A Machine Learning Approach to Mapping Streaming Workloads to Dynamic Multicore Processors

- Paul-Jules Micolet
- Aaron Smith
- Christopher Dubach

University of Edinburgh, UK

Objective:

- Determine the optimal number of threads for a given application.
- Determine the best core composition that leads to optimal performance.

Problem Definition:

- Extract relevant static code features.
- Use machine learning technique to predict both parameters.

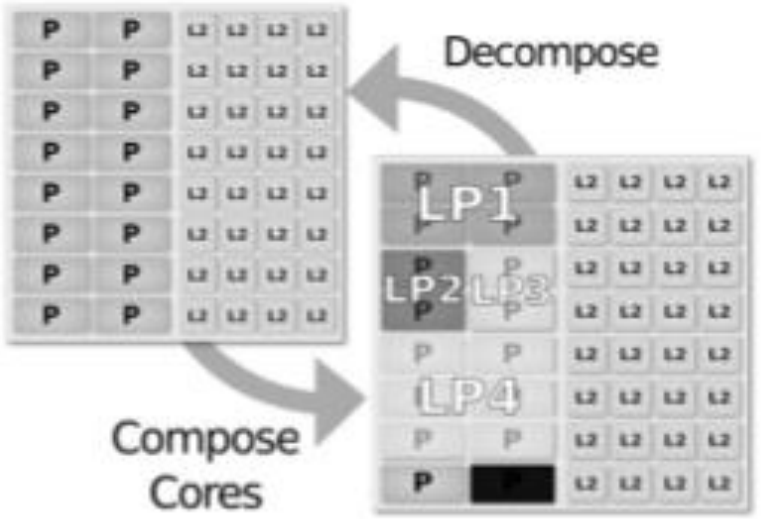
Approach in the Paper:

- Dynamic Multicore Processors.
- Importance of predicting the optimal number of threads and optimal core composition.
- Methodology to predict the optimal number of threads and optimal core composition.
- Design Space Exploration.
- Machine Learning Models.
- Results.

Dynamic Multi-Core Processors

- Homogeneous multi-core processors: Smaller cores of the same nature are placed in tiled architecture.
- Heterogeneous multi-core processors: Consist of cores that work at different levels of power and performance.
- Need for Dynamic multi-core processors: Many of the trade-offs between power and performance can't be changed after fabrication.
- Dynamic multi-core processors: Provide on-demand heterogeneity by fusing cores together.

Dynamic Multi-Core Processors



- Consists of logical cores formed by fusing multiple cores.
- Large sequential pieces of code requires ILP.
- ILP is achieved by fusing multiple cores together.
- Parallel workloads requires TLP.
- TLP is achieved by separating the cores.

Streaming Programming Languages:

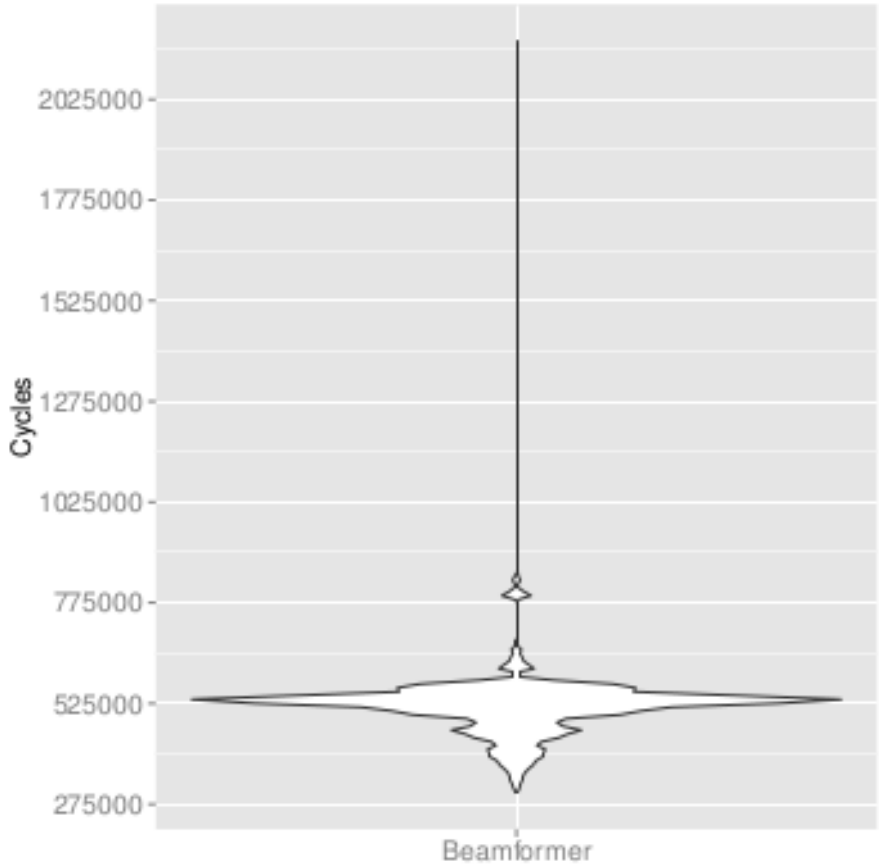
- Programming paradigm: Dataflow programming.
- Used for developing applications that deal with constant stream of data.
- Programs are described as directed graphs.
- Expose parallel and serial parts of the program to the compiler.
- Ideal for targeting multicore processors.

