Pipelined Processor Energy Optimizations

Cory Avrutis
Florida State University

February 2022

Abstract

Over the last two decades we have seen power consumption severely limit the capability of processor performance growth. With this limiting factor being so prevalent, it has driven computer architects and engineers to present innovative techniques to reduce the energy consumption within a processor. This paper discusses three distinct techniques that seek to address this concern, and their implementations. Dynamic Voltage and Frequency Scaling (DVFS), Pipeline Stage Unification (PSU), and Static Pipelining are the techniques presented in this paper. DVFS is the technique with the most widespread use in modern machines but is severely limited by the pace of technological advances. PSU is a proposed technique with lesser limitations and is predicted to have a continual increase in effectiveness over DVFS as technology advances. Static Pipelining is a more recent technique proposed which aims to restructure a processor at the architectural level in order to circumvent various inefficiencies found in the typical pipelined processor. Industry has yet to impose a new standard for processor energy optimization, but as academia continues to push for innovation, new techniques will emerge that will overshadow the dated and less efficient techniques.

1 Introduction

Society has seen a great boom in processor speed over the past five decades, but within the last two decades, power consumption has been a prevalent limiting factor of these advances. The modern-day challenge of increasing processor performance stems from the thermal limitations of these processors, which is directly impacted by power consumption. This is just one reason why scientists and engineers are working towards reducing the energy consumption of processors. Reducing the energy consumption in processors is not a new idea, but new techniques to achieve this reduction continue to emerge.

This paper will discuss three techniques to reduce the energy consumption of a processor. First, Section 2 will discuss Dynamic Voltage and Frequency Scaling, a brief overview of its implementation, and its current limitations for reducing energy consumption. Section 3 will discuss Pipeline Stage Unification, its implementation, and the ways it reduces energy consumption. Section 4 introduces a fairly new approach to energy optimization called Static Pipelining and the various ways it can reduce energy consumption. Lastly, Section 5 will give a synopsis of the paper and the various energy reduction techniques.

2 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling ($DVFS$) is a technique to reduce the energy consumption of a processor. The optimization comes in two parts, the clock frequency scaling and the supply

voltage scaling for the circuit. As we reduce the clock frequency, we can also reduce the supply voltage to the circuit, since the supply voltage is dependent on the clock frequency of the circuit. After a management interval passes, the workload of the CPU is analyzed and the clock frequency is adjusted accordingly[1]. If we look at the equation for total power consumption within a circuit it shows the relationship between the frequency and supply voltage [1].

$$PowerConsumption = C * f * V^2 + Power_{static}$$

[1]

In the above formula, f represents the clock frequency of the circuit and V represents the supply voltage to the circuit. We can ascertain that most of the power being consumed is within CfV^2 the portion of the formula and that the additional static power is of a lower order so can be ignored for optimization purposes. The majority of power reduction is attained through the V^2 relationship as it is the only exponential term in the formula [1]. Therefore, reducing the clock frequency (f) will proportionally reduce the supply voltage resulting in a discernible reduction in power consumption[1].

2.1 Limitations on DVFS Effectiveness

E. Le Sueur and G. Heiser describe different factors that influence the effectiveness of DVFS. Overtime, transistor feature size has decreased, and will continue to decrease[1]. As these sizes decrease, the voltage leakage will naturally grow resulting in higher static power consumption[1]. DVFS reduces power consumption by reducing the dynamic consumption (CfV^2) and has no control over the static power ($Power_{static}$) being consumed. Consequently, as the static power consumption naturally increases, the effectiveness of DVFS will naturally decrease[1]. There is no solution to this increase in static power consumption, so this is a heavy limitation on DVFS.

Another DVFS limitation was introduced with the wide-spread use of multi-core processors[1]. For single core processors, DVFS must only analyze the workload of one core to determine whether to scale or not. On multi-core processors, DVFS implementations force each core to operate at the same voltage, which will always be the voltage supplied to the core with the highest frequency. Since DVFS must analyze the workload of each core on a multi-core processor, there will always be a small inaccuracy when these workloads are averaged using the highest frequency[1].

