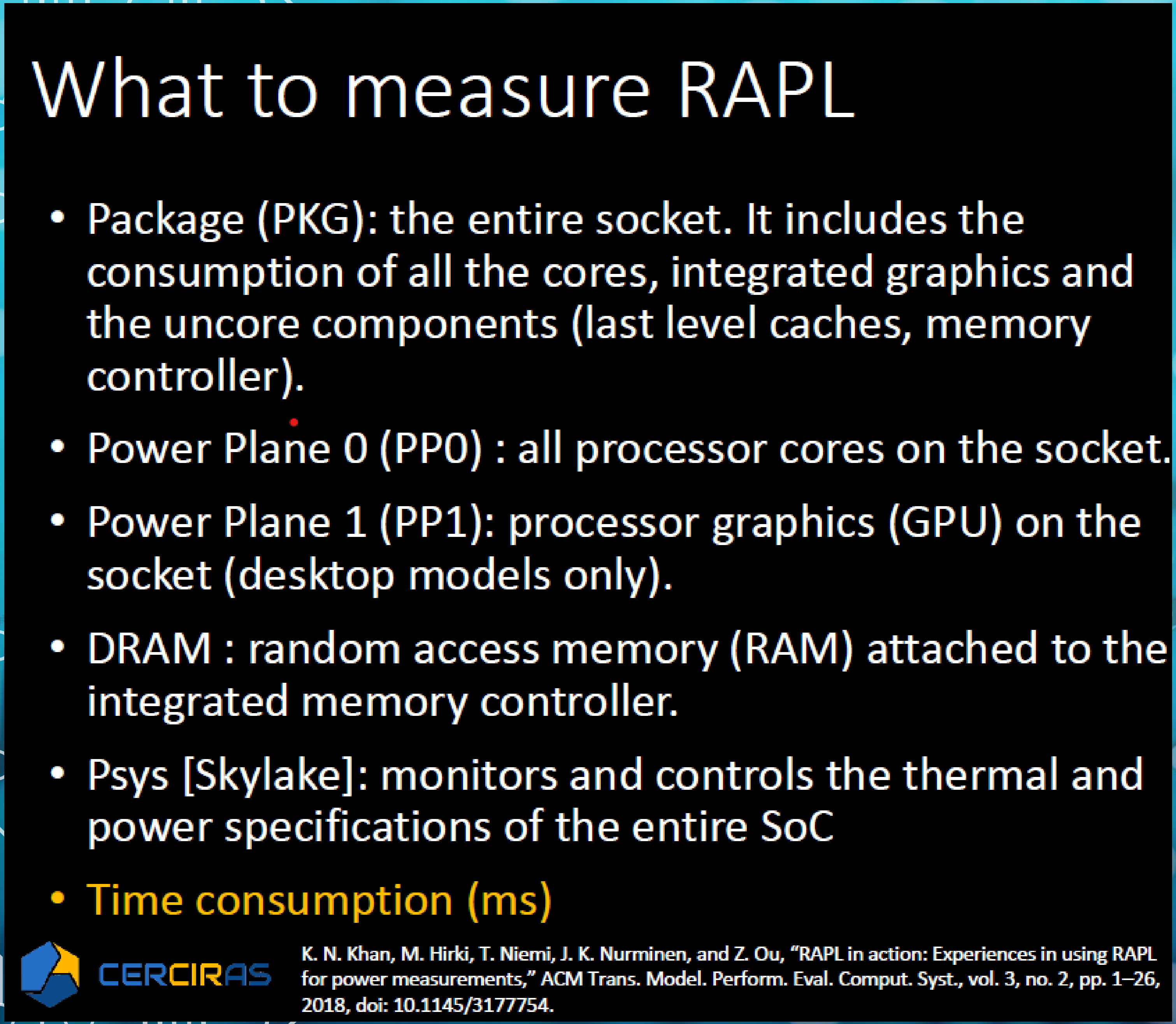
BUBBLE SORT ALGORITHM

RAPL PERFORMANCE TEST



What to measure RAPL

- Package (PKG): the entire socket. It includes the consumption of all the cores, integrated graphics and the uncore components (last level caches, memory controller).
- Power Plane 0 (PP0) : all processor cores on the socket.
- Power Plane 1 (PP1): processor graphics (GPU) on the socket (desktop models only).
- DRAM : random access memory (RAM) attached to the integrated memory controller.
- Psys [Skylake]: monitors and controls the thermal and power specifications of the entire SoC
- Time consumption (ms)

CERCIRAS

K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in action: Experiences in using RAPL for power measurements," ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 3, no. 2, pp. 1–26, 2018, doi: 10.1145/3177754.

