Evolving Power-Efficient RNGs

David R. White
University of Glasgow

39th CREST Open Workshop
Tuesday 24th February 2015

Table of Contents

Table of Contents

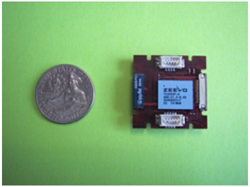
My First Paper

This work was my first paper, at GECCO 2008.

Followed by work on non-functional properties with GP and a collaboration with Andrea Arcuri.

Some of the earliest work on using GP to optimise non-functional properties.

Resource Efficient Software: Target Platforms



WSN Motes

RFID Tags

- Hardware limited (e.g. 64KB, 12MHz Mote)
- Multiple, demanding, requirements
- Interdependence between requirements
- What happens when you just don't have enough resources?

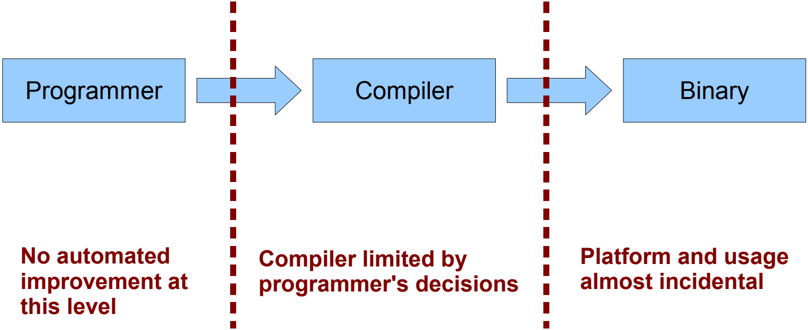
The Difficult of Manual Power Optimisation

AKA "the loneliness of the long-distance embedded systems programmer."

> *"The internals of a modern processor are complete chaos".*
>
> Anonymous RTS Professor

Standard Programming

Standard Programming: Limitations



Standard Programming: Limitations

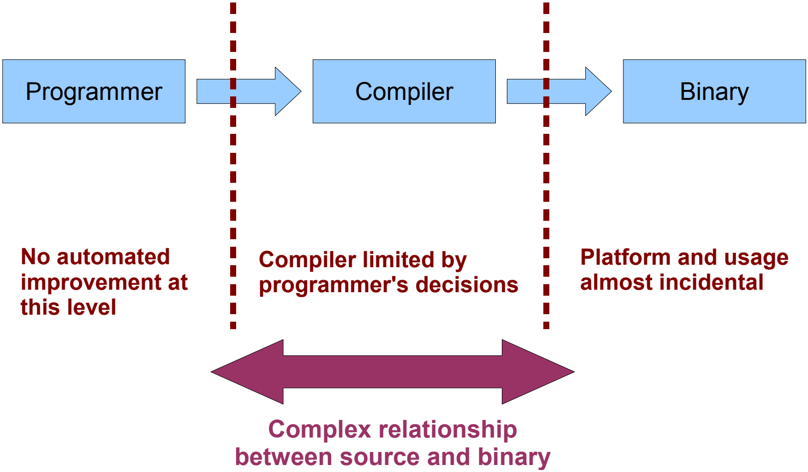


Table of Contents

Why RNGs?

A common component; small and frequently executed.

We were interested in trade-offs, not solely optimisation.

AnyScale Apps

Proof of Concept: Resource-Efficient PRNGs

Lamenca-Martinez et al. [2006] evolved PRNGs with a view to efficiency, building on work by Hernandez et al. [2004].

Tried to ensure efficiency through choice of function set.

See also Koza [1991], Jannink [1994], Sipper and Tomassini [1996].

Table of Contents

Measuring Fitness

Quality was measured using the Strict Avalanche Criterion as per Lamenca-Martinez et al. [2006].

Power efficiency was measured using the Sim-Wattch Simulator [Brooks et al., 2000].

Wattch

Cycle-level power simulator based on SimpleScalar simulator.

Estimates power usage through approximation for each logic unit:

$$P_d = CV_{dd}^2 af \tag{1}$$

C is the load capacitance; V_{dd} the supply voltage, f the clock frequency and a an activity that estimates the amount of transistor switching.

Partly estimated, validation given by Brooks et al. [2000].

Problems

Simulation is (allegedly) inaccurate and expensive.

For our purposes it only has to be *relatively* accurate, *most of the time*.

We reduced the sample size compared to previous work, to increase speed.

Could we have just used execution time as a proxy?

