GREEN PROGRAMMING

Tina Ulbrich – Hendrik Niemeyer

ROSEN Technology and Research GmbH

C++ on Sea 2025

Follow Along

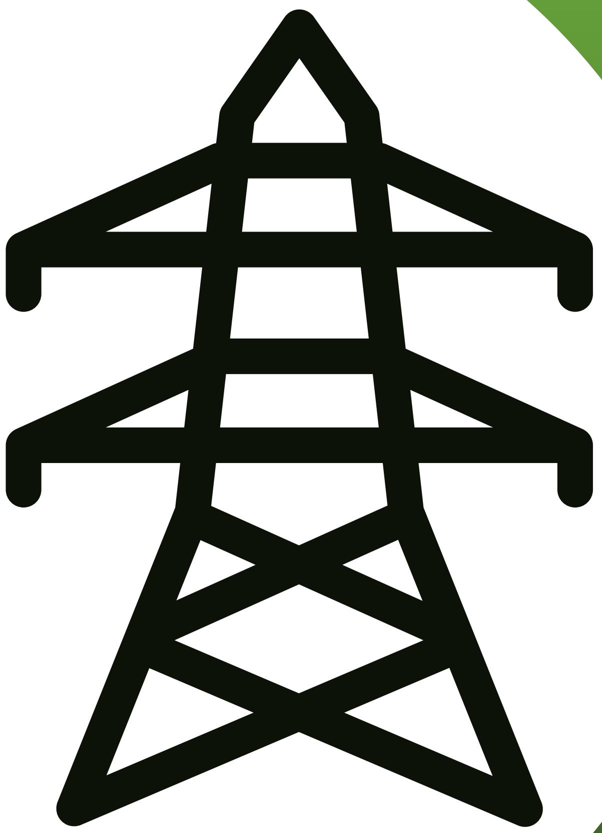
https://tinyurl.com/3chwtrba

OUTLINE

Energy Consumption | Green Software | Green Programming

The Algorithm | Measurements | Results | Evaluation

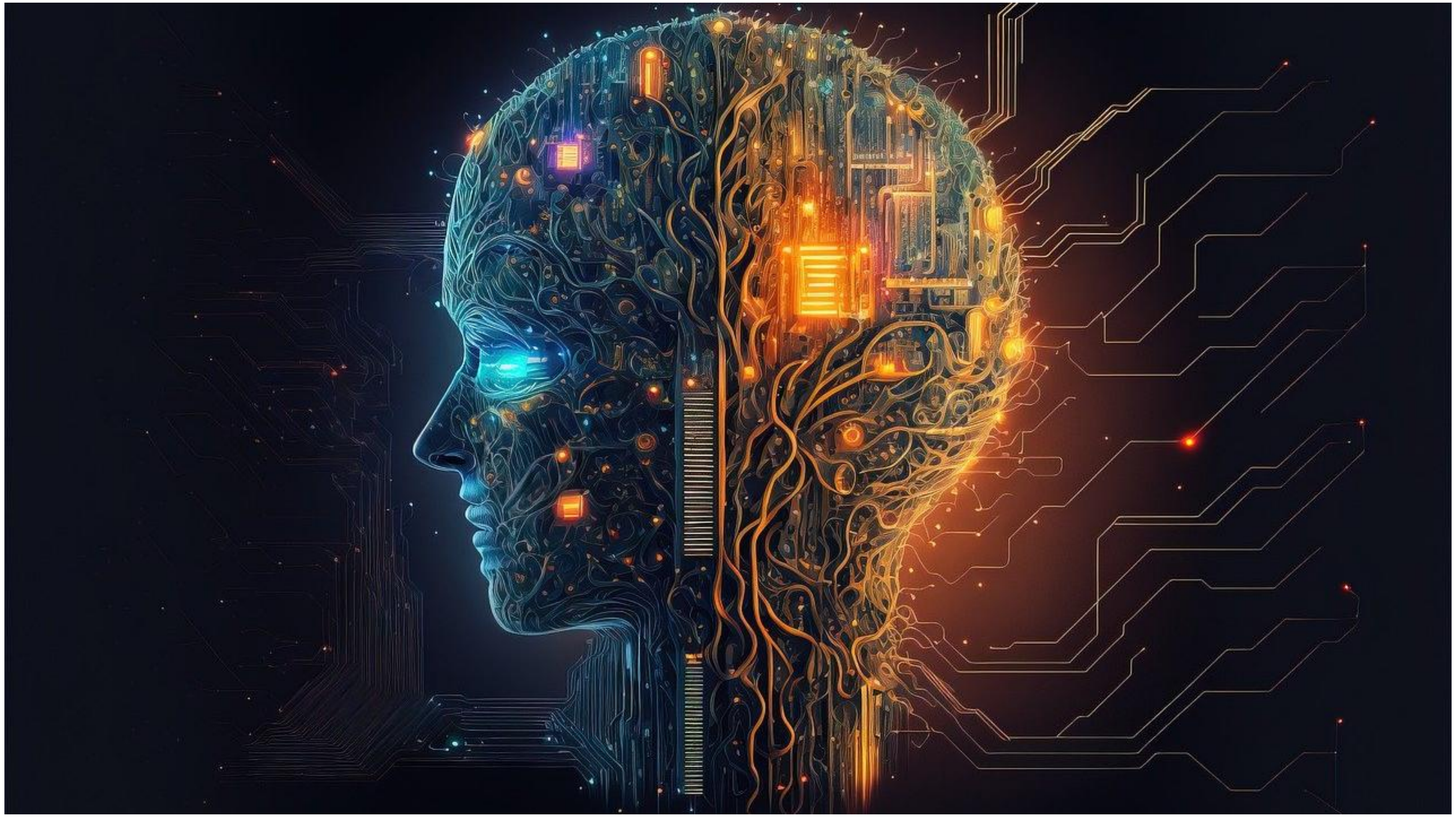
Tools | More than CO_2 | Going Greener | Conclusion



Energy Consumption

Data Centers

Artificial intelligence and machine learning



Increase in data traffic

Growth of cloud services

DATA CENTERS

Blockchain and cryptocurrencies

Data Centers

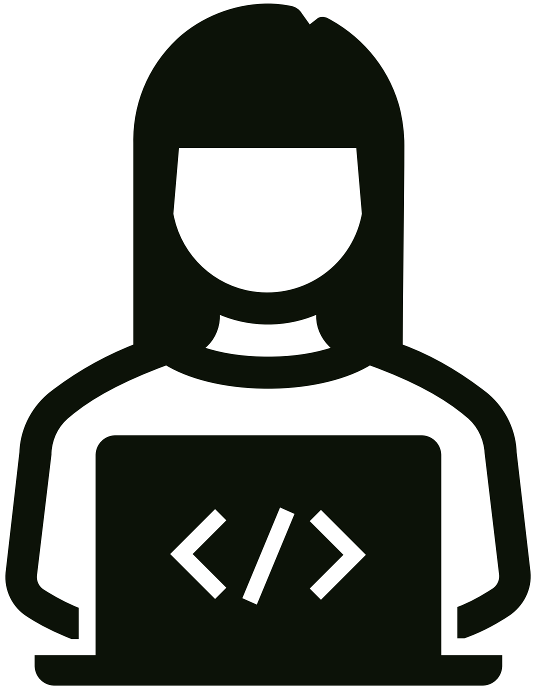
Increased availability and redundancy requirements

Data Centers

Cooling requirements

Data Centers

Exponential growth of networked devices

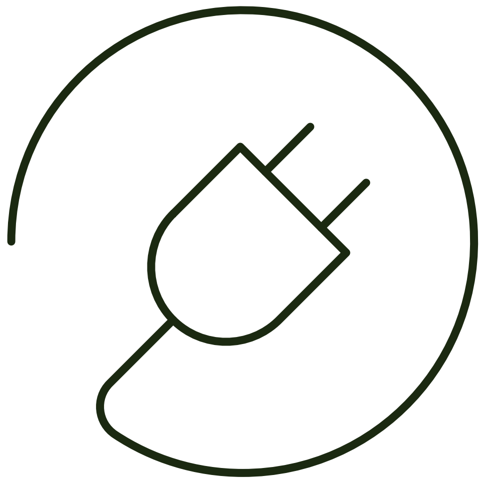


Green Software

Green Software

using fewer physical resources

using energy more intelligently



using less energy

Green Software

Green Software Foundation

https://greensoftware.foundation/

Sustainable Digital Infrastructure Alliance

SUSTAINABLE DIGITAL INFRASTRUCTURE ALLIANCE

https://sdialliance.org/

PLAYING FOR THE PLANET

playing4theplanet.org accu conference 2024

Green Software Architecture

Dos Don'ts and Some Surprises

Giovanni Asproni

Green Programming

Green Programming Definition

Green Programming or green coding is a series of principles applied to software development that aims to reduce the ecological footprint of software.

