Static Resource Analysis Evaluation for the Cortex M0

This technical report evaluates the Static Resource Analysis introduced in [1] on the Cortex M0 processor

University of Bristol

Jamie Maddocks, Kyriakos Georgiou, Kerstin Eder

15/12/2016

This technical report evaluates the performance of the energy model created for the ARM Cortex-M0 when used with the static analysis introduced in [1]. This is done on the set of validation benchmarks shown in Table 1.

1 Comparison between Static Analysis and the Energy Model

Figure 1 shows the energy consumption predicted by the model against the energy consumption predicted by the static analysis for each benchmark in the validation set. Evaluating the static analysis estimation directly against hardware energy will take the inaccuracies of the energy model into account. Therefore, to evaluate the methods and implementation of the static analysis it is compared directly against the energy model's predictions, treating the energy model predictions as a ground truth.

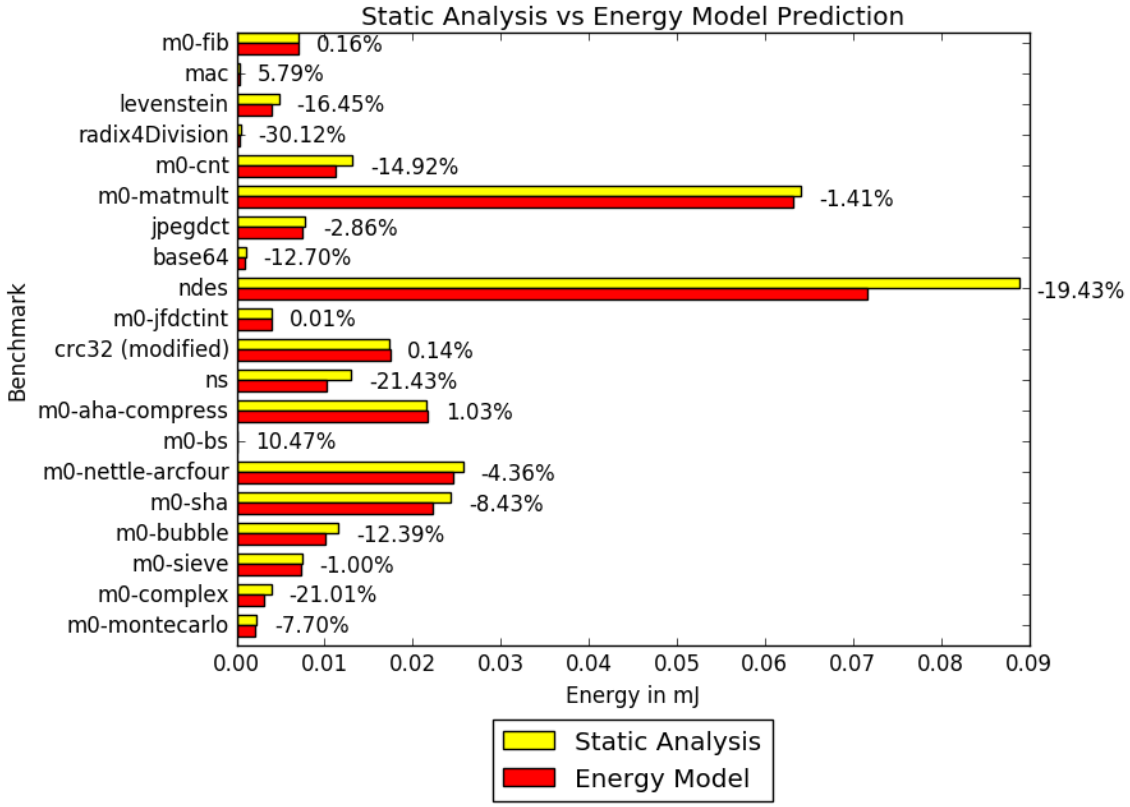


Figure 1: Static Analysis compared to the Energy Model

Figure 2 shows the absolute error for each benchmark, the average being 9.59%. A negative error indicates that the static analysis process has overestimated the energy consumption compared to the value predicted by the energy model, while a positive value indicates that it was underestimated.

