Machine Learning Assisted Parallelism Discovery

Students: Aleksandr Maramzin, Christos Vasiladiotis
Supervisory team: Prof. Björn Franke, Prof. Murray Cole, Prof. Michael O'Boyle

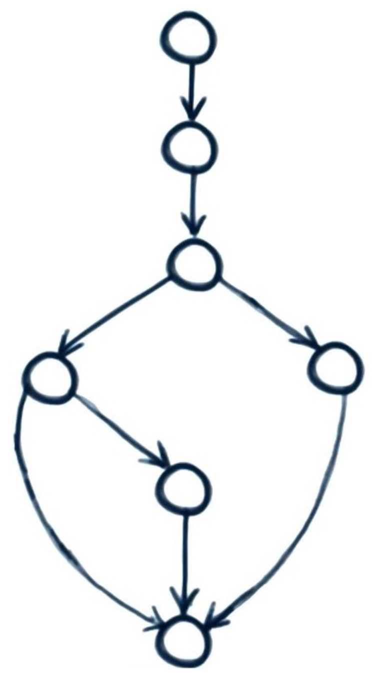
Abstract

All modern hardware is highly parallel, but in order to effectively utilise all these available resources software must be parallel as well. Traditionally software parallelisation has been done either manualy or with the help of compiler's static analysis. While being effective manual parallelisation requires from a programmer skills and expertise. Automatic parallelisation based on static compiler analysis is overly conservative and limited.

Limitations of static analysis have been traditionally tackled with the use of additional dynamic profile-guided methods, but they require to actually run a program and are tied to a particular program input.

In this project we investigate a relatively new semi-automatic approach to the task of program parallelisation: machine learning assisted one. We train an oracle to predict parallelisability property of SNU NAS benchmark loops. Our oracle achieves generalised prediction accuracy of 93% across all SNU NAS benchmark loops and in the right use scenario increases parallelising coverage of state-of-the-art Intel C/C++ compiler's static analysis from 86% to 99%. The right mapping and tuning of these additinally exposed opportunities materialises into the real perfromance increase.

Software quality metrics



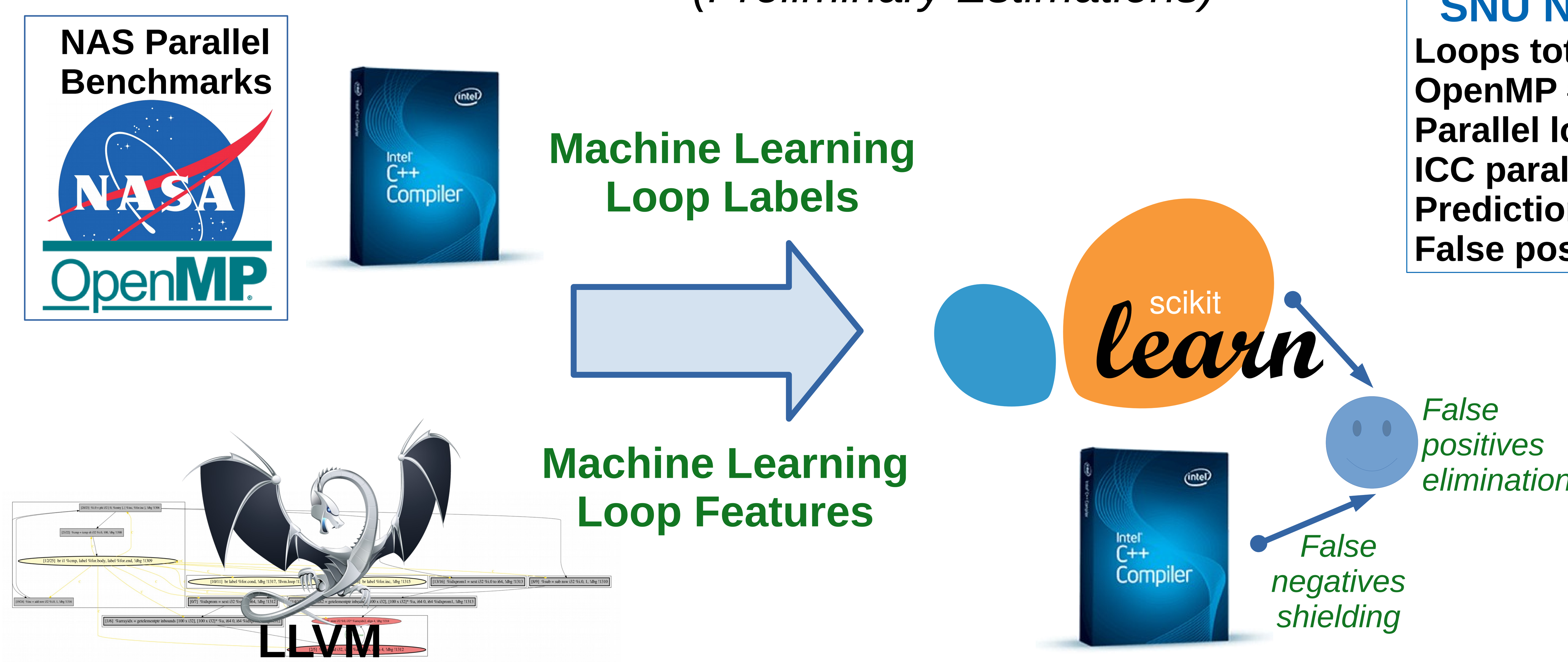
Initial motivation behind software metrics for parallelism and their later use as machine learning features was the vast body of work done in the field of software quality metrics.