Research Paper

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Research Paper

Table 1. CLBG corpus of programs.

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

Table 2. Languages sorted by paradigm

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;

Research Paper

fannkuch-redux				
	Energy	Time	Ratio	Mb
(c) C $\Downarrow_2$	215.92	6076	0.036	2
(c) C++ $\Uparrow_1$	219.89	6123	0.036	1
(c) Rust $\Downarrow_{11}$	238.30	6628	0.036	16
(c) Swift $\Downarrow_5$	243.81	6712	0.036	7
(c) Ada $\Downarrow_2$	264.98	7351	0.036	4
(c) Ocaml $\downarrow_1$	277.27	7895	0.035	3
(c) Chapel $\uparrow_1$ $\Downarrow_{18}$	285.39	7853	0.036	53
(v) Lisp $\downarrow_3$ $\Downarrow_{15}$	309.02	9154	0.034	43
(v) Java $\uparrow_1$ $\Downarrow_{13}$	311.38	8241	0.038	35
(c) Fortran $\Downarrow_1$	316.50	8665	0.037	12
(c) Go $\uparrow_2$ $\Uparrow_7$	318.51	8487	0.038	2
(c) Pascal $\Uparrow_{10}$	343.55	9807	0.035	2
(v) F# $\downarrow_1$ $\Downarrow_7$	395.03	10950	0.036	34
(v) C# $\uparrow_1$ $\Downarrow_5$	399.33	10840	0.037	29
(i) JavaScript $\downarrow_1$ $\Downarrow_2$	413.90	33663	0.012	26
(c) Haskell $\uparrow_1$ $\Uparrow_8$	433.68	14666	0.030	7
(i) Dart $\Downarrow_7$	487.29	38678	0.013	46
(v) Racket $\Uparrow_3$	1,941.53	43680	0.044	18
(v) Erlang $\Uparrow_3$	4,148.38	101839	0.041	18
(i) Hack $\Downarrow_6$	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript $\downarrow_4$ $\Uparrow_4$	6,898.48	516541	0.013	26
(i) Jruby $\uparrow_1$ $\Downarrow_4$	7,819.03	219148	0.036	669
(i) Lua $\downarrow_3$ $\Uparrow_{19}$	8,277.87	635023	0.013	2
(i) Perl $\uparrow_2$ $\Uparrow_{12}$	11,133.49	249418	0.045	12
(i) Python $\uparrow_2$ $\Uparrow_{14}$	12,784.09	279544	0.046	12
(i) Ruby $\uparrow_2$ $\Uparrow_{17}$	14,064.98	315583	0.045	8

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

RESEARCH PAPER

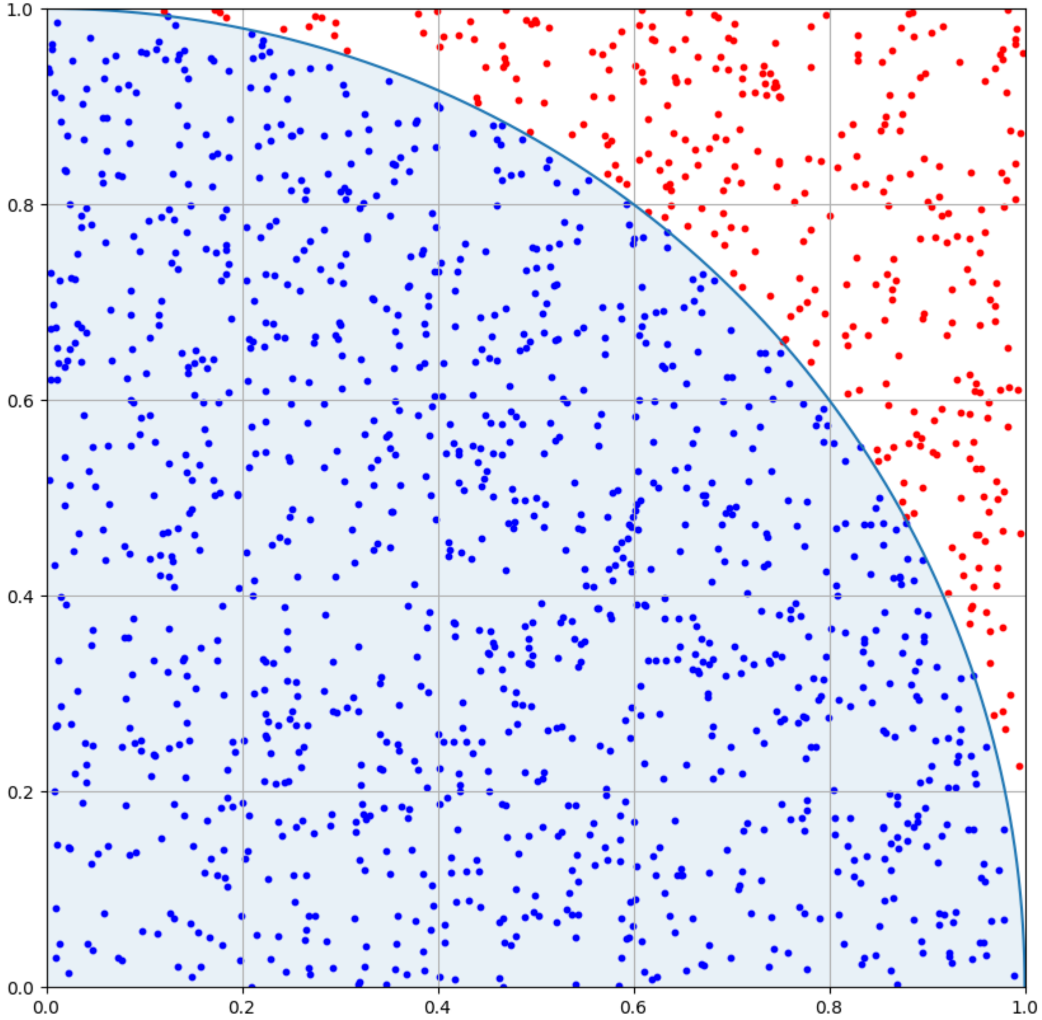
fannkuch-redux				
	Energy	Time	Ratio	Mb
(c) C $\Downarrow_{2}$	215.92	6076	0.036	2
(c) C++ $\Uparrow_{1}$	219.89	6123	0.036	1
(c) Rust $\Downarrow_{11}$	238.30	6628	0.036	16
(c) Swift $\Downarrow_{5}$	243.81	6712	0.036	7
(c) Ada $\Downarrow_{2}$	264.98	7351	0.036	4
(c) Ocaml $\downarrow_{1}$	277.27	7895	0.035	3

