ISSUES IN MANY-CORE ARCHITECTURE

Levs Dolgovs
Sheenam Jayaswal

Agenda

- Why Many Core?
- What is Many Core?
- Issues With Many-Core Processors
- An Example of a Many-Core Processor
- Applications
- Future Scope

Why Many-Core?

Moore's Law

- Moore's Law: The number of transistors on a chip doubles every 18 months.

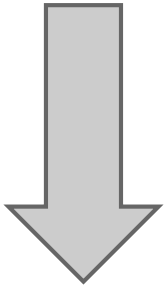
Limitations Of Single Core

- The Power Wall
  - Limit on the scaling of clock speeds.
  - Ability to handle on-chip heat has reached a physical limit.
- The Memory Wall
  - Need for bigger cache sizes.
  - Memory access latency still not in line with processor speeds
- The ILP Wall
  - Superlinear increase in complexity without linear increase in application performance.

Need for Multi-Core Processors

Power Wall + Memory Wall + ILP Wall =

Brick Wall for Serial Computing!



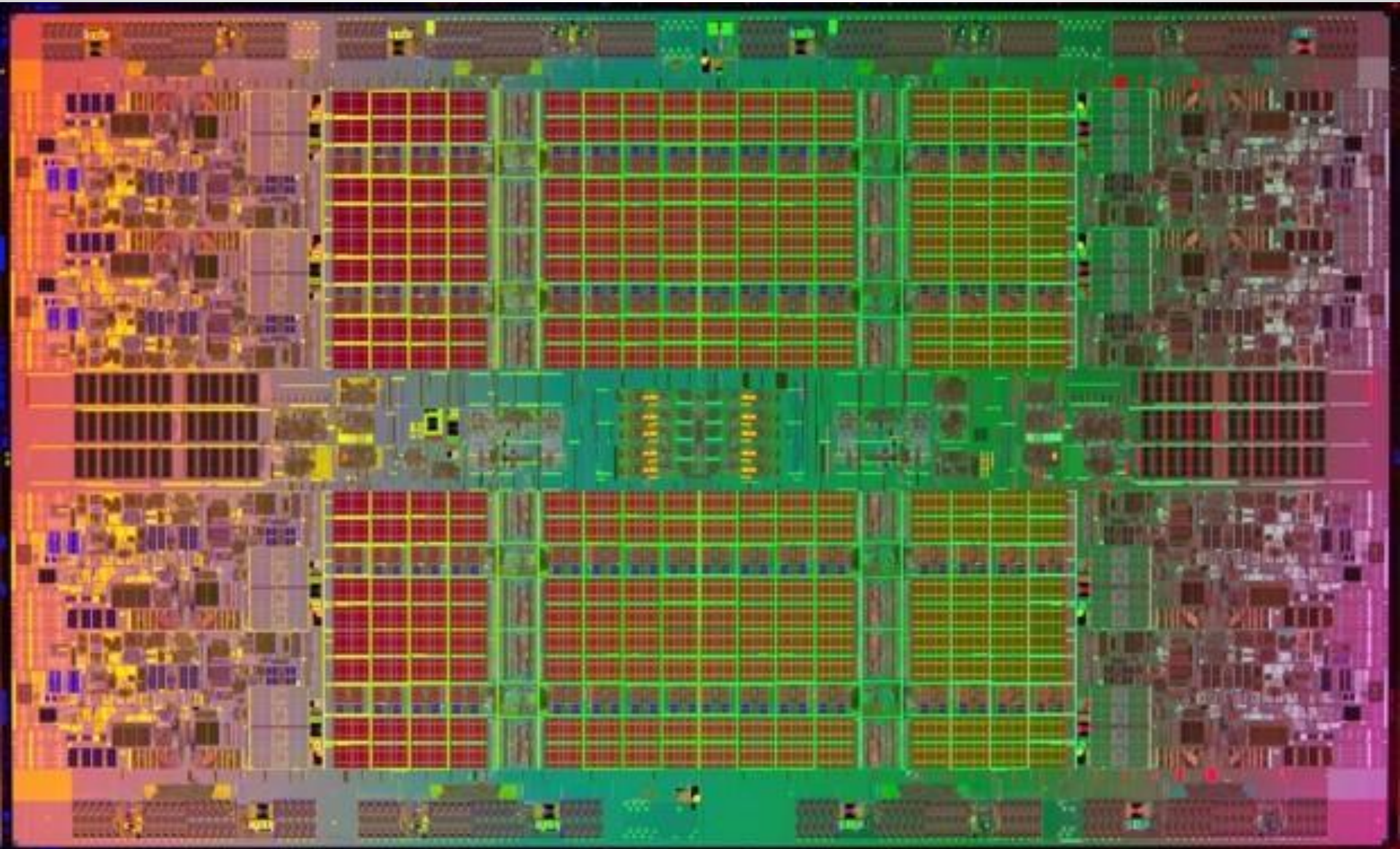
MULTI-CORE PROCESSORS

Limitations of Multi-Core Processors

- Imperfect scaling.
  - Performance was dependent on serial code. (Amdahl's Law).
- Difficulty in software optimization.
  - Easier to add cores. Difficult for software to take advantage of them.
- Maintaining concurrency over a number of cores.

The limitations of Multi-Core Processors led to the need for Many-Core Processors.

Multi-core example: Intel Itanium

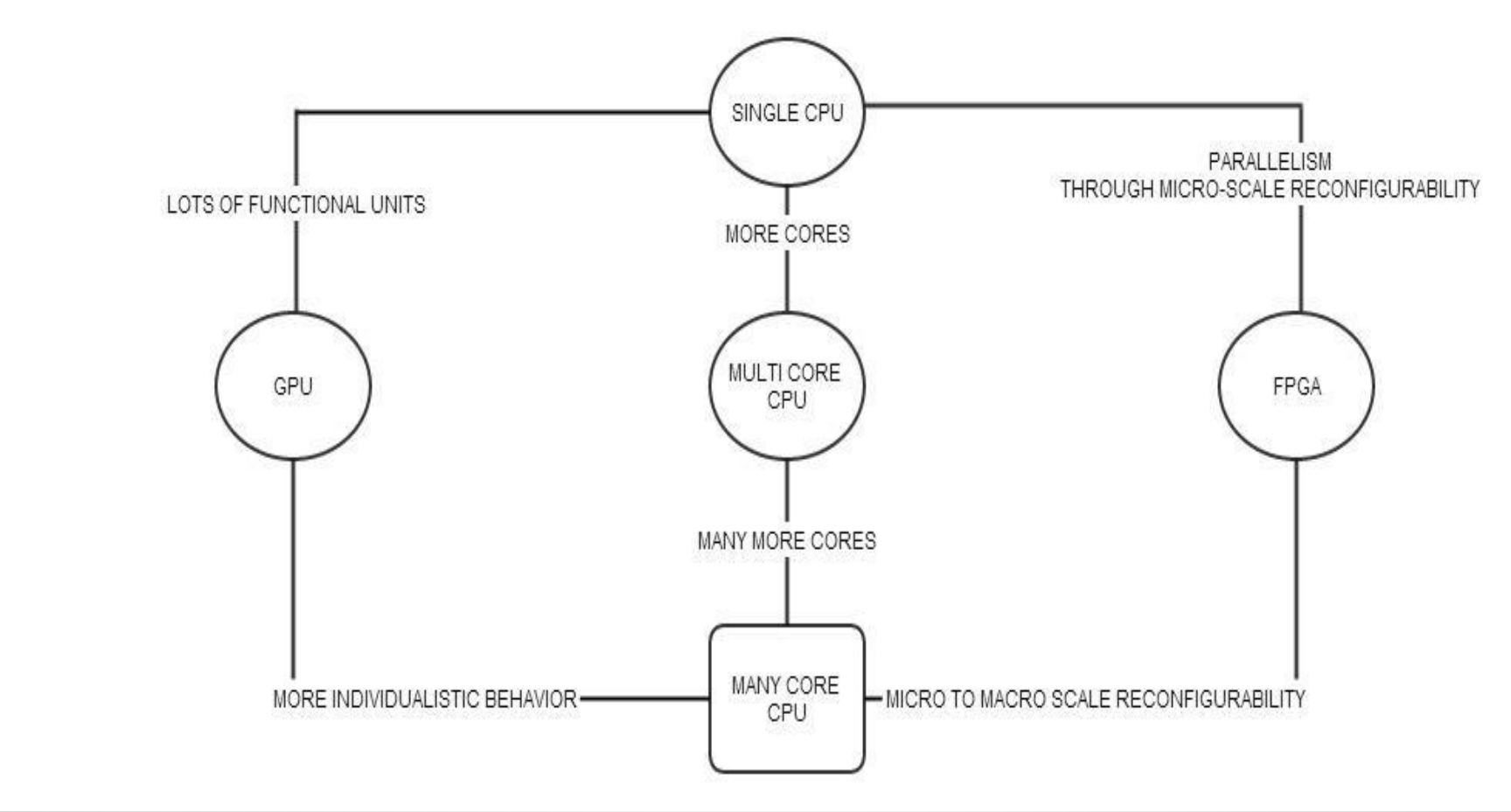


What is Many-Core?

