K-HOT PIPELINING

BY

DAO LU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Associate Professor Rakesh Kumar

ABSTRACT

Computing systems in almost every application domain now support techniques to trade off power and performance. Such techniques are used to enforce power and thermal constraints, manage power and thermal budgets, and respond to temperature and aging. Unfortunately, many of the current techniques are limited in the dynamic range they provide and scale poorly with technology. Techniques that can supplement or replace current techniques are needed. We propose k-hot pipelining, a novel technique to support multiple power-performance points in a processor. The key idea is to provide power and clock to only k stages of an m-stage pipeline ($k < m$); the k stages to be powered on change as instructions flow through the pipeline. Since the remaining $m - k$ stages do not consume power, the technique results in power savings at the expense of performance. k-hot pipelining can be software or hardware-controlled, workload-agnostic or workload-adaptive, and can be used to provide power-performance points not supported by existing techniques. For one implementation of k-hot pipelining, we show that up to 49.9% power reduction is possible over the baseline design. Power reduction is up to 47% over the lowest power point supported by DVFS.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Rakesh Kumar, for keeping me focused on the important aspects of this work while driving me to produce quality research. I would like to thank Matt Tomei, Henry Duwe and Weidong Ye for their helpful comments, insights, and suggestions. Their intellectual rigor and humor were greatly appreciated. I would like to thank my friends outside of my research group, Qingkun Li, Kuan-Yu Tseng, Yixiao Lin, Siqi Li, Yi-Kai Lee, Yan Yan, Chia-Hao Lee and Lingbo Sun for their company, encouragement, and camaraderie.

Last but not least, I would like to express my deep gratitude to my parents, Lijun Li and Cong Lu, and my girlfriend, Kexuan Wang, for their greatest love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Most current and emerging computing systems support techniques to trade performance for power. There are several reasons why such techniques are supported. In some systems, such techniques help enforce power and thermal constraints [1–23] As a recent example, renewable energy sources are increasingly being used to power data centers and provide intermittent power. In the extreme case where only renewable sources are available, this intermittency requires a way to match power consumed to available power [3]. In some other systems, such techniques are used to manage power and thermal budgets. For example, various methods for power capping (budgeting average power through dynamic power constraints) exist to allow under-provisioning the cooling system [4,5]. There are many systems where such techniques are used to respond to temperature and aging. For example, architectural performance limiting techniques can be used to limit the maximum temperature of a chip, providing better lifetime reliability [6].

Unfortunately, current techniques to trade performance for power have limited dynamic range. Consider dynamic voltage-frequency scaling (DVFS), for example, which is arguably the most effective and widely used technique to trade performance for power. However, the voltage and frequency range

Table 1.1: Voltage and frequency ranges for some processors

	Voltage (V)	Frequency (GHz)
Intel		
Pentium M [24] (2004)	0.988-1.340	0.6-2.0
AMD		
Windsor X2 [25] (2006)	1.25-1.35	2.0-3.2
FX-8120 [26] (2011)	0.875-1.312	1.4-4.0
Samsung		
Exynos 5410 [27] (2014)	0.9-1.25	0.8-1.8

supported through DVFS is limited. Table 1.1 shows the maximum and minimum frequency and voltage for a set of processors for sustained operation. We see that the voltage reduction provided by DVFS is limited to 100-437 mV for different processors. This is not surprising since the reliability problems with scaling to low voltages are well documented [28]. Corresponding power reduction is also limited, particularly considering that a large fraction of power (30%-55% [29]) is leakage.

To make matters worse, the efficacy of current techniques is diminishing with time. For example, as can be seen from Table 1.1, the available voltage range has not scaled commensurately with transistor scaling over the last several years. Again, this is not surprising considering (a) Dennard scaling across process generations has slowed down considerably and (b) PVT and aging variations limit the V_{min} of the design. Clearly, new techniques are needed that can trade performance for power. These techniques can either be used in conjunction with current techniques to increase the available resolution and dynamic range in terms of power or they can be used to replace current techniques as their effectiveness diminishes in the future.

We propose *k-hot pipelining*, a novel technique to support multiple power-performance points in a processor. The key idea is to provide power and clock to only k stages of an m-stage pipeline ($k < m$); the k stages to be powered on change as instructions flow through the pipeline. Since the remaining $m - k$ stages do not consume power, the technique results in power savings at the expense of performance. For example, only one stage of a processor pipeline is on at a time in a one-hot pipeline. The on stage moves through the pipeline from left to right as an instruction flows through the pipeline. As another example, a conventional pipeline where all stages are always on can be considered a full-hot or m-hot pipeline where m is the number of pipeline stages. k-hot pipelining can be software or hardware-controlled, workload-agnostic or workload-adaptive and can be used to provide power-performance points not supported by existing techniques.

For one implementation of k-hot pipelining, we show that up to 49.9% power reduction is possible over the baseline design. Power reduction is up to 47% over the lowest power point supported by DVFS. The benefits are expected to increase with scaling.

In a multi-core setting, k-hot pipelining presents new challenges and opportunities. A naive application of k-hot pipelining can result in high peak power

dissipation as well as high power variability due to simultaneous turning-on of the high-power states in different cores. We investigated k-hot pipelining implementations specific to multi-core processors. Our multi-core specific implementation of k-hot pipelining reduces processor peak power by up to 15% and power variability by up to 58% over a naive k-hot implementation.

This research makes the following contributions:

- We present *k-hot pipelining*, a novel approach to trade performance for power.
- We implement k-hot pipelining for a five-stage, two-wide in-order processor at 65 nm technology node, and show up to 49.9% power reduction over the baseline design.
- We use k-hot pipelining in conjunction with DVFS, and show that k-hot pipelining can reduce power by up to 47% over the lowest power point supported by DVFS.
- We propose multi-core specific optimizations for k-hot pipelining. We evaluate these optimizations in context of a four-core multi-core processor, and show peak power reduction of 15% and power variability reduction of 58% for this processor.

CHAPTER 2

RELATED WORK

A large body of work exists on techniques for trading performance for power. DVFS [9]-[16] scales down processor voltage to decrease power. Since this decreases the speed of transistors, a corresponding decrease in frequency is required. Idle cycle injection (ICI) [17, 18] is used to dynamically power cap machines by running an idling instruction for some percentage of the time. Clock cycle modulation (CCM) [19, 30] decreases the duty cycle of the clock to save dynamic power and meet thermal requirements. Unfortunately, the power benefits from these techniques are limited. While $\frac{V_{min}}{V_{max}}$ is limited for DVFS due to PVT and aging variations, ICI and CCM target only dynamic power. Also, with the slowdown of Dennard scaling [31], the range supported by DVFS is expected to decrease. k-hot pipelining can be applied either in conjunction with these techniques to increase the resolution and range of possible power benefits or as a replacement to these techniques.

Other related work includes work on coarse-grained [20–22, 32] and fine-grained [33–36] power gating. Power gating cuts off power to unused logic, eliminating static power for the gated logic. However, coarse-grained power gating has large wake-up and sleep overheads. It also does not reduce peak power. We use fine-grained power gating at a pipeline stage granularity in such a way that only k out of m stages in a processor pipeline are on at a time.

CHAPTER 3

MOTIVATION

Consider a canonical scalar, in-order, five-stage processor pipeline (Figure 3.1a). In steady state, five instructions flow through this pipeline in parallel and each stage is occupied. Now consider an alternative configuration where only one instruction flows through the pipeline at a time and only the stage in use is powered on (Figure 3.1b). Since the powered off stages do not consume static power, the *one-hot* operating mode's power consumption is much less than the fully on (*full-hot*) processor. In fact, assuming all pipeline stages are independent, the one-hot pipeline consumes one-fifth the power of the full-hot baseline.

The one-hot operating mode is only one of a set of operating modes possible given power gating at the granularity of a pipeline stage. Any number of pipeline stages can be powered on simultaneously. We refer to an operating mode with k simultaneously powered stages as a k-hot operating mode. In general, for a pipeline with m independent stages, k-hot pipelining will result in α power savings according to Equation 3.1.

$$\alpha = 1 - \frac{k}{m}, 0 \leq k \leq m \tag{3.1}$$

k-hot pipelining results in a reduction of not only average power, but peak power[1] as well. Peak power savings can be described according to Equation 3.2 where P_i is the peak power of the ith stage of the processor and S is any set of k stages that are on simultaneously during k-hot operation.

$$\beta = \frac{\sum_{i=1}^{m} P_i - \max_S(\sum_{i \in S} P_i)}{\sum_{i=1}^{m} P_i} \tag{3.2}$$

Since there are m possible values of k, there are m possible k-hot power-

[1] measured over a cycle

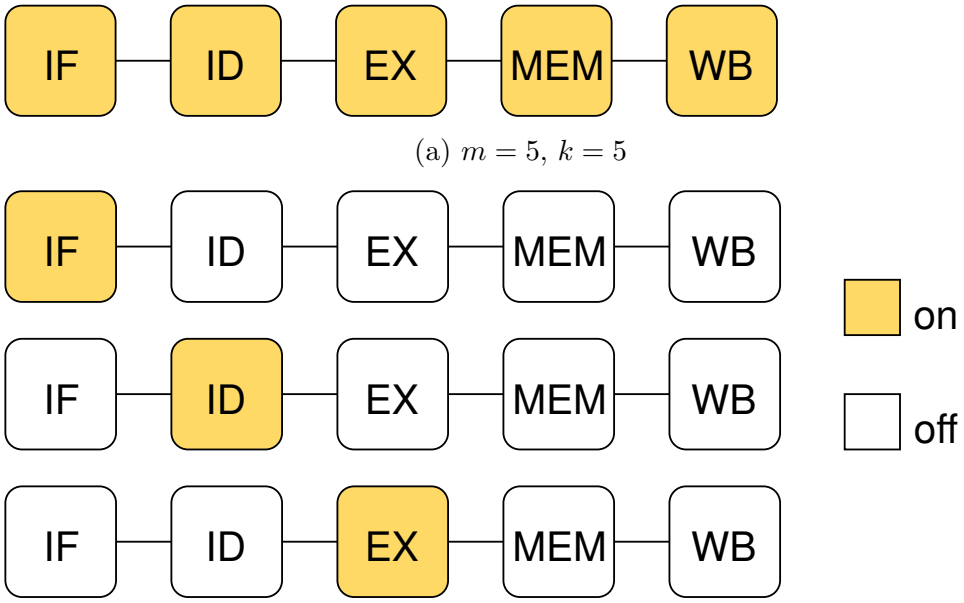


Figure 3.1: (a) Full hot mode: each stage is powered on and occupied by a distinct instruction. (b) One-hot mode: one stage is powered on at a time, allowing 1 instruction to be in the pipeline.

performance points. Additional points are possible by mixing values of k. For example, a pipeline operating in one-hot mode half of the time and two-hot mode the other half of the time will be at a power-performance point in between that of one-hot and two-hot modes. We discuss a method for mixing values of k in Chapter 4.

In addition to having high resolution and dynamic range in terms of power consumption, k-hot pipelining has a bounded performance degradation. Since the maximum speedup due to pipelining is $m\times$, the performance of the one-hot pipeline is no more than $m\times$ lower than the m-hot pipeline.

