Software Analysis and Testing

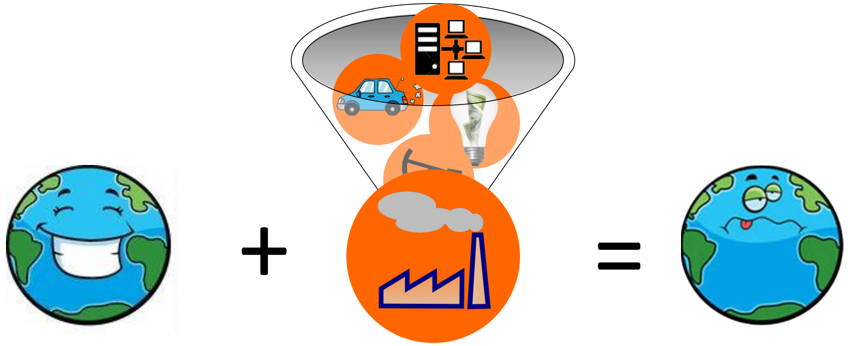
Green Software

João Saraiva

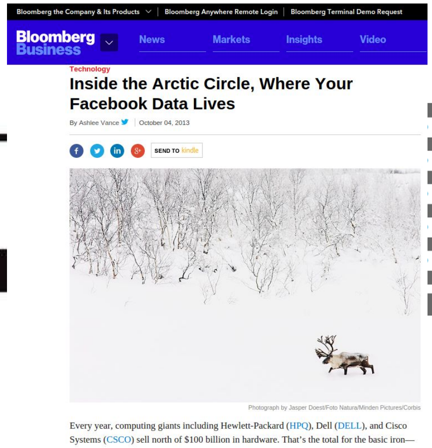
2020/2021

Departamento de Informática
Universidade do Minho, Portugal

Going Green



Global energy system is unsustainable


Bloomberg the Company & Its Products | Bloomberg Anywhere Remote Login | Bloomberg Terminal Demo Request

Bloomberg Business — News — Markets — Insights — Video

Technology

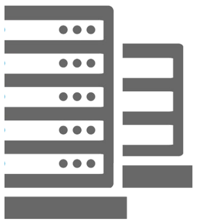
Inside the Arctic Circle, Where Your Facebook Data Lives

By Ashlee Vance | October 04, 2013

SEND TO kindle

Photograph by Jasper Doest/Foto Natura/Minden Pictures/Corbis

Every year, computing giants including Hewlett-Packard (HPQ), Dell (DELL), and Cisco Systems (CSCO) sell north of $100 billion in hardware. That's the total for the basic iron—



Green Computing

- Caught the attention of many companies allowing them to save:

"close to 50% of the energy costs of an organization can be attributed to the IT departments"
- [PICMET, 2009]

"up to 90% of energy used by ICT hardware can be attributed to software"
- [The Greenhouse Gas Protocol Report, 2013]

Green Software

- Reducing energy consumption through software analysis and optimization

- Problem:
  - How to analyze
  - How to interpret
  - How to improve

Green Software

- Problems (extended to programmers):
  - How to analyze
  - How to interpret
  - How to improve

Mining questions about software energy consumption - [MSR'14]

Integrated energy-directed test suite optimization - [ISTA'14]

Seeds: A software engineer's energy-optimization decision support framework - [ICSE'14]

Energy vs. Power

- *Power (w)* - rate (or effort) at which that work is done

- *Energy* (J) - amount of work done

- *Power* can change constantly while *Energy* is the accumulation

Energy = Power x Seconds

360,000 J = 100W x 3,600s

Which languages are the most energy efficient?

Motivation

- Understanding the energy efficiency of programming languages

What paradigms are most energy efficient? Is a faster language always a more energy efficient one?

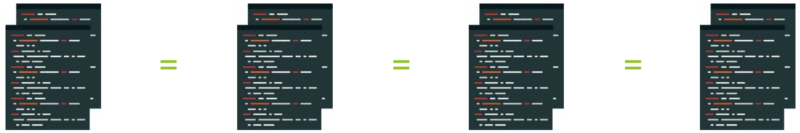
Are there languages which run slower while consuming less energy?

How much energy consumption is attributed to CPU? How much energy does memory usage consume?

Which are the most energy efficient languages?

Comparing Programming Languages

C Java Haskell Fortran



= = =

The Computer Language Benchmarks Game

* Formerly known as *The Great Computer Language Shootout*

The Computer Language Benchmarks Game (CLBG)

- Project to compare programming languages
  - 28 different programming languages
  - 13 different benchmarks
- Experts compete to code the fastest solutic
- Same underlying algorithms
- Publicly available:
  - Source Code
  - Compiler Versions
  - Optimization Flags
- Comparable and easily replicable programs/solutions!

Ada C Erlang Haskell Ja OCaml Pa Racket Ruby S

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse complement	Read DNA sequences, write their reverse-complement	fasta output
binary trees	Allocate, traverse and deallocate many binary trees	21
chameneos redux	Symmetrical thread rendezvous requests	6M
meteor contest	Search for solutions to shape packing puzzle	2,098
thread ring	Switch from thread to thread passing one token	50M

Design

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse complement	Read DNA sequences, write their reverse-complement	fasta output
binary trees	Allocate, traverse and deallocate many binary trees	21

Ada C Chapel C# C++ Dart

Erlang F# Fortran Go Hack

Haskell Java JavaScript Lisp Lua

OCaml Pascal Perl PHP Python

Racket Ruby JRuby Rust

Swift TypeScript

Paradigm	Languages
Functional	Erlang, F#, Haskell, Lisp, Ocaml, Perl, Racket, Ruby, Rust;
Imperative	Ada, C, C++, F#, Fortran, Go, Ocaml, Pascal, Rust;
Object-Oriented	Ada, C++, C#, Chapel, Dart , F#, Java, JavaScript, Ocaml, Perl, PHP, Python, Racket, Rust, Smalltalk, Swift, TypeScript;
Scripting	Dart, Hack, JavaScript, JRuby, Lua, Perl, PHP, Python, Ruby, TypeScript;