What is Many-Core?

“The terms many-core and massively multi-core are sometimes used to describe multi-core architectures with an especially high number of cores(tens or hundreds)”- Andras Vajda

What is Many-Core?



Issues With Many-Core Processors

- Power
- Connectivity
- Arbitration
- Memory Issues
- Cache Coherence
- Scheduling
- Programmability

Power

- Difficult to predict on chip power.
  - Difficult to predict the utilization of the different processors at any given instance.
  - May lead to overheating if all cores are running at full potential.
  - Solution: Efficient tools for power estimation
- Increased power dissipation with decreasing feature size
  - Increase in leakage current with decreasing feature size.
  - Solution: Advances in Material Sciences

Connectivity

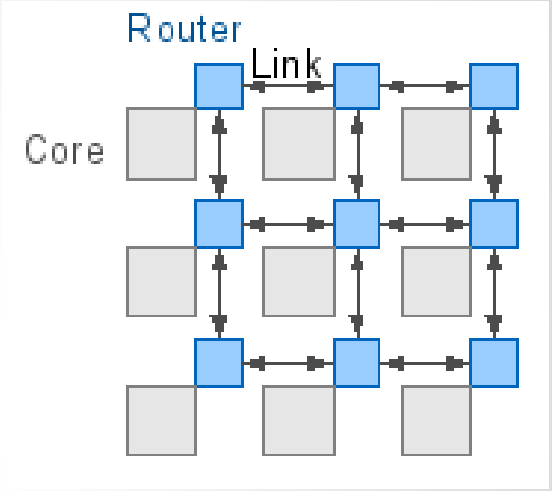
- Increase in the number of wires
- Increase in power dissipation
- Need for long wires

Solution?

Network on Chip

Network on Chip

- On chip modules exchange data through a network
- Routers are used to transmit and receive the information
- A simple protocol is followed
- Ongoing research to minimize the number of hops in the case of many core chips



Arbitration

Few-core processors: global cache (before - even a global bus) → we have a defined global order of shared memory operations.

Not the case with many cores!

Memory: Common RAM

Parallel programs will use the cores efficiently, and use the same pages of data and instructions. → Common cache, prefetching!

Serial programs will have to be combined (time sharing), and then memory access will be random. → Individual cache

Memory: Individual Cache

Cache coherence problem: what if two processors write to their caches of a common memory page? (Not even simultaneously)

As soon as somebody writes, he invalidates all other caches. Instead of communicating data, we communicate the state of caches.

Cache still transparent!

Memory: Private Memory

What if we intentionally manage the flow of data between chips? (opposed to RAM-chip-RAM)

What if we don’t keep the cache coherent AKA give each core its own memory?

Scheduling

Processes will be able to request

- Time
- Specific configuration of cores
- Specific global placement

How to schedule that?

- Same as 1-core + load balancing + NUMA
- Core allocator, then scheduler for each core

Programmability: How To Leverage Many-Core Processors?

- Spatial memory management - explicit/implicit?
- Task affinity management - explicit/implicit?
- Message routing - explicit/implicit?
- Parallel task count management - explicit/implicit?

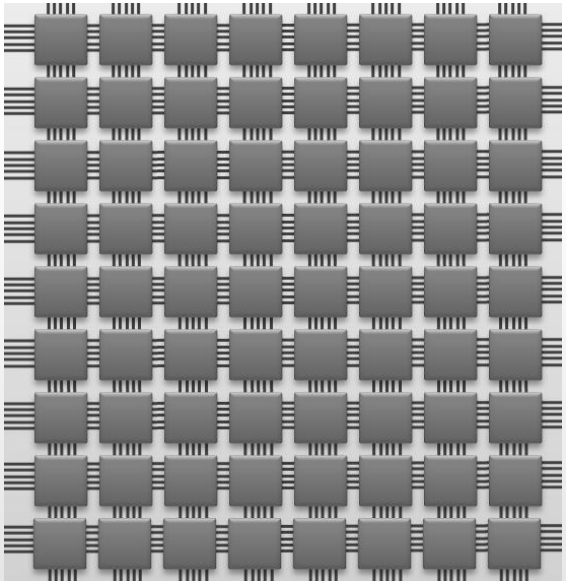
Several programming models:

