Communicating Research to the General Public

At the March 5, 2010 UW-Madison Chemistry Department Colloquium, Prof. Bassam Z. Shakhashiri, the director of the Wisconsin Initiative for Science Literacy (WISL), encouraged all UW-Madison chemistry Ph.D. candidates to include a chapter in their Ph.D. thesis communicating their research to non-specialists. The goal is to explain the candidate's scholarly research and its significance to a wider audience that includes family members, friends, civic groups, newspaper reporters, program officers at appropriate funding agencies, state legislators, and members of the U.S. Congress.

Over 50 Ph.D. degree recipients have successfully completed their theses and included such a chapter.

WISL encourages the inclusion of such chapters in all Ph.D. theses everywhere through the cooperation of Ph.D. candidates and their mentors. WISL is now offering additional awards of $250 for UW-Madison chemistry Ph.D. candidates.

Wisconsin Initiative for Science Literacy

The dual mission of the Wisconsin Initiative for Science Literacy is to promote literacy in science, mathematics and technology among the general public and to attract future generations to careers in research, teaching and public service.

UW-Madison Department of Chemistry
1101 University Avenue
Madison, WI 53706-1396
Contact: Prof. Bassam Z. Shakhashiri
bassam@chem.wisc.edu
www.scifun.org

Modeling and Designing Secure Tightly-Coupled Accelerators in CPUs

By

David J. Schlais

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2021

Date of final oral examination: 12/15/2020

The dissertation is approved by the following members of the Final Oral Committee:

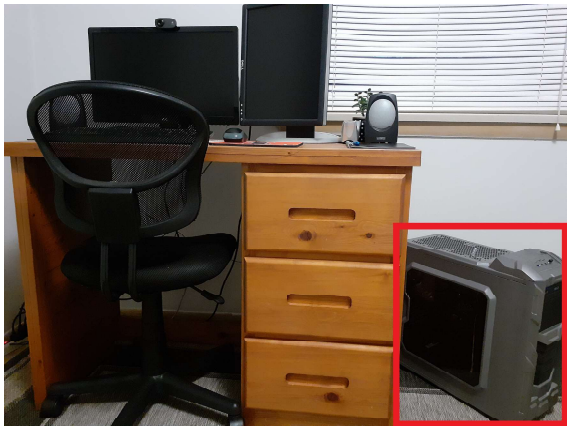
Mikko H. Lipasti, Professor, Electrical and Computer Engineering
Mark D. Hill, Professor Emeritus, Computer Sciences
Gurindar S. Sohi, Professor, Computer Sciences
Joshua San Miguel, Assistant Professor, Electrical and Computer Engineering

1 EXPLAINING MY PHD RESEARCH TO THE GENERAL PUBLIC

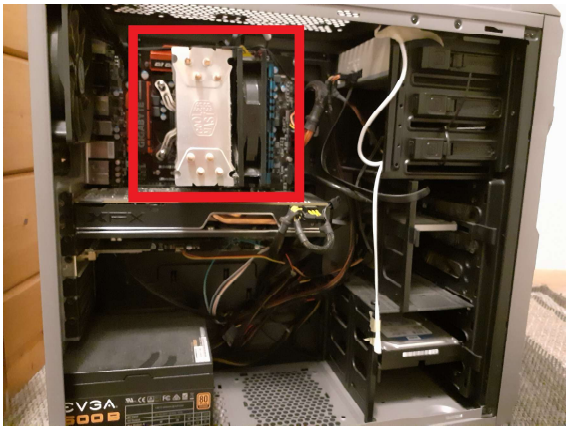
This section is written as part of the Wisconsin Initiative for Science Literacy (WISL) program to explain PhD Research to the general public. Most of the work contained further in this defense, as well as the majority of PhD research in the scientific community, is focused towards an audience with a specific technical background. However, many of my friends, family, as well as others that are excited to learn more about what I have been doing for the past six years should be able to see the work I've conducted without having to go through all the jargon, technical details, and background of numerous Computer Engineering classes. Additionally, the scientific community should strive to excite future young minds to pursue the fields and see the excitement that research allows without digging through hundreds of pages of technical detail. I am excited to be a part of this initiative to dedicate a chapter of my research to explain things through the lens of the everyday person not studying Computer Engineering or Computer Architecture.

