### The Compiler Optimization Problem

- Diversity of today's architectures including many platforms and compiler frameworks
- Two major classes: (i) The Selection Problem (ii) The Phase-ordering Problem
- Having Different goals in different domains (Power, Performance, Area, etc.)
- Many different optimization flags working with/versus each other within different levels
  - Around 100 opt in LLVM, 200 in GCC and 75 in ICC.

### (I) Statistical Analysis of Compiler Optimization Effect: Design Space Exploration (DSE) Approach [1,2]

- Two level SW/HW co-design approach:
  - Providing custom VLIW architecture & Applying compiler options/statistical analyses
  - Integrating Roofline Model to map upper-bound for memory and computation
- Testing on LLVM and VLIW compiler/simulator (VEX)

### (II) Selection of Best Compiler Optimizations Problem: Machine Learning Approach [3]

- Avoiding (i) manual hand-crafted and (ii) long process of iterative-compilation techniques
- Utilizing independent dynamic features to characterize applications using PIN tools
- Probabilistic Belief network utilizing Bayesian Networks to construct model per application:
  - INPUT: Application features plus top 20% speedup values corresponding to the use of compiler options
  - OUTPUT: Inference the best compiler optimizations to be applied to maximize performance


- Conventional approaches can not be useful for the phase-ordering problem as design space is huge, i.e. having repetitions, n opts and w as maximum length: \( \exp (n \times w) \)
- Classic predictive modeling needs fixed-length feature vector
- Necessity for adapting an immediate-speedup predictor given the current status of the code
- Defining two search policies: (i) Greedy DFS (ii) Exhaustive search within sub-nodes

### References


### Acknowledgements

This project is partially funded by the European Union’s H2020 grant FET-HPC ANTA2014-671023.