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

The Algorithms

Monte Carlo Simulation

$$\pi \approx \frac{inside\ circle}{all\ points} \cdot 4$$



$$\pi \approx \frac{1160}{1500} \cdot 4 = 3{,}0933$$

$$\pi \approx \frac{3921}{5000} \cdot 4 = 3{,}1368$$

$$\pi \approx \frac{39218}{50000} \cdot 4 = 3{,}1374$$

$$\pi \approx \frac{7855055}{10000000} \cdot 4 = 3{,}1420$$

Monte Carlo Simulation

```
auto montecarlo_pi(const int num_iterations)
{
    auto rand = std::random_device();
    auto random_engine = std::default_random_engine(rand());
    auto uniform_dist = std::uniform_real_distribution(0.0, 1.0);

    auto count_inside = 0;
    for (int i = 0; i < num_iterations; ++i)
    {
        const auto x = uniform_dist(random_engine);
        const auto y = uniform_dist(random_engine);
        if ((x * x + y * y) <= 1.0)
        {
            ++count_inside;
        }
    }

    return 4.0 * count_inside / static_cast<double>(num_iterations);
}
```

Monte Carlo Simulation

```
auto montecarlo_pi(const int num_iterations)
{
    auto rand = std::random_device();
    auto random_engine = std::default_random_engine(rand());
    auto uniform_dist = std::uniform_real_distribution(0.0, 1.0);

    auto count_range = std::views::iota(0, num_iterations)
        | std::views::filter([&](const auto) {
                              const auto x = uniform_dist(random_engine);
                              const auto y = uniform_dist(random_engine);
                              return (x * x + y * y) <= 1.0;
                          });

    return 4.0 * std::ranges::distance(count_range) / static_cast<double>(num_iterations);
}
```

Monte Carlo Simulation

```
def monte_carlo_pi(num_iterations):
    inside_circle = 0

    for _ in range(num_iterations):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)

        if x * x + y * y <= 1.0:
            inside_circle += 1

    return 4.0 * inside_circle / num_iterations
```

ADLER-32 CHECKSUM

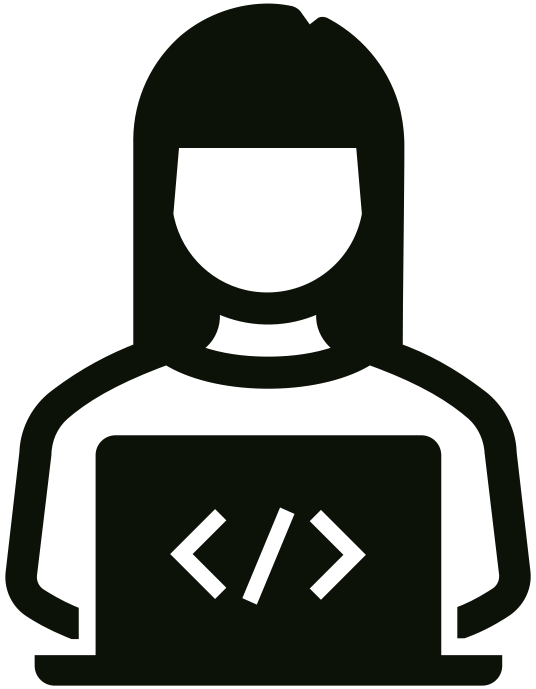
```cpp
uint32_t adler32(const std::vector<uint8_t>& data)
{
    constexpr uint32_t mod_adler = 65521;

    uint32_t a = 1;
    uint32_t b = 0;

    for (size_t i = 0; i < data.size(); ++i)
    {
        a = (a + data[i]) % mod_adler;
        b = (b + a) % mod_adler;
    }

    return (b << 16) | a;
}
```

Adler-32 Checksum

```cpp
uint32_t adler32_ranges(const std::vector<uint8_t>& data)
{
    constexpr uint32_t mod_adler = 65521;

    uint32_t a = 1;
    const auto b = std::ranges::fold_left(data, 0, [&](const auto sum, const auto d) {
        a = (a + d) % mod_adler;
        return (sum + a) % mod_adler;
    });

    return (b << 16) | a;
}
```

Adler-32 Checksum

```python
def adler32(data):
    mod_adler = 65521
    a = 1
    b = 0

    for byte in data:
        a = (a + byte) % mod_adler
        b = (b + a) % mod_adler

    return (b << 16) | a
```



Measurements

Programming Language Categories

machine code	byte code	interpreted code
manual memory	garbage collector	
C++	Java	Lisp

Chosen Programming Languages

C	D	Ruby	Kotlin
C++	Rust	Elixir	Typescript
C#	Python	Java	Lua

Measurement setup

- All code runs within Windows Subsystem for Linux 2
- 8 GB RAM
- Intel Core i7-7700K Cpu @ 4.20 GHz with 8 cores
- Monte Carlo Pi for 10 million iterations
- Adler32 checksum for 50 MB data

Code Carbon

- CodeCarbon Python Extension
  - CodeCarbon assumes 3 Watts for 8 GB of RAM
  - Tracks Intel and AMD processors energy consumption
    - Directly via Intel Power Gadget, RAPL files or the powermetrics tool
    - Fallback: 50 % of TDP of the processor
  - Nvidia GPUs are tracked via pynvml
  - CO_2 emission via energy mix per country
- Compilation is not part of the measurement
- Periphery is also not measured

Code Carbon Code

```python
from codecarbon import EmissionsTracker
import os
import subprocess

os.system("dotnet build -c Release ./csharp/MonteCarloPi") # compilation step (not measured)
with EmissionsTracker(project_name="csharp") as tracker:
     os.system("dotnet run -c Release  --project ./csharp/MonteCarloPi/") # execution step (measured)

with EmissionsTracker(project_name="python") as tracker:
     os.system("python3 python/montecarlopi.py")

...
```

