PARAGON
PARALLEL ARCHITECTURE GROUP @NORTHWESTERN

Grade Level:	*11th -12th*
Subject:	*Comp. Sci. & Eng.*
Time needed:	*120 minutes*

How to Design a Microprocessor

Nikos Hardavellas and Zhenduo Zhai

Purpose

Technology is already tightly integrated into our everyday lives. The computing power we have at our fingertips today allow us to stream and decode movies online, meet colleagues in teleconferencing rooms, shop at online stores, compose music, produce videos, and perform countless other activities. Our ability to harness the power of computation has practically become a prerequisite to breakthroughs in scientific knowledge, from rational drug design to new materials, to data-driven discovery in chemistry, astrophysics, economics, sociology, and almost every other scientific endeavor.

Underneath all these newly found capabilities that augment human experience is the microprocessor. The microprocessor is tasked with executing the computer programs that enable us to perform all these activities. Designing a microprocessor, however, is a challenging task. Like any other physical device, the microprocessor is subject to physical constraints. We cannot make it too big without making it impractical to manufacture. The more electrical power we give it the faster it will compute, but we can supply only limited power to a microprocessor before it becomes too hot and burns out. The higher power consumption of a microprocessor will also lead to higher environmental cost – is the additional performance worth it? The microprocessor needs access to data to perform useful work, but we can bring data to it at only a limited rate because we cannot drive the wires any faster or pack more of them in the given package. If we give more power to the microprocessor and run it faster, what good will that do if most of the time it simply sits there idle, twiddling its digital thumbs, waiting for data to arrive?

Given all these complex interdependencies, how can we best design a microprocessor? This lesson plan illustrates the typical tradeoffs underlying microprocessor design and guides the students through experiments that can lead them to an answer.

Overview

In this lesson students will learn some basic concepts in processor architecture. Using a graphical user interface of a parameterized processor model from recent computer architecture research, the students will run simple experiments in which they can modify the architectural parameters of a microprocessor model and estimate their impact on performance, area occupancy, power consumption and off-chip data rates. These estimates, in turn, will shed light on the tradeoffs that arise in microprocessor design, and guide the students to an optimal design that conforms to the given constraints.

Student Outcomes

- Students will learn the basic parameters that computer architects optimize for when designing a microprocessor, and the physical constraints that limit the design space.
- Students will be able to utilize a mathematical model to estimate the impact of processor design parameters in performance, power consumption, area occupancy and off-chip data bandwidth.
- Students will be able to interpret results returned by the model, and design scientific experiments to discover real-world tradeoffs in microprocessor design.
- Students will be able to utilize their discovery of these tradeoffs to arrive at a near-optimal microprocessor design that conforms to the given physical constraints.

Partially supported by NSF award CCF-1453853, *CAREER: Energy-Efficient and Energy-Proportional Silicon-Photonic Manycore Architectures.*

Standards Addressed

NGSS:

Science and Engineering Practices (SEP):

- Constructing Explanations and Designing Solutions: Design a solution to a complex real-world problem, based on scientific knowledge, student-generated sources of evidence, prioritized criteria, and tradeoff considerations.
- Constructing Explanations and Designing Solutions: Evaluate a solution to a complex real-world problem, based on scientific knowledge, student-generated sources of evidence, prioritized criteria, and tradeoff considerations.
- Using Mathematics and Computational Thinking: use mathematical models and/or computer simulations to predict the effects of a design solution on systems and/or the interactions between systems.

Disciplinary Core Ideas (DCI):

- **HS-ETS1-2** – Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.
- **HS-ETS1-3** – Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.
- **HS-ETS1-4** – Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Cross Cutting Concepts (CCC):

- Systems and system models.
- Constructing Explanations and Designing Solutions.

Time

2 class periods or approx. 2 hours

Level

11^{th} - 12^{th} grade, Computer Science and Engineering

Materials and Tools

- ✓ Access to an Intel-based computer running one of the following operating systems
  - Apple OS-X 10.10 (Yosemite) or later, or Microsoft Windows 8.0 or later
- ✓ Access to the multicore designer modeling tool and guides at http://paragon.cs.northwestern.edu/K12/Multicore_Designer/Multicore_Designer.zip
- ✓ Access to the library dependencies required to install the modeling tool (see User Guide)
  - Mac: XQuartz X11, Homebrew, GTK+2; Windows: MSYS2, GTK+2
- ✓ Access to a graphing tool, such as Microsoft Excel, Numbers or Google Sheets

Preparation

Download and unzip the Multicore_Designer.zip file. It contains executable files for Windows and Mac computers, along with a User Guide. Install the multicore_designer application on the students' computers following the installation procedure described in the User Guide. Ensure that students have access to a graphing tool. Preparation should be done ahead of the class session to minimize downtime.

