“Smart” Software Parallelisation Assistant

Student: Aleksandr Maramzin
Supervisory team: Prof. Björn Franke, Prof. Murray Cole

Project Overview

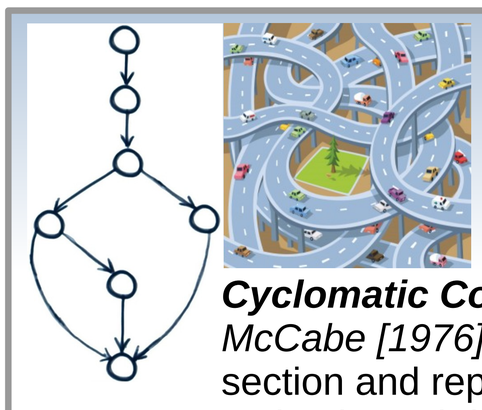
Despite decades of research into parallelising compiler technology, software parallelisation remains a largely manual task where the key resource is expert time. Typically, a programmer has to identify those program loops, which contribute the most to the ultimate program execution time, analyse if they are parallelisable and if so transform the program into a parallel form. All these tasks require a programmer to posess a set of extra skills and are time consuming. In this project we propose a **methodology and assistant tool** [1] which make better use of expert time by guiding their effort directly towards those loops, where the largest performance gains can be expected while keeping analysis and transformation effort at a minimum.

Our novel assistant tool is based on a machine learning (ML) model of loop parallelisability, which is captured by a vector of **static program features** [2]. Assistant takes a program profile as an input and provides a programmer with a ranking of all loops in a program based on their overall (worthwhile and feasible to parallelise) merit. The ranking function combines for each loop its potential contribution to the speedup (taken from a profile) and an estimated probability for its successful parallelisation (taken from a trained ML model). We use a set of 1415 SNU NAS Parallel Benchmark (NPB) loops as a data set for ML. Despite its limited size our model demonstrates **a prediction accuracy of greater than 90%** [3]. Being less conservative than a state-of-the-art compiler our assistant extends **the amount of discovered parallelism** in the SNU NPB suite from 81% to 96% [4]. That discovery comes at a price of 6,5% of cases being false positives.

We deploy our parallelisation assistant against separate SNU NAS benchmarks and show that our methodology achieves results comparable to those of expert programmers while requiring a programmer to examine fewer loops, i.e. it requires less expert time. On average, our assistant reduces programmer efforts required to reach an expert-level performance manually by 20% and **converges to the maximum achievable performance faster** [5].

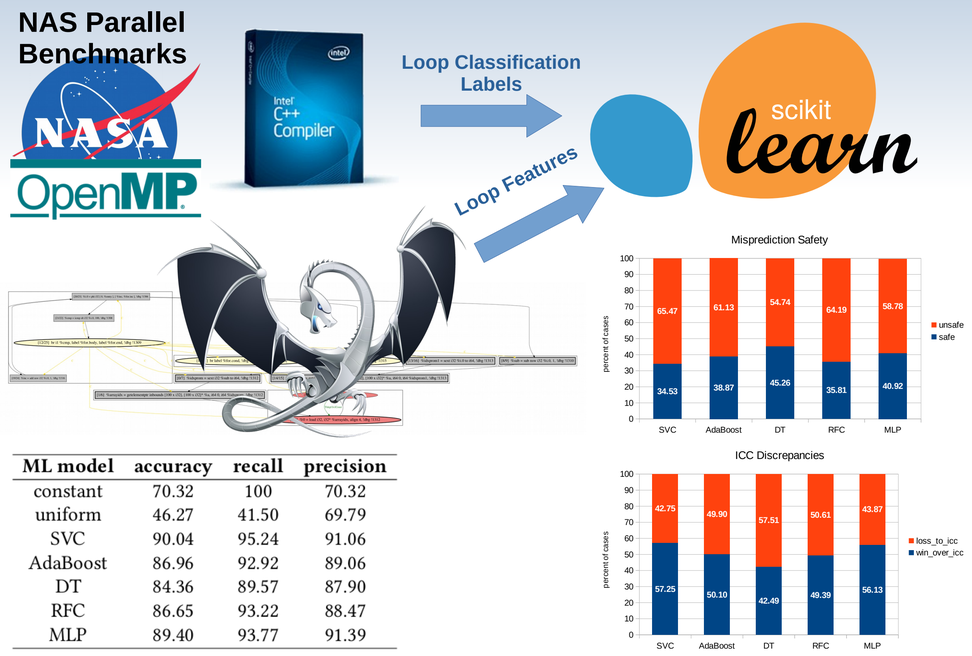
Software Quality Metrics

(the initial motivation for the project)



Cyclomatic Complexity (***CC***) (*Thomas J. McCabe [1976]*) is based on the CFG of the code section and represents the number of independent paths through it.