WHAT RAPL MEASURES WE LOOK FOR?

- For this test we are looked for three measures:
  - Package (PKG)
  - Power Plane 0 (PP0)
  - Time Consumption (ms)

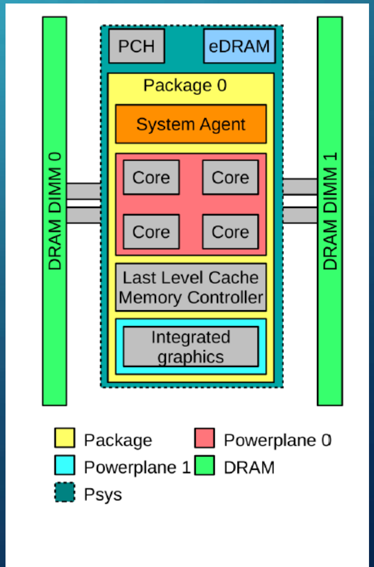


Fig. 1. Power domains supported by RAPL.

EQUIPMENT CHIPSET

- ThinkPad x260, Hardware: Intel(R) Core(TM) i5-6200U CPU
- @ 2.30GHz, Architecture: x86_64, CPU(s): 4, Memory: 7.6Gi

LANGUAGE

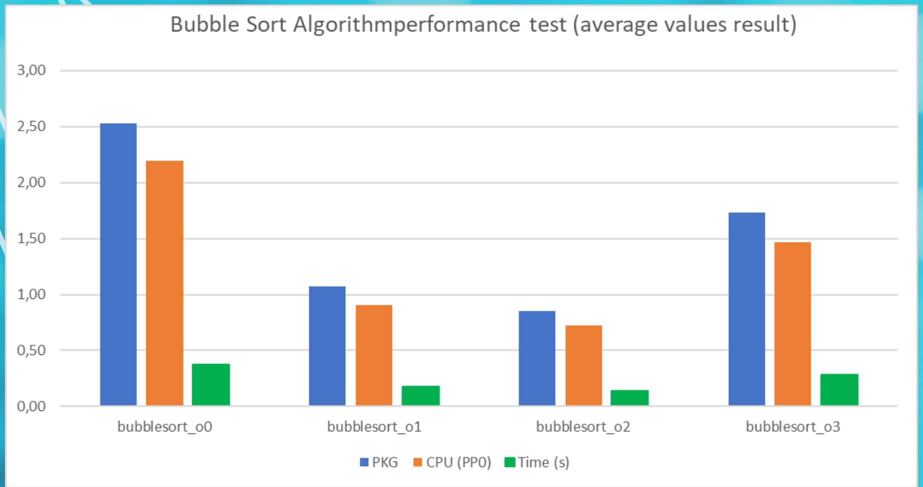
- C

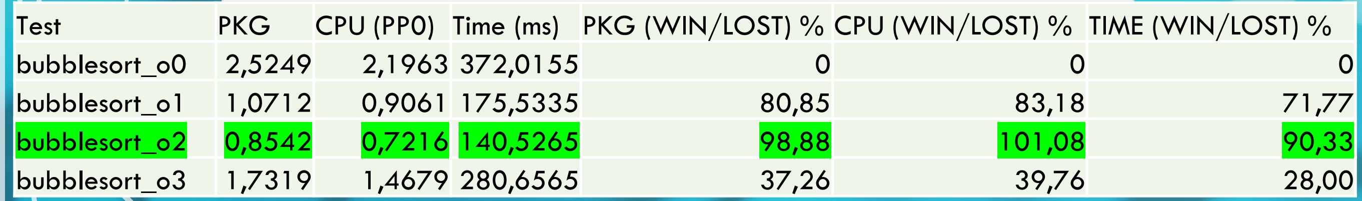
COMPILER

- gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0

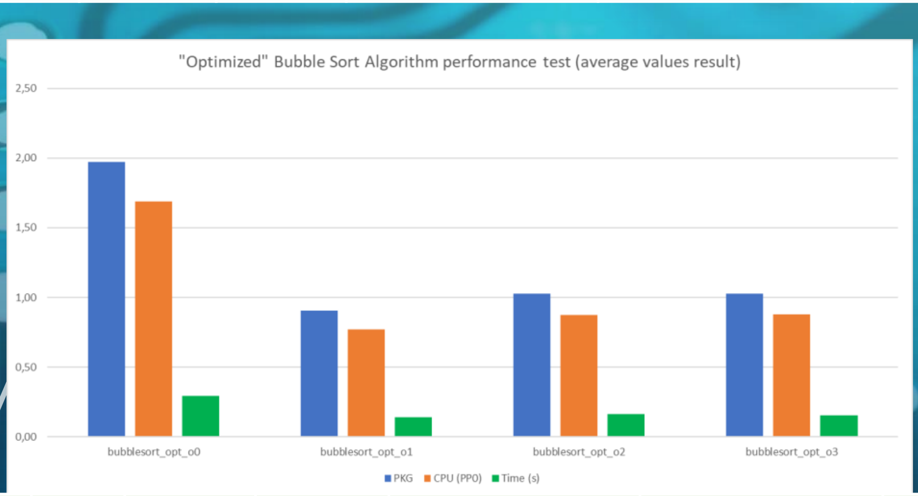
STUDY METHODOLOGY

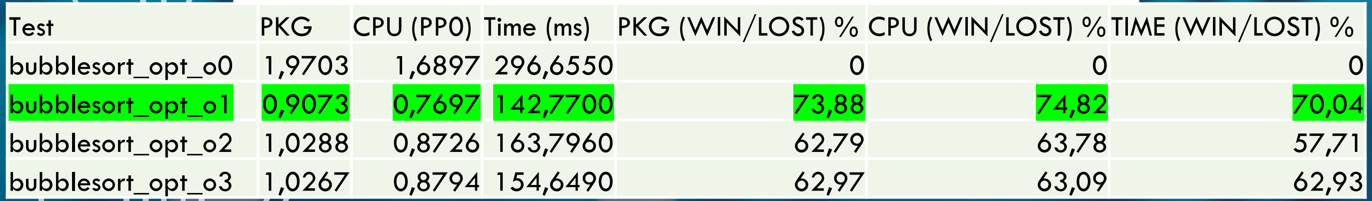
- For each algorithm the RAPL Makefile was configured to apply the following settings:
- Apply 4 compiler optimization flags
  - O0 - none
  - O1
  - O2
  - O3
- To run code 400 times per flag
- Take measurements of each run
- Calculate average execution value per flag
- Bubble sort algorithms courtesy of
  - https://www.programiz.com/dsa/bubble-sort





Test	PKG	CPU (PP0)	Time (ms)	PKG (WIN/LOST) %	CPU (WIN/LOST) %	TIME (WIN/LOST) %
bubblesort_o0	2,5249	2,1963	372,0155	0	0	0
bubblesort_o1	1,0712	0,9061	175,5335	80,85	83,18	71,77
bubblesort_o2	0,8542	0,7216	140,5265	98,88	101,08	90,33
bubblesort_o3	1,7319	1,4679	280,6565	37,26	39,76	28,00





