CSE 237A Introduction to Embedded Systems

Winter 2015

Final Project: Kernel Level DVFS Policy for Qualcomm 8660

Andreas Prodromou
A53049230
aprodrom@eng.ucsd.edu

David Lane
A53064662
dlane@eng.ucsd.edu

Motivation

The power budget of a device is a critical component of embedded system design. It directly leads to the amount of heat that a device outputs, which has several negative impact, such as heat that the user feels, time between battery charges, and longevity (reliability) of components on the device.

Power can be managed in a number of ways. Architectural changes can make a device more efficient or better suited for a specific purpose; using a hierarchy of power states can decrease the amount of static power consumed; and using dynamic voltage and frequency scaling (DVFS) can lessen the amount of dynamic power consumed by a device. This project focuses on the implementation of a kernel level governor policy to limit power consumption.

Related Work

The premise of a DVFS policy is rooted in the observation that memory access times are not dependent on CPU speed. Therefore, if a process is frequently going to main memory, then its performance will is not related to the CPU speed and power savings can be made by lowering the CPU speed.

There are a number of papers that work to exploit this observation, however there is a definite challenge in how to use available information about recent behavior to accurately and efficiently predict future behavior. In [1], the relationship between types of memory accesses is explored and a difference between data and instruction level misses is quantified for an out-of-order superscalar. They show that during instruction stalls there is still a large amount of work that can be done if the instruction fetch pipeline is full. They also show that multiple

data misses do not add sequentially, and that useful work can get done with subsequent data misses due to the pipelined architecture.

In [2], the authors talk about the idea of quantifying the miss rates in terms of miss per instruction executed vs. the absolute number of misses counted. They introduced the term Memory Access Rate as cache miss per instruction, as opposed to a total miss rate. This offered the insight that a given amount of misses at a low frequency is more indicative of a memory bound process than the same amount of misses at a higher frequency. One perceived drawback of their implementation though was the availability and use of counters. They only monitored the counter that monitored data cache misses.

The main ideas for our governor policy came largely from these two papers. We included the concept of putting all absolute counter values in terms of the total instructions executed during the time quantum like is mentioned in [2], however we wanted to make use of the additional counters we had available. The authors in [1] made the argument that instruction level memory accesses have more impact on a DVFS policy because of the underlying architecture of an out-of-order superscalar. We expanded upon this because of an intuitive notion that instruction cache misses are more predictive in nature, and that an instruction miss corresponds to a jump in the program which will lead to several subsequent cache misses.

DVFS Governor Policy

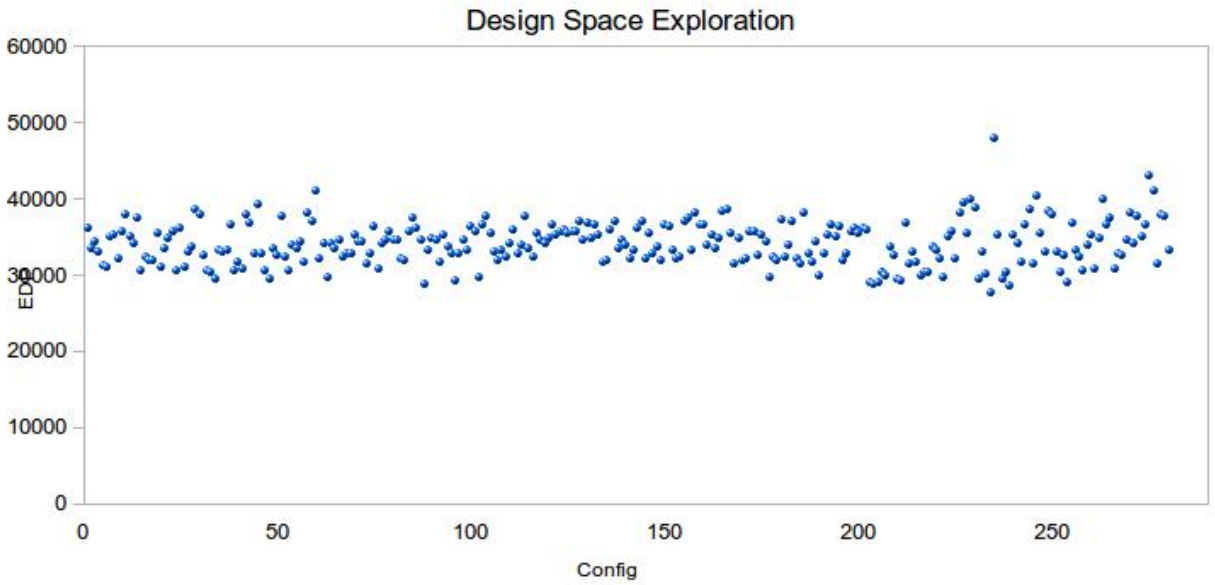
Our policy monitors four architectural counters. For each counter we developed a tiered structure where a given counter value would correspond to a decision on where to set the frequency. To avoid a ping-ponging situation between very high and very low values, we limited the size of any decision to a value of {+3, +2, +1, 0, -1, -2, -3} from the current frequency setting. Then to make a final decision, we use a weight for each of the four competing target values to come up with the value that we set the frequency to.