Finally, the estimated energy of an ideal k-hot pipeline is equal to the energy of the original m-hot pipeline. Since k-hot operation can be applied on top of DVFS, it may be possible to extend the energy optimal DVFS power-performance point to lower power-performance points (i.e., similar energy at lower power targets). This means k-hot pipelines may have a larger range of near energy-optimal operation.

CHAPTER 4

IMPLEMENTING K-HOT PIPELINES

K-hot pipelining minimally requires one power domain per pipeline stage. It also requires logic to generate control signals for each domain. In this chapter, we describe how to generate control signals so that k stages are powered on in each cycle and, barring stalls, each instruction has the same latency as in the full-hot pipeline mode.

Consider again the one-hot canonical five-stage pipeline presented in Figure 3.1b. Each stage requires a dedicated control bit; let these control bits be active high. Consider the arrangement of these bits into a *control vector* where bits from most to least significant correspond to stages from the front to the back of the pipeline. For example, the control vector of the pipeline at the bottom of Figure 3.1b is $\{0, 0, 1, 0, 0\}$.

Figure 4.1 shows a waveform of the control vector during one-hot operation. In every cycle, all but one of the bits are zero. At the end of each cycle, the set bit moves to the next most significant position or, when it is at the least significant position, rotates back to the most significant. One instruction is executed every five cycles, and the only stage powered on is the one required for the current instruction to make progress. We implement the control vector using a 5-bit rotating shift register where each bit in the shift register controls a stage in the pipeline.

We use the same 5-bit rotating shift register to control power domains in a k-hot pipeline but set k bits instead of one. For a pipeline with m stages, there are $\binom{m}{k}$ choices of m-bit vectors with k bits set. If we continue to assume that all stages are independent, all control vectors with k bits set will result in the same average power according to Equation 3.1. This limits us to m possible power-performance points for an m bit vector.

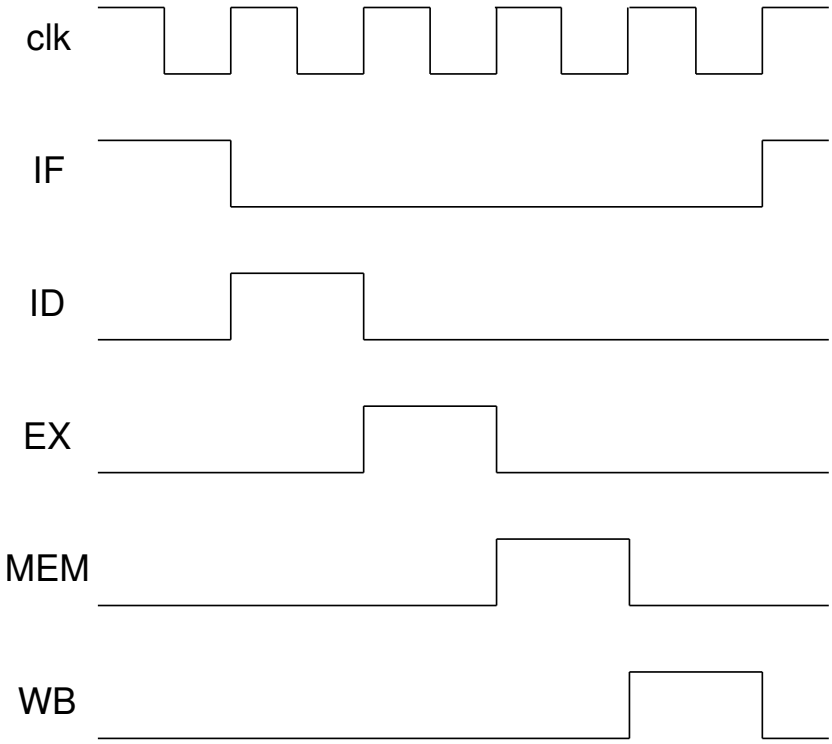


Figure 4.1: In a one-hot pipeline, one stage is on in each cycle, and only one instruction is in the pipeline at a time.

Consider extending the m bit vector to $n > m$ bits. A block diagram for the canonical pipeline with a rotating n bit shift register is shown in Figure 4.2. The five least significant bits control the power gating of the five stages. The hotness of the pipeline can now be any rational number of the form $m * \frac{k}{n}$, where k is the number of bits set in the n-bit control vector.

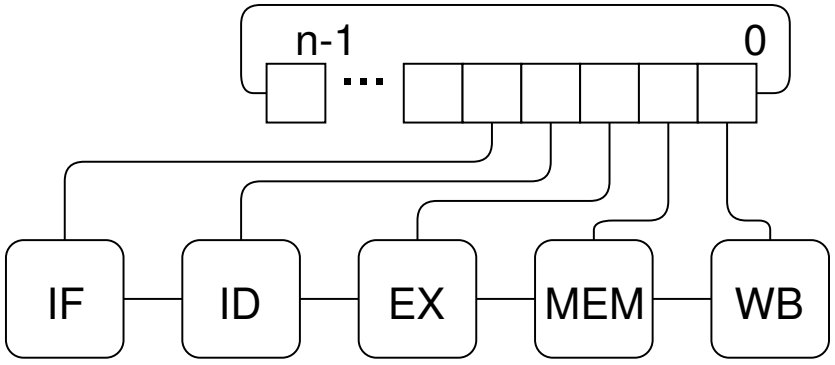
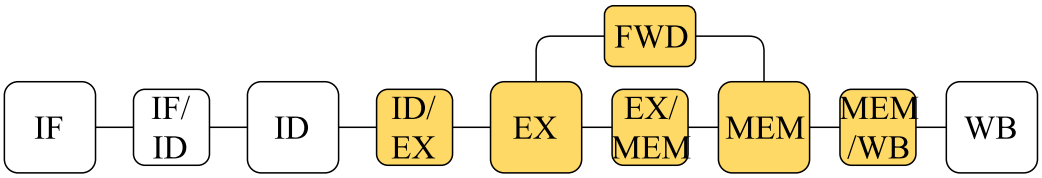
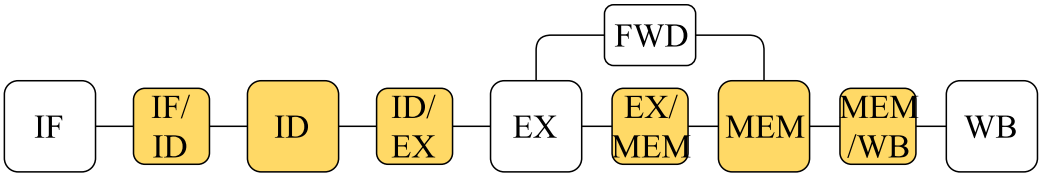


Figure 4.2: Pipeline block diagram with control register.



(a) $k = 5$, $v = \{0, 0, 1, 1, 0\}$



(b) $k = 2$, $v = \{0, 1, 0, 1, 0\}$

Figure 4.3: (a) A two-hot example where three pipeline latches, two pipeline stages and one forwarding unit are on. (b) Another two-hot example where four pipeline stages and two pipeline stages are on.

Since arrangement of the set bits in the control vector does not matter for our simple pipeline, an n-bit vector allows n power-performance points, or $n - m$ more power-performance points than the m-bit vector. The extension also increases the denominator in Equation 3.1 from m to n. Therefore, the range of power consumption is increased, and the set of possible power-performance points is changed.

4.1 Pipeline Stage Interaction

Practical pipeline stages are not independent. Consider the block diagram in Figure 4.3 corresponding to the canonical five-stage processor. Blocks have been added for pipeline latches, forwarding logic, and the branch unit. All three additions are examples of logic shared between stages. Pipeline latches are shared by their reader and writer; forwarding logic is shared by its source and destination, and the branch unit is shared by the fetch and resolving stages.

Sharing complicates the division of the pipeline into power domains. Consider an example power domain configuration for the five-stage pipeline shown in Figure 4.4 where each block is in its own domain. Our control scheme generates control bits for each stage which must then be used to generate the

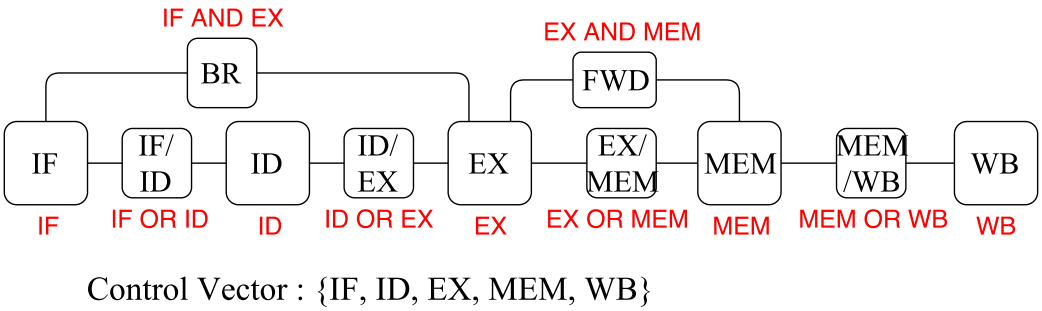


Figure 4.4: Power domains and their control signals for a five-stage pipeline. Some modules are shared between stages and thus have multiple control signals driving its power.

control bits of all domains in Figure 4.4, including the domains for the added blocks. The gating control signals for the pipeline latches are calculated as the logical OR of the connected stages. The branch and forwarding unit control signals are calculated as the logical AND of both connected stages.

Since stages are no longer assumed to be independent, Equation 3.1 no longer holds. Therefore, all $\binom{m}{k}$ configurations of a k-hot control vector may correspond to different power-performance points. Consider the pipeline latches shown in Figure 4.3, for example. When the control vector is $\{0, 0, 1, 1, 0\}$ (Figure 4.3a), three sets of pipeline latches are on. On the other hand, when the control vector is $\{0, 1, 0, 1, 0\}$ (Figure 4.3b), four sets of pipeline latches are powered on, representing a different power/performance point. In Chapter 7, we present our results for k-hot pipelining that maximizes average power savings for the modeled processor (Chapter 6) for our workloads. We also show power benefits for the case where for each application, the chosen control vector is one that maximizes power savings for an application. In general, sharing decreases the power benefits from k-hot pipelining since shared logic is much more likely to be on for a given control vector than unshared logic.

4.2 Switching between Values of k

A processor's power-performance point may need to change to adapt to changing workloads, power budgets, and power constraints. To switch to a different power-performance point in a k-hot pipeline, the degree of hot-

ness of the pipeline (value of k) needs to be changed. As such, we need a technique for dynamically changing the control vector ($\{0,0,0,0,0,1\} \rightarrow \{0,0,0,0,1,1\}$ to move from one-hot to two-hot, for example) without affecting correctness of execution.[1] A method for safely switching between control vectors also allows us to target non-integer values of k. For example, if we spend equal time in $k = 1$ and $k = 2$, we would achieve an average power consumption corresponding to $k = \frac{3}{2}$ in Equation 3.1.

One way to safely change the control vector to a vector corresponding to a new value of k is to flush the pipeline, change the control vector, and then resume execution. The switch will take a maximum of $2m$ cycles for an m-stage pipeline: m to drain the pipeline and m cycles to refill the pipeline. Since bits are only toggled when the control vector is not in use, execution is unaffected, but draining and refilling the pipeline for every switch is costly. An alternative method would be to add hardware support for saving and restoring the instruction state such that a control vector can be changed safely from one to another.