Code Carbon Offline Mode

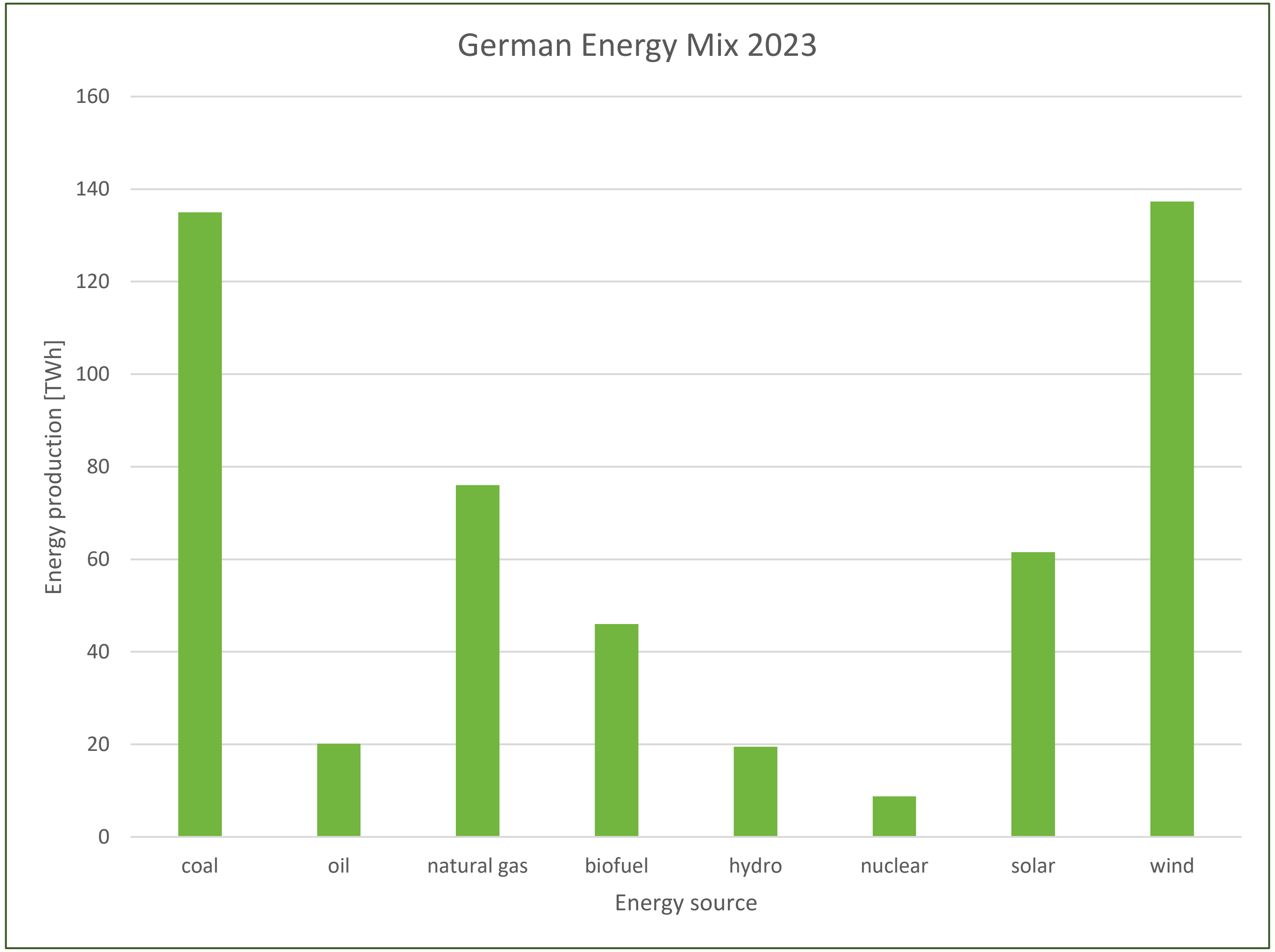
```
from codecarbon import OfflineEmissionsTracker
import os
import subprocess

os.system("dotnet build -c Release ./csharp/MonteCarloPi") # compilation step (not measured)
with OfflineEmissionsTracker(project_name="csharp“, country_iso_code=“NOR“) as tracker:
     os.system("dotnet run -c Release  --project ./csharp/MonteCarloPi/") # execution step (measured)
```

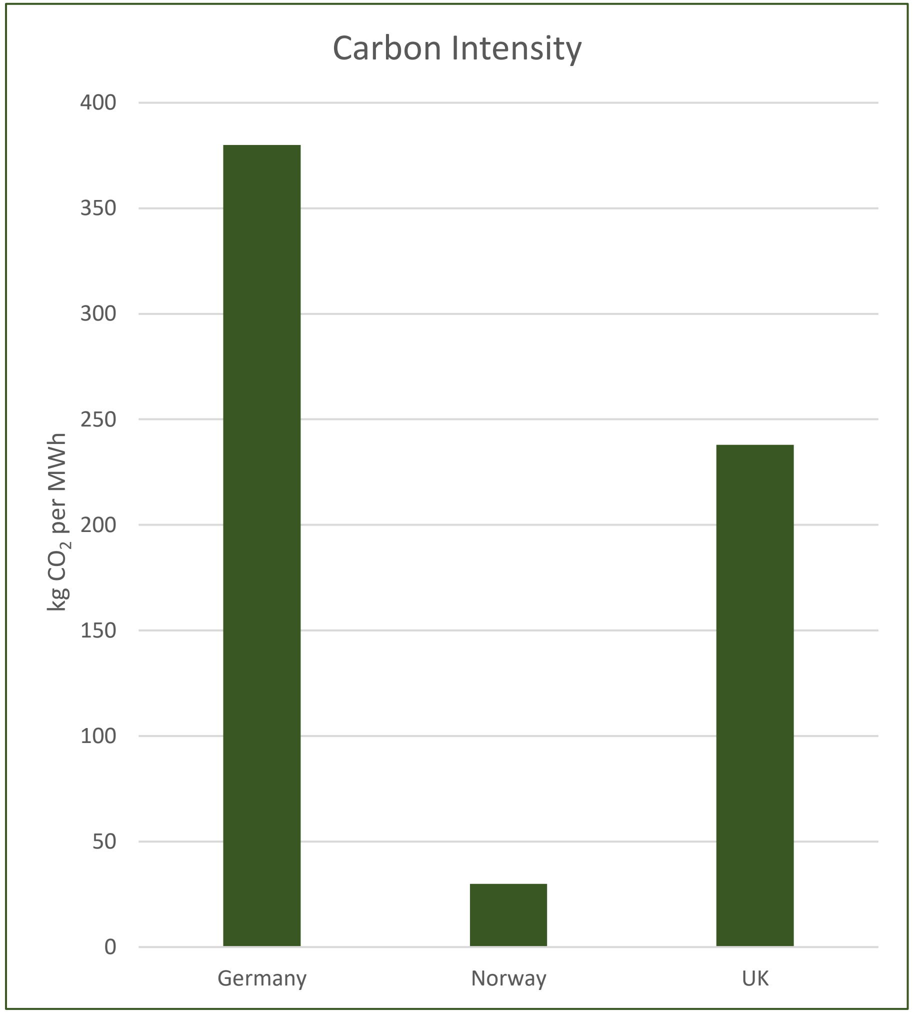
How are CO_2 Emissions Calculated

Energy Source	Carbon Intensity (kg/MWh)
Coal	995
Petroleum	816
Natural Gas	743
Geothermal	38
Hydroelectricty	26
Nuclear	29
Solar	48
Wind	26

Example country: 60 % coal, 40 % solar: 616.2 kg CO_2 / MWh

ENERGY MIXES





Cloud providers

- Google Cloud CO_2 emissions
- Renewable energy only AWS regions
- AWS carbon footprint dashboard
- Emissions Impact Dashboard for Azure