Prerequisites
Students should feel comfortable generating graphs from arithmetic data and interpreting them.

Background
Underneath all the newly found capabilities that augment human experience is the microprocessor. The microprocessor is tasked with executing the computer programs that transform raw data into actionable knowledge. These programs are executed in what is commonly known as a **core**, a circuit that executes a program by running one instruction after another. A processor can pack multiple such cores on a single processor chip, also known as the processor *die*. A processor with many cores on chip is known as a **multicore** processor. A multicore processor can execute multiple independent programs simultaneously, thus providing several times higher performance. In general, the more cores a processor has, the faster it can execute multiple programs. Designing a multicore microprocessor, however, is a challenging task.

A core requires access to data to perform its computations. Without data to operate on, cores cannot perform useful work, and just sit idling, doing nothing, waiting for data to arrive. Data are typically stored in far-away components such as the **main memory** (DRAM), which is implemented on its own circuits outside the main processor chip (hence accesses to such data are called *off-chip accesses*). Main memory is slow. Within the time required to access a single piece of off-chip data, the processor can execute hundreds of instructions. To help the processor run faster, modern systems also put a smaller memory, called a **cache**, on the same chip as the processor. Because a cache is implemented on the same chip as the cores, it is 10x faster to access. The hardware orchestrates movements of data between the on-chip cache and the off-chip main memory, with the aim to bring data close to the cores ahead of their access.

The bigger the cache, the more data it can keep close to the cores. However, this also means it occupies more area on the silicon chip. This area is limited. Thus, if a computer architect puts a very large cache on chip, only little area will be available for cores. This gives rise to the first physical constraint: **the area constraint**. There is only a limited chip area on which to put cores and cache; more cores allow more programs to run in parallel, improving performance, hence there is an incentive to use the die area to put as many cores as possible. At the same time, a bigger cache allows for more data to be close to the cores improving performance, so there is an incentive to put bigger caches. How much of the area should be used for cores and how much for a cache?

To simplify design, the circuits that comprise a core operate in lockstep, following a clock. Processor clocks today operate at a frequency of a few GHz, i.e., they “tick” a few billion times per second; at each tick a processor may be able to complete an instruction. This clock frequency is called the **processor clock frequency**. The faster the processor clock frequency, the faster each core can operate. However, to be able to run a circuit at a higher clock frequency, the circuit must also receive a higher voltage. Thus, running a processor at higher frequency also means running the processor at higher voltage, in which case the processor will consume more power. Unfortunately, power is also a limited resource. A portion of this power is dissipated on the surface of the chip as heat and circuits cannot operate reliably past a certain temperature. However, our ability to cool-off chips economically is limited. This gives rise to the second physical constraint: **the power constraint**. There is a maximum limit of power that a processor can consume because we cannot cool off the surface of a chip fast enough. The higher the clock frequency the faster a core can run, but the more power it will also consume. Thus, we can put on a processor a few, really fast cores, or many slower ones; which one is better and how to pick the best number of cores and frequency that “fit” in our power budget? To make matters worse, caches consume power too, adding to the complexity of the constraint.

Finally, as we said earlier, cores require access to data to perform useful work. The processor must bring such data from off-chip memory to the on-chip cache, and then let the cores read them. This transfer of data, though, happens at a very low rate. Processor chips interface with the outside world through the *off-chip pins*, which we can think of as a special type of a wires. We can only pack off-chip pins at a low density, and we can only drive them at a relatively slow pace to ensure correct transfers. Thus, data come from off-chip memory at a very low rate, known as **off-chip bandwidth**. This gives rise to the third physical constraint: the **off-chip bandwidth constraint**. The faster the processor runs, the faster its cores will want to access data, hence the higher the rate that the processor must bring data from off-chip memory. However, the off-chip bandwidth is limited. Making a processor faster than a certain level is ineffective, as it will not result in a performance increase; it will just still idle, waiting for data to arrive. The off-chip bandwidth is also shared among all cores, so the limited off-chip bandwidth affects both the clock frequency of the cores, and the number of cores that it can support. To make matters worse, the larger a cache is, the less off-chip bandwidth the processor will need, for example, a cache that is as big as the executing program's dataset size will only need to bring the data once from main memory, while one that is $1/10^{th}$ the dataset size may need to bring the same data multiple times.