Measurements

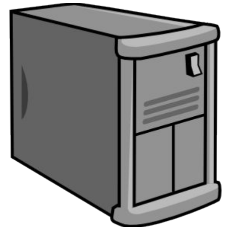
Energy

- Running Average Power Limit (RAPL)
- Designed by intel for i5/i7 architectures (SandyBridge, IvyBridge, Haswell, etc)
- Monitors energy consumption info for Machine-Specific Registers (MSRs)
- Allows **very precise** and **fine-grain** measurements through function calls
  - **DRAM/GPU**
  - **CPU**
  - **Package**

Peak Memory

- Unix's time tool

Execution



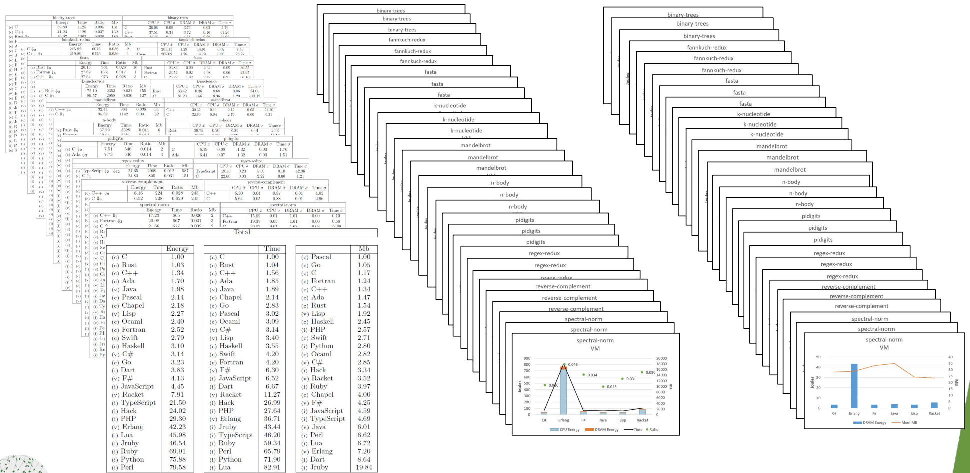
- Linux Ubuntu 16.0 4.8.0-22-generic
- 16GB Ram
- Intel(R) Core(TM) i5-4460 CPU @ 3.40GHz

- Compiler versions: https://sites.google.com/view/energy-efficiency-languages/setup
- Source Code: http://benchmarksgame.alioth.debian.org/

- Compile -> Run -> Measure

- 27 Languages x 10 Benchmarks = 270 Results

Results



Energy vs. Time

Energy vs. Memory

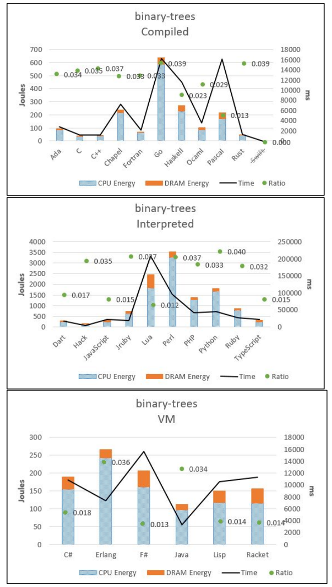
Energy vs. Time vs. Memory

Energy vs. Time

binary-trees				
	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1$ $\Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1$ $\Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\downarrow_3$ $\Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\downarrow_4$ $\Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\uparrow_2$ $\Downarrow_9$	156.71	4497	0.035	502
(v) C# $\downarrow_1$ $\Downarrow_1$	189.74	10797	0.018	427
(v) F# $\downarrow_3$ $\Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\downarrow_3$ $\Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\uparrow_5$ $\Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\uparrow_5$ $\Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\uparrow_2$ $\Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\downarrow_1$ $\Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\downarrow_2$ $\Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\downarrow_2$ $\Downarrow_2$	315.10	21686	0.015	915
(c) Go $\uparrow_3$ $\Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\uparrow_2$ $\Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482
(i) PHP $\Uparrow_3$	1,397.51	42316	0.033	786
(i) Python $\Uparrow_{15}$	1,793.46	45003	0.040	275
(i) Lua $\downarrow_1$	2,452.04	209217	0.012	1961
(i) Perl $\uparrow_1$	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

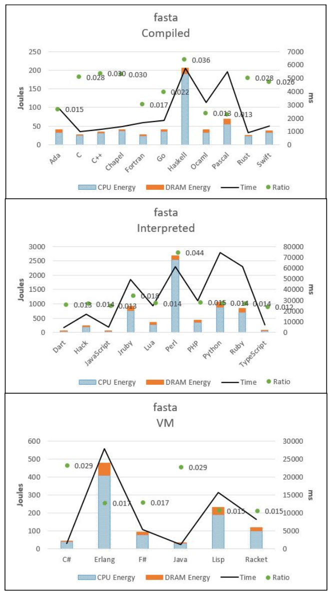
Energy vs. Time

binary-trees				
	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1$ $\Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1$ $\Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\downarrow_3$ $\Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\downarrow_4$ $\Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\uparrow_2$ $\Downarrow_9$	156.71	4497	0.035	502
(v) C# $\downarrow_1$ $\Downarrow_1$	189.74	10797	0.018	427
(v) F# $\downarrow_3$ $\Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\downarrow_3$ $\Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\uparrow_5$ $\Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\uparrow_5$ $\Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\uparrow_2$ $\Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\downarrow_1$ $\Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\downarrow_2$ $\Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\downarrow_2$ $\Downarrow_2$	315.10	21686	0.015	915
(c) Go $\uparrow_3$ $\Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\uparrow_2$ $\Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482
(i) PHP $\Uparrow_3$	1,397.51	42316	0.033	786
(i) Python $\Uparrow_{15}$	1,793.46	45003	0.040	275
(i) Lua $\downarrow_1$	2,452.04	209217	0.012	1961
(i) Perl $\uparrow_1$	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