The most illustrative of those is probably the cyclomatic complexity. ***Cyclomatic Complexity*** (***CC***) (*Thomas J. McCabe [1976]*) is based on the control flow graph (CFG) of the section of the code and basically represents the number of linearly independent paths through that section.

Mathematically, cyclomatic complexity of the section of the code is defined as $CC = E - N + 2P$, where E is the number of edges, N is the number of nodes, P is the number of connected components in the section's CFG.

Machine Learning Methodology

(Preliminary Estimations)



SNU NAS Benchmarks

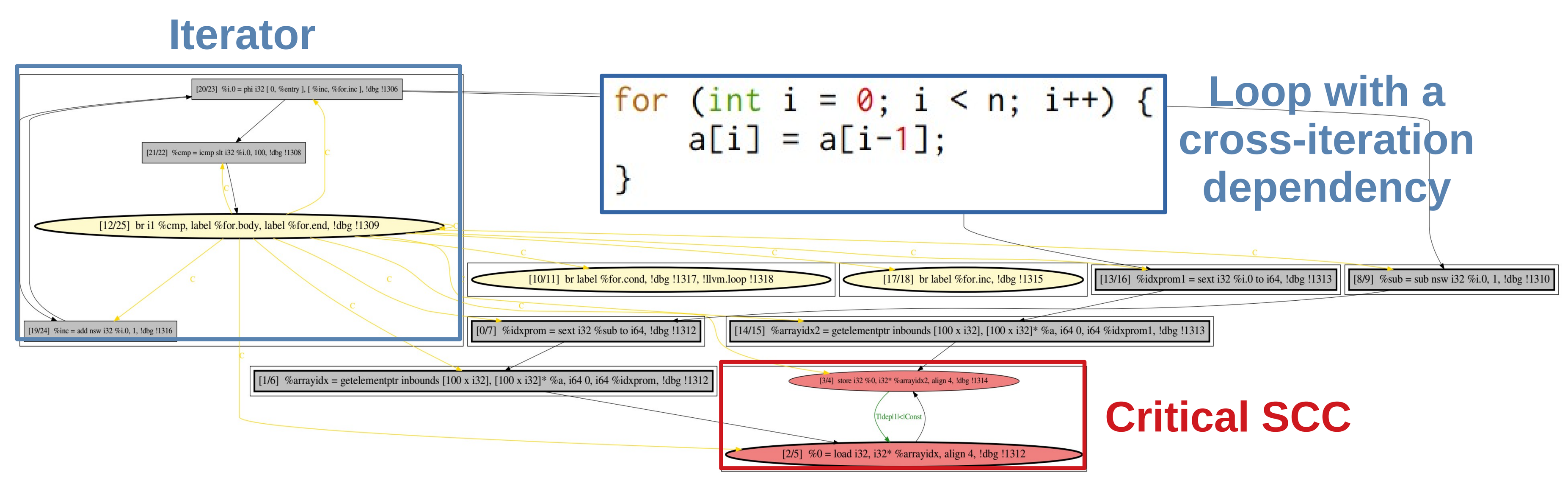
Loops total number: ***1420***
OpenMP #pragmas: ***211***
Parallel loops: ***901***
ICC parallel loops: ***812***
Prediction accuracy: ***93%***
False positives: ***3.5%***

