Software Metrics for Parallelism

MSc by Research project

Student:

Aleksandr Maramzin, s1736883

Supervision team:

Björn Franke, Michael O'Boyle, Kenneth Heafield

EPSRC Engineering and Physical Sciences Research Council

EPSRC Centre for Doctoral Training in Pervasive Parallelism

THE UNIVERSITY of EDINBURGH informatics

General words on parallel programming and its problems

Problem statement

Abundance of parallel hardware across the whole spectrum from small embedded devices to warehouse-scale server farms

Difficulty of manual parallel programming

Existent parallelizability assistance tools give either conservative binary “yes”/”no” answers, or are quite complex and require serious training

Software Metrics in Computer Science

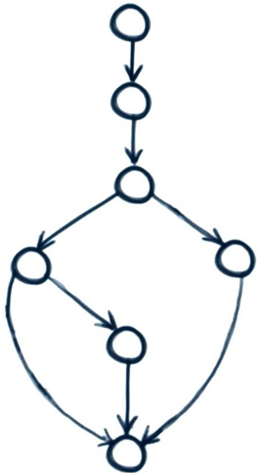
Software Source Code Metrics in Software Engineering

Software Quality

Bugs per line of code

Lines Of Code (LOC)

Cyclomatic Complexity (CC)



Halstead's Software Science

For a given problem, Let:

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

Software Coupling
&
Cohesion

Code coverage

Parallel Programming

The only metrics in the subfield are different variations of speedup

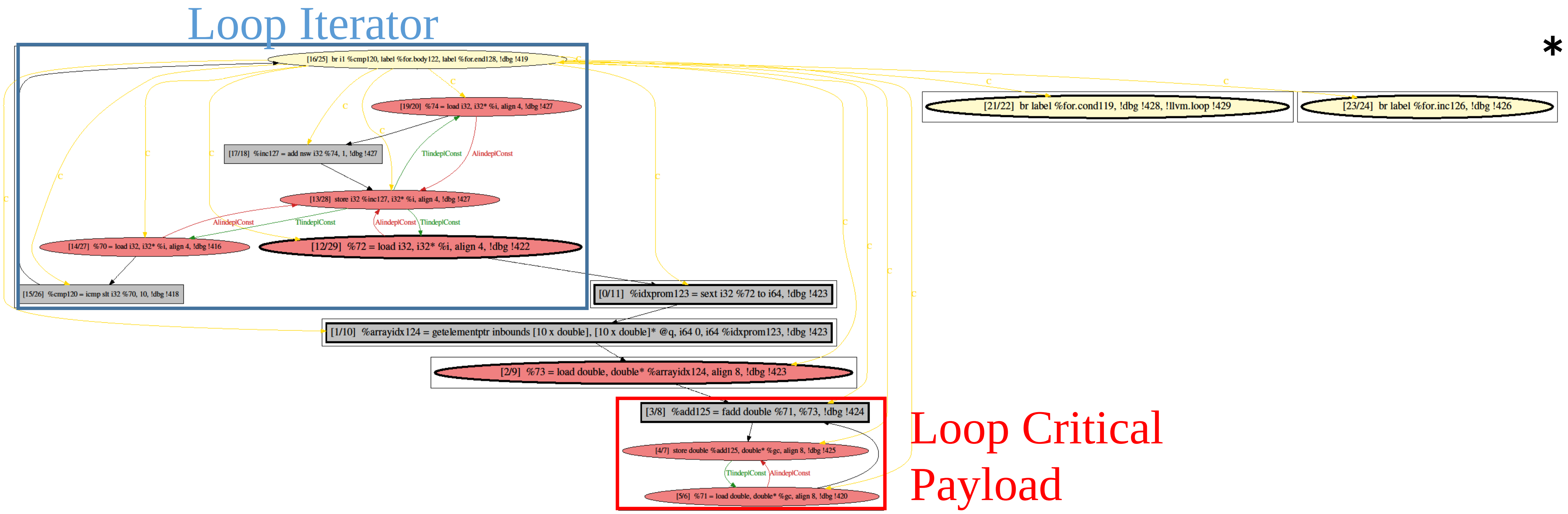
$$speedup = \frac{\text{serial execution time}}{\text{parallel execution time}}$$

- **Relative speedup** (serial algorithm vs parallel algorithm)
- **Real speedup** (best known serial algorithm vs parallel algorithm)
- **Absolute speedup** (best known serial algorithm on the best serial hardware vs parallel algorithm)
- **Analytical speedup**
- **Asymptotic speedup** $(O_{serial}(n)/O_{parallel}(n))$

Idealistic MSc by Research Goal

- Establish a set of Software Metrics for Parallelism, which would provide a software engineer with a real-time feedback about parallelizability of his source code
- Parallelizability Metrics must reflect algorithmic parallelizability and at the same time capture lower-level details of source code implementation
- Parallelizability Metrics must supplement compiler’s conservative binary “yes”/”no” answers with continuous function of source code parallelizability
- This continuous function must be monotonous and its values must grow from non-parallelizible and hard to parallelize loops to less parallelizability constrained and parallelizible loops

Dependence-based Anatomy of a Loop



* This Program Dependence Graph (PDG) has been produced by the developed PPar tool, thanks to DOT and Graphviz

* This loop has one Iterator SCC (with no incoming dependencies) and 6 Payload SCCs, one of those is proper (contains more than one LLVM IR instruction), thus critical

Software Metrics for Parallelism

Metric Group	Metric	Metric Definition	Intuition
Loop Proportions	Absolute Size	*Number of LLVM IR instructions in a whole loop*	The bigger the loop, the harder it is to parallelize it
	Payload Fraction	$\frac{\textit{Payload Instructions Number}}{\textit{Total Loop Instructions Number}}$	The smaller the payload fraction (hence, the more complex iterator is), the harder it is to parallelize a loop
	Proper SCCs Number	*Number of SCCs with more than one LLVM IR instruction in a payload of a loop*	The more proper SCCs we have, the harder this loop is for parallelization
	Critical Payload Fraction	$\frac{\textit{Critical Payload Instructions Number}}{\textit{Payload Instructions Number}}$	The bigger the critical part of a loop, the harder it is to parallelize a loop
Loop Dependencies Number	Payload Dependencies Number	*Number of PDG edges in a payload (**True**, **Anti**, **Output*** and ***Total**)*	The more dependencies we have, especially in the critical part of a loop payload, the harder it is to parallelize a loop
	Critical Payload Dependencies Number	*Number of PDG edges in a critical payload (**True**, **Anti**, **Output*** and ***Total**)*	
Loop Cohesion	Iterator/Payload Cohesion	$\frac{\textit{Edges Number Between Iterator / Payload}}{\textit{Total Loop Edges Number}}$	No apparent intuition
	Critical/Regular Payload Cohesion	$\frac{\textit{Edges Number Between Critical / Non-critical Payload}}{\textit{Total Payload Edges Number}}$	The tighter a regular payload is coupled with payload’s critical part (more edges in between, bigger cohesion value), the harder it is to parallelize a loop

Computed data

Proposed metrics have been computed for all NAS Parallel Benchmark loops

All NAS Parallel Benchmark loops have been classified (labelled) by Intel C/C++ compiler as parallelizible or not

loop location	ICC parallel	loop absolute size	loop payload fraction	loop proper sccs number	loop critical payload fraction	iterator payload total cohesion	iterator payload non-CF cohesion	critical payload total cohesion	critical payload non-CF cohesion	payload total dependencies number	payload true dependencies number	payload anti dependencies number	critical payload total dependencies number	critical payload true dependencies number
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/add.c(45)	1	84	0.119	1	0.7	0.04444	0	0	0	20	10	10	20	10
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/add.c(46)	0	72	0.1389	1	0.7	0.05714	0	0	0	20	10	10	20	10
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/add.c(47)	0	60	0.1667	1	0.7	0.08	0	0	0	20	10	10	20	10
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/add.c(48)	1	48	0.8125	1	0.07692	0.4078	0.02913	0.07895	0.07895	38	37	1	4	3
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(218)	0	91	0.8242	1	0.1867	0.2651	0.01606	0.1049	0.08392	143	94	36	74	38
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(212)	0	14	0.4286	0	0	0.2581	0.06452	0	0	3	3	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(180)	0	20	0.25	0	0	0.1818	0.02273	0	0	0	0	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(175)	0	10	0.3	0	0	0.1905	0.04762	0	0	0	0	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(161)	0	10	0.3	0	0	0.1905	0.04762	0	0	0	0	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(133)	0	5	0.2	0	0	0.125	0	0	0	0	0	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/bt.c(131)	0	5	0.2	0	0	0.125	0	0	0	0	0	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(66)	1	46	0.4348	2	0.55	0.2113	0.02817	0.0625	0.0625	32	20	12	26	14
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(67)	0	25	0.72	1	0.3333	0.3455	0.01818	0.1579	0.1579	19	17	2	9	7
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(50)	0	97	0.03093	1	0.6667	0.03361	0.01681	0.3333	0.3333	3	2	1	2	1
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(52)	0	80	0.0375	1	0.6667	0.04124	0.02062	0.3333	0.3333	3	2	1	2	1
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(54)	0	63	0.3016	2	0.5263	0.1667	0.02381	0.0625	0.0625	32	19	13	26	13
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(58)	0	41	0.7561	1	0.3226	0.3571	0.04082	0.1081	0.1081	37	32	5	18	13
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/BT/src/error.c(46)	1	13	0.4615	0	0	0.2593	0.03704	0	0	3	3	0	0	0
						.....								
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(190)	0	59	0.1695	1	0.7	0.08696	0	0	0	20	10	10	20	10
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(191)	0	49	0.8163	1	0.075	0.4095	0.02857	0.02564	0.02564	39	38	1	4	3
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(222)	0	128	0.5391	3	0.9565	0.1176	0.01103	0	0	216	113	89	214	112
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(225)	0	51	0.1373	1	0.8571	0.03846	0.01282	0.05882	0.05882	17	9	8	16	8
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(237)	1	27	0.6667	1	0.1667	0.339	0.0339	0.1765	0.1765	17	16	1	4	3
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(227)	1	26	0.6923	1	0.1667	0.3509	0.03509	0.1765	0.1765	17	16	1	4	3
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(278)	1	47	0.2128	1	0.5	0.12	0	0.07143	0.07143	14	8	6	12	6
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(279)	0	34	0.2941	1	0.5	0.1579	0	0.07143	0.07143	14	8	6	12	6
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/UA/src/utils.c(280)	0	21	0.619	0	0	0.3256	0.02326	0	0	10	10	0	0	0
/home/s1736883/Work/PParMetrics/benchmarks/nauseous/common/randdp.c(122)	0	67	0.8806	1	0.7966	0.3571	0.005952	0.08989	0.08989	89	68	21	79	58