Streamit:

- Open source project maintained by a research group in CSAIL at MIT.
- **StreamIt** is a programming language and a compilation infrastructure.

Optimal Number of Threads and Core Composition

- Analysis of 32k points in design space.
- Each point is unique combination of number of threads and core composition.
- Wrong choice often leads to suboptimal performance.



DMP Processor Used

- Dynamic multi-core processor used is based on Explicit Data Graph Execution instruction set.
- Customizable cycle level simulator.
- 16 cores. 1 core for main thread and runtime management
- Each core having 16KB private L1 cache.
- 2 MB shared L2 cache.

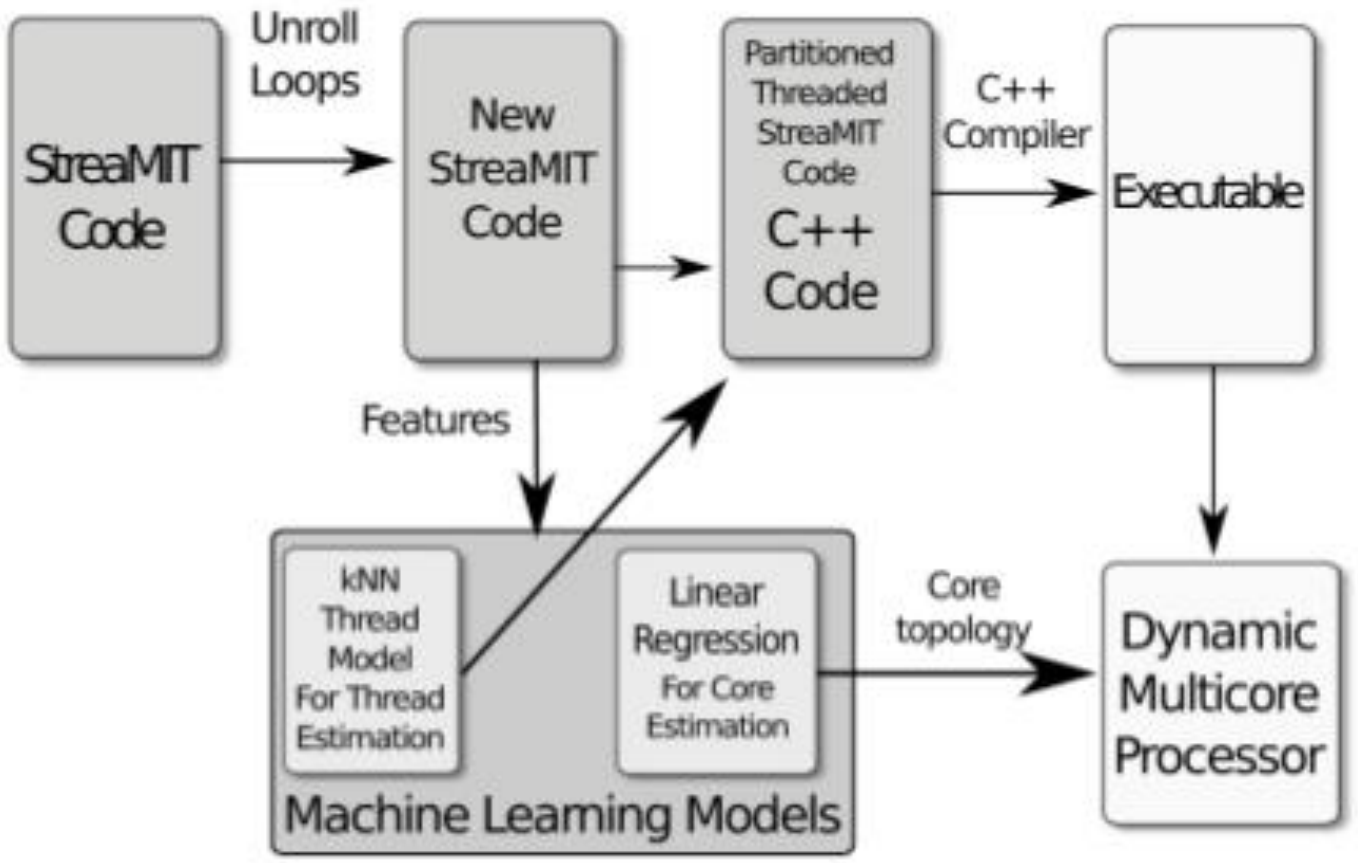
Streamit Benchmarks

- 15 Streamit benchmarks are taken from the official repository.

Design Space

- Number of Cores available: 1-15
- Number of Threads Available: 1-15
- Total Number of points in Design Space: 32,767
- Practically impossible to explore the entire design space.
- Sample Space: 1,316 points. (100 core composition per thread number).
- Best point found is at least within 5% of real best with 95% confidence.
- Motivation to use Machine Learning approach to determine the best thread and core combination.

