Code Optimisations and Performance Models for MATLAB

Patryk Kiepas[1,2], Claude Tadonki[1], Corinne Ancourt[1]
Jarosław Koźlak[2]

[1]MINES ParisTech/PSL University

[2]AGH University of Science and Technology, Poland

January 30, 2019

Outline

- Motivation – Why MATLAB?
- Three approaches to speedup MATLAB
- Code transformations
  - Loop coalescing
  - Loop interchange
  - Loop unrolling
  - Strength reduction (power)
- Problem with vectorization
- MATLAB is JIT compiling
- Building an optimisation heuristics
- Conclusions

MATLAB is popular

Dec 2018	Dec 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.932%	+2.66%
2	2		C	14.282%	+4.12%
3	4	↑	Python	8.376%	+4.60%
4	3	↓	C++	7.562%	+2.84%
5	7	↑	Visual Basic .NET	7.127%	+4.66%
6	5	↓	C#	3.455%	+0.63%
7	6	↓	JavaScript	3.063%	+0.59%
8	9	↑	PHP	2.442%	+0.85%
9	-	⇈	SQL	2.184%	+2.18%
10	12	↑	Objective-C	1.477%	-0.02%
11	16	⇈	Delphi/Object Pascal	1.396%	+0.00%
12	13	↑	Assembly language	1.371%	-0.10%
13	10	↓	MATLAB	1.283%	-0.29%
14	11	↓	Swift	1.220%	-0.35%

Figure: TIOBE Index for December 2018. https://www.tiobe.com/tiobe-index/

Motivation

MATLAB

+ Dynamic language with simple and intuitive syntax
+ Great for fast-prototyping
  - Built-ins: 2940 (R2018b)
  - MATALB toolboxes: 66 (e.g. phased array, aerospace)

– Vendor lock-in, closed source

– Lack of formal semantics

– Performance is lagging behind other solutions

Performance comparison

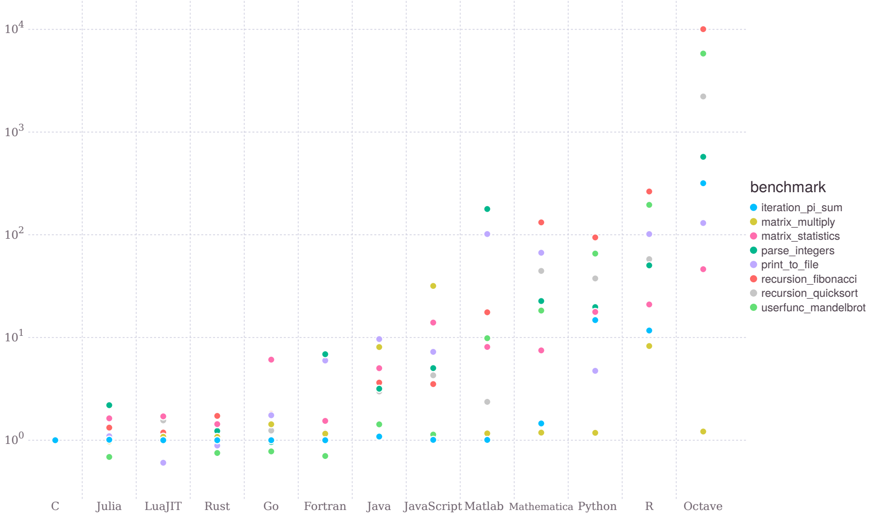
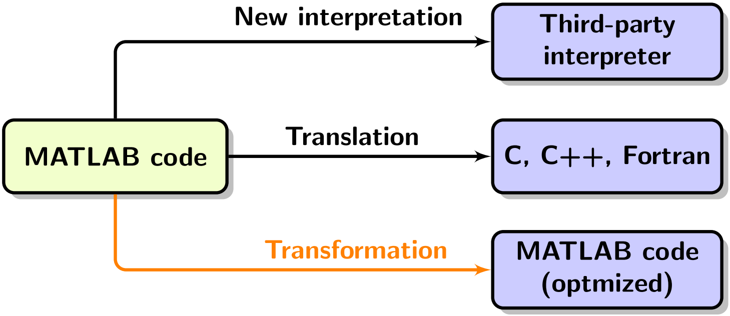