We can avoid both the performance and hardware overhead of the previous methods by observing that the IF bit of a control vector can be switched safely without any correctness concerns (since no instruction will get lost). Given a set of bits in the current control vector that need to switch in order to produce a control vector corresponding to the new value of k, we will simply wait for those bits to rotate into the IF stage and then toggle those bits to create intermediate control vectors. Since we are only toggling the IF bit, it will take at most m cycles to modify the control vector corresponding to the new value of k.

Consider, for example, a six-stage pipeline with a current control vector of $\{0,0,0,1,1,0\}$. Let us say that the desired control vector is $\{1,0,1,1,0,0\}$. The quickest way to switch to the new control vector is to toggle the IF bit for three cycles. The sequence of control vectors would be $\{0,0,0,1,1,0\} \rightarrow \{1,0,0,1,1,0\} \rightarrow \{1,1,0,0,1,1\} \rightarrow \{0,1,1,0,0,1\}$.

Note that the starting and ending control vectors correspond to two-hot and three-hot respectively, but the second intermediate control vector is four-hot. In scenarios where k-hot pipelining is being used to guarantee peak power, this overshoot may not be acceptable. However, as discussed above,

[1] Note that a careless switching between control vectors can lead to incorrect execution since state of an in-flight instruction can get lost.

it is easy to avoid overshooting the hotness at the expense of greater switching latency.

CHAPTER 5

K-HOT AND MULTI-CORE

In a multi-core setting, k-hot pipelining presents new challenges and opportunities. A naive application of k-hot pipelining can result in high peak power dissipation as well as high power variability due to simultaneous turning-on of the high-power states in different cores. Consider a two-core system with two independent five-stage pipelines that have identical one-hot control vectors. Since the control vectors are identical, the same pipeline stage is on in each core. Therefore, the power consumed by the system in each cycle will be twice the power consumed by the stage that is on. If there is a large range of power consumption across stages, the variability of power consumed could result in voltage noise and unnecessarily high peak power.

As a specific example, consider a five-stage two-core system whose per stage power consumption is 5 W, 4 W, 5 W, 1 W and 1 W. Figure 5.1a shows the overall power consumed when each stage is powered on. The peak power is 10 W in IF and EX, and the range between the minimum and maximum is 8 W.

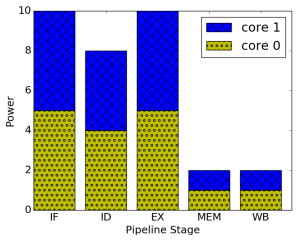
One way to reduce the variability in power consumption and decrease peak power is to stagger the ones in the control vectors. For example, the one-hot control vectors in the two-core example could be set to the following:

$$\{1, 0, 0, 0, 0\}$$
$$\{0, 1, 0, 0, 0\}$$

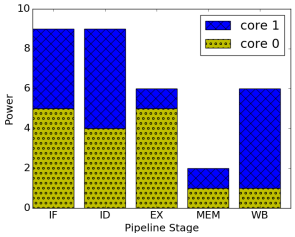
For this selection of control vectors, two copies of the same stage are never on simultaneously. The power consumption of the system in any cycle will be an average of both powered stages. The power that the system consumes in this case is shown in Figure 5.1b where the x-axis now corresponds to the stage powered in core 0. The peak power and range both decreased by 1 W.

To discuss multi-core k-hot systems further, we consider the *control vector*

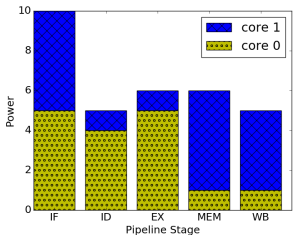
sum, which is the vector sum of all current control vectors in the the system. For example, the control vector sum of the system with identical control vectors $\{1, 0, 0, 0, 0\}$ and $\{1, 0, 0, 0, 0\}$ is $\{2, 0, 0, 0, 0\}$, and the control vector sum in the case of the previous example is $\{1, 1, 0, 0, 0\}$. Note that the range of elements in the first vector $(2 - 0 = 2)$ is larger than the range of elements in the staggered vector $(1 - 0 = 0)$. Because the control vectors are constantly rotating and we assume that all control vectors in the system rotate simultaneously, a control vector sum is equivalent to all other control vector sums that are a rotation of the original. For example, $\{1, 1, 0, 0, 0\} \equiv \{0, 1, 1, 0, 0\} \equiv \{1, 0, 0, 0, 1\}$.



(a) Identical control vectors in a multi-core system result in a high peak power since two copies of the peak power stage are on simultaneously.



(b) Staggering the bits in the control vectors can decrease both peak and range of power consumption across cycles.



(c) There is another possible staggering of the control vectors. The staggerings best for decreasing peak power consumption and minimizing range of power consumption are not necessarily the same.

Figure 5.1: Power consumption of stages in a two-core five-stage system for different control vector staggerings.

In the previous example (Figure 5.1b), we staggered the bits in the control vectors to minimize the range of the control vector sum. There is one other possible staggering of the second control vector, $\{0, 0, 1, 0, 0\}$, that minimizes the range in the control vector sum. Note that any other staggering of bits where the first bit of the control vector sum is set will be equivalent to one of the two control vector sums presented so far: $\{1, 1, 0, 0, 0\}$ and $\{1, 0, 1, 0, 0\}$. Power consumption of the second staggering is presented in Figure 5.1c. While this staggering does not benefit peak power, it decreases the range in power consumption by an additional 2 W over the first staggering. The first staggering decreases the maximum power consumption, while the second

staggering increases the minimum by a larger amount than what peak power was decreased by the first. Optimal peak power and optimal range in power consumption cannot always be achieved simultaneously.

The previous two examples demonstrate that current control vectors that minimize the range of the control vector sum can decrease the peak and range of power consumption. Listing 5.1 presents a method for generating control vectors that minimizes the control vector sum for a two-core processor implementing k-hot pipelining. The method involves calling the function `initializeCore(core, k)` for each core `core` for the desired hotness `k`. `initializeCore`, in turn, calls the function `incrementK` which generates the appropriate control vector the core.

Listing 5.1: Function to generate control vectors that have decreased peak power and power variability

```
c[2] = {{0, 0, 0, 0, 0},  // control
        {0, 0, 0, 0, 0}}  // vectors

initializeCore(core, k) {
  for i in [0, k)
    incrementK(core)
}

incrementK(core) {
  c_sum = c[0] + c[1] // vector sum
  minVal = min(c_sum)
  for i in [0, len(c_sum)) {
    if ( c_sum[i] == minVal &&
         c[core][i] == 0 ) {
      c[core][i] = 1
      break
    }
  }
}
```

CHAPTER 6

METHODOLOGY

We evaluated k-hot pipelining using a seven-stage implementation of the DEC Alpha ISA simulated with GEM5 [37]. Microarchitectural parameters were chosen to be similar to the ARM Cortex-A7 [38] and are enumerated in Table 6.1. Power consumption of the baseline as well as the k-hot designs was evaluated using McPAT [39]. We simulated benchmarks from the Spec CPU2000 [40] and Spec CPU2006 [41] benchmark suites, fast forwarding for 1 billion instructions and executing for 1 billion instructions. To evaluate multi-core specific optimizations discussed in Chapter 5, we considered sixteen mixes of four of these benchmarks (Table 6.2) to model a four-core multi-core system with the assumption that the cores are independent.

Table 6.1: Core characteristics

Implementation	7d2w
Number Cores	1 in-order
Pipeline Depth	7
Pipeline Width	2
Register Files	32 Int, 32 FP
Fetch/Decode/Issue Width	2/2/2
BTB Size	4096 entries
RAS Size	16 entries
Branch Predictor	Tournament
ALUs/FPUs/MDUs	2/1/1
Cache Line Size	64B
L1 I$	32kB, 4-way
L1 D$	32kB, 4-way
L2 Unified $	1MB, 8-way
Memory Configuration	2 GB of 1066 MHz DDR3

McPAT outputs average power consumption of components in the full-hot processor. To calculate the power consumption of a k-hot pipeline, we use the values reported by McPAT and the k-hot control vector designating which stages should be powered on. Each stage requires a set of components to be powered on, and these sets may not be disjoint. Power consumed in a given cycle of k-hot operation is the power consumed by the union of all sets of required components.

State saving logic (e.g., the register file and caches) must always be powered on. In one set of evaluations, we assume that such logic is always operated at nominal voltage (1 V), even in the cycles they are not used. We also perform evaluations where such logic is operated at data retention voltage (DRV) (300 mV) [42] when they do not need to be accessed. We assume that SRAM leakage power scales cubically with voltage [43].

Table 6.2: Benchmark mixes used to evaluate multi-core systems

mix0	ammp, applu, apsi, art470	mix8	lbm, libquantum, lucas, mesa
mix1	applu, apsi, art470, facerec	mix9	libquantum, lucas, mesa, mgrid
mix2	apsi, art470, facerec, gromacs	mix10	lucas, mesa, mgrid, milc
mix3	art470, facerec, gromacs, lbm	mix11	mesa, mgrid, milc, omnetpp
mix4	facerec, gromacs, lbm, libquantum	mix12	mgrid, milc, omnetpp, soplex
mix5	gromacs, lbm, libquantum, lucas	mix13	milc, omnetpp, soplex, swim
mix6	omnetpp, soplex, swim, wupwise	mix14	swim, wupwise, ammp, applu
mix7	soplex, swim, wupwise, ammp	mix15	wupwise, ammp, applu, apsi

To estimate power gating overheads, we fully implemented power gating in openMSP430 [44] using an industry-standard unified power format (UPF) [45] methodology that accounts for all power gating overheads. We performed this implementation for different numbers of power domains. Based on this complete implementation of power gating on the openMSP430 [38], we estimate the overhead due to isolation, retention, header, and footer cells for our modeled processor to be 5%. We account for this overhead in all of our reported results. We discuss the effects of power gating overhead further in Chapter 7. We assume a zero cycle turn-on time for power domains but discuss the sensitivity of our results to power gating latency in Section 7.4.2.

To calculate performance of a k-hot pipeline, we use the active and idle cycle counts output by GEM5. The latency of a benchmark run on a k-hot pipeline is $\frac{k}{m} * c_{active} + c_{idle}$ where c_{active} is the number of active cycles,

c_{idle} is the number of idle cycles, and m is the pipeline depth. This is an overestimate of latency since cycles spent flushing the pipeline on branch mispredicts are not subtracted. It also does not account for the decrease in resource contention due to fewer mispredicts.