Methodology

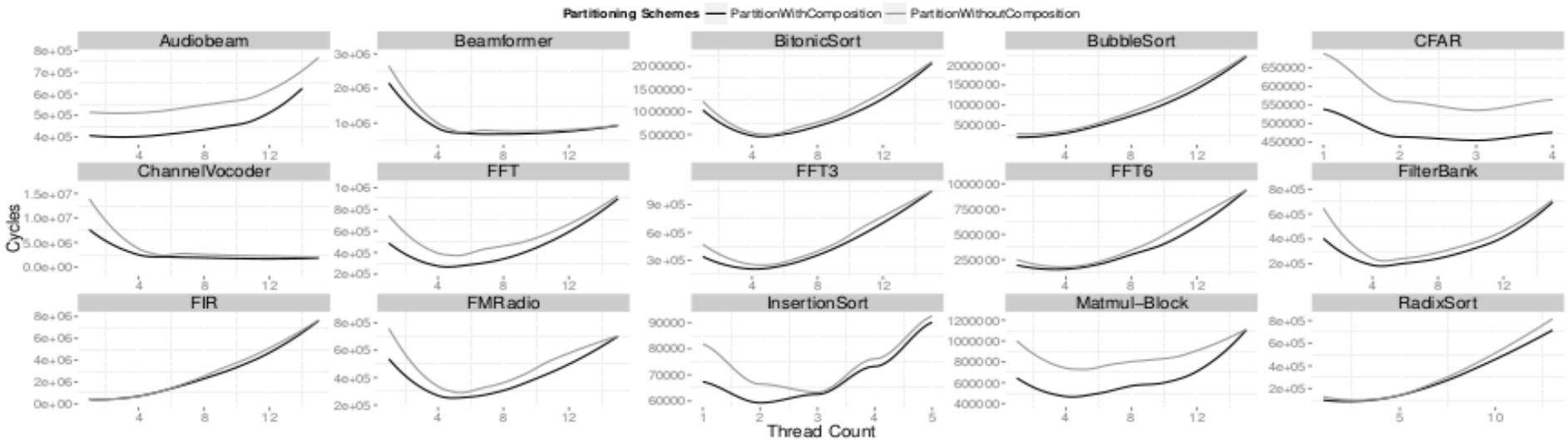


Impact of Thread Partitioning

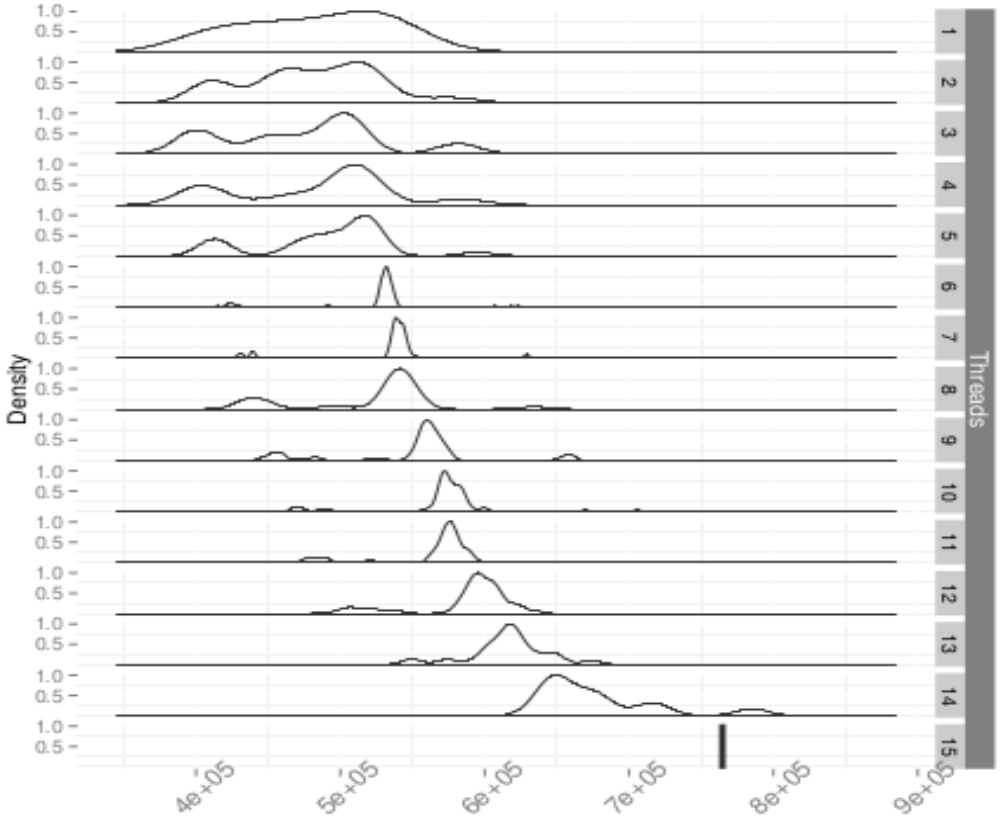
- Analysis of performance of a benchmark applications by varying the number of threads.
- Split the application into multiple threads with number of threads varying form 1-15.
- Core Composition: Without fusing and With fusing.
- Trend in performance is same for both hardware scenarios.

Impact of Thread Partitioning

- Regardless of the core composition, performance curves follow the same trend.
- First determine optimal number of threads and then core composition.