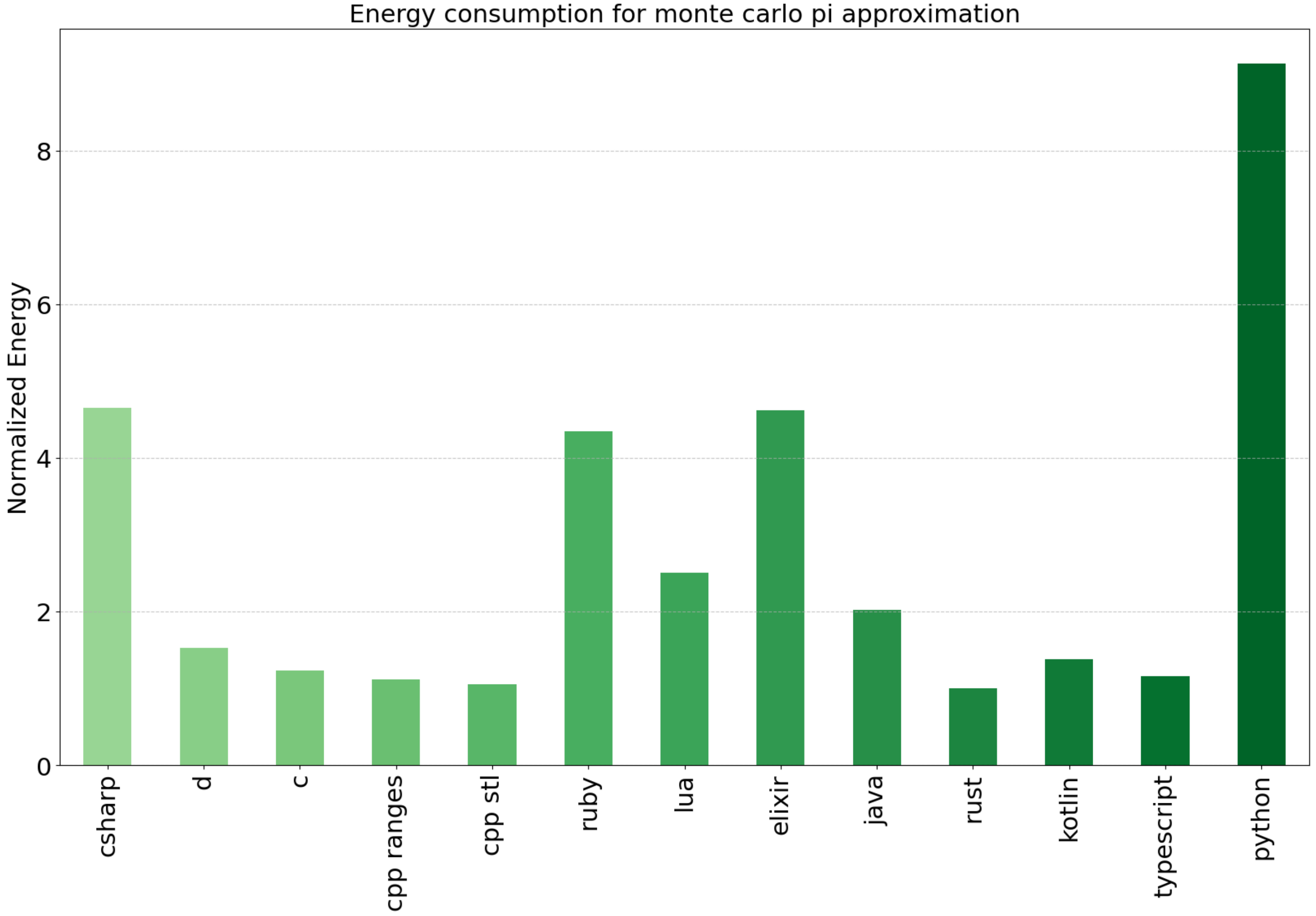
Results

Results

1 normalized energy unit Monte Carlo Pi	3.48e-6 kWh

CO_2 emission MonteCarlo (Germany)	1.33 mg CO_2
CO_2 emission MonteCarlo (Norway)	0.10 mg CO_2
CO_2 emission MonteCarlo (UK)	0.83 mg CO_2