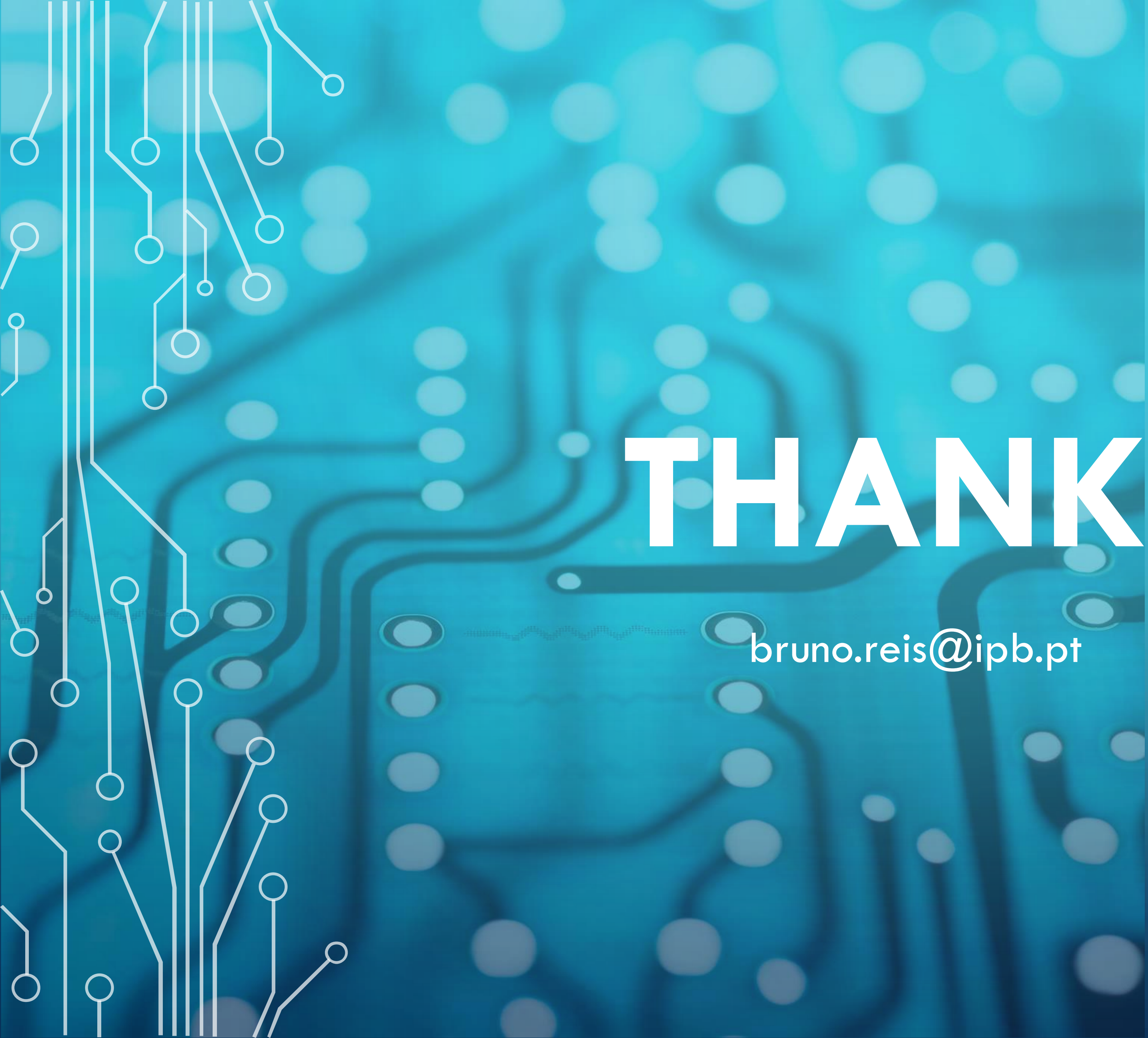
Test	PKG	CPU (PP0)	Time (ms)	PKG (WIN/LOST) %	CPU (WIN/LOST) %	TIME (WIN/LOST) %
bubblesort_opt_o0	1,9703	1,6897	296,6550	0	0	0
bubblesort_opt_o1	0,9073	0,7697	142,7700	73,88	74,82	70,04
bubblesort_opt_o2	1,0288	0,8726	163,7960	62,79	63,78	57,71
bubblesort_opt_o3	1,0267	0,8794	154,6490	62,97	63,09	62,93

RESULTS

- Comparing the results visually by the graph and by the data in the table, we can conclude:
- The optimized bubble sort algorithm, in general, is more efficient on average
- The optimized bubble sort algorithm is more efficient when no compiler optimization is applied
- That the optimization flag that maximizes each algorithms efficiency is different
- That the unoptimized bubble sort algorithm is the most efficient, when the correct flag is applied

CONCLUSION

- The choice/implementation of the algorithm influences its energy efficiency
- Different approaches to the same problem also have different compiler flags that maximize your energy efficiency
- It is mandatory to make a good evaluation of the flag to be used when compiling our programs, as this has a great impact on the energy efficiency of our programs, by our example we can have gains of more than 80%
- The most far-fetched solution is not always the most efficient, either at an energy level or at a general level, namely, execution time



THANKS

bruno.reis@ipb.pt