Figure: Julia Micro-Benchmarks. https://julialang.org/benchmarks/

Three approaches to speedup MATLAB



Existing solutions

- New interpretation
  - Scilab[1]
  - Octave[2]
  - MaJIC [Almasi and Padua, 2001]
  - McVM [Chevalier-boisvert, 2009]
- Translation
  - MATALB Coder (C) – official MathWork's compiler
  - SILKAN eVariX[3] (C)
  - Menhir (C) [Chauveau and Bodin, 1999]
  - Mc2For (Fortran) [Chen et al., 2017]
  - FALCON (Fortran) [DeRose et al., 1995]
- Transformation
  - Mc2Mc [Chen et al., 2017] – performs vectorization

[1] https://www.scilab.org/
[2] https://www.gnu.org/software/octave/
[3] http://www.silkan.com/products/evarix/

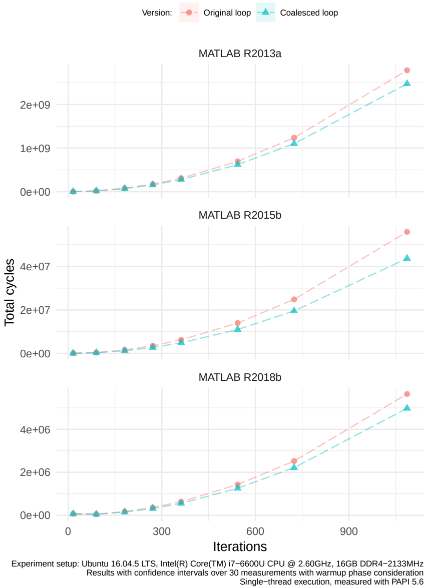
Loop coalescing

Before:

```
for k = 1:N
 for l = 1:M
  a(l, k) = a(l, k) + c;
 end
end
```

After:

```
for T = 1:(N .* M)
 a(T) = a(T) + c;
end
```



Example: Bacon, D. F., Graham, S. L., & Sharp, O. J. (1994). Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4), 345–420.

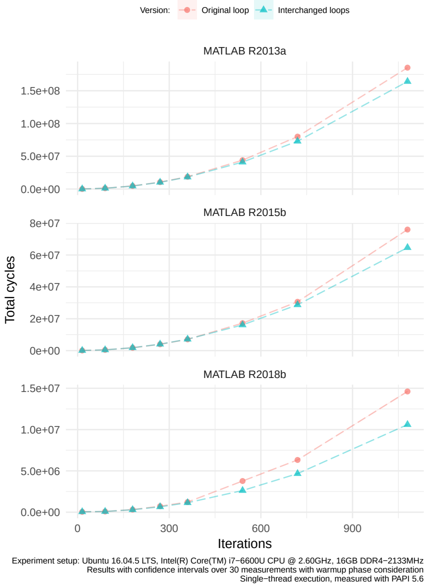
Loop interchange

Before:

```
for k = 1:N
 for l = 1:M
  total(k) = total(k) + a(k, l);
 end
end
```

After:

```
for l = 1:M
 for k = 1:N
  total(k) = total(k) + a(k, l);
 end
end
```



Example: Bacon, D. F., Graham, S. L., & Sharp, O. J. (1994). Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4), 345–420.

Loop unrolling

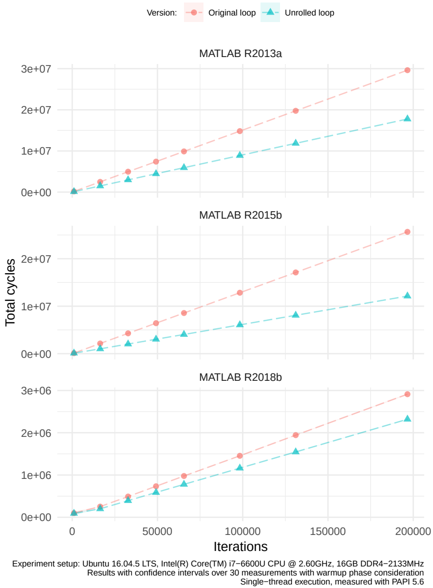
Before:

```
for k = 2:(N - 1)
 a(k) = a(k) + a(k-1) .* a(k+1);
end
```

