Layout-oblivious Optimization for Matrix Computations

Huimin Cui
SKL Computer Architecture,
Institute of Computing Technology, CAS
Beijing, China
cuihm@ict.ac.cn

Qing Yi
Depart. of Computer Science
University of Texas at San Antonio
San Antonio, TX, USA
qingyi@cs.utsa.edu

Jingling Xue
School of Computer Science and Engineering,
University of New South Wales
Sydney, NSW, Australia
jingling@cse.unsw.edu.au

Xiaobing Feng
SKL Computer Architecture,
Institute of Computing Technology, CAS
Beijing, China
fxb@ict.ac.cn

ABSTRACT

Most scientific computations serve to apply mathematical operations to a set of preconceived data structures, e.g., matrices, vectors, and grids. In this paper, we use a number of widely used matrix computations from the LINPACK library to demonstrate that complex internal organizations of data structures can severely degrade the effectiveness of compilers optimizations. We then present a data layout oblivious optimization methodology, where by isolating an abstract representation of computations from complex implementation details of their data, we enable these computations to be much more accurately analyzed and optimized through varying state-of-the-art compiler technologies.

Categories and Subject Descriptors:
D.3.4 [Processors]: Compilers, Optimization

Keywords: Compiler, Optimization, Packed matrix, High-performance computing

1. INTRODUCTION

A majority of scientific computation kernels typically serve to apply a sequence of domain-specific operations to a preconceived set of compound data structures, e.g., matrices, vectors, grids, and graphs. To effectively optimize these computations, a compiler must be able to correctly decipher the dependence relations among statements operating on different data. While compilers have used dependence analysis as the foundation of optimization for more than thirty years, the accuracy of dependence analysis could be severely degraded by complex data structures. Take the well-understood dense matrix multiplication for instance. When a non-rectangular matrix is stored in a packed layout such as those in Figure 1(b), the array subscripts used to reference different elements of the matrix could become extremely complex, which could easily overwhelm the internal dependence analysis of a compiler and thus thwart all optimizations to the matrix computation code.

We present a layout-oblivious optimization methodology to overcome these difficulties. In particular, using a normalization algorithm, our method seeks to isolate the high level semantics of operations from complex implementation details of their data and thereby to derive an abstract specification of operations to be applied to the data. The simplified computation can then be accurately analyzed and optimized through varying state-of-the-art compiler technologies. Finally, the optimized operations are combined with the previously isolated implementation details of their data to generate the final optimized code of the original computation.

2. THE OVERALL APPROACH

Figure 2 shows an overview of our approach, which takes two inputs, the computational kernel to be optimized and a set of annotations that define the internal organization of matrices used in the kernel. It produces optimized code through the following steps:

1. Matrix normalization. This step seeks to isolate the high level semantics of matrix operations from implementation details of matrices within the input computation, by automatically converting all the relevant array references to a higher level representation which uses (row,column) coordinates to access each abstract matrix.
variable annotations in our language serve to declare additional type information for matrix variables inside a given input code, especially those that use special storage forms. These matrix annotations can be embedded inside the input code by enclosing them in C language comments if desired, so that they can be ignored by other compilers.

4. EXPERIMENTAL RESULTS

We have implemented our layout-oblivious approach using the POET language [4], and used the Pluto [1] and the EPOD [2] source-to-source C compilers to optimize triangular matrix multiply variants - mm.tri.abs and mm.tri.packed, which represent a triangular matrix in the abstract and packed layout, respectively.

We used the icc compiler version 11.0 to compile the sources to machine code with the -openmp and -fast options. We evaluated the benchmarks using randomly initialized 2048*2048 matrices, and carried out all of our experiments on an 8-core Intel platform with two 2.33GHz quad-core Xeon 5410 processors. Each processor has 32KB L1 data cache per core, 32KB L1 instruction cache per core, and 6MB L2 unified data/instruction cache shared by all cores.

<table>
<thead>
<tr>
<th></th>
<th>icc</th>
<th>EPOD</th>
<th>EPOD+LO</th>
<th>Pluto</th>
<th>Pluto+LO</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>0.04</td>
<td>102.7</td>
<td>102.7</td>
<td>12.3</td>
<td>12.3</td>
</tr>
<tr>
<td>packed</td>
<td>0.04</td>
<td>94.1</td>
<td>0.04</td>
<td>11.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Performance results for triangular matrix multiply using our approach with EPOD and Pluto, in GFLOPS.

Table 1 shows the performance results when using our approach together with EPOD and Pluto. With our layout-oblivious optimization approach, the computational kernels which originally cannot be analyzed by compilers can benefit from normalizing its data accesses and become optimizable.

5. REFERENCES