To determine the set points for each individual counter was difficult, as we used a combination of intuition (TLB misses are far less frequent than L1 cache misses; instruction cache misses are less frequent that data cache misses), and monitoring the counter values under various loads. To achieve the weighting factors, we created a script to evaluate the performance of a single test load with all combinations of weighting factors possible with the rules of:

- all weights are in increments of 10 (0 - 100)
- the collection of weights must sum to 100

As the base benchmark for our design-space exploration, we used one of the interactive workload provided in this class for the second small project, specifically one that plays the game Angry Birds. We accepted that benchmark as a representative example of mainly used smartphone applications (i.e. games).

After trying all possible weighting factors (total of 286 combinations) and plotting them against the EDP, we were able to choose the best combination for our final governor policy. The results of this exploration is shown in the following figure:



We observed the minimum EDP measurement, under configuration #234, which corresponds to the weights 50, 0, 30, 20 for our four counters respectively. It is interesting to note that zero weight is given to the second counter. Even though this seemed strange at first, several configurations with the same characteristic tend to provide good (low) EDP measurements.

This result suggests that counting the number of “instruction TLB misses” is not beneficial or useful when deciding on a new voltage level. We found this result to be counter-intuitive to our expectations. On the other hand, it is possible that our benchmark has a very low instruction TLB misses count, resulting in an insignificant measurement.

Hardware Components

The main hardware decision was which architectural counters to use in our policy. We decided on:

- 0x01: This monitors the L1 instruction cache misses.
- 0x02: Instruction TLB misses
- 0x05: Data TLB misses
- 0x10: Branch mispredictions

We chose TLB misses because they are particularly expensive and require two accesses to main memory (once to find the page table, another for the data). Because of the insights offered in [1], and our intuition on instruction caches, we choose to also include the L1 I

cache. We also felt that branch misprediction as a good indicator of future memory access because it likely means that the progress of the program has changed and will result in several data cache misses

Software Components
The software component of the project consisted of writing our DVFS policy and inserting that module into the kernel running on a Qualcomm 8660.

We encountered a number of difficulties with this process. For several iterations, we failed to get the kernel code for the 8960 to compile. The 8960's more up-to-date kernel, did not export the necessary symbols (in our case functions) required by our DVFS kernel module. After several attempts to get the kernel module to work, we decided to use 8660 phones instead. Eventually, we switched to the 8660 which we were able to find some appropriate fixes and get to compile.

As for the DVFS policy, there were several design decisions.

- The sampling frequency was kept at 10 milliseconds for sampling and 1 second for implementing a decision. This is a reasonable balance between responsiveness (too infrequent) and overcorrection (too frequent).
- The decision to have multiple counters each competing with separate choices seemed like a logical approach that could be reasonably implemented. It did have two implications though:
    - It required that we have a threshold for each counter value to be translated into a "vote" for which frequency setting (+3 through -3 from the current setting). This required some finesse as it would not be appropriate to have a single translation table for all counters since L1 misses are far more frequent that TLB data misses that are still more frequent than TLB instruction misses. This required some using of the phone under general work loads to gain an intuition of what might be reasonable cutoff points.
    - It also lends itself to a weighting method that lets each of the four counters compete to influence the actual policy. We liked this aspect because it allowed us to automate a script to test weighing possibilities across the full range of options, giving an ideal policy for a particular workload.

