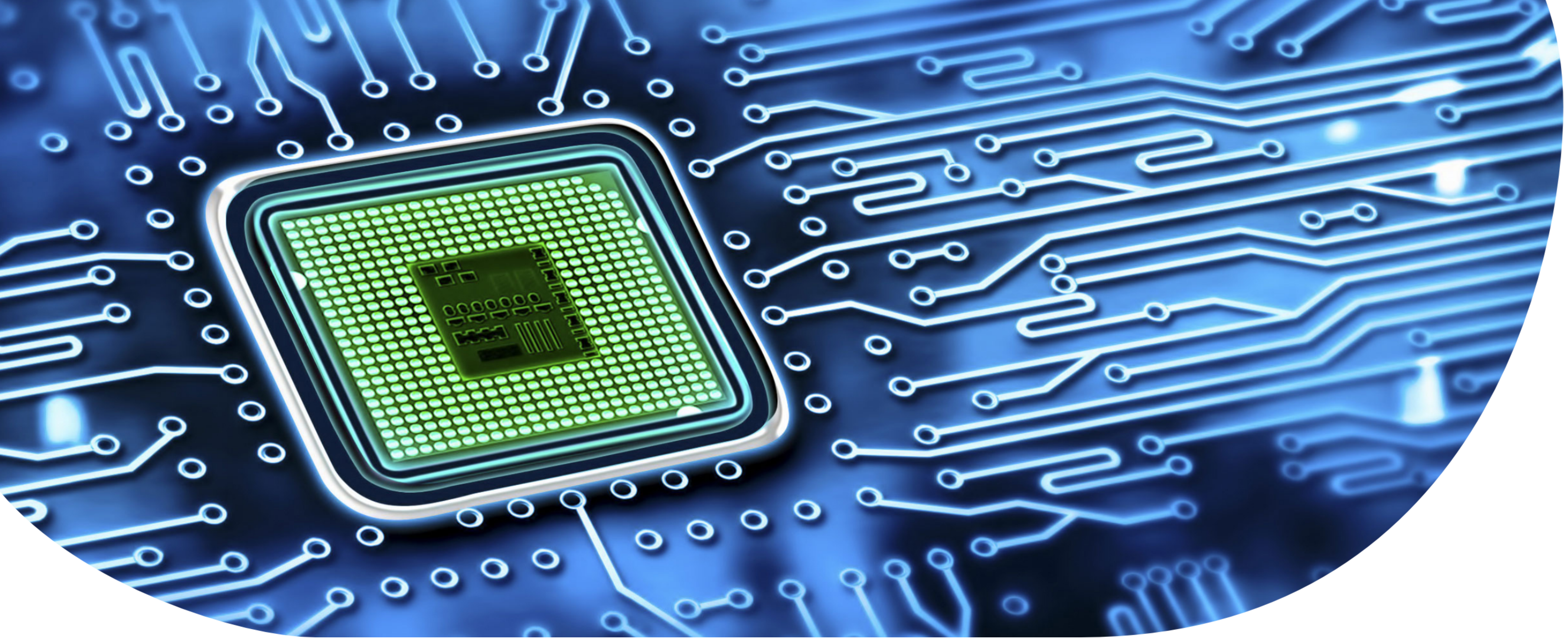

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CS356 Hardware Accelerator Architectures

Research Patterns

Suhaib Fahmy

suhaib.fahmy@kaust.edu.sa

Slides based on “Research Patterns” by Nick Feamster and Alex Gray

General Approach

1. Find a problem
2. Understand the problem
3. Somehow make a plan for a solution, carry it out
4. Review the solution

General Approach

1. **Find a problem**

2. Understand the problem

3. Somehow make a plan for a solution, carry it out

4. Review the solution

Finding Problems

- Hop on a trend
- Find a nail that fits your hammer
- Revisit old problems (with new perspective)
- Making life easier
  - Pain points
  - Wish lists
- “*-ations”
  - Generalization
  - Specialization
  - Automation

Hop on a trend

- Need places to discover trends
- Funding agencies
  - Funded proposals
  - Calls for proposals
- Conference calls for papers
- Industry/technology trends: trade publications

Hop on a trend – examples

- Presently:
  - Large Language Models specifically and Machine Learning generally
  - Quantum computing
  - Self-driving cars/robotics
  - Distributed ledgers/cryptocurrency
- Varying stages, longevity, applicability

Finding a nail for your hammer

- Become an expert at something
  - You'll become valuable to a lot of people
- Develop a system that sets you ahead of the pack
- Apply your "secret weapon" to one or more problem areas
  - Algorithm
  - System
  - Expertise
- "Turn the crank"

Revisiting problems

- Previous solutions may have assumed certain problem constraints
- What has changed since the problem was "solved"?
  - Processing power
  - Cost of memory
  - New protocols
  - New applications
  - …

Revisiting problems – examples

- Machine learning accelerators based on systolic arrays from the 1960s!
- Networking to software-defined networks to in-network computing
- Cryptographic methods with scaling compute capability/quantum

Pain points

- Look to industry, other researchers, etc. for problems that recur
- In programming, if you have to do something more than a few times, script!
- In research, if the same problem is recurring and solved the same silly way, there may be a better way…

Automation

- Some existing problems, tasks, etc. are manual and painful
  - Automation could make a huge difference
  - It's also often very difficult because it requires complex reasoning
- Related to pain points