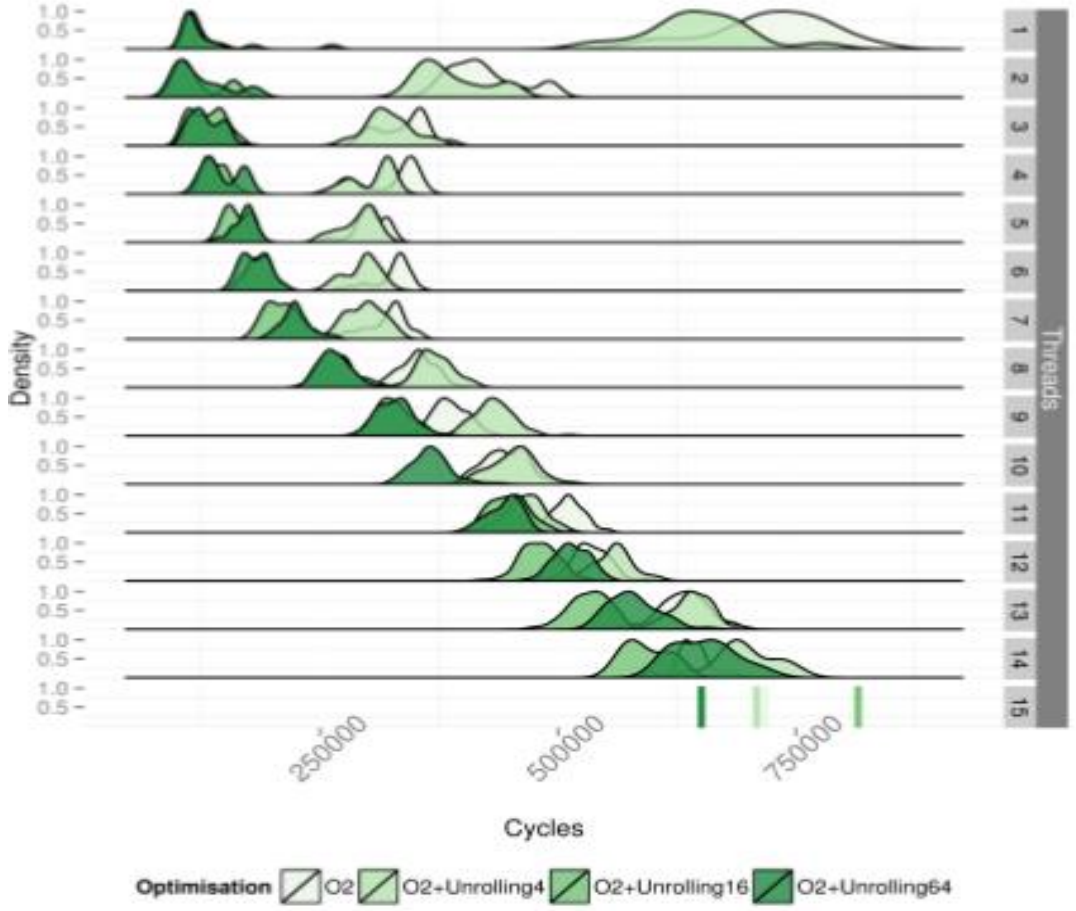


Impact of Core Composition



- Impact of composition is significant for application versions having 1-5 threads.
- Performance degrades as we increase the number of threads.

