Advanced Computer Architecture: Instruction-Level-Parallelism in Rivest-Shamir-Adleman Cryptography

Rushil Patel

November 7, 2022

Introduction

This report aims to find the hardware configuration for the SimpleScalar v2 simulator that minimises the total energy consumption of the processor, whilst providing some insights on why a specific hardware change may cause a change in energy consumption. The energy consumption is measured by the Wattch, a framework for architecture-level energy analysis.

Method

Running the simulator with the default configuration provides the output illustrated by Figure 1. The method this report will follow aims to reduce the energy consumed by individual components and stages of the processor by changing the hardware configurations based on hypotheses. //

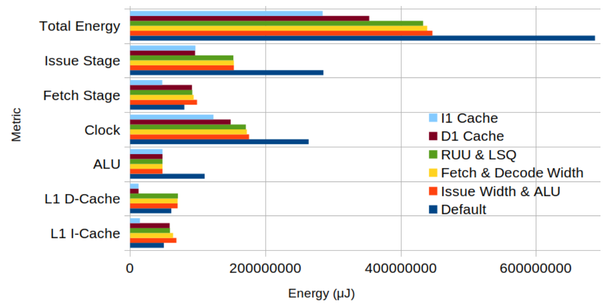

Figure 1: Energy consumption metrics recorded by changing components of the architecture.

Code Analysis

The RSA program makes use of for-loops, used for operations like checking whether a number is prime. GCC makes use of loop unrolling, increasing the size of the program by explicitly translating each branch, allowing for pipelining. Where unable to unroll, a branch instruction will be used. RSA doesn't use any floating point arithmetic, so floating point functional units are not needed in our architecture.

Issue Width & No. of ALUs

The highest contributor to power consumption is the issue stage energy, which is defined as a sum of the ALU, Data Cache, Result Bus, Window (register) and LSQ energies, according to the source code for Wattch. The highest of the individual components of the issue stage energy is the ALU energy, a combination of the power used in both integer and floating point ALUs. As RSA does not use floating point arithmetic, the correct approach should be to set the number of floating point ALUs to 0.

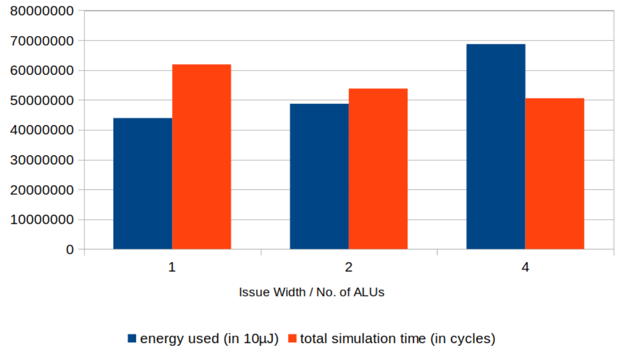

Figure 2: Energy consumption and simulation time (in cycles) with varied issue width and no. of ALUs.

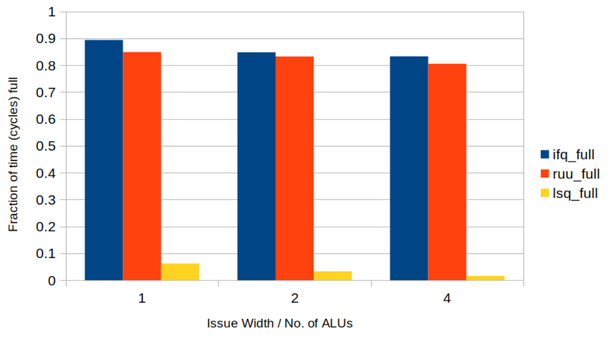

Figure 3: IFQ, RUU and LSQ occupancy rates with varied issue width and no. of ALUs.

The integer ALUs are relatively large hardware components in the CPU, and therefore require more hardware to integrate (wiring) and more energy to run. The number of ALUs is bottlenecked by the issue width of the processor – the maximum number of instructions issued in each cycle – so it makes sense to set our number of integer ALUs and issue widths to the same number. If our number of ALUs was lower than the issue width, we could be stalling as the ALU is busy, using energy as our execution time increases. If the number of ALUs were higher, we'd have ALUs unnecessarily using energy and increasing the size of the processor (and subsequently the wiring). Figure 2 shows us that as the simulation time (in cycles) increases, the energy consumption decreases. It's also worth observing that the rate at which the instruction fetch queue (IFQ), load-store queue (LSQ) and register update unit (RUU) are full is maximised when the ALU and issue width are set to 1, according to Figure 3. The maximisation of the LSQ in particular allows for the use of load-store forwarding as much as possible. As a result, D1 cache accesses are reduced, and functional units in the processor (i.e. the ALU) can use the LSQ instead to save energy.