Automation – examples

- Manual code optimisations for specific architectures to maximize performance
- Generators to automate use of intrinsics for auto-vectorisation
- Automated passes and backends for compiler flows to solve the general case

Wish lists

- What systems do you wish you had that would make your life easier?
  - Lighter VR headset?
  - Faster response time from an LLM?
  - …
- What questions would you like to know the answer to?
  - Chances are there is data out there to help you find the answer…

Generalize from specific problems

- Previous work may outline many points in the design space
- There may be a general algorithm, system, framework, etc., that solves a large class of problems instead of going after "point solutions"
- E.g. highly application-specific accelerators, vs. more generalised architectures optimized for a domain of applications

Specialise a general problem

- Finding general problems
  - Look for general “problem areas”
  - Look for taxonomies and surveys that lay out a problem space
- Applying constraints to the problem in different ways may yield a new class of problems
  - Example: Routing (in wireless, sensor networks, wired, delay-tolerant networks, etc.)

General Approach

1. Find a problem
2. **Understand the problem**
3. Somehow make a plan for a solution, carry it out
4. Review the solution

Exhaustive search

- Collect data
  - Often this can enhance your expertise as a side effect
- Model the problem
  - List all of the constraints to a problem space
  - Consider all of the different angles within your model that you might be able to attack the problem (example: phishing attacks, routing configuration errors)
- Consider many other examples
  - May suggest general framework or approach
  - You may also see a completely different approach

Formalization

- Define metrics
  - Consider ways to measure the quality of various solutions
  - What constitutes a "good solution"
  - Objective functions can be optimized
- Formalization/modeling can lead to simplifying assumptions (hopefully not over-simplifying)
  - Can also suggest ways to attack the problem
  - …or an algorithm itself

Formalization

- Think about what works best for your domain
  - What metrics are of interest at the component and system level?
  - How are the components composed and how do they interact?
  - What methodology can you use to model at the component level and higher?
    - Event-driven, networks of queues, time schedules, etc.

Decomposition

- Given a model, it often becomes easier to break a solution into smaller parts
- Understand each part in detail, and how they interact
- Then revisit the whole

General Approach

1. Find a problem
2. Understand the problem
3. **Somehow make a plan for a solution, carry it out**
4. Review the solution

Consider related problems

- Try to restate the problem, or create an equivalent problem
  - Consider different terminologies and representations
- See if your problem matches a general form already formalized
- Can you use the solution to a related problem?
- E.g. routing in circuit design relates to general path-finding algorithms

Make analogies

- Make an analogy to another problem, then look at its solution
  - “structural transference”
  - look for “symmetries”, or interchangeable parts

Change the problem to a solvable one

- Make simplifying assumptions
  - Violate some of the constraints of the problem
  - Define a sense of approximation to the ideal solution
- Then revisit the original problem
- Make the minimally-simpler problem; then relate the solutions to the two problems
  - “mathematical induction”

Just start, with anything

- Start with a strawman solution, then modify as needed
  - e.g. (in algorithms): Propose a simple algorithm, check its correctness
  - e.g. (in data modeling): Look at simple statistics of a dataset, then dive into anomalies
  - e.g. (in systems): Just whip up some code
- A working simple solution is better than an intractable complex formulation

Consider nature

- Introspection: How does a human naturally solve this problem?
- How does nature solve this problem?
- E.g. neural networks somewhat nature inspired, vision algorithms
- E.g. various optimization approaches: genetic algorithms, ant colony optimization, etc.

Work backward from the goal

- Visualize the solution, and what it must look like, or probably looks like
- See what's needed to get there
- Consider all the solutions that can't work

Solve a part, or each part

- Solve each part separately, then stitch the solutions together
    - Start with the part which is most tractable
    - "divide-conquer-merge"
    - Be careful: it's always best to avoid separate objective functions when possible
- Perhaps finding a good solution to a part is a good problem in itself

Think in speech or pictures

- Use dialogues with others
  - Or yourself
  - Talk to people who approach things differently from yourself
- Draw pictures
  - Add auxiliary elements, to be able to relate to other problems/solutions
- Maths is not the only way to reason about ideas

Come from all angles

- Keep coming with a new twist on the problem
  - Break out of a thinking pattern or dead end
  - A new twist renews motivation
  - "Where there's a will, there's a way"
- Keep track of all your ideas and partially-completed paths

Let your subconscious work

- Immersion
- Stay relaxed
- Or: use deadlines to force shortcuts
- Wider reading can stimulate new approaches to thinking

General Approach

1. Find a problem
2. Understand the problem
3. Somehow make a plan for a solution, carry it out
4. **Review the solution**

Look back at your solution

- Check that it really works
  - If it works, note the key to why, more abstractly
  - Were all of the constraints, difficulties, and facts used and accounted for
- Try to improve upon it
  - Can you achieve the same thing more directly or easily

What else can your solution do?

- Now you have a hammer
- Can you use the solution for some other problem?
  - A more general form of the problem?
  - An interesting special case?
  - A related problem or analogous problem?

Making a "theory"

- If you're very successful, you may have a "theory" = a framework for characterizing problems and/or solutions
  - Says when it applies, when it doesn't
  - Characterizes the hardness of different problems
    - May identify simple special cases
  - Characterizes the quality of different solutions
    - How long it takes, amount of resources it uses
  - Show/characterize solution meeting criteria
    - correctness, convergence, etc.