Impact of Unrolling the Loops



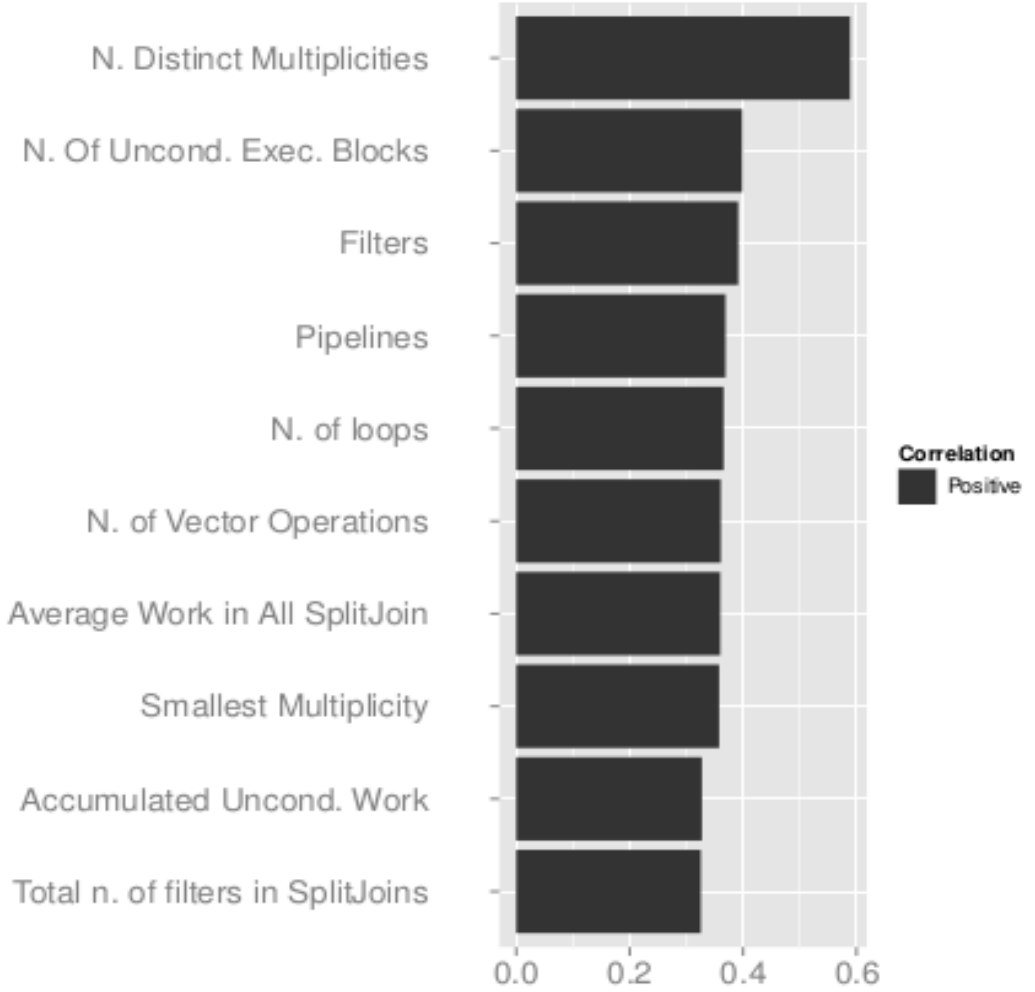
- Unrolling increases the degree of parallelism.
- There is a need to consider the number of resources allocated to a thread.