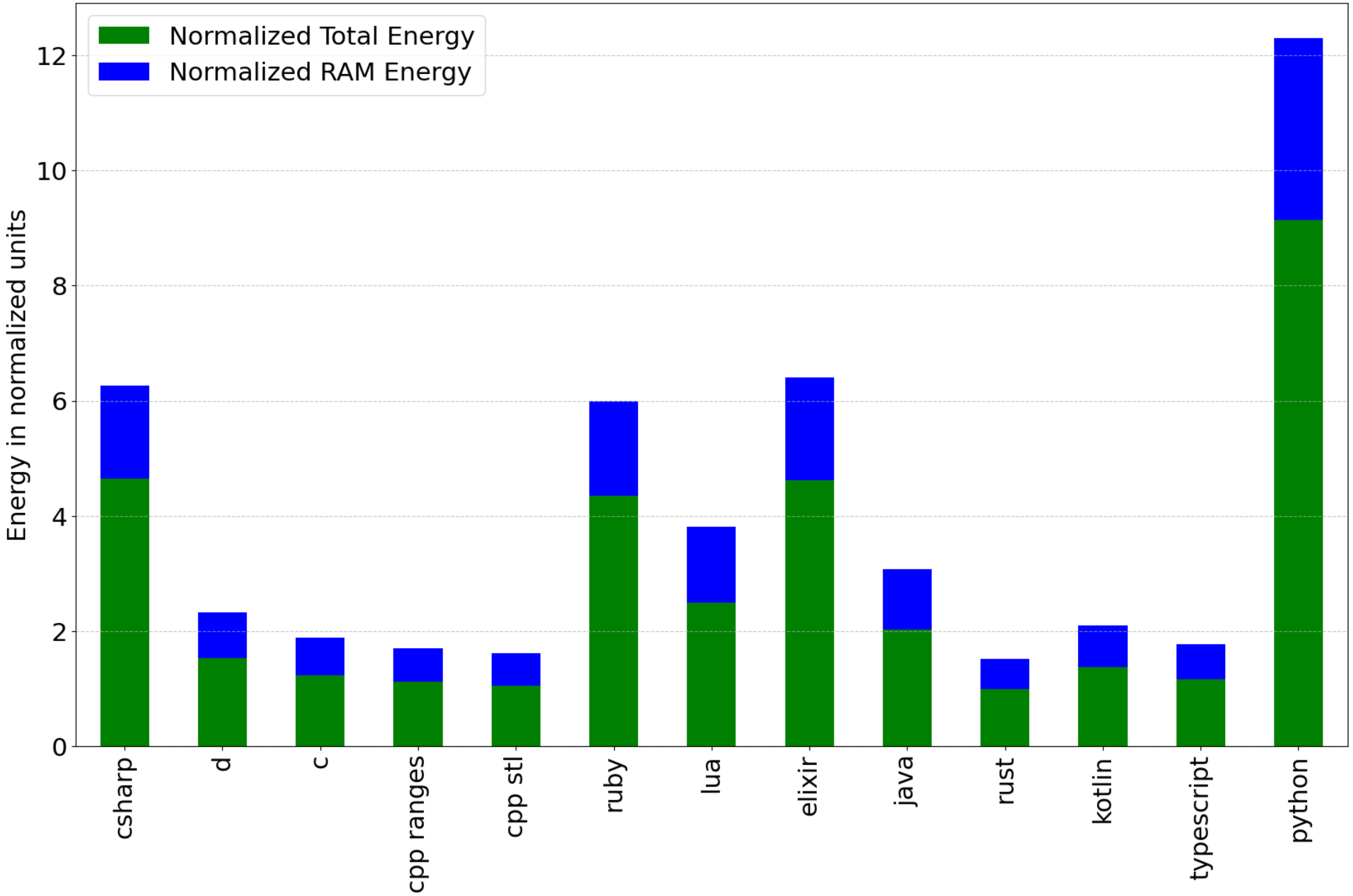
Results Monte Carlo Pi



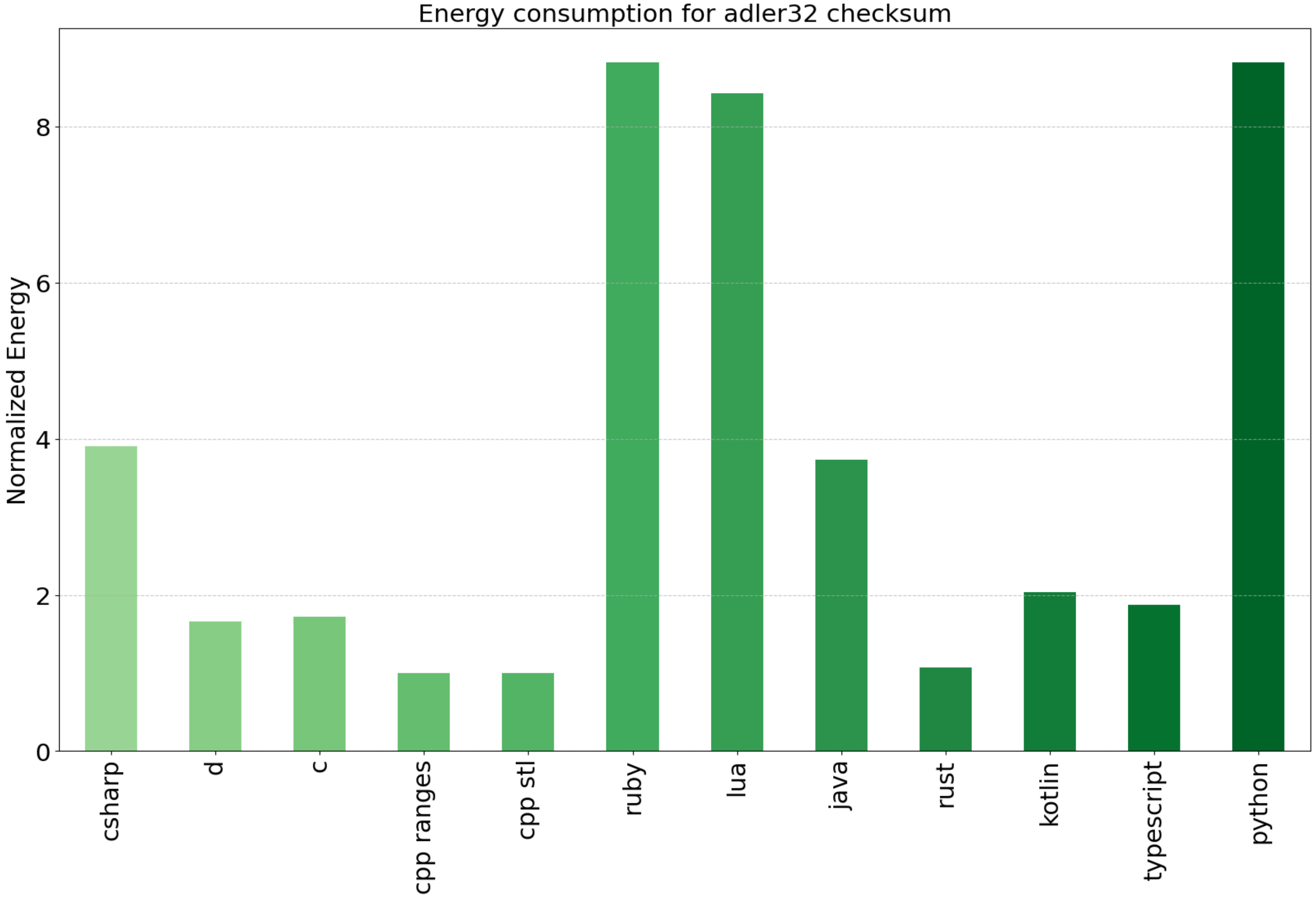
Results Monte Carlo Pi

Language	Norm. total energy	Norm. RAM energy
C++	1.06	0.55
C++ with ranges	1.12	0.59
C	1.24	0.65
Rust	1.00	0.52
D	1.53	0.80
Typescript	1.17	0.61
Kotlin	1.38	0.72
Python	9.13	3.16
Ruby	4.35	1.65
Lua	2.50	1.31
Elixir	4.62	1.78
Java	2.02	1.06
C#	4.65	1.62

Results Monte Carlo Pi



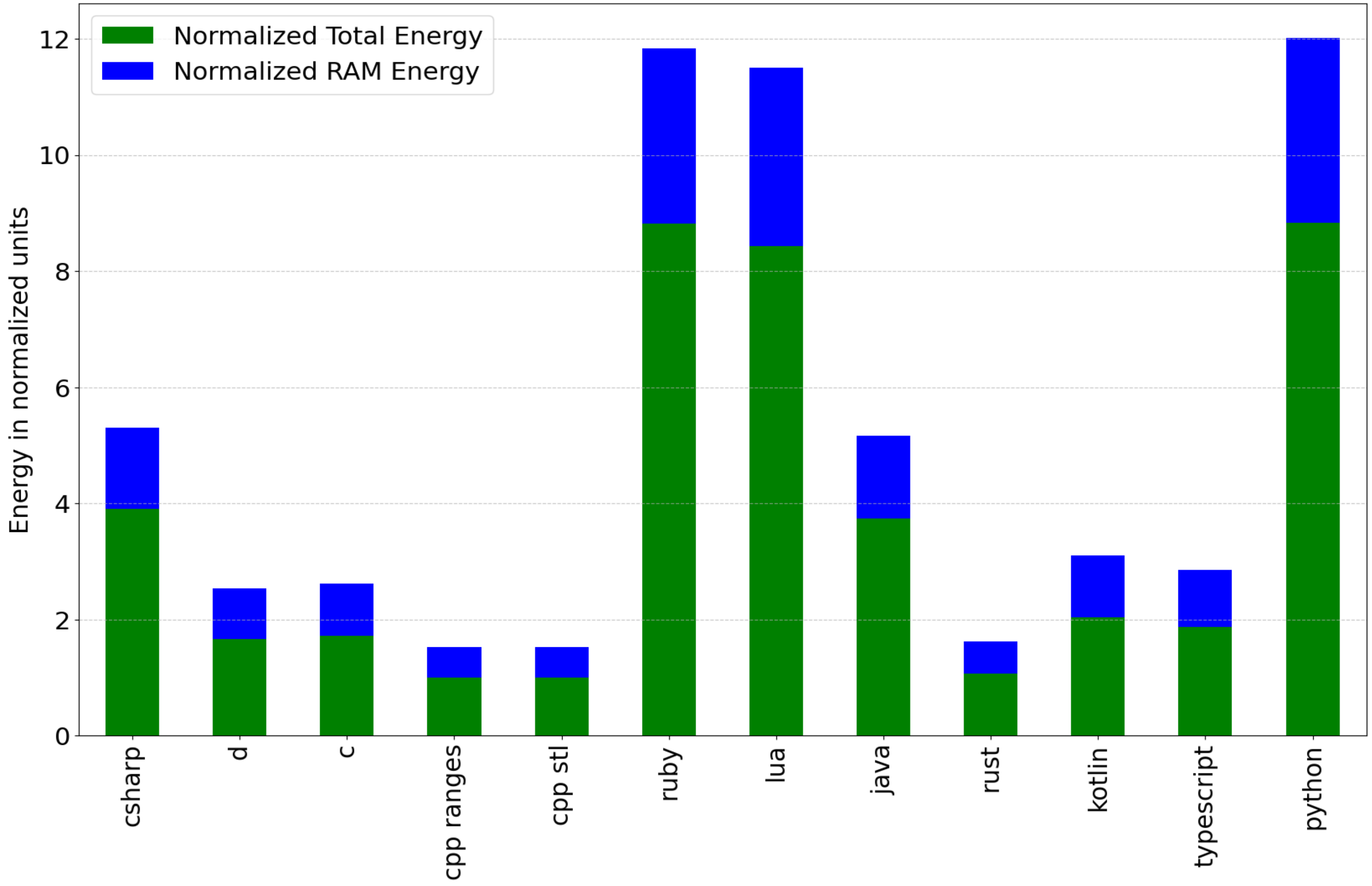
Results Adler32



Results Adler32

Language	Norm. total energy	Norm. RAM energy
C++	1.00	0.52
C++ with ranges	1.00	0.53
C	1.72	0.90
Rust	1.07	0.56
D	1.66	0.87
Typescript	1.88	0.98
Kotlin	2.04	1.07
Python	8.83	3.19
Ruby	8.83	3.01
Lua	8.43	3.08
Java	3.74	1.43
C#	3.91	1.39

RESULTS ADLER32



Evaluation

EVALUATION

- Duration is the main driver of energy consumption
- RAM is a small contributing factor

EVALUATION

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++

Evaluation

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs

Evaluation

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites

Evaluation

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites
- Choose the right language for the purpose

Evaluation

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites
- Choose the right language for the purpose
- Use optimized libraries

