Neural Network Assisted Tile Size Selection

Mohammed Rahman, Louis-Noël Pouchet and P. Sadayappan

Dept. of Computer Science and Engineering
Ohio State University

June 22, 2010

iWAPT 2010 Workshop
Berkeley, USA

THE OHIO STATE UNIVERSITY

Overview

Situation:

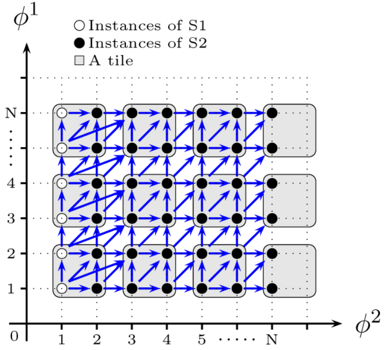
- New advances in parametric tiling $\longrightarrow$ more user code to be tuned
- The problem of tile size selection is complex and unsolved!

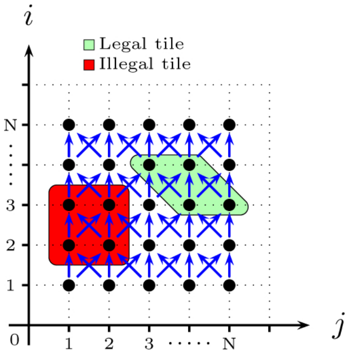
Our approach:

- Use machine learning to create a performance predictor of tile size performance, for a specific program
- Rely on the distribution shape to extract promising subspaces for empirical search
- Outcome: $< 2\%$ of the space traversed $\longrightarrow$ 90+% of maximal speedup achieved

Tiling

- Tiling partition the computation into blocks
- Note we consider only rectangular tiling here
- For tiling to be legal, such a partitioning must be legal





Parametric Tiling

Automatic parametric tiling [ICS'09,CGO'10]:

- Produce code where the tile dimensions are parameters
- Seamlessly find/apply all required transformation to make the code tilable
- Actual tile sizes are given at run-time
- very useful for tile size selection (no need to recompile)
- recent progresses have generalized the approach:
  - Operates on arbitrary affine-control loops (imperfectly nested)
  - Produce good quality code
  - Even expose pipeline-parallelism if needed
  - Software (from OSU): Pluto, PrimeTile/DynTile/PTile

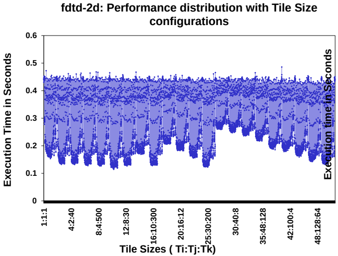
Tile Size Selection

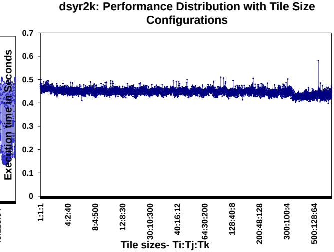
Problem: how to select the tile size to have the best performance?

- data reuse *within the execution of a tile*;
- data reuse *between tiles*;
- the layout in memory of the data used in a tile;
- the relative penalty of misses at each level of the hierarchy, which is machine-dependent.
- the cache replacement policy;
- the interaction with other units, such at prefetching;
- the interaction with vectorization, to enable a profitable steady-state for the vectorized loop(s);
- ...

Performance Distribution

Performance distribution of fdtd-2d and syr2k





- Search space: 10648 possible tile sizes
  - $\{1, 2, 4, 6, 8, 10, 12, 16, 30, 32, 40, 48, 64, 100, 128, 150, 200, 256, 300, 400, 500, 600\}$
- Machine: Core i7 (1 thread)
- 2 "standard" distribution shapes

Ojectives

Correlate execution time with tile sizes

- (Static) performance models do exist...
- ... but fail to capture the interplay between all hardware components
- Usually better suited for well-known problems (eg, uniform reuse + square tiles)
- Another view: pruning the space of poor-performing tile sizes

Our approach:

- Build a neural network to model the performance distribution
- Focus directly on the execution time
- ANN dedicated to a specific program + dataset size

Neural Network

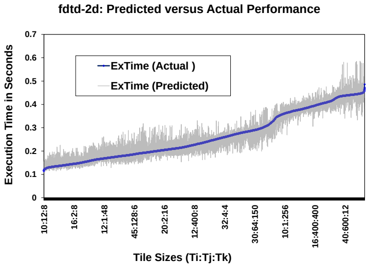
Layout:

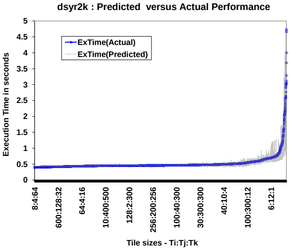
- Fully connected, multi-layer perceptron (MLP)
- Input layer: the tile sizes (T_i, T_j, T_k)
- Output layer: predicted execution time
- One hidden layer consisting of 30 hidden neurons
- Use Stuttgart Neural Network Simulator library

Training:

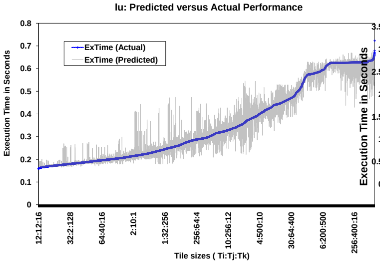
- Select 5% (530 tuples) from the search space of 10648
- Run the program on the machine using the tile size specified by the tuples
- Train with resilient back-propagation (rprop), using the actual execution time for a tuple
- Standard 10% cross-validation procedure