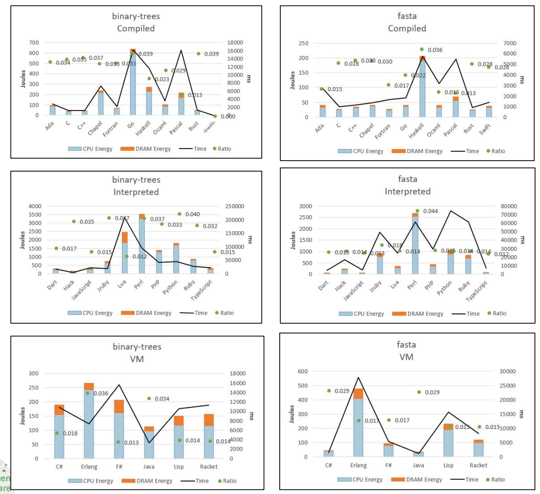


Energy vs. Time

fasta				
	Energy	Time	Ratio	Mb
(c) Rust $\Downarrow_9$	26.15	931	0.028	16
(c) Fortran $\downarrow_6$	27.62	1661	0.017	1
(c) C $\uparrow_1$ $\Downarrow_1$	27.64	973	0.028	3
(c) C++ $\uparrow_1$ $\Downarrow_2$	34.88	1164	0.030	4
(v) Java $\uparrow_1$ $\Downarrow_{12}$	35.86	1249	0.029	41
(c) Swift $\Downarrow_9$	37.06	1405	0.026	31
(c) Go $\downarrow_2$	40.45	1838	0.022	4
(c) Ada $\downarrow_2$ $\Uparrow_3$	40.45	2765	0.015	3
(c) Ocaml $\downarrow_2$ $\Downarrow_{15}$	40.78	3171	0.013	201
(c) Chapel $\uparrow_5$ $\Downarrow_{10}$	40.88	1379	0.030	53
(v) C# $\uparrow_4$ $\Downarrow_5$	45.35	1549	0.029	35
(i) Dart $\Downarrow_6$	63.61	4787	0.013	49
(i) JavaScript $\Downarrow_1$	64.84	5098	0.013	30
(c) Pascal $\downarrow_1$ $\Uparrow_{13}$	68.63	5478	0.013	0
(i) TypeScript $\downarrow_2$ $\Downarrow_{10}$	82.72	6909	0.012	271
(v) F# $\uparrow_2$ $\Uparrow_3$	93.11	5360	0.017	27
(v) Racket $\downarrow_1$ $\Uparrow_5$	120.90	8255	0.015	21
(c) Haskell $\uparrow_2$ $\Downarrow_8$	205.52	5728	0.036	446
(v) Lisp $\Downarrow_2$	231.49	15763	0.015	75
(i) Hack $\Downarrow_3$	237.70	17203	0.014	120
(i) Lua $\Uparrow_{18}$	347.37	24617	0.014	3
(i) PHP $\downarrow_1$ $\Uparrow_{13}$	430.73	29508	0.015	14
(v) Erlang $\uparrow_1$ $\Uparrow_{12}$	477.81	27852	0.017	18
(i) Ruby $\downarrow_1$ $\Uparrow_2$	852.30	61216	0.014	104
(i) JRuby $\uparrow_1$ $\Downarrow_2$	912.93	49509	0.018	705
(i) Python $\downarrow_1$ $\Uparrow_{18}$	1,061.41	74111	0.014	9
(i) Perl $\uparrow_1$ $\Uparrow_8$	2,684.33	61463	0.044	53



Energy vs. Time



CPU-based Energy			
	Avg	Min	Max
Compiled	88.94%	85.27%	91.75%
Interpreted	87.98%	81.57%	92.90%
VM	88.94%	86.10%	92.43%

Energy vs. Time

Total

	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++		C	1.17
(c) Ada	1.70	(c) Ada		Fortran	1.24
(v) Java	1.98		1.89	(c) C++	1.34
(c) Pascal	2.14		2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54		59.34		6.72
(i) Ruby	69.91		65.79		7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

57J

4604J

2019ms

167416ms

Average	Joules	ms
Compiled	120	5103
VM	576	20623
Interpreted	2365	87614

Average	Joules	ms
Imperative	125	5585
OO	879	32975
Functional	1367	42740
“Scripting”	2320	88322

regex-redux	Energy	Time	Ratio	Mb
(i) TypeScript $\downarrow_2$ $\Downarrow_{12}$	24.65	2009	0.012	587
(c) C $\uparrow_1$	24.83	805	0.031	151
(i) JavaScript $\downarrow_2$ $\Downarrow_9$	25.68	2096	0.012	525
(i) PHP $\uparrow_2$ $\Downarrow_1$	34.57	1667	0.021	182
(c) Pascal $\downarrow_1$ $\Uparrow_4$	35.20	2282	0.015	106
(i) Hack $\uparrow_2$ $\Downarrow_2$	38.96	2052	0.019	268
(c) Rust	40.26	2287	0.018	218
(c) Chapel $\Downarrow_{11}$	97.19	4534	0.021	1055
(c) Ada $\Uparrow_5$	148.66	5157	0.029	157
(i) Python $\downarrow_1$	161.62	7116	0.023	429
(c) Ocaml $\downarrow_3$ $\Downarrow_6$	172.43	12978	0.013	948
(c) C++ $\downarrow_1$ $\Uparrow_6$	176.24	10656	0.017	216
(i) Ruby $\downarrow_4$ $\Uparrow_4$	192.88	14282	0.014	305
(v) Java $\uparrow_4$ $\Downarrow_6$	194.65	5694	0.034	1225
(i) Dart $\downarrow_1$ $\Uparrow_4$	197.92	13485	0.015	459
(i) Perl $\uparrow_4$ $\Uparrow_{13}$	236.24	7164	0.033	154
(i) Jruby $\uparrow_2$ $\Downarrow_4$	348.44	13477	0.026	1369
(v) Racket $\downarrow_1$	358.20	26152	0.014	983
(v) C# $\uparrow_1$ $\Uparrow_3$	522.59	14723	0.035	851
(c) Swift $\Uparrow_5$	538.11	41703	0.013	677
(v) F# $\Uparrow_7$	650.51	46905	0.014	667
(c) Haskell		n.a.		
(c) Fortran		n.a.		
(c) Go		n.e.		
(i) Lua		n.e.		
(v) Erlang		n.a.		
(v) Lisp		n.e.		

