“Smart” Manual Software Parallelisation Assistant

Student: Aleksandr Maramzin

Supervisory team: Prof. Björn Franke, Prof. Murray Cole, Prof. Michael O'Boyle

EPSRC Engineering and Physical Sciences Research Council

EPSRC Centre for Doctoral Training in Pervasive Parallelism

icsa Institute for Computing Systems Architecture

THE UNIVERSITY of EDINBURGH informatics

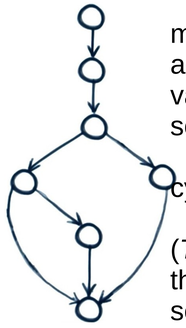
Abstract

Since automatically parallelizing compilers have failed to deliver significant performance improvements, programmers are still forced to parallelize legacy software manually for all but some niche domains. Rather than hoping for an elegant silver bullet, we acknowledge the role of a human expert in the parallelization process and train a smart parallelization assistant.

In its essence our assistant is yet another application of machine learning techniques to the field of optimizing compilers, which tries to predict the parallelisability property of program loops. Together with Intel compiler’s optimization reports we use a version of NAS Parallel Benchmarks (NPB) hand-annotated with OpenMP parallelisation pragmas to train our model. We achieve a good prediction accuracy of around 90% (baseline 40-70%) for our problem using only static program features.

But good prediction accuracy is not enough for a practical use, and hence we propose 2 application schemes where we might integrate out predictor into. The first scheme extends parallelism discovery capabilites of Intel Compiler. The second scheme takes application profile as an input and points out program loops, that are highly likely to be parallelisible and the most profitable.

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.

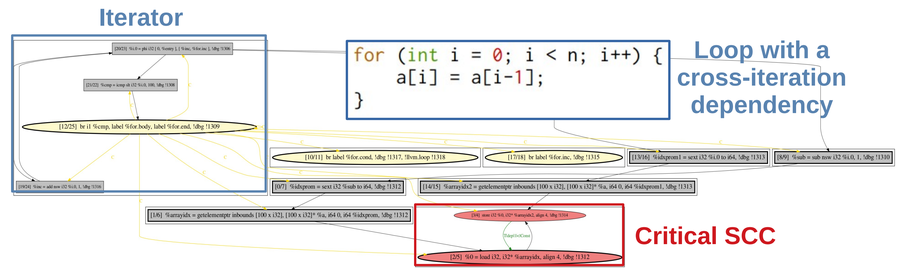
The most illustrative of those is the cyclomatic complexity.

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 Loop Features

(Loop Parallelisability Metrics)

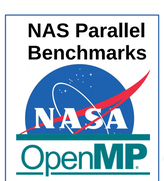


Software parallelisability metrics [***40***]

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	

Smart Assistant Training

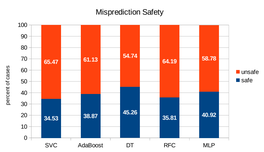
NAS Parallel Benchmarks

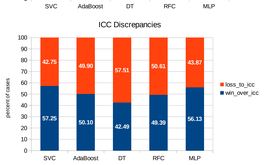


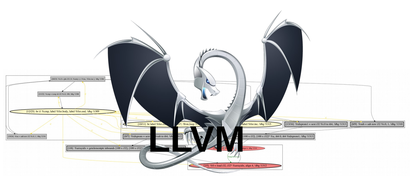
Machine Learning Loop Labels

Machine Learning Loop Features

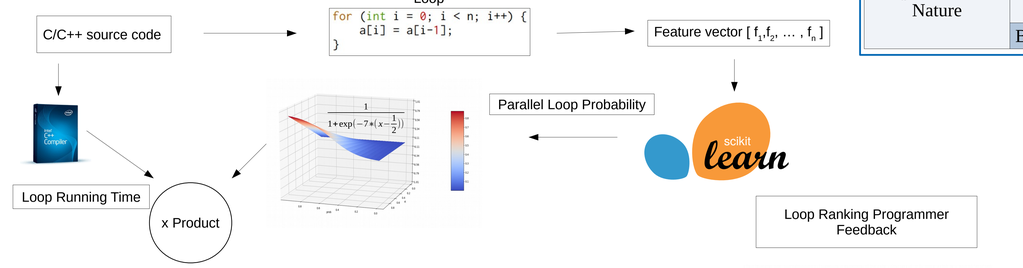
ML model	accuracy	recall	precision
constant	70.32	100	70.32
uniform	46.27	41.50	69.79
SVC	90.04	95.24	91.06
AdaBoost	86.96	92.92	89.06
DT	84.36	89.57	87.90
RFC	86.65	93.22	88.47
MLP	89.40	93.77	91.39

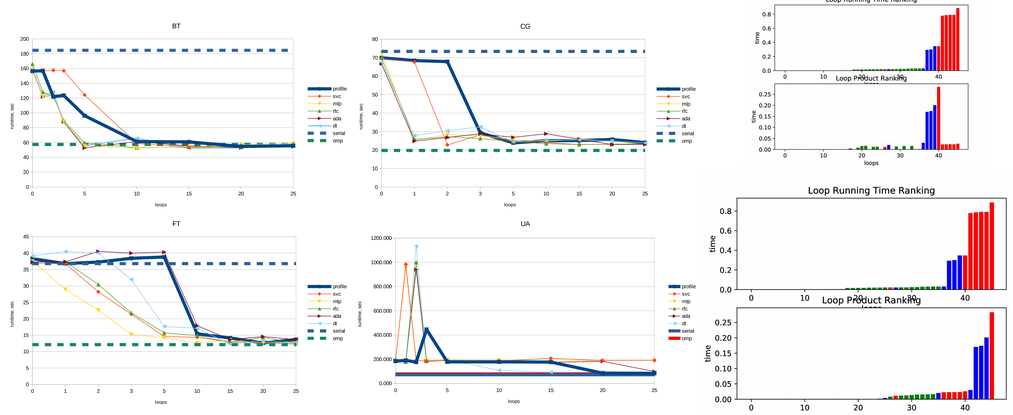




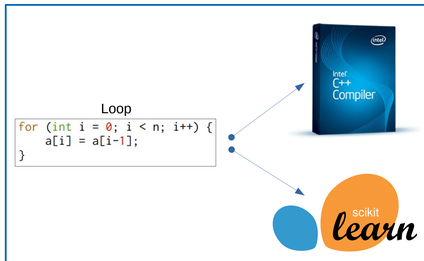


Manual Parallelisation Assistant





Machine Learning Loop Features



The scheme improves parallelism discovery capabilities of Intel Compiler. In SNU NPB benchmarks the scheme increases the amount of available parallelism that has been discovered from 82% to 96%.