Efficient Many-Core Systems

Florian Schmaus, Stefan Reif

2016-11-08
Moore’s Law (Computer History Museum, Mountain View, CA)
Moore’s Law

“The number of transistors incorporated in a chip will approximately double every 24 months”

- Not really a law, but an observation.
- Area of Integrated Circuit stays (roughly) the same
- Transistors get smaller → Can switch at higher speeds
- Computation power grows exponentially
Dennard Scaling [2]

As transistors get smaller, their power density stays constant.

- In other words: Smaller transistors need less current and voltage
- Power demand remains constant while transistor count grows
- “[...] even if many more circuits are placed on a [...] chip, the cooling problem is essentially unchanged.”
Dennard Scaling

Dennard Scaling [2]
As transistors get smaller, their power density stays constant.

“[...] even if many more circuits are placed on a [...] chip, the cooling problem is essentially unchanged.”
Breakdown of Dennardian Scaling

Why?

- Static power losses have increased [5]
  - because of complex quantum effects
  - which manifested because of the smaller component sizes

- Manufactures lost the ability to drop the voltage and the current
  - Because they need to counter the power losses

- As result, the power consumption per area is now increasing
  - Would eventually reach power density of a nuclear reactor core
  - Danger of overheating
Breakdown of Dennardian Scaling

Why?

- Static power losses have increased [5]

  We hit the **Power Wall** [7]

- As result, the power consumption per area is now increasing
  - Would eventually reach power density of a *nuclear reactor core*
  - Danger of overheating
Effects of the breakdown

- Low supply voltage
  - Lower supply voltage $\Rightarrow$ less leakage current
  - Low static power consumption

- Energy-inefficient software runs slowly [3]
  - Processor throttles due to thermal constraints
  - Energy management improves system performance

- Thermal runaway is possible
  - Higher temperature $\Leftrightarrow$ higher leakage current
  - “Hotspots” are dangerous
Effects of the breakdown

- Low supply voltage
  - Lower supply voltage $\Rightarrow$ less leakage current
  - Low static power consumption

- Energy-inefficient software runs slowly [3]
  - Processor throttles due to thermal constraints
  - Energy management improves system performance

- Thermal runaway is possible
  - Higher temperature $\Leftrightarrow$ higher leakage current
  - “Hotspots” are dangerous

- Clock speed increases no longer
  - Transistors switch less often $\Rightarrow$ lower dynamic power consumption
  - Supply voltage can be reduced $\Rightarrow$ lower static power consumption
The free lunch is over

“Most classes of applications have enjoyed free and regular performance gains [...] , because the CPU manufacturers [...] have reliably enabled ever-newer and ever-faster mainstream systems”

“ [...] the clock race [...] is over”

“ [...] if you want your application to benefit from the continued exponential throughput advances in new processors, it will need to be a well-written concurrent [...] application”

“programming languages and systems will increasingly be forced to deal well with concurrency”
The free lunch is over

- CPU manufactures can’t increase clock rate any more
- Herb Sutter: “Free lunch is over” [8]
  - “Free Lunch”
    - Software benefited from rising clock speed
    - Automatically, without any modifications necessary
- But: Sequential processing speed is reaching its limits
- Existing non-parallel software no longer profits from new parallel hardware
- Developers need to write parallel code

- We are on the edge from multi-core to many-core systems
  - Parallelism defines performance
  - Even for small-scale devices

- This trend requires new approaches and concepts from
  - Libraries / Runtime
  - Programming Languages
  - Operating Systems
The free lunch is over

- CPU manufactures can’t increase clock rate any more
- Herb Sutter: “Free lunch is over” [8]
  - “Free Lunch”
    - Software benefited from rising clock speed
    - Automatically, without any modifications necessary

We need Concurrency Platforms

- Even for small-scale devices
- This trend requires new approaches and concepts from
  - Libraries / Runtime
  - Programming Languages
  - Operating Systems
Cilk [1] is a C language extension **and** runtime library
- Keywords to express parallelism
- Provably efficient scheduler using work-stealing [4]
Cilk [1] is a C language extension and runtime library

Keywords to express parallelism

Provably efficient scheduler using work-stealing [4]

Parallel Fibonacci Function using Cilk

```c
uint64_t fib(uint32_t n) {
    if (n < 2)
        return n;
    uint64_t a = spawn fib(n-1);
    uint64_t b = fib(n-2);
    sync;
    return a + b;
}
```
Covers all layers from application down to hardware
- Hardware: Dark Silicon, accelerator units, . . .
- Software: POS, X10i, . . .

Tiled architecture
Tiles are interconnected with a two-dimensional NoC

Partitioned Global Address Space
Cores within tile share a coherent memory view
But no inter-tile cache coherence

Resource aware programming
- Resources are granted exclusively
OcotoPOS [6]  

A parallel operating system

- Enforces resource-allocation requests
  - PEs, Memory, NoC channels, accelerator units, ... 

- Works similarly to a distributed system
  - One OS instance per tile
  - Inter-tile communication via messages

- Kernel support for micro-parallelism
  - Async Syscalls, Futures, ... 

- Basic unit of execution: i-let
  - Consists of a function- and two data-pointer

- Interchangeable scheduler in user-space
  - HW-accelerated scheduling, work-stealing, ...
Conclusion

- Microprocessors hit a power wall
- Clock speed increases no longer
- Only parallel software is fast
- Parallel software needs support from
  - Libraries / Runtime
  - Programming languages
  - Operating systems
Conclusion

- Microprocessors hit a power wall
- Clock speed increases no longer
- Only parallel software is fast
- Parallel software needs support from
  - Libraries / Runtime
  - Programming languages
  - Operating systems

Concurrency Platforms
How to process the paper assigned to you:

- Summarize
- Present motivation, proposed solution and evaluation
- Put in perspective
- Who wrote it?
- When was it written?
- Related work and delta to related work?
- Citation count?
- Discuss and constructively criticize
- Threats to validity discussed?
- Weak motivation/evaluation?
- Approach inconclusive?
- Incomplete implementation?
How to process the paper assigned to you:

- Summarize
  - Present motivation, proposed solution and evaluation
Seminar Requirements

Short Recap

How to process the paper assigned to you:

- **Summarize**
  - Present motivation, proposed solution and evaluation

- **Put in perspective**
  - Who wrote it?
  - When was it written?
  - Related work and delta to related work?
  - Citation count?

Discuss and constructively criticize:

- Threats to validity discussed?
- Weak motivation/evaluation?
- Approach inconclusive?
- Incomplete implementation?
How to process the paper assigned to you:

- Summarize
  - Present motivation, proposed solution and evaluation

- Put in perspective
  - Who wrote it?
  - When was it written?
  - Related work and delta to related work?
  - Citation count?

- Discuss and constructively criticize
  - Threats to validity discussed?
  - Weak motivation/evaluation?
  - Approach inconclusive?
  - Incomplete implementation?
Seminar Motivation

- Techniques learned will become handy

- You will read a lot of papers for your BA/MA

- It will help you writing a good BA/MA
Seminar Motivation

- Techniques learned will become handy
- You will read a lot of papers for your BA/MA
- It will help you writing a good BA/MA
- Because you have to
Thanks for your attention!

Questions?


References II


