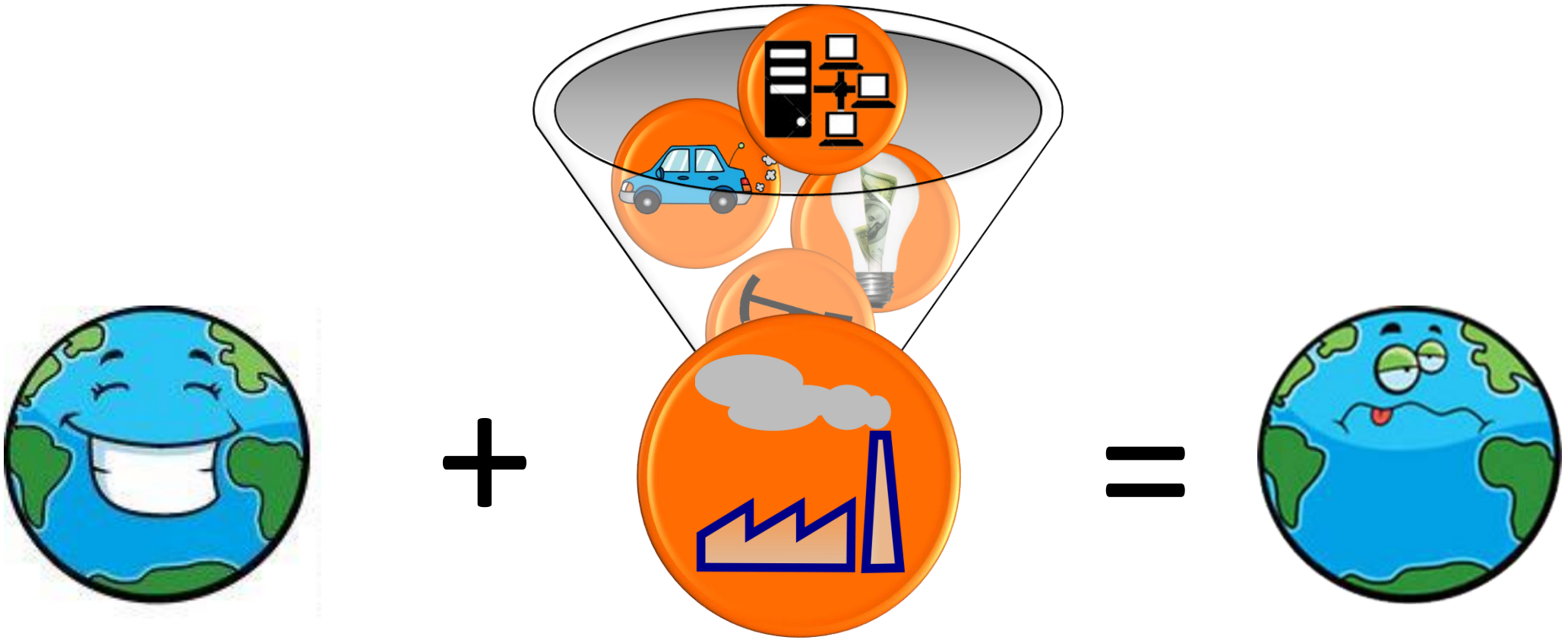
Green Computing as an Engineering Discipline

Rui Pereira

Talk @ InfoBlender

ruipereira@di.uminho.pt

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

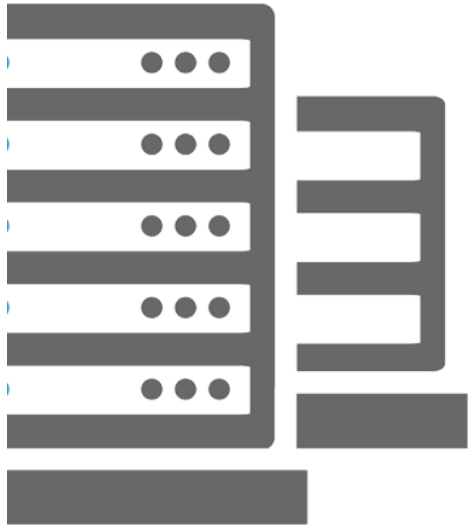
Inside the Arctic Circle, Where Your Facebook Data Lives

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 Lab

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]

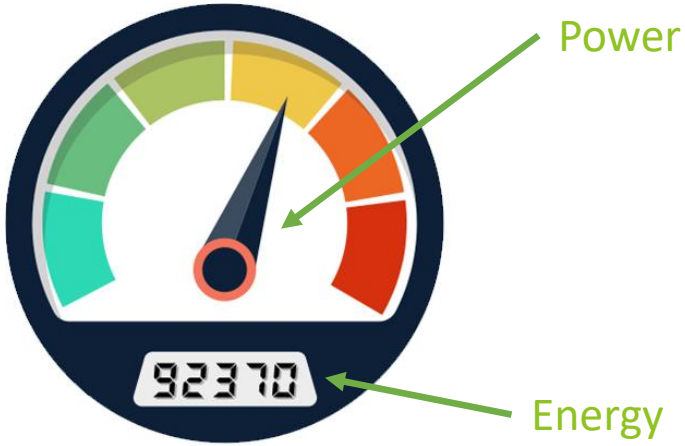
Integrated energy-directed test suite optimization - [ISTA'14]

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

Green Software Lab

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

Is ~~faster~~ greener?
slower

Energy-aware software: Challenges, opportunities and strategies - [Journal of Computational Science'13]

Understanding energy behaviors of thread management constructs - [OOPSLA'14]

An experimental survey of energy management across the stack - [OOPSLA'14]

Power and energy implications of the number of threads used - [PPMG'15]

Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency - [IGSC'15]

Haskell in green land: Analyzing the energy behavior of a purely functional language - [SANER'16]

What is greener?

