Report of *Power Balanced Pipel*es

Yiwei Yang

1 Introduction

As I learned in the *CS120 Parallel Computing*, [?], static scheduling is an NPC problem that is equivalent to the makespan problem, it's unique to the job list is already desirable during the process of compilation, while dynamic scheduling can deal with tasks that are created by the processes dynamically. There is roughly 2 method for dynamic load balancing:

- difusion
  - If a process has too many tasks, it sends some to its neighbors. If a neighbor becomes overloaded, it does the same thing.
  - Eventually, load spreads out and equalizes.
  - But might take a long time and cause lots of communication.
- work stealing - Processes without work steal work from processes with work.
  - In work stealing, each process maintains a double-ended queue (deque).
  - The rocess performs the task on the top of the deque.
  - If the process creates a task, it pushes it onto top of the deque.
  - If the process's deque is empty, it needs to load balance.
    - It picks a random other process and steals a task from the bottom of that process's deque
    - This minimizes (but doesn't completely avoid) contention between the two processes since one takes tasks from the top and one from the bottom.
  - Work stealing doesn't incur any overhead when processes have tasks.
  - Overhead when stealing is also borne by idle processes.
    - In contrast, for work pushing busy processes incur overhead for load balancing.
  - Work stealing is used in the Cilk parallel runtime.

A good example [?] of Tomasulo's algorithm in dynamic scheduling register and instruction is

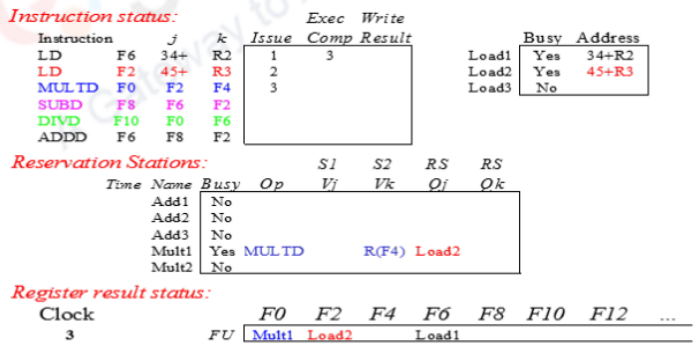

Figure R-1: The starting state for the example

- First to declare that the sequence of instruction considered along with the three data structures maintained. Assume 3 add reservation stations and 3 mul reservation stations. They are also 3 load buffers. Assume that add has 2 unit time latency and mul has a 10 unit time latency.
- The first 3 cycles are to complete Load1, Load2 and set them busy. Then Mul is issued. It is allocated Mult1 entry, the contents of register F4 are fetched and stored in the place of Vk. The other operand F2 cannot be fetched now and has to wait for Load2 to finish. This is indicated in the Qj field as Load2. Mult1 reservation station is marked busy. The register result status shows that register F0 is to be written by the instruction identified by Mult1.
- After another 3 cycles, after each of 3 instructions are issued, there exists a name dependency for F6 for the Add instruction with both the Div as well as the Sub instruction, which is WAR, but the operand F6 for both the Sub and Div is read and buffered in the reservation status, so there's no need to rename.
- The Sub instruction finishes execution in the seventh clock cycle since it has a latency of 2. Nothing else happens in this clock cycle. It writes into the CDB in the eighth clock cycle. The result of Sub is written into the register F8 and it also writes into the reservation station Add2 of the Add subtraction. The Add1 entry is now freed up.

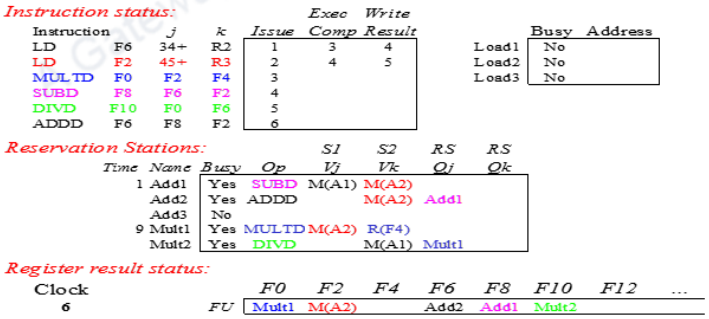

Figure R-2: The ending state for the example

The idea of dynamic load balancing overpower is borrowed from the load balancing on the top and in terms of power consumption. Before reading the paper, from my perspective, the first step is to quantify the power consumption and make a online algorithm for it, and control the power by dynamic tweak the frequency of the core.

According to the paper, they observe that pipeline delay balancing is a key feature of processor power. The reason may lies in that each microarchitectural pipeline stage in a delay balanced processor gets the same amount of time to complete. For power-optimized proc that circuit and design-level optimizations reclaim all timing slackto save power, they make a scheduling imbalancement between power-intensive micro instructions and non-intensive ones even with the same frequency. The author use cycle stealing policy of the power balancing to reduce power while guaranteeing the same frequency, thereby significantly improving power efficiency.