Evaluation of Parallelizability Metrics

Proposed software source code parallelizability metrics have been evaluated in several ways:

1) **Loop Metric Values versus Loop Parallilizability Property**

   Metric values of all NAS loops have been plotted with parallelizability labels for a visual examination on correlations

2) **Principal Component Analysis (PCA) & K-Means clustering**

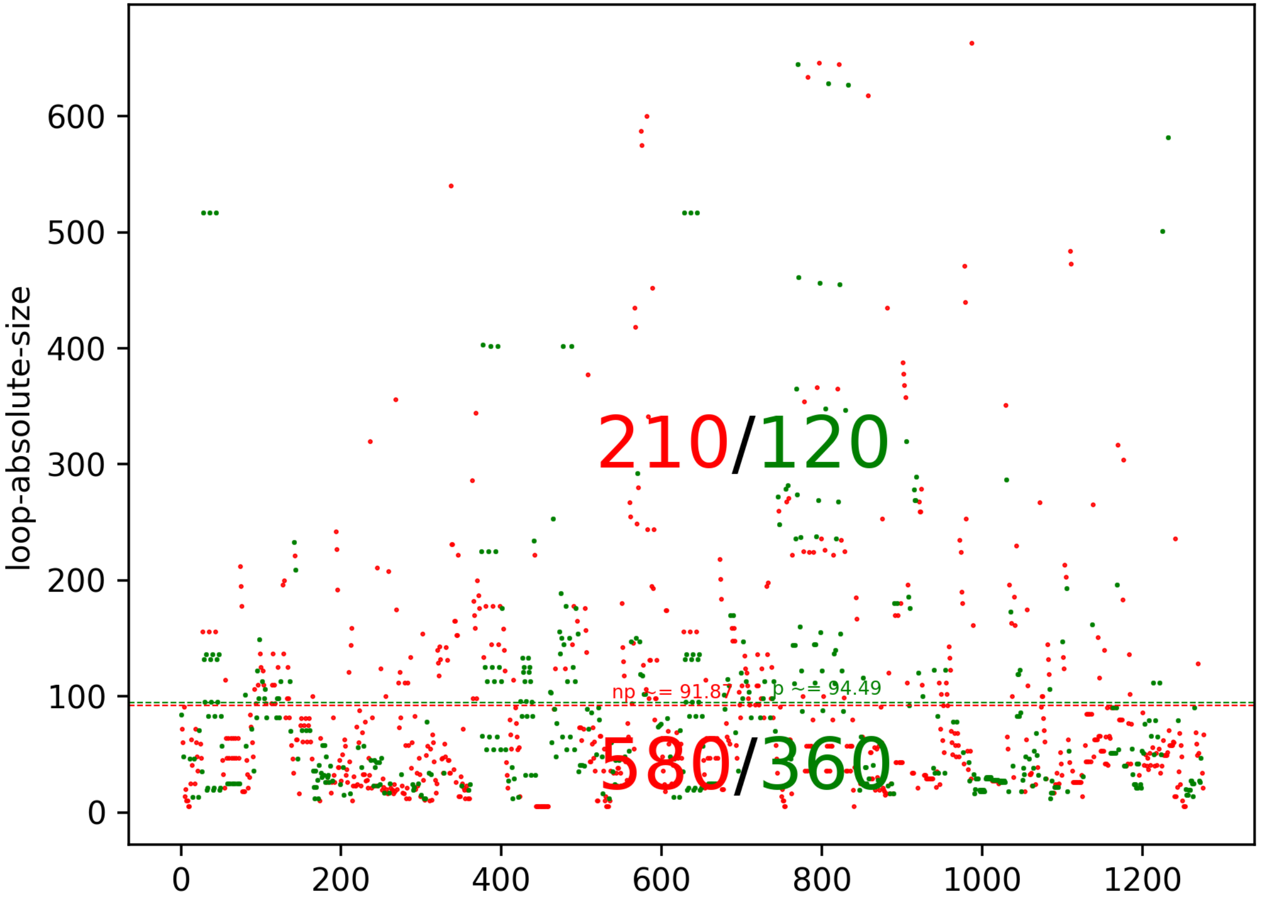
   For all NAS loops 13-dimensional metric points have been projected onto 3D space for examination of structural and spacial properties of dots distribution thanks to Principal Components Analysis (PCA) and K-Means clustering algorithms.

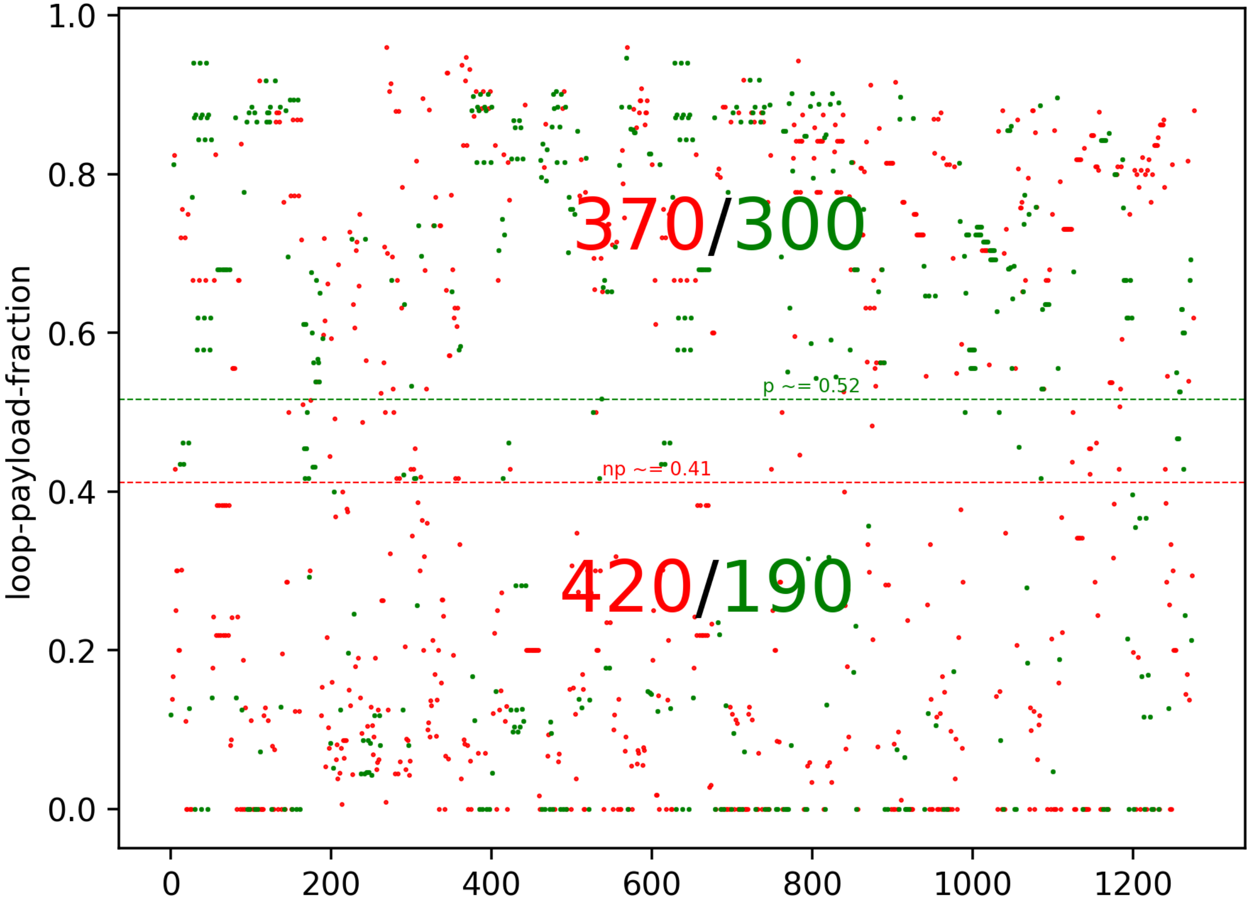
3) **Machine Learning based Metrics Ranking** Computed data table has been used for training of different machine learning models in order to see, which metrics train models (and hence reflect parallelizability property) the best.

