Software Energy Efficiency:
Between Technical and Human Approaches

Dr Adel Noureddine
Associate Professor
University of Pau and Pays de l’Adour

Green Software and Human Actors: design, code, and behavior - community workshop
ICT4S 2023 - Rennes, France

5 June 2023
About me

Adel Noureddine
Associate Professor in Computer Science
@University of Pau and Pays de l’Adour
@LIUPPA laboratory

Researcher in Green IT, Software Engineering and Autonomic Computing

noureddine.org
What is software?
Software?

- Software is intangible (cannot be touched)
- Idea, written in text in a specific language
- Book of instructions to operate a machine
- Automated book of instructions
- Different execution contexts: PC, server, mobile, smart speaker, etc.
Software energy?

- Only physical machines consume energy (mainly electric power)
- Machines execute the instruction of software
- Energy consumed by machine to execute instructions is defined as software energy
How to measure software energy?
Energy and power

- Energy is the quantity of effort transferred to an object to achieve a certain work.
- Power is the quantity of energy consumed per unit of time.
- Electric power is the amount of electric energy transferred by an electric circuit.
Measuring software energy

- Physical meters (power meters, sensors, power plugs, etc.)
- Software power meters (estimation models, APIs, etc.)
- Most notable APIs: Intel RAPL (Intel & AMD CPUs), Nvidia SMI (Nvidia GPUs)
- Most notable tools: PowerJoular, PowerAPI, Scaphandre, many more...
- Monitor source code energy: JoularJX, pyJoules
- Black box or white box measurements
PowerJoular: multi-platform monitoring software

PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools

Adel Noureddine

Université de Pau et des Pays de l’Adour, E2S UPPA, LIUPPA, Anglet, France

In the 18th International Conference on Intelligent Environments (IE2022)
PowerJoular

- PowerJoular is a GNU/Linux tool to monitor power consumption of hardware and software
- Support multiple architectures: x86/64, ARM on Raspberry Pi and Asus Tinker Board
- Monitors CPU with Intel RAPL or our power models on ARM
- Monitors GPU with Nvidia SMI
PowerJoular

- Can monitor hardware (CPU, GPU) and software (process and applications)
- Provides a systemd service (daemon) to continuously monitor power
- Expose power consumption on runtime in terminal and CSV files
- Low overhead (Ada, compiled to native code)
- GPL 3
- noureddine.org/research/joular/powerjoular
- github.com/joular/powerjoular
Monitoring processes and applications

- For a process: collects CPU statistics from `/proc/stat` and `/proc/PID/stat`, to calculate CPU usage of the process every second.
- For an application: every second, search all PIDs of the application (by its name), then monitor and sum their power consumption.
- PowerJoular can keep up with process creation/destruction by applications.
[adel@linux ~]$ sudo powerjoular
System info:
   Platform: intel
   Intel RAPL psys: TRUE
CPU: 1.68 %  28.77 Watts \  -5.04 Watts

PowerJoular Q...  -  x
43.29 watts
Energy consumption of Ray casting algorithm on different programming languages and platforms
JoularJX: source-code level monitoring

PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools

Adel Noureddine

Université de Pau et des Pays de l’Adour, E2S UPPA, LIUPPA, Anglet, France

In the 18th International Conference on Intelligent Environments (IE2022)
JoularJX

- JoularJX is a Java-based agent for power monitoring at the source code level
- Support multiple architectures: x86/64, ARM on Raspberry Pi and Asus Tinker Board (same approach and models as PowerJoular)
- Works on Windows and GNU/Linux
- Real time power monitoring of the source code
• Measures energy for every method of the application and/or the JDK
• Measures methods’ call tree and branches
• Monitor power evolution of every method
• Exposes all monitored data in CSV files
• GPL 3
• noureddine.org/research/joular/joularjx
• github.com/joular/joularjx
How it works

Code monitoring cycle

Application monitoring cycle
How it works

10 samplings
45% of energy

12 samplings
55% of energy
How it works

- Code monitoring cycle
- Application monitoring cycle

- t to t + d

10 samplings
45% of energy

12 samplings
55% of energy

- t to t + d
Example of a sample GUI
Energy consumption of the Java implementation of the Ray casting methods

- RayCasting.main: 139.75 Joules
- RayCasting.intersects: 431.16 Joules
- RayCasting.contains: 53.31 Joules
Results folder structure
<table>
<thead>
<tr>
<th>JoularJX: evolution through time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power consumption evolution of each method</td>
</tr>
</tbody>
</table>

| 1 | 1675460936, 1.1271 |
| 2 | 1675460938, 13.2983 |
| 3 | 1675460933, 6.4101 |
| 4 | 1675460997, 6.1450 |
| 5 | 1675460934, 8.0175 |
| 6 | 1675460998, 1.6743 |
| 7 | 1675460993, 1.5137 |
| 8 | 1675460928, 4.2620 |
| 9 | 1675460992, 2.6224 |
| 10 | 1675460931, 4.1710 |
| 11 | 1675460995, 13.8719 |
| 12 | 1675460938, 6.4126 |
| 13 | 1675460925, 2.7802 |
| 14 | 1675460988, 7.7874 |
| 15 | 1675460926, 18.6637 |
| 16 | 1675460998, 6.9797 |
| 17 | 1675460921, 18.6875 |
| 18 | 1675460985, 1.9955 |
| 19 | 1675460928, 1.8296 |
| 20 | 1675460984, 3.1813 |
| 21 | 1675460923, 8.0808 |
| 22 | 1675460987, 4.3382 |
| 23 | 1675460916, 8.8189 |
| 24 | 1675460918, 4.9692 |
| 25 | 1675460982, 8.6361 |
| 26 | 1675460913, 3.7777 |
| 27 | 1675460912, 5.6473 |
| 28 | 1675460915, 6.4254 |
| 29 | 1675460908, 5.2418 |
| 30 | 1675460910, 6.1808 |
| 31 | 1675460907, 4.4196 |
JoularJX: call tree & branches