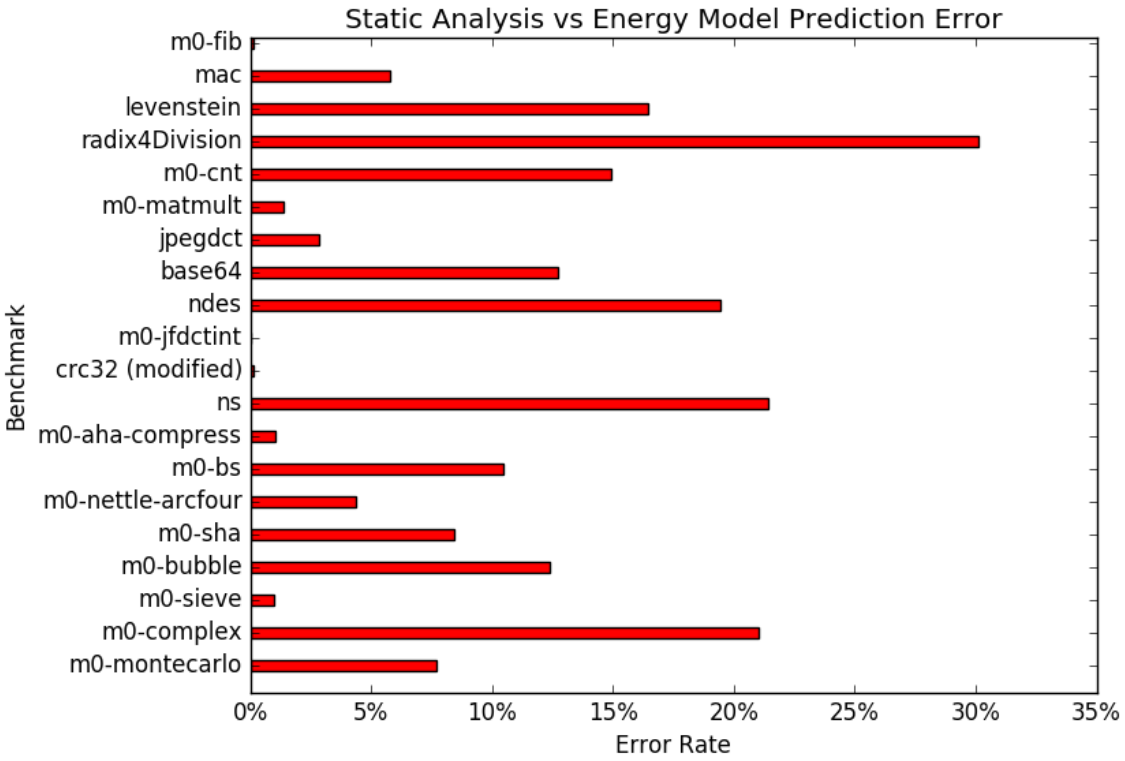
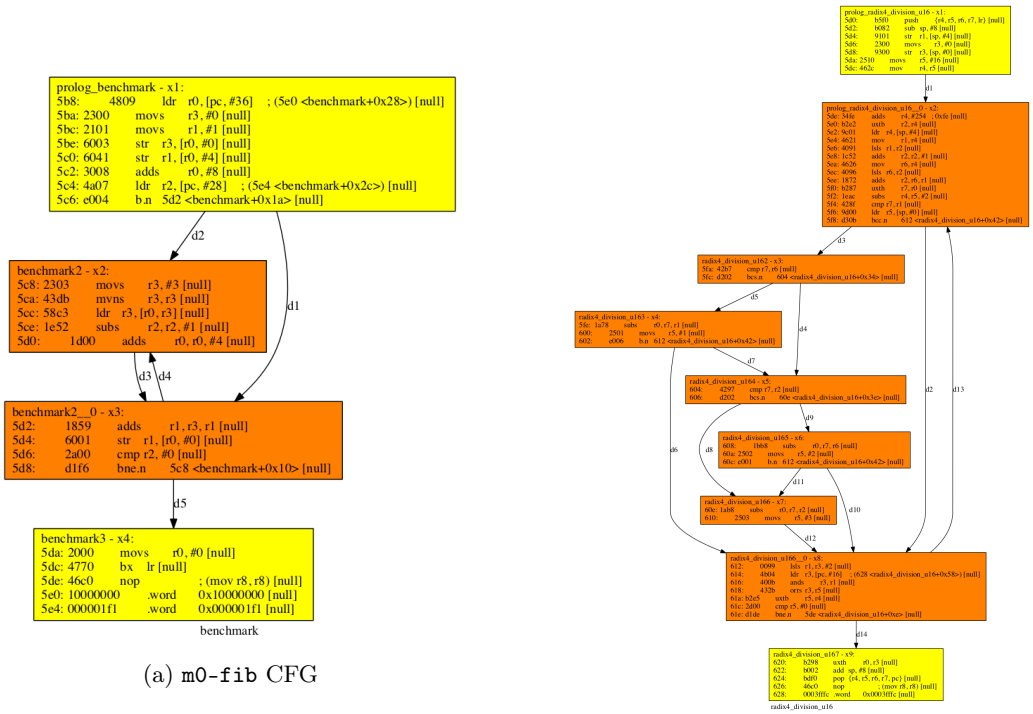