1.1 Background

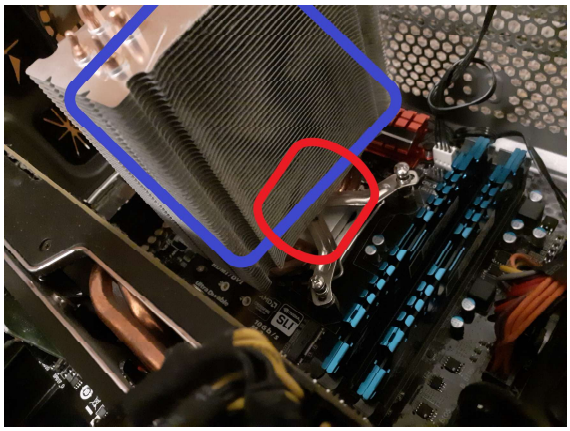
What do you think of when you hear the word "computer?" You probably think of a big rectangular box next to a desk, laptop, tablet, or maybe a large company datacenter/server. These are used in our everyday life to solve complicated problems, store our information, or entertain us through video games or streaming entertainment. However, most of the things we think that the whole laptop, desktop, smartphone, etc. are doing are in fact done within the processor. You may have heard the term CPU before, which stands for Central



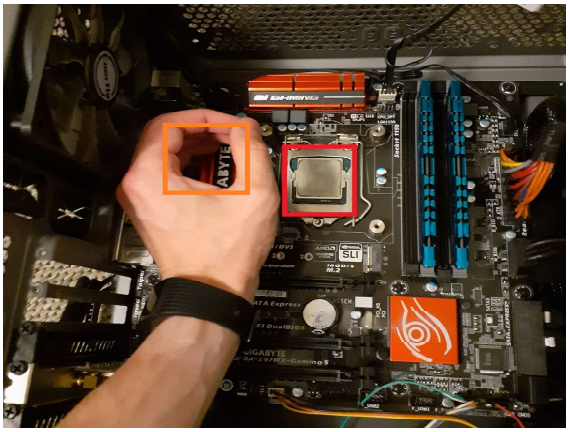
(a) Desktop – The actual desktop computer is shown in the red box. The monitors on the desk only show you what the computer is working on.



(b) Inside a desktop computer. The processor sits under the big radiator shown in the red box.



(c) A different angle of the CPU cooling radiator (blue square), which sits on top of the processor (red square). The radiator keeps the processor from overheating.



(d) The processor is only visible after removing the radiator. The size of the the computer's brains, or processor (red square), compared to the size of my hand (orange square).

Figure 1.1: Photos of my desktop computer, zooming in from the desktop housing (a), to inside the desktop housing (b), all the way to the actual processor (d). Computer architects are responsible for designing the circuitry within the processor shown in photo (d).

Processing Unit. Whenever you do something on your computer, the CPU is almost always the part responsible for making your commands happen. The CPU determines when and how your "enemy" or "opponent" moves in your video games, determines what to do next based on your mouse movements, solves difficult math problems, searches for files on your computer, determines what to display on your screen when you surf the web, and much more. The CPU can be thought of as the "brains" of a computer. For simplicity, whenever I use the terms "CPU," "processor," and "computer," think of the brain. It may be surprising to learn that this brain is only about the size of the square created by your finger outline if you touch your index finger to your thumb on the same hand (See photos in Figure 1.1)! My field of research is in "computer architecture," which is responsible for designing and upgrading this processor.

Computers operate by running "computer programs," also known as "software." Software tells the processor what operations (also called instructions) to perform, and in what order. But what instructions are available for the software programmer to use? This is where "hardware" comes in, which is the focus of computer architecture. Hardware is a group of electrical circuits that performs those instructions, such as addition, subtraction, multiplication, comparisons, reading data, and writing data. A few examples are things like "add these two numbers together," "move the number from location A to location B," or "tell me if number C is less than number D." That's it. You might be asking yourself right about now "but how can a computer solve so many complicated problems so quickly if all it can do is simple operations?" The answer is because it can do these operations extremely quickly. A modern processor can compute a few billion of these operations every second. So, although the operations are simple, by breaking down big problems into a series of

these small operations, the processor can still solve the problems much faster than we can. The processor has no idea what it is doing – it will blindly do the operations it is told to do. That is why one philosophical view is that computers are not very smart, they're just fast at what they do. Once the hardware has a circuit built to perform an instruction, the programmer can group sequences of these instructions together to perform complex tasks.