Energy vs. Memory

Total

	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56		1.17
(c) Ada	1.70	(c) Ada	1.85		1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34		6.72
(i) Ruby	69.91	(i) Perl	65.79		7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

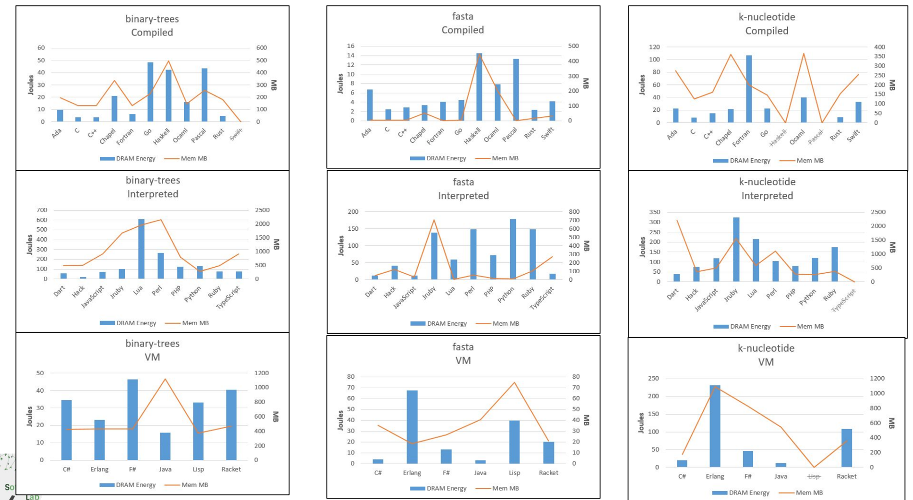
66Mb

1309Mb

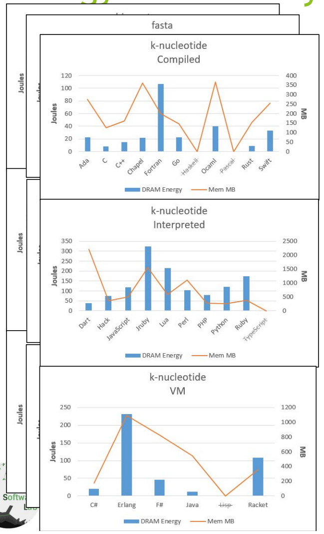
	Average
	Mb
Compiled	125
VM	285
Interpreted	426

	Average
	Mb
Imperative	116
OO	249
Functional	251
“Scripting”	421

Energy vs. Memory



Energy vs. Memory



- ***Spearman*** rank-order correlation coefficient
- ***Spearman*** ***p*** **= 0.2091**

+1 = perfect positive

0 = no relationship

-1 = perfect negative

0.3 = weak uphill

p = 0.2091

Energy vs. Time vs. Memory

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Energy vs. Time vs. Memory (Pareto Optimization)

Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

Wrap Up

- General ranking of programming languages
- Unexpected results
- Time and Energy are not always proportional
- C is still the King (Rust is close behind)

Wrap Up

- General ranking of programming languages
- Unexpected results
- Time and Energy are not always proportional
- C is still the King (Rust is close behind)

Future Work

- Add new languages
- Update certain compilers
- Measure continuous RAM usage
- Suggestions?

Howard Chu @hyc_symas

If you're coding in any language other than C you're murdering polar bears. And drowning Florida.

> **Frank Denis** @jedisct1
> Energy Efficiency across Programming Languages
> greenlab.di.uminho.pt/wp-content/upl...

Rob Evans @internetplumber

Replying to @hyc_symas @ISCdotORG

OTOH, if you're coding in C you're responsible for all the buffer overflows and resulting exploits. Polar bears and Florida, or security?

Michał Łomnicki @mlomnicki

so if you like birds and trees and all the stuff then better avoid Python :) http://greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf ...

Paul Woegerer @woepaul

sites.google.com/view/energy-ef ... is unfair towards python. For fair comparison they should also account for energy consumption needed for coding

Can we save energy by refactoring Java **programs** to use different **data structure implementations**?

Research Questions

- **(RQ1)** Can we define an energy consumption quantification of Java data structures and their methods?
- **(RQ2)** Can we use such quantification to decrease the energy consumption of software systems?

Towards a Ranking of Java data structures

Design

- Simple scenario of:
  - Storing
  - Retrieving
  - Deleting
- ***String*** values

Towards a Ranking of Java data structures

Design – Data Structures

- *Java Collections Framework* (JCF) library

Sets	Lists	Maps
ConcurrentSkipListSet	*ArrayList*	*ConcurrentHashMap*
CopyOnWriteArraySet	*AttributeList*	*ConcurrentSkipListMap*
HashSet	*CopyOnWriteArrayList*	*HashMap*
LinkedHashSet	*LinkedList*	*Hashtable*
TreeSet	*RoleList*	*IdentityHashMap*
	RoleUnresolvedList	*LinkedHashMap*
	Stack	*Properties*
	Vector	*SimpleBindings*
		TreeMap
		UIDefaults
		WeakHashMap

Towards a Ranking of Java data structures

Design - Methods

Sets	Lists	Maps
add	*add*	*clear*
addAll	*addAll*	*containsKey*
clear	*add(index)*	*containsValue*
contains	*addAll(index)*	*entrySet*
containsAll	*clear*	*get*
iterateAll	*contains*	*iterateAll*
iterator	*containsAll*	*keySet*
remove	*get*	*put*
removeAll	*indexOf*	*putAll*
retainAll	*iterator*	*remove*
toArray	*lastIndexOf*	*values*
	listIterator	
	listIterator(index)	
	remove	
	removeAll	
	Continues…	
	remove(index)	
	retainAll	
	set	