Figure 2: Static Analysis error relative to the Energy Model



(a) `m0-fib` CFG

(b) `radix4Division` CFG

Figure 3: CFG Comparison

Due to the more complex nature of static analysis a larger error is to be expected than when comparing the energy model against hardware measurements.

The worst performing benchmark was `radix4Divison` which had an error of -30.12% compared to the value predicted by the energy model. The best performing benchmark was `m0-fib` with a 0.16% difference in energy consumption values.

In order to evaluate the potential causes of the inaccuracy seen in the `radix4Division` benchmark, the CFGs of `radix4Division` and `m0-fib` are shown in Figure 3.

The `radix4Division` CFG shown in Figure 3b is considerably more complex than the one for the `m0-fib` benchmark shown in Figure 3a. Both benchmarks contain only a single loop, however `radix4Division` contains multiple possible paths within the loop due to the presence of an `if, else if` statement with four cases. In order to calculate the WCEC, IPET uses the worst case path; for this CFG that includes three sets of comparisons before executing the final `else` case in each iteration. However, in the actual running of the program this path is infeasible and the three sets of comparisons will not occur for every iteration, thus reducing the energy consumed. In order to improve the accuracy of the static analysis result the user could be prompted to input infeasible paths so that the static analysis can disregard them.

The `ns` benchmark also receives a high overestimation of 21.43%. The benchmark searches through a 4D array for some target value. Given an array `a[25][25][25][25]`, in the worst case 390,625 iterations of the innermost loop are executed. The innermost loop contains a branching instruction dependent on whether or not the target value has been found. During the worst case analysis performed by the IPET method the branching will always be followed when in reality, the instructions enclosed by the `if` block will be executed only once during execution. These extra instructions are believed to be the cause of the large discrepancy in energy consumption values. As with the `radix4Division` benchmark this could be avoided through the use of functional constraints provided by the user.

70% of the benchmarks evaluated overestimate the energy consumption when compared to the energy model. This level of overestimation is expected due to worst case paths being used by the static analysis. The values produced by the energy modules use worst-case instruction traces, therefore only valid paths can be followed. However, using the static analysis, various infeasible paths can also be executed and therefore result in overestimation of energy consumption as seen previously in the `radix4Division` benchmark.

2 Comparison between Hardware Measurement and Static Analysis

This section compares static analysis results against hardware measurements taken from the Cortex-M0 for each benchmark. These results indicate the level of overall accuracy an end-user would experience through the use of the system.