4) **Manual NAS Benchmark loops examination**

Loop Metric Values vs Loop Parallelizability

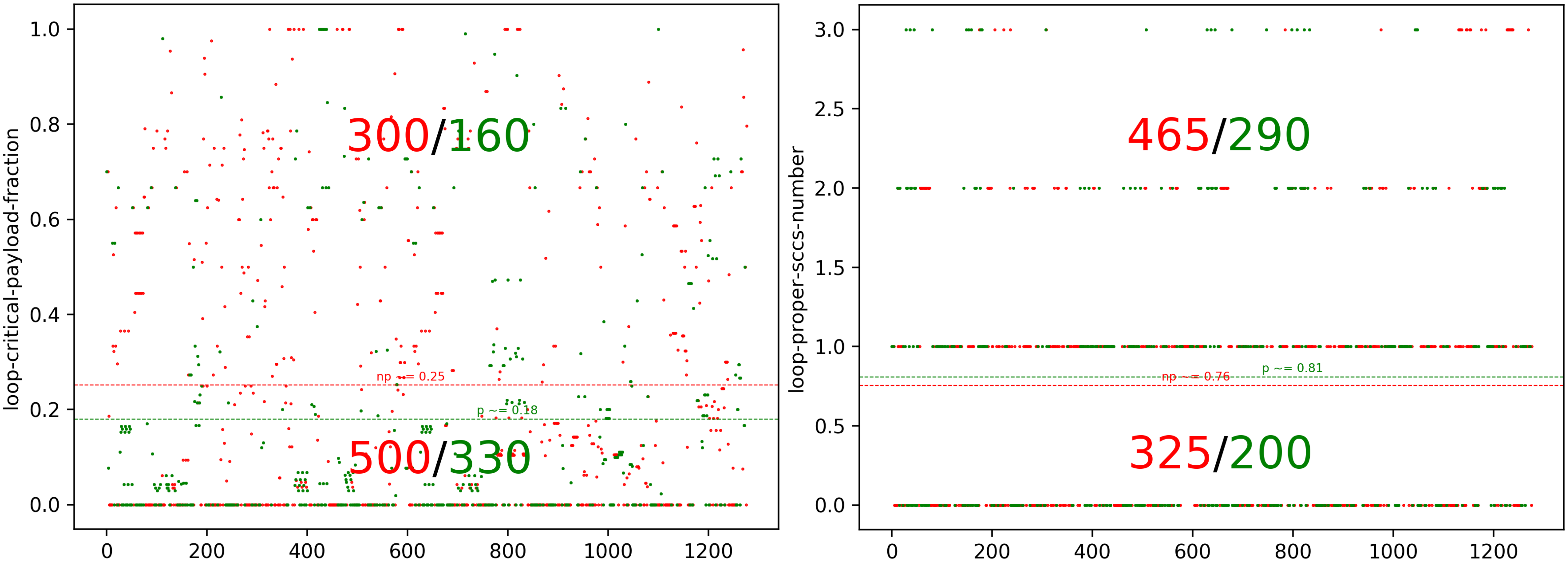
Loop Proportion Metrics [1]





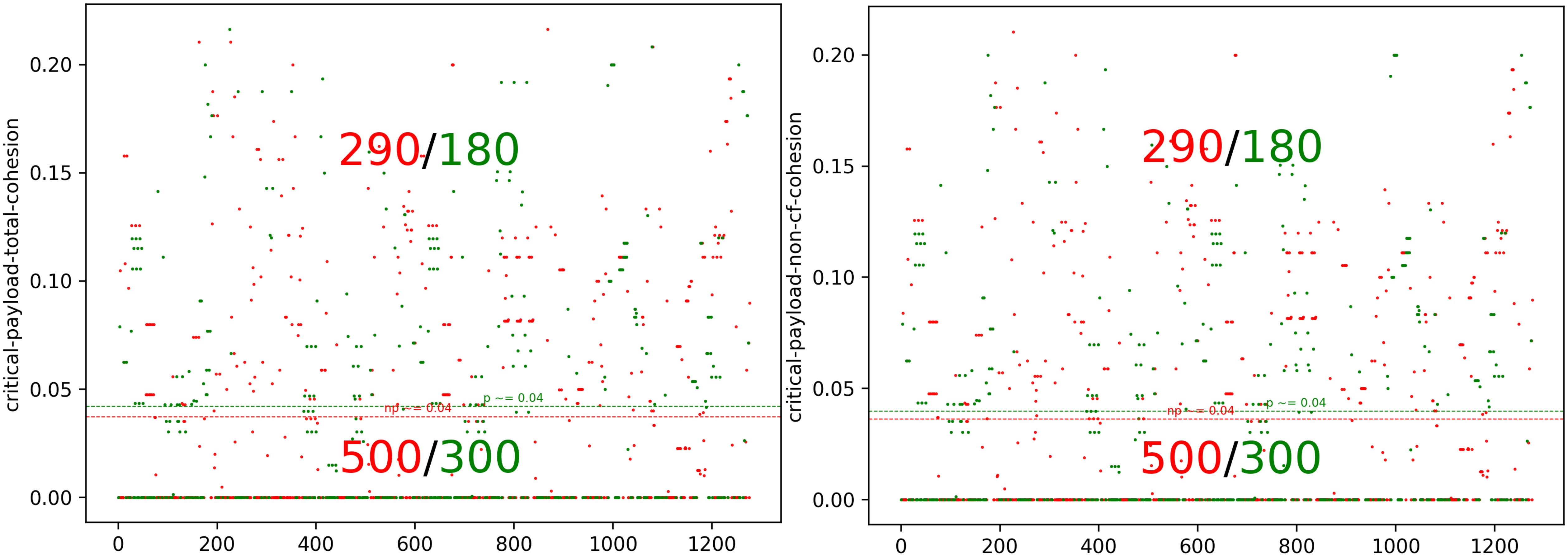
- While parallelizability property appears to be dispersed quite uniformly in the space of ***loop absolute size*** metric values, parallelizible loops tend to clutter at the top of ***loop payload fraction*** metric value range
- Mean ***loop payload fraction*** metric value for the subset of parallelizible (green) loops is slightly higher, than the mean metric value for the subset of non-parallelizible (red) loops: 0,52 against 0,41

Loop Proportion Metrics [2]



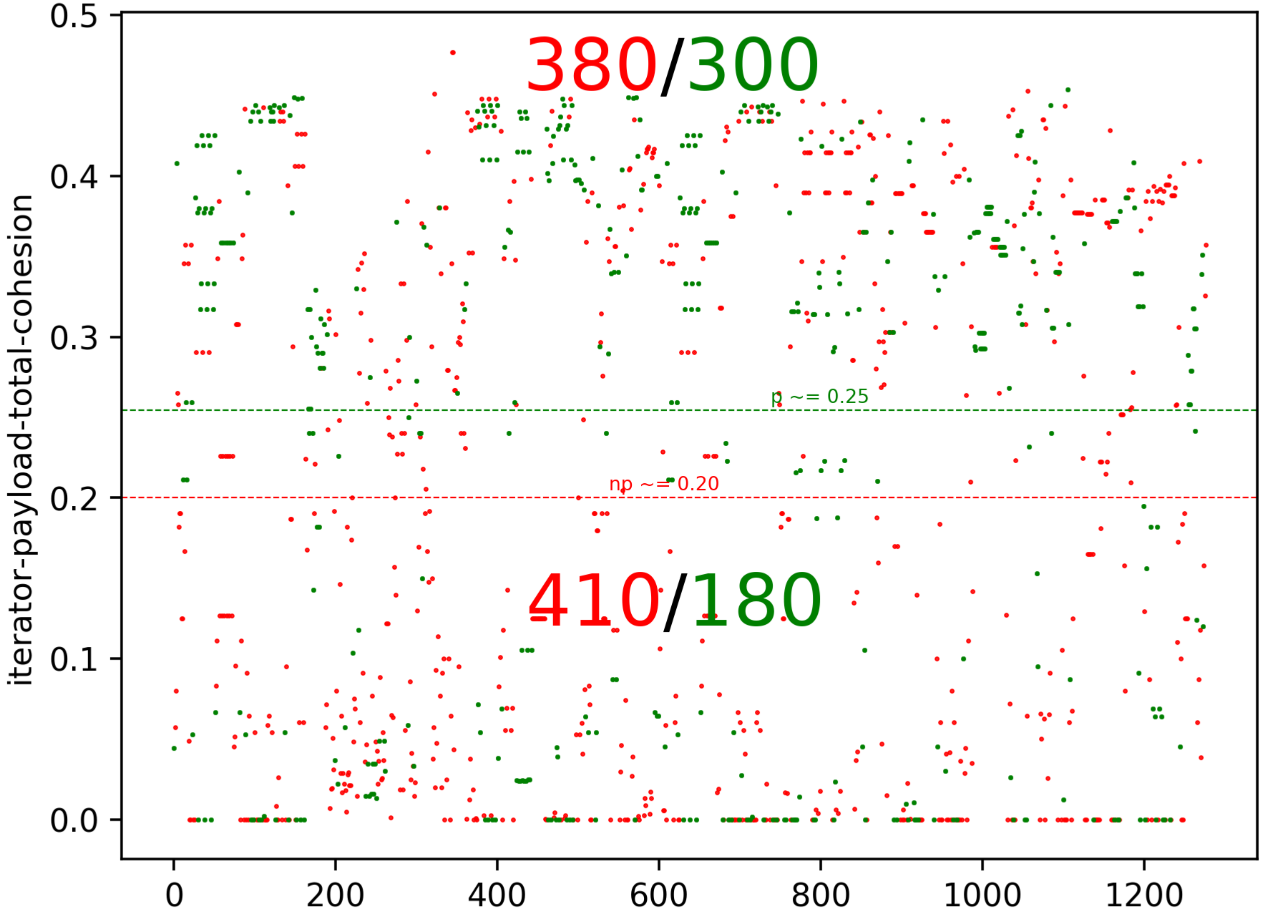
- Despite the fact, that both ***loop critical payload fraction*** and ***loop proper SCCs number*** metrics reflect essentially the same loop property (being just two different forms of it), ***loop critical payload fraction*** appears to capture parallelizability property better, with higher concentration of parallelizible (green) loops at the bottom of its value range (what correlates with the basic intuition behind the metric)