After:

```
for k = 2:2:(N - 2)
 a(k) = a(k) + a(k-1) .* a(k+1);
 a(k+1) = a(k+1) + a(k) .* a(k+2);
end

if mod((N-2), 2) == 1
 a(N-1) = a(N-1) + a(N-2) .* a(N);
end
```



Example: Bacon, D. F., Graham, S. L., & Sharp, O. J. (1994). Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4), 345–420.

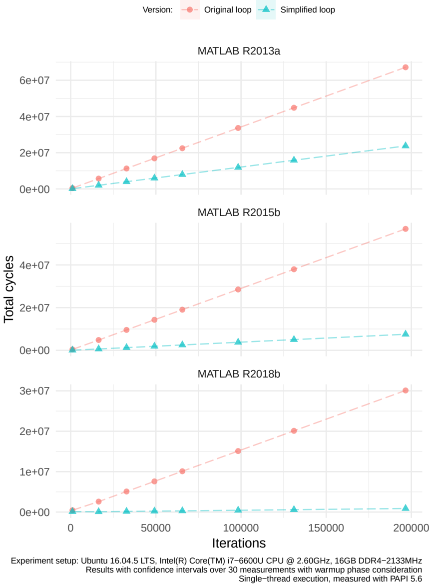
Strength reduction (power)

Before:

```
for k = 1:N
 a(k) = a(k) + c.^k;
end
```

After:

```
T = c;
for k = 1:N
 a(k) = a(k) + T;
 T = T .* c;
end
```



Example: Bacon, D. F., Graham, S. L., & Sharp, O. J. (1994). Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4), 345–420.

Vectorization in MATLAB

```matlab
% scalar form
for i = 1:N
  c(i)=a(i)*b(i)
end
% vector form
c(1:N)=a(1:N).*b(1:N)
% after simplification
c=a.*b
```

- For many years vectorization was a prevalent optimisation, usually applied systematically

+ Performing more floating-point operations simultaneously

− Sometimes decreases performance in comparison to JIT-compiled loops (Chen et al. 2017 and Kiepas et al. 2018)

Reproduction of [Chen et al., 2017]

- Benchmarks from *Ostrich*-suite[4]
- Vectorized with Mc2Mc
- Executed on MATLAB R2015b

Benchmark	Dwarf	Chen et al.	Us
backprop	unstructured grid	0.71	0.81
bs	–	15.0	8.33
capr	dense linear algebra	0.79	0.85
crni	structured grid	0.83	0.81
fft	spectral method	0.59	0.64
nw	dynamic programming	0.96	1.00
pagerank	Monte Carlo/MapReduce	0.94	0.94
mc	Monte Carlo/MapReduce	2.02	2.22
spmv	sparse linear algebra	0.013	0.02

Table: Kiepas, P., Kozlak, J., Tadonki, C., & Ancourt, C. (2018). Profile-based vectorization for MATLAB. ARRAY 2018 (pp. 18–23).

[4] https://github.com/Sable/Ostrich2

Is vectorization still relevant?

