Software metrics for parallelism *MSc by Research project*

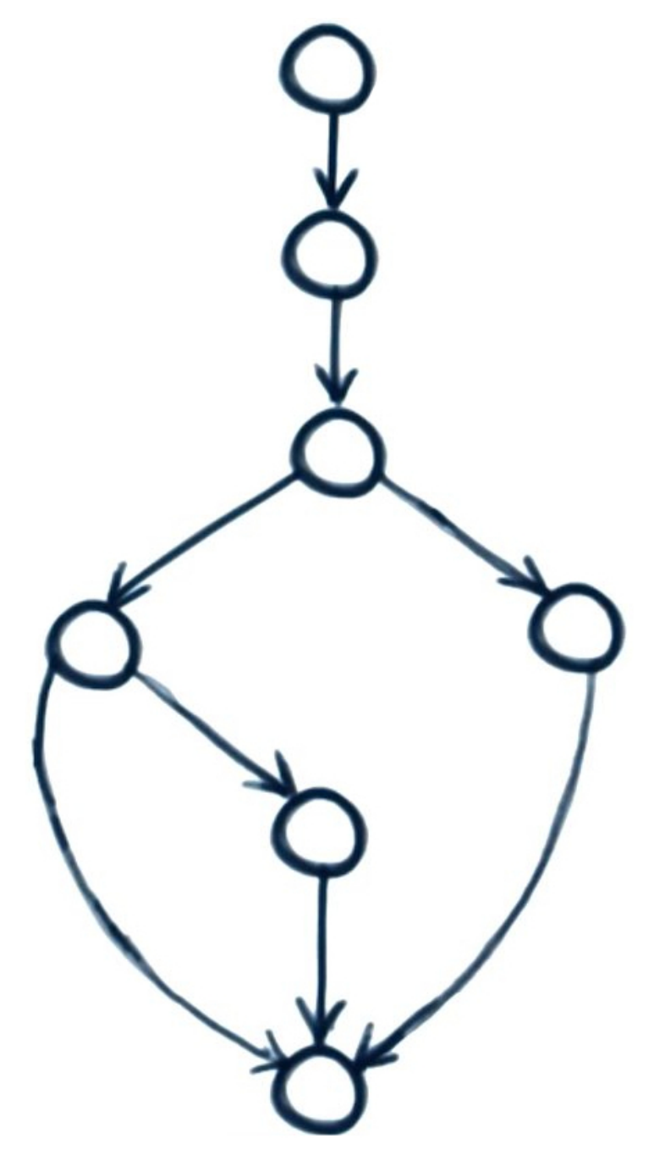
Student: Aleksandr Maramzin **Primary Supervisor:** Prof. Björn Franke **Secondary Supervisor:** Prof. Michael O'Boyle

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

In this project we investigate methods for providing programmers with real-time feedback on the quality of their code with respect to parallelisation opportunities and scalability to address short-comings before they manifest as “bad” and hard-to-parallelise code. We draw on the experience of the software engineering community and software metrics originally developed to identify “bad” sequential code, typically prone to errors and hard to maintain. The ambition of this project is to develop novel software parallelisability metrics, which can be used as quality indicators for parallel code and guide the software development process towards better parallel code.

Software quality metrics



One of the motivations for research is the vast body of work done in the field of software quality metrics. Work ranges from research results to practical applications in the form of static analysis tools and Interactive Development Environment (IDE) plug-ins.

A good example of software quality metric is cyclomatic complexity of the section of the code.

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 a 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.

Motivating example

(*Two alternative implementations of vector multiplication – different data structures, representing vectors in memory, lead to different code parallelisability)*

```
int a[1000];
int b[1000];
int c;

// initialize arrays

c = 0;
for (int i = 0; i < 1000; i++) {
    c += a[i] * b[i];
}
```