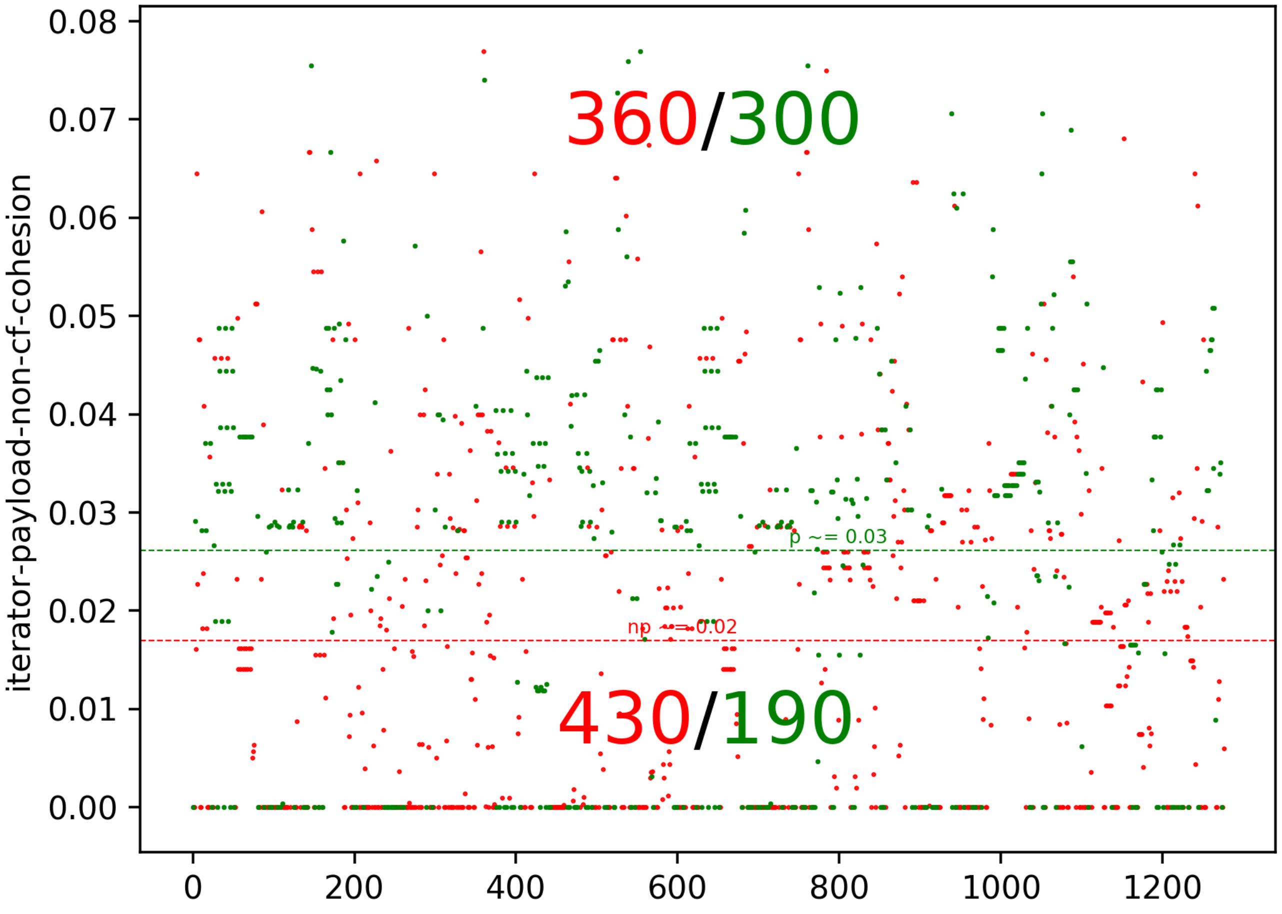
Loop Cohesion Metrics [1]



- Mean values of loop ***critical payload cohesion*** metric on the set of parallelizible (green) and non-parallelizible (red) loops are approximately the same and there are no obvious visible correlations between loop parallelizability and dots distributions for both ***total*** and ***non-control-flow*** cohesion metrics

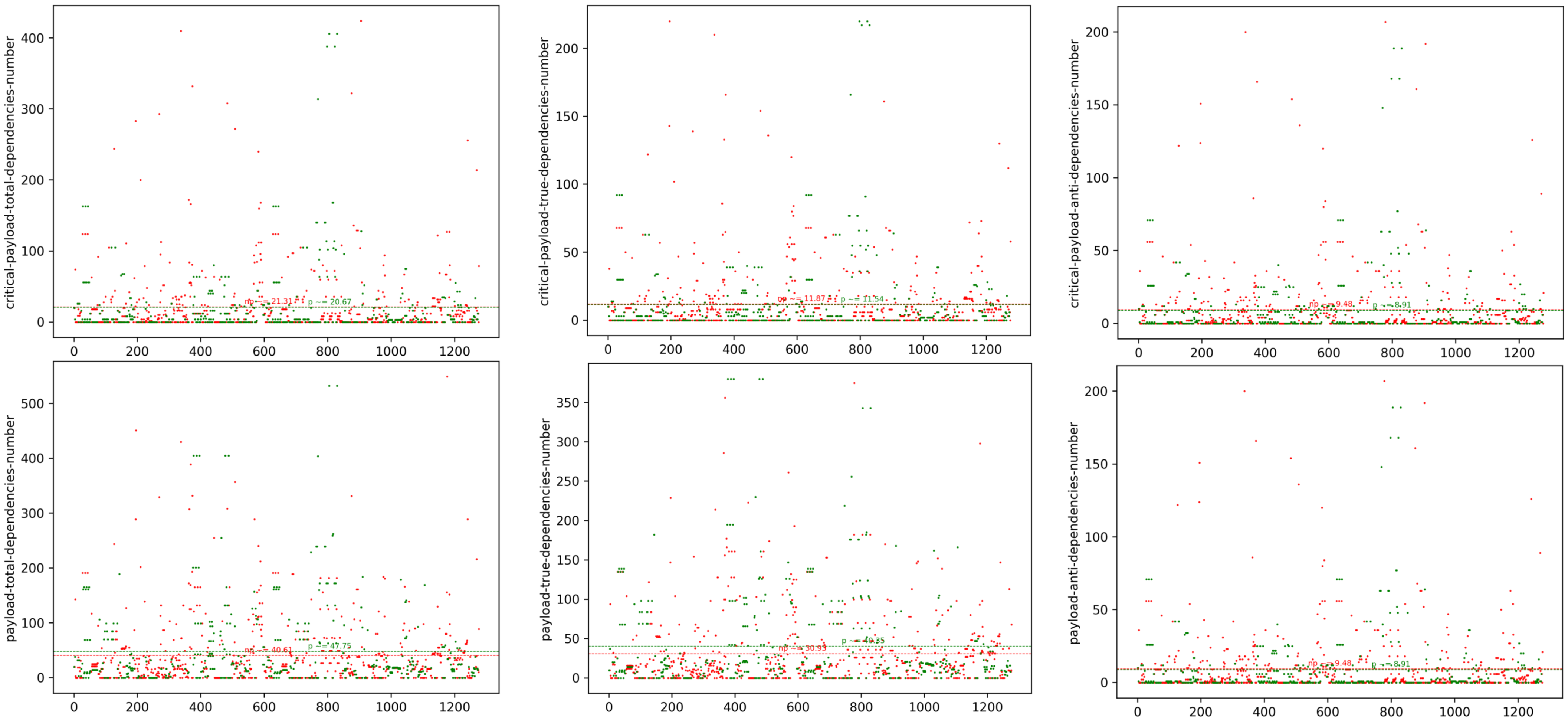
Loop Cohesion Metrics [2]





- For loop ***iterator payload cohesion*** metrics, parallelizible (green) dots appear to clutter at the top and in the middle of the range of metric values for ***total*** and ***non-control-flow*** cohesion metrics correspondingly

