TOWARDS AN ENERGY EFFICIENT COMPUTATION IN ERLANG

Áron Attila Mészáros

Gergely Nagy

ELTE IK, Computer Science BSc

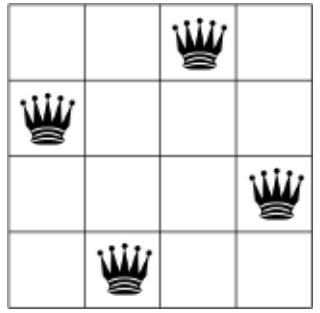
STCS 2019

Motivation

- Environmental awareness
  - Energy conservation, minimizing energy consumption
  - Even in computers
  - Green computing

- Why Erlang?
  - Popular languages – lots of research (e.g. C++, Java, Haskell)
  - Erlang
    - Widely used in the industry
    - No such research yet

Previous Work



- Tool for measuring energy consumption
- Measuring complex algorithms
  - N-queens
  - Sparse matrix multiplication
- Special attention to:
  - Lists vs arrays
  - Higher order functions
  - Parallelization (parallel map)

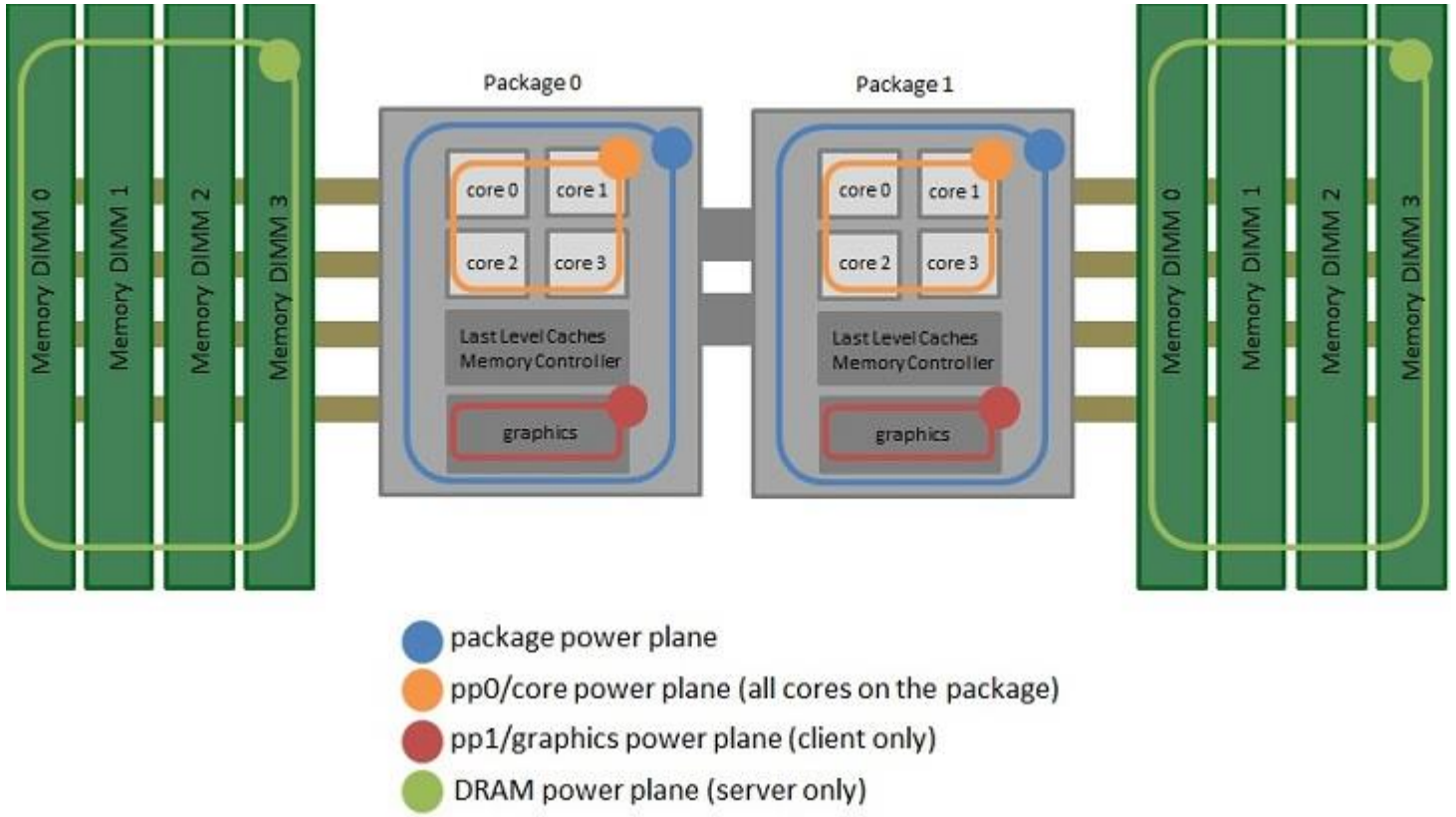
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 5 & 3 & 0 & 7 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 15 & 0 & 16 & 0 \\ 15 & 9 & 0 & 21 \\ 0 & 0 & 12 & 0 \end{bmatrix}$$

New goals

- Improve methodology
- Directly measure language elements
- Measure data structures independently
- Further inspection of parallelization
  - Spawning processes
  - Sending messages
- Refactoring in order to minimize energy consumption

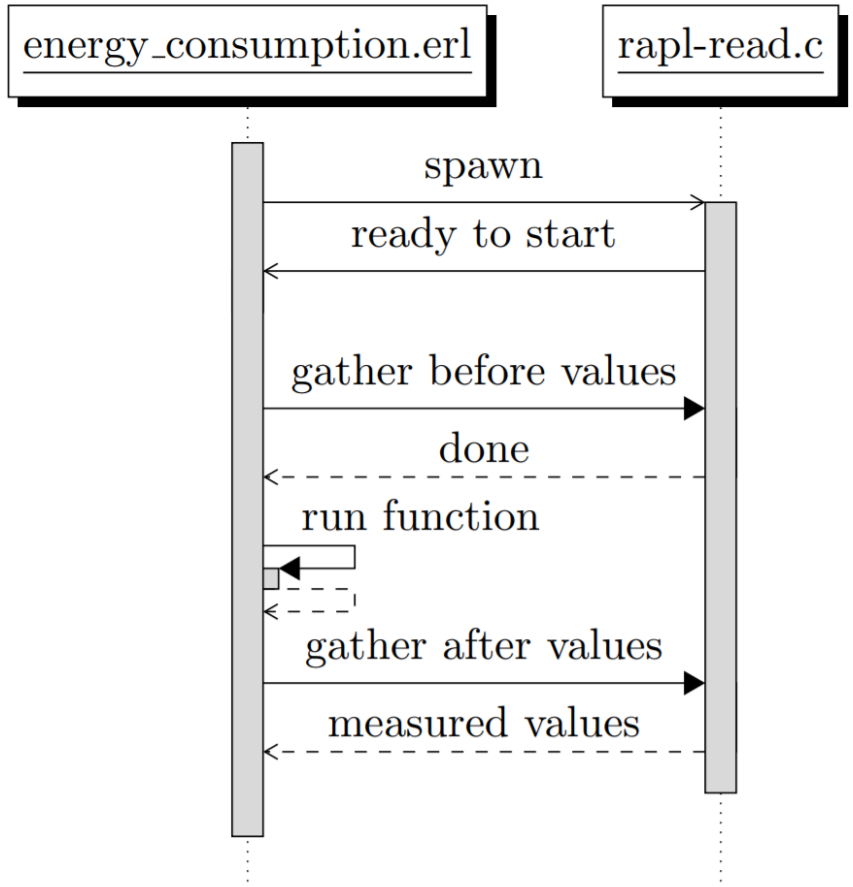
How to measure energy consumption

- Running Avarage Power Limit (RAPL)
  - Intel
  - Methods:
    - MSR
    - perf_event,
    - sysfs
  - Domains:
    - PKG – package
    - PP0 – core
    - PP1 – uncore
    - DRAM