Predicting optimal number of Threads: Synthetic Benchmark Generation

- Generate synthetic Streamit benchmarks using set of micro-kernels found in examples.
- Ensure total number of filters and splits in a benchmark is within average of realistic benchmark.
- Generate 15 different threaded versions for each benchmark.
- Single core per thread and record the cycles.
- Generate 1000 such different benchmark applications.

Predicting optimal number of Threads: Extracting Features

- Distinct Multiplicities: The distinct number of rates (number of times) of filter executions in an application.
- High number implies subsets of filters executing at different rates.
- More threads required to group together filters with similar multiplicities.
- Splitjoins, Pipelines and Filters have less impact on number of threads.

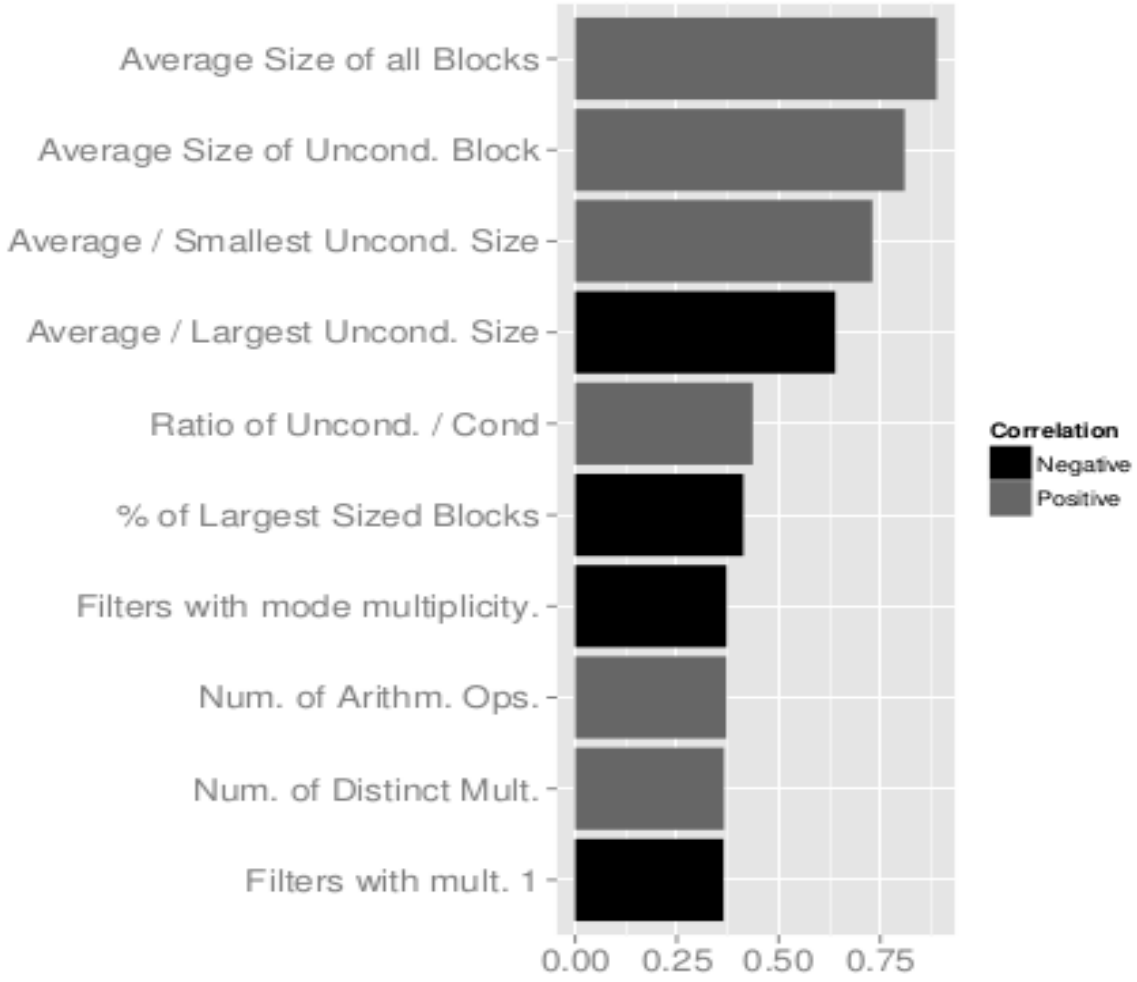


Predicting optimal number of Threads: Prediction

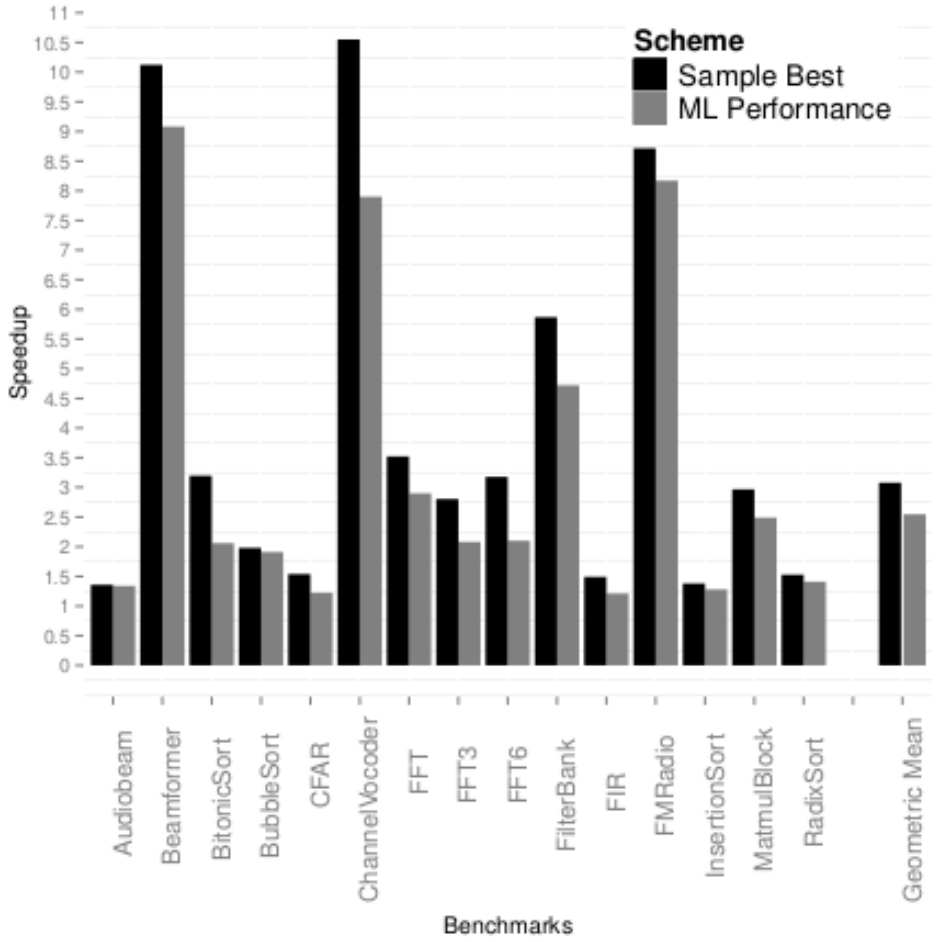
- KNN Model used for prediction.
- KNN model determines k closest generated applications.
- Distance between features is measured using Euclidean distance for each application.
- Value of k=7.
- The optimal number of threads is average number of threads of k neighbors.

Predicting Core Composition

- Most correlating feature is the number of statements in the block.
- Next correlating feature is size of unconditional block.
- Unconditional blocks affect the size of instruction pipeline which is facilitated by number of cores.
- Streamit programs do not tend to have large size of conditional blocks.
- Linear regression to predict the number of cores.



Results:



- The speedups obtained compared with the sample's best.
- Performance of the predicted configuration is within 16% of the sample's best.

Conclusion:

- Pros
  1. Prediction based on static/compile time features.
  2. Reduce the complexity for selecting the best configuration in design space.
- Cons
  1. Should clearly explain the procedure to generate training data for predicting number of cores.
  2. How to determine the core composition even if we know the optimal number of cores.