Towards a Ranking of Java data structures

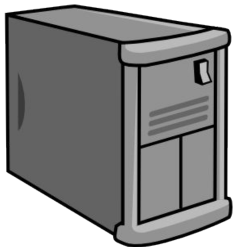
Design - Benchmark

Test description of Set methods

Method	Description of the test for the method
add	add popsize/10 elements. half existing, half new
addAll	addAll of secondaryCol 5 times
clear	clear 5 times
contains	contains popsize/10 elements. half existing, half new
containsAll	containsAll of secondaryCol 5 times
iterateAll	iterate and consult popsize values
iterator	iterator popsize times
remove	remove popsize/10 elements. half existing, half new
removeAll	removeAll of secondaryCol 5 times
retainAll	retainAll of secondaryCol 5 times
toArray	toArray 5 times

Towards a Ranking of Java data structures

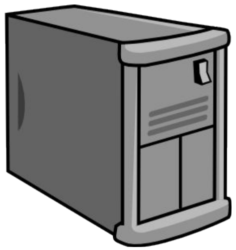
Execution - Specifications



- Linux 3.13.0-74-generic OS
- 8GB Ram
- Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz
- Java Interpreter/Compiler versions 1.8.0_66
- RAPL/jRAPL

Towards a Ranking of Java data structures

Execution – For every test



- Warm-up
  - Instantiated
  - Populated w/ **popsize**
  - Performed simple actions on the data structure
- Each test x20
  - Extracted time/Joules consumed
  - Removed lowest/highest 20%
- 336 different test (Collection.method) configurations
- 6720 executions for each **popsize**
- 20,000+ different executions

Towards a Ranking of Java data structures

Results (25k pop)

Methods	Concurrent SkipListSet J	ms	HashSet J	ms	Linked HashSet J	ms	TreeSet J	ms
add	1.6822	87	1.7749	87	1.4917	75	1.4817	92
addAll	1.4549	93	1.4771	89	1.9335	94	1.5101	93
clear	1.4901	78	1.0586	64	1.3288	60	1.8566	73
contains	1.4213	88	2.0685	78	1.0401	76	2.0446	79
containsAll	1.8317	96	1.4000	77	2.1748	88	1.4443	89
iterateAll	1.9225	99	1.4554	92	1.2907	83	1.3844	83
iterator	1.6096	83	1.7596	75	0.9613	76	1.7239	76
remove	1.7877	78	1.2633	75	1.2458	93	1.0700	76
removeAll	1.8072	85	2.1359	77	1.9145	100	1.3920	91
retainAll	3.2607	206	2.4092	200	2.2512	199	3.2222	193
toArray	1.4789	86	1.3833	80	1.3776	79	1.6292	80

Methods	ArrayList J	ms	AttributeList J	ms	CopyOn Write ArrayList J	ms	LinkedList J	ms
add	0.9773	71	1.1510	67	1.7839	117	1.8016	86
addAll	1.3353	76	1.0492	88	1.3586	82	1.1043	88
addAlli	1.7855	86	1.6035	68	1.1789	86	1.7272	99
addI	1.7125	93	1.3849	87	1.6558	119	1.6404	96
clear	1.1284	76	1.2409	75	1.7155	68	1.6497	74
contains	2.7568	166	2.4228	165	3.1768	167	3.1552	193
containsAll	1.5993	87	1.8053	92	2.1889	92	2.2887	118
get	2.0029	83	1.1171	78	1.4918	77	2.0168	109
indexOf	1.4447	76	2.0325	84	1.5682	70	2.6289	101
iterateAll	2.0701	79	1.0473	77	1.0103	73	2.6401	107
iterator	1.4893	84	1.1589	84	1.3922	72	1.7666	108
lastIndexOf	1.7750	99	1.7666	98	2.0383	94	2.5019	127
listIterator	1.4457	76	1.6190	84	1.3737	71	2.5003	106
listIteratori	1.7356	78	1.1552	81	1.5160	77	2.1996	105
remove	1.1308	96	1.4480	85	2.1946	162	1.6924	98
removeAll	8.0905	671	7.8108	697	7.3237	666	8.3150	752
removei	1.9135	85	1.3534	92	2.2858	118	1.7174	100
retainAll	2.7037	193	2.7845	200	2.6052	198	2.5982	205
set	0.9476	64	1.5943	70	1.9669	110	2.0474	112
sublist	1.3108	76	1.6021	80	1.4792	80	1.8457	98
toArray	1.6418	84	1.5024	84	2.0934	73	1.6739	106

Methods	RoleList J	ms	Role Unresolved List J	ms	Stack J	ms	Vector J	ms
add	1.4801	76	1.1865	74	1.5659	76	1.5177	69
addAll	1.6661	76	1.8672	88	1.1015	88	1.7903	73
addAlli	1.5980	81	1.2497	85	1.2962	72	1.6268	90
addI	1.2718	85	1.3124	86	1.5287	83	1.4554	86
clear	1.6705	76	1.4304	80	1.6199	73	1.0574	71
contains	2.1751	162	2.4688	164	2.0128	166	2.1558	168
containsAll	1.3244	100	1.3930	96	1.2054	89	1.5091	87
get	2.2110	81	1.6613	71	1.8956	86	1.4978	73
indexOf	1.5674	79	1.1944	81	1.8090	81	2.0788	75
iterateAll	1.3605	85	1.7822	71	1.6036	81	1.1336	87
iterator	1.9760	73	1.3300	79	2.1895	84	1.6505	83
lastIndexOf	1.8914	92	1.4211	95	1.2260	84	1.2296	96
listIterator	1.3380	80	1.5176	85	1.6354	69	1.2746	81
listIteratori	1.7588	79	1.0334	80	1.8799	85	1.7545	78
remove	1.4560	84	1.1368	85	1.2663	96	1.4973	82
removeAll	7.6148	692	7.9911	664	7.3824	654	7.1281	665
removei	1.6308	85	1.6369	89	1.5850	81	1.5486	90
retainAll	3.0973	197	2.4172	200	2.7635	242	3.4019	245
set	1.5249	76	1.2312	73	1.4938	75	1.4957	72
sublist	1.4910	85	1.5117	71	1.7082	75	0.9414	75
toArray	1.5418	79	1.7455	83	1.5694	69	2.0213	80