The influence of the JCF on overall energy consumption

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	RoleList J	ms	Role Unresolved List J	ms	Stack J	ms	Vector J	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

Methods	Concurrent HashMap J	ms	Concurrent SkipListMap J	ms	HashMap J	ms	Hashtable J	ms	Linked HashMap J	ms	Properties J	ms	Simple Bindings J	ms	TreeMap J	ms	UIDefaults J	ms	Weak HashMap J	ms
clear	2.0276	94	2.2961	88	1.8395	104	1.5761	94	1.5025	97	2.0777	98	2.1401	106	1.6706	98	1.8143	105	1.9941	95
containsKey	2.3132	105	2.1693	123	2.1343	103	1.8582	94	1.8726	103	1.6018	107	1.8055	99	1.9452	100	2.3366	89	1.9675	108
containsValue	21.5611	2305	7.8032	643	8.3615	683	8.4957	765	6.1326	462	7.3755	692	7.9912	678	9.1771	847	7.9341	714	6.7072	562
entrySet	2.2878	93	2.2363	116	1.8531	108	2.1332	107	1.8362	113	1.7800	97	2.1557	102	2.1617	115	1.7087	105	1.4666	102
get	2.3106	103	1.9972	119	1.8120	102	1.4071	100	1.8252	116	1.7851	97	1.5359	100	2.2331	115	1.5252	89	1.7185	103
iterateAll	2.1041	96	1.8353	115	2.6673	100	1.5343	91	1.6462	111	1.6362	100	2.0472	116	1.9122	111	1.6574	95	1.7139	106
keySet	1.7287	95	2.4889	124	1.6813	114	2.2226	99	1.8328	103	1.4866	92	2.0630	106	2.1680	110	1.5547	99	1.8749	105
put	1.8591	104	2.2888	102	2.4628	92	1.3123	96	2.0338	108	1.7038	107	2.1646	102	1.4355	91	2.1204	93	2.5784	105
putAll	1.4147	95	2.2852	122	1.7564	100	1.5949	105	1.8608	113	1.3097	95	2.1461	112	1.8914	116	2.3094	87	2.0750	108
remove	1.8574	92	2.2131	105	1.9256	109	1.6067	97	2.2300	106	1.9660	98	2.2178	106	1.8133	101	1.6888	92	2.4103	103
values	1.8279	85	2.4690	116	2.5755	109	2.2266	94	2.0009	107	1.9120	111	2.0692	108	1.4467	105	1.6533	100	2.4628	111

The influence of the JCF on overall energy consumption

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

Slower

Faster

Green Software Lab

The influence of the JCF on overall energy consumption

	Collection	Energy (J)		Collection	Time(ms)
	Properties	24.6344	-3	Linked HashMap	1538
	Linked HashMap	24.7736	+1	Weak HashMap	1607
	Hashtable	25.9677	-4	UIDefaults	1667
	UIDefaults	26.303	+1	Properties	1693
	Weak HashMap	26.9691	+3	HashMap	1724
	TreeMap	27.8546	-3	Simple Bindings	1735
	Simple Bindings	28.3363	+1	Hashtable	1742
	HashMap	29.0692	+3	Concurrent SkipListMap	1771
	Concurrent SkipListMap	30.0826	+1	TreeMap	1909
	Concurrent HashMap	41.2922	0	Concurrent HashMap	3267