Figure 6.1 demonstrates how modules modeled by McPAT are either assigned to each of the seven pipeline stages (FETCH and MEM take up two pipeline stages, DECODE, WRITEBACK, and EXECUTE correspond to one stage each) or shared across multiple stages. We use this information to form power domains, so that the logic required by any pipeline stage could be powered on independently and no state information is lost. We divide our seven-stage processor into 12 power domains. Each of the seven modules that store processor state (Branch Predictor (BP), Branch Target Buffer (BTB), I-Cache, D-Cache, ITLB, DTLB, and register file) is in its own power domain. Although these modules are required to be powered all the time to retain their stored values, having their own power domains allows these modules to be put in DRV separately. The logic in "Always On" belongs to the core but is not specifically modeled by McPAT. We conservatively assume this domain is always powered on. Lastly, since each of the remaining modules (IFU, LSU, EXEU and MMU) is used by a different set of stages, as shown in Figure 6.1, each of them is put in its own power domain to ensure that a module is powered on if and only if a stage that needs that module is active.

The resulting power domains and their control vectors are shown in Figure 6.2. In the figure, the twelve rectangles correspond to the twelve power domains discussed earlier. The power domains are color coded to show different power behaviors: red means the power domain is always powered on; yellow means the power domain is held at DRV by default, and is fully powered on when enabled (we do present evaluations in Chapter 7 where these domains only operate at the nominal voltage); white means the power domain is powered off by default, and is fully powered on when enabled. The five circles (Fetch, Decode, Execute, Memory and Writeback) represent the control vector bit that corresponds to that stage. An arrow between a control vector bit and a power domain indicates the enabling relationship between a control vector bit and a power domain. Having multiple arrows coming into a power domain means having multiple enablers, and the power domain should be enabled if any of its enablers is active. Having multiple arrows coming out

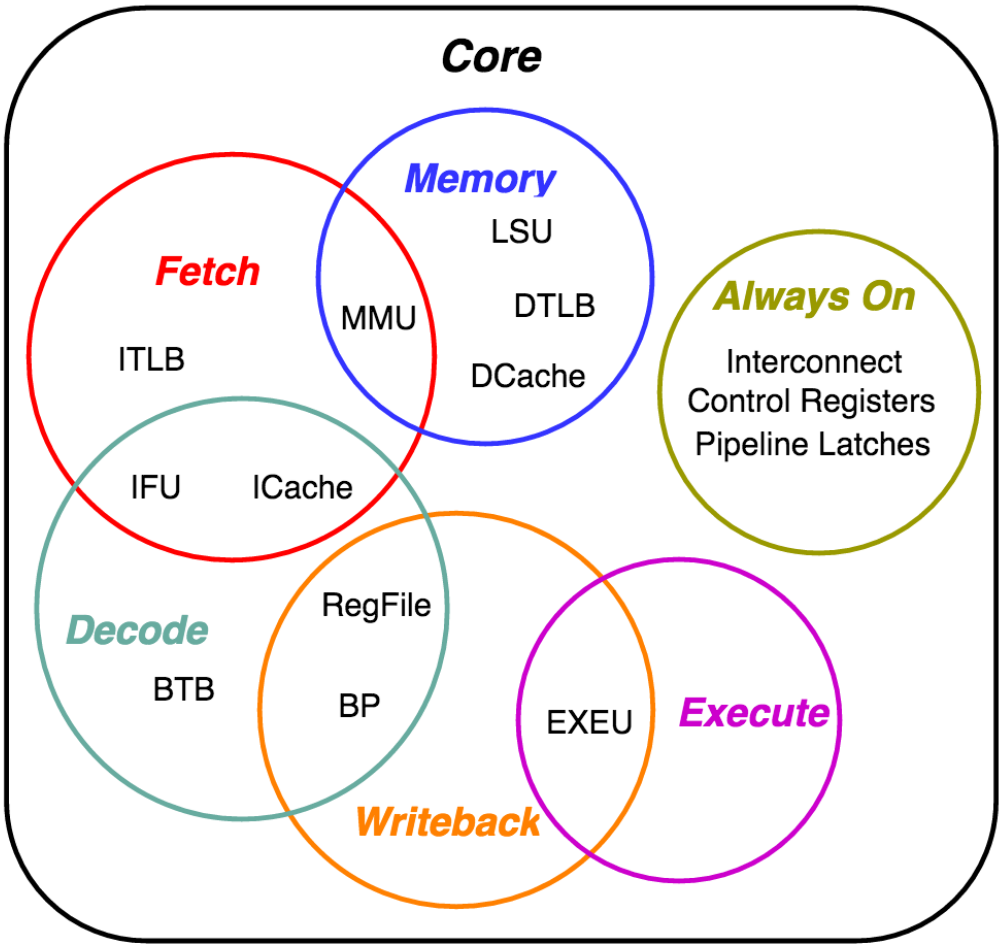


Figure 6.1: Modules required by each pipeline stage.

of a control vector bit means that the control vector enables multiple power domains when it is set.

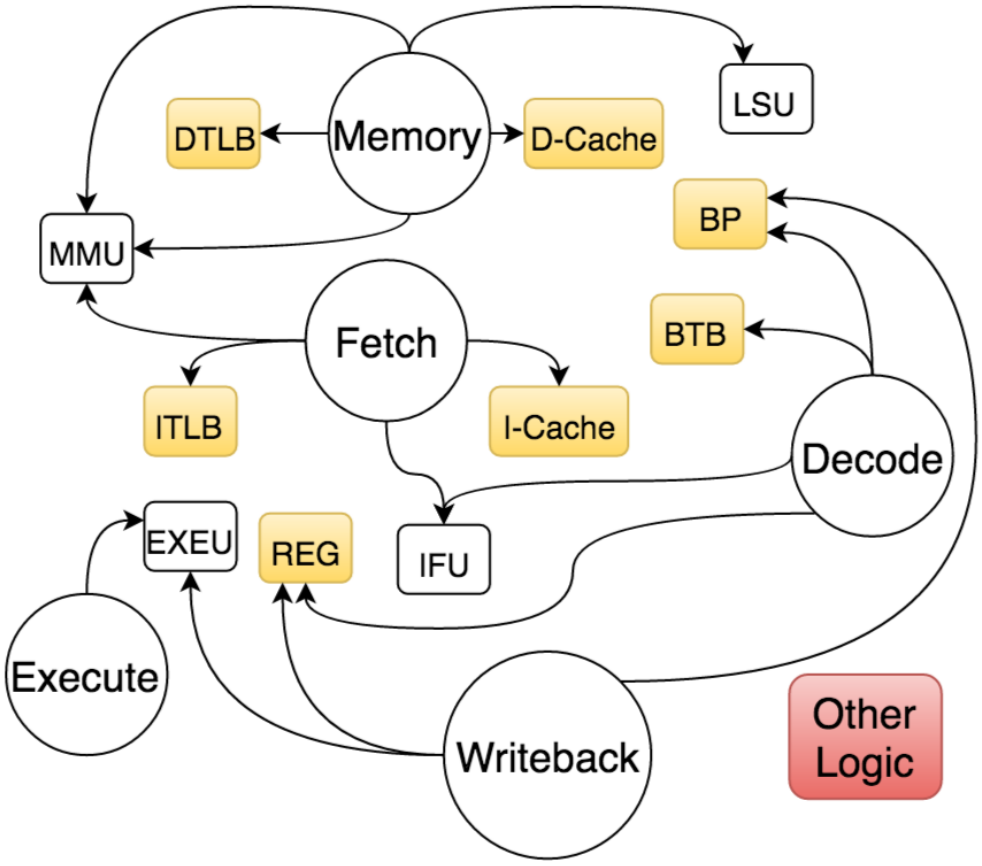


Figure 6.2: Power domains and their control signals.

CHAPTER 7

RESULTS

This chapter presents our evaluation of k-hot pipelining with and without DRV and shows that k-hot pipelining can allow significant power reduction over the baseline design (Section 7.1). We also analyze how performance of k-hot pipelining is affected by the choice of workload (Section 7.2). In Section 7.3, we show that our staggering-based multi-core specific optimizations allow significant reduction in the peak power and power variability of a four-core multi-core processor. In Section 7.4 we present sensitivity studies.

7.1 Power

Figure 7.1 shows the total power benefit of k-hot pipelining relative to a full-hot baseline for different benchmarks running on an in-order seven-stage two-wide superscalar design (Table 6.1). For k greater than one, there are multiple possible control vectors for the same k. Due to module sharing, these different control vectors result in variable power consumption (Section 4.1). The primary bars in Figure 7.1 show power reduction from k-hot pipelining when we use a single vector for each value of k that minimizes the average power across all benchmarks. The error bars show the power reduction corresponding to the best and the worst control vectors for each benchmark. For these results we assume that all the state preserving logic operates at 1 V even when it is not being accessed. On average, we see roughly a 32% decrease in power consumption for one-hot, a 13% decrease for two-hot, a 7% decrease for three-hot, and a 1% decrease for four-hot.

There are several reasons why we do not get the ideal savings. First, register files, both instruction and data caches, and the TLBs are not power gated at any time since they hold processor state. A reasonable but more aggressive implementation of k-hot would be to decrease the voltage of these modules

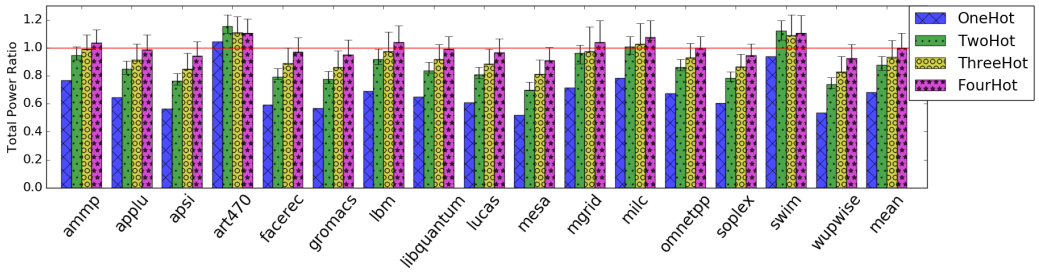


Figure 7.1: Total power benefits for k-hot across different benchmarks.

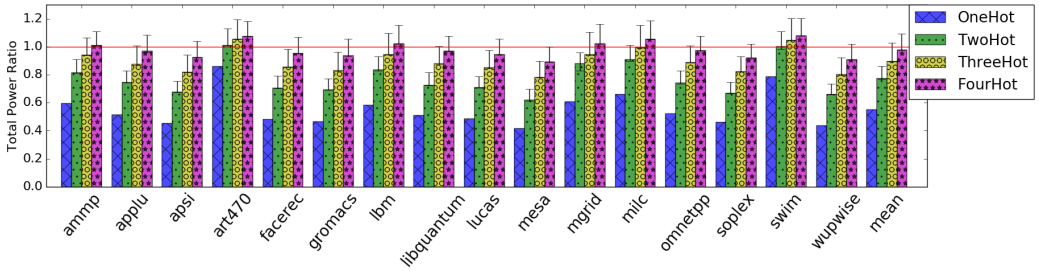


Figure 7.2: Total power benefits for k-hot with DRV across different benchmarks.

to the DRV when they are not being actively used. DRV reduces the leakage power consumed during idle cycles of these modules. Figure 7.2 shows the power consumption of a k-hot pipeline assuming a 300 mV DRV implementation. On average, we see roughly a 45% decrease in power consumption for one-hot, an additional 13% benefit compare to the implementation without DRV. The savings from DRV decrease as we move to larger value of k because the duty cycle of the applicable structures increases with k, and DRV can only be applied when the structures are idle.

