance to the huge improvement potential

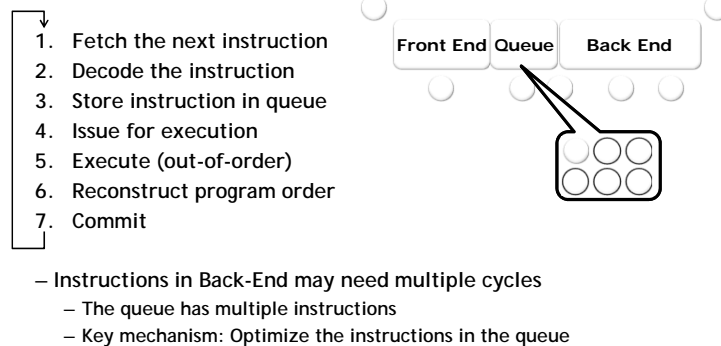
6

Designing Tomorrow's Microprocessors



The key idea

- How traditional processors work



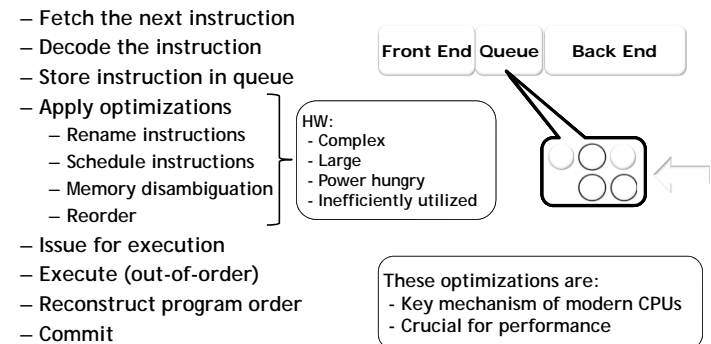
7

Designing Tomorrow's Microprocessors



The key idea

- How traditional processors work



8

Designing Tomorrow's Microprocessors



The key idea

- In a traditional design
 - There must be different HW for different optimizations
 - This HW needs to be designed and validated
 - The cost to optimize the same code is paid multiple times
 - The hardware is very aggressive
 - Consumes a lot of power
 - The hardware tries to optimize always
 - Optimizations do not always pay-off (power / performance)
 - HW exploits limited information (limited #instructions in the queue)
- Non-efficient resource utilization
 - Area increase
 - Validation cost
 - Power consumption increase
 - Pay the optimization cost multiple times

9

Designing Tomorrow's Microprocessors



The key idea

- Golden rule: *“90% of execution time to 10% of the code”*
i.e. Most execution time is spent in a small portion of the application

Example: *Matrix Multiply code*

```
read array A;
read array B;

for (i=0 .. n)
  for (j=0 .. n)
    for (k=0 .. n)
      C[i][j] += A[i][k] * B[k][j];

print array C;
```

These instructions dominate the execution time

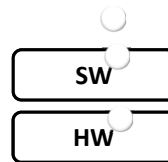
10

Designing Tomorrow's Microprocessors



The key idea

- Co-designed Processors
 - The instruction stream passes through the SW
 - The SW observes the execution (profile)
 - Instructions are sent to the HW for execution
- By “observing” the instruction stream:
 - The SW “learns” the behavior of the code
 - Identifies hot regions of the code
 - SW optimizes the code once
 - Store the optimized regions
 - HW executes the optimized code many times

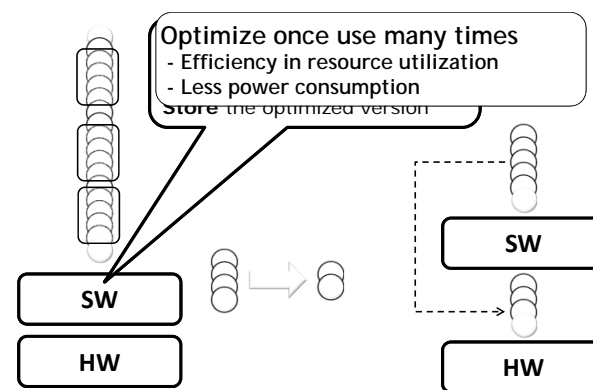


11

Designing Tomorrow's Microprocessors



The key idea



12

Designing Tomorrow's Microprocessors



The key idea

- Cost of running the SW:
 - Running the SW comes with some cost
 - “Observing” the dynamic code
 - Optimizing the code
 - Store optimized regions
 - The SW cost is amortized
 - The optimized segments are executed many times
 - Using staged optimization helps significantly:
 - Frequent regions : Few optimizations / low SW cost
 - Hot regions : Further optimization / medium SW cost
 - Critical regions : Maximum optimizations / high SW cost

13

Designing Tomorrow's Microprocessors



The key idea

- Comparison between HW-only and co-designed CPUs
 - “Amount” of hardware
 - Traditional approach : More hardware
 - Co-designed processors : Less hardware
 - Complexity of hardware
 - Traditional approach : Very complex (e.g. support for ooo)
 - Co-designed processors : Much simpler
 - Power Consumption
 - Traditional approach : High (optimization, HW complexity)
 - Co-designed processors : Significantly lower
 - Performance
 - Traditional approach : High
 - Co-designed processors : In the same order