The last example also shows that the executing application impacts the demands on limited resources: the amount of data a program uses, how many arithmetic computations are between data reads and writes, etc., all affect how much off-chip bandwidth is needed, how big the on-chip cache should be, how fast the processor clock should be, and how many cores to place on a chip.

Tradeoff Experimentation

After discussing the background, students can use the muticore_designer application to design and perform simulation experiments aiming to understand these tradeoffs and make design decisions. To understand how the application works, the students should read the User Manual that comes with the application. For this lesson, the students need not understand the specifics of the modeled applications (DB2/TPCC, DB2/TPCH, Apache/Web), transistor technology (LOP, HP/LOP/ HP), semiconductor technology (65nm, 45nm, 32nm, 20nm), core technology (Niagara, FPGA, ASIC, ARM), or memory technology (planar and 3D memory). While the multicore_designer tool allows for experimenting with these parameters too, they are advanced concepts that are not necessary in this lesson.

An example set of experiments the students can perform is the following. First, the students should set the pull-down parameters in the tool to some values and leave them unchanged for the duration of the experiments. The example graph below is for the DB2/TPC application, Plain memory, HP transistors, Niagara cores, and 20nm semiconductor technology. These parameters correspond to the technology specifics that are not necessary to discuss for this lesson. Then, the students can select their physical constraints. The experiments described here assume 310 mm^2 area limit, 130 W power limit and 1000 GB/s off-chip bandwidth limit (essentially, no bandwidth limitation—for now; we will revisit this later). Then, the students can set the supply voltage (V_{dd}) and clock frequency to certain values and "lock" them, i.e., tick the V_{dd} box. Here we assume 0.36 V and 2.7 GHz.

Then, the students can perform a sweep over the possible values of cache (on-chip memory) size by changing the values of the cache size slider. Each slider value we set forces the tool to calculate the remaining free parameters to arrive at a design with the highest performance, in this case the number of cores. For example, for 1 MB cache the tool will calculate that the most performant design of a multicore will have 96 cores. As we increase the cache size, the number of cores for the most performant design drops, as more cores do not "fit" within the physical constraints. At 40 MB cache the performance drops to zero. This is because such a design is not feasible: even only 1 core will bring the power

consumption to 136.89 W, which is above our 130 W limit (the students can see this by ticking the cache size box to lock its size to 40 MB, and then sliding the cores slider to 1 core). The curve we obtain by this experiment shows the tradeoff between core count and cache size for a given voltage-frequency setting and technological parameters. The same set of experiments can be repeated for different voltage-frequency combinations, including the maximum frequency setting (0.91 V, 10 GHz). The result is a graph similar to Fig.1, which illustrates the tradeoffs between core count and cache size for each voltage-frequency setting under area and power constraints. Running at maximum frequency (black line) can only support up to 4 cores; any additional core will exceed the power budget of the chip (a.k.a. the power wall). We can only support more cores by running them slower.

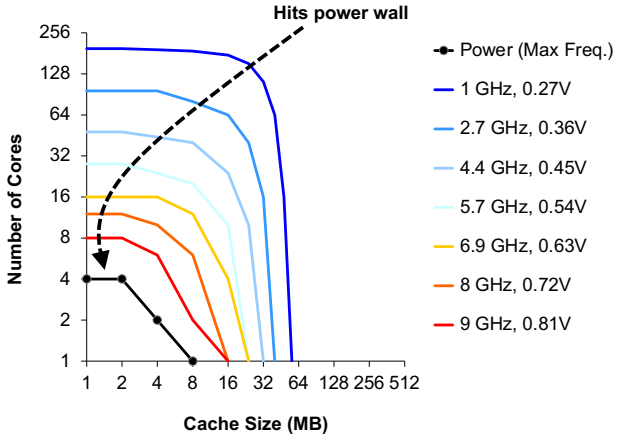

Figure 1. Area- & power-constrained designs across clock frequencies.

The experiment thus far does not consider any off-chip bandwidth limitations. Our bandwidth constraint setting was 1000 GB/s, which none of the configurations exceeds. This, however, is not realistic. We can approximate reality by placing a bandwidth limitation, say 76 GB/sec. Then, it is no longer possible to get peak performance with a 1MB-cache 176-core processor at 1GHz. Rather, the performance peaks at 48 cores—any additional core requires bandwidth that cannot be provided. The 1GHz bandwidth constraint curve in Fig.2 (dotted dark blue line) shows the core/cache-size settings that lead to the maximum performance for a 1GHz processor.