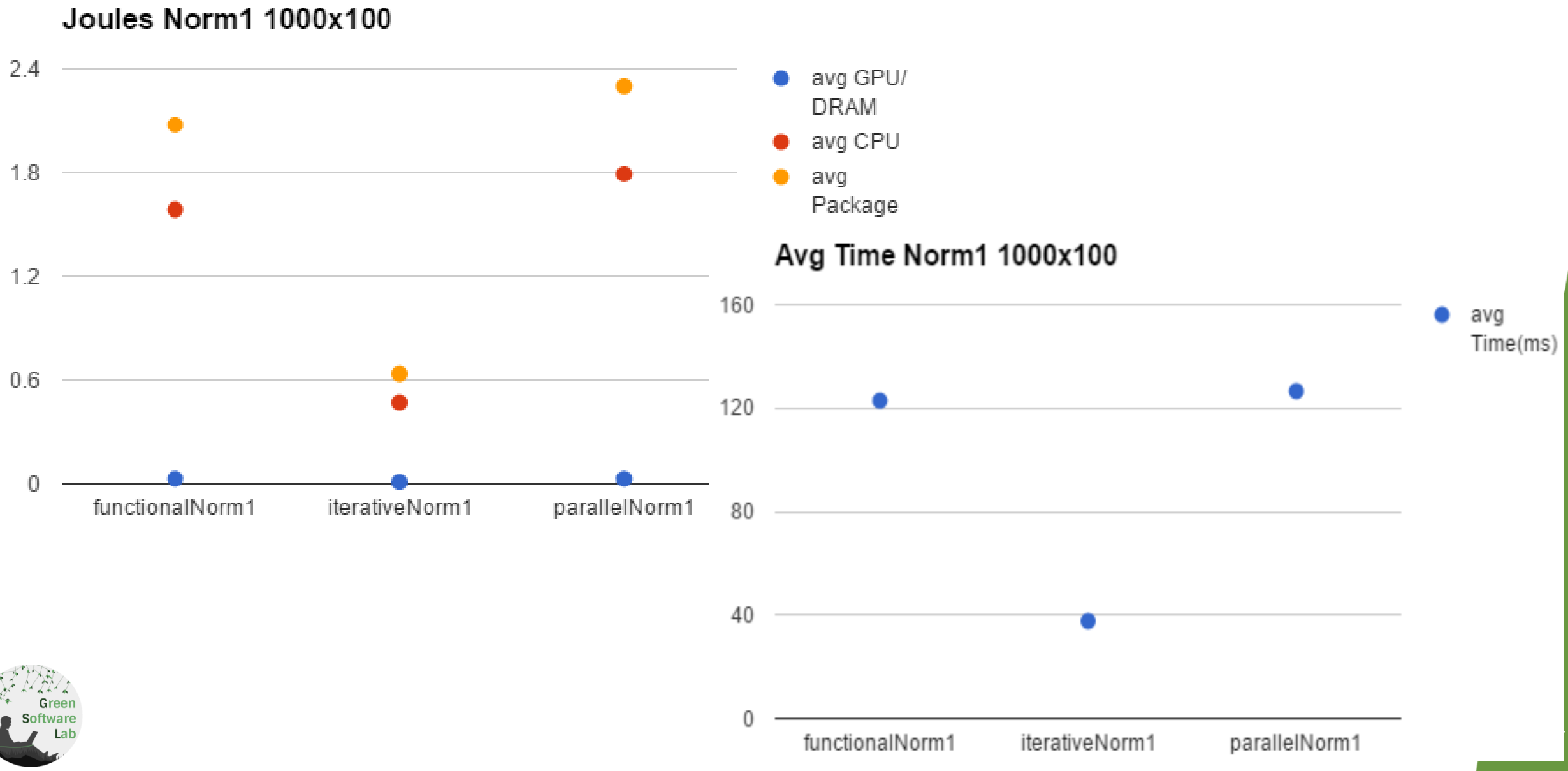
Functional Java

```java
public Matrix iterativeMultiply(){
 Matrix res = null;
 if (popSize < 2)
   return null;
 else{
   res = mList.get(0).times(mList.get(1));
   for (int i = 2 ; i<popSize; i++){
     res = res.times(mList.get(i));
   }
 }
 return res;
}
```

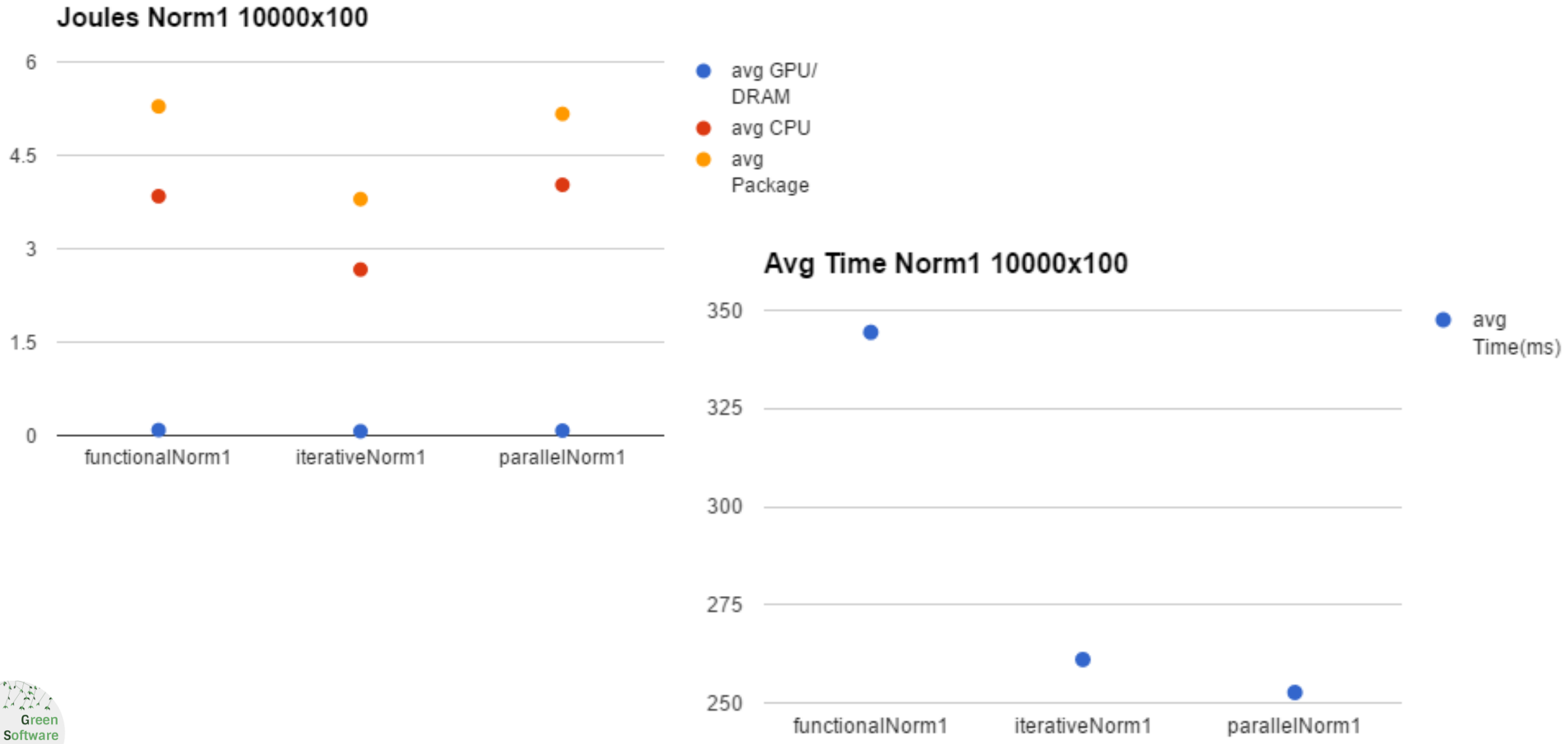
```java
public Matrix functionalMultiply(){
 return mList.stream().reduce(null, (a,b)-> times(a,b));
}

public Matrix parallelMultiply(){
 return mList.parallelStream().reduce(null, (a,b)-> times(a,b));
}
```