As a computer architect, I work on both designing individual circuits to perform operations, and linking existing circuits together in new ways to be better than prior designs. I organize the circuits and components to detail how a new processor should be built, just like building architects design the details of a new building's blueprint. Computer architects attempt to make the processor faster, more powerful in its computation ability, use less energy, and modify anything to optimize it for both today's and tomorrow's needs. Computer architects also have to be aware of and involved with the software community to understand what requirements and what building blocks they wish they had in future processors.

1.2 Motivation Behind Accelerators

Processors are extremely powerful because they are extremely general-purpose. By this, I mean that there is an infinite number of computer programs that a processor will be capable of running. Software can be updated, and the processor will still be able to run it. You can create a new program, and the processor will be able to run it. It is only up to the developer's imagination of how to combine these operations together, the computer will be able to run it. It is flexible enough to run any valid computer program across any domain,

whether video games, company software, warehouse management, medical record storage, etc. However, this flexibility comes at some cost. By having only basic building blocks, sometimes you need very long sequences of building blocks to perform a task. If that task is performed often enough, the CPU is simply repeating the same blocks over and over again. However, if the computer knew how to perform the task (rather than only the small building blocks), it could perform that task through a single command, rather than by a sequence of many commands. This is where "accelerators" come into play. An accelerator is a circuit that is specialized to do a single task extremely well. The benefit of accelerators is that they are fast and efficient. Their downside is that they only know how to perform that one task. If the same task needed to be even slightly adjusted or modified, the accelerator cannot perform that task at all, and must go back to using basic instructions. This is why accelerators are typically only built when it is fairly certain that (a) this task is used often enough to justify adding it (otherwise it will just sit doing nothing most of the time), and (b) that the task will not need to be updated or changed in the near future.

When accelerating large tasks involving lots of work (such as taking the video captured by your smartphone camera and converting it into a movie file), accelerators are built far away from the processor core (the "heart" or center of the processor), and are called loosely-coupled accelerators. However, when accelerating fine-grained tasks (something like renaming a computer file) where the time it takes to perform operations cannot afford to move data far away from the processor, these need to be tightly-coupled accelerators. Being tightly-coupled makes data movement faster and should be justified only if used frequently, as it takes the place of other circuits that could be close to the core. My work focuses on tightly-coupled accelerators, as it is a new and emerging type of accelerator that

has the potential to provide significant speedups for many emerging computer needs, such as machine learning, artificial intelligence, and server optimization.

1.3 Contribution 1 – Performance Modeling of Tightly-Coupled Accelerators

I started working on tightly-coupled accelerators because of their potential future impact. There is a large amount of excitement in the computer field around artificial intelligence (AI) and machine learning (ML). These are software techniques to try to predict information about either the here-and-now or the future based on information seen in the past. AI and ML work by showing thousands or millions of examples to a computer, telling the computer what the example is, and having the computer try to find similarities in each of the examples. Eventually, the computer can correctly understand the details or "features" that made that example what it was. For example, a computer could be shown several different CAT scan images of cancerous and non-cancerous regions. After seeing enough examples that doctors have determined cancerous and non-cancerous, the computer will determine similarities between the cancerous images, and similarities between the non-cancerous images. The idea is that when the computer is presented with a new CAT scan, it can say whether it looks more like the cancerous or non-cancerous images it has seen in the past. Websites like Amazon, Netflix, or Youtube can also use AI and ML to predict what movies to recommend to you based on what you have stated you like and don't like.