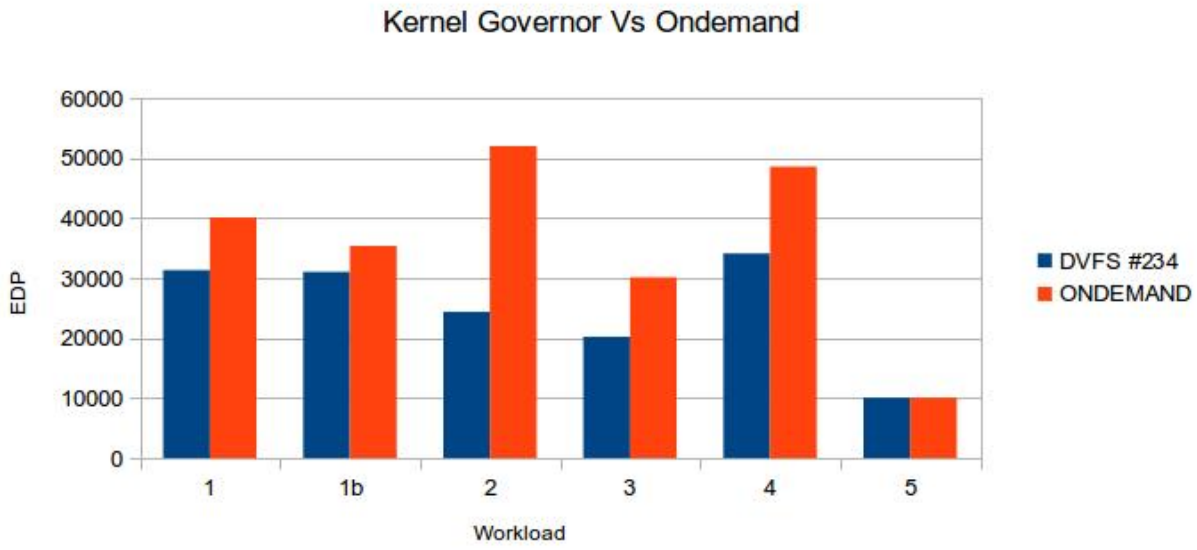
Hardware / Software Integration
Because of the nature of the project (i.e…. working with a development platform) there was not a great deal to be done with the integration of hardware and software. One key aspect though was the qualitative analysis of whether our governor policy was too aggressive and degregated the performance of the phone to below an acceptable level for user interaction. For this, we used the phone with a variety of different loads while the governor policy was running. During these workloads (games, web browsing) we verified that the

frames-per-second rate and the overall responsiveness did not get to a point where it was noticeable to the user.

Experiments and Results

Experimentation focused on the quantitative analysis of the energy delay product for a given workload with a given weight set. The weights for each counter was set according to our best observation from the design-space exploration presented earlier. Specifically, the weights were set to 50, 0, 30 and 20 for the four counters respectively.

For our experimental evaluation, we are comparing our kernel-level governor to the "ondemand" governor. We chose to compare against the ondemand governor, since it's one of the most widely used governors. We compare the two governors using the EDP metric. We couldn't use a performance (execution time) metric, since the interactive workloads will have the same duration under any governor. The workloads used are the interactive workloads provided for our second small project. The following charts demonstrate the differences between the two governors.



In almost all the cases, our kernel-level governor outperforms ondemand in terms of EDP, leading to significant power savings when used over time. This clearly shows that there is an advantage in having low-level information when making decisions regarding power consumption. Interestingly enough, our governor seems to perform identically to the ondemand governor in the case of workload 5. Both governors demonstrate identical behavior in that case.

Conclusions

We found that a DVFS policy implemented at the kernel level that can have access to the architectural counters can be an effective method to preserve power for a smartphone running interactive applications.

Even though it looks beneficial, elaborate kernel-level governors are not used in the majority of smartphones. We believe the reason for this is the complexity it would incur to create and tweak such a governor for the variety of Android smartphones in the world.

In our case, we had a target phone, which allowed us to run a full space exploration and figure out the desired weights of our four predetermined counters. In a more realistic scenario, for each Android device, more counters would have to be evaluated, against all possible weight combinations and against a wider range of workloads, resulting is an extremely high number of possible combinations.

References

[1] Stijn Eyerman, Lieven Eeckhout, 2010. *A Counter Architecture for Online DVFS Profitability Estimation*

[2] Leo Singleton, Christian Poellabauer, Karsten Schwan . *Monitoring of Cache Miss Rates for Accurate Dynamic Voltage and Frequency Scaling*

[3] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, Ricardo Bianchini. *CoScale: Coordinating CPU and Memory System DVFS in Server Systems*

[4] Vasileios Spiliopoulos, Stefanos Kaxiras, Georgios Keramidas. *A Framework for Continuously Adaptive DVFS*