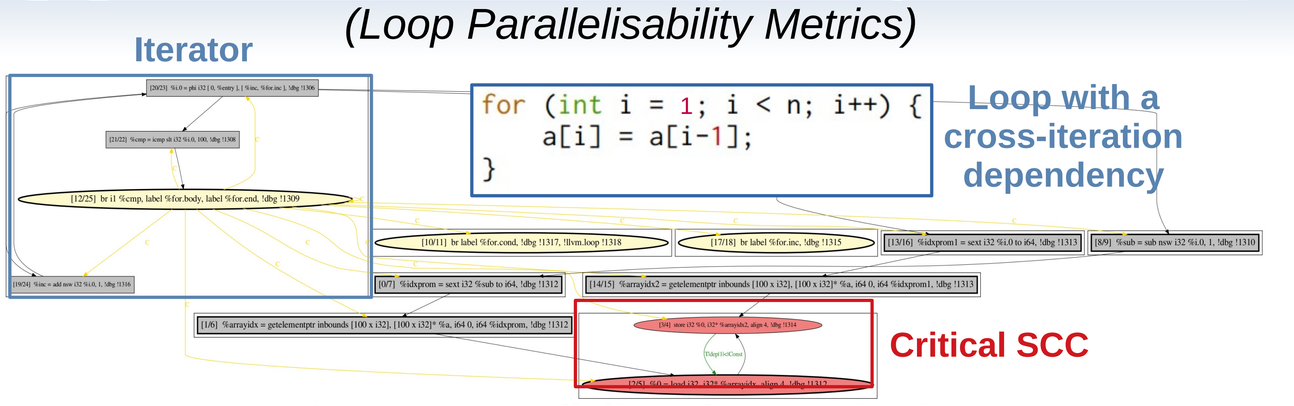
[3] Machine Learning Predictive Performance



ML model	accuracy	recall	precision
constant	70.32	100	70.32
uniform	46.27	41.50	69.79
SVC	90.04	95.24	91.06
AdaBoost	86.96	92.92	89.06
DT	84.36	89.57	87.90
RFC	86.65	93.22	88.47
MLP	89.40	93.77	91.39

[2] Machine Learning Loop Features

(Loop Parallelisability Metrics)

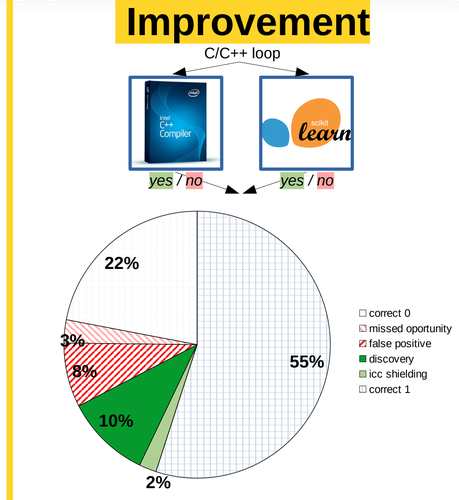


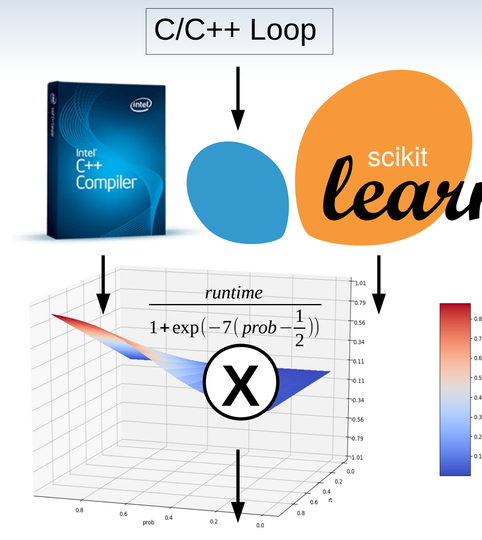
Software parallelisability metrics [*74*]