Green Software Lab

Functional Java

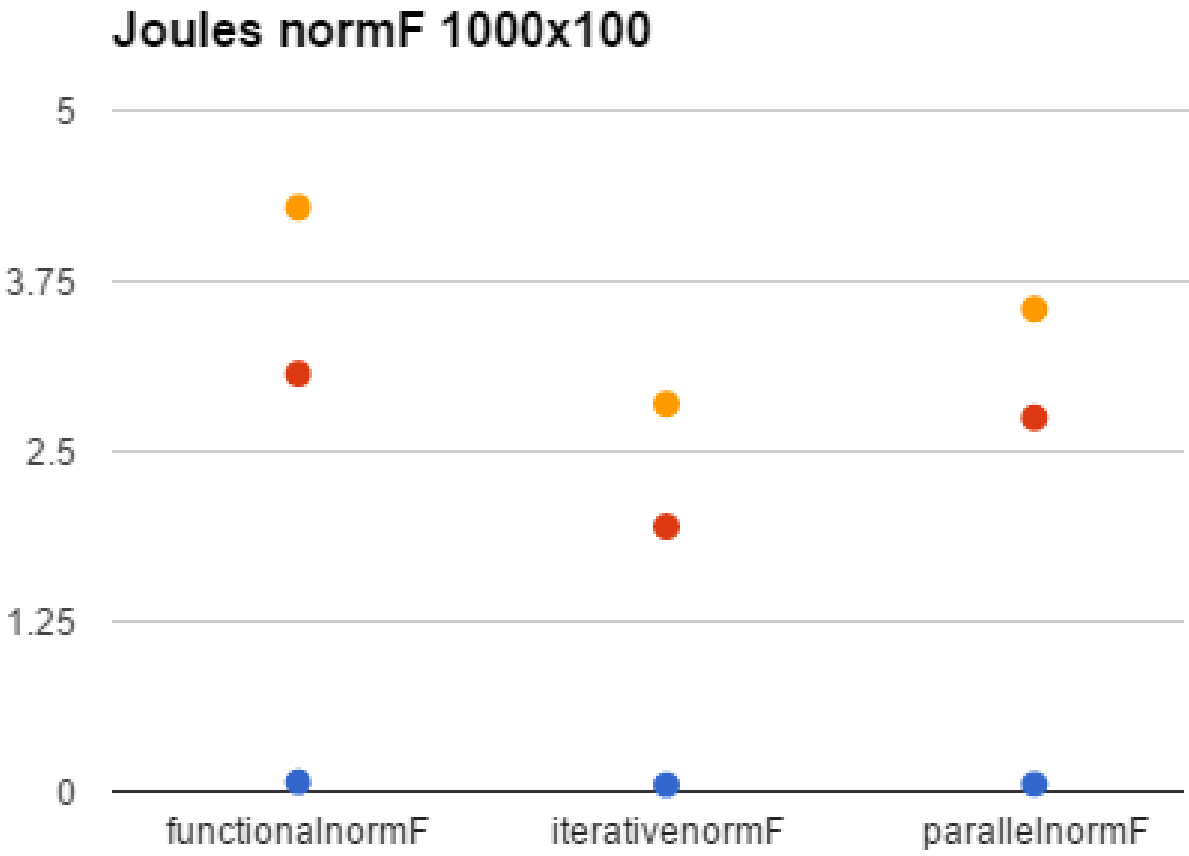


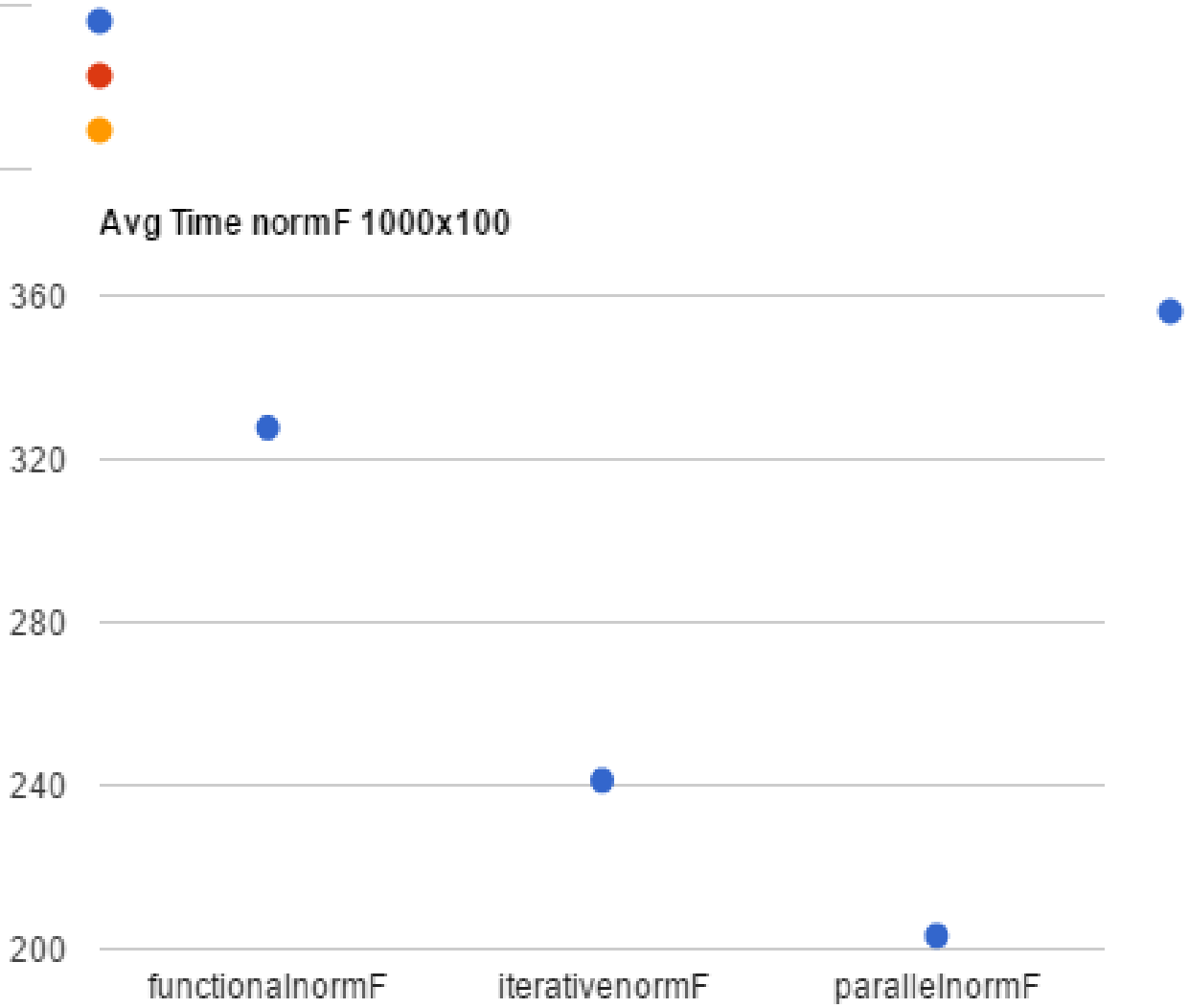
Functional Java



Green Software Lab

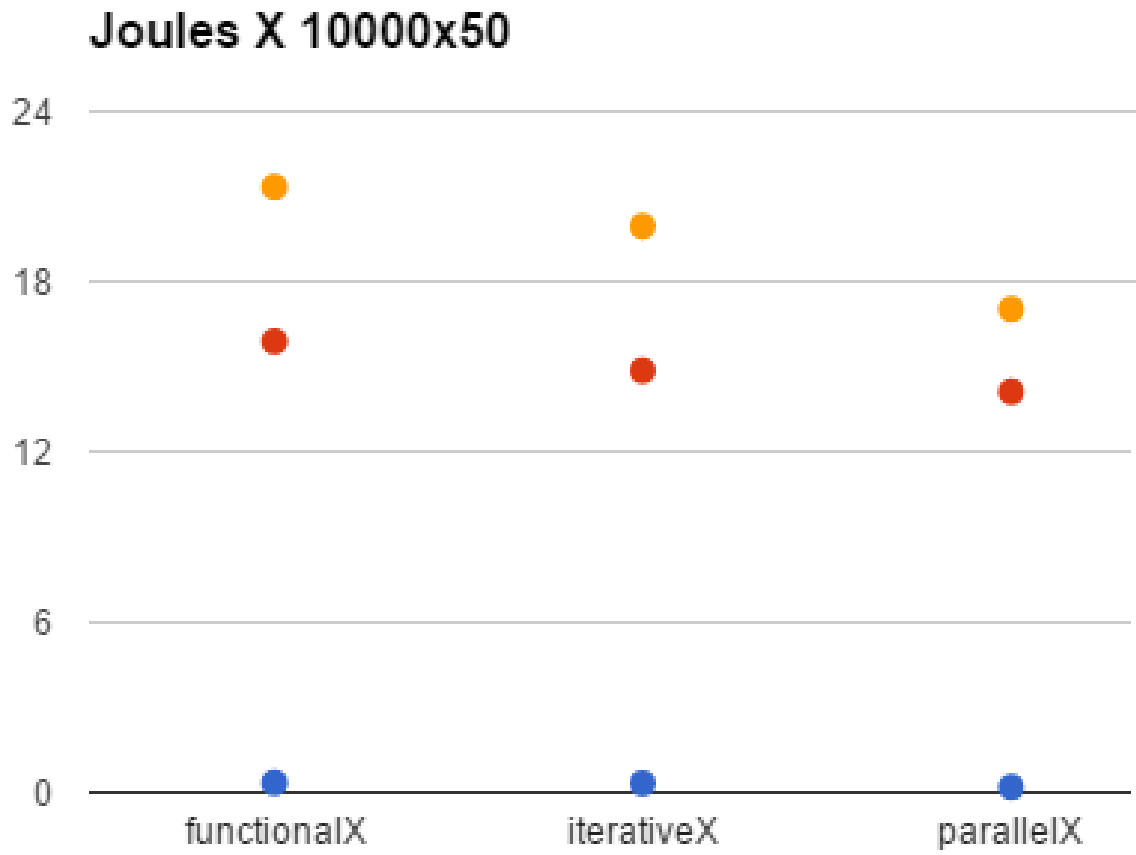
Functional Java

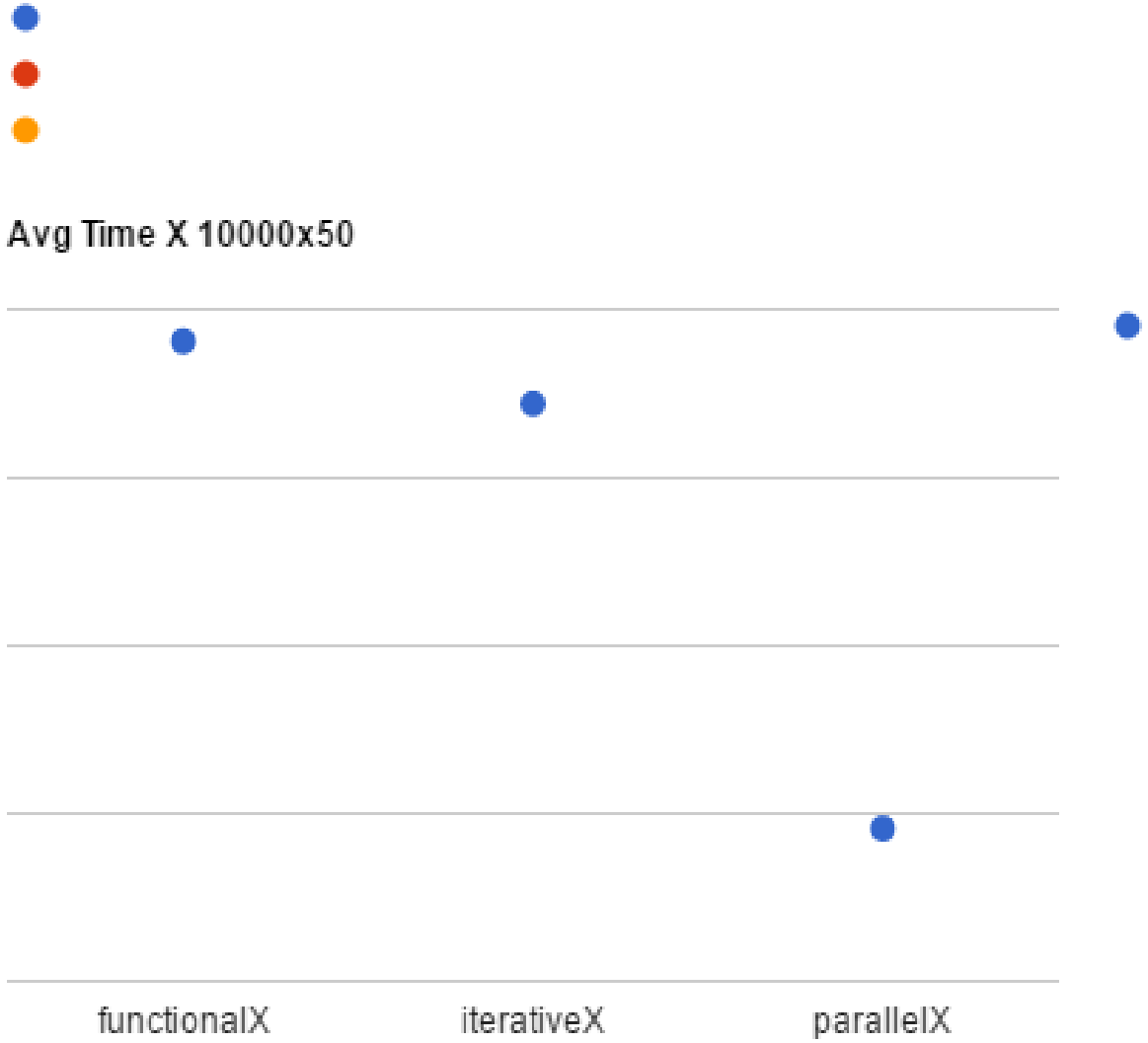




Green Software Lab

Functional Java



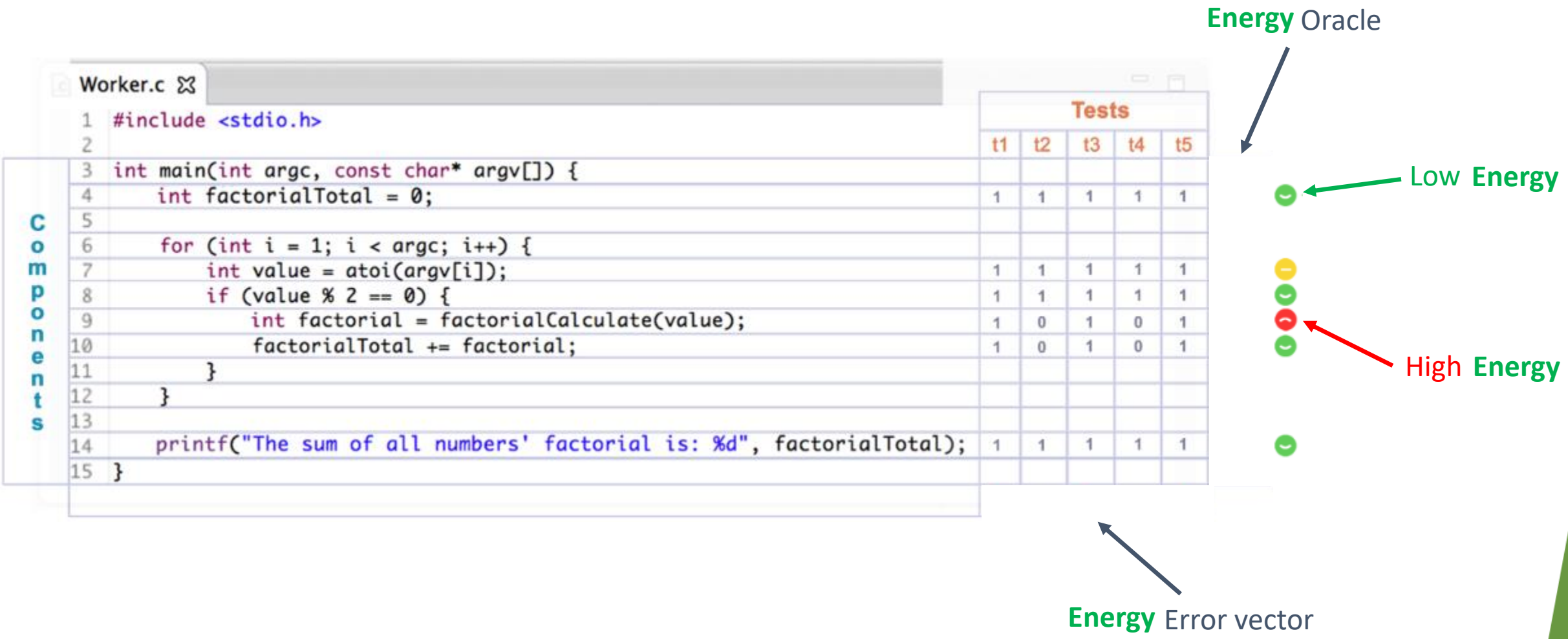


Green Software Lab

SPELL - Spectrum-based Energy Leak Localization