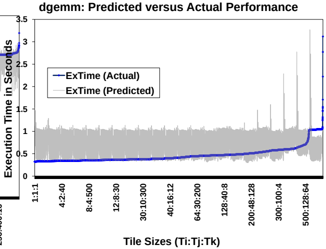
Performance Prediction [1/2]





Performance Prediction [2/2]





Discussions

- for trmm, lu, 2d-jacobi, syr2k and doitgen, predict more than 90% of our search space with less than 10% deviation for the actual execution time
- In total, can predict 80% and more with less than 10% deviation
- Usually smaller deviation for the best tile sizes

→ **These ANN are able to model the performance distribution**

Openings:

- Program classifier w.r.t. performance distribution
- Training: do not "fit" that much the training points?

Selecting the Best Tile Size

The performance distribution can drive the empirical search to focus on promising subspaces

Tile size selection:

- Random approach has a huge variability on some distribution shapes
- Exhaustive search is likely not needed
- Need for an intermediate solution
  - Low number of empirical runs
  - Good convergence, good variability
  - General enough to work on arbitrary user codes

Overview of the Algorithm

1. Generate a parametrically tiled code
2. Randomly select x% of the tile size space, and run them on the machine
3. Train an ANN using this data
4. Use the ANN to predict performance of the entire space
5. Collect y tile sizes that are predicted best and not already ran
6. Run the y tile sizes on the machine, output the best found

Experimental Setup

- Studied various kernels (perfectly/imperfectly nested, BLAS & stencils)
- Only focused on single-threaded execution, on an Intel Core i7

- Comparison: simple random search (R), ANN search (ANN)
- Repeat each experiment 100 times, for various sampling rate

Experimental Results ($y = 50$)

		doitgen	gemm	syr2k	lu	2d-jacobi	fdtd-2d
1%	R-best	100%	99.86%	98.15%	99.89%	99.91%	97.75%
	R-average	98.71%	96.29%	94.80%	92.19%	94.10%	84.15%
	R-worst	**95.35%**	**69.64%**	**89.81%**	**40.63%**	**17.69%**	**31.02%**
	ANN-best	100%	99.86%	100%	100%	99.91%	100%
	ANN-average	98.89%	96.35%	96.01%	92.62%	98.51%	84.50%
	ANN-worst	**97.26%**	**82.93%**	**89.79%**	**79.68%**	**94.23%**	**66.53%**
2%	R-best	99.97%	99.86%	98.71%	99.89%	100%	100%
	R-average	98.71%	96.42%	94.80%	92.87%	97.60%	84.10%
	R-worst	**86.49%**	**67.89%**	**88.20%**	**45.29%**	**55.98%**	**27.30%**
	ANN-best	100%	99.86%	100%	100%	100%	100%
	ANN-average	98.89%	96.76%	96.69%	95.34%	98.55%	88.61%
	ANN-worst	**97.26%**	**89.83%**	**89.65%**	**85.80%**	**94.17%**	**60.65%**
3%	R-best	99.97%	99.86%	98.71%	99.89%	100%	100%
	R-average	98.77%	96.47%	94.80%	94.27%	98.39%	85.47%
	R-worst	**94.89%**	**63.58%**	**87.99%**	**61.24%**	**84.54%**	**47.99%**
	ANN-best	99.97%	99.86%	100%	100%	100%	100%
	ANN-average	98.93%	97.14%	97.17%	95.34%	98.74%	91.45%
	ANN-worst	**97.64%**	**91.01%**	**92.27%**	**85.80%**	**94.50%**	**63.34%**
4%	R-best	99.97%	99.86%	98.71%	99.89%	100%	100%
	R-average	98.80%	96.65%	94.93%	92.19%	98.41%	85.55%
	R-worst	**96.86%**	**69.73%**	**88.57%**	**52.03%**	**82.47%**	**43.74%**
	ANN-best	100%	99.86%	100%	100%	100%	100%
	ANN-average	98.99%	97.67%	97.20%	95.79%	98.90%	93.55%
	ANN-worst	**98.28%**	**93.65%**	**92.66%**	**85.80%**	**94.50%**	**79.26%**

Some Related Work

Epshteyn *et al.* [LCPC'05]:

- Search-oriented contribution
- Uses regression curves to approximate the performance distribution
- Uses active learning to select good candidates for empirical evaluation
- Good results for BLAS kernels

Yuki *et al.* [CGO'10]:

- Aims at selecting/combining between different static models
- Uses program features to characterize accesses, train ANN
- Results demonstrated for matrix-like kernels

Conclusions and Future Work

ANN is a candidate approach to connect tile sizes with performance

- Good prediction quality
- Deviation usually smaller for the good points
- Combined search heuristic proposed:
  - Strong variability improvement over naive random approach
  - 90+% efficiency using $<$ 2% of the space, likely can be improved further

Future work:

- **Generalization!**
  - Categorize benchmarks reg. the performance distribution shape
  - Dataset size
- Do not try to fit the random samples during training
  - Reduce the training time
  - problem: ANN configuration

Acknowledgements

This work was funded in part by the U.S. National Science Foundation through award 0926688 and the Defense Advanced Research Projects Agency through AFRL Contract FA8650-09-C-7915. The opinions and findings in this document do not necessarily reflect the views of either the United States Government or the Ohio State University.