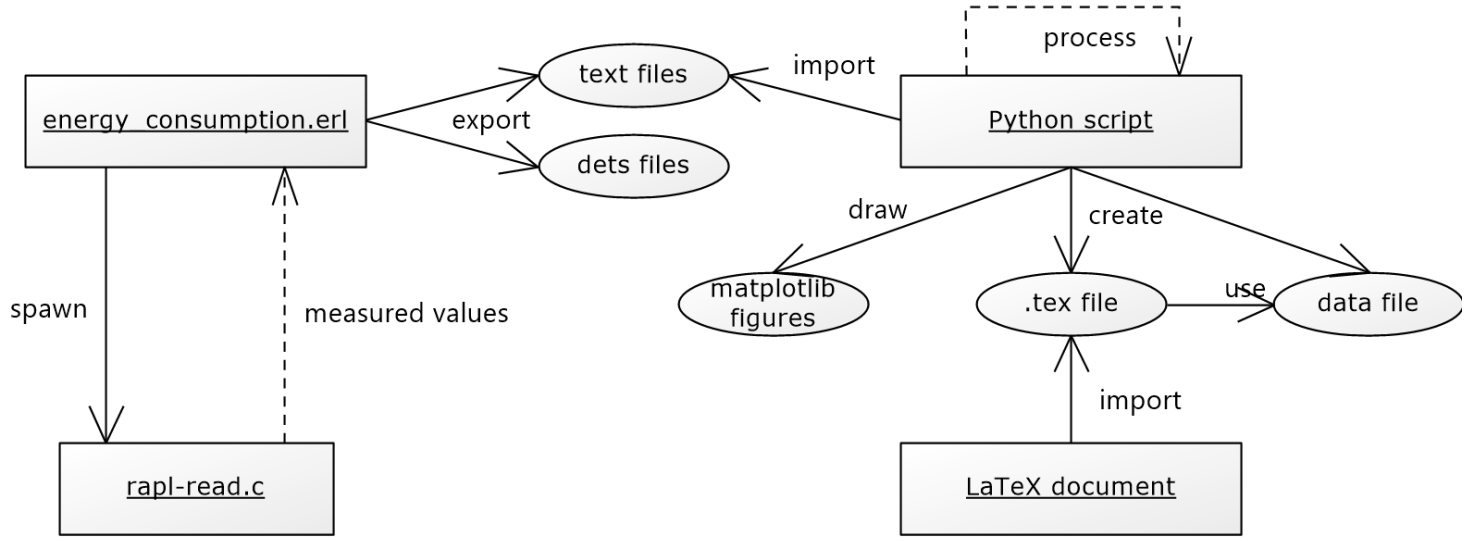
ERLANG FRAMEWORK

- C program to read RAPL values
- Erlang module to run measured functions
  - Communication via ports
  - Runs the functions
  - Tells the C program when to measure
- Python GUI
  - Using TkInter
  - Setup measurement
  - Helps with organizing the measurements
  - Visualizing results



Processing Data

- Methodology
  - Everything is measured 10 times
  - Discarding minimal and maximal values
  - Calculate average energy consumed
  - Also measuring runtime
- Visualization
  - Python – matplotlib



Higher order functions

- Parameters or the result is a function
- Integral part of functional programming languages
- Examples
  - map
  - filter
  - fold
- We wanted to find the cost of passing functions as a parameter

HOFs - Measurements

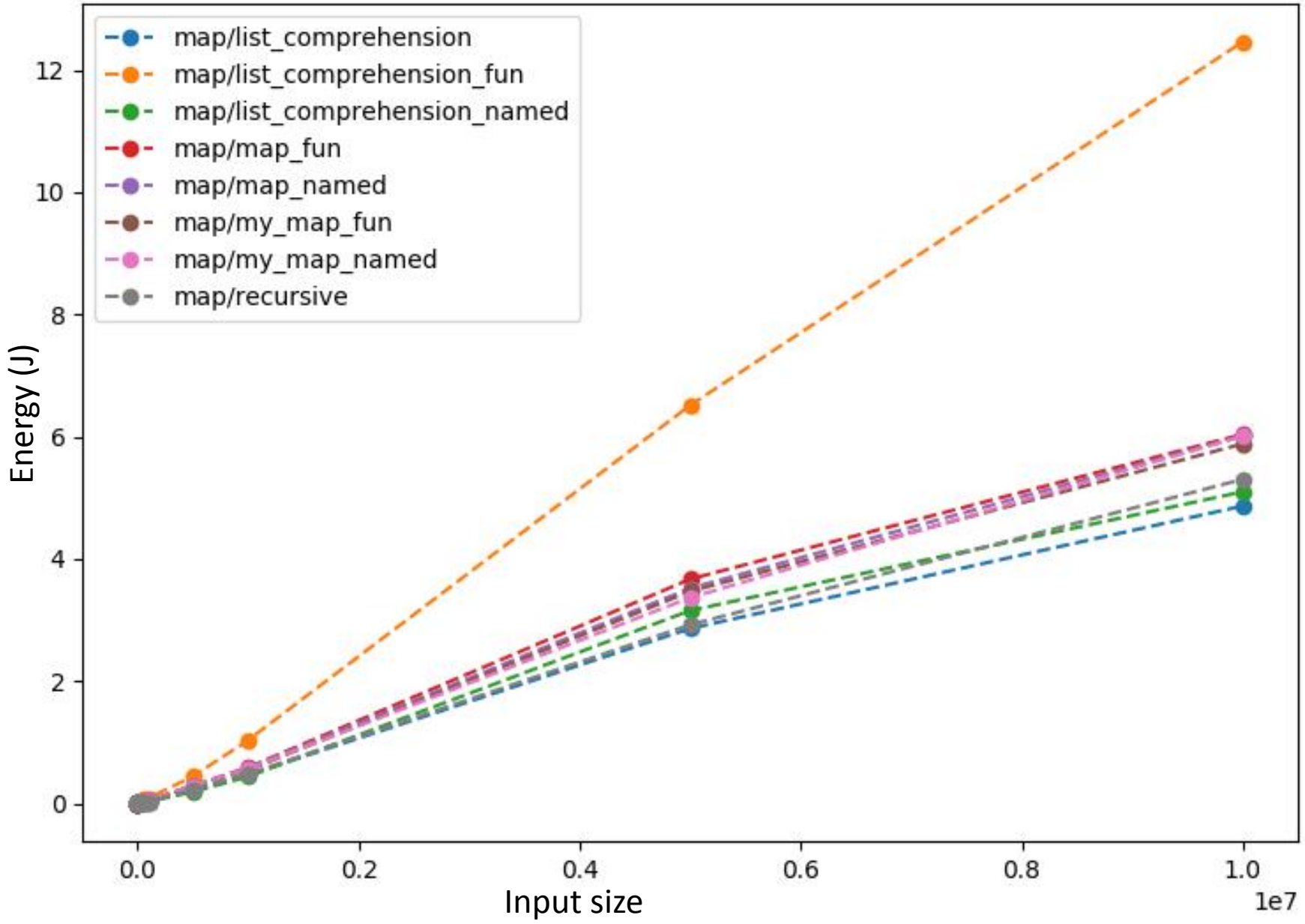
- Passing named or unnamed function
- Using HOF or list comprehension or recursion
- Implementing our own higher order functions
- Measured:
  - map
  - filter
  - map ∘ filter

```
lists:map(fun increase/1, L).
```

```
lists:filter(fun even/1, L).
```