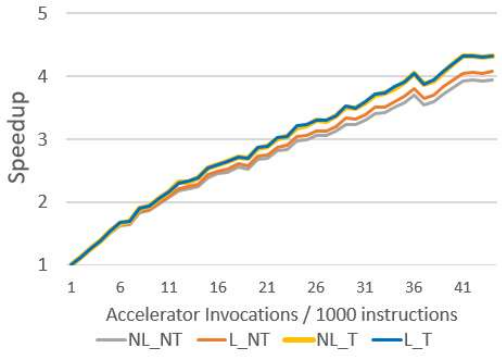
All of these AI and ML tasks can be calculated by the computer through a series of

matrix-matrix multiplications. Simply put, the computer performs mathematical operations to understand similarities and dissimilarities. Since many different AI and ML tasks all perform these mathematical operations, it makes sense to build an accelerator to perform this task. However, as computer architects, we need to decide how much faster this accelerator would run compared to the CPU as it runs today. This task is more difficult than it may originally seem. Processors have become more complex over the years. Computer architects have optimized processors to run instructions out-of-order as well as predict what the computer program will attempt to do next. The computer processor can perform multiple computer program instructions simultaneously before it even knows if it should perform those instructions. An example might be predicting the value of a complex mathematical formula before the final calculation is finished. The reason the processor does this is because correct predictions make the program run faster. In the end, any mistakes the processor made in guessing (called incorrect execution) will always be undone, ensuring the program still ran correctly.

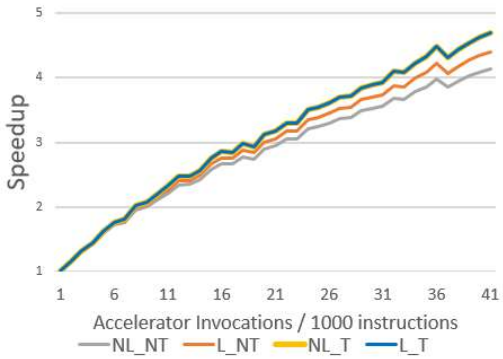
However, having a complex processor brings challenges for computer architects to integrate additional accelerators into the processor. A complex way for an architect to address this issue would be to design an accelerator that can always operate no matter what else is going on around it, and be able to recover from any mistakes made by the processor in guessing where it should go next. A less complex way for an architect to address this issue would be to design an accelerator that waits for the processor to be certain that the task should be performed, and only resume its guessing work after the accelerator has finished its task. Having this 'wait if unsure' policy means that the accelerator never has to correct mistakes, but waiting will slow down the processor from useful work it could have

been doing while the accelerator ran.

The architect could design multiple complex and simple accelerator versions, see how they do, and decide which one to use in the next processor. However, designing accelerators that will not be used takes time and will waste several months of engineering work. Additionally, to actually build processors/accelerators costs millions of dollars per different design. Throwing out designs after building them would waste lots of money. For this reason, computer architects use simulators. These simulators can be thought of as a computer program that is acting like a computer. So really, it's like running a computer within your computer. The reason this is useful is that you can make modifications to your simulated computer without having to build or pay for a new computer. The downside is that you have to tell the program every detail about your simulated computer, how the accelerator interacts with the other components around it, and it runs thousands of times slower than the computer itself. Also, the architect needs to know enough about the



(a) Simulator-based estimation of tightly-coupled accelerator speedup.



(b) Our mathematical model estimation of tightly-coupled accelerator speedup.

Figure 1.2: Estimated performance speedup for a tightly-coupled accelerator. Our model is slightly different than the simulator results, but the overall trend and insights are maintained. The detailed simulator took over 100 hours, and our mathematical formulas took under 5 minutes.

computer simulator to be able to make the modifications required to even test their new designs. This can take hundreds of hours to complete.

My contribution is to create mathematical formulas to estimate the impact of different designs on the execution time of any specific program. Figure 1.2 shows the output from our mathematical formula as compared to the detailed simulator. Instead of building the hardware for millions of dollars, or spending dozens of hours modifying a simulator, an architect can use the formulas that I've created in order to get a good estimation of the overall speedup of various designs in the matter of seconds or minutes. Although these mathematical formulas aren't as exact as the detailed simulator, by having a good estimate, the designer can be confident which design is worthwhile to try to model in the simulator or to build without wasting time focusing on designs that would have been quickly thrown out through simple mathematical estimations. This allows designers to potentially save millions of dollars and hundreds of engineering hours, allowing for a much more rapid time to deploy these accelerators into real-world models!

