Some Opportunities and Obstacles in Cross-Layer and Cross-Component (Power) Management

Onur Mutlu
onur@cmu.edu

Carnegie Mellon
What Are These Layers and Components?

- Well-defined (mostly) interfaces between transformation layers and components/resources create abstractions
Advantages of abstraction layers

- Improved productivity at each layer/component
  - No need to worry about decisions made in higher/lower levels
- Designer sanity

Disadvantages

- Overall suboptimal design
  - Uncoordinated decisions contradict each other, degrade efficiency
An Example: Lack of OS-Architecture Coordination

Layers and Components: Pros and Cons

- **Advantages of abstraction layers**
  - Improved productivity at each layer/component
  - No need to worry about decisions made in higher/lower levels
  - Designer sanity

- **Disadvantages**
  - Overall suboptimal design
    - Uncoordinated decisions contradict each other, degrade efficiency
  - Waste due to overprovisioning at each level
  - System-level metrics hard to optimize
    - Application-level QoS, user experience, TCO, total energy, ...
    - Minimize energy while satisfying QoS

- **Research question:** How do we eliminate disadvantages without sacrificing the advantages of layers?
One General Problem

- Local optimizations *relatively easy* to think of; local metrics *easy* to optimize
- Global optimization much less so

- We may be lacking *unifying principles* that span across layers to
  - Coordinate information flow and policies across layers
  - Guide/ease the co-design and management of resources within and across layers
How Do We Fix The Problem?

- Find and employ **cross-cutting (unifying)** design and resource management **principles**
  - Ensure these principles are conveyed in the interfaces

- Design and enhance **interfaces** (driven by principles)
  - E.g., the application-OS-architecture interface for resource management

- Develop **design tools** to enable global optimization
  - Cross-layer evaluation infrastructures

- Change **mentality**
  - Be open-minded to proposals that co-design multiple layers
  - Performance- vs energy- vs user-centric design
  - Provide commensurate focus on important subsystems: memory and interconnect → **develop new execution models**
Interconnect Energy Scaling

![Chart courtesy of Shekhar Borkar, Intel](image-url)

- Interconnect Energy: 1.6X
- Compute Energy: 6X
One Unifying Principle: Criticality/Slack

- **Dynamically identify critical/bottleneck tasks** (pieces of work) and spend most of the energy on them
  - Not all **instructions** are equally important [Fields+ ISCA’01,02]
  - Not all **network packets** are equally important [Das+ ISCA’10]
  - Not all useful **cache blocks** are equally important [Qureshi+ ISCA’06]
  - Not all **threads** are equally important [Bhattacherjee+ ISCA’09]
  - Not all **queries** are equally important
  - Not all **applications** are equally important

- Create a **criticality interface** between layers and components
  - Application/compiler ↔ OS/runtime ↔ hardware
  - Enable ways of discovering and expressing task criticality and managing resources based on criticality/slack
Exploiting Criticality/Slack

- **Identify** the criticality/slack present in each task
- **Convey** criticality information across layers/components
- **Co-design** the layers and resources such that they can prioritize tasks based on criticality/slack
- **Manage** execution to minimize slack and spend little energy on non-critical tasks

Identify bottleneck threads that cause serialization and dynamically ship them to high-power high-performance cores

[Suleman+ ASPLOS’09, Joao+ ASPLOS’12]
Exploiting Dynamic Criticality: Other Examples

- Identify critical instructions and execute them in fast pipelines [Fields+ ISCA’02]

- Identify critical threads and slow down others [Bhattacherjee+ ISCA’09]

- Identify critical/limiter threads in a parallel application and prioritize them in the memory controller [Ebrahimi+ MICRO’11]

- Identify latency-sensitive applications/threads and prioritize them in the memory controller [Kim+ MICRO’10] or entire shared memory system [Ebrahimi+ ASPLOS’10]

- Identify critical interconnect requests from core’s point of view and prioritize them in routers [Das+ ISCA’10]
Thread Cluster Memory Scheduling [Kim+ MICRO’10]

1. Group threads into two clusters
2. Prioritize non-intensive cluster
3. Different policies for each cluster

TCM, a heterogeneous scheduling policy, provides the best fairness and system throughput
Other Unifying Principles and Research Issues

- **Consolidate** many tasks to maximize efficiency and provide the QoS each task needs
  - QoS interface across layers
    - Application $\leftrightarrow$ OS/runtime $\leftrightarrow$ architecture $\leftrightarrow$ uarch.

- **Coordinate** the management of shared resources across components and layers
  - Goal: no efficiency loss due to contradictions in policies
  - Policy co-design (e.g., between cores, interconnect, memory controllers, memory): criticality can again help here

- **Exploit heterogeneity** at all layers
  - Both architectural/microarchitectural and technology heterogeneity can provide large efficiency gains
Role of Education and Funding

- We need to break the layers in education first, to be truly successful
  - Cross-layer undergraduate courses
- Fund both within-layer and cross-layer approaches
- Fund widely many areas – there is likely no silver bullet
Some Opportunities and Obstacles in Cross-Layer and Cross-Component (Power) Management

Onur Mutlu
onur@cmu.edu

Carnegie Mellon