Second, many modules are shared between multiple pipeline stages. For example, the execution unit, the most power hungry combinational component of the design, is used both in execution stage and the Writeback stage. During one-hot operation, the execution unit is powered on for two cycles per instruction compared to one cycle during full-hot operation. Therefore, the execution unit consumes double the energy per instruction.

Finally, our power calculations are conservative since we do not shut down modules that can be obsolete during k-hot (e.g., the branch predictor and forwarding network). Our calculations also do not account for resources consumed by mispredicted instructions. Previous work has estimated that resource consumption due to mispredicted instructions accounts for 16% of

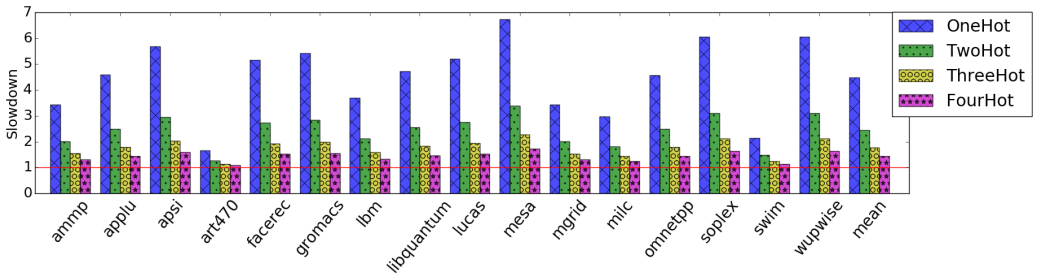


Figure 7.3: Performance degradation for k-hot across different benchmarks.

the total processor power of a six stage pipeline [1].

Note that all reported power values in this and other chapters assume a 5% power overhead for isolation, retention, header, and footer cells required for power gating (Chapter 6). Any additional power gating overhead will result in a straightforward subtraction of the additional percentage from the percentage benefits (e.g., an increase to 10% overhead would decrease the average savings from one-hot pipelining from 32% to 27%).

7.2 Performance Degradation

Figure 7.3 shows the performance degradation of k-hot pipelining relative to full-hot. The performance degradation we observe varies widely across different benchmarks. Without loss of generality, consider one-hot operation. The smallest performance degradation, 1.65×, is seen during execution of *art470*, a memory intensive benchmark (Figure 7.3). The worst performance degradation, 6.73×, is seen during execution of *mesa*, a compute intensive benchmark (Figure 7.3).

Most benchmarks do not see the $\frac{1}{7}\times$ worst-case performance degradation from one-hot pipelining. One reason is that the number of idle cycles is constant for all values of k. In other words, compared to full-hot pipelines, k-hot pipelines executing the same workload spend a smaller percentage of execution time stalling. As a result, performance of the processor is only slightly affected by k-hot when running memory intensive workloads like *art470* and *swim*, since most cycles during full-hot operation were idle, as shown in Figure 7.4. The other reason is that for low values of k, specifically one-hot and two-hot, an instruction is fetched after its previous instruction finishes the execution stage, where branches are resolved. Therefore, branch mispredic-

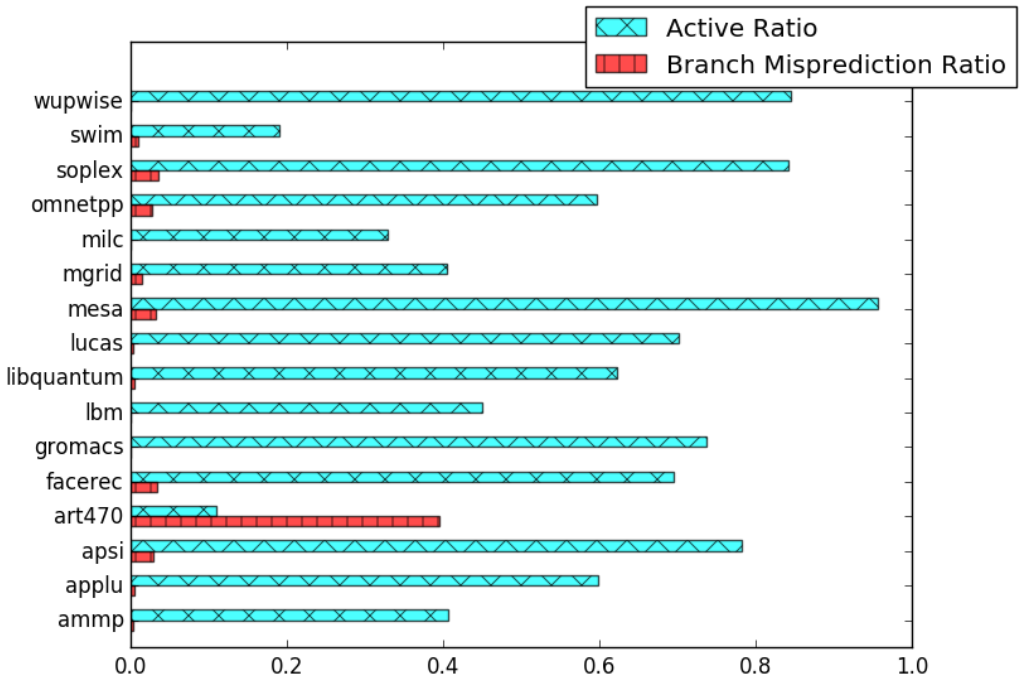


Figure 7.4: The fraction of time the processor is active and misprediction rate vary across benchmarks.

tion does not incur a penalty for one-hot and two-hot. The misprediction rates of the benchmarks we evaluated are shown in Figure 7.4.

7.3 K-hot in Multi-Core Processors

Chapter 5 introduced the possibility of benefits from staggering set bits across control vectors in multi-core systems. Staggering the set bits in control vectors minimizes the number of copies of any stage being on in the same cycle, which decreases the range and peak of power consumption in a system. Staggering can be accomplished by using the `initializeCore` function presented in Listing 5.1.

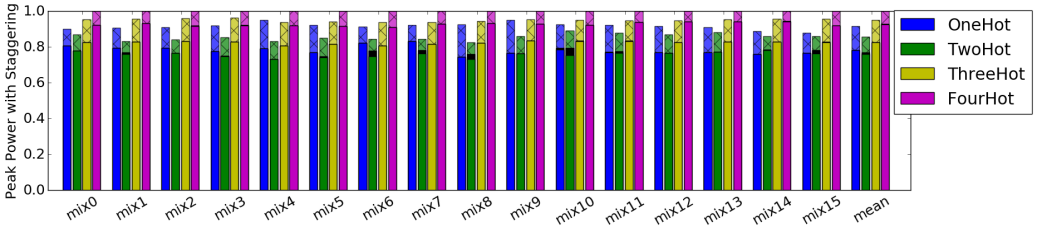


Figure 7.5: Peak power savings with control vector staggering and control vector sum range minimization.

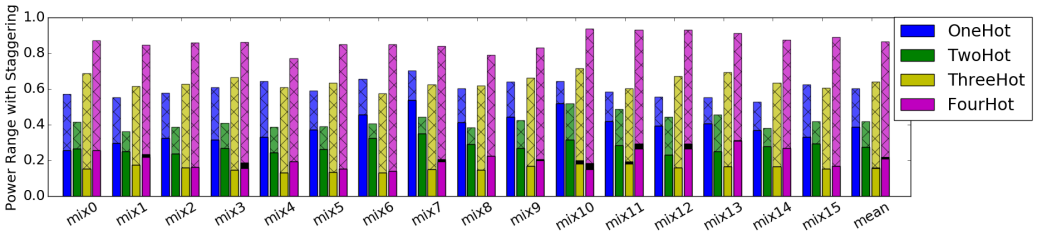


Figure 7.6: Power consumption range savings with control vector staggering and control vector sum range minimization.

Given four one-hot cores with seven stages each, the range in the control vector sum is minimized if there are four ones and three zeroes in the vector sum. Minimal range does not require a particular arrangement of those numbers, so there are $\binom{7}{4}$ unique staggerings that satisfy the minimal range requirement. All of these arrangements will have different power characteristics and so will benefit the peak and range of power consumption differently. Note that the best arrangements for peak power and power variability are not necessarily the same. Similar choices exist for other values of k.

To evaluate the goodness of a set of control vector staggerings on our multi-core system, we used the mixed workload discussed in Table 6.2. Every k-hot control vector has seven possible configurations since it rotates one bit each cycle. We assumed that the rotation of the control vectors in a multi-core system always occurs at the same time on every core, so the system also has seven possible control vector sum configurations. We measured the average power of a core during each of the seven possible control vector configurations. Then we found the average power consumed for the each of the seven possible control vector sum configurations by summing the powers of the cores during that configuration. We then report the maximum average power of any configuration as the peak power and the difference between the

maximum and minimum of any configurations as the power range.

To evaluate the savings possible using our algorithm in Listing 5.1, we found the peak and range of power consumption relative to no staggering for every set of control vectors that satisfy the minimal control vector sum range requirement. The peak and range savings are shown in the hatched bars of Figure 7.5 and Figure 7.6 respectively. There are many staggerings that minimize control vector sum range, so there is a range of possible savings represented by the hatched bar. The top of the hatched bar represents the lower bound of possible savings from Listing 5.1. The algorithm delivers 15% peak power savings and a 58% power range decrease on average in the best case of two-hot.

Minimal range in the control vector sum is neither a necessary nor sufficient condition for minimizing peak or range of power consumption. Therefore, we also show the maximum possible savings (though a brute force search of all staggerings) in the solid bars of Figure 7.5 and Figure 7.6. The maximum peak power saving is 24% and the maximum decrease in power range is 84% on average in the best case of two-hot.

7.4 Sensitivity Studies

7.4.1 Technology Node

The previous evaluations used the 65 nm technology node. In this chapter, we discuss sensitivity of our results to scaling by estimating benefits for a 45 nm technology node. Figure 7.7 shows the power benefits with DRV. The average power savings of k-hot pipelining are and 53%. On average, the benefits of k-hot pipelining using DRV are 3% higher for the 45 nm core compared to the 65 nm core. K-hot pipelining allows state-saving logic to spend more time operating at DRV. Since DRV decreases leakage, and leakage increases with scaling. Because leakage increases with scaling, we expect the benefits of k-hot pipelining to increase further with technology scaling.