2 Metrics

The power break-down is based on a full design-level layout and evaluation of processor RTL. For the following graph power consumption varies from 0.3% of the total processor power in the Dispatch stage to almost [illegible] in the Fetch state. Different microarchitural microarchitectural pipe stages has a large power imbalance, too.

2.1 Voltage Comparator

The complexity of the work is in correlation with the Voltage percentage when running the process. They define the Comparator for which instruction is taking up more power by computing the ΔV

$$\Delta P_{hi} + \Delta P_{lo} < 0 \Longrightarrow \left|\frac{\Delta V_{hi}}{\Delta V_{lo}}\right| > \frac{P_{\text{lo,leak}} + P_{\text{lo,dyn}}\left(2 + \frac{\Delta V_{lo}}{V}\right)}{P_{hi,\text{leak}} + P_{hi,dyn}\left(2 + \frac{\Delta V_{hi}}{V}\right)}$$

The equation is a linear one if we assume that $|\Delta V_{hi}| \approx |\Delta V_{lo}|$, which will not trigger extra overhead. Note that the parameter of the delay of one pipeline stage can be defined statically, so the model is easily doing some static modulation.

2.2 Voltage-Delay Relationship

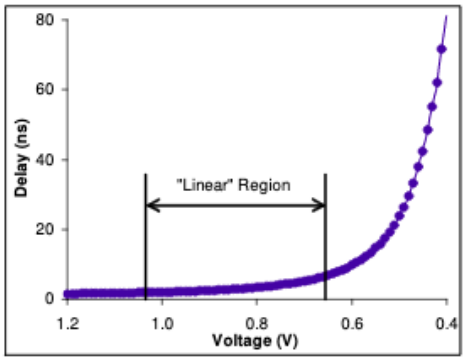

Figure R-3: The linear part of the voltage-delay relationship

The circuit delay can be described as Delay $\propto \frac{V}{(V-V_t)^c}$ which is inversely related.

3 Implementations

Designs are implemented with theTSMC65GP standardcell library (65nm), using Synopsys Design Compiler for synthesis and Cadence SoC Encounter for layout. FP benchmarks are primarily to evaluate dynamic power balancing on the FabScalar processor. The SRAM part is not taken into consideration by assuming it's already optimized at the lowest possible power consumption.

3.1 Static Power Balancing

Balancing during design.

3.1.1 Design-Level Power Balancing

It takes as input the hardware description (RTL) for a pipeline,the desired operating frequency (f), and the number of allowable voltage domains (NV), chooses the implementa-tion and voltage for each mi-croarchitectural pipeline stagesuch that processor power is minimized and the throughput target is met, and performs synthesis, placement, and rout-ing (SPR) to implement the power balanced pipeline. Since design-level power balancing changes how the processor is synthesized, benefits over a delay balanced pipeline can be in terms of both power and area reduction.

The algorithm to find a minimum-power is by a heuristic recursively updating the stage power consumption and make an output of the best way currently.

Algorithm 1 Exhaustive Power Balancing Algorthm

```
1. find_valid_datapoints(stage, loop_data);
2. for each datapoint ∈ valid_datapoints do
3.     update_stages(stage_data_copy, datapoint);
4.     update_loops(loop_data_copy, datapoint);
5.     if stage = NUM_STAGES then
6.         calculate_power_and_save(stage_data_copy);
7.     else
8.         recurse(stage + 1, stage_data_copy, loop_data_copy);
9.     end if
10. end for
```

Figure R-4: Algorithm 1

This process is similar of getting a clock tree synthesis from vivado after done the cpus. It'll generate a power summary and help you to optimize it.

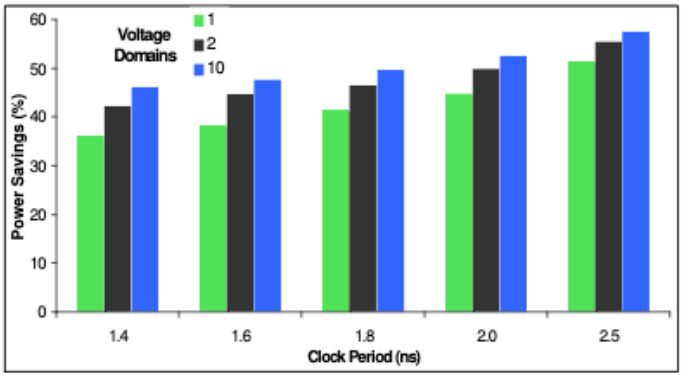

Figure R-5: Result for the Design-level power balancing

the power savings afforded by balancing pipeline power rather than delay can be significant, even when only a single voltage domain is allowed. Power savings increase for higher clock periods because designs are less tightly constrained at higher clock periods, allowing more flexibility to perform cycle time stealing. This is especially helpful for design-level power balancing, because the added flexibility allows more options for trading power and delay by changing the design implementation, which may be more efficient in some scenarios than changing the voltage.

