Alexander Powell

11.02.2016

CSCI 680

**Paper Critique for “A Stencil Compiler for Short-Vector SIMD Architectures”**

Summary

Stencil computations are a common type of computation in scientific computing applications. SIMD instructions are often used to improve the performance of such applications through data level parallelism. Generally, this is done with short vector SIMD extensions or data locality optimizations. In this paper, the authors lay out a new domain-specific language and compiler for this kind of stencil computations that effectively utilized multi-core parallelism. They also provide a number of loop based and data layout based transformations to effectively increase performance.

Overall Pros and Cons

One of the strengths of this paper is that it clearly presents a novel research topic by providing a stencil domain specific language compiler that integrates data layout based transformation with loop tiled parallel execution for multi-statement stencils. Not only does it present this novel compiler, it also demonstrates a performance improvement on several multi-core platforms for a number of benchmarks. It’s safe to say that this is a significant contribution because there has been an increasing interest in developing domain-specific frameworks for high-performance scientific computing for a while due to the diversity of emerging parallel architectures.

A problem I had with the paper is that I think they could have spent more time on the background and motivation. After reading it I am still a bit unsure what stencil computations actually entail any why they are so often used in the first place. Some high level examples in the beginning of the paper would help a lot. Also, some of the pseudocode they provide when explaining the algorithm is a bit overwhelming. It would help if they explained this more.

Questions

1. In the SDSL language described in section 3.1, the syntax varies noticeably from other more common programming languages (like C). What’s the benefit to creating a whole new language to deal with these kind of stencil computations when most high-level languages would be able to do the same things shown in figure 4, for example.
2. (More of a technical question) In the nested loop shown on page 2 in Figure 2a, is it supposed to represent a triple nested loop or two loops nested inside one larger loop? (The curly braces are confusing)
3. Why do you want a small tile size on the loops? They mention on page four that the tile sizes can be compacted to a smaller size to compensate for the larger tile sizes required by split-tiled dimensions. But doesn’t this reduce the amount of parallelism?