Spectrum-based Fault Localization (SFL)

Spectrum-based Fault Localization (SFL)




```
1  #include <stdio.h>
2
3  int main(int argc, const char* argv[]) {
4      int factorialTotal = 0;
5
6      for (int i = 1; i < argc; i++) {
7          int value = atoi(argv[i]);
8          if (value % 2 == 0) {
9              int factorial = factorialCalculate(value);
10             factorialTotal += factorial;
11         }
12     }
13
14     printf("The sum of all numbers' factorial is: %d", factorialTotal);
15 }
```

	t1	t2	t3	t4	t5
3					
4	1	1	1	1	1
5					
6					
7	1	1	1	1	1
8	1	1	1	1	1
9	1	0	1	0	1
10	1	0	1	0	1
11					
12					
13					
14	1	1	1	1	1
15					

Problem: Energy cannot be defined as binary values

SPELL

Hit Spectrum

$$\text{Hit Spectrum} = \begin{pmatrix} \text{Energy} \\ \text{Number} \\ \text{Time} \end{pmatrix}$$

where Energy =

$\{E_{CPU}, E_{DRAM}, E_{fans}, E_{BT}, E_{GPU}, E_{Wi\text{-}Fi}, E_{etc}\}$

Component Similarity

$$\phi_j = (\alpha_1(A(j), e), \alpha_2(A(j), e), \alpha_3(A(j), e))$$

$$\text{where} \quad \alpha_x(A(j), e) = \frac{\sum_{i=1}^{n} \min(A(j,x)_i, e(x)_i)}{\sum_{i=1}^{n} \max(A(j,x)_i, e(x)_i)}$$