3 Pipeline Stage Unification

Pipeline Stage Unification (PSU) is a technique to lower energy consumption in processors introduced by H. Shimada, H. Ando, and T. Shimada [2]. PSU is an adaptive way to scale the depth of a pipeline depending on the current processor workload. Similar to DVFS, PSU will scale down the clock frequency of a processor depending on its workload. In contrast to DVFS, PSU will not scale the supply voltage, but instead will bypass one or more pipeline registers in order to unify two or more pipeline stages[2]. Therefore, when a pipeline register is bypassed, the combinational logic on both sides of the register will come together as one pipeline stage[2]. This is visualized in Figure 1 (Derived from Shimada et al. [2]), which depicts the difference in the flow of logic when pipeline stages are unified.

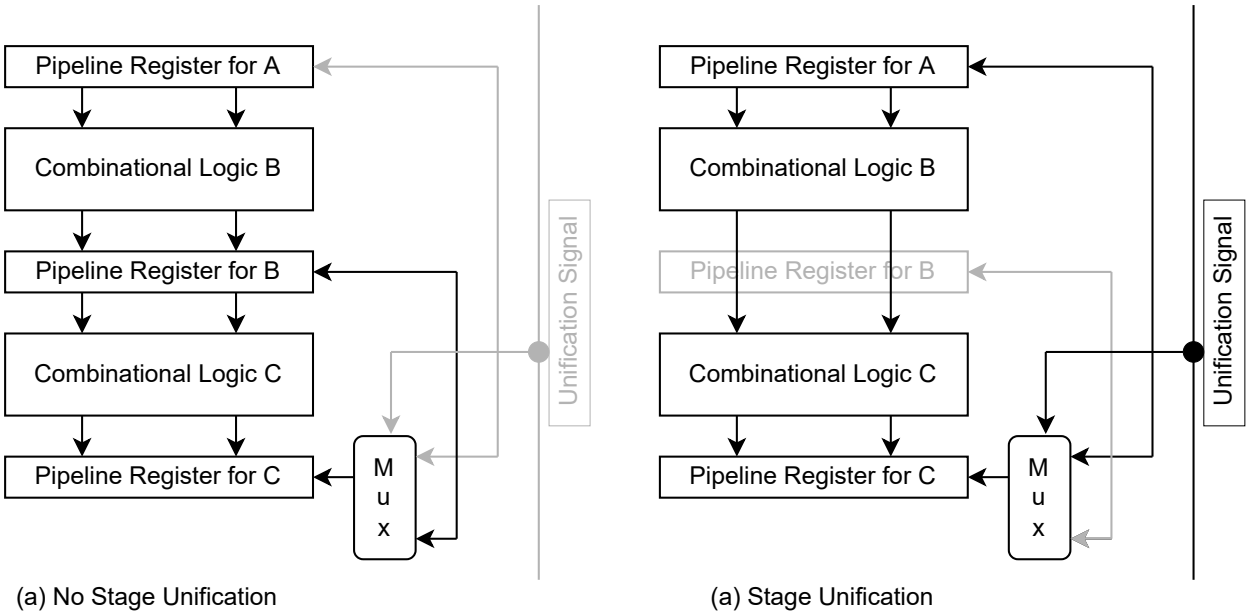

Figure 1: Pipeline Stage Unification [2]

3.1 PSU Implementation

In the PSU implementation described by the authors, there are three predefined signals to be used in the pipeline circuit. A full-time clock signal, part-time clock signal, and a unification signal are each used for this implementation. The full-time and part-time clock signals will remain active while there is no stage unification, but once pipeline stages are unified, the part-time clock signal is disabled while the full-time signal remains active[2]. The unification signal is used to indicate that pipeline stages are unified. The unification signal also drives a multiplexor that chooses between the outputs of the most recent pipeline register and the one prior to it, and sends the values to the next pipeline register[2]. This multiplexor can be seen in figure 1, where the active signals are represented with solid black lines. The *unification degree* U_N is used to represent that each set of N pipeline stages are unified as a single pipeline stage [2]. In this context, degree U_1 will have no stage unification, while degree U_2 will merge every set of 2 pipeline stages into a single stage. In the analysis of J. Yao, S. Miwa, and H. Shimada, it was found that the latency of a unification switch comes down to flushing the pipeline and scaling down or up the clock frequency [3].