Methods	Concurrent HashMap J	ms	Concurrent SkipListMap J	ms	HashMap J	ms	Hashtable J	ms	Linked HashMap J	ms
clear	2.0276	94	2.2961	88	1.8395	104	1.5761	94	1.5025	97
containsKey	2.3132	105	2.1693	123	2.1343	103	1.8582	94	1.8726	103
containsValue	21.5611	2305	7.8032	643	8.3615	683	8.4957	765	6.1326	462
entrySet	2.2878	93	2.2363	116	1.8531	108	2.1332	107	1.8362	113
get	2.3106	103	1.9972	119	1.8120	102	1.4071	100	1.8252	116
iterateAll	2.1041	96	1.8353	115	2.6673	100	1.5343	91	1.6462	111
keySet	1.7287	95	2.4889	124	1.6813	114	2.2226	99	1.8328	103
put	1.8591	104	2.2888	102	2.4628	92	1.3123	96	2.0338	108
putAll	1.4147	95	2.2852	122	1.7564	100	1.5949	105	1.8608	113
remove	1.8574	92	2.2131	105	1.9256	109	1.6067	97	2.2300	106
values	1.8279	85	2.4690	116	2.5755	109	2.2266	94	2.0009	107

Methods	Properties J	ms	Simple Bindings J	ms	TreeMap J	ms	UIDefaults J	ms	Weak HashMap J	ms
clear	2.0777	98	2.1401	106	1.6706	98	1.8143	105	1.9941	95
containsKey	1.6018	107	1.8055	99	1.9452	100	2.3366	89	1.9675	108
containsValue	7.3755	692	7.9912	678	9.1771	847	7.9341	714	6.7072	562
entrySet	1.7800	97	2.1557	102	2.1617	115	1.7087	105	1.4666	102
get	1.7851	97	1.5359	100	2.2331	115	1.5252	89	1.7185	103
iterateAll	1.6362	100	2.0472	116	1.9122	111	1.6574	95	1.7139	106
keySet	1.4866	92	2.0630	106	2.1680	110	1.5547	99	1.8749	105
put	1.7038	107	2.1646	102	1.4355	91	2.1204	93	2.5784	105
putAll	1.3097	95	2.1461	112	1.8914	116	2.3094	87	2.0750	108
remove	1.9660	98	2.2178	106	1.8133	101	1.6888	92	2.4103	103
values	1.9120	111	2.0692	108	1.4467	105	1.6533	100	2.4628	111

Towards a Ranking of Java data structures

Methods	Concurrent SkipListSet J	ms	HashSet J	ms	Linked HashSet J	ms	TreeSet J	ms
add	14.0472	824	17.5243	1072	15.0643	876	14.1021	758
addAll	19.5092	1518	17.8589	1100	16.5155	983	13.5737	983
clear	11.5958	747	12.0199	764	12.2874	770	11.5565	758
contains	13.6576	870	16.6950	1014	15.6210	880	11.2337	682
containsAll	16.9809	1212	17.2110	1038	15.8865	886	12.3979	844
iterateAll	13.0184	785	18.1706	1091	15.4155	865	11.2088	684
iterator	13.2534	752	16.7433	1013	15.5284	850	11.0499	641
remove	12.7444	789	15.5699	949	13.6615	799	11.2653	675
removeAll	17.2849	1293	17.0514	998	14.5821	841	13.2071	937
retainAll	3621.9872	346898	3912.0129	384829	3584.3529	346337	4111.2397	408297
toArray	14.8120	875	17.8458	1070	14.3511	848	13.1271	750

Towards a Ranking of Java data structures

Methods	ArrayList J	ArrayList ms	AttributeList J	AttributeList ms	CopyOn Write ArrayList J	CopyOn Write ArrayList ms	LinkedList J	LinkedList ms	RoleList J	RoleList ms	Role Unresolved List J	Role Unresolved List ms	Stack J	Stack ms	Vector J	Vector ms
add	0.9773	71	1.1510	67	1.7839	117	1.8016	86	1.4801	76	1.1865	74	1.5659	76	1.5177	69
addAll	1.3353	76	1.0492	88	1.3586	82	1.1043	88	1.6661	76	1.8672	88	1.1015	88	1.7903	73
addAlli	1.7855	86	1.6035	68	1.1789	86	1.7272	99	1.5980	81	1.2497	85	1.2962	72	1.6268	90
addI	1.7125	93	1.3849	87	1.6558	119	1.6404	96	1.2718	85	1.3124	86	1.5287	83	1.4554	86
clear	1.1284	76	1.2409	75	1.7155	68	1.6497	74	1.6705	76	1.4304	80	1.6199	73	1.0574	71
contains	2.7568	166	2.4228	165	3.1768	167	3.1552	193	2.1751	162	2.4688	164	2.0128	166	2.1558	168
containsAll	1.5993	87	1.8053	92	2.1889	92	2.2887	118	1.3244	100	1.3930	96	1.2054	89	1.5091	87
get	2.0029	83	1.1171	78	1.4918	77	2.0168	109	2.2110	81	1.6613	71	1.8956	86	1.4978	73
indexOf	1.4447	76	2.0325	84	1.5682	70	2.6289	101	1.5674	79	1.1944	81	1.8090	81	2.0788	75
iterateAll	2.0701	79	1.0473	77	1.0103	73	2.6401	107	1.3605	85	1.7822	71	1.6036	81	1.1336	87
iterator	1.4893	84	1.1589	84	1.3922	72	1.7666	108	1.9760	73	1.3300	79	2.1895	84	1.6505	83
lastIndexOf	1.7750	99	1.7666	98	2.0383	94	2.5019	127	1.8914	92	1.4211	95	1.2260	84	1.2296	96
listIterator	1.4457	76	1.6190	84	1.3737	71	2.5003	106	1.3380	80	1.5176	85	1.6354	69	1.2746	81
listIteratori	1.7356	78	1.1552	81	1.5160	77	2.1996	105	1.7588	79	1.0334	80	1.8799	85	1.7545	78
remove	1.1308	96	1.4480	85	2.1946	162	1.6924	98	1.4560	84	1.1368	85	1.2663	96	1.4973	82
removeAll	8.0905	671	7.8108	697	7.3237	666	8.3150	752	7.6148	692	7.9911	664	7.3824	654	7.1281	665
removei	1.9135	85	1.3534	92	2.2858	118	1.7174	100	1.6308	85	1.6369	89	1.5850	81	1.5486	90
retainAll	2.7037	193	2.7845	200	2.6052	198	2.5982	205	3.0973	197	2.4172	200	2.7635	242	3.4019	245
set	0.9476	64	1.5943	70	1.9669	110	2.0474	112	1.5249	76	1.2312	73	1.4938	75	1.4957	72
sublist	1.3108	76	1.6021	80	1.4792	80	1.8457	98	1.4910	85	1.5117	71	1.7082	75	0.9414	75
toArray	1.6418	84	1.5024	84	2.0934	73	1.6739	106	1.5418	79	1.7455	83	1.5694	69	2.0213	80