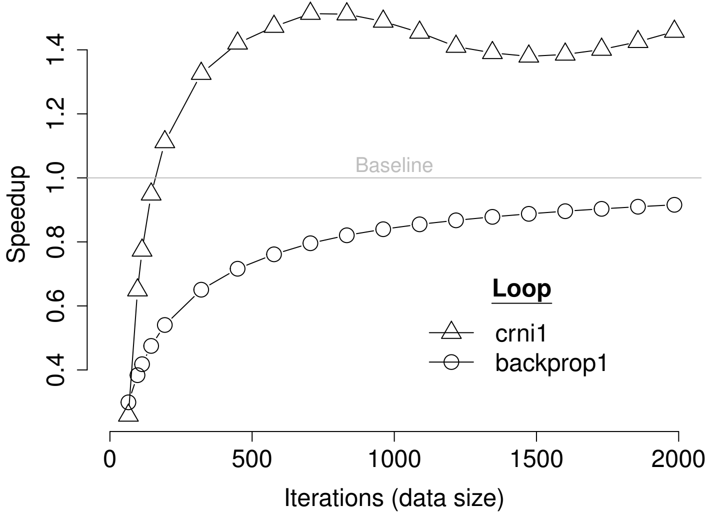


Figure: Kiepas, P., Kozlak, J., Tadonki, C., & Ancourt, C. (2018). Profile-based vectorization for MATLAB. ARRAY 2018 (pp. 18–23).

Improving Mc2Mc code generation

Range inlining

```
% From
k = 1:N;
B = A(k) + 2;
% To
B = A(1:N) + 2;
```

Removing explicit index-all

```
% From
B(:) = A(1:end);
% To
B = A;
```

Range conversion

```
% From
B = A(2*(1:N)-1);
% To
B = A(1:2:(2*N-1));
```

Profitable vectorization point (PV)

Loop	Benchmark iterations	PV iterations	Improved PV iterations
backprop1	$\{17, 2850001\}$	$\emptyset$	≥ 255
backprop2	2	≥ 4033	≥ 257
backprop3	$\{17, 2850001\}$	$\emptyset$	≥ 385
backprop4	2	$\emptyset$	≥ 257
capr1	8	≥ 20	≥ 17
capr2	20	≥ 3329	≥ 385
capr3	49	≥ 5953	≥ 321
crni1	2300	≥ 161	≥ 193
crni2	2300	$\emptyset$	≥ 289
crni3	2300	$\emptyset$	≥ 1217
fft1	256	$\emptyset$	≥ 417
fft2	$2, 4, 8 \dots 256$	$\emptyset$	≥ 129
nw1	4097	$\emptyset$	≥ 65
nw2	4097	≥ 1665	≥ 257
nw3	4097	≥ 7681	≥ 193
pagerank1	1000	$\emptyset$	≥ 273
spmv1	$\{2, 3\}$	≥ 6337	≥ 321

Table: Kiepas, P., Kozlak, J., Tadonki, C., & Ancourt, C. (2018). Profile-based vectorization for MATLAB. ARRAY 2018 (pp. 18–23).

Profile-guided vectorization

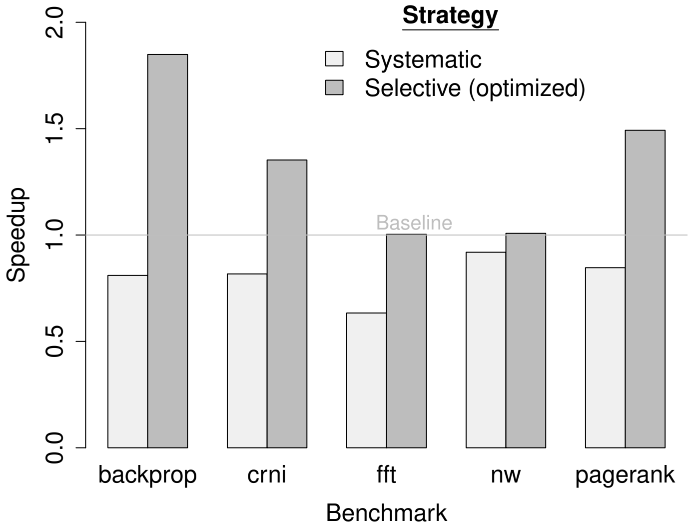


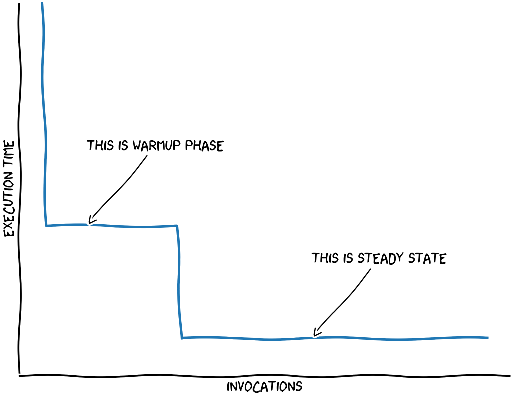
Figure: Kiepas, P., Kozlak, J., Tadonki, C., & Ancourt, C. (2018). Profile-based vectorization for MATLAB. ARRAY 2018 (pp. 18–23).