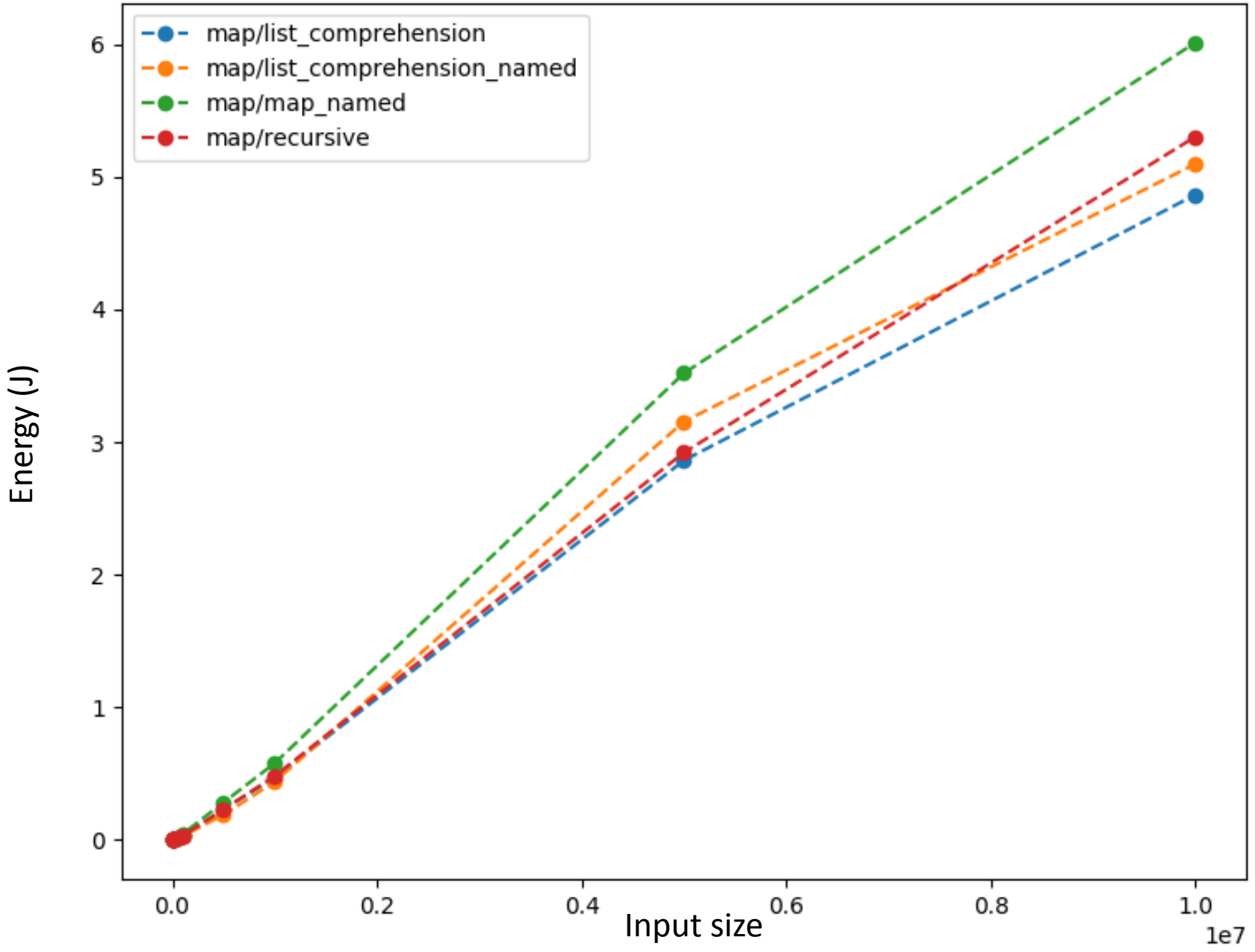
```
lists:map(fun increase/1, lists:filter(fun even/1, L)).
```

MAP - RESULTS

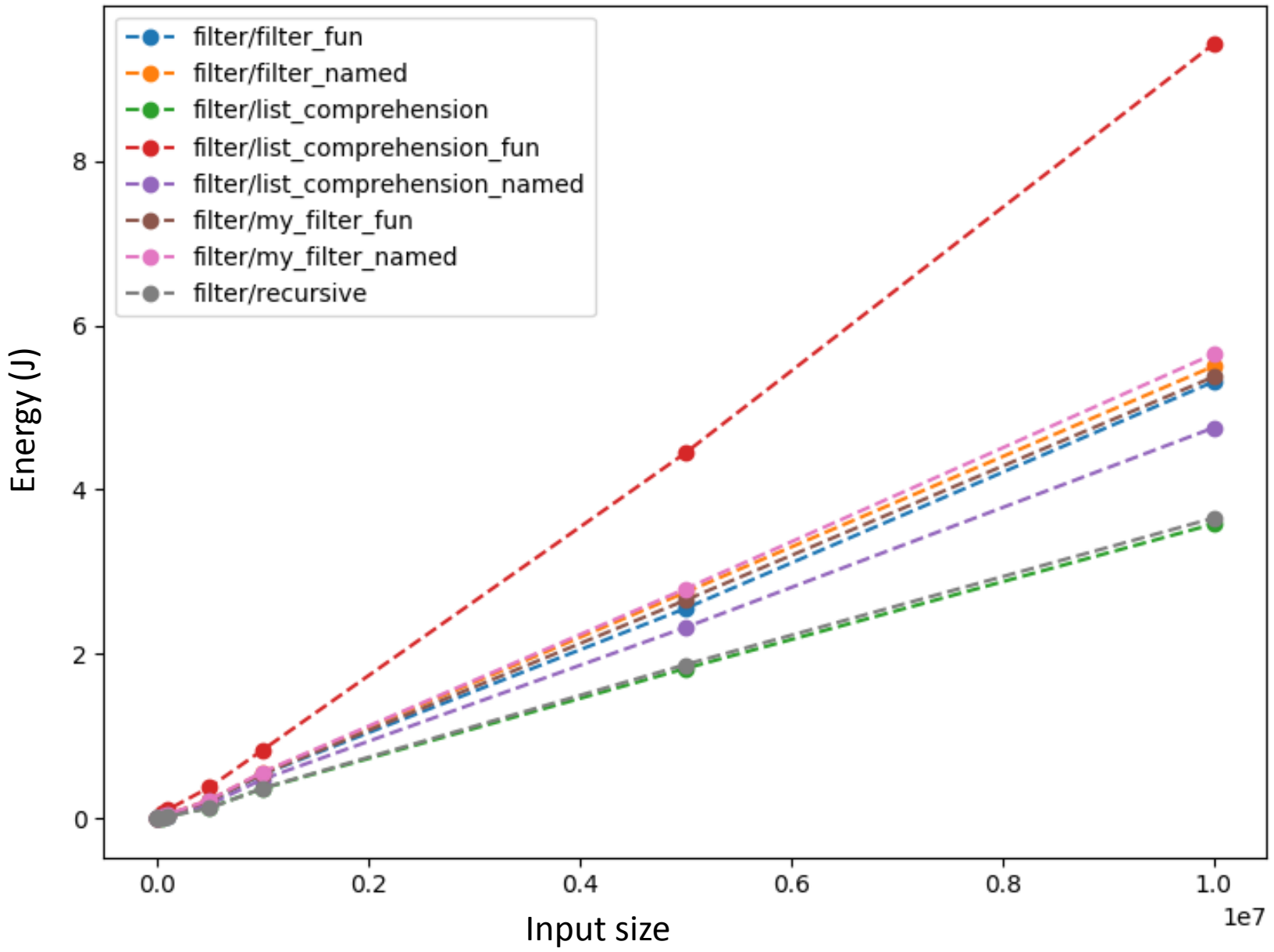


- Best:
  - List comprehension
  - Recursion
- Worst:
  - List comprehension with unnamed function
  - All map implementations
- Ranking is same based on energy and time