1.4 Contribution 2 – Removing Unnecessary Data Movement in Tightly-Coupled Accelerators

So far, I've been talking about processors performing calculations, such as doing addition, multiplication, or matrix-matrix multiplication operations. However, in order to perform these operations, the data that is being operated on (also called operands), needs to be supplied to the units doing the computation. This can be thought of as a blender making a

smoothie – the blender will mix all the ingredients, but you have to bring the ingredients to the blender for a smoothie to be possible. All data in a computer is stored through memory. Memory is simply the storage of numbers. Typically, in order to perform operations in a processor, data is transferred from memory into something called a Register File (RF). Anytime an operation is performed, the operands must come from the RF, and if the data is not currently in the register file, then it must be brought from memory into the RF. The RF can only hold a certain number of operands, so if there is not enough space, then something needs to be removed from the RF (and back into memory if it needs to be saved) before a new value can be brought back in. The RF gains the most usefulness when a value is used many times in the RF before it is removed. This eliminates the need to keep transferring the value to and from memory.

A second analogy could be someone operating a carnival dart game, where small and large prizes can be won. All prizes are originally kept in the back room, which in our analogy is memory. The worker brings in a basket from home, which is the RF in our analogy. Contestants are more likely to win small prizes, so the worker fills their basket with two boxes, one for each small prize – pencils and rulers. In reality, there might be dozens of items in the RF, but for simplicity, we'll say there are only two. When a contestant wins a small prize, the worker simply gives the small prize to the winner from the basket without having to go into the back room. However, when a contestant wins a large prize, like a stuffed animal, the worker goes to the back room to get the prize. If the worker is required to give all prizes from the basket, which can only fit two boxes, they might remove the ruler box to replace it with the stuffed animal box. However, as more small prizes are won later on, the worker will have to go to the back room again to replace the stuffed

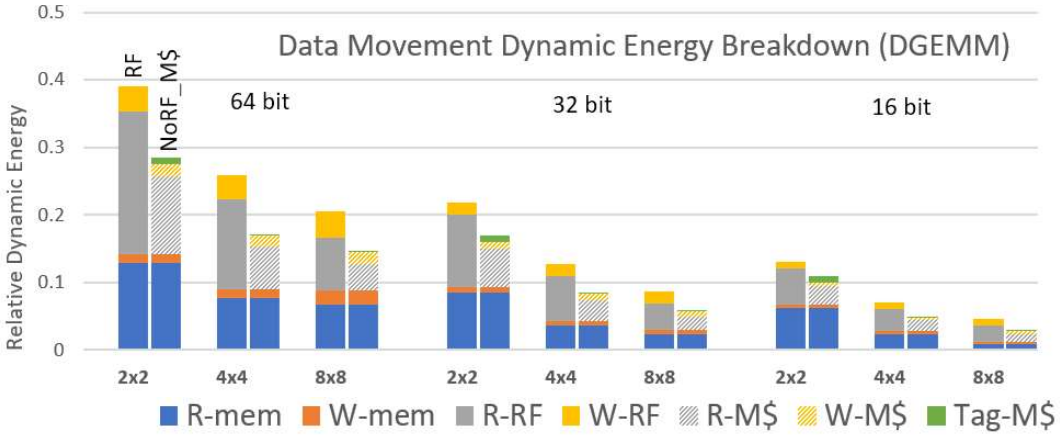


Figure 1.3: Energy consumption moving data in tightly-coupled accelerators using the old method of the Basket/RF (left bar per pair) compared to using our proposed method (right bar per pair). Lower is better. Overall height is decreased for every pair, showing a reduction in energy consumption, which is good.

animal box with the ruler box. Here, the requirement to pass prizes from the basket forced the worker to remove an item they knew they would likely use again soon.

This is how operand passing has been done for decades – forcing all operands (prizes) to be passed to the processor through the register file (basket). However, we make the argument for many of the proposed accelerators for today, such as matrix-matrix multiplication, there is not enough reuse to warrant using the basket, or register file. Alternatively, the accelerator/worker could use a method to simply take a large prize from the back room, never touching the basket. However, the downside of bypassing the basket comes when contestants win large prizes consecutively. In that case, bringing the entire stuffed animal box from the back room could have spared the worker the extra trips. Our analysis shows that there is not as much reuse in these designs, and that significant energy can be saved in the processor by eliminating the cost to fill and replace items in the basket, even if the backroom is used more often. Figure 1.3 shows the reduction in energy consumption of our method (denoted NoRF_M$) compared to the RF method for various tightly-coupled

accelerators. This creates a big incentive to move away from the long-lived tradition of using the RF as the only means to operate on data. We also created a clever technique to prevent multiple trips to the large prize box without using the basket if consecutive large prize winners are expected. Using our technique instead of the basket operates at a fraction of the energy consumption as the old technique.

