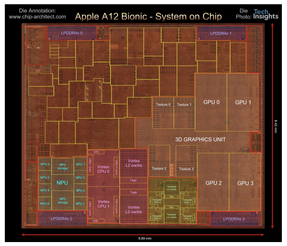
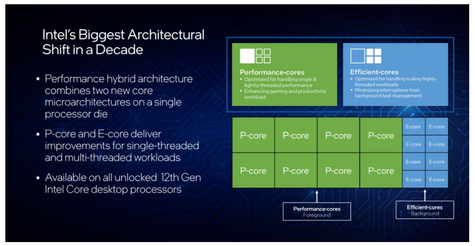
Fast Thread Migration in a Heterogenous ISA System

Project #10: John Fang, Charles Hong, Max Banister

Problem

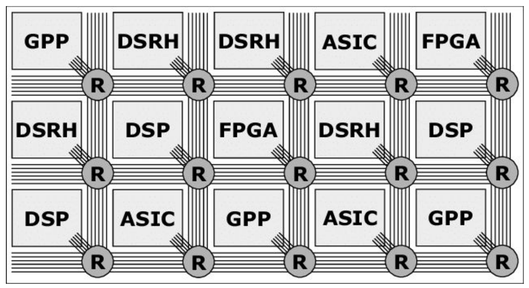
- Heterogenous SoCs are an increasingly popular design choice
- SMP kernels aren't built to run across diverse ISA multicore systems
- System designers create custom software interfaces per IP block, duplicates effort





Background

- There is a movement towards using RISC-V as the base ISA across all IP's



- Accelerators with an attached processor are a good idea
- Control processors have ISA extensions to control the accelerator
- Observation: Having the same base ISA allows a thread to start on one processor and migrate to another, based on what ISA extension it requires at the time

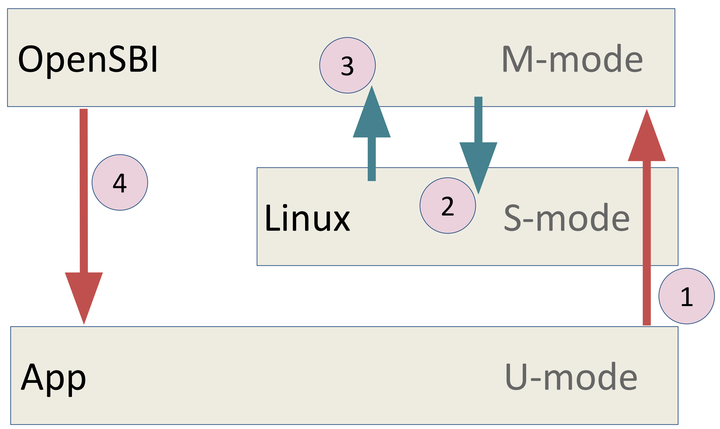
Solution

A fast and automatic thread migration mechanism addresses the heterogeneity of such systems.

- Only a few general-purpose application cores will run Linux
- Firmware migrates threads to corresponding accelerator cores when executing extension instructions and back to general-purpose cores for OS services
- A single program can move across accelerators supporting various ISA extensions

Idea: Trap on accelerator-specific custom instructions, and use that event to trigger a thread migration to an accelerator control processor

- We use the OpenSBI firmware to detect hart capabilities and coordinate communication between firmware and the kernel
- Minimal changes to Linux to handle scheduling tasks on accelerator cores (~100 LOC)
- Application writer may be oblivious to core migrations, though intimate knowledge of the platform can be used for further optimization



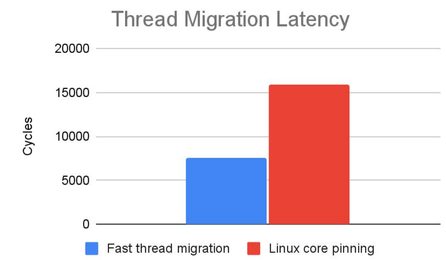
1: A illegal instruction trap is taken on a custom instruction not supported on this hart

2: M-mode software delegates the trap to Linux, which in turn makes an SBI call to begin the migration process

3: OpenSBI reads the user's register state and initiates an IPI to the accelerator peripheral core

4: OpenSBI, now on the accelerator core, dequeues the requisite execution state and prepares to jump to U-mode to run the task

Results



- Time for first migration of a task longer due to time to pin physical memory pages
- Migration requires additional data movement of saved processor state, so latency scales with how much state must be saved
  - Accelerators with a large scratchpad are a bad fit for thread migration
- No significant resource contention overhead expected (i.e. latency scales linearly with number of application processors vying for same accelerator)