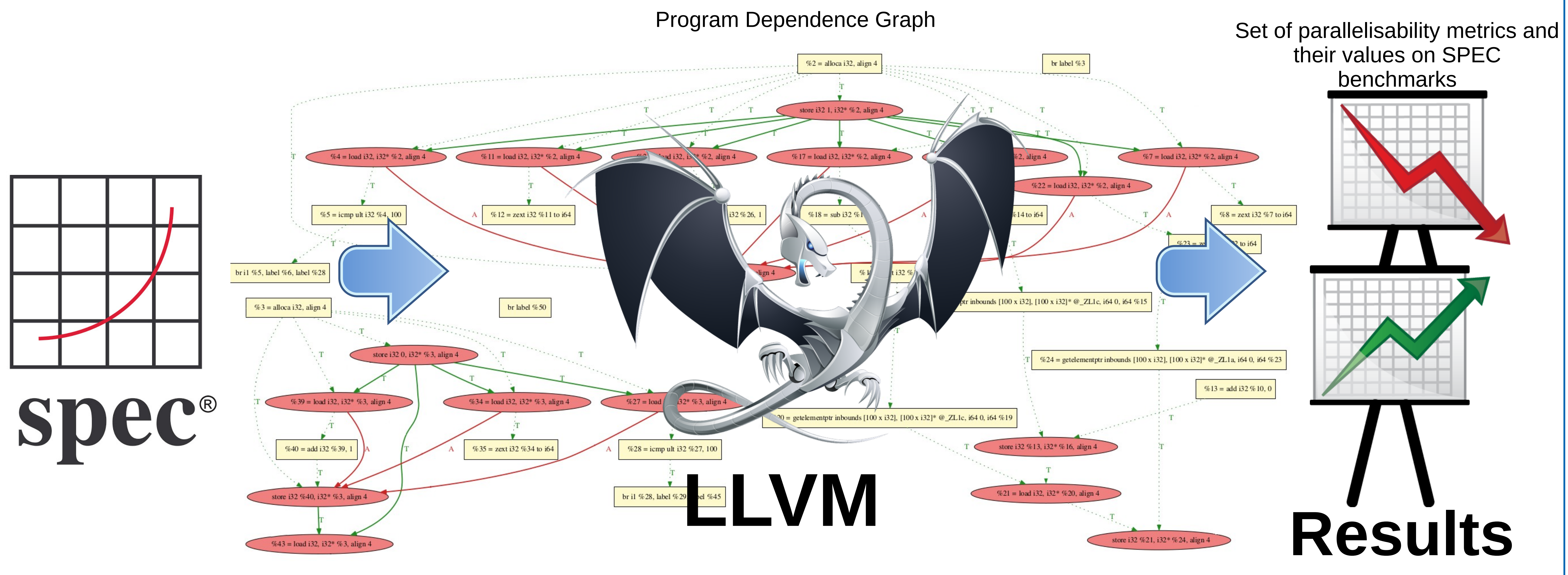
Good Score

```
Node* a;
Node* b;
int c;

// initialize linked lists

c = 0;
while (a && b) {
    c += a->val * b->val;
    a = a->next;
    b = b->next;
}
```

Bad Score

When array implementation is parallelisible, the while loop on the right has cross iteration dependency - we cannot proceed to the next iteration until we compute address of the next element in the list. This example clearly demonstrates the problem. Suboptimal data structure choice, made by the programmer, hid all the inherent parallelism of the task and made it further impossible for compiler to automatically parallelise the program. If only a programmer had some parallelisation feedback tools, which could mark the code on the left as “***good***” and the code on the right as “***bad***”, then he would have paid more attention to this piece of code and gave it a second thought.

Workflow


Program Dependence Graph
Set of parallelisability metrics and their values on SPEC benchmarks
spec®
LLVM
Results

Program Dependence Graph = Data Dependence Graph + Memory Dependence Graph + Control Dependence Graph

A – anti-dependence (WAR), T – true dependence (RAW, flow), O – output dependence (WAW)

	Metric$_1$ (M_1)	Metric$_2$ (M_2)	…	Metric$_{m-1}$ (M_{m-1})	Metric$_m$ (M_m)	$F(M_1, M_2, \ldots, M_{m-1}, M_m)$
Benchmark$_1$						Bad Score
Benchmark$_2$						Bad Score
Benchmark$_3$						Bad Score
Benchmark$_4$						Bad Score
…						
Benchmark$_{n-4}$						Good Score
Benchmark$_{n-3}$						Good Score
Benchmark$_{n-2}$						Good Score
Benchmark$_{n-1}$						Good Score
Benchmark$_n$						Good Score

The ambition of the project

The main goal of the project is to find a set of software parallelisibility metrics, which could be used to score arbitrary software. Fot that we are using the LLVM library of compiler components. LLVM performs alias and dependence analyses on results of which we rely in our work. Successful metrics could be used in major IDEs.