Towards a Ranking of Java data structures

Methods	Concurrent HashMap J	Concurrent HashMap ms	Concurrent SkipListMap J	Concurrent SkipListMap ms	HashMap J	HashMap ms	Hashtable J	Hashtable ms	Linked HashMap J	Linked HashMap ms
clear	2.0276	94	2.2961	88	1.8395	104	1.5761	94	1.5025	97
containsKey	2.3132	105	2.1693	123	2.1343	103	1.8582	94	1.8726	103
containsValue	21.5611	2305	7.8032	643	8.3615	683	8.4957	765	6.1326	462
entrySet	2.2878	93	2.2363	116	1.8531	108	2.1332	107	1.8362	113
get	2.3106	103	1.9972	119	1.8120	102	1.4071	100	1.8252	116
iterateAll	2.1041	96	1.8353	115	2.6673	100	1.5343	91	1.6462	111
keySet	1.7287	95	2.4889	124	1.6813	114	2.2226	99	1.8328	103
put	1.8591	104	2.2888	102	2.4628	92	1.3123	96	2.0338	108
putAll	1.4147	95	2.2852	122	1.7564	100	1.5949	105	1.8608	113
remove	1.8574	92	2.2131	105	1.9256	109	1.6067	97	2.2300	106
values	1.8279	85	2.4690	116	2.5755	109	2.2266	94	2.0009	107

Methods	Properties J	Properties ms	Simple Bindings J	Simple Bindings ms	TreeMap J	TreeMap ms	UIDefaults J	UIDefaults ms	Weak HashMap J	Weak HashMap ms
clear	2.0777	98	2.1401	106	1.6706	98	1.8143	105	1.9941	95
containsKey	1.6018	107	1.8055	99	1.9452	100	2.3366	89	1.9675	108
containsValue	7.3755	692	7.9912	678	9.1771	847	7.9341	714	6.7072	562
entrySet	1.7800	97	2.1557	102	2.1617	115	1.7087	105	1.4666	102
get	1.7851	97	1.5359	100	2.2331	115	1.5252	89	1.7185	103
iterateAll	1.6362	100	2.0472	116	1.9122	111	1.6574	95	1.7139	106
keySet	1.4866	92	2.0630	106	2.1680	110	1.5547	99	1.8749	105
put	1.7038	107	2.1646	102	1.4355	91	2.1204	93	2.5784	105
putAll	1.3097	95	2.1461	112	1.8914	116	2.3094	87	2.0750	108
remove	1.9660	98	2.2178	106	1.8133	101	1.6888	92	2.4103	103
values	1.9120	111	2.0692	108	1.4467	105	1.6533	100	2.4628	111

Is faster, Greener?!

Slower → containsKey

Faster → entrySet

Methods	Properties J	Properties ms	Simple Bindings J	Simple Bindings ms
clear	2.0777	98	2.1401	106
containsKey	1.6018	107	1.8055	99
containsValue	7.3755	692	7.9912	678
entrySet	1.7800	97	2.1557	102
get	1.7851	97	1.5359	100
iterateAll	1.6362	100	2.0472	116
keySet	1.4866	92	2.0630	106
put	1.7038	107	2.1646	102
putAll	1.3097	95	2.1461	112
remove	1.9660	98	2.2178	106
values	1.9120	111	2.0692	108

Is slower, Greener?!

Research Questions

- ~~**(RQ1)** Can we define an energy consumption quantification of Java data structures and their methods?~~
- **(RQ2)** Can we use such quantification to decrease the energy consumption of software systems?

Optimizing Energy Consumption of Java Programs

Methodology

1. Compute which implementations/methods are used in the programs
2. Look up the appropriate energy tables for the used implementations/methods
3. Choose the most energy efficient alternative

Optimizing Energy Consumption of Java Programs

Applying the methodology – Data acquisition

- Obtained Java projects of a Journalism support platform
  - First year OO course
  - Collaborators, Journalists, Readers, Editors
  - Write chronicles and reports
  - Give likes and comments
  - Etc.
- Average of
  - 36 classes
  - 104 methods
  - 2000 lines of code

Optimizing Energy Consumption of Java Programs

Applying the methodology – Data acquisition

- Obtained 7 test cases to simulate usage
  - Size varied vetween 2000-10000 for each test case/each entity
- Chosen popsize – 25,000 (smallest)
- Applied methodology on 5 projects
  - Detected usage of any JCF implementation
  - Detected which methods were used for each implementation
  - Chose the most efficient implementation for each project
  - Measured the changes before and after