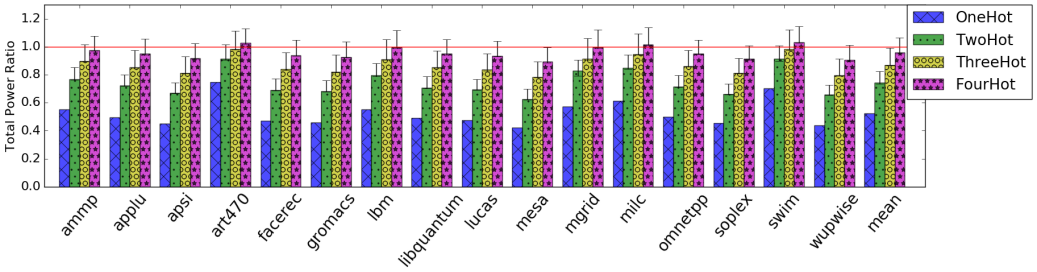


Figure 7.7: Sensitivity study on technology node - 45 nm, with DRV

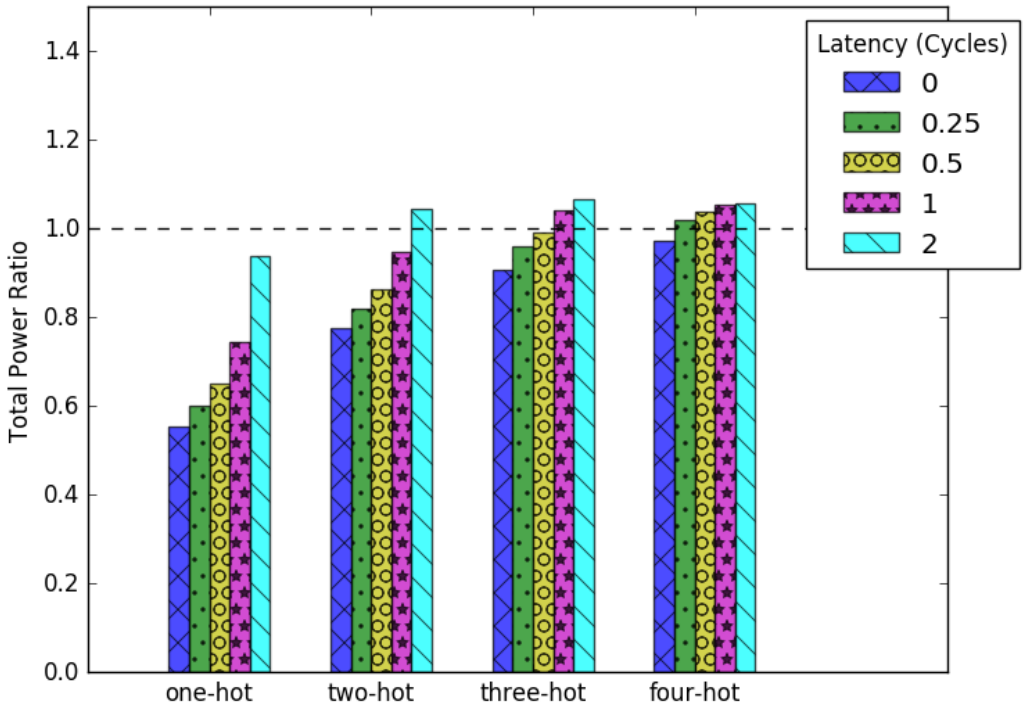


Figure 7.8: Sensitivity of power savings from k-hot pipelining to the latency of power mode transitions

7.4.2 Power Gating Latency

Results in this chapter are reported assuming zero power gating latency. Figure 7.8 shows the power consumption of k-hot pipelines relative to the full-hot baseline for different values of k and different power gating latencies. The latency is the time to either turn-on or turn-off, and the power consumption during both periods by a power domain is assumed to be half of the average power of the power domain when fully powered on. Since modules are shared between pipeline stages, there may be only one or two cycles between the time a module can start turning off and the time it needs to be back on. Any module that would not have time to turn both off and back on again was kept on.

The results show that k-hot pipelining cannot tolerate a large power gating latency since modules are turned on and off frequently. The frequency of turning on and off is even higher than just once every m cycles for an m-stage pipeline since modules are shared among pipeline stages. Since, for high values of k, the majority of the pipeline modules are almost always on anyway, the impact of increasing power gating latency decreases with increasing k.

To put the results in Figure 7.8 into context, conservatively assume a turn-on latency of 1.2 million gates to be 200 ns [46]. If we assume that the turn-off latency scales linearly with gate count, the latency of turning on and off a pipeline stages will scale linearly with the size of the stage. The largest stage of TI's MSP430 microcontroller [44] has 4.6k gates. Therefore, it will take 0.8 ns to power on. At 25 MHz (the top CPU frequency for MSP430), this corresponds to 0.02 cycles of power gating latency. The latency (in cycles) will be even lower in active low power states where frequencies are scaled. As another example, the largest stage of a 2-issue, 10-stage out-of-order FabScalar [47] pipeline is 36k gates. Therefore, it will take 6 ns to wake up the stage. At 667 MHz, its nominal frequency [47], this corresponds to about 4 cycles. For this processor, k-hot pipelining is useful only in active low power modes.

In general, k-hot pipelining is most effective for processors with relatively simple pipelines (i.e., relatively small number of gates per pipeline stage) or processors running at low frequencies. Since these characteristics are common across embedded processors targeting IoT, wearables, implantables, and

sensors, on one hand, and processors targeting high throughput computing on the other, we expect k-hot pipelining to be useful for a broad class of emerging applications.

7.4.3 Clock Gating-based Implementation of k-hot Pipelining

Our above results assume fine-grained power gating at the granularity of a single pipeline stage. In this chapter, we evaluate the effectiveness of a clock gating-based implementation of k-hot pipelining where all but k pipeline stages are clock gated. Since DRV is widely used in industry to reduce leakage of SRAM structures [48], we still assume the availability of DRV for the state-saving structures. Figure 7.9 shows the decrease in average power from k-hot pipelining implemented with clock gating and DRV relative to the full-hot baseline. We see that the benefits of clock gating paired with DRV are greater than the benefits of power gating alone. This result suggests that k-hot pipelining may be a beneficial low-cost choice for designers who would like to expand the set of power-performance points available in systems that already implement clock gating and DRV. The result also suggests that a clock gating-based implementation of k-hot pipelining may be a promising alternative to a power gating-based implementation for processors where power gating latency may be a concern (e.g., large processors or processors operating at high frequency). It may be particularly effective for processors operating at high clock frequencies since the dynamic power may be a bigger fraction of overall power.

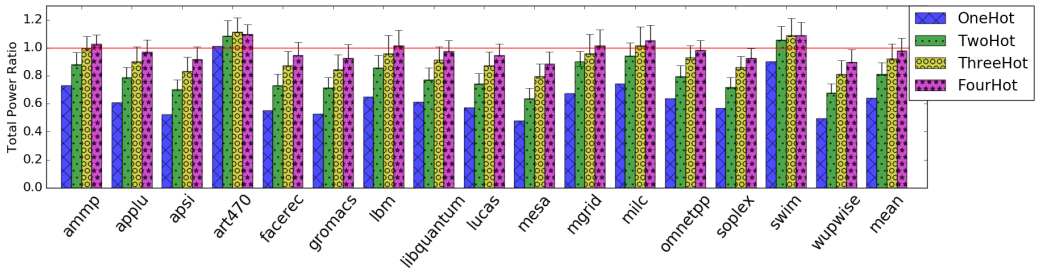


Figure 7.9: Benefits of k-hot pipelining with clock gating and DRV are similar to the benefits of power gating.

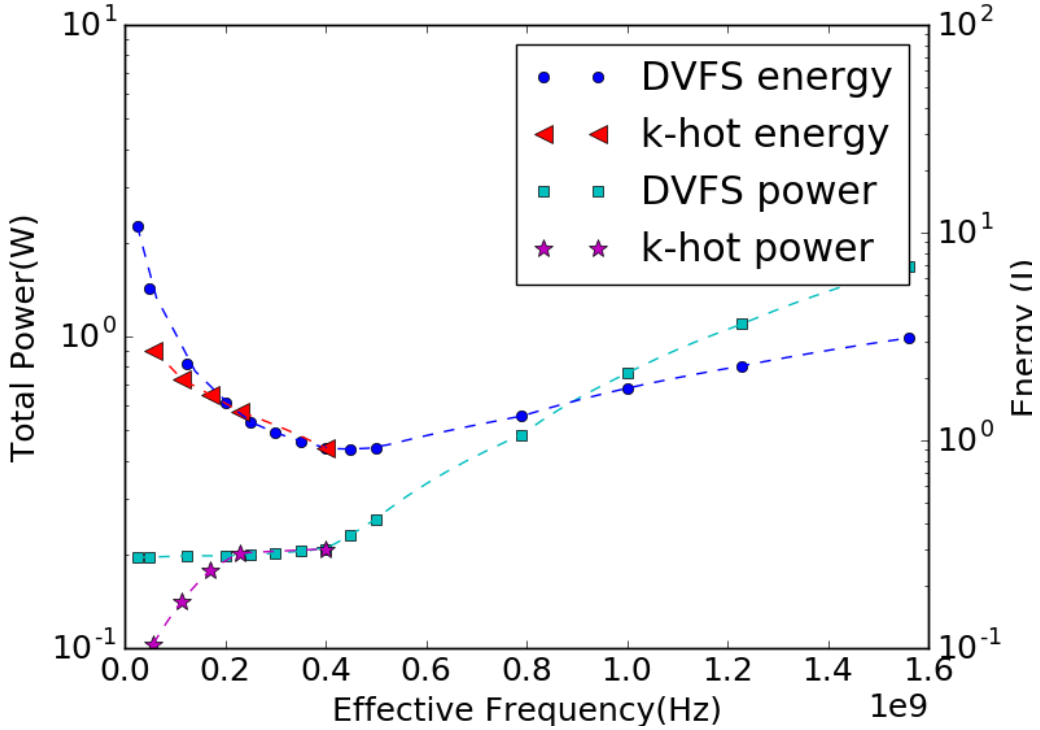


Figure 7.10: k-hot pipelining can continue to provide lower power-performance operating points after DVFS stops scaling due to reliability reasons.

7.4.4 Comparison against DVFS

Consider the DVFS relationship shown in Figure 7.10 where the minimum safe voltage for the processor is 500 mV [49]. We apply k-hot pipelining to the minimum voltage DVFS operating point. We are able to run at even lower power-performance points that are not supported by DVFS. Since the leakage power consumed by DVFS at the minimum voltage point is a lower bound on power consumed by DVFS for any frequency, k-hot can achieve up to 47% power savings over DVFS. Also, if frequency continues to scale after voltage stops scaling, energy consumption by DVFS increases. Figure 7.10 shows that by running k-hot at the first voltage minimum point (500 mV), k-hot saves up to 47% of energy compared to DVFS running at the same effective frequency. The results show that k-hot pipelining can be used both in conjunction with DVFS to provide higher dynamic range and resolution in terms of power and performance as well as a replacement for DVFS when voltage stops scaling.

CHAPTER 8

TOWARD IMPLEMENTING K-HOT PIPELINING FOR COMPLEX PROCESSORS

Previous chapters discussed k-hot for simple processors. In superscalar processors, multiple instructions can be in a pipeline stage simultaneously. Since instructions can spend a variable number of cycles in a pipeline stage due to hazards or variable stage latency, the control vector based implementation presented in Chapter 4 is not directly applicable. In order to support superscalar and other more complex processors, we would like a control scheme that is not as closely tied to the microarchitecture as control vector-based implementations.

Control schemes using instruction fetch throttling [50] make up one such class. By throttling the rate at which instructions are fetched in a processor with multi-cycle instruction latencies, we limit the utilization of the processor. Note that even the control vector scheme throttles instruction fetch, but also restricts the utilization of other components of the processor. By only limiting instruction fetch, we do not introduce additional requirements on the microarchitecture.