MAP - RESULTS

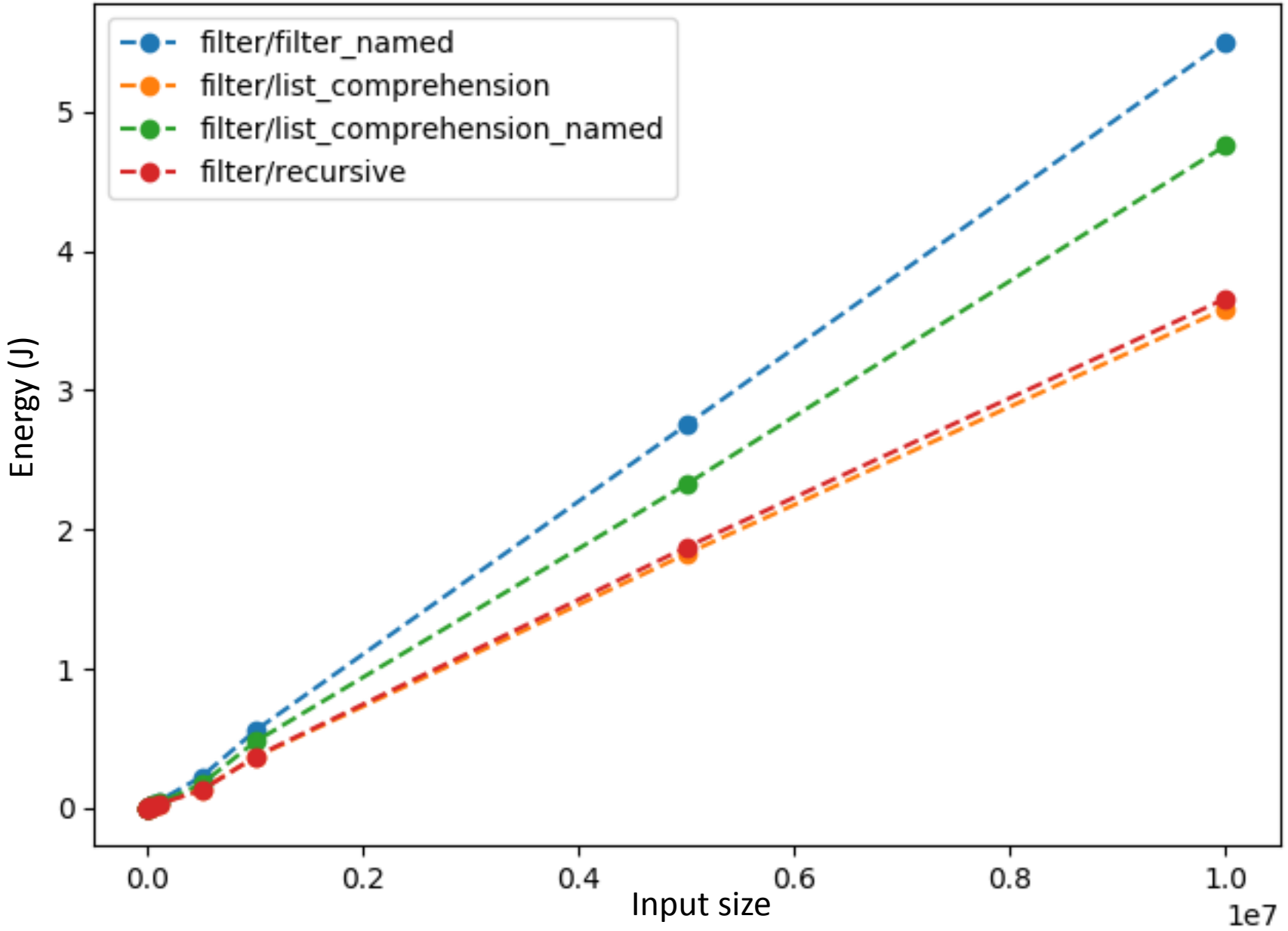

map/list_comprehension
map/list_comprehension_named
map/map_named
map/recursive
Energy (J)
0
1
2
3
4
5
6
0.0
0.2
0.4
0.6
0.8
1.0
1e7
Input size

FILTER - RESULTS

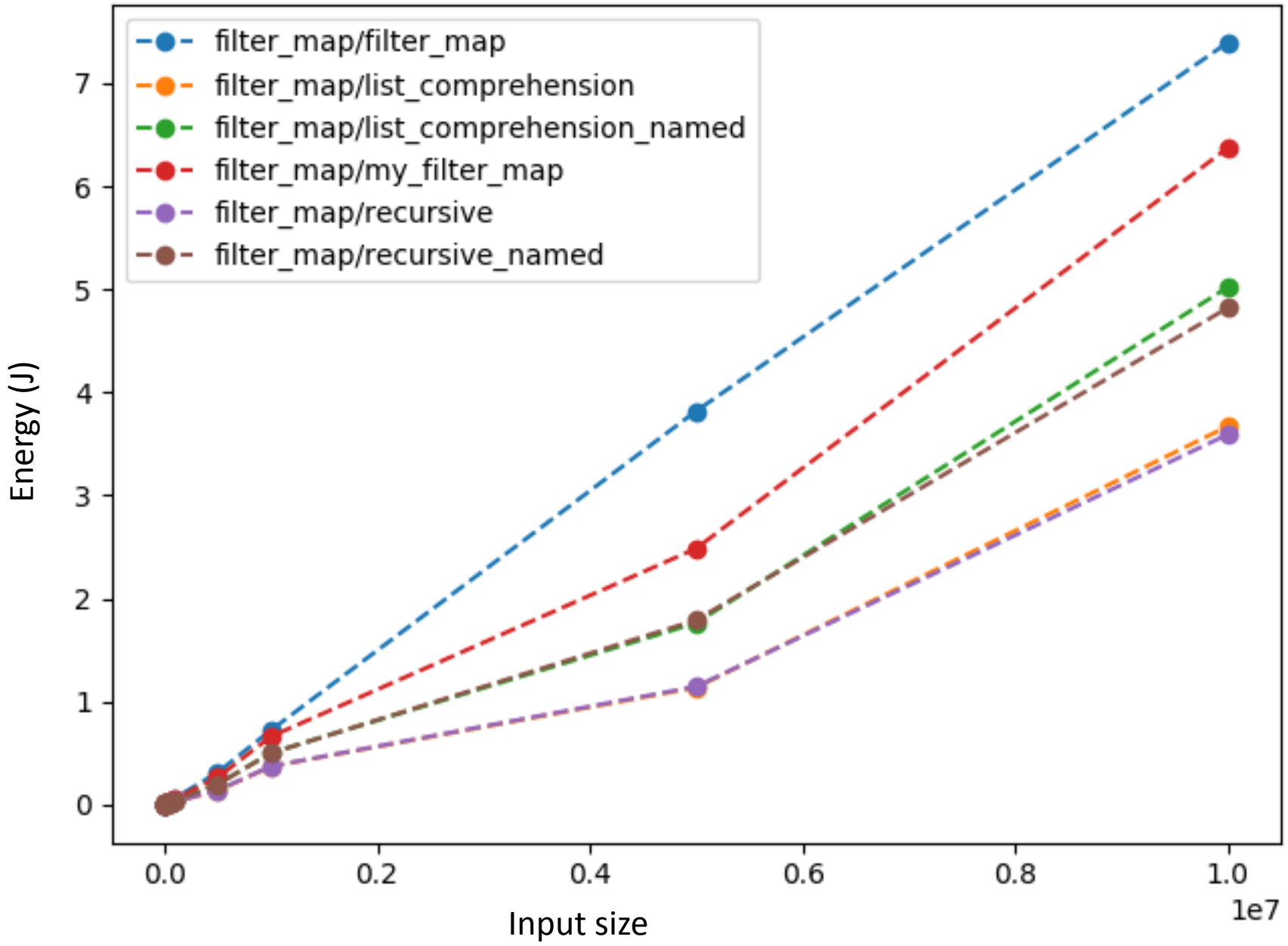


- Best:
  - List comprehension
  - Recursion
- Worst:
  - List comprehension with unnamed function
  - All map implementations
- Ranking is same based on energy and time
- Same trends as with map

FILTER - RESULTS



MAP ∘ FILTER - RESULTS



- Best:
  - List comprehension
  - Recursion
- Worst:
  - Filter then map
  - Own HOF implementation
- Ranking is same based on energy and time

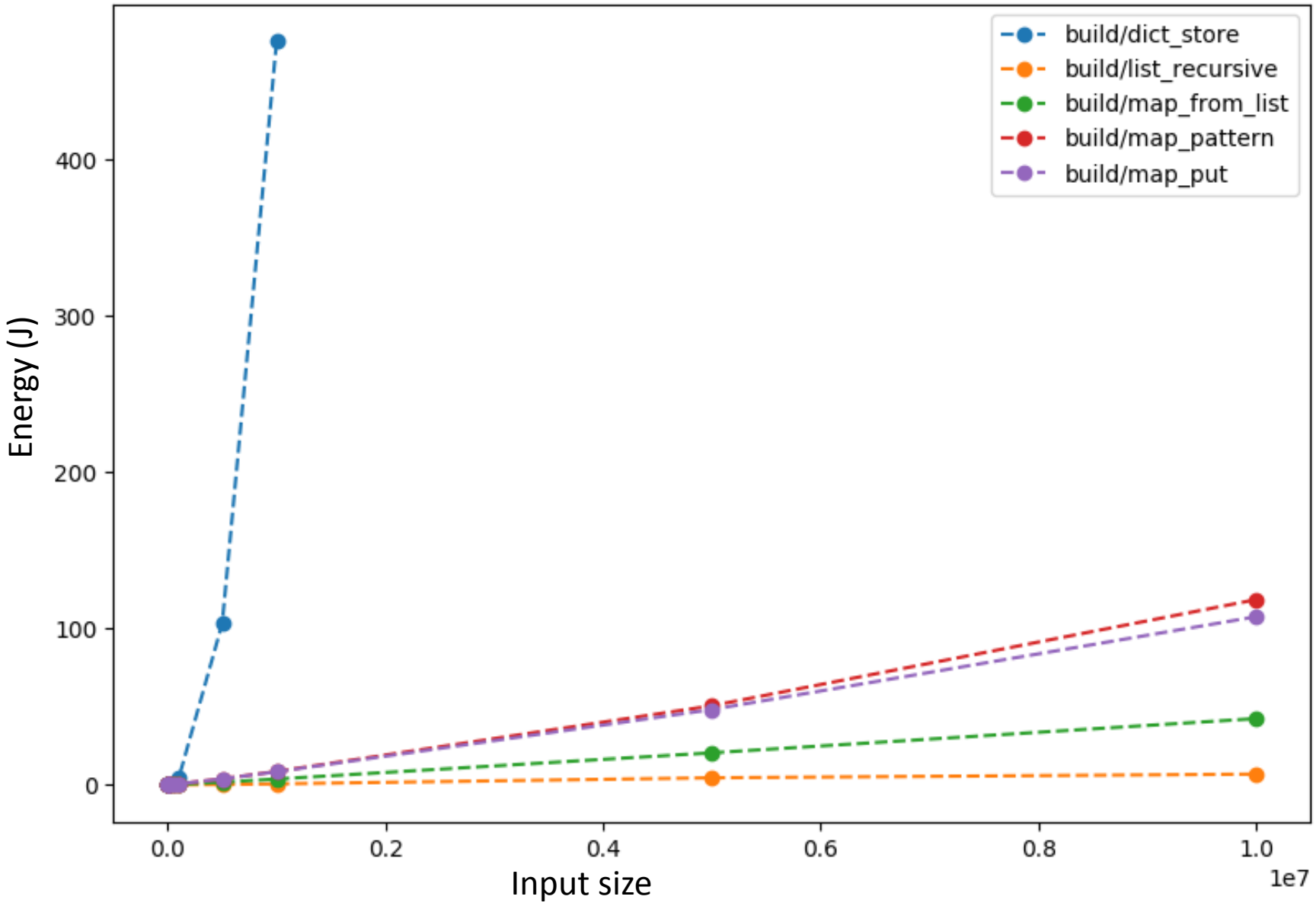
HOFs - Findings

- Refactoring higher order function calls to a simple recursive function or list comprehension reduces energy consumption and runtime
- Eliminating the HOF is not enough, we need to eliminate the called function as well to minimize energy consumption.
- This is consistent with previous findings (N-queens)