3.2 PSU Energy Reduction

Many modern processors aim for a high clock frequency, and many architectures impose a deep pipeline strategy to achieve this [2]. In the author's processor implementation, they impose a deep pipeline consisting of 20 pipeline stages, with the exception of load instructions having 2 extra stages for write-back and commit [2]. Using unification degrees U_2 and U_4, this deep 20 stage pipeline can be adaptively scaled to a shorter depth of 10 and 5 stages respectively, with load instructions having 1 extra stage for both degrees [2, 3]. Due to the reduction in pipeline stages, there is a corresponding reduction in clock cycles needed to execute a program [2]. When scaling the pipeline depth, energy consumption is first reduced by scaling down the clock frequency and disabling the part time clock signal to pipeline registers being bypassed [2]. Limiting the signals

to only the full time clock and unification signal will reduce the total load needed for the clock driver [2]. Ideally, for the author's PSU processor [2], the clock driver's power consumption can be reduced by $1/N$ for degree U_N. Energy reduction is further increased by the reduction in program execution clock cycles that comes from having fewer pipeline stages[2].

4 Static Pipelining

Static Pipelining, as described by I. Finlayson, G.-R. Uh, D. Whalley, and G. Tyson, takes a different approach to optimizing energy consumption in a processor [4]. The authors explain that in the classic pipeline, instructions will perform unnecessary actions leading to inefficient energy consumption [4]. For example, non-immediate instructions will always sign extend the lower 16 bits (in 32 bit machine) and send the value to the ALU. This led the authors to propose a technique that focuses on removing these unnecessary inefficiencies. First, the authors introduce a micro-architecture to support static pipelining which consists of a number of *internal registers*, in place of pipeline registers [4]. To support compiler use of these internal registers, they propose an instruction set with the ability to encode, into each instruction, the control for each action that the processor should perform by utilizing these internal registers [4]. Table 1, which was derived from the author's descriptions of these internal registers, lists the 10 internal registers and their uses [4].

Internal Register	**Register Use**
RS1 RS2	Register Source Values, used to hold values read from register file.
LV	Load Value, used to hold values loaded from the data cache.
SEQ	Sequential Address, used to hold the address of the next sequential instruction at time of writing.
SE	Sign Extend, used to hold a sign-extended immediate value.
ALUR TARG	These registers are used to hold the values calculated by the ALU. If the PC is used as input to the ALU, then the result will go in the TARG register, otherwise result goes to the ALUR register.
FPUR	FPU Result, used to hold values calculated by the FPU.
CP1 CP2	Copy 1 and 2, used to hold values copied from other internal registers and for use during various optimizing compiler techniques.

Table 1: Internal Registers and Uses [4]

4.1 Static Pipeline Implementation

Static pipelining is supported in hardware by using internal registers, and supported in software via assembly code compiler techniques. Instructions executing on a statically pipelined processor are not broken up into multiple stages, as there are no pipeline registers the internal registers will be explicitly read or written by the instruction each cycle [4]. Unlike pipeline registers, internal registers are only written to or read from when needed by an instruction [4]. The processor proposed by the author is split into the fetch stage and the rest of the calculations, in-essence representing two stages [4]. Every operation after the instruction fetch is done in parallel because the instruction will only perform the necessary actions required of itself by reading or writing to the internal registers

within a clock cycle[4]. These internal register values are held across clock cycles and can be utilized by separate instructions [5].

The instruction set architecture for the author's processor allows for each instruction to have a set of *effects*. These effects roughly correspond to the operations found in the typical 5-stage pipeline, where each effect is independent and will operate in parallel with the other effects of an instruction [4]. Allowing the compiler to access the internal registers is what drives the unnecessary actions in a traditional pipeline to be ignored [4]. In the author's example of how source code is compiled into assembly code, they first compile C source code into a partially optimized intermediate MIPS assembly representation [4]. The portions of the MIPS code that are not compatible with the static pipeline are then manually transformed into intermediate MIPS code, and the entirety of this intermediate code is compiled into optimized assembly code for a statically pipelined processor [4].