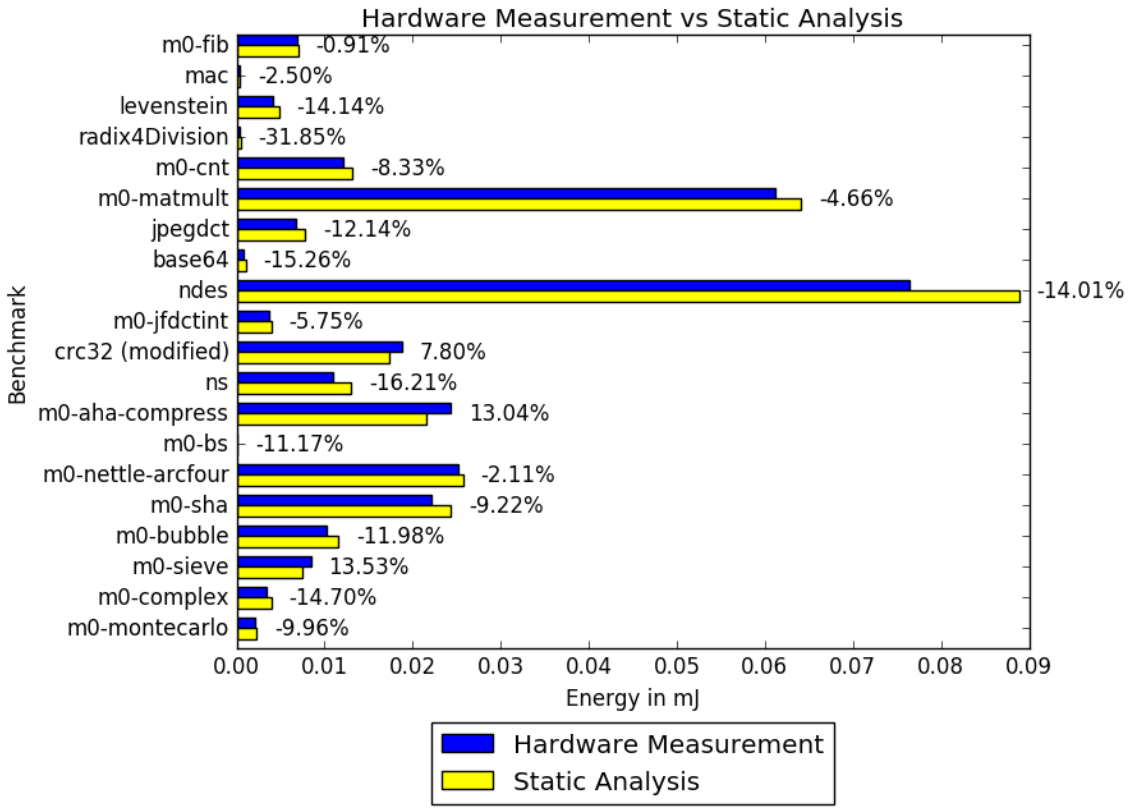


Figure 4: Comparison between Hardware Measurement and Static Analysis

Figure 4 shows the energy consumption values measured from hardware against the values predicted by the static analysis estimation. Negative errors indicate that the static analysis has overestimated energy consumption, a positive value indicates that it has been underestimated. Figure 5 shows the absolute error present for each benchmark. The average error over all validation benchmarks is 10.96%. The `radix4Division` has an error of 31.8%, removing this benchmark from the set results in an average of 9.87%.

Static analysis incorporates inaccuracies present in the energy model as well as inaccuracies present within the IPET method. Therefore the average accuracy is expected to be larger than comparison of static analysis directly against the energy model results. The static analysis accuracy