- Communicating Sequential Processes (CSP)
- Actor model
- Task-based programming models

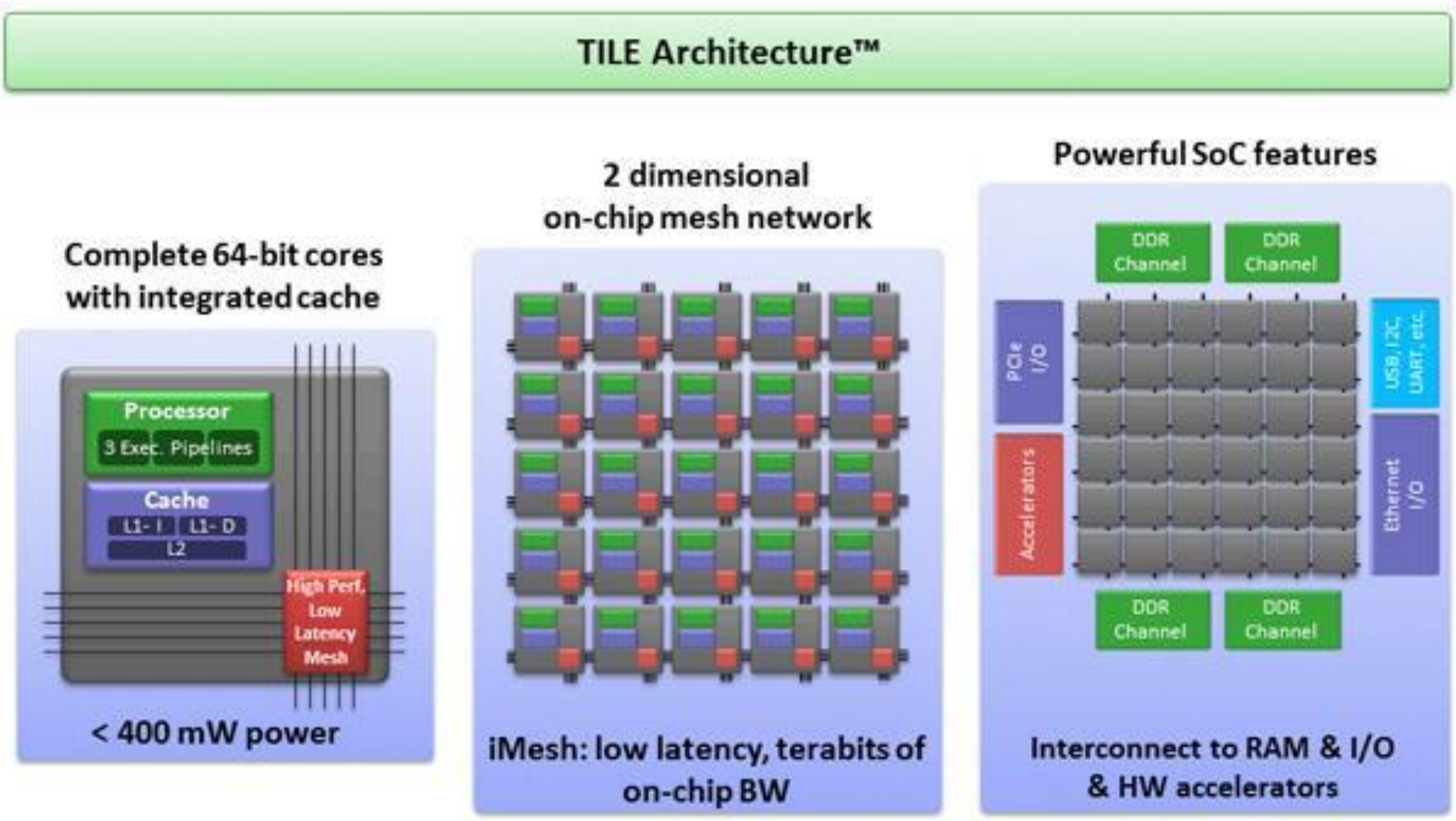
Write the program in a parallel paradigm, leave the routing and process allocation to the lower level.