1.5 Contribution 3 – Security within the Tightly-Coupled Accelerator

It is common to hear on the news about security breaches or hackers that have been allowed to see information from computers that they did not have permission to see. Security is always a cat-and-mouse game where hackers are trying to find new techniques to extract information, while researchers are trying to come up with ways to close loopholes before they can be exploited. In 2018, a new classification of hardware security vulnerabilities was discovered. The two large exploits were called Spectre and Meltdown. These attacks took advantage of a process computer architects had been using to increase performance for decades called speculative execution. Speculative execution means that the computer is predicting what it should do next before knowing for sure that it should be done. It's a strange process to think about, because programs are written in-order, or sequentially. The issue arises when the processor hits a "fork in the road," and it will take a while before it learns which path to take. It could simply wait at the fork until it knows for sure, or it could pick one of the two paths, and later be told whether or not it was correct. When it

predicts correctly, the program will run faster. In the past, if it was incorrect, all operations it had done were thought to be erased, and the processor would simply go back and go down the correct path.

Here's an analogy: let's say you are doing a scavenger hunt on a team in the woods with many forks in the road. Each fork has a security guard and a hard puzzle to solve which tells you which direction to go. Your team members communicate via cell phone which path to take, and you're looking for specific items. When you hit a fork in the road, you call your team, tell them the puzzle, and they start to solve it. To save time, you guess which path is the correct one, and start searching for items. You get a call back at some point to determine if you went down the correct path or not. If you went down the correct path, you continue on your way. If you did not, you need to backtrack to the fork, at which point the security guard will take away any items that you found on the wrong path. The vulnerability comes when researchers found a way to slingshot a **single** item (1 Byte, or 8 bits of a secret) from the incorrect path into the correct path. So even when passing the security guard, the searcher did not have any items on them, and the security guard let them go. They could then later pick the item up after changing direction into the new path. A cheater may be able to exploit this to cheat during the scavenger hunt. Figure 1.4 shows the attack. The math problem is the fork in the road, and the scavenger went down the 'YES' path, slingshot the sensitive data (such as a password letter) from the 'YES' to 'NO' path, and later went back to the NO path once the math problem was completely solved. Since the searcher no longer has the value in their pockets after using the slingshot, the security guard does not catch the cheater.

Now let's say we want to create an arbitrary accelerator into the processor design. You

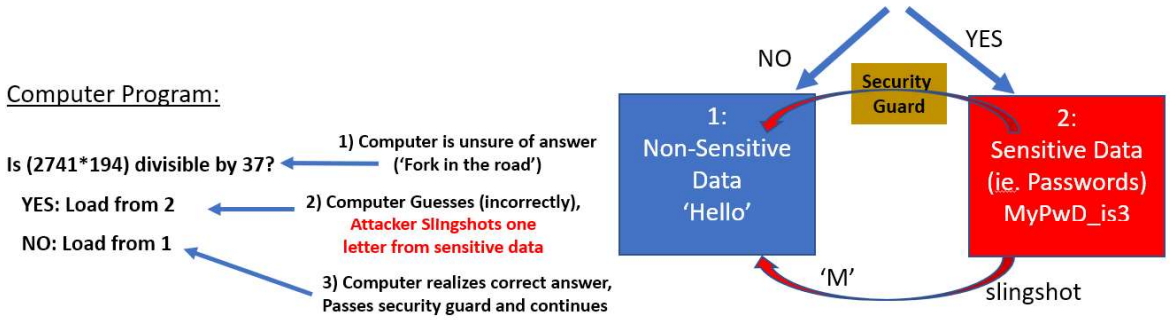


Figure 1.4: Example of Spectre-like attack. Even though the sensitive data is supposed to be discarded once passing the security guard, the attacker found a way to 'slingshot' information from the sensitive data. Since the attacker doesn't physically hold any values, the security guard lets them through. Doing this multiple times eventually tells the attacker 'MyPwD_is3'

