CS 516 Compilers and Programming Languages II

Compiler Optimizations for Power / Energy / Thermal Introduction
Announcements

• Sakai site is up, together with piazza. Please check whether you have access

• First homework will be posted by Monday

• Office hours: Friday 2:00-3:00pm or by appointment
What other optimization goals are there?

- Performance (dynamic execution time)
- Size of executable
- Power (peak power dissipation)
- Energy (battery life)
- Thermal (cooling)

How do these different optimization goals interact?

- Does one optimization goal subsumes another, or are they all different?
- Can one optimization goal conflict with another? (e.g.: power vs. performance, thermal vs. performance)
NVIDIA TX-1 and TX-2 Boards

Jetson TX-1

TX-1 < $400

Jetson TX-2

TX-2 < $600

Technical Specifications

**TX-1**
- **GPU**: 256-core NVIDIA Maxwell™ GPU
- **CPU**: Quad-Core ARM® Cortex®-A57 MPCore
- **Memory**: 4GB 64-bit LPDDR4 Memory
- **Storage**: 16GB eMMC 5.1
- **Power**: Under 10W
- **PCIE**: Gen 2 | 1x4 + 1x1
- **CSI**: 12x CSI2 D-PHY 1.1 lanes (1.5 Gbps/Lane)
- **Wi-Fi**: Yes

**TX-2**
- **GPU**: 256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores
- **CPU**: Dual-Core NVIDIA Denver 2 64-Bit CPU
  - Quad-Core ARM® Cortex®-A57 MPCore
- **Memory**: 8GB 128-bit LPDDR4 Memory
  - 1866 MHz - 59.7 GB/s
- **Storage**: 32GB eMMC 5.1
- **Power**: 7.5W / 15W

Single board “supercomputers” (1 TFLOPS) for edge cloud and robotics applications.
mPower™

mFi® Controllable Power Outlets

The mFi® mPower™ products are mFi controllable power outlets with Wi-Fi capability. Once connected, they can be managed by the mFi Controller software and configured to operate on a schedule with customizable power on/off rules.

Models: mPower, mPower (EU), mPower mini, mPower mini (EU), mPower PRO, mPower PRO (EU)

---

**Single TX-1 systems on individual ports**

**Web interface and command line options**

**Will be used for homework and first project**
**Instantaneous power (P):** activity level at a given point in time

We only consider DC circuits here.

**Ohm's law**

\[ I = \frac{V}{R} \]

**Power equation**

\[ P(t) = V(t) \times I(t) \]
power \((P)\): activity level at a given point in time (watts)
energy \((E)\): total amount of activity (joules)

same energy, different (peak) power

\[ E = \int P(t) dt \quad \text{general} \]
\[ E = P \times t \quad \text{constant } P \text{ over time} \]

optimizing for (peak) power == optimizing for energy?

**ANSWER:** Not necessarily! Example: re-schedule activities
**Power/Energy vs. Performance**

**Performance**: overall program execution time (seconds)

---

**Optimizing for power/energy == optimizing for performance?**

---

**Answer**: (a) Mostly Yes, at least for traditional optimizations that reduce overall computation and memory activity

- redundancy elimination (CSE, dead code elimination)
- strength reduction (e.g.: replace $2 \times a$ with $a + a$), loop invariant code motion
- memory hierarchy (locality) optimizations (register allocation, loop interchange, loop distribution, blocking for cache)

---

**Answer**: (b) Not really, in particular for optimizations that exploit tradeoff between power/energy usage and performance

- loop invariant code motion, aggressive speculation
- blocking for cache
- DFVS, resource hibernation, remote task execution, QoR (approximations)
Which code is better in terms of power/energy and which in terms of performance?

**ANSWER:** It depends

- simple RISC architecture: (B)
- VLIW or superscalar architecture with empty “slots”: (A)

The **critical path** is the key abstraction for performance optimizations.
Performance is determined by the length of the critical path of a computation.

Sample task graph

Tasks that are independent may be executed in parallel.
Performance is determined by the length of the critical path of a computation.

Sample task graph

Critical path has 17 cycles.
Some important lessons to learn:

You can run, but you cannot hide
- pushing instructions onto the non-critical execution path; ("hiding") does not necessarily reduce energy/power
- higher threshold for profitability of speculation

You cannot beat hardware (need for heterogeneous architectures)
- if an operation is implemented in hardware, and an applications needs it, that's the best you can do (e.g.: floating-point unit, )
- need to be able to disable hardware if not in use (dark silicon)

Keep the overall picture in mind
- performance is measured for the entire program
- power/energy should also be measured for the entire system, in addition to optimized system component(s)

There is no free lunch
- you typically cannot avoid tradeoffs
- tradeoffs are application specific and/or user specific
power (when): activity level at a given point in time
energy (what): total amount of activity
thermal (where): location of activity / power density

Thermal optimization: Spread activities across spatial dimensions: Where to do things? A larger surface is easier to cool!
Dynamic power is dissipated due to switching activities. Static power is dissipated without switching activities (leakage power).

- Dynamic power is required to charge and discharge load capacitances $C$ when transistors switch.
- A transistor has a rising (0→1) and falling (1→0) output.
- On rising output, charge $Q = CV_{DD}$ is required.
- On falling output, charge is dumped to GND (ground).
- This repeats $T \times f_{\text{clock}}$ times over a time interval of $T$.

$$P \sim C \times V_{DD}^2 \times f_{\text{clock}}$$
Dynamic Frequency and Voltage Scaling (DFVS)
- power (P) and energy (E) are proportional to \( C \cdot V_{dd}^2 \cdot \text{freq} \)
- reducing freq alone reduces P, but not E
- many architectures support sets of (V\(_{dd}\), freq) pairs
- CMOS technologies have fixed threshold voltages \( V_{\text{threshold}} \) such that \( V_{\text{threshold}} < V_{dd} \), i.e., limit on how low \( V_{dd} \) can go
- leakage current (I\(_{\text{leakage}}\)) problem ("static power"): increases with heat and when \( V_{dd} \) gets closer to \( V_{\text{threshold}} \)

Dynamic Resource Configuration/Hibernation
- system components are "switched off" or "put to sleep"
- many architectures support several sleep/hibernation states/levels
- the deeper the "sleep", the more energy you save, but the cost of "waking up" becomes higher (time, power, and energy)

Remote Task Execution
- use of remote services (let someone else do the work for you)
- cost (communication) < cost (doing it yourself)

QoR Optimizations (Approximation Redundancy)
- may save resources by reducing quality outcomes
Who may perform these optimizations?

- **User / application programmer**
  - introduce new programming language constructs / abstractions
  - crucial if semantic of program is affected

- **Compiler**
  - May perform (static) full-program analyses (execution context)
  - May reshape program behavior
  - Low runtime overhead

- **Operating System (OS)**
  - adaptable to runtime behavior
  - optimization in multi-programming environments
  - limited execution contexts of single application

- **Hardware**
  - fine-grain optimizations
  - low overhead, but also least execution context

- **Combination of techniques from levels above**
More detailed discussion of compile-time power/energy optimization