We can see from Figure 1 that this decreases Issue Stage and ALU energy consumption drastically, whilst slightly increasing cache energy consumption.

Fetch and Decode Width

Figure 3 shows us the processor minimises energy best when the decode width and instruction fetch size are set to the same number, which is the default. They are likely the same value as everything fetched needs to ideally be decoded in the same clock cycle. 4 seems to be the optimal value, as in the event of a misprediction, the over-fetched instructions would result in redundant instruction cache (I1) accesses if the fetch size was high. If the fetch size was too low, this would still result in higher I1 cache accesses, as the processor would have to fetch instructions from I1 cache more frequently, rather than using the fetch queue to its full extent. As a result, a configuration that best utilises the benefits of over-fetching and misprediction risk-mitigation is required. Similarly to varying the issue width and no. of ALUs, this configuration maximises simulation time and minimises energy consumption.

RUU and LSQ Sizes

To further decrease energy consumption in the fetch, decode and issue stages, we can look at the RUU and

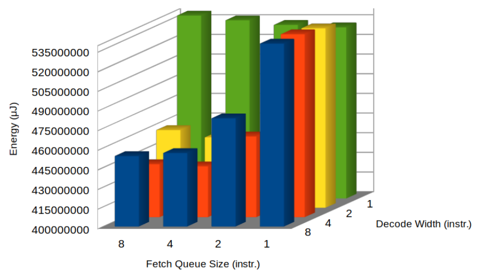

Figure 4: Energy consumption recorded by changing fetch width and decode width.

LSQ sizes. Figure 5 shows us the ideal configuration for the RUU and LSQ size is 16 and 4 respectively. The RUU performs best when set to a factor of 4 higher than the LSQ, according to the results. This could indicate that load and store instructions occur once every 4 instructions, where 16 is the ideal RUU size due to the fetch and decode instruction width also being 4. This gives the issue unit 4 clock cycles to execute instructions, which is perfect for an issuer with an issue width of size 1. The RUU and LSQ caches are therefore smaller and save energy.

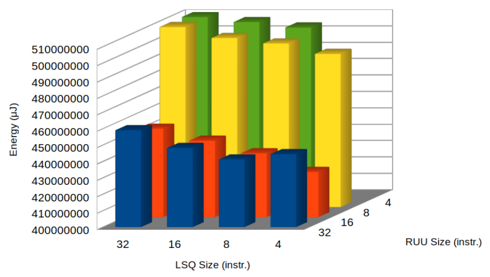

Figure 5: Energy consumption recorded by changing RUU and LSQ size.

D1 Cache

Our changes in the previous steps caused an increase in D1 and I1 cache energy consumption, most likely to do with the fact the default configuration for caches is too big.

Figure 6 shows us that for the D1 cache, energy consumption is linear to the number of cache sets and block size. Therefore, the smallest D1 cache possible is best for our energy efficient architecture.

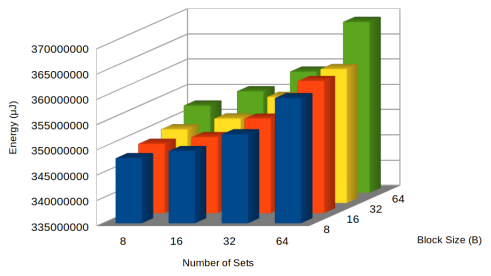

Figure 6: Energy consumption recorded by changing D1 cache sets and block sizes.

It seems obvious that miss rates for the D1 cache increase as size decreases, but it's interesting to observe that miss rates decrease for the unified D2/I2 cache (UL2), shown by Figure 7. This is likely because there are more UL2 accesses, so the miss rate decreases. It shows us that in terms of energy consumption, accessing the UL2 cache is less expensive in terms of energy than having a larger D1 cache.

In addition to reducing the energy consumption for the D1 cache, optimising led to reduction in energy consumption for the issue stage, as D1 cache is an important component of issuing instructions. Clock energy is also optimised, possibly to do with the fact that the clock hasn't got as much cache to keep synchronised.

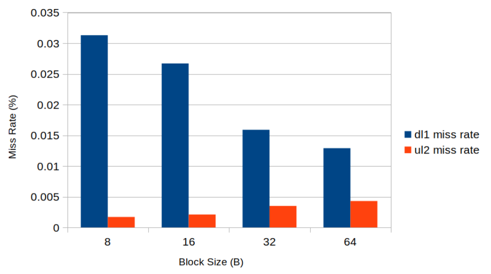

Figure 7: Cache miss rates for D1 and UL2 caches for caches with 8 sets.

