(1) Yes, it could be useful to have a program example here that non-computational scientists can understand

A Compiler-Driven Optimization Approach to Solving Partial Differential Equations

(2) Why LISZT?

A description (instead of a snippet) is fine.

(3) Why do you need optimizing them?

This is the main motivation and it should be super-clear

Fabio Luporini
Department of Computing
Imperial College London
London, UK
Email: f.luporini12@imperial.ac.uk

Paul H. J. Kelly
Department of Computing
Imperial College London
London, UK
Email: p.kelly@imperial.ac.uk

David Ham
Department of Computing
Imperial College London
London, UK
Email: d.ham@imperial.ac.uk

***Abstract*—The abstract goes here.**

I. Introduction

(1) What we do in computational science. Applications motivating our study. The programming model of lizst and op2 (2) "simple" kernels applied to sets of mesh elements. The need for optimizing such kernels. Contributions of the paper:

(3)
- A **compiler-driven** optimisation approach that can be (4) generalised to a wide variety of kernels employed in many computational science codes. The key feature common to these kernels is that they usually fit the L1 cache of commodity processors.
- (5) Automation of such code optimizations in a framework for the resoloution of partial differential equations through the Finite Element Method, called Firedrake. This allows us to perform an in-depth performance evaluation of the proposed optimization strategy in real-world computations.

OK

II. Background

OK

Here we discuss about the computational characteristics of computational science kernels. Briefly cite op2 kernels. Emphasis on Finite Element Assembly. Generalization and formalization of a Finite Element Assembly kernel using quadrature representation.

III. A Compiler-driven Approach to Code Optimization

move

All the work done in the polyhedral model should be useful, but our kernels fit L1, so they need special attention for what concerns key aspects like managing registers, exploiting ILP in its various forms, and reducing loop overhead. Follows a description of code transformations using our compiler-driven approach.

IV. The PyOP2 Compiler

Design and structure of the PyOP2 Compiler. Show the steps through which the IR is transformed (need to cite Firedrake here). Briefly describe a simple cost model that allows us to prune the space of code transformations.

Maybe this is subsection of V.

V. Performance Evaluation

Performance evaluation. We show all of the results got so far and we explain them. We demonstrate the claim that a compiler-driven approach is necessary to obtain maximum performance. Also, our results prove that current vendor compilers still have notable limitations, despite optimizing loop nests that are 1) affine, 2) have no stencil, and 3) ?

VI. Related Work

OK

Show other work of people working on optimizations for computational science kernels. Inter-kernel vectorisation paper from Istvan. Previous work in FFC. Spencer et-al + Shin et. al. attempt to optimize computations that could benefit from using BLAS, but in practice they don't, due to the very small dgemm operations employed. Saday's model-driven SIMD code vectorisation for the tensor contraction engine.

VII. Conclusions

Acknowledgment

The authors would like to thank... "David Ham"

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

(4) I think generality is not needed here: what is needed is what do you do in the compiler

(5) Be wary that someone will understand that you are talking about an optimizing compiler, while you only have a set of transformations