Optimizing Energy Consumption of Java Programs

Applying the methodology – Example

Projects	Data Structures Original	Data Structures Optimized	Methods
1	TreeMap	Hashtable	{containsKey, get, put, values}

	Concurrent	Concurrent		Linked	Simple		Weak	
Methods	ArrayList J	AttributeList J	CopyOn Write ArrayList J	LinkedList J	RoleList J	Role Unresolved List J	Stack J	Vector J
add	0.9773	1.1510	1.7839	1.8016	1.4801	1.1865	1.5659	1.5177
listIterator	1.4457	1.6190	1.3737	2.5003	1.3380	1.5176	1.6354	1.2746
Total	**2.4229**	**2.7700**	**3.1575**	**4.3018**	**2.8181**	**2.7041**	**3.2012**	**2.7923**

Optimizing Energy Consumption of Java Programs

Applying the methodology - Results

Results of pre and post optimization

Projects	Data Structures Original J	Data Structures Original ms	Data Structures Optimized J	Data Structures Optimized ms	Improvement J	Improvement ms
1	23.744583	1549	22.7071302	1523	4.37%	1.68%
2	24.6787895	1823	23.525123	1741	4.67%	4.50%
3	25.0243507	1720	22.259355	1508	11.05%	12.33%
4	17.1994425	1258	16.2014997	1217	5.80%	3.26%
5	19.314512	1372	18.3067573	1245	5.22%	9.26%

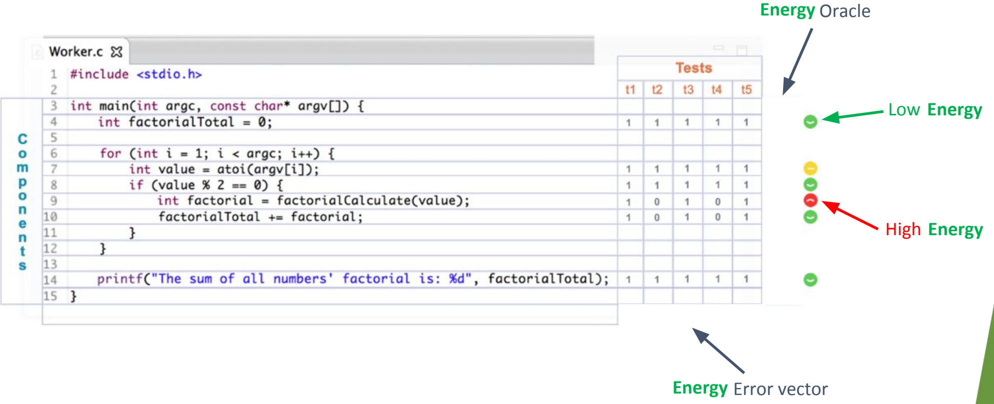
- Between 4.37% - 11.05%
- Average of 6.2%

FW + Conclusion

- Presented detailed study of the energy consumption of Sets, Lists, and Maps
  - Quantification of the energy spent by each method (**RQ1 Answer**)
- Introduced a very simple methodology to optimize Java programs **(RQ2 Answer)**
- Consider other object types (int, objects, etc.)
- Implement an automatic refactoring plugin
- Found @ our page: http://greenlab.di.uminho.pt/

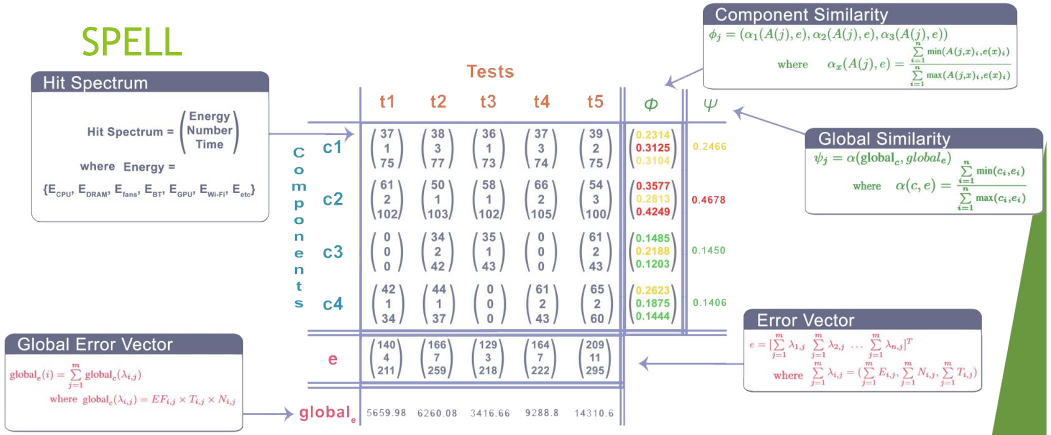
Can we detect energy hotspots in source code?

Spectrum-based Fault Localization (SFL)

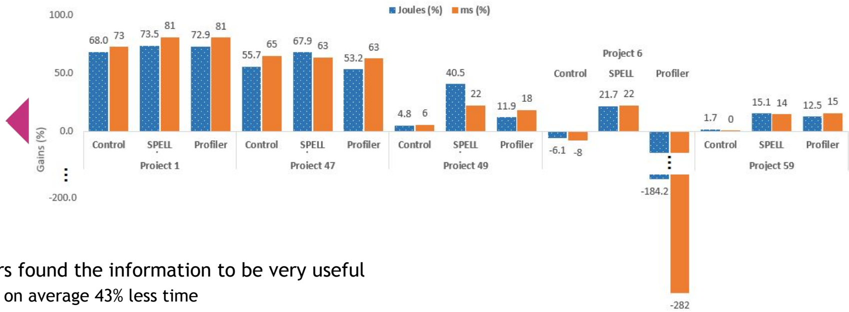


Problem: Energy cannot be defined as binary values

SPELL



SPELL – initial studies



- Developers found the information to be very useful
  - Spent on average 43% less time
  - Produced more energy efficient programs (26% less energy on average)
- SPELL is:
  - Language independent
  - Multi level analysis
  - Multi hardware component analysis
  - Points to probable hot spots in source code