Branch Prediction

The SimpleScalar simulator allows us to pick one of three branch prediction algorithms: bimodal, two-level and combinatorial. Two-level branch prediction typically consumes more energy than bimodal, due to the additional hardware complexities a global branch prediction table and branch history registers. Combinatorial is a combination of the two. Therefore, throughout our experiment, a bimodal branch predictor is used.

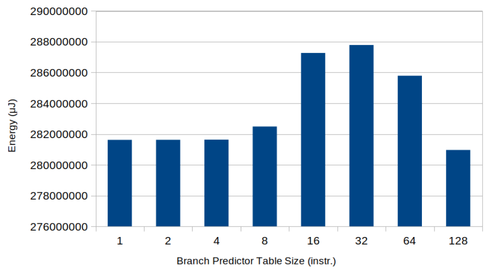

Figure 8: Energy consumed with different branch predictor table sizes for the bimodal branch prediction algorithm.

For the bimodal branch predictor, varying the branch predictor table size gives us the results displayed by Figure 8. 128 instructions is optimal for our configuration, which becomes clear when looking at Figure 9. Intuition tells us that a low branch predictor table size will be more energy efficient, as cache is energy-expensive. However, Figure 9 shows us that the branch direction prediction rate is optimal at 128, saving us more energy through predicting branch direction correctly. We mentioned earlier that the RSA code contained lots of loops, and that loop unrolling was used where possible. Loop unrolling increases the distance between adjacent branch instructions, and from our results, we can see that this number may be close to 128 instructions.

I1 Cache

The clock and fetch stages are now the bottleneck for energy consumption, according to Figure 1. We noted earlier that the RSA program stores negligi-

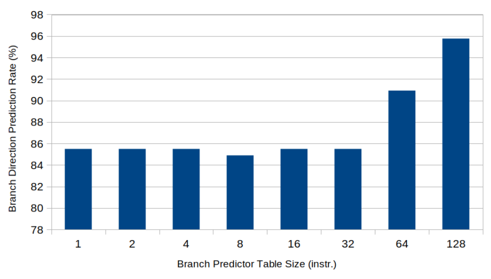

Figure 9: Branch direction prediction rates for the bimodal branch prediction algorithm.

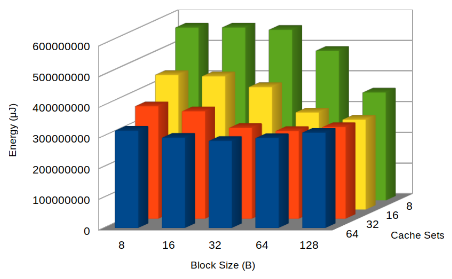

Figure 10: Energy consumption for I1 cache, based on number of cache sets and block size.

ble data to memory, but it does need instructions (which are stored sequentially in memory). Therefore, it would need a larger I1 cache, in comparison to the D1 cache optimised in the previous step. Figure 10 shows us the ideal configuration for minimising energy consumption would be 32 cache sets with 64B blocks.

It's important to consider simulation time in this case, unlike when considering issue width and no. of ALUs, as the extra time wasted would when fetching from the UL2 cache. It's worth observing from Figure 10 that the ideal configurations all use an instruction cache of around 1024B. Figure 11 shows us that the highest simulation time, for a block size 64B, gives us the ideal configuration specified by Figure 10.

Conclusion

Optimisations in this experiment reduced energy consumption further than a factor of 2 (Figure 1). For

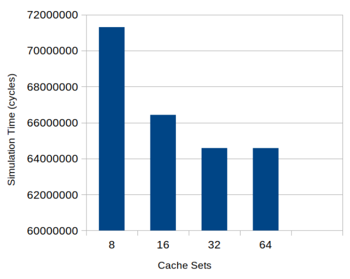

Figure 11: Simulation time (in cycles) for an I1 cache with 64B blocks and a variable number of cache sets.

future investigation, it may be worth looking into the UL2 and branch target buffer (BTB). The BTB is likely to be quite energy-expensive, being extremely fast, high access memory, so reducing it to fit our purpose without harming our branch direction prediction rates would be ideal. Similarly, the UL2 could be modified similarly to DL1.

The RSA program provided was best suited, in terms of energy, to a low data cache architecture, as it didn't require many stores to memory. Reducing energy from the default configuration meant removing unnecessary hardware components like multiple ALUs, and changing the issue width size to match this. Changing the architecture also allowed us to learn more about the program based on energy outputs from benchmark tests, such as GCC's loop unrolling could have meant branch instructions were around 128 instructions apart on average. We also saw that an increase in execution time would sometimes increase energy consumption, like with issue width and no. of ALUs, and sometimes decrease energy consumption, as seen with configuring I1 cache. Following a methodical approach allowed for finding the ideal architectural configuration for RSA in terms of energy consumption.