Runtime Aware Architectures

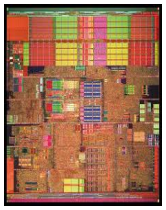
Prof. Mateo Valero
Director of BSC

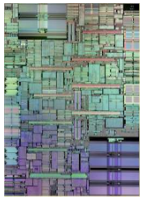
- Design of Superscalar Processors
  - Decoupled from the software stack

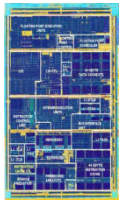
Applications

ISA

Fetch → Decode → Rename → Instruction Window / Wakeup+ select → Register file → Bypass → Data Cache → Register Write → Commit







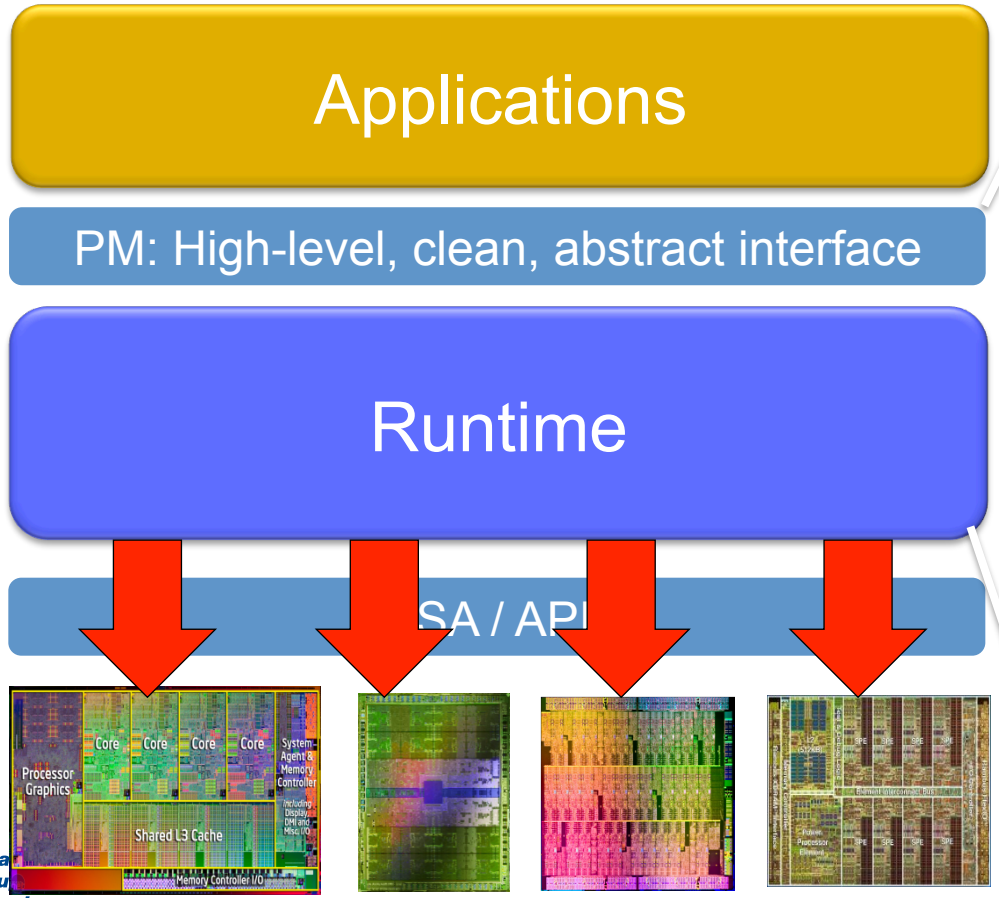
Programs "decoupled" from hardware

Simple interface Sequential program

ILP

Power to the Runtime

- Efficient usage of the underlying hardware
- Tailored for future technologies
- The runtime **drives** the hardware design



Task-based PM annotated with data dependencies

Task Dependency Graph built at runtime

Dynamic scheduling

OmpSs

"Reuse" architectural ideas under new constraints

Runtime Aware Architectures Challenges

- Re-design memory hierarchy
  - Hybrid (cache + local memory)
  - Non-volatile memory, 3D stacking
  - Simplified coherence protocols, non-coherent islands of cores
- Exploitation of data locality:
  - Reuse, prefetching, in-memory computation

Memory Wall

- Heterogeneity of tasks and Hardware
  - Critical path exploitation
- Accelerators
  - Numerical, data bases, proteomics, big data

Power Wall

- Efficient data movement
  - Overlap communication and computation
  - Latency aware interconnection network

Program. Wall

- Hardware acceleration of the runtime system
  - Task dependency graph management
- Load balancing and scheduling
  - Asynchrony and critical path exploitation

Resilience Wall

- Task-based check-pointing
- Algorithmic-based fault tolerance

Conclusions

- The design of future multicore/parallel systems has to dramatically change
  - Hardware-Software co-design
- Runtime has to drive the design of future multicores
- Runtime Aware Architectures will allow
  - Efficient management of parallelism and energy
  - Improve memory management and reduce data movements
  - Increase reliability
- Ensure continued performance improvements, once more **R**iding **o**n **Mo**ore's **L**aw (RoMoL)