Evaluation

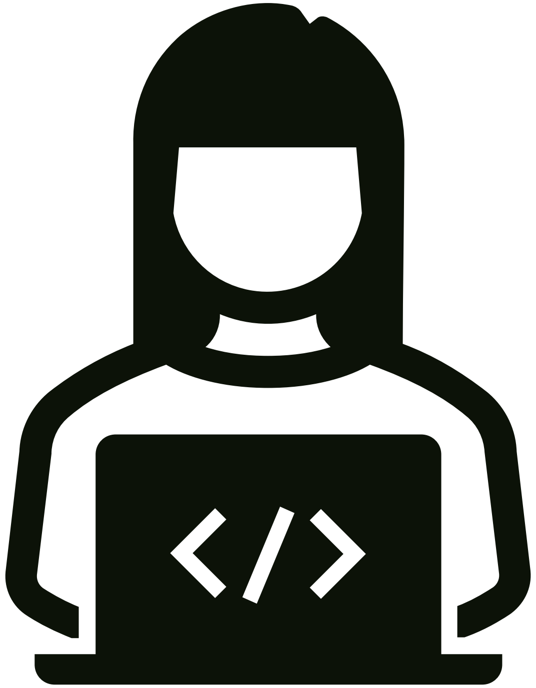
- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites
- Choose the right language for the purpose
- Use optimized libraries
- Alternative implementations of your language

EVALUATION

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites
- Choose the right language for the purpose
- Use optimized libraries
- Alternative implementations of your language
- Analyze runtime complexity of your algorithm

Evaluation

- Duration is the main driver of energy consumption
- RAM is a small contributing factor
- Best energy footprint: C, C++
- Efficient compilers lead to efficient programs
- No large rewrites
- Choose the right language for the purpose
- Use optimized libraries
- Alternative implementations of your language
- Analyze runtime complexity of your algorithm
- Not every program needs to be optimal

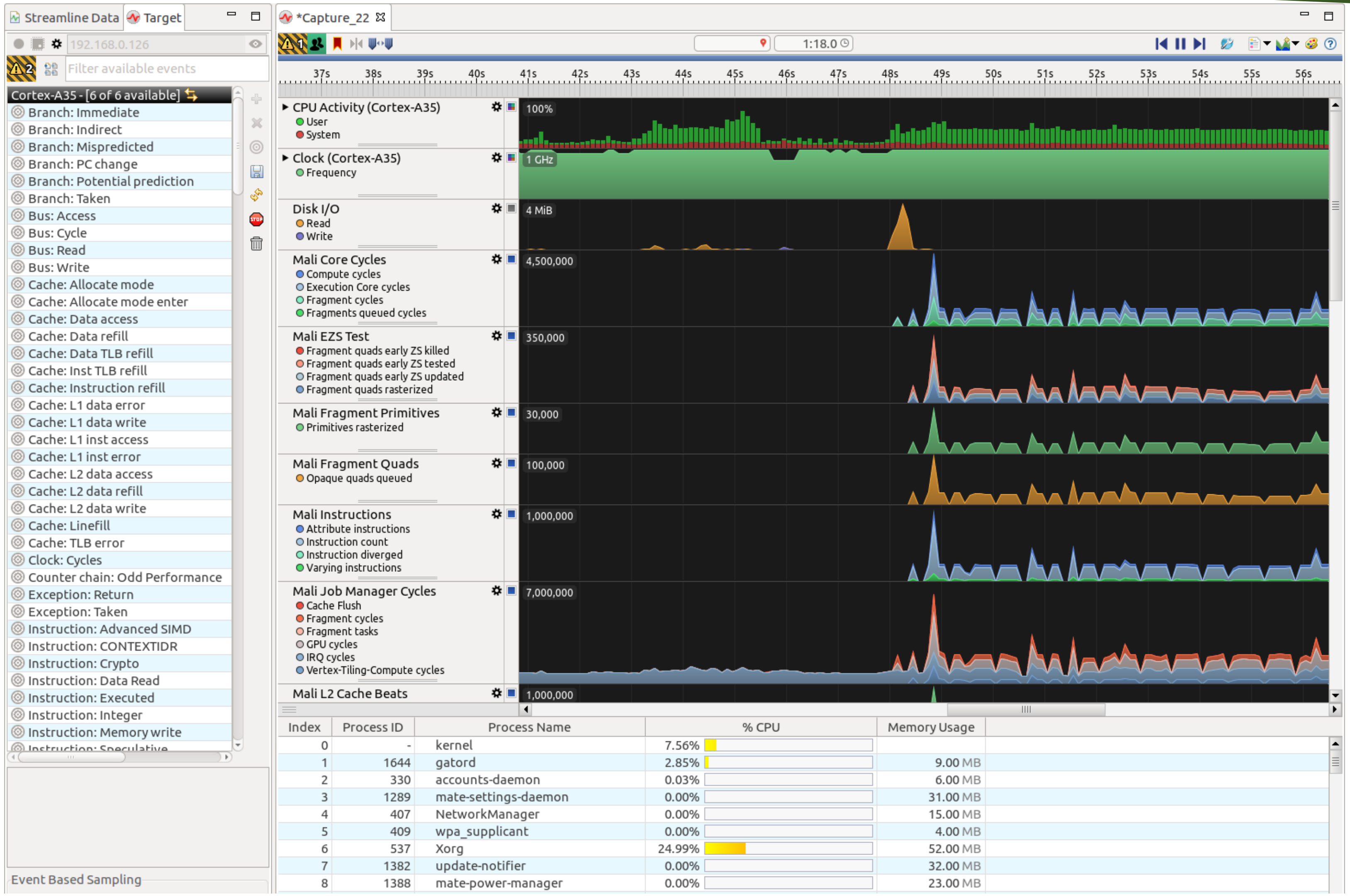


Tools

Intel Performance Counter Monitor

```
Intel(r) UPI data traffic estimation in bytes (data traffic coming to CPU/socket through UPI links):

               UPI0     UPI1     UPI2     UPI3    |  UPI0   UPI1   UPI2   UPI3
--------------------------------------------------------------------------------------------------------------
 SKT    0       43 M     41 M     42 M     41 M   |    0%     0%     0%     0%
 SKT    1       49 M     46 M     47 M     45 M   |    0%     0%     0%     0%
--------------------------------------------------------------------------------------------------------------
Total UPI incoming data traffic:  359 M     UPI data traffic/Memory controller traffic: 0.00

Intel(r) UPI traffic estimation in bytes (data and non-data traffic outgoing from CPU/socket through UPI links):

               UPI0     UPI1     UPI2     UPI3    |  UPI0   UPI1   UPI2   UPI3
--------------------------------------------------------------------------------------------------------------
 SKT    0      131 M    125 M    128 M    122 M   |    0%     0%     0%     0%
 SKT    1      126 M    121 M    124 M    119 M   |    0%     0%     0%     0%
--------------------------------------------------------------------------------------------------------------
Total UPI outgoing data and non-data traffic:  999 M

MEM (GB)->|  READ |  WRITE | PMM RD | PMM WR | CPU energy | DIMM energy | UncFREQ (Ghz)|
--------------------------------------------------------------------------------------------------------------
 SKT   0    253.83     0.10      0.00      0.00     307.19      34.67             2.00
 SKT   1    253.27     0.08      0.00      0.00     305.08      27.96             2.00
--------------------------------------------------------------------------------------------------------------
       *    507.10     0.18      0.00      0.00     612.27      62.63             2.00
```