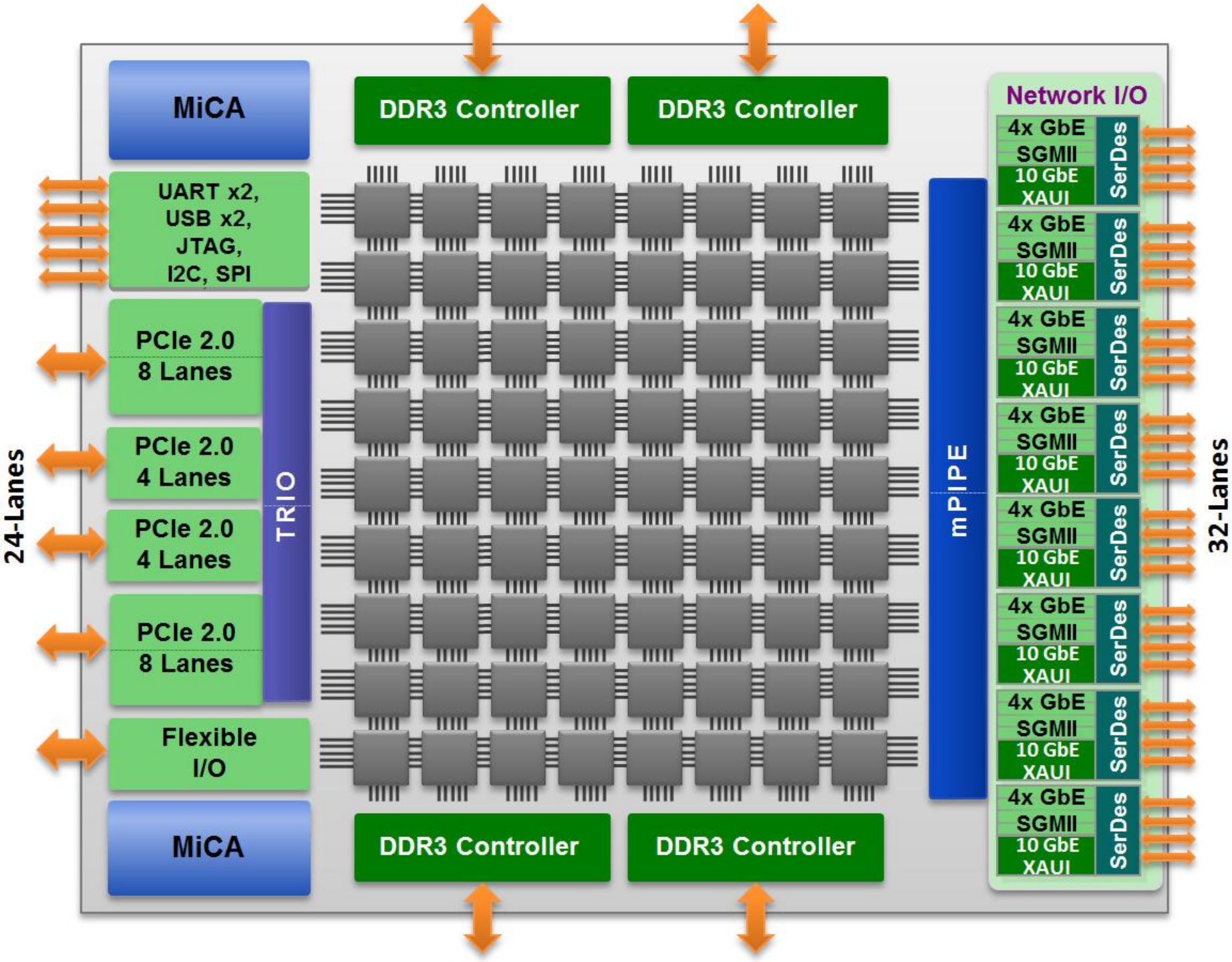
Software Side

Constructs that work well in serial programming	Constructs that work well for very parallel systems
Shared memory+Locks (less communication)	Message-based
Monolithic OS kernels (less security check overhead)	Microkernels/exokernels? Something else?



A Working Example: Tilera




MiCA
DDR3 Controller
DDR3 Controller
Network I/O
UART x2,
USB x2,
JTAG,
I2C, SPI
PCIe 2.0
8 Lanes
PCIe 2.0
4 Lanes
PCIe 2.0
4 Lanes
PCIe 2.0
8 Lanes
TRIO
Flexible
I/O
MiCA
DDR3 Controller
DDR3 Controller
mPIPE
4x GbE
SGMII
10 GbE
XAUI
SerDes
24-Lanes
32-Lanes

Possible Applications

- Improvement in Human Computer Interfaces
- Replace some FPGAs (more programmable)
- A supercomputer for every scientist!
- More cloud-based software (cheap servers)

The Future

- There will be more cores!
- They will be programmed and managed differently!
- More data-intensive computations, less user intervention