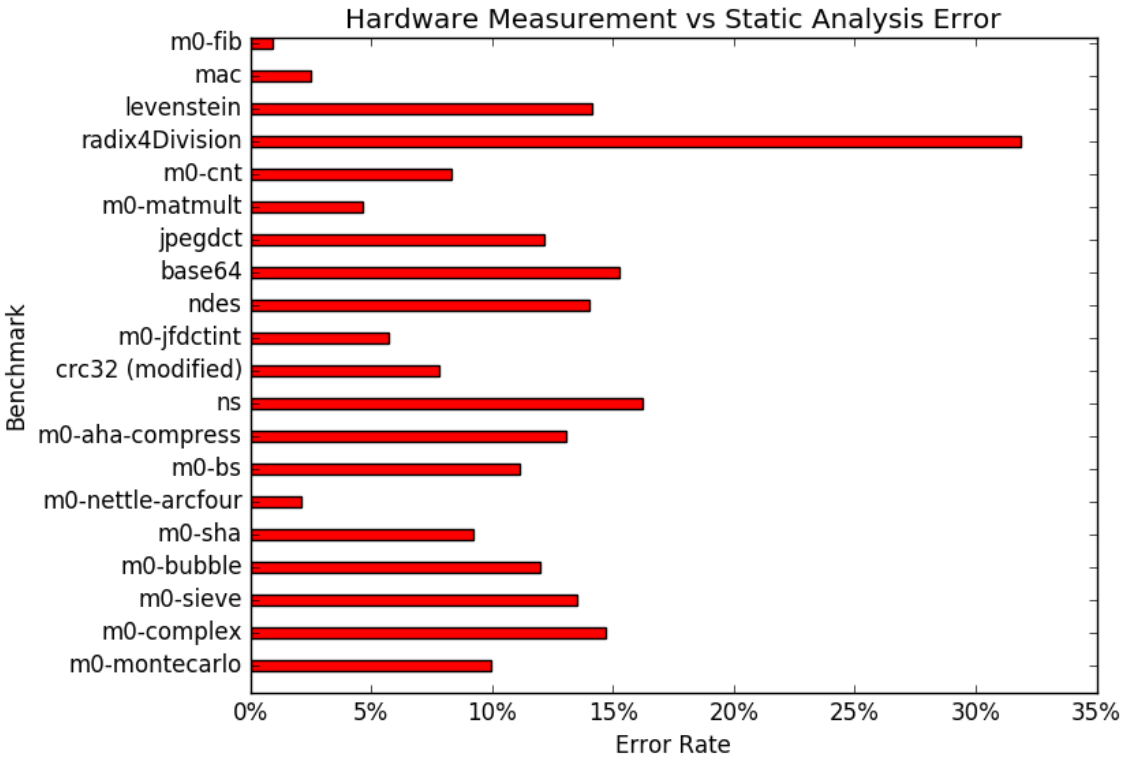


Figure 5: Static Analysis error relative to Hardware Measurement

is ultimately dependent on the accuracy of the underlying energy model. Static analysis may show a lower error against hardware measurements for some benchmarks compared to the equivalent energy model predictions. Take for example the `mac` benchmark: its absolute error is lower for static analysis than it is for the energy model prediction. This actually shows inaccuracies present within the IPET method adopted by the static analysis. If the static analysis estimation were to be 100% accurate it would predict the same energy consumption values as the energy model.

3 Conclusion

In conclusion, embedded devices continue to become integral parts of everyday life for many people. The emerging IoT market will only increase their influence and abundance. With negligible increases in battery capacity over the past decade, increasing power demand, and device usage, the energy efficiency of modern microprocessors and therefore the software they run will become of larger importance for developers and consumers alike.

Evaluations of both the energy model used with the static analysis from [1] is presented; this was achieved by gathering a set of pre-existing benchmarks and modifying them in order to run on the Cortex-M0 with the energy measurement harness. A variety of benchmarks were also created from scratch in order to exercise parts of the Cortex-M0 architecture. These benchmarks were combined to create a validation set. Measurements were then taken using the hardware setup in order to compare against the energy consumption values predicted by the energy model and static analysis energy consumption estimation.

The static analysis and energy model had an average error of 9.59%. Finally, comparing the static analysis directly against hardware measurements resulted in an average error of 10.96%. Removing the single outlier reduced this error to 9.87%.

The static analysis used in this report provides a good basis for providing energy consumption bounds the Cortex-M0 microprocessor. The work shown provides embedded developers with greater insight into the energy consumption of the software they create, making energy consumption a more accessible metric during the development process.

References

[1] K. Georgiou, S. Kerrison, Z. Chamski, and K. Eder. Energy Transparency for Deeply Embedded Programs. *ArXiv e-prints*, Aug 2016. URL `https://arxiv.org/abs/1609.02193`. Accepted to appear in ACM Transactions on Architecture and Code Optimization (TACO).

A Benchmarks Table

Benchmark	Source	Description	Attributes					
			L	N	A	B	CP	MF
m0-fib	Original	Computes the first 500 fibonacci numbers	✓		✓			
mac	MDH WCET	Dot product of two vectors and sum of squares	✓		✓			
levenstein	BEEBS	Edit distance between two strings	✓	✓	✓	✓		✓
radix4Divison	Online	Software implemented division	✓			✓	✓	
m0-cnt	MDH WCET	Count non-negative numbers in a matrix	✓	✓	✓			
m0-matmult	BEEBS	Multiply two matrices together	✓	✓	✓			
jpegdct	MDH WCET	Perform JPEG Discrete Cosine Transform	✓	✓	✓	✓		
base64	Online	Compute base64 encoding a string	✓		✓	✓		
ndes	MDH WCET	Symmetric key encryption	✓		✓	✓	✓	✓
m0-jfdctint	BEEBS	Slow integer DCT	✓		✓			
crc32 (modified)	BEEBS	Compute 32-bit CRC	✓		✓	✓		
ns	MDH WCET	Search within multi-dimensional array	✓	✓	✓			
m0-aha-compress	BEEBS	Data compression on random data	✓	✓	✓			
m0-bs	MDH WCET	Binary search within array	✓		✓			
m0-nettle-arcfour	BEEBS	ARCFOUR cryptographic cipher	✓		✓	✓		✓
m0-sha	Online	SHA256 cryptographic hash function	✓		✓	✓	✓	✓
m0-bubble	Original	Bubble sort	✓	✓	✓			
m0-sieve	Original	Sieve of Eratosthenes prime number generation	✓	✓	✓			
m0-complex	Original	Complex number and operations implementation	✓		✓			✓
m0-montecarlo	Original	Pi estimation using Monte Carlo method	✓		✓			

Table 1: Validation Benchmark Set

L: Contains loops

N: Nested loops

A: Uses arrays/matrices

B: Bitwise operations

CP: Complex CFG structure

MF: Multiple functions