Energy consumption of the call tree and branches

```plaintext
MainArray.main; MainArray.ArrayAccess; ArrayList2.access, 49.9456
MainArray.main; ArrayList2.access, 122,5503
```
Optimizing software energy
Optimizing software energy

- Source code optimizations (instructions, snippets): HashMap vs ArrayMap, reduce function calls, reduce objects creation, recursive/iterative algorithms, etc.
- Three principles of eco-design (Syntec numérique, 2013):
  - Minimize software usage time and the required resources to deliver the software workload (build efficient software)
  - Reduce the functionalities of software to its most adapted usage by users (avoid software bloatware)
  - Forecast the durability and evolution over time of the software solutions
- Reduce *feature creep* (addition of new features that go beyond the basic functionalities of an application)
- Avoid *bloatware* (software that become slower and use more hardware resources to deliver the same functionalities or suffer from feature creep than previous versions)
Software bloat

Software bloat: Android

<table>
<thead>
<tr>
<th>Android Version</th>
<th>Release Date</th>
<th>LOC¹</th>
<th>LOC Growth %</th>
<th>GNU Make Build Time²</th>
<th>Build Time Growth %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>May 2010</td>
<td>8,837,858</td>
<td>-</td>
<td>28m55s</td>
<td>-</td>
</tr>
<tr>
<td>2.3.4</td>
<td>April 2011</td>
<td>11,492,324</td>
<td>30%</td>
<td>33m10s</td>
<td>15%</td>
</tr>
<tr>
<td>4.0.1</td>
<td>October 2011</td>
<td>12,827,330</td>
<td>12%</td>
<td>1h13m54s</td>
<td>123%</td>
</tr>
<tr>
<td>4.1.1</td>
<td>July 2012</td>
<td>15,028,331</td>
<td>17%</td>
<td>1h28m11s</td>
<td>19%</td>
</tr>
<tr>
<td>4.2.2</td>
<td>February 2013</td>
<td>15,266,803³</td>
<td>2%</td>
<td>1h32m56s</td>
<td>2%</td>
</tr>
</tbody>
</table>

¹ LOC: 1 LOC = 1,000 characters. ² Builds performed on an 8-core 16GB RAM server. ³ LOC data as of December 2012.
Software bloat: Linux kernel
Software bloat

Studying co-running avionic real-time applications on multi-core COTS architectures, 2014

Fig. 1. Evolution of code size in space, avionic and automotive embedded systems
Reduce the usage of these resources

- **Processing resources**
  - Function calls which trigger CPU context switches
- **Memory access**
  - Number of objects created in object-oriented programming
- **Network usage**
  - Limit data transferred to the minimum needed
  - Limit internet connections opening (ex. open connection in a loop)
- **Unused resources**
  - Close database connections, network sockets, etc.
Reduce the usage of these resources

- Hardware component usage and leakage
  - Prefer network triangulation on GPS if an exact position is not needed
  - Synchronize GPS usage with other software and usages
  - Screen wake-up, bright colors on OLED screens
  - Non-core functionalities

- Non-core functionalities
  - Advertisement ("up to 75% of energy in mobile free apps are consumed by 3rd party advertisement modules" - *Fine Grained Energy Accounting on Smartphones with Eprof, 2012*)
Eco-design guidelines

- Référentiel général d’écoconception de services numériques (RGESN) (in French): https://ecoresponsable.numerique.gouv.fr/publications/referentiel-general-ecoconception/
- And many more...
Green software

• Most current approaches are technical and/or dev-centric
• Humans, the ones using software, not coding, are often forgotten and outside the sustainability loop
• End-users have an important role to play
• Shifting towards user-centric sustainable software
• For example: green software feedback and behavioral change in using software (Noureddine et al., 2023)
At a crossroad...
Path ahead

- What motivates users to sustainability
- How they understand green software and its energy consumption
- Study the impact of green feedback on user behaviors
- Drive end-users behavioral changes
My current research around the topic

- Technical side: measurement tools and energy models, understand factors impacting software energy
- Human side: study the impact of green feedback, push for users behavioral change, Behave project
My current research around the topic

- Technical side: measurement tools and energy models, understand factors impacting software energy
- Human side: study the impact of green feedback, push for users behavioral change, Behave project
- Behave project: noureddine.org/research/behave
Adel Noureddine
Associate Professor in Computer Science
@University of Pau and Pays de l’Adour
@LIUPPA laboratory

Researcher in Green IT, Software Engineering and Autonomic Computing

noureddine.org