8.1 Fetch Throttling and Power Gating for k-hot Pipelining on Complex Processors

This section discusses two fetch throttling algorithms that control when to insert instructions into the accompanying power gating algorithm. We will denote w as the width of the superscalar processor, k as the desired *hotness* and n as the current number of instructions residing in non-fetch stages. We define a stage to be hot if it has made progress on at least one instruction during an arbitrary cycle.

8.1.1 Up-To-K-Hot

We call the first implementation Up-To-K hot(*up2k*). In a w-wide n-stage pipeline, one way to make sure there are always no more than k stages hot at any time is to limit the number of instructions in the pipeline to be up to k. This is achieved by constantly monitoring the in-flight instructions in non-fetch stages and using this number to make decisions on the number of instructions to be fetched of any cycle. The pseudo code of the *up2k* algorithm we have implemented is shown in Listing 8.1.

Listing 8.1: Algorithm of *up2k*

```
#Execution Algorithm - Up to K hot
# n - number of stages, w - width of stage
# m - current instructions in flight
if stage is not fetch:
  if stage is on:
    if stage has instruction ready to execute:
      execute instructions
    else:
      do nothing
if stage is fetch:
  if stage is on:
    if m < k:
      fetch min(w, k-m) instructions
    else
      do nothing
```

This approach is a conservative one as the actual number of hot stages at any given time would be no more than k. Given a specific processor, the peak power of the processor running in k-hot mode should never exceed the sum of the peak power of its k most power consuming stages. As a result, this approach helps when there is a need to have maximum power bound on the system.

8.1.2 Average-K-Hot

We call the second implementation Average-K hot(*avgk*). As opposed to the *up2k* approach, which sets an upper bound of the number of stages being hot, the *avgk* algorithm put a soft upper bound not only on the number of in-flight instructions but also on the number of stages that are currently hot.

In *avgk*, instructions are only fetched when both of the following conditions are met:

- The current number of instructions in non-fetch stages is less than $w*k$, where w is the width of the superscalar.
- The current number of hot stages is less than k.

Note that the number of hot stages at any time could exceed k (due to same stage dependence) or be less than k (due to squashing or instruction committing), but the average number of hot stages across a long execution should be approximately k. We have observed that for the vast majority of the time, the hotness of the pipeline is indeed k. The pseudocode of the algorithm of *avgk* is shown in Listing 8.2.

Listing 8.2: Algorithm of *avgk*

```
#Execution Algorithm - Average K hot
# n - number of stages, w - width of stage
# m - current instructions in flight
# u - number of stage currently hot(executing instructions)
if stage is not fetch:
  if stage is on:
    if stage has instruction ready to execute:
      execute instructions
    else:
      do nothing
if stage is fetch:
  if stage is on:
    if m < wk and u < k:
      fetch min(w, wk - m) instructions
    else
      do nothing
```

This approach is more aggressive than the *up2k* approach and it achieves much better performance with moderate extra power consumption. However, *avgk* does not guarantee a power bound.

8.1.3 Power Gating Algorithm

Since we no longer have a fixed control vector to control the power gating of each stage, a more complex power gating algorithm is required. In both

up2k and *avgk*, instructions that are already fetched into the pipeline are allowed to freely progress without any impediment. To ensure correctness, it is required that any pipeline stage about to execute an instruction should be powered on already and is left on throughout the duration of the instruction.

Thus we have come up with a conservative power gating algorithm that satisfies the safety requirement. It can be summarized as follows, given t is the powering on or off latency.

- If there is a chance that an instruction will arrive at the stage at an arbitrary cycle c, its powering on process needs to start at cycle $c - t$.
- When a stage finishes an instruction, do not power off if it can possibly be needed in less than $2 * t$ cycles.
- Power on fetch if number of in-flight instruction is less than desired due to pipeline squashes.

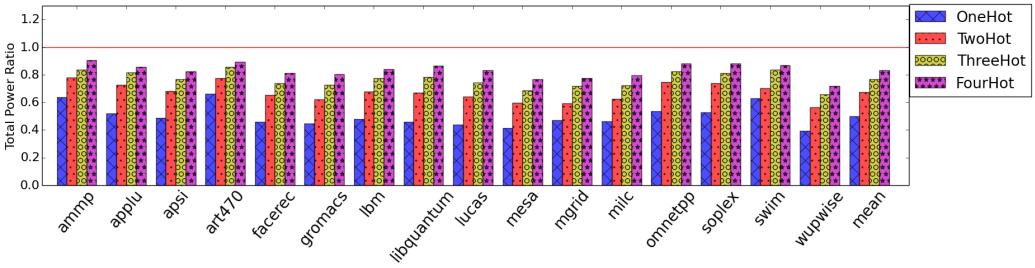
Since the number of cycles an instruction will stay at each stage is determined during run-time, the algorithm is conservative. We pay the price of extra power and energy when instructions take longer than one cycle at any stage, but we guarantee execution functionality and correctness. Listing 8.3 shows the pseudocode of our power gating algorithm.

Listing 8.3: Power Gating Algorithm

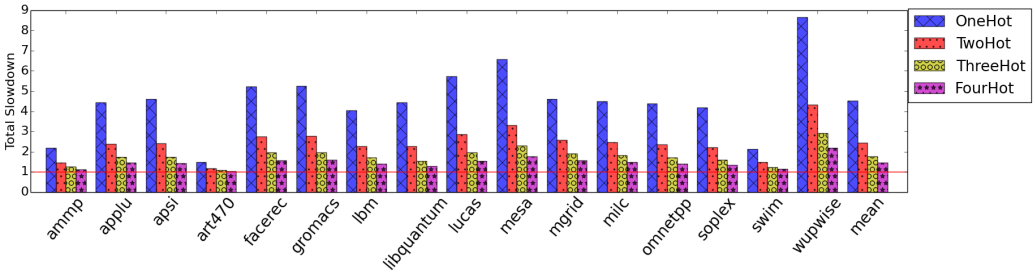
```
# Turn On/Off algorithm
# t - turn on/off latency
# m - current instructions in flight
# u - number of stage currently hot(executing instructions)
# +/- wraps around

if stage i is off:
  # To make sure a stage is on when needed
  turn on if any stage between stage i and i - t is hot

  if stage i is fetch:
    # when squash happens
    # also try to turn on if the fetch condition met
    # from the execution algorithm
    turn on if : (m < k) for up2k or ((m < wk) and u < k)
    for avgk

if stage i is on:
  #only turn off if there is enough time
  turn off if none of the stages between stage i and i - 2t
  is hot
```

8.2 Results

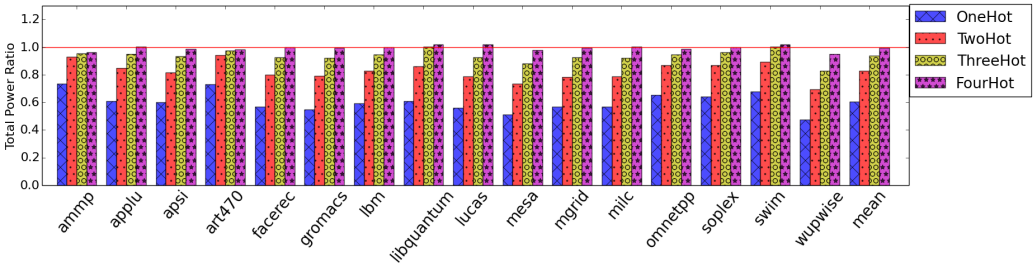
This section demonstrates the power saving and performance degradation of both *up2k* and *avgk* for a five-stage two-wide in-order processor. Figure 8.1a and Figure 8.1c show the power saving for *up2k* and *avgk* across different benchmarks, respectively. Figure 8.1b and Figure 8.1d show the performance degradation for *up2k* and *avgk* across different benchmarks, respectively.



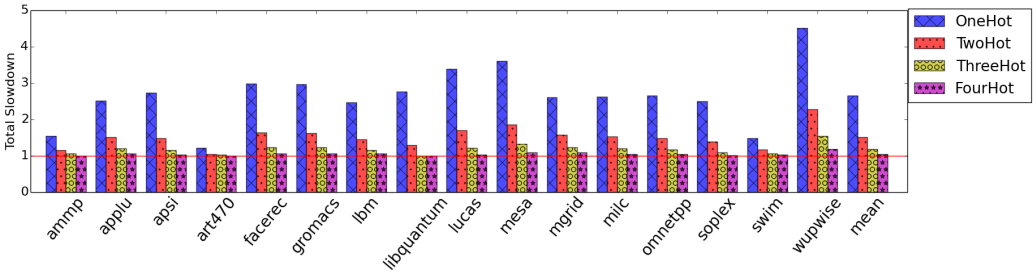
(a) Power savings for *up2k* across different benchmarks



(b) Performance Degradation for *up2k* across different benchmarks



(c) Power savings for *avgk* across different benchmarks



(d) Performance Degradation for *avgk* across different benchmarks

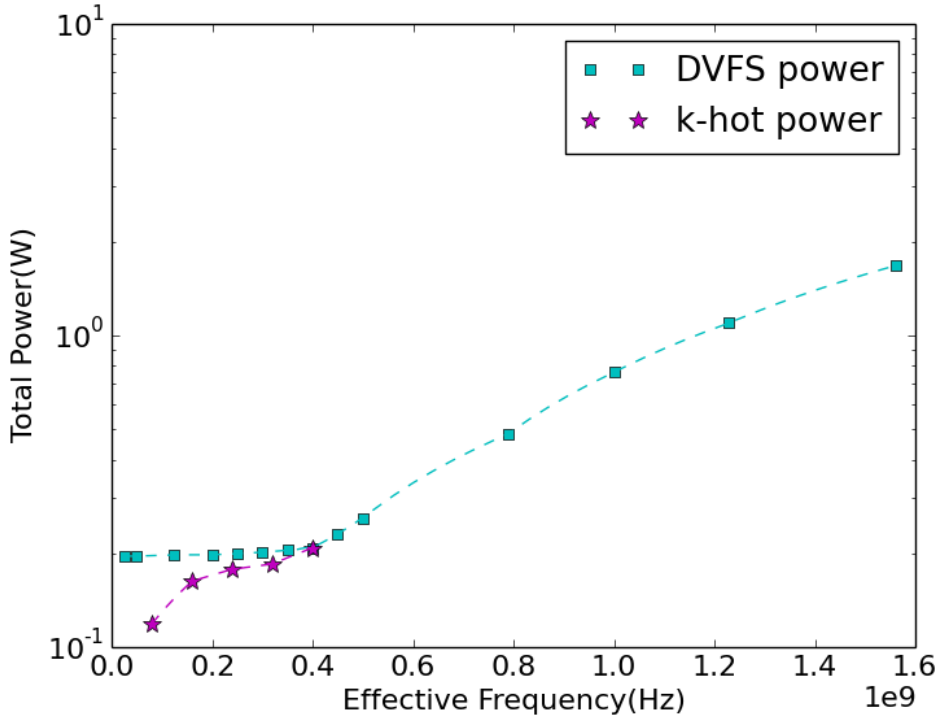
Figure 8.1: Implementation results.

Table 8.1 shows the power and performance data averaged across all benchmarks. As expected, compared to *up2k*, *avgk* has better performance but worse power savings, since it usually has more instructions in-flight and more stages hot for the same number of k.