Feedback Scheme

Increases SNU NAS available parallelism utilisation by ICC from 90% to 99%

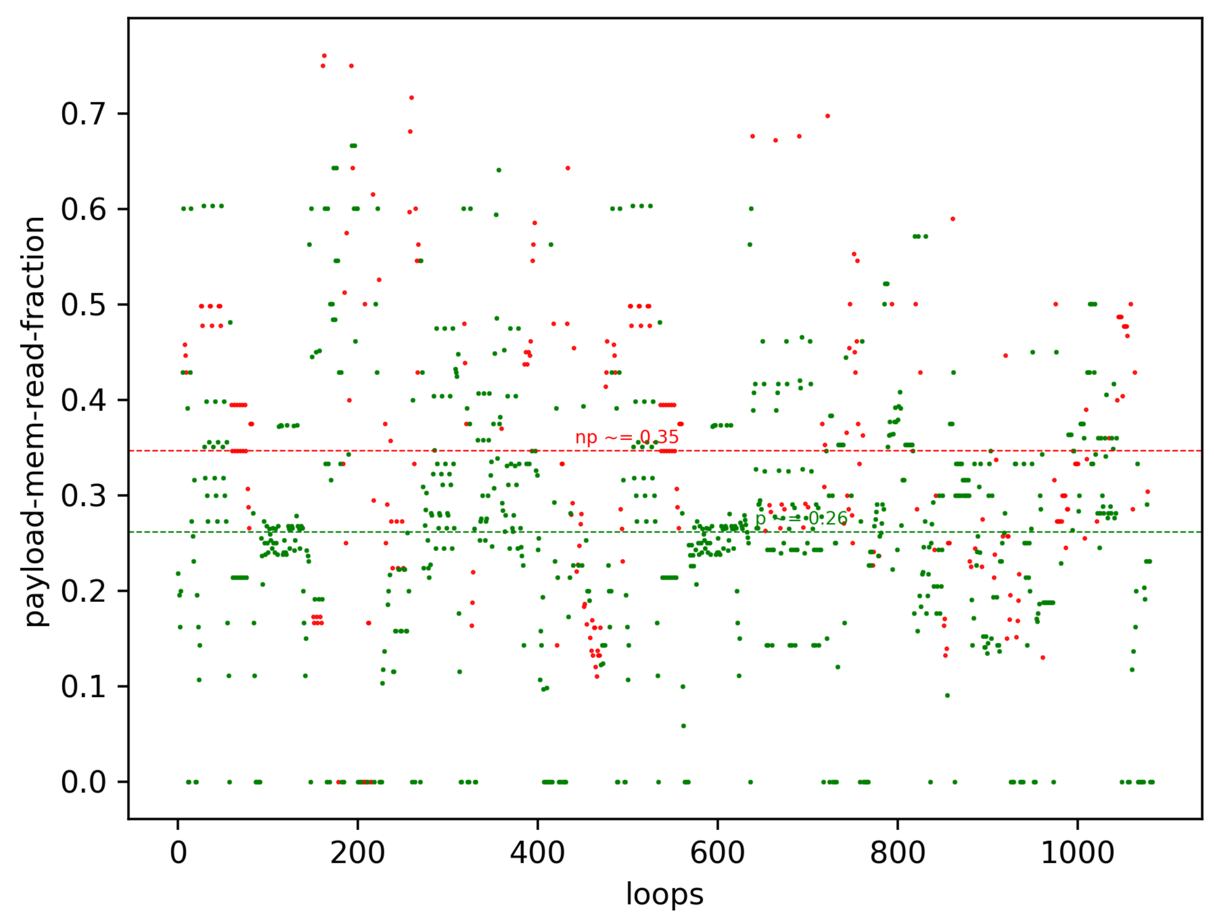
Machine Learning Loop Features

(Loop Parallelisability Metrics)



Software parallelisability metrics [*40*]

Metric Group	Metric	Metric Definition
Loop Proportions	Absolute Size	*Number of LLVM IR instructions in a whole loop*
	Payload Fraction	
	Proper SCCs Number	*Number of SCCs with more than one LLVM IR instruction in a payload of a loop*
	Critical Payload Fraction	
Loop Dependencies Number	Payload Dependencies Number	*Number of PDG edges in a payload (**True**, **Anti**, **Output** and **Total**)*
	Critical Payload Dependencies Number	
Loop Cohesion	Iterator/Payload Cohesion	*Normalised number of edges between iterator and payload*
	Critical/Regular Payload Cohesion	
Loop Instructions Nature	Call instructions count	*The number of LLVM IR call instructions in a loop*
	Branch instructions count	



All parallelisability metrics have certain ground, the basic intuition behind them and correlate with parallelisability of loops as one might expect, but correlations are generally weak and are not sufficient to draw a fine line between parallelisible and non-parallelisible loops.

As our research shows, SNU NAS benchmark loops are quite diverse and in order to capture their true parallelisability we **must use all metrics harnessed altogether in a machine learning model**.