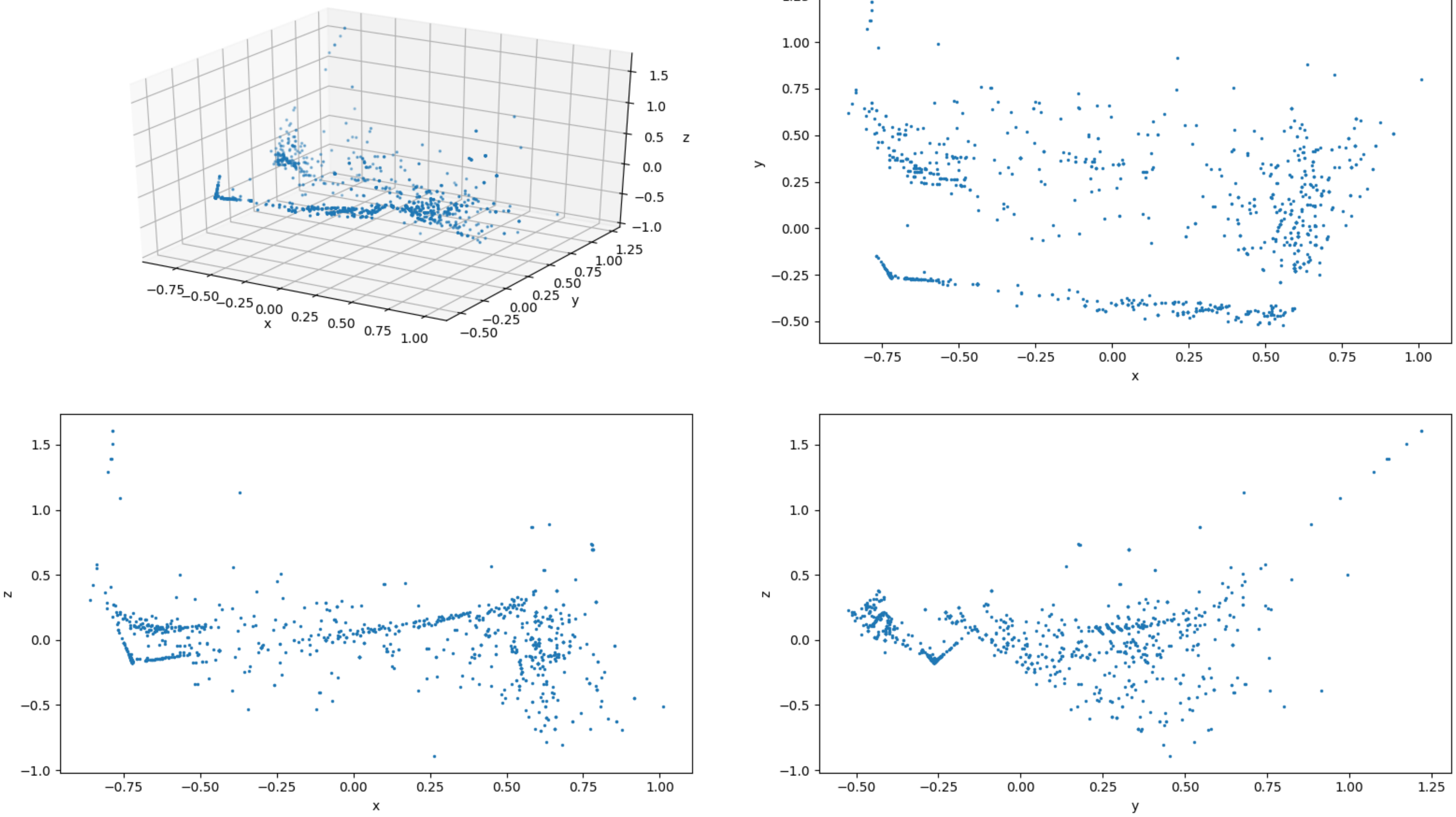
Loop Dependencies Number Metrics



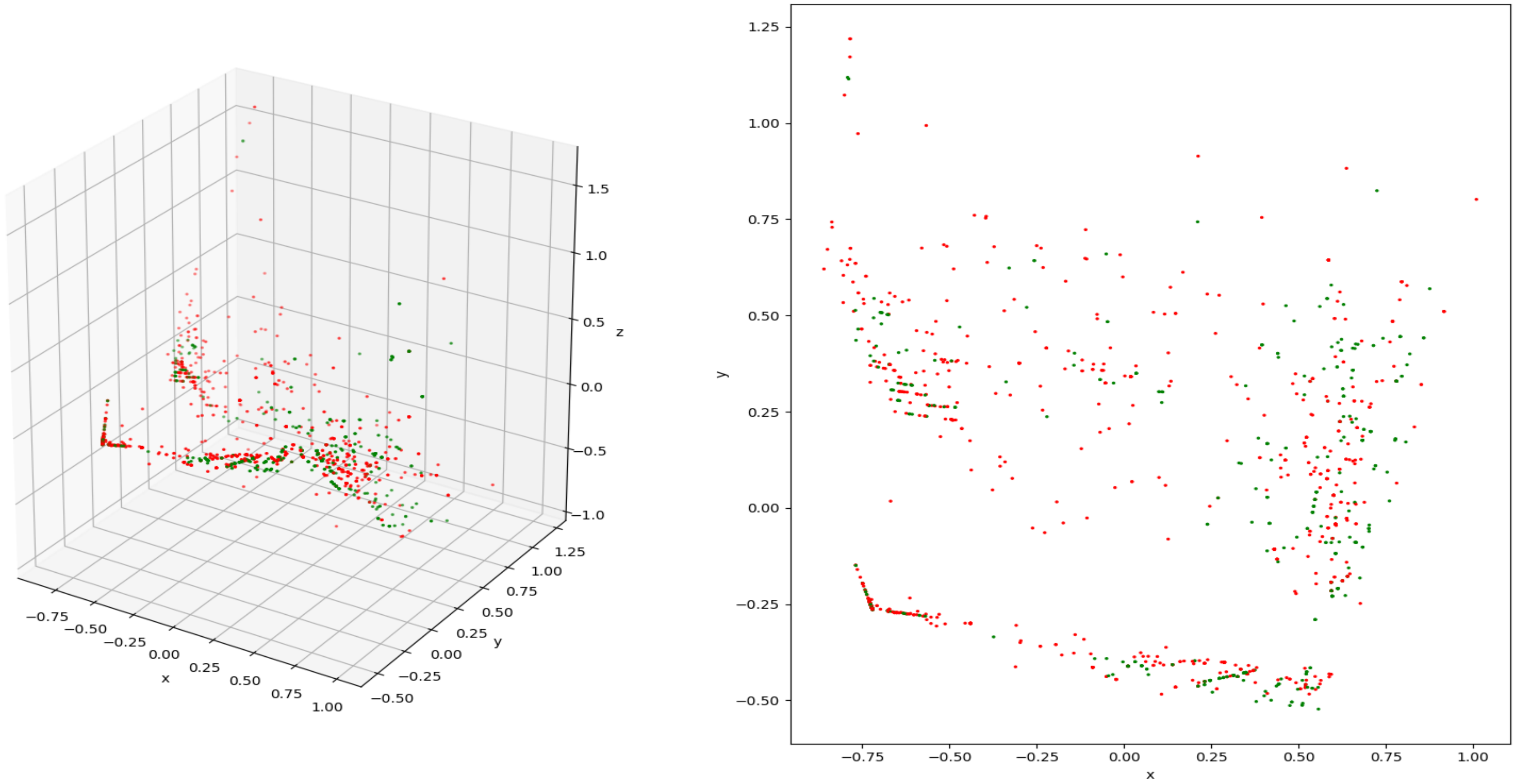
- Loop ***dependencies number*** metrics do not appear to correlate with loop parallelizability property in any possible way