A bit of history of MATLAB

- Starts as an interpreter (1984)
- Introduces JIT along the interpreter around 6.5 (2002)
- Combines JIT with the interpreter in R2015b
- Introduces PGO (profile-driven optimisations) around R2018b

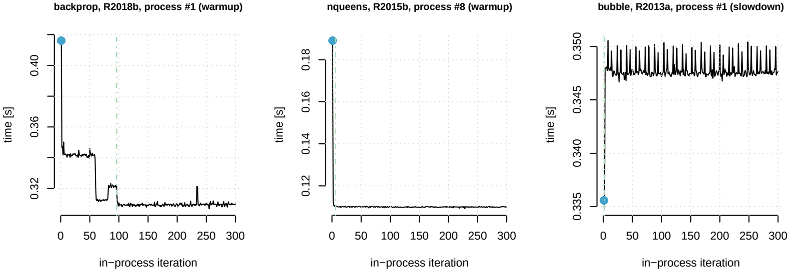
Warmup phase

Warmup is an observable effect of some JIT policy performing compilation on a code. Policy is a set of rules *if*, *when* and *how* to compile the code [Kulkarni 2011].



[Kulkarni 2011]: Kulkarni, P. A. (2011). JIT compilation policy for modern machines. ACM SIGPLAN Notices, 46(10), 773.

Warmup phase patterns



The patterns come in different flavours [Barrett et al. 2017]:

- Warmup
- Slowdown
- Flat
- Inconsistent

[Barrett et al. 2017]: Barrett, E., Bolz-Tereick, C. F., Killick, R., Mount, S., & Tratt, L. (2017). Virtual machine warmup blows hot and cold. Proceedings of the ACM on Programming Languages, vol. 1 (Issue OOPSLA), 1–27.

About our heuristics

Our heuristics is a binary choice (*optimise* – *positive* / *do nothing* – negative) that takes into consideration the code, trip count and/or the machine's properties.

Designing goal

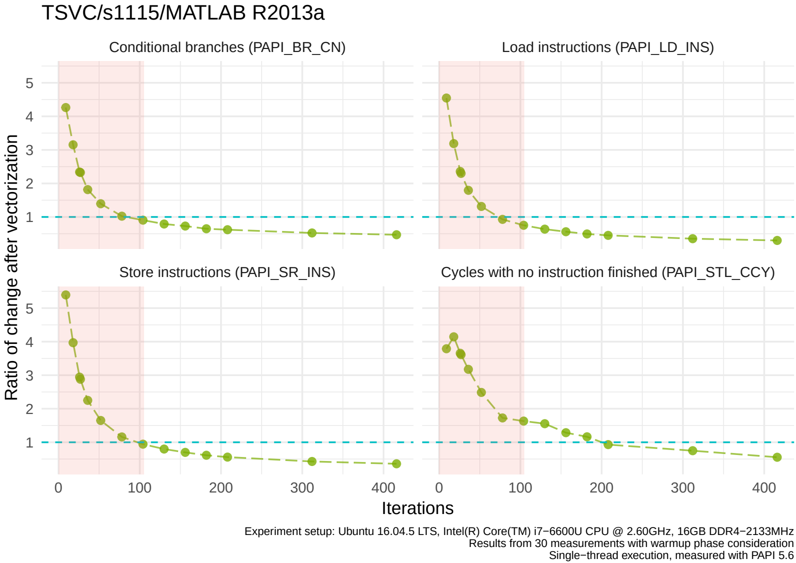
Prefer being conservative (false negatives FN are OK) than optimising wrongly (false positives $FP > 0$).