Data structures

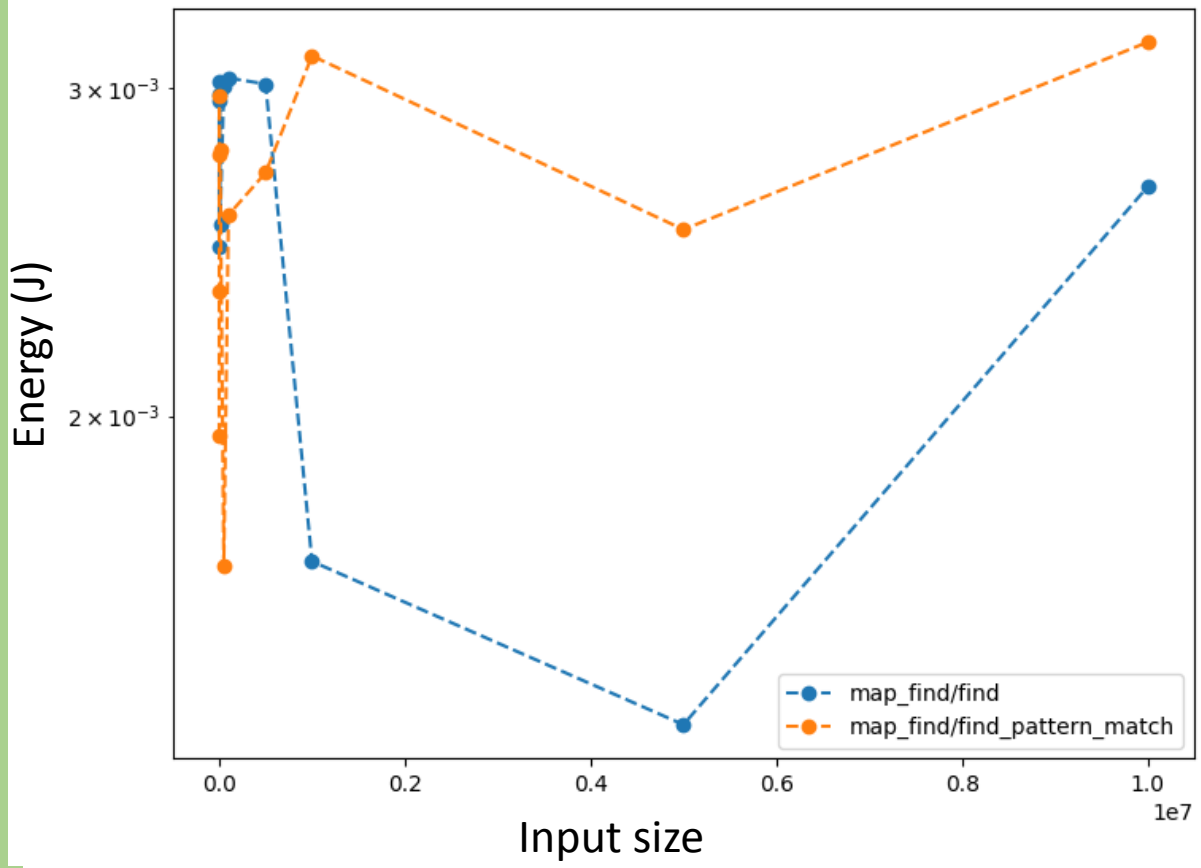
- Investigating different ways to store key-value pairs
  - list of tuples
  - map
  - dict
- Measuring different operations on those data structures
  - find
  - delete
  - update
  - building the data structure

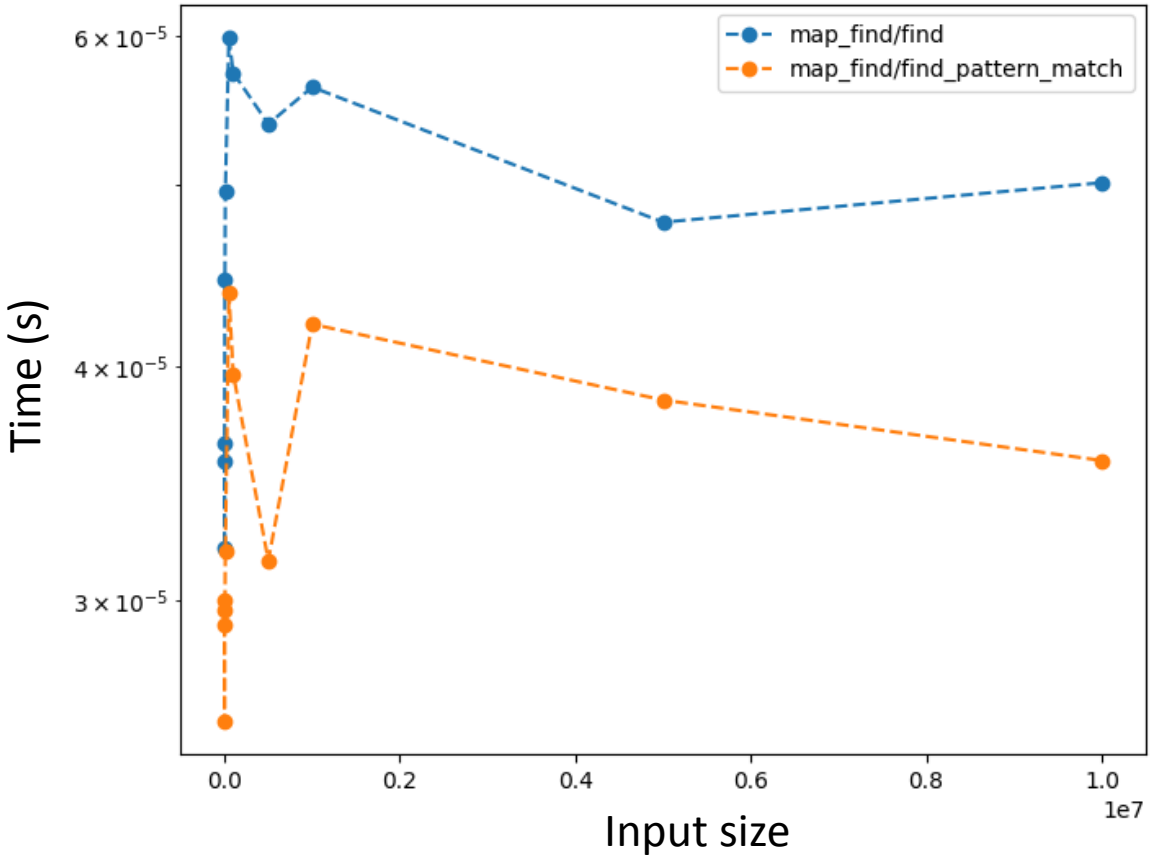
BUILDING - RESULTS



- Dicts are really expensive to build
- Lists are the cheapest
- Building a map from a list of tuples is cheaper than building the map directly