Measuring Randomness: Strict Avalanche Criterion

Trying to create a function $r(a_0 \ldots a_7)$

Maximise nonlinearity of this function:

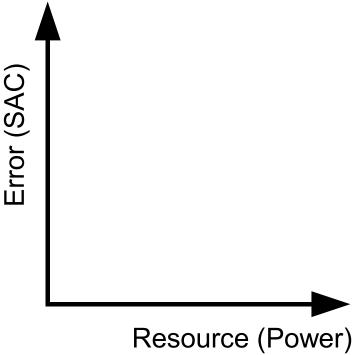
1. Create a random input $a_0 \ldots a_7$
2. Evaluate $o_1 = r(a_0 \ldots a_7)$
3. Flip one bit of input to create $b_0 \ldots b_7$
4. Evaluate $o_2 = r(b_0 \ldots b_7)$
5. Record hamming distance $H(o_1, o_2)$

Measuring Randomness: Strict Avalanche Criterion II

Compare the distribution of bit flips to the binomial distribution:

$$\sum_{i=0}^{n} \frac{(C_i - E_i)^2}{E_i}$$

Our Objective Space



A Single Program

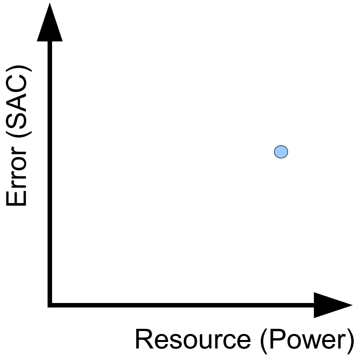


Table of Contents

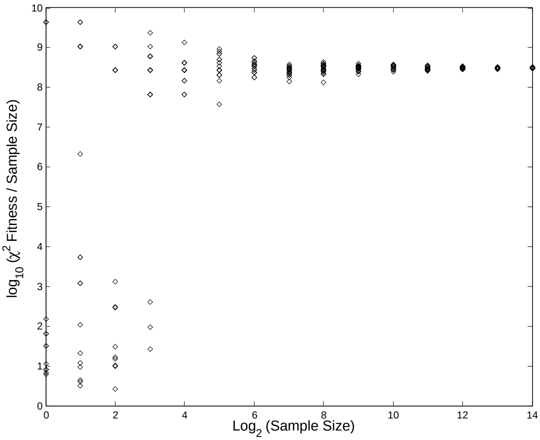
Experimental Method

Replicated work by Hernandez et al. [2004] and achieved similar results.

Used reduced sample size.

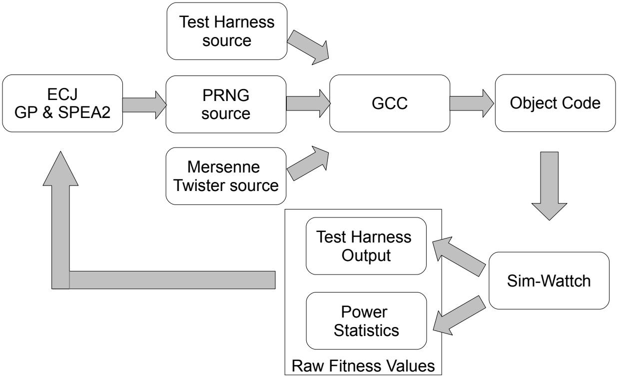
Searched using ECJ and SPEA2.

Variance of Fitness Measures



Repeated for other programs and power consumption measure.

Experimental Framework



Parameter Settings

Function set	MULT, AND, SUM, NOT, OR, XOR, Logical Shift Left (LSL), LSR, Circular Shift Left (CSL), CSR

Parameter settings left to their defaults in ECJ, as specified by koza.params.

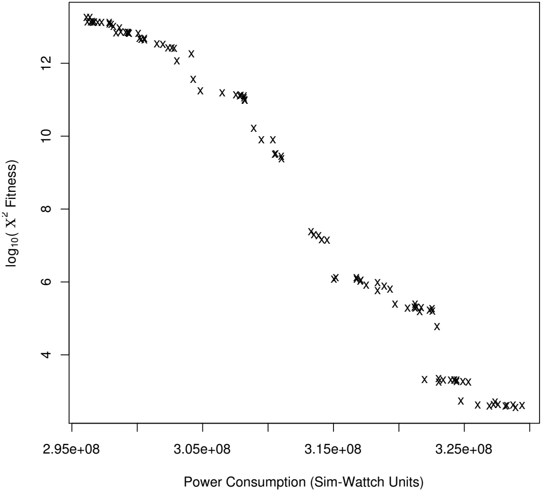
All non-default settings taken from previous work, except for MOO-specific settings.

Table of Contents

Evolution of the Pareto Front

Animation

Example Pareto Front



A Fantastically Low Power RNG

int genRand (int a0, int a1, int a2, int a3, int a4, int a5, int a6, int a7) {
    return $\neg(\neg(1997453768ul))$;
}

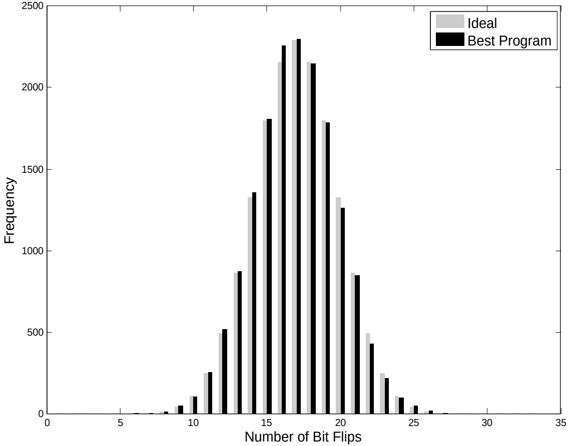
A Middle Ranking Low Power RNG

int genRand (int a0, int a1, int a2, int a3, int a4, int a5, int a6, int a7) {
return $a2 \vee \neg(((a2 + a0) * ((a4 \oplus ((a6 + a5) \oplus a7)) + (a1 \oplus a3))) \wedge \neg(\neg(1997453768ul)))$;
}