$$precision = \frac{TP}{TP + FP} \to 1 \tag{1}$$

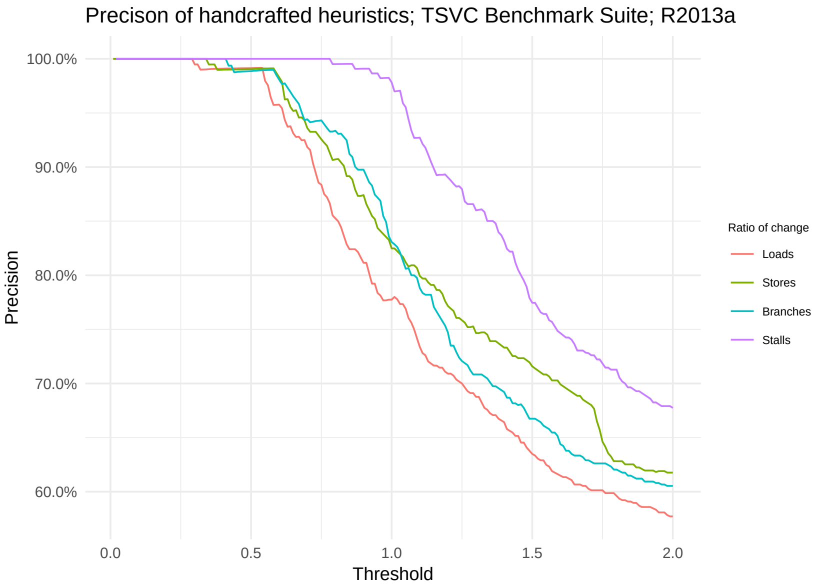
However, too much FN means we are optimising only a little!

1. Handcrafted optimisation heuristics

We pose a question: *What does vectorization change?*



Precision



Precison of handcrafted heuristics; TSVC Benchmark Suite; R2013a
100.0%
90.0%
80.0%
70.0%
60.0%
Precision
0.0
0.5
1.0
1.5
2.0
Threshold
Ratio of change
Loads
Stores
Branches
Stalls

2. Automatic dynamic model

Followed by the work of [Cavazos et al., 2007] – we have build a model using machine learning and dynamic set of features (performance counters).

Methodology

1. Collecting performance counters (TSVC Benchmark Suite)
2. Normalising (by *PAPI_TOT_INS*, hybrid)
3. Oversampling for dealing with class imbalance
4. Training on TSVC, testing on LCPC16 [Chen et al., 2017]
5. Only out-of-the-box components, no fine-tuning (meta-learning, hyper parameter optimisations)

[Cavazos et al. 2007]: Cavazos, J., Fursin, G., Agakov, F., Bonilla, E., O'Boyle, M. F. P., & Temam, O. (2007). Rapidly Selecting Good Compiler Optimizations using Performance Counters. CGO'07 (pp. 185–197).

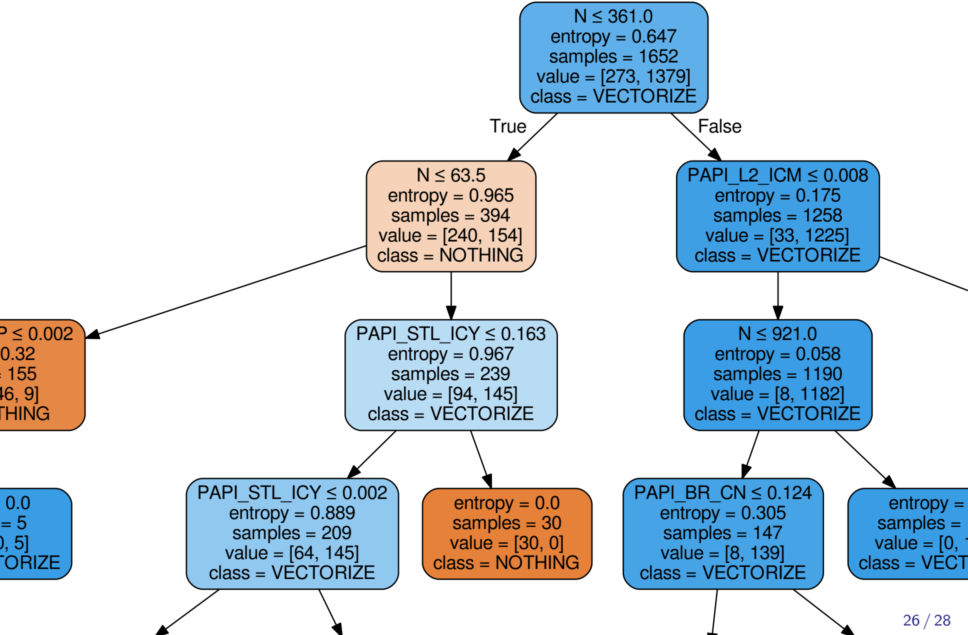
.