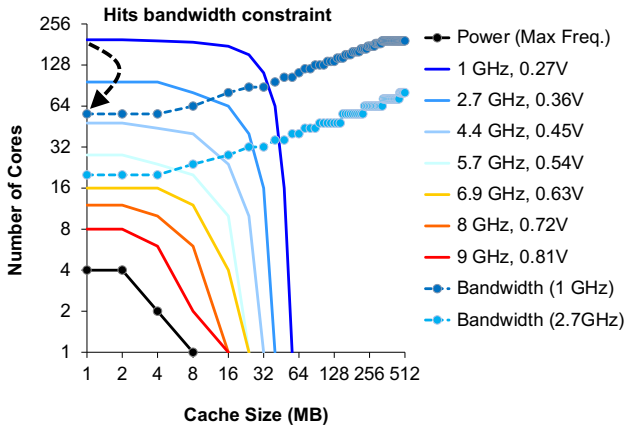

Figure 2. Additional bandwidth-constrained designs.

To obtain the bandwidth-constrained curve we set bandwidth limits to 76 GB/s and remove the power limits by placing a 1000 W power constraint. Then, we slide the cache size from 1MB upwards and for each cache size setting the modeling tool identifies the number of cores that give the highest performance. Similarly for the bandwidth-constrained curves for other clock frequencies. The gap between the area- and power-limited 1GHz processor (solid deep blue) and the bandwidth-constrained 1GHz processor (dotted blue) shows designs that,

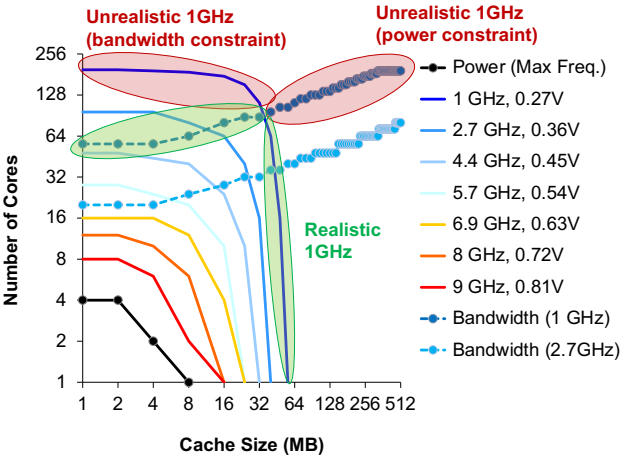

Figure 3. Identifying realistic and unrealistic 1GHz designs.

while within the area and power constraints, they exceed the bandwidth constraint. These designs are unrealistic. Similarly, the designs that satisfy the bandwidth constraint but not the power one, are also unrealistic. Figure 3 shows the unrealistic design points for a 1GHz processor with read-hued ovals, and the realistic design points with green-hued ovals.

Now, for each one of the x-axis values (cache size design points), we can use the performance estimates already obtained from the modeling tool to devise the number of cores that will result to the highest-performance across all frequency-voltage combinations subject to all constraints. Adding this *peak performance* curve to the graph results in Figure 4. The peak performance curve illustrates the complete tradeoff between cache size, number of cores, voltage, frequency, and performance, while subject to area, power, and bandwidth constraints.

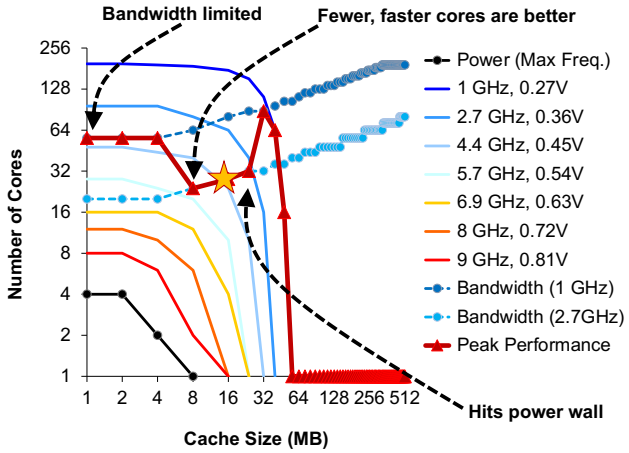

Figure 4. Identifying peak-performance designs.