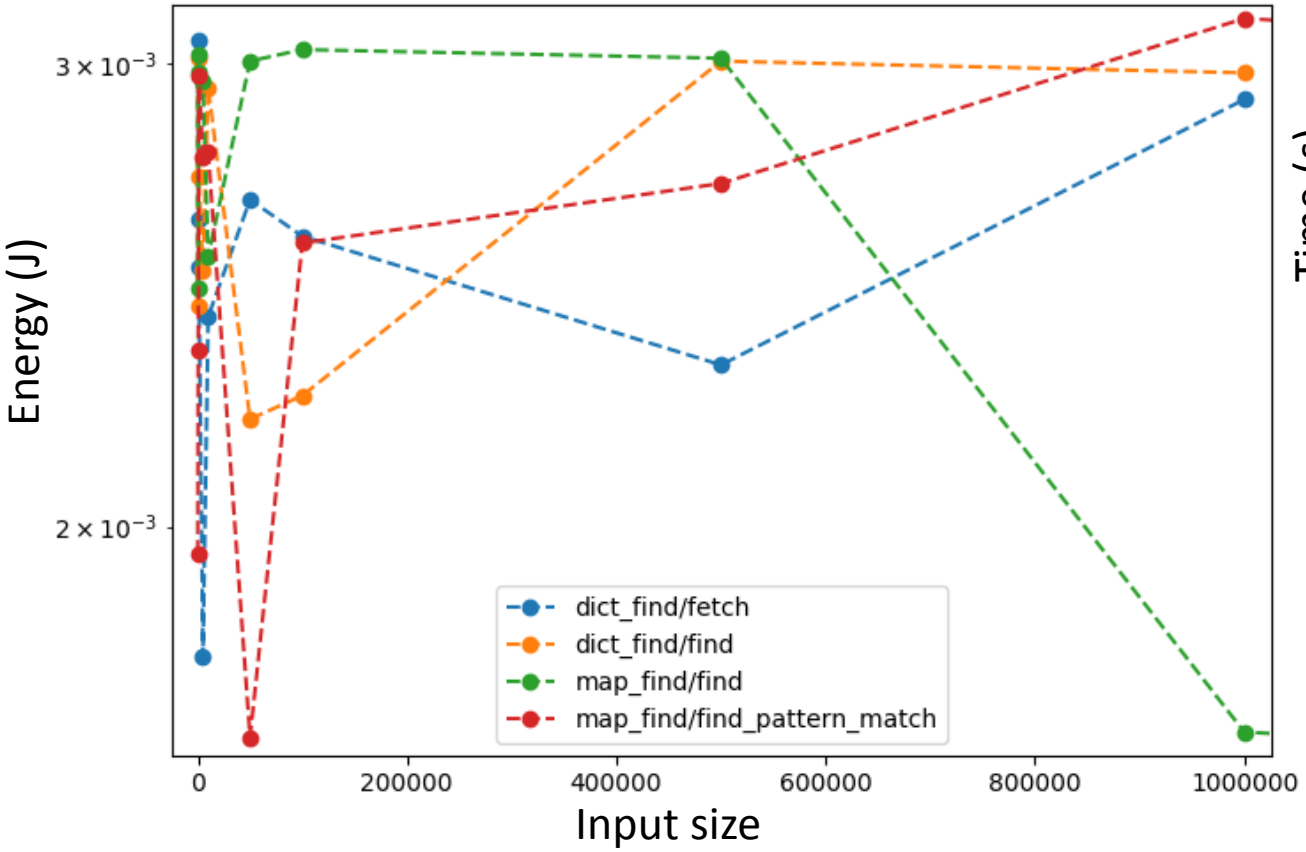
FIND - RESULTS

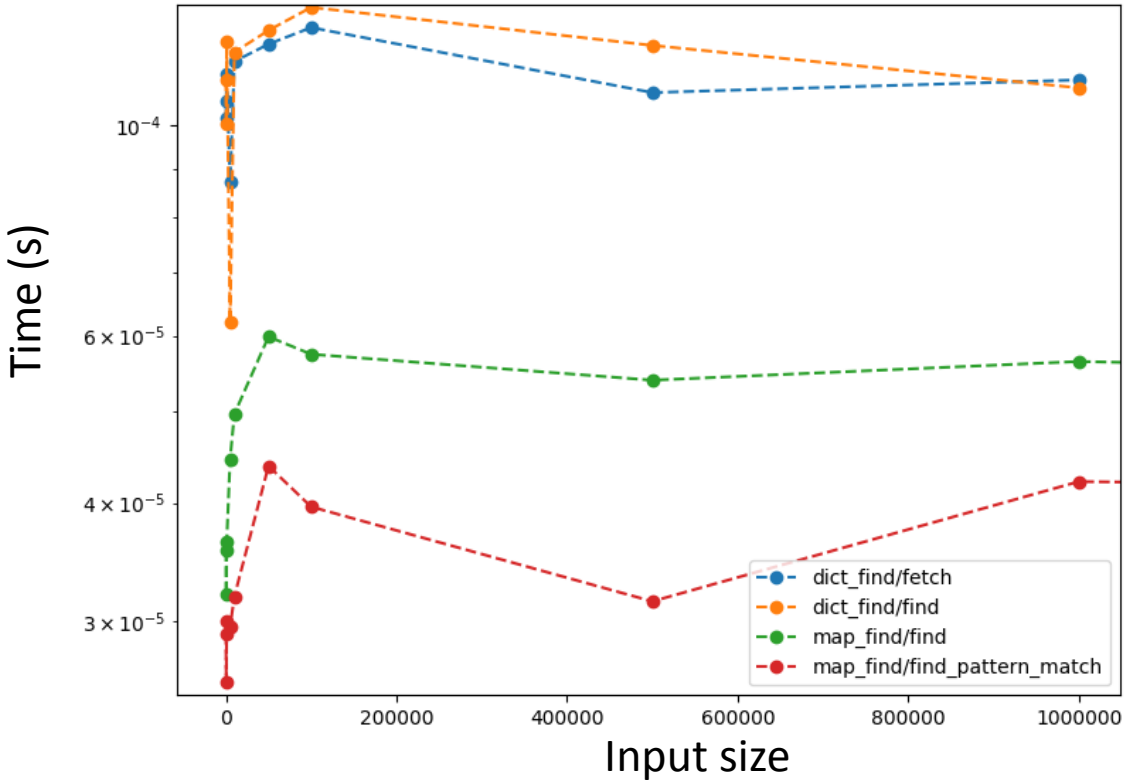




- Finding with pattern matching is faster, but consumes more energy
- Using the find function consumes the least energy but is slower