Global Similarity

$$\psi_j = \alpha(\text{global}_c, global_e)$$

$$\text{where} \quad \alpha(c, e) = \frac{\sum_{i=1}^{n} \min(c_i, e_i)}{\sum_{i=1}^{n} \max(c_i, e_i)}$$

Error Vector

$$e = [\sum_{j=1}^{m} \lambda_{1,j} \quad \sum_{j=1}^{m} \lambda_{2,j} \quad \dots \quad \sum_{j=1}^{m} \lambda_{n,j}]^T$$

$$\text{where} \quad \sum_{j=1}^{m} \lambda_{i,j} = (\sum_{j=1}^{m} E_{i,j}, \sum_{j=1}^{m} N_{i,j}, \sum_{j=1}^{m} T_{i,j})$$

Global Error Vector

$$\text{global}_e(i) = \sum_{j=1}^{m} \text{global}_c(\lambda_{i,j})$$

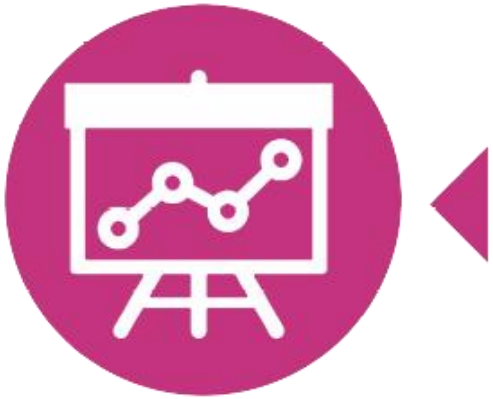
$$\text{where} \quad \text{global}_c(\lambda_{i,j}) = EF_{i,j} \times T_{i,j} \times N_{i,j}$$

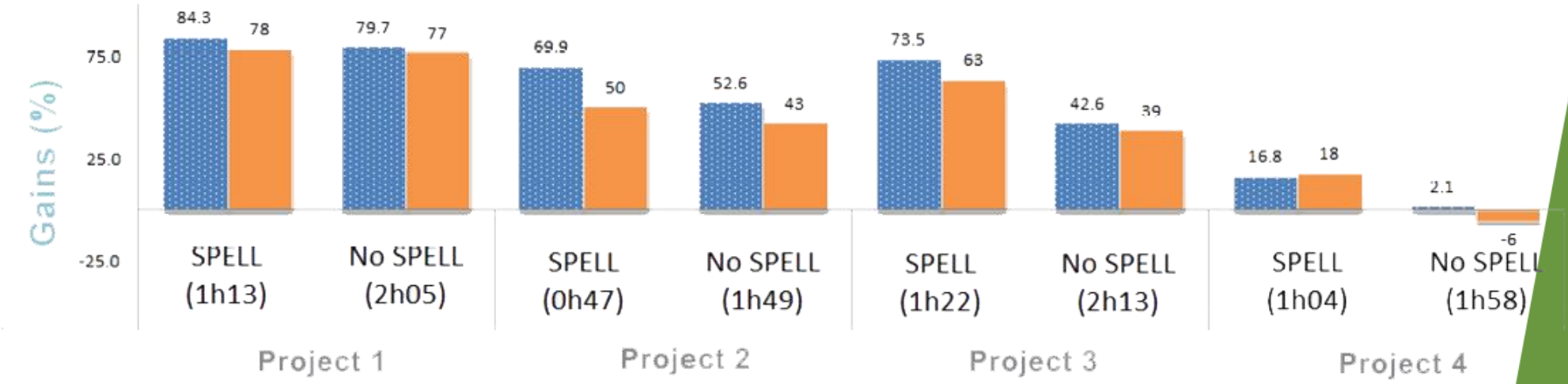
Tests

Components	t1	t2	t3	t4	t5	Φ	Ψ
c1	(37, 1, 75)	(38, 3, 77)	(36, 1, 73)	(37, 3, 74)	(39, 2, 75)	0.2314, 0.3125, 0.3104	0.2466
c2	(61, 2, 102)	(50, 1, 103)	(58, 1, 102)	(66, 2, 105)	(54, 3, 100)	0.3577, 0.2813, 0.4249	0.4678
c3	(0, 0, 0)	(34, 2, 42)	(35, 1, 43)	(0, 0, 0)	(61, 2, 43)	0.1485, 0.2188, 0.1203	0.1450
c4	(42, 1, 34)	(44, 1, 37)	(0, 0, 0)	(61, 2, 43)	(65, 2, 60)	0.2623, 0.1875, 0.1444	0.1406
e	(140, 4, 211)	(166, 7, 259)	(129, 3, 218)	(164, 7, 222)	(209, 11, 295)		
global_e	5659.98	6260.08	3416.66	9288.8	14310.6		

Green Software Lab

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

Green Software Lab

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

Green Software Lab

Green Software Lab

- 3 Year FCT project
- University of California @ Irvine
- University of Kosice
- Ericson @ Budapest

Universidade do Minho
Escola de Engenharia

· U C ·
UNIVERSIDADE
DE COIMBRA

UNIVERSIDADE NOVA DE LISBOA
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT

TÉCNICO
LISBOA

Green Software Lab

Green Software Lab

- Find more info @ our page: http://greenlab.di.uminho.pt/

Questions?

Green Computing as an Engineering Discipline

Rui Pereira

Talk @ InfoBlender

ruipereira@di.uminho.pt