3.1.2 Post-Silicon Static Voltage Assignment

Once the chip is designed, we can also add another voltage assignment to do power balancing. Overheads can be reduced by limiting the number of voltage domains, or even by implementing the cycle stealing and voltage assignment strategy at design time. o enable post-silicon voltage assignment and delay adapta-tion, support for dynamic voltage scaling (DVS) or multiple voltage domains and tunable delay buffers [10] is incorpo-rated into the design. Once the power balancing strategy is chosen, inputs to delay and voltage select lines are set or fuses are burned to finalize the cycle time stealing strategy for the chip.

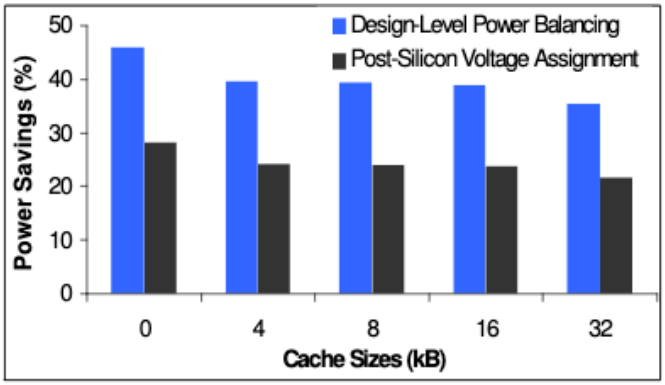

Figure R-6: Result for the Post-Silicon Static Voltage Assignment

Compared to the design level optimizations, Post-Silicon is not that useful.

3.2 Dynamic Balancing

For processors in which the relative power breakdown between pipeline stages may change due to changes in theworkload,dynamic power balancing may afford additional power reduction over static power balancing. The algorithm is to utilize the input and find the best way of optimized value.

Algorithm 2 A Fast Gradient Descent-based Power Balancing Heuristic for Reducing Time Overhead

1. **for each** $stage$ **do**
2. $\quad stage\_data[stage].voltage = MAX\_VOLTAGE;$
3. **end for**
4. **while** $(stage = max\_savings\_stage(stage\_data)) \neq -1$ **do**
5. $\quad stage\_data[stage].voltage = stage\_data[stage].voltage - v_{step};$
6. $\quad update\_loops(stage\_data);$
7. **end while**

Figure R-7: Algorithm 2

3.2.1 Cycle Time Stealing

Once the online value of the cycle time fraction is lower, one instruction will steal cycle time from other instrucitons to make sure different flip-flops' $D_{\min} \geq \delta_f - \delta_i + T_{\text{hold}}$

3.2.2 Local Voltage Scaling

While power balancing does not require multiple volt-age domains, benefits may improve with local voltage scaling then scaling cross Voltage domain.

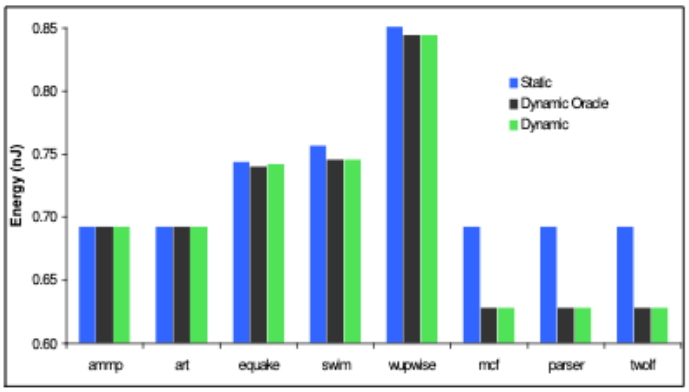

Figure R-8: Result for Dynamic Balancing

For INT benchmarks of FPU, we have that multiple Voltage domain does not make big difference and it's worse than the static load balancing.

4 Summary

The design-level power balancing, is a sound approach to significantly decrease power usage. As I declared in the Introduction, I think this paper is just applying dynamic scheduling in the hardware pipelines. This requires the definition of the metrics and useful algorithm. They actually add the Tomasulo's idea when Implementing the dynamic scheduling part. And their Design-Level Power Balancing and Post-Silicon Static Voltage Assignment can be designed as a profiling tool to test the design of the pipelined CPUs like during the RTL process. The implementation actually lives up to my expectation.

The problem with this article is that the linearity of the voltage vs. power relationship he gives is not clearly stated. Also, as quantum effects become larger and larger at very small processes, the focus of this article shifts today, and whether this relationship continues to work at even smaller processes is a topic that could be investigated.