Principal Component Analysis (PCA)
&
K-Means clustering

Loop Metric 13D vectors projection onto 3D space with PCA

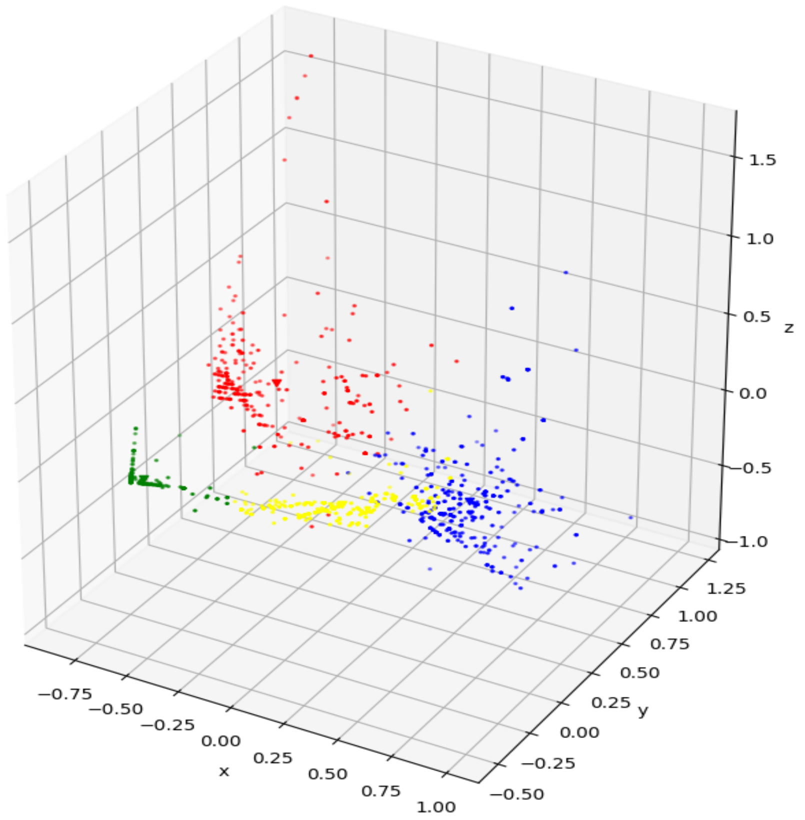


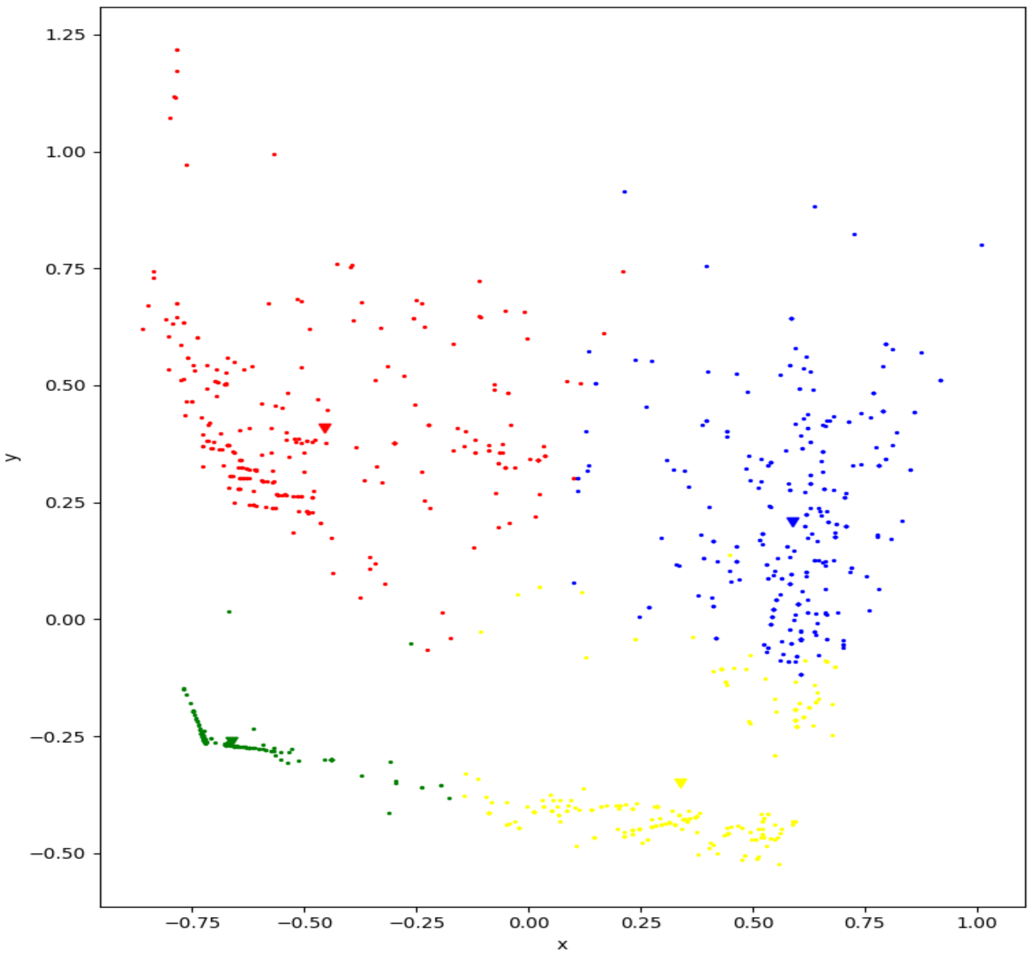
Loop Metric 13D vectors projection onto 3D space



- Loop parallelizability property does not seem to correlate with projection’s structural and spacial properties: all condensed loop point clots have both parallelizible (green), as well as non-parallelizible (red) loops

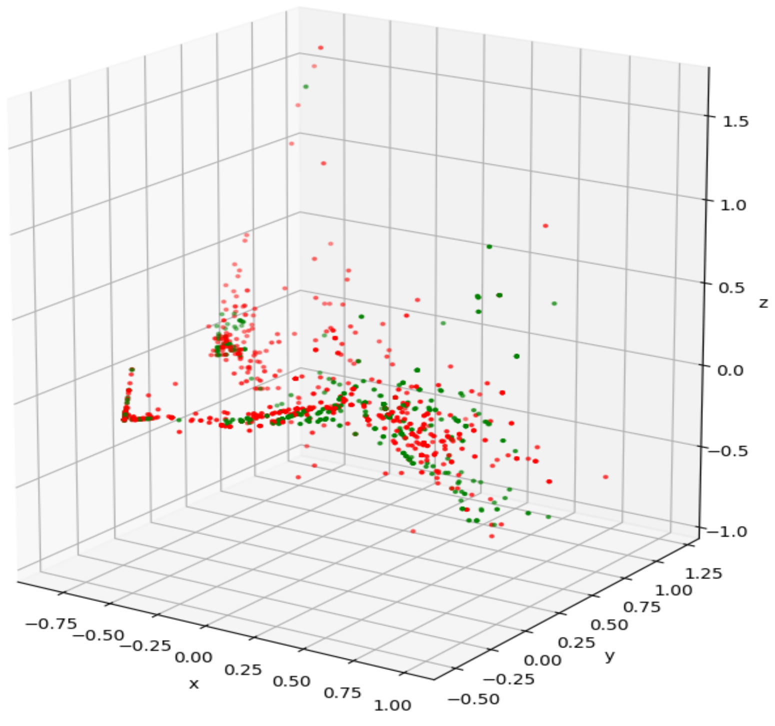
K-Means (K=4) Clustering

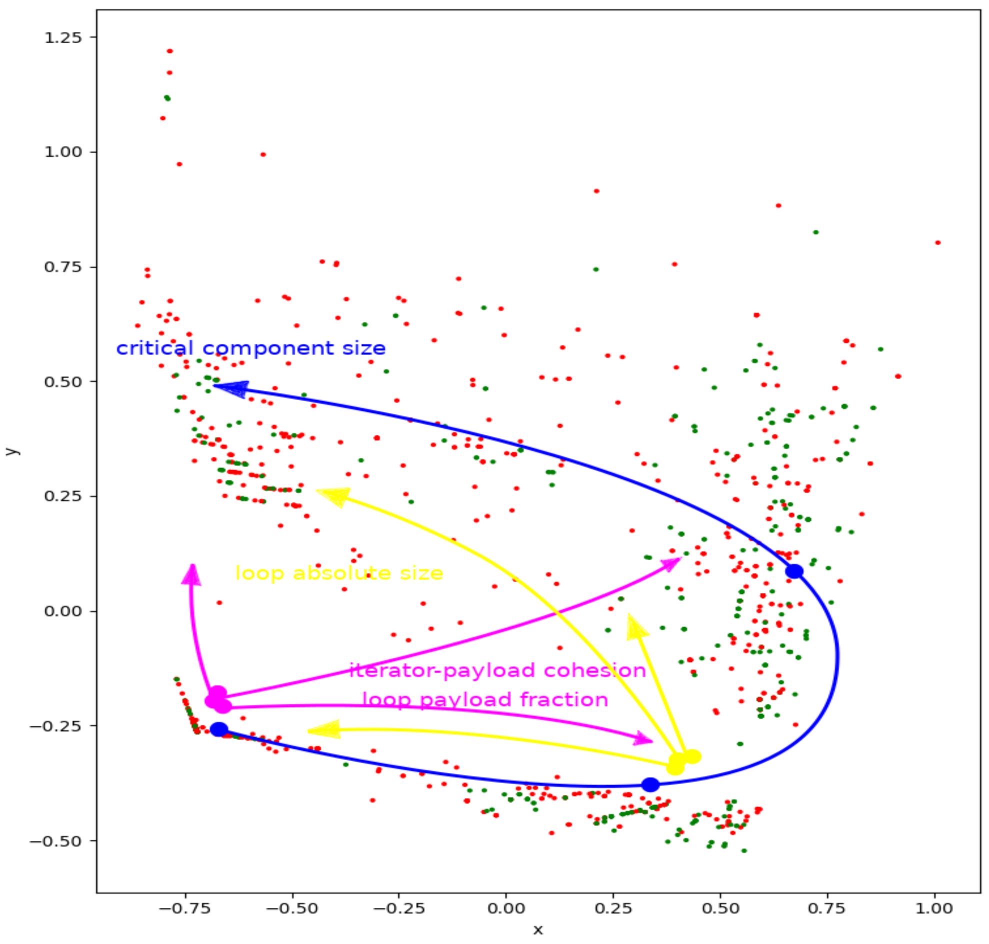




cluster	critical payload total cohesion	critical payload non cf cohesion	iterator payload total cohesion	iterator payload non cf cohesion	payload total dependencies number	payload true dependencies number	payload anti dependencies number	loop absolute size	loop payload fraction	loop proper sccs number	loop critical payload fraction	critical payload total dependencies number	critical payload true dependencies number	critical payload anti dependencies number
0	0.02	0.02	0.09	0.01	97.47	51.62	44.06	174.37	0.22	1.52	0.69	91.74	46.47	44.06
1	0.00	0.00	0.02	0.00	0.39	0.36	0.02	149.43	0.04	0.01	0.00	0.06	0.03	0.02
2	0.12	0.12	0.36	0.03	111.02	94.80	15.34	104.51	0.76	1.61	0.21	39.08	23.43	15.34
3	0.01	0.01	0.35	0.04	46.86	44.70	1.97	61.17	0.68	0.33	0.02	4.81	2.84	1.97