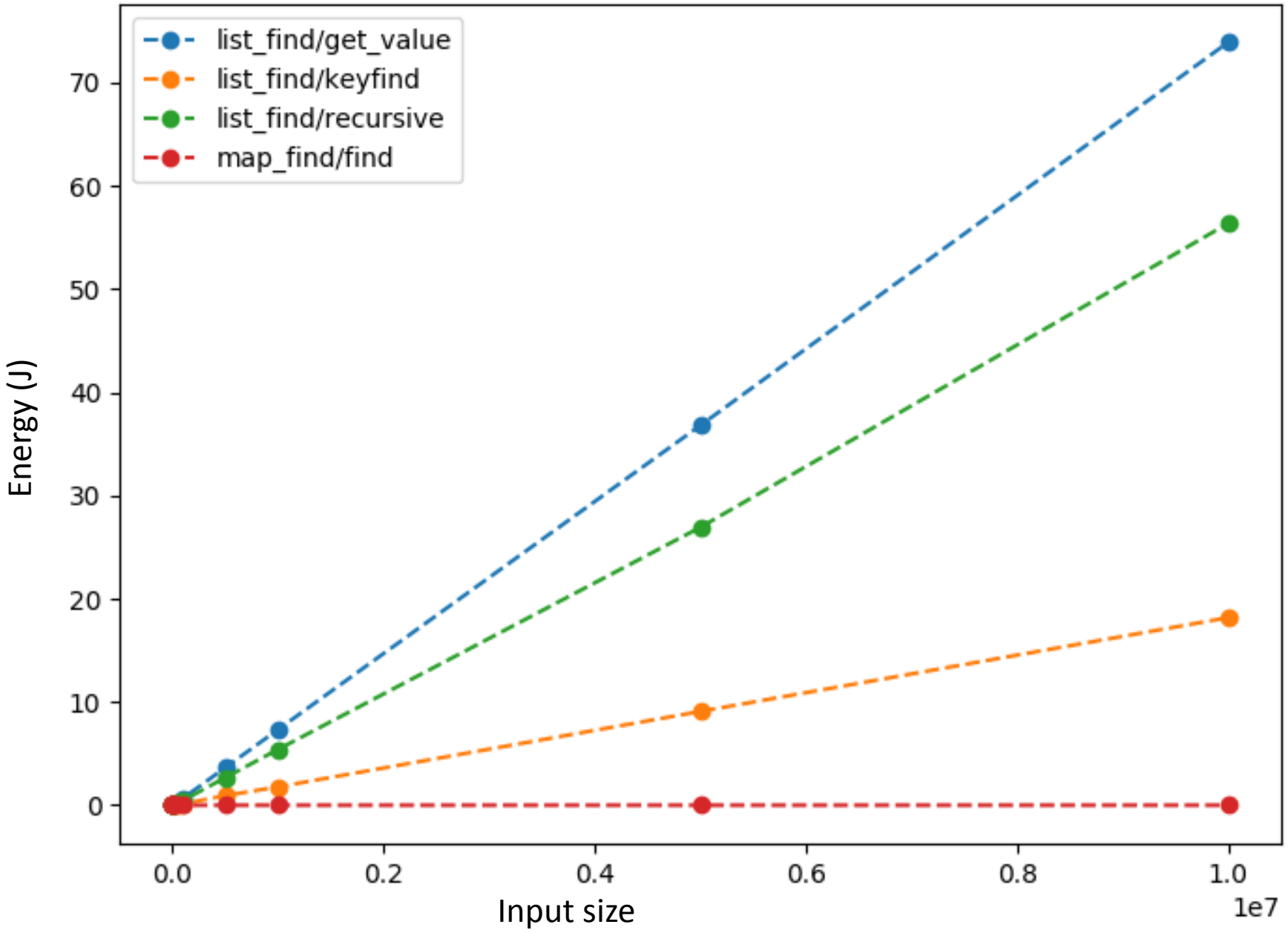
FIND - RESULTS





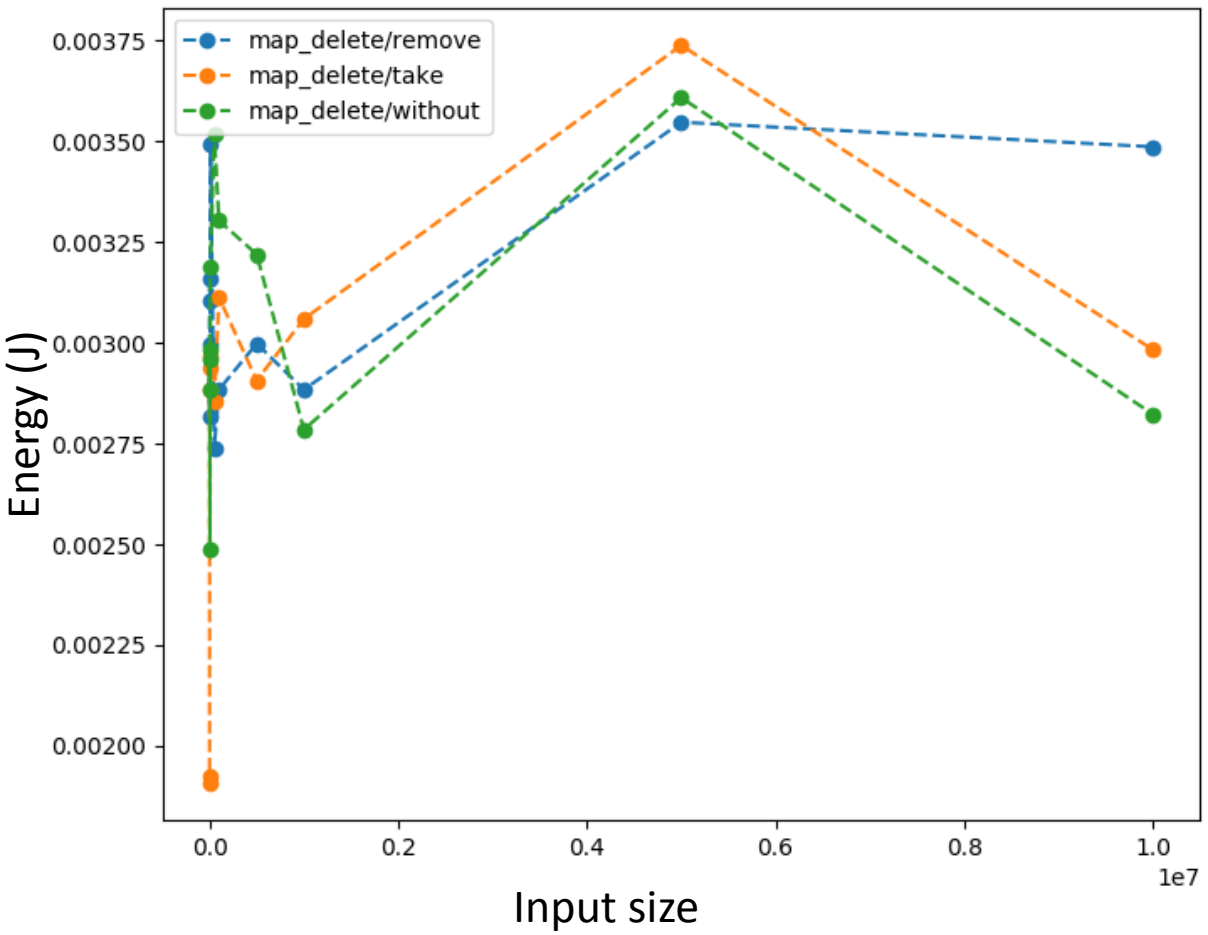
- Finding in a dict is similar to finding in a map,
- Maps are quicker
- Energy consumption may be better for maps (cannot construct large enough dicts)

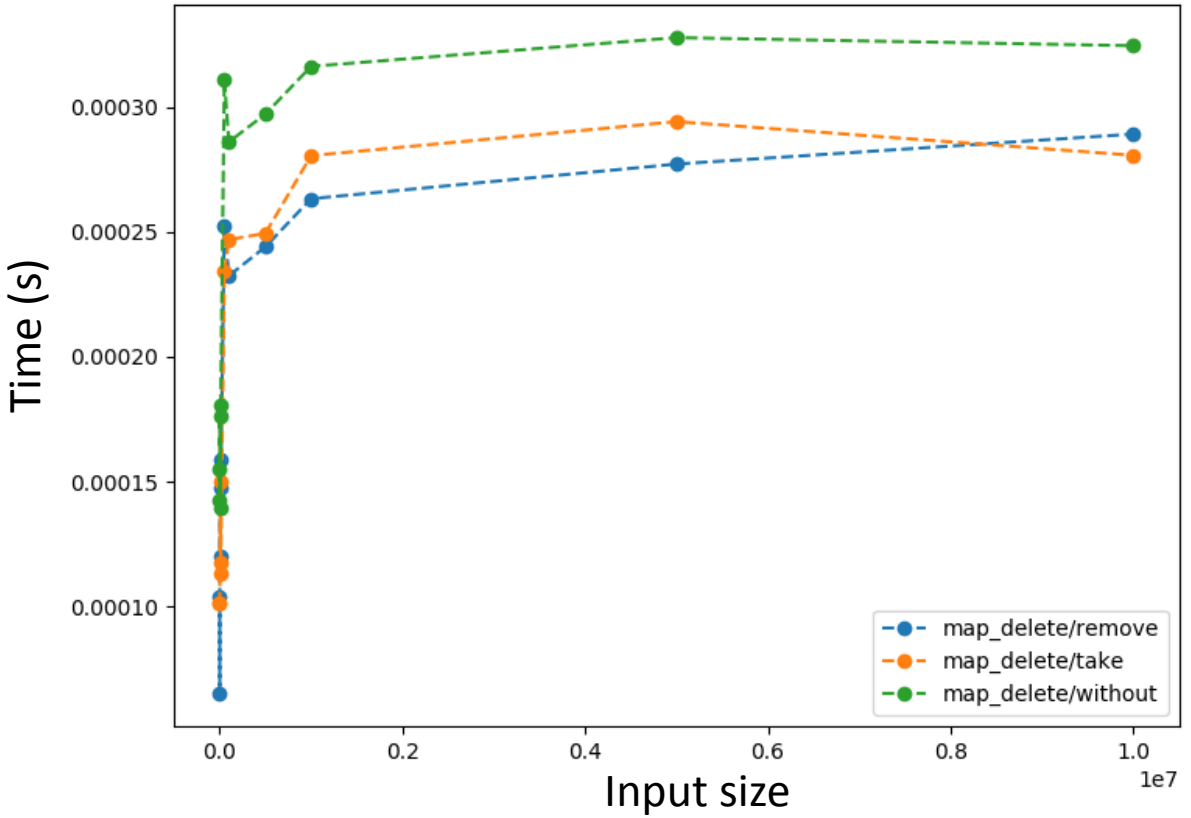
FIND - RESULTS



- proplists:get_value is inefficient, even a simple recursive find is better
- lists:keyfind is better
- Compared to maps all are worse (expected, because of complexity differences)