might want to do this in order to make a frequently-used computer program run faster. Building this new accelerator is as if instead of you wearing a standard uniform with only a single pocket for the security guard to check, you're allowed to wear any uniform you want, which may have secret compartments. If the security guard only knows how to check a single type of pocket, the cheater could wear a hat, for instance, and stick hidden items in their hat without being detected by the guard. This is even worse than the slingshot case, because instead of a single item being stolen, you could stash as many items as possible for each wrong path. The benefit, however, to having an arbitrary accelerator would be huge in the computer architecture community, as you could accelerate virtually anything you want, making programs much faster. This impact would be even larger if the accelerator could be switched in and out. Although it may sound like fantasy to have circuits and physical parts that can change within a computer, there's a technology called Field-Programmable-Gate-Arrays (FPGA) that has the capability to implement a desired circuit, and can later be reconfigured to physically implement a different circuit. If this

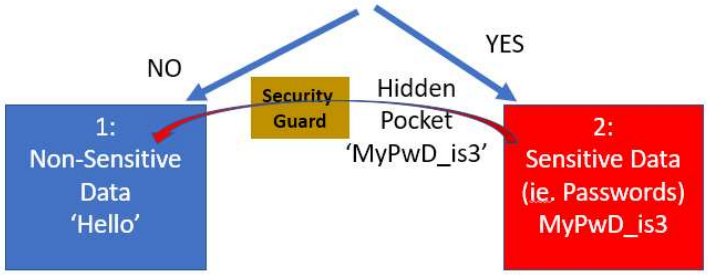


Figure 1.5: Same attack as Figure 1.4, but this time the attacker has a hidden pocket, containing the entire password 'MyPwD_is3' – Our security rules prevent the searcher from having pockets that it doesn't know about, preventing this much faster and more detrimental attack.

technology could be combined into the processor as a tightly-coupled accelerator, then the same FPGA hardware could implement an infinite number of accelerators, where the programmer can choose which accelerator to use for their specific program.

The risk of having any arbitrary accelerator is clear in the analogy of being able to stash lots of secret information without the security guard's detection. Our work solves this problem by making rules that the accelerator designer must use in order to run that accelerator. For example, in our analogy, an outfit might only be allowed zipper pockets or open pockets, but not be allowed stitched-shut pockets or hats. The security guard then knows exactly what to look for. In the analogy this might be silly, but from the hardware perspective, certain circuit components that are allowed to hold information long-term (called flip-flops) can be required to be connected to a wire from the CPU that clears all data from wrong paths before it can be stolen. If the designer does not follow the rule, then the circuit cannot be loaded. If the rule is followed using the trusted, safe, building blocks we provide, we can make certain guarantees about safety.

This set of rules can be extremely powerful. If a processor company such as Intel or

AMD wants to design accelerators, by following our rules, they can make certain guarantees about the security of their accelerators. Alternatively, this allows for the possibility of even allowing the everyday programmer to make whatever accelerator they want. Traditionally, hardware is designed up front and not changeable once it is shipped. This forces programmers to use the existing tools to make their designs, but having the possibility to make their own hardware could lead to significant benefits in everyday computing. By having the security check rules in place, companies like Intel or AMD may be able to have reconfigurable hardware to accelerate many tasks without risking someone with malicious intent using that hardware to obtain secret information.

1.6 Concluding Remarks and Future Work

The focus of my work is tightly-coupled accelerators, which are specialized circuits to significantly speed up programs within a general-purpose processor. I first created a series of mathematical formulas to estimate the benefits of various accelerators without the need for many hours of detailed simulation or millions of dollars to test. Second, I demonstrated how data movement in these tightly-coupled accelerators should not follow the same rules that computer architects have been using for decades to move data, and that our method can save significant energy without compromising performance. I lastly created a framework to allow arbitrary tightly-coupled accelerators to be built in a way that will guarantee that speculative execution attacks are not more detrimental.

In the future, I hope to see this work used by processor companies that will allow both the general processor instructions (such as addition, subtraction, multiplication,

comparisons, etc.), as well as a reconfigurable accelerator instruction that the programmer can use for anything they would like. This would be a huge move forward in the history of processors, as a general-purpose processor could be specialized in millions of ways without losing its existing generality or having to incorporate millions of accelerators at design-time. That means that a processor built in 2025, for example, could be configured to specialize for a program invented in 2027! I hope this work begins the journey of a new era of accelerators and computing.