Figure 2 shows an example of the process of going from a single C statement to statically pipelined code. Some assumptions made when producing the code is that the value of a is stored in r[1], the value of b is stored in r[2], and the address of c is stored in r[9]. The C code is first compiled into its intermediate MIPS representation, then manually transformed to support a statically pipelined processor. As seen in the figure, each MIPS instruction is broken down into its corresponding set of effects, with each MIPS instruction's corresponding statically pipelined code being seperated by dashed lines. As an example, the first MIPS instruction which adds two register values and stores the result in a separate register breaks down into setting the source register values, processing the summation, and storing the summation back into a register. Therefore we set RS1 and RS2 accordingly, process the summation which is stored in ALUR, and set the r[3] register to the value stored in the ALUR register.

4.2 Static Pipeline Energy Reduction

The static pipeline technique proposed by the author's reduces energy consumption in a number of ways. First, the energy consumption is reduced from the reduction in useless and inefficient operations performed by the processor. Some of these inefficient operations include: 1) Accessing register file to retrieve a value when it's value will be provided by forwarding. 2) Sign extending a value for a non-immediate instruction. 3) Writing value to register when the only consumer gets the value via forwarding. 4) Executing memory address calculation when the offset operand is 0. Since the operations for a statically pipelined instruction are performed in parallel, there is no need for pipeline registers, which nullifies the overhead of having to pass values through the pipeline. The author also mentions that each internal register can be accessed at a lesser energy cost than accessing a centralized register file because internal registers are small and are placed closest tot the portion of the processor that will use it [4]. Finally, since this technique exposes these internal registers at the architecture level, it opens the door for undiscovered compiler optimizations leading to better performance [4].

C Source Code	Statically Pipelined Code

`c += a + b;`	`RS1 = r[1]`
	`RS2 = r[2]`
	`ALUR = RS1 + RS2`
	`r[3] = ALUR`

MIPS Code	`RS1 = r[9]`
	`LV = MEM[RS1]`
	`r[4] = LV`

`r[3] = r[1] + r[2]`
`r[4] = Mem[r[9]]`
`r[3] = r[3] + r[4]`
`Mem[r[9]] = r[3]`
	`RS1 = r[3]`
	`RS2 = r[4]`
	`ALUR = RS1 + RS2`
	`r[3] = ALUR`

	`RS1 = r[3]`
	`RS2 = r[9]`
	`MEM[RS2] = RS1`

Figure 2: C Code to Statically Pipelined Code [4]

5 Conclusions

In this paper, I presented three distinct techniques that can be used to reduce the energy consumption in a processor. DVFS is currently being used as a standard for reducing power consumption in modern machines, but unfortunately the effectiveness of its energy reduction will continue to decrease as transistor technology evolves. Pipeline stage unification is a technique used to conjoin sets of pipeline stages into single stages. The advantage of PSU over DVFS, according to the authors, is that PSU's effectiveness will not decrease, but increase as hardware technology advances [2]. The PSU study showed that current PSU implementations are 11% - 14% more effective at reducing energy consumption than DVFS, and also that in 10 years PSU is predicted to be 27% - 34% more effective than DVFS [2]. The last technique I analyzed was Static Pipelining, which is achieved through an architectural overhaul of the processors datapath. This technique utilizes internal registers to carry data over cycles, instead of using pipeline registers, and allows individual instructions to access these registers as they are needed. In this way, many inefficient operations can be avoided, as well as the pipeline information passing overhead, and energy consumption can be reduced. All in all, reducing the energy consumption of processor is a very relevant field of research, and I believe that scientists and scholars will continue to push for advancements leading to more energy efficient processors.

References

[1] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010, pp. 1–8.

[2] H. Shimada, H. Ando, and T. Shimada, "Pipeline stage unification: a low-energy consumption technique for future mobile processors," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03.*, 2003, pp. 326–329.

[3] J. Yao, S. Miwa, H. Shimada, and S. Tomita, "A fine-grained runtime power/performance optimization method for processors with adaptive pipeline depth," *Journal of Computer Science and Technology*, vol. 26, no. 2, pp. 292–301, 2011.

[4] I. Finlayson, G.-R. Uh, D. Whalley, and G. Tyson, "Improving low power processor efficiency with static pipelining," in *2011 15th Workshop on Interaction between Compilers and Computer Architectures*, 2011, pp. 17–24.

[5] D. Whalley, G. Uh, I. Finlayson, and G. Tyson, "An overview of static pipelining," *IEEE Computer Architecture Letters*, no. 01, pp. 17–20, jan 2012.