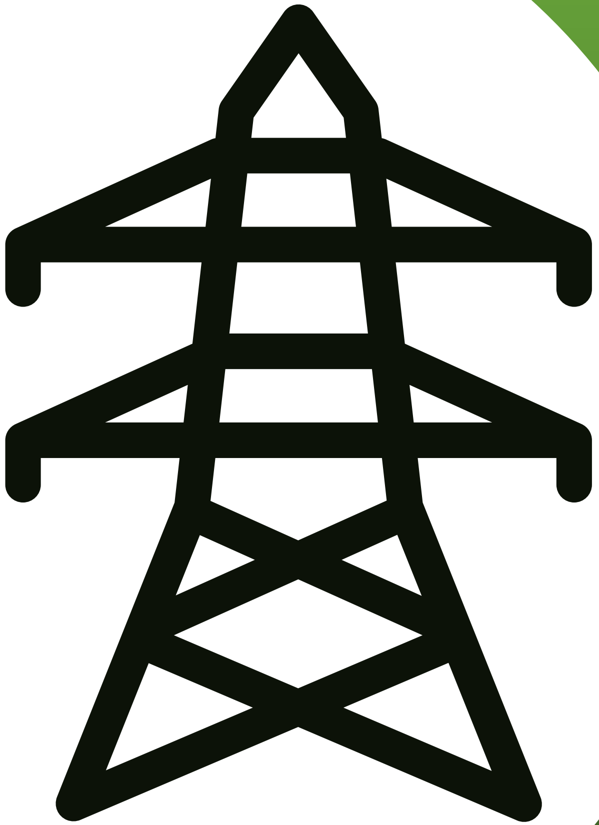
ARM Development Studio



Oaklean

```
export class TypescriptParser {
  static traverseSourceFile( Energy Consumption (self): 0 mJ (0.00%)
    }
  ) {
    const { enter, leave } = callback

    traverseNode(sourceFile)

    function traverseNode(node: ts.Node) { Energy Consumption (self): 0.046052476265895426 mJ (0.05%)
      enter(node)
      ts.forEachChild(node, traverseNode)
      leave(node)
    }
  }

  static posToLoc(sourceFile: ts.SourceFil                                           : 0.
    const lineAndChar = sourceFile.getLine

    return {
      line: lineAndChar.line + 1,
      column: lineAndChar.character
    }
  }

  static isProgramStructureType (node: ts.Node) { Energy Consumption (self): 0.04700711648250533 mJ (0.
    return ts.isFunctionDeclaration(node) ||
      ts.isFunctionExpression(node) ||
```

type	value	unit
profilerHits	714	
CPU Time (self)	6290	µs
CPU Time (summed up)	18925	µs
CPU Time (own code)	12551	µs
CPU Time (libraries)	84	µs
CPU Time (node internal)	0	µs
Energy Consumption (self)	0.04700711648250533	mJ
Energy Consumption (summed up)	0.1221168650037431	mJ
Energy Consumption (own code)	0.07510974852123786	mJ



More than CO_2

Water Usage

- Data center consume water in two ways
  - Indirectly through using power
  - Directly cooling and humidification
- Data Centers are in the top 10 water-consuming commercial industries in the US
- Areas with lots of renewable energy in the US are often water stress regions
- Nature study estimates 57 % consumption drawn from drinking water (2019)

WATER USAGE

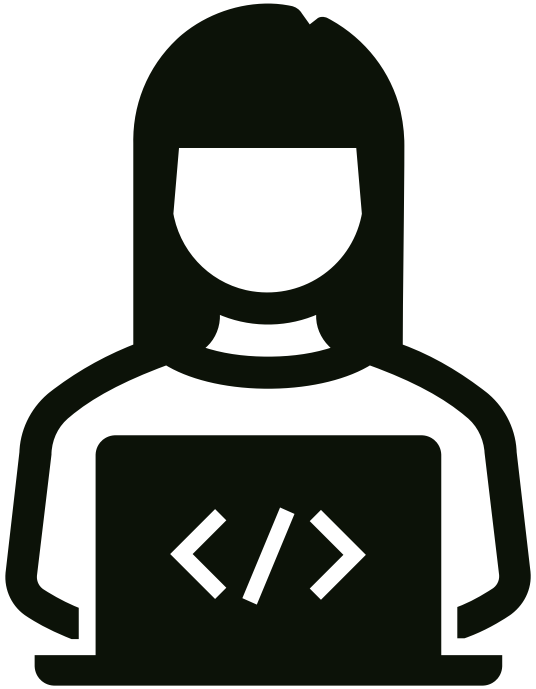
- Data center consume water in two ways
  - Indirectly through using power
  - Directly cooling and humidification
- Data Centers are in the top 10 water-consuming commercial industries in the US
- Areas with lots of renewable energy in the US are often water stress regions
- Nature study estimates 57 % consumption drawn from drinking water (2019)

Water Usage Mitigation

- All major cloud providers have pledged to be „water positive“ by 2030
- Personal contribution: Efficient software and choice

Hardware Manufacturing

- New server: up to 1750 kg CO_2 due to mining, manufacturing and transport
- Some raw materials used for IT equipment are scarce
- Mitigation: Green software and choice



Going Greener

Going Greener

- Green Software by design

Going Greener

- Green Software by design
- Green Requirements

Going Greener

- Green Software by design
- Green Requirements
- Rethink accuracy of stored data

Going Greener

- Green Software by design
- Green Requirements
- Rethink accuracy of stored data
- Move to data centers that use renewable energy

Going Greener

- Green Software by design
- Green Requirements
- Rethink accuracy of stored data
- Move to data centers that use renewable energy
- Clean up code

Going Greener

- Green Software by design
- Green Requirements
- Rethink accuracy of stored data
- Move to data centers that use renewable energy
- Clean up code
- Clean up data

Conclusion

Conclusion

- Worldwide energy consumption is on the rise
- AI needs a lot of energy
- Fastest programming languages are the most efficient
- Runtime and CPU usage is most important
- Optimize for what you need
- Measure energy consumption of your program
- Water consumption and new IT equipment also contribute to environmental damage
- Lots of steps can be taken already to go greener
- Contribute to a greener future by keeping sustainability in mind

	Tina Ulbrich	Hendrik Niemeyer
	@tinaul@mastodon.social	@hniemeyer@mastodon.social
	tina_ulbrich@gmx.de	heniemeyer@icloud.com
	@tinaul.bsky.social	@hniemeyer.bsky.social
#include <C++>	tinaul	hniemeyer

ROSEN Technology and Research GmbH

ROSEN

Sources

- https://www.informatik-aktuell.de/betrieb/server/rechenzentren-energiefresser-oder-effizienzwunder.html
- https://www.heise.de/news/Gruenes-Programmieren-C-und-Rust-energieeffizient-Python-und-Perl-Schlusslicht-7259319.html
- https://scienceblogs.de/rupture-de-catenaire/2021/05/03/die-energie-effizienz-von-programmiersprachen/
- https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
- https://www.goldmansachs.com/insights/articles/AI-poised-to-drive-160-increase-in-power-demand
- https://www.enviam-gruppe.de/energiezukunft-ostdeutschland/verbrauch-und-effizienz/stromverbrauch-ki
- https://www.datacenter-insider.de/nachhaltigkeit-in-code--tools-sprachen-roadmaps-a-4488612f6f7799531b9ad852d0beff35/
- https://datascience.aero/green-programming-reducing-your-carbon-emissions-when-coding/
- https://github.com/Green-Software-Foundation/awesome-green-software
- https://sdialliance.org/
- https://github.com/intel/pcm
- https://developer.arm.com/documentation/101816/0902/Capturing-Energy-Data
- https://www.oaklean.io/
- https://iopscience.iop.org/article/10.1088/1748-9326/abfba1
- https://www.washingtonpost.com/climate-environment/2023/04/25/data-centers-drought-water-use/
- https://think.ing.com/articles/data-centres-growth-in-water-consumption-needs-more-attention/

Sources

- https://www.nature.com/articles/s41545-021-00101-w
- https://nordiccomputer.com/resources/what-is-the-effect-of-datacenter-hardware-on-the-climate
- https://en.wikipedia.org/wiki/Environmental_impact_of_mining#:~:text=Mining%20can%20cause%20erosion%2C%20sinkholes,which%20contributes%20to%20climate%20change.
- https://datacentersustainability.org/data-centers-and-critical-raw-materials/
- https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/fannkuchredux.html#fannkuchredux
- https://benchmarksgame-team.pages.debian.net/benchmarksgame/description/fasta.html
- https://www.playing4theplanet.org/