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	

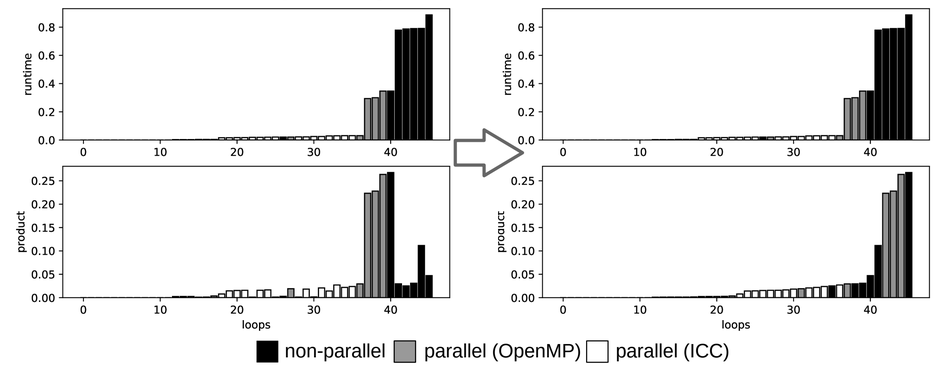
[1] Methodology and assistant tool

[4] Parallelism Discovery Improvement

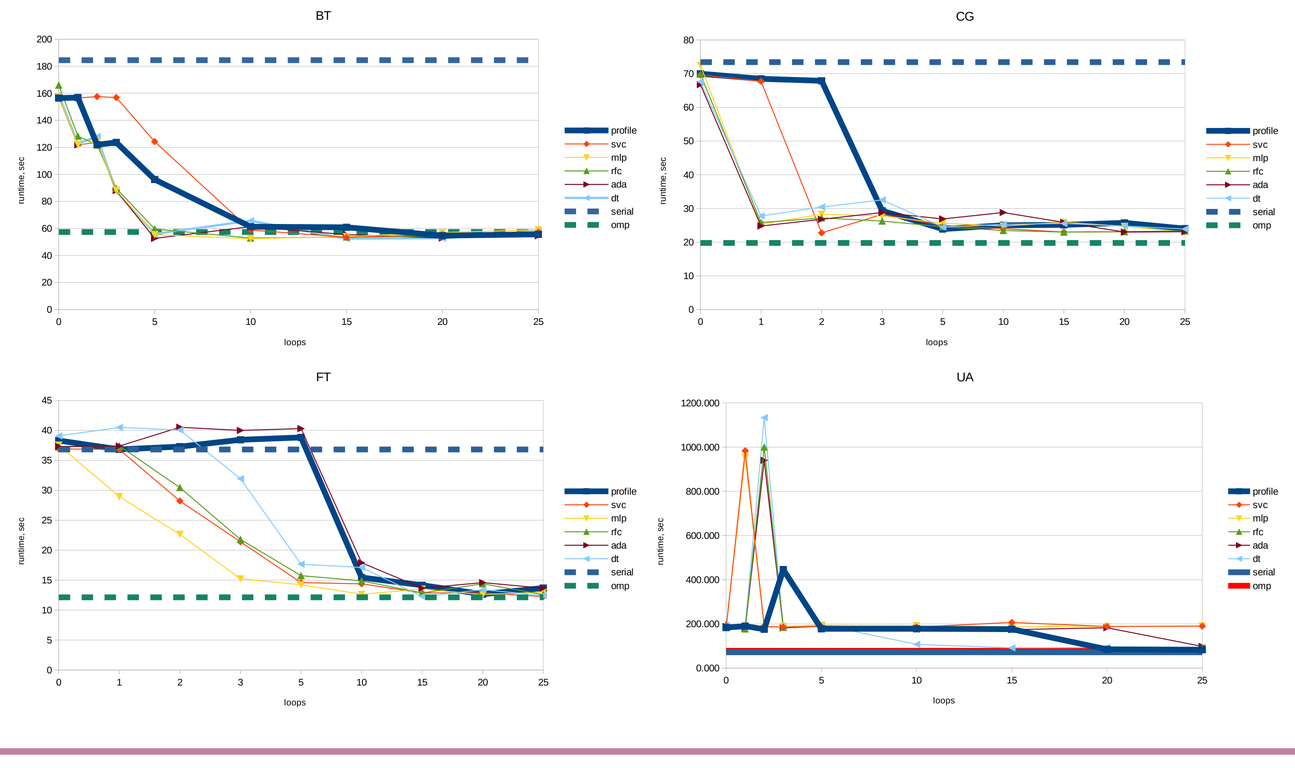




Loop Rankings: runtime order vs. product order



[5] Performance convergence curves



EPSRC Centre for Doctoral Training in Pervasive Parallelism

icsa | Institute for Computing Systems Architecture

THE UNIVERSITY of EDINBURGH informatics