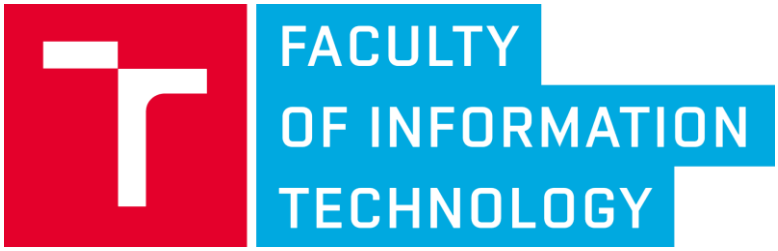

ABSTRACT

INTRODUCTION TO COMPILER DESIGN – OPTIMISATIONS FOR LOOPS

VÍT SIKORA (XSIKOR15) TOMÁŠ DVOŘÁK (XZNEBE00)

Most of the real world application execution time is spent inside loops. In scientific and specialized applications is this figure often even higher and is sometimes very close to 100%. Loops, either having a condition in the begging or at the end, have some overhead – a condition that needs to be repetitively tested and often an index or counter that needs to be periodically incremented. So it is obvious that good loop optimization is something essential for a good compiler.

Loop optimization can be described as a sequence of applied loop transformations [1]. These must be legal (i.e. they don't change the result of the program). There are many possible loop transformations with varying benefits and different possible usages. Examples include fission to multiple loops, fusion of adjacent loops with the same number of iterations, loop unwinding (unrolling) to more complex like parallelization that can take advantage of a multi-processor system. Some transformations have a drawback of needing to know the number of iteration during compile time.

Our presentation will focus on a comparison of most used transformations, their respective advantages and caveats, examples of effective uses of transformations, and some cases when a transformation was not beneficial.

REFERENCES

1. David F. Bacon, Susan L. Graham, and Oliver J. Sharp. Compiler transformations for high-performance computing. Report No. UCB/CSD 93/781, Computer Science Division-EECS, University of California, Berkeley, Berkeley, California 94720, November 1993