Metric values distribution across 3D projection





cluster	critical payload total cohesion	critical payload non cf cohesion	iterator payload total cohesion	iterator payload non cf cohesion	payload total dependencies number	payload true dependencies number	payload anti dependencies number	loop absolute size	loop payload fraction	loop proper sccs number	loop critical payload fraction	critical payload total dependencies number	critical payload true dependencies number	critical payload anti dependencies number
0	0.02	0.02	0.09	0.01	97.47	51.62	44.06	174.37	0.22	1.52	0.69	91.74	46.47	44.06
1	0.00	0.00	0.02	0.00	0.39	0.36	0.02	149.43	0.04	0.01	0.00	0.06	0.03	0.02
2	0.12	0.12	0.36	0.03	111.02	94.80	15.34	104.51	0.76	1.61	0.21	39.08	23.43	15.34
3	0.01	0.01	0.35	0.04	46.86	44.70	1.97	61.17	0.68	0.33	0.02	4.81	2.84	1.97

Machine Learning based Metrics Ranking

Accuracy of Parallelizability Classification Predictors

1) Single metric values and Intel compiler labels have been used to train different types of classifiers available in Scikit-Learn Python library. K-fold (K=10) cross validation technique has been used. All computed accuracies have been averaged to a single mean value.

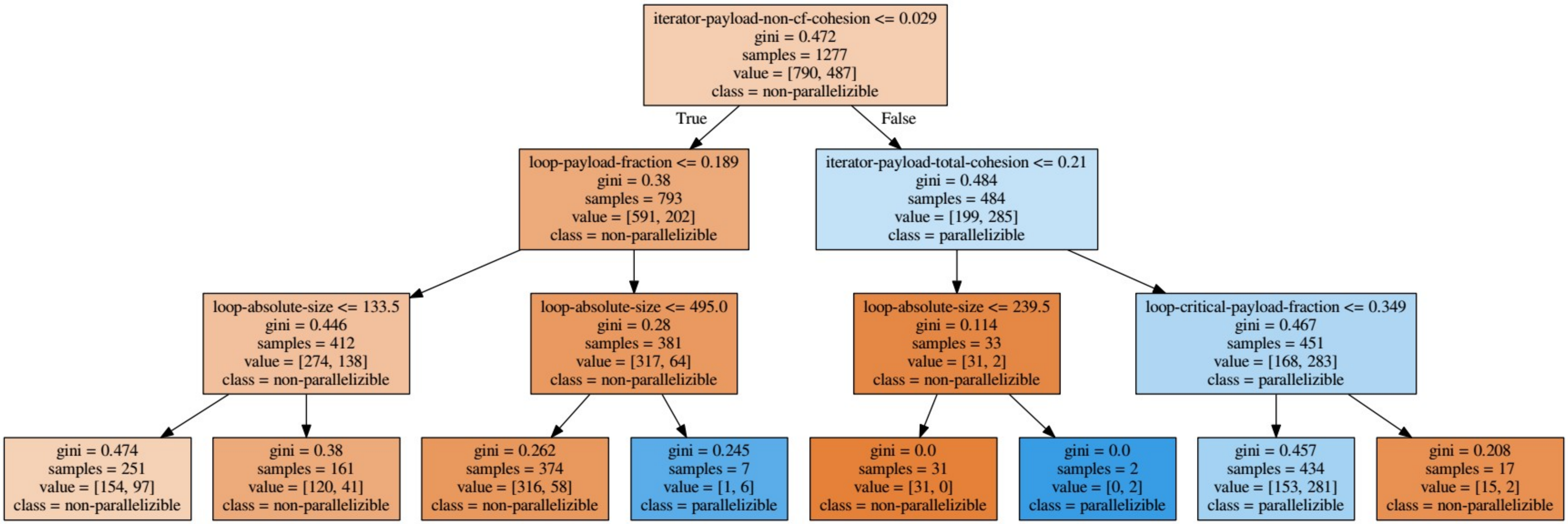
single metrics / Machine Learning algorithm	loop absolute size	loop payload fraction	loop proper sccs number	loop critical payload fraction	iterator payload total cohesion	iterator payload non-CF cohesion	critical payload total cohesion	critical payload non-CF cohesion	payload total dependencies number	payload true dependencies number	payload anti dependencies number	critical payload total dependencies number	critical payload true dependencies number	critical payload anti dependencies number
SVN	66%	62%	62%	62%	62%	62%	62%	62%	66%	67%	67%	67%	66%	67%
Decision Tree	67%	69%	62%	68%	72%	71%	67%	66%	68%	66%	66%	65%	66%	66%
MPL	62%	62%	62%	62%	62%	62%	62%	62%	62%	62%	63%	64%	64%	63%

2) Single metric values and Intel compiler labels have been used to train different types of classifiers available in Scikit-Learn Python library. K-fold (K=10) cross validation technique has been used. All computed accuracies have been averaged to a single mean value.

metric sets / Machine Learning algorithm	loop-proportions + iterator-payload-cohesion + critical-payload-cohesion + payload-dependencies-number + critical-dependencies-number	iterator-payload-total-cohesion + iterator-payload-non-cf-cohesion + loop-critical-payload-fraction + loop-payload-fraction	loop-proportions + iterator-payload-cohesion + critical-payload-cohesion	loop-proportions + payload-dependencies-number + critical-dependencies-number	iterator-payload-cohesion + critical-payload-cohesion + payload-dependencies-number + critical-dependencies-number	loop-proportions + iterator-payload-cohesion + critical-payload-cohesion + critical-dependencies-number	loop-proportions + iterator-payload-cohesion + critical-payload-cohesion + payload-dependencies-number
SVN	75%	62%	62%	75%	74%	71%	74%
Decision Tree	77%	77%	77%	75%	77%	77%	77%
MPL	62%	62%	62%	62%	62%	62%	62%

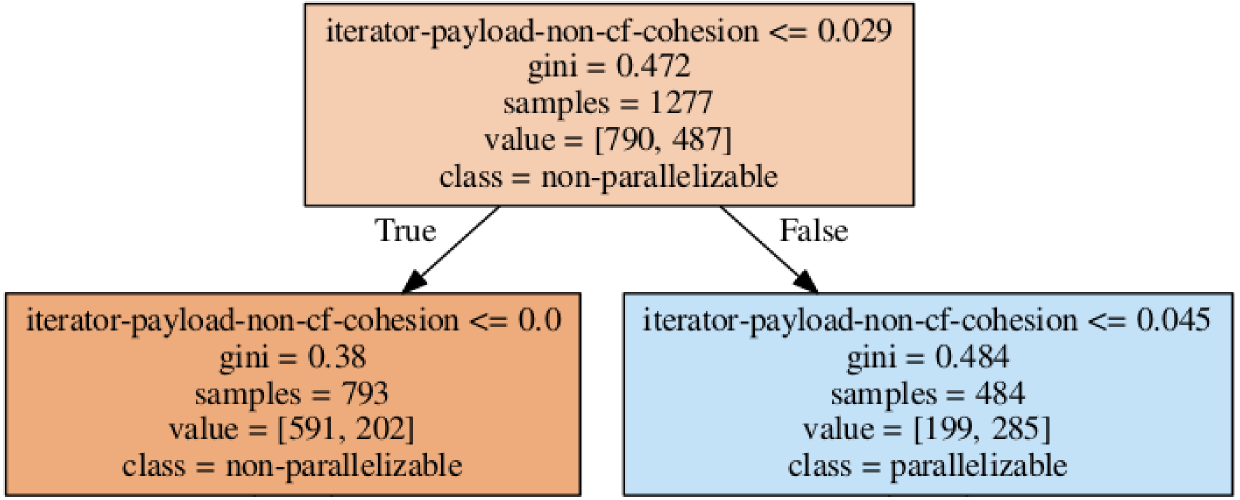
Decision Tree based Parallelizability Classification

