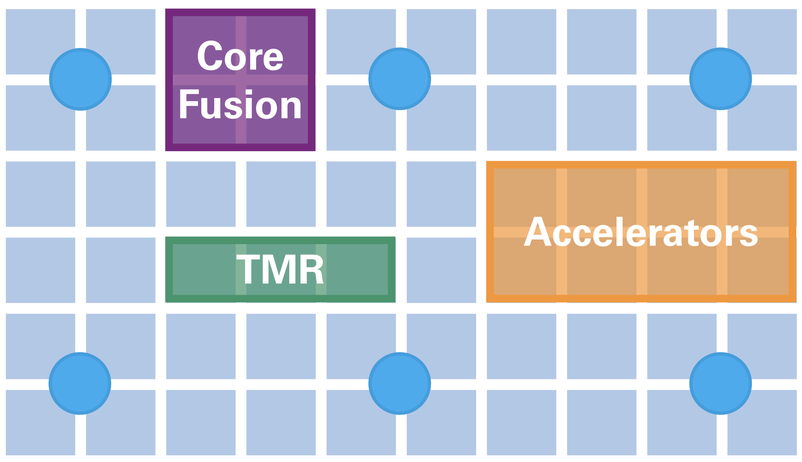
TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Operating Systems

WHO IS GOING TO PROGRAM THIS?

MARCUS VÖLP, MICHAEL ROITZSCH, HERMANN HÄRTIG

Imagine future processors look like this:



Dynamically Heterogeneous Manycores

- Core Fusion: **pooling resources** of small cores to form a larger high-throughput core
- Resilient Cores: **redundancy** to compensate hardware errors
- Specialized Cores: **accelerator units**
- 3D-stacking: **connect DRAM** or more cores

We face a mismatch between system and applications:

Operating System Challenges

- **two-way mediation** between applications and hardware resources
- **adapt** software parallelism and accelerator use
- **reconfigure** hardware to match throughput needs
- **spatial placement** for data proximity

Today's Applications

- **hardcoded** threads
- **cumbersome** use of accelerators
- **opaque** data use
- **future behavior invisible**

We propose an asynchronous lambda infrastructure:

lambdas instead of threads for parallelism:

```
void worker() {
  work();
}
start_thread(worker);
```

```
dispatch_async(q, ^{
  work();
});
```

queues for asynchronous execution

- expose pending work
- latent parallelism

Application

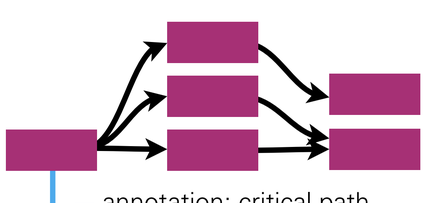
expose static and dynamic program structure
receive non-functional properties, QoS

analyze queued work

- serial queues show dependencies
- alternative implementations for accelerators

learn lambda metadata

- memory accesses
- instruction throughput



Runtime

expose execution alternatives, requirements
receive feedback from global scheduling

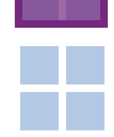
execution stretch as a penalty for:

- distance to data
- unused accelerators
- lower single-thread throughput

scheduling decisions

- decentralized by gossiping

run critical path on fused core with high throughput



run parallel section on multiple individual cores

Operating System

We call this concept "Elastic Manycore Architecture."

IMDATA

Europa fördert Sachsen. ESF Europäischer Sozialfonds

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Who is Going to Program This?

Marcus Völp, Michael Roitzsch*, Hermann Härtig
Technische Universität Dresden, Germany
{voelp, mroi, haertig}@os.inf.tu-dresden.de
* Student

The past years have shown us a wide variety of chip multiprocessor systems, rapidly evolving in terms of parallelism, varying distance to memory and heterogeneity. However, the pace of computer architecture change is not slowing down. Instead, micro-architectural trends and 3D stacking technology forecast a whole new class of systems with unique adaptation capabilities to the varying demands of applications. The role of the operating system (OS) is to mediate between increasingly complex applications and evermore complex hardware. This abstract makes the case for a new programming model and system interface to leverage the performance of these new architectures while maintaining programability.

Micro-Architectural Trends

Heterogeneous multicores are known to offer significant advantages in terms of energy and area efficiency by integrating cores with heterogeneous performance characteristics and sometimes also heterogeneous instruction sets. Applications demanding high single-thread performance could be run on large superscalar cores while many lightweight cores offer the compute power needed for parallel workloads. Placing the right parts of an application on the right cores is already a challenge with statically floorplanned chips. However, what if a group of neighboring lightweight cores could be dynamically fused to generate the performance of one larger core [1,2]? What if there are specialized accelerator cores for functionality such as encryption or vector operations? What if 3D stacking [3] enables high-bandwidth connections to memory for selected nearby cores? A future OS will have to make placement and scheduling decisions to deliver the benefits of such *elastic manycores* to applications.

Local Knowledge, Global Management

Application demand may be dynamically growing and shrinking. To be able to assign resources not used by one application to another, we want to steer away from static partitioning. However, cross-application elasticity requires global management, which needs information about the applications to base its decisions on and knobs to influence their behavior.

Unfortunately, knowledge on application behavior like available parallelism is hidden within today's programs. Many applications devise their own, local thread pool with poor or no feedback from global system state.

The OS on the other hand only sees threads to manage. It can place them on cores and schedule their execution, but it does not know their current importance or progress, or which execution units of the cores they will stress. Neither can the OS control their number when one application demands more cores and the system decides to cut down another.

The control portion of the problem has been explored since Anderson et al. presented Scheduler Activations. We propose an application runtime and OS co-design, where applications export a richer representation of their internal execution behavior, thus contributing local knowledge to facilitate global control.

Reality Check

Exporting previously internal knowledge requires a different application structure. Is this proposal another case of researchers burdening developers with rewriting all software? Fortunately, modern programming paradigms with industry traction already walk this path. Languages and runtimes such as Apple's Grand Central Dispatch (GCD), Microsoft's Parallel Patterns Library or IBM's X10 encourage a programming model called *asynchronous lambdas*. Independent work is not organized in threads, but encapsulated in lambda functions and submitted to work queues for asynchronous execution.

For our solution, we modify the GCD runtime to collect metadata on lambdas. Such data can include a dependency graph to plan parallel execution, the expected runtime of each lambda to aid scheduling, trace data on the use of accelerator instructions like FPU or vector processing for efficient core assignment, or a data access analysis for placing code close to its data. The runtime exports this knowledge to the OS for global management. Our intended scheduler aggregates core fusing options, and proximity to data and accelerators in a distance penalty which it minimizes. An activation-like interface allows the OS to control, which lambda runs when on which core.

We aim for a win-win-situation, where the OS gains more information and control options to utilize new hardware efficiently. At the same time, we hope to simplify developers' lives by relieving them of local management decisions.

References

[1] E. İpek, M. Kırman, N. Kırman, J. F. Martínez: Core Fusion: Accommodating Software Diversity in Chip Multiprocessors, ISCA 2007

[2] H. Homayoun, V. Kontorinis, A. Shayan, T. Lin, D. M. Tullsen: Dynamically Heterogeneous Cores Through 3D Resource Pooling, HPCA 2012

[3] G. H. Loh, Y. Xie, B. Black. Processor design in 3D die-stacking technologies. IEEE Micro 2007