An Excellent Low Power RNG

int genRand (int a0, int a1, int a2, int a3, int a4, int a5, int a6, int a7) {
$(2307363674ul \oplus (a2 * a6)) + ((\neg((((((a2 * a6) \oplus (a7 \oplus a1)) * (a0 \oplus a3)) \oplus (a2 * a6)) + (a5 * a4)) >> 2307363674ul) \oplus (a0 \oplus a3)) + \neg((a5 * a4) * ((2307363674ul \oplus (a7 \oplus a1)) + (a0 \oplus a3))));$
}

Bit Flip Distribution for Best Individual



The Impact of the Mult Function

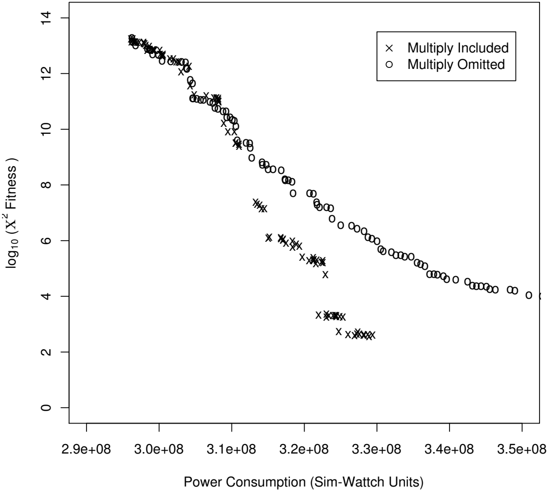


Table of Contents

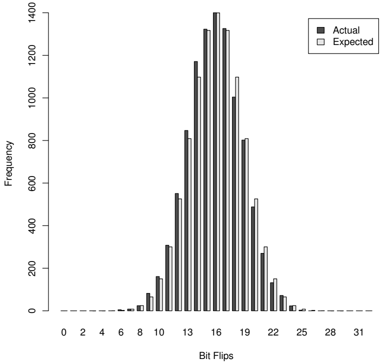
Conclusions

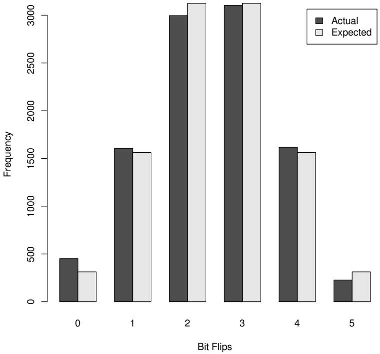
This is about more than straightforward optimisation.

Multiple solutions vs a single, tunable, scalable algorithm.

More radically, consider manipulation rather than optimisation.

Postscript: Time Avalanche Criterion





See White et al. [2010]

Acknowledgements

My coauthors John Clark, Jeremy Jacob and Simon Poulding.

Andrea Arcuri for his collaboration.

Juan E. Tapiador and Julio Cesar Hernandez-Castro, for inspiration and creative discussion.

david.r.white@glasgow.ac.uk


Genetic Improvement Workshop
(at GECCO 2015)
Submission deadline: April 3, 2015
geneticimprovement2015.com

References I

David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, pages 83–94, 2000. URL http://www.cs.ucr.edu/~{}harry/classes_files/CS269_02/papers/Brooks_ISCA_00.pdf.

Julio Cesar Hernandez, Pedro Isasi, and Andre Seznec. On the design of state-of-the-art pseudorandom number generators by means of genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1510–1516, 2004. ISBN 0-7803-8515-2.

Jan Jannink. Cracking and Co-Evolving Randomizers. *Advances in Genetic Programming*, pages 425–443, 1994. URL citeseer.ist.psu.edu/jannink94cracking.html.

John R. Koza. Evolving a computer program to generate random numbers using the genetic programming paradigm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 37–44. Morgan Kaufmann, 1991. URL http://citeseer.ist.psu.edu/john91evolving.html.

References II

Carlos Lamenca-Martinez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda. Lamar: A new pseudorandom number generator evolved by means of genetic programming. In *Parallel Problem Solving from Nature IX*, volume 4193, pages 850–859. Springer-Verlag, 2006. URL `http://ppsn2006.raunvis.hi.is/proceedings/202.pdf`.

Moshe Sipper and Marco Tomassini. Co-evolving parallel random number generators. In *Parallel Problem Solving from Nature – PPSN IV*, pages 950–959. Springer, 1996.

DavidR. White, Juan M.E. Tapiador, Julio Cesar Hernandez-Castro, and JohnA. Clark. Fine-grained timing using genetic programming. In *EuroGP 2010*. 2010. doi: 10.1007/978-3-642-12148-7_28.