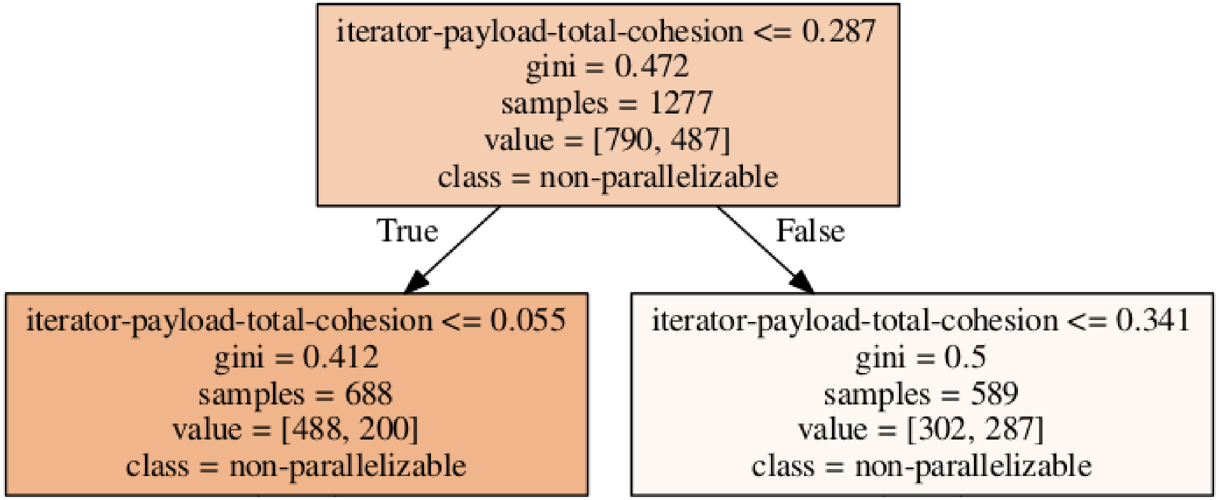
- Decision Tree has been fitted to the collected data with metric values and ICC parallelizability labels
- Decision Tree building algorithm places at the root condition, which decreses entropy the most and is the most effective in the classification task
- ***Iterator/Payload non-CF Cohesion*** metric has been ranked as the best for the task

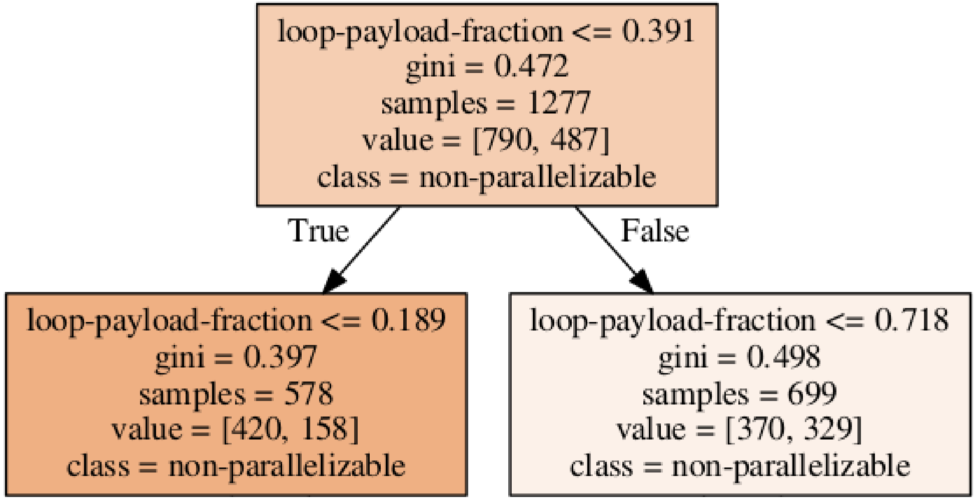


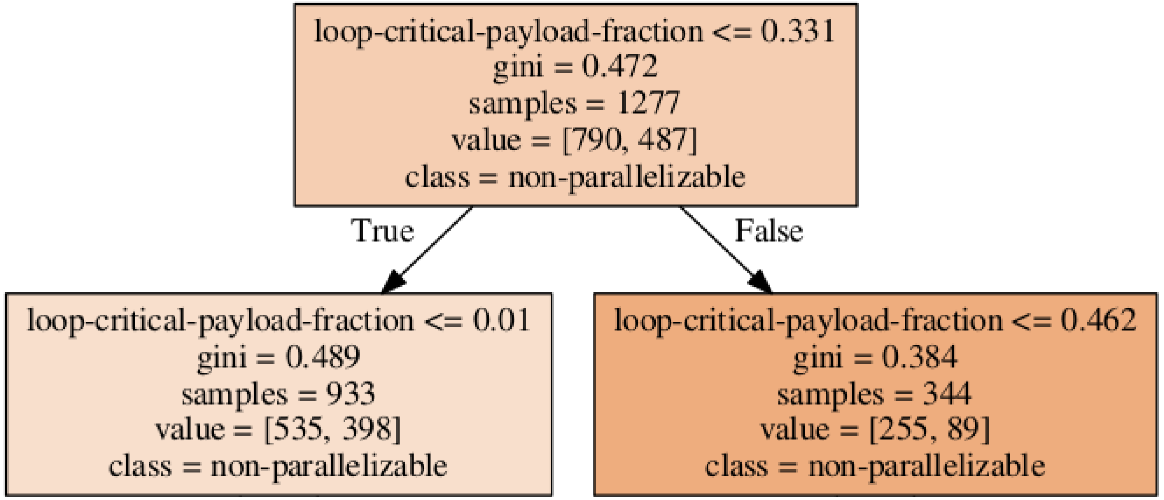
Decision Tree based Parallelizability Classification

- Decision Tree based algorithm can also be applied to get optimal decision boundaries for other correlating metrics









Results [1]

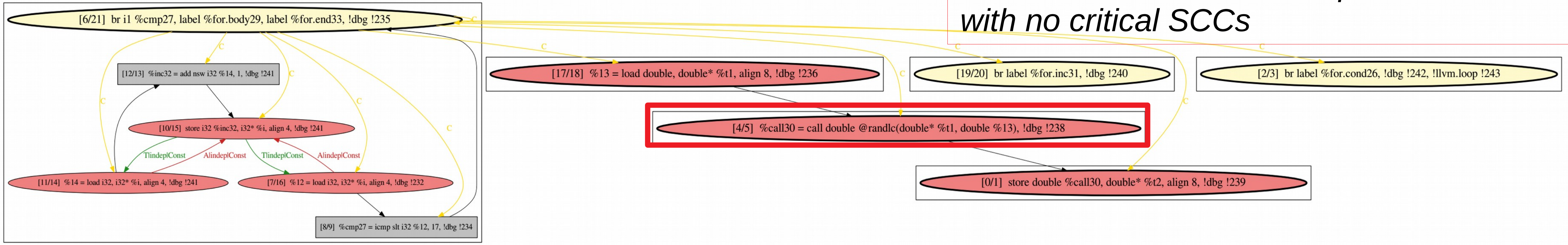
- If we combine all metric evaluation methods and round up the results, we get the following picture:

All proposed metrics have been <u>*ranked*</u> according to their correlation with loop parallelizability property:

1) ***Iterator/Payload non-CF Cohesion***
2) ***Iterator/Payload total Cohesion*** and ***Iterator/Payload Fraction***
3) ***Loop Critical Payload Fraction*** and ***Loop Absolute Size***
4) ***Payload Dependencies Number***
5) ***Critical Payload Dependencies Number***

- Despite the presense of some correlations current software metrics for parallelism cannot provide precise parallelizability decision boundary and are quite blurred
- Even ***Critical SCCs Number/Fraction*** metrics (which seemed to directly represent parallelization constraining element) did not show significant correlations

Call to *randlc() - compiler has to be conservative and does not parallelize this loop with no critical SCCs*



Results [2]

Proposed metrics work the best in their combination, but still give only ***probabilistic answers***. For example, loop conformance to the following system of inequalities (derived form fitted Decision Tree) predicts, that this loop is going to be 65% parallelizible and 35% non-parallelizible:

$$\begin{cases} \textit{iterator-payload non-cf cohesion} \geq 0,029 \\ \textit{iterator-payload total cohesion} \geq 0.21 \\ \textit{loop-critical-payload-fraction} \leq 0.349 \end{cases}$$

```
for (k = 1; k <= grid_points[2]-2; k++) {
    for (j = 1; j <= grid_points[1]-2; j++) {
        for (i = 1; i <= grid_points[0]-2; i++) {
            for (m = 0; m < 5; m++) {
                add = rhs[k][j][i][m];
                rms[m] = rms[m] + add*add;
            }
        }
    }
}
```

$$rms[m] = \sum_{k \in grid\_points[2]-2} \sum_{j \in grid\_points[1]-2} \sum_{i \in grid\_points[0]-2} rhs[k][j][i][m]$$

Intel Compiler did not parallelize this loop due to conservative assumption that *rhs[]* and *rms[]* arrays address the same memory location and introduce *TRUE* and *ANTI* cross-iteration dependencies

Above inequalities system classifies this loop as more likely parallel, than not

Project Limitations and Future Work

- Only NAS benchmarks have been used (these benchmarks do not contain any data structures besides regular linear arrays)

- Intel Compiler has been used as a parallelizability expert

- Proposed metrics require further tuning

Discovery Project
&
Data-Centric Parallelilization

PhD Vision

Student:

Aleksandr Maramzin, s1736883

Supervision team:

Björn Franke, Michael O'Boyle, Kenneth Heafield

EPSRC
Engineering and Physical Sciences Research Council

EPSRC Centre for Doctoral Training in Pervasive Parallelism

THE UNIVERSITY of EDINBURGH informatics

Thank you!