14

Designing Tomorrow's Microprocessors



The key idea

- The benefits
 - Similar performance (often higher performance)
 - Less Power (extended battery life)
 - Smaller Area (lower cost)
 - Easier to design and validate (lower cost / shorter time-to-market)
- How?
 - Use SW instead of HW for optimizing
 - SW is usually easier to debug than HW
 - Keep the optimized code for future use
 - Efficient resource utilization (optimize once, use many times)

15

Designing Tomorrow's Microprocessors



Outline

- Overview of the technology
 - What are the HW/SW co-designed processors
- Key Ideas and Advantages
 - From the HW-only CPU to the co-design paradigm
- Research Projects / Market examples
 - Academic Research
 - Products
- Potential and Open Issues
 - A glance to the huge improvement potential

16

Designing Tomorrow's Microprocessors



Research projects / Market examples

- Many researchers identify the value of the approach
- A lot of work in academia
 - Parrot
 - ISCA 2004: “Power Awareness through Selective Dynamically Optimized Traces”
 - Targets both performance and power
 - The processor has 2 pipelines
 - Simple - lower power for “cold” regions
 - Aggressive - higher power for “hot” regions
 - Operation
 - Instructions initially go through the “cold pipeline”
 - Hot regions are identified and optimized
 - Optimized regions are stored and reused

17

Designing Tomorrow's Microprocessors



Research projects / Market examples

- Many researchers identify the value of the approach
- A lot of work in academia
 - rePLay
 - IEEE Transactions on computers 2001 “rePLay: A Hardware Framework for Dynamic Optimization”
 - Mainly targets higher performance
 - It is a HW only solution but follows the same principles
 - Is equipped with an optimization engine
 - Identify hot regions / optimize / store
 - Includes HW mechanisms to enable more optimizations
 - Uses aggressive HW

18

Designing Tomorrow's Microprocessors



Research projects / Market examples

- Market Examples
 - Transmeta™ Corporation (1995 - 2009)
 - Transmeta™ built the first co-design processors
 - Crusoe™ - 2000
 - Efficeon™ - 2004
 - VLIW architectures
 - Host ISA : x86
 - Elaborated Software : Code Morphing Software
 - Simple Hardware which provides special support for the optimizer
 - Main target
 - 100% compatibility
 - Similar performance
 - Lower power

19

Designing Tomorrow's Microprocessors



Agenda

- Overview of the technology
 - What are the HW/SW co-designed processors
- Key Ideas and Advantages
 - From the HW-only CPU to the co-design paradigm
- Research Projects / Market examples
 - Academic Research
 - Products
- Potential and Open Issues
 - A glance to the huge improvement potential

20

Designing Tomorrow's Microprocessors



Open Issues

- Conventional processors have been evolving for many years
- Co-designed processors is a new paradigm
 - A lot of work is needed for full exploitation
 - This is an amazing topic! Compilers + Computer Architecture + ...
- Some important questions
 - Exploit the dynamic information
 - Mechanisms to exploit dynamic information
 - Speculation techniques for performance / power
 - Leverage for multiprocessing
 - Software for efficient execution of parallel code
 - Fault tolerance

21

Designing Tomorrow's Microprocessors



Open Issues

- Conventional processors have been evolving for many years
- Co-designed processors is a new paradigm
 - A lot of work is needed for full exploitation
 - This is an amazing topic! Compilers + Computer Architecture + ...
- Some important questions
 - Segment Specific designs
 - Can we have CPUs optimized for different market segments?
 - One HW and different SW (maximize for performance / power)
 - Are traditional mechanisms good enough?
 - e.g. pre-fetchers, branch predictors
 - How to take advantage of the simpler circuitry

22

Designing Tomorrow's Microprocessors



Market examples / Research projects

- Reading:
 - “The Architecture of Virtual Machines”
IEEE Computer 2005, James E. Smith, Ravi Nair
Gives an excellent introduction to the whole technology
 - “Power Awareness through Selective Dynamically Optimized Traces”
ISCA 2004, Rosner et al.
Easy to understand and follow overview of where the benefits come from
 - “The Technology Behind Crusoe™ Processors”
White Paper 2000
A lot of information of the underlying implementation issues

23

Designing Tomorrow's Microprocessors



Conclusions

- Radical improvements usually come from radical solutions
- Do not use HW for everything
 - Consumes power, more complex design, higher cost...
 - Non-efficient resource utilization
- Co-designed processors: CPU=SW+HW
 - Efficient resource utilization
 - Less power, less complexity, lower cost, similar (higher) performance
 - Huge room for technology innovation
 - A huge interest in the research community

24

Designing Tomorrow's Microprocessors



Questions

?

`kyriakos.stavrou@intel.com`

25 Designing Tomorrow's Microprocessors 