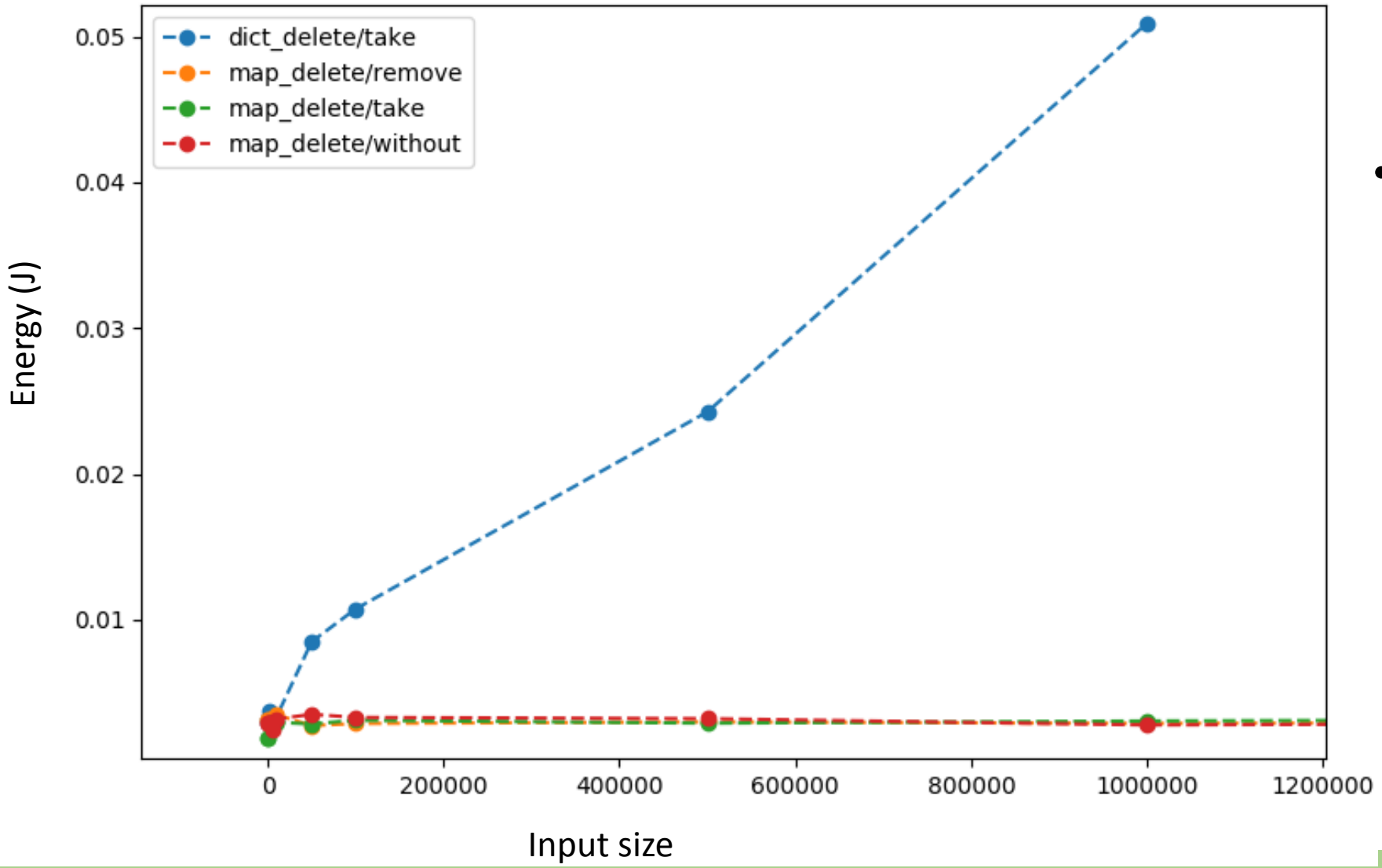
DELETE - RESULTS





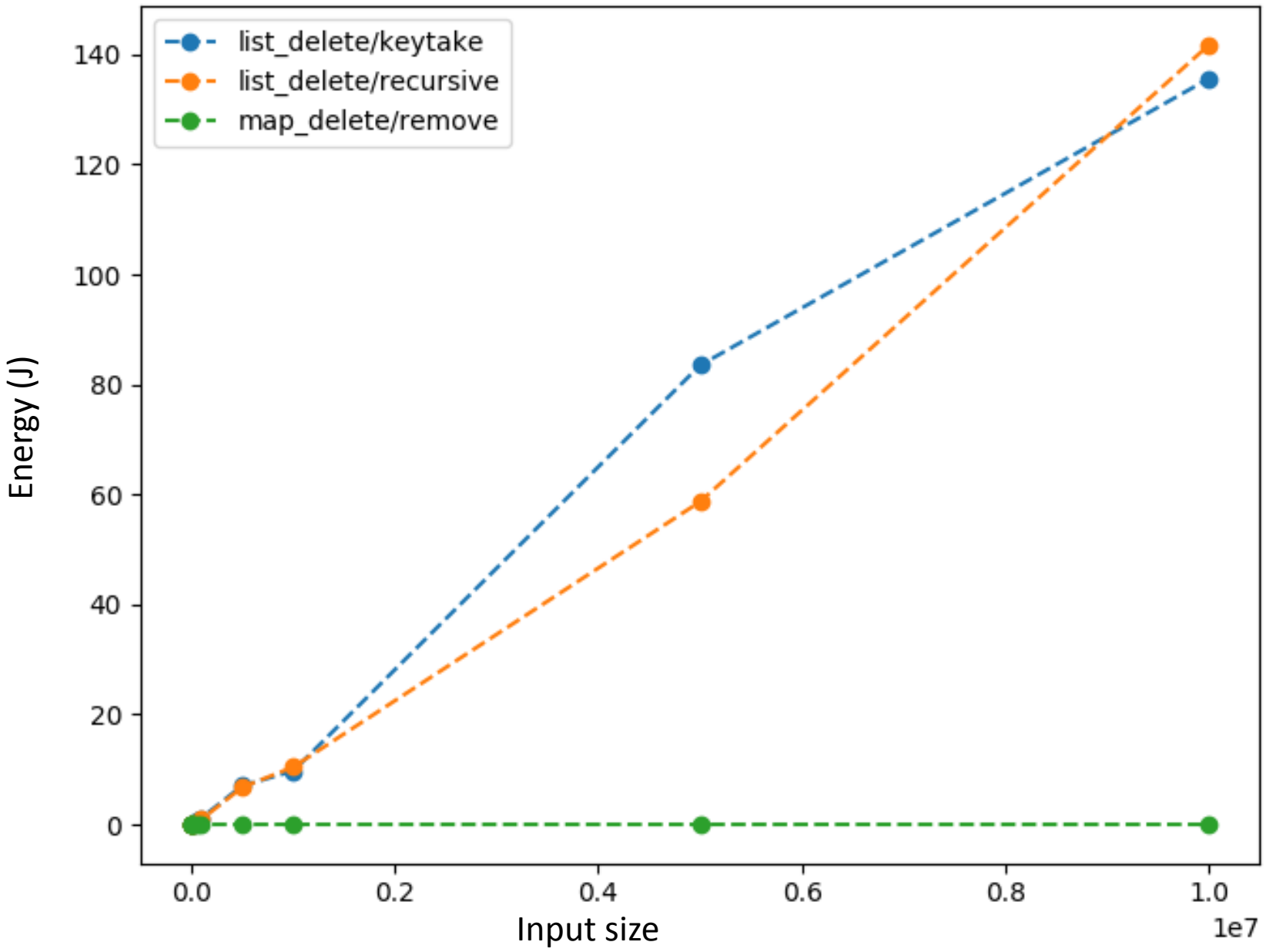
- Deleting with maps:without is the slowest, but consumes the least energy

DELETE - RESULTS



- Deleting from a dict is worse than deleting from a map
- Complexity difference

DELETE - RESULTS



- Deleting from list is much worse than all previous examples
- Complexity difference
- No conclusive difference between the two ways to delete

UPDATE - RESULTS

- Results for updating an element in out dictionaries show the same patterns as deleting an element.
- Maps are the most efficient
- Dicts are in the middle
- Lists perform the worst

DATA STRUCTURES - FINDINGS

- Most of the differences between the data structures can be attributed to different representations
- Interesting findings are:
  - We should use lists:keyfind instead of proplists:get_value
  - With maps the quickest solution is not always the more efficient
    - More investigation is needed, because measured values are so small
- Refactoring may be possible between different implementations of functions on the same data structure
- Based on the cost to build every data structure we may be able to decide how many of each operation is needed until it is worth building a different data structure

Conclusions

- Tool for measuring energy consumption
- Higher order functions have an overhead
  - It may be eliminated by rewriting the calls to higher order functions with recursion or list comprehensions
  - Function calls should also be eliminated
- Erlang built-in functions on the same data structures can have significant differences in performance

FUTURE WORK

- Investigating the cost of spawning processes and sending messages
- Measuring different algorithmic skeletons for parallelization
- Creating refactorings based on our findings using RefactorErl

THANK YOU FOR YOUR ATTENTION