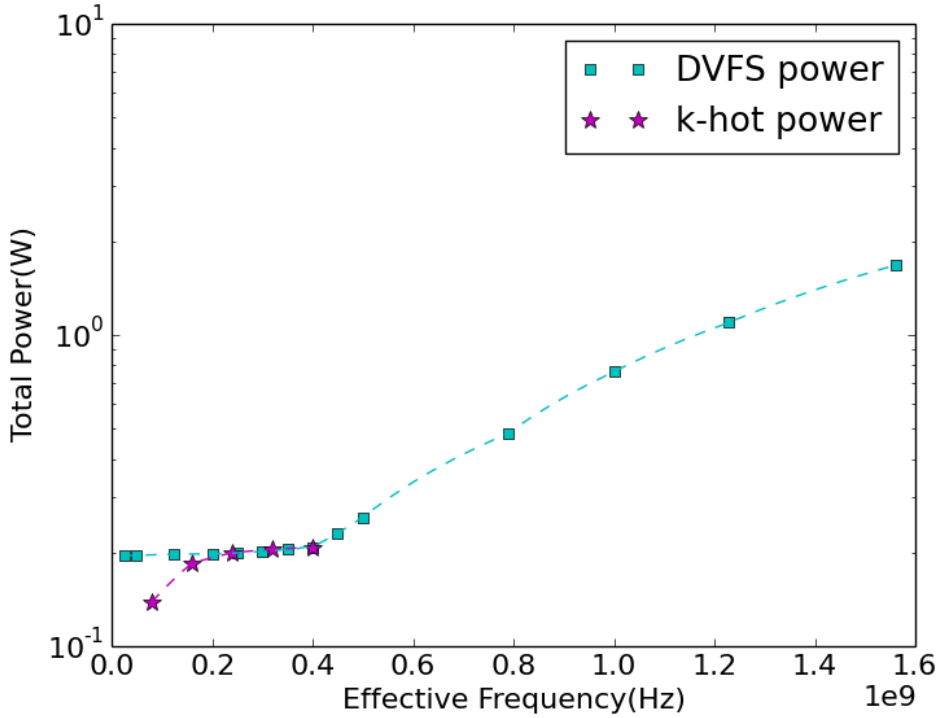
Table 8.1: Comparison between *avgk* and *up2k*

	Average Power Savings (%)				Average Performance Degradation			
	1-hot	2-hot	3-hot	4-hot	1-hot	2-hot	3-hot	4-hot
up2k	49.9	35.3	23.2	16.9	4.527	2.445	1.777	1.460
avgk	39.7	17.4	6.3	0.7	2.659	1.511	1.180	1.049

As mentioned in Section 7.4.4, due to reliability reasons, DVFS can only reduce voltage to a certain point. This constraint limits the power saving achievable by DVFS, and using k-hot pipelining on top of this operation point allows us to further trade performance for power. Figure 8.2a and Figure 8.2b show that by running k-hot at the first voltage minimum point (500 mV), *up2k* and *avgk* save up to 40% and 31% of total power, respectively. These results, combined with what was shown earlier in Section 7.4.4, show that k-hot pipelining could be attractive to applications that demand to trade performance for power beyond what is permitted by DVFS alone.



(a) Power savings for *up2k* at 500mV point



(b) Power savings for *avgk* at 500mV point

Figure 8.2: Power results of implementing *up2k* and *avgk* alongside DVFS.

CHAPTER 9

CONCLUSION

We presented k-hot pipelining, a novel technique for performing power-performance trade-offs in processors. The technique involves providing power and clock to only k stages of an m-stage pipeline and changing the k stages to be powered on as instructions flow through the pipeline. Since the remaining $m - k$ stages do not consume power, the technique results in power savings at the expense of performance. Our implementation of one-hot pipelining for a five-stage, two-wide in-order processor showed up to a 49.9% power reduction over the baseline design. Power reduction is up to 47% over the lowest power point supported by DVFS. Power benefits depend on the value of k and increase with technology scaling. As many of the current techniques to trade power and performance diminish in their effectiveness, k-hot pipelining may be used to supplement or replace these techniques.

REFERENCES

[1] J. L. Aragon, J. Gonzalez, and A. Gonzalez, "Power-aware control speculation through selective throttling," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings*, Feb. 2003, pp. 103–112.

[2] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: speculation control for energy reduction," in *ACM SIGARCH Computer Architecture News*, vol. 26. IEEE Computer Society, 1998, pp. 132–141.

[3] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: Managing server clusters on intermittent power," in *ACM SIGPLAN Notices*, vol. 46. ACM, 2011, pp. 185–198.

[4] A. Gandhi, M. Harchol-Balter, R. Das, J. O. Kephart, and C. Lefurgy, "Power capping via forced idleness," in *Proceedings of Workshop on Energy-Efficient Design*, 2009.

[5] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, 2008. [Online]. Available: http://link.springer.com/article/10.1007/s10586-007-0045-4

[6] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *ACM SIGARCH Computer Architecture News*, vol. 32. IEEE Computer Society, 2004, p. 276.

[7] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: adaptive DVFS and thread packing under power caps," in *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 175–185.

[8] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *IEEE Computer Architecture Letters*, vol. 8, no. 2, pp. 48–51, Feb. 2009.

[9] M. W. B. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Usenix Assoc, 1994, p. 13.

[10] D. Grunwald, C. B. Morrey III, P. Levis, M. Neufeld, and K. I. Farkas, "Policies for dynamic clock scheduling," in *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association, 2000, pp. 6–6.

[11] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. ACM, 2001, pp. 260–271.

[12] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," in *ACM SIGPLAN Notices*, vol. 38. ACM, 2003, pp. 38–48.

[13] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer, "Energy-conscious compilation based on voltage scaling," in *ACM SIGPLAN Notices*, vol. 37. ACM, 2002, pp. 2–11.

[14] E. Talpes and D. Marculescu, "Toward a multiple clock/voltage island design style for power-aware processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 5, pp. 591–603, May 2005.

[15] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*. IEEE, 2002, pp. 29–40.

[16] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34. IEEE Computer Society, 2006, pp. 78–88.

[17] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer, "Dimetrodon: Processor-level preventive thermal management via idle cycle injection," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 89–94.

[18] J. Corbet, "Idle cycle injection," LWN.net, Apr. 2010.

[19] Q. Gao, "Investigation of power capping techniques for better computing energy efficiency," thesis, Politecnico di Milano, 2014.

[20] D. Park, J. Lee, N. S. Kim, and T. Kim, "Optimal algorithm for profile-based power gating: A compiler technique for reducing leakage on execution units in microprocessors," in *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 2010, pp. 361–364.

[21] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*. ACM, 2004, pp. 32–37.

[22] A. Lungu, P. Bose, A. Buyuktosunoglu, and D. J. Sorin, "Dynamic power gating with quality guarantees," in *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, 2009, pp. 377–382.

[23] S. Turullols, "Multiple clock domain cycle skipping utilizing optimal masks to minimize voltage noise," U.S. Patent 20 140 095 909 A1, Apr., 2014, U.S. Classification 713/322, 713/501; International Classification G06F1/06, G06F1/32; Cooperative Classification H03K17/16, G06F1/08, G06F1/10, Y02B60/1217, G06F1/324.

[24] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal response to DVFS: analysis with an Intel Pentium M," in *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*. ACM, 2007, pp. 219–224.

[25] Wikipedia, "Athlon 64 X2," Feb. 2016, page Version ID: 704736864.

[26] J.-T. Wamhoff, S. Diestelhorst, C. Fetzer, P. Marlier, P. Felber, and D. Dice, "The TURBO diaries: Application-controlled frequency scaling explained," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 193–204.

[27] C. Kim, "Circuit reliability: Mechanisms, monitors, and effects in near-threshold processors," ISCA, 2014.

[28] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[29] H. Wong, "A comparison of Intel's 32nm and 22nm Core i5 CPUs: Power, voltage, temperature, and frequency," Oct. 2012. [Online]. Available: http://blog.stuffedcow.net/2012/10/intel32nm-22nm-core-i5-comparison/

[30] Intel, "Intel 64 and IA-32 Architectures Developer's Manual: Vol. 3b," Dec. 2015.

[31] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H.-S. P. Wong, "Device scaling limits of Si MOSFETs and their application dependencies," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–288, Mar. 2001.

[32] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 189–200.

[33] K. Usami and N. Ohkubo, "A design approach for fine-grained run-time power gating using locally extracted sleep signals," in *International Conference on Computer Design, 2006. ICCD 2006*, Oct. 2006, pp. 155–161.

[34] B. H. Calhoun, F. A. Honore, and A. Chandrakasan, "Design methodology for fine-grained leakage control in MTCMOS," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03*, Aug. 2003, pp. 104–109.

[35] Changbo Long and Lei He, "Distributed sleep transistor network for power reduction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 937–946, Sep. 2004.

[36] M. Anis, S. Areibi, M. Mahmoud, and M. Elmasry, "Dynamic and leakage power reduction in MTCMOS circuits using an automated efficient gate clustering technique," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 480–485.

[37] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[38] ARM, "Cortex-A7 MPCore Technical Reference Manual," 2012.

[39] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42*, Dec. 2009, pp. 469–480.

[40] J. L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *Computer*, vol. 33, no. 7, pp. 28–35, July 2000.

[41] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[42] D. E. Holcomb, A. Rahmati, M. Salajegheh, W. P. Burleson, and K. Fu, "DRV-Fingerprinting: Using data retention voltage of SRAM cells for chip identification," in *Radio Frequency Identification. Security and Privacy Issues*. Springer, 2012, pp. 165–179.

[43] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*. IEEE, 2008, pp. 203–214.

[44] O. Girard, "OpenMSP430 project," 2013. [Online]. Available: http://opencores.org/project,openmsp430

[45] "IEEE standard for design and verification of low-power, energy-aware electronic systems," *IEEE Std 1801-2015 (Revision of IEEE Std 1801-2013)*, pp. 1–515, Mar. 2016.

[46] P. Royannez, H. Mair, F. Dahan, M. Wagner, M. Streeter, L. Bouetel, J. Blasquez, H. Clasen, G. Semino, J. Dong, D. Scott, B. Pitts, C. Raibaut, and U. Ko, "90nm low leakage SoC design techniques for wireless applications," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, Feb. 2005, pp. 138–589 Vol. 1.

[47] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-Abadi, and E. Rotenberg, "FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 11–22.

[48] Texas Instruments, *CC253x and CC2540/41 User's Guide*, Apr. 2014.

[49] G. Ruhl, S. Dighe, S. Jain, S. Khare, S. Yada, A. V, P. Salihundam, S. Ramani, S. Muthukumar, S. M, A. Kumar, S. Kumar, R. Ramanarayanan, V. Erraguntla, J. Howard, S. Vangal, P. Aseron, H. Wilson, and N. Borkar, "An IA-32 processor with a wide voltage operating range in 32nm CMOS," Microprocessor & Programming Research, Intel Labs.

[50] H. Wang, Y. Guo, I. Koren, and C. M. Krishna, "Compiler-based adaptive fetch throttling for energy-efficiency," in *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, March 2006, pp. 112–119.