Evaluation

Test	Metrics	AdaBoost	Decision Tree (CART)
TSVC (Cross-validation[5])	Precision (%)	96.63 %	97.02 %
	Accuracy (%)	94.38 %	93.95 %
LCPC16 Test set	Precision (%)	99.51 %	99.36 %
	Accuracy (%)	92.85 %	72.26 %

[5] 10-folds

Decision tree



3. Automatic static model

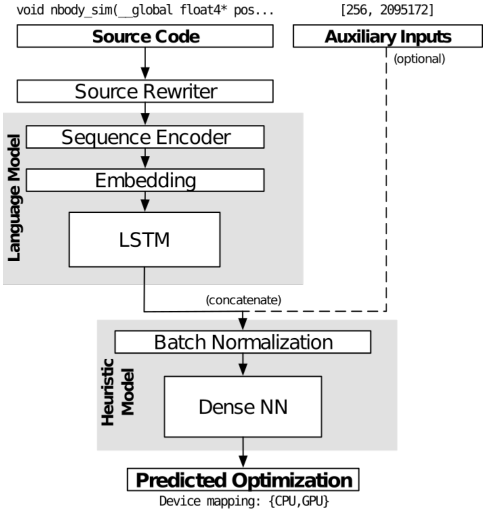


Image: Cummins, C., Petoumenos, P., Wang, Z., and Leather, H. (2017). End-to-End Deep Learning of Optimization Heuristics. In 2017 26th IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT). [Cummins et al., 2017]

- Sequences of codes are the input
- Auxiliary inputs: number of iterations
- No dynamic features
- In order to force learning from sequences – shorten sequences (less padding)
- Small precision – more data? Around 1652 data points, but only 118 code sequences.

Conclusions

- Working optimisation heuristics without opening the MATLAB's black-box (which might be infeasible)
- Deeper understanding of how to measure MATLAB's performance
- Perspective: fine-tuning of models and extending evaluation for other machines and versions of MATLAB

Thank you!

Almasi, G. and Padua, D. (2001).
MaJIC: A Matlab just-in-time Compiler.
In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 2017, pages 68–81.

Cavazos, J., Fursin, G., Agakov, F., Bonilla, E., O'Boyle, M. F., and Temam, O. (2007).
Rapidly Selecting Good Compiler Optimizations using Performance Counters.
In *International Symposium on Code Generation and Optimization (CGO'07)*, pages 185–197. IEEE.

Chauveau, S. and Bodin, F. (1999).
Menhir: An Environment for High Performance Matlab.
Scientific Programming, 7(3-4):303–312.

Chen, H., Krolik, A., Lavoie, E., and Hendren, L. (2017).
Automatic Vectorization for MATLAB.

In Ding, C., Criswell, J., and Wu, P., editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10136 LNCS of *Lecture Notes in Computer Science*, pages 171–187. Springer International Publishing, Cham.

Chevalier-boisvert, M. (2009).
MCVM : An Optimizing Virtual Machine for The MATLAB Programming Language.

Cummins, C., Petoumenos, P., Wang, Z., and Leather, H. (2017).
End-to-End Deep Learning of Optimization Heuristics.
In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, volume 2017-Septe, pages 219–232. IEEE.

DeRose, L., Gallivan, K., Gallopoulos, E., Marsolf, B. A., and Padua, D. (1995).
FALCON: An Environment for the Development of Scientific Libraries and Applications.

Proc. First International Workshop on Knowledge-Based System for the (re)Use of Program Libraries, (November).