As the figure shows, the first few designs (at 1-4 MB cache) are primarily constrained by bandwidth. The cache is simply too small to keep all the cores fed with data; more data need to come from off-chip memory, severely stressing the available off-chip bandwidth. At 16MB cache the processor remains bandwidth constrained, but it jumps from the 1GHz to the 2.7GHz bandwidth constrained line. Fewer but faster cores apparently provide higher performance, than more (but necessarily slower) ones. At 32MB cache the peak-performance designs jump back to the 1GHz line. The system is again hitting a power wall; it can no longer efficiently support both a large cache and fast cores. Rather, more but slower cores are now better.

Overall, the highest performance is obtained by a design with 24MB cache and 32 cores running at 2.7GHz. This design is indicated in the figure by a yellow star. Even more interesting is that this design is very small. Only a fraction of the 310 mm^2 chip area is used. What happened to the rest? The power constraint is so severe that we are much better served by diverting all the available power to a fraction of the silicon area to implement a few reasonably fast cores and a sensibly sized cache; the rest of the silicon area is powered off, or dark. This phenomenon has been dubbed **dark silicon.** Combating dark silicon has spurred a significant amount of research in accelerator-based and specialized computing, and is a leading reason for the recent renaissance period in computer architecture.

Overall, our experiments illustrate some of the basic design decisions that microprocessor architects face, and the interplay between processor attributes and physical constraints that impact performance.

Teaching Plan

1. Provide a handout of the background information in this lesson plan and the User Guide to students (paper copy or digital).
2. Discuss for about 30 minutes. Ask students to experiment with the multicore_designer modeling tool to familiarize themselves with its interface and output.
3. Guide the students through three sets of experiments similar to the tradeoff experiments described above

   a. Set 1: devise and graph the area-/power-constrained curves across frequencies (Fig.1). Adjust for time by limiting the frequencies to explore to about 4 interesting ones (e.g., maximum 0.91V 10GHz, fast 0.63V 7GHz, medium 0.36V 2.7GHz and slow 0.27V 1GHz).
   b. Set 2: add to the graph the bandwidth-constrained curves across frequencies (Fig.2). Could perform the experiments for all frequencies used in set 1, or (if time is an issue) at least the frequencies that lead to peak performance designs (e.g., 1GHz and 2.7GHz).
   c. Set 3: add to the graph the peak-performance curve across all examined frequencies subject to all constraints (Fig.4)
4. Give 30 minutes for each experiment set for students to work in pairs or small groups, monitoring students for questions and misunderstandings.
5. Wrap up the experience with a class discussion. There will surely be rabbit trails to follow, which is encouraged if time allows!
6. There are additional resources linked at the end of this document. These may help for directing students that have particular interest.

Assessment

The students can be assessed separately for each one of the experiment sets by assessing the correctness of the graphs they obtain.

- Set 1: assess the correctness of the student's version of Fig.1.
- Set 1 question: explain why the higher the frequency is, the fewer cores we can support.
  - Answer: cores running at faster clock frequencies will consume more power; they cannot be supported because they exceed the power limit of the chip.
- Set 2: assess the correctness of the student's version of Fig.2.
- Set 2 question: for a 1GHz clock frequency, what are the realistic processor designs?
  - Answer: shown in Fig. 3 in this handout.
- Set 3: assess the correctness of the student's version of Fig.4.
- Set 3 question: what is the highest-performance design across all constraints and frequencies?
  - Answer: for the example in this handout, a design with 24MB cache and 32 cores running at 2.7GHz. The tool can also automatically provide the best design across the board by clicking the "Best Performance" button (ensure no boxes are ticked).

Additional Information

Advanced students could also read the following two articles, which explain these trade-offs in more detail. The articles can be downloaded from http://paragon.cs.northwestern.edu/#Publications.

- *The Rise and Fall of Dark Silicon.* N. Hardavellas. USENIX ;login:, Vol. 37, No. 2, pp. 7-17, April 2012. This article is geared toward a computer-literate audience but not experts in the field.
- *Toward Dark Silicon in Servers*. N. Hardavellas, M. Ferdman, B. Falsafi and A. Ailamaki. IEEE Micro, Special Issue on Big Chips, Vol. 31(4), pp. 6-15, July/August 2011. Also, IEEE Micro Spotlight Paper at Computing Now, February 2012. This article is geared toward computer architects.