Correlating Cost with Performance in LLVM

Angela Pohl, Biagio Cosenza, Ben Juurlink

Embedded Systems Architecture, Technische Universität Berlin, Germany
{angela.pohl, cosenza, b.juurlink}@tu-berlin.de

ABSTRACT

A common technique to exploit data level parallelism is code vectorization. When performed by a compiler, it needs to find a valid vectorization and assess its benefit. In LLVM, this analysis is based on a cost calculation, which will approve the transformation if the cost of the vectorized code is lower than the original scalar code. However, the calculated cost does not correlate to the actual speedup gain. We therefore propose a pluggable cost model that correctly accounts for vectorization overheads and further features of the target hardware platform. Using such a platform specific model, the compiler can assess a code transformation's impact on application performance, make safe choices whether to transform or not, and compare different optimization options.

KEYWORDS: Compiler, Vectorization, Cost Model, LLVM

1 Introduction

Today, compilers apply multiple optimization passes to speed up code. One of these code transformations is vectorization, where data level parallelism is exploited by grouping together instructions and applying vector operations instead of scalar ones.

For this optimization, the compiler has to find a valid vectorization first, and then needs to determine if the transformation is beneficial; in other words, the additional overhead of vectorization must not efface the potential speedup gain. For this purpose, a cost analysis is performed to assess the transformation's efficiency. During this analysis, the cost of each instruction is determined based on the target vector size using underlying lookup-tables. The two total costs, i.e. the costs of the scalar and the vectorized version, are then compared and the code transformation is applied only if its cost is lower. Cost thus serves as a binary indicator, since there is no threshold regarding the cost value or the cost difference; the decision is solely based on its relation to the reference cost. It is therefore critical that the cost model correctly represents the underlying hardware. Otherwise, code transformations are prohibited that would achieve a speedup, and, even worse, transformed code might exhibit slowdowns due to wrong predictions.

In this work, we analyze the correctness of LLVM's cost model. For this purpose, we run 151 loop patterns on two x86 based platforms supporting the AVX and AVX2 instruction sets. Results show that

	i5-2500K			E5-2679		
	Auto-vec	Forced Vec	Δ	Auto-vec	Forced Vec	Δ
Vectorized Loops	72	92	+20	68	96	+28
Slowdowns	3	11	+8	4	14	+10
No Change	4	11	+7	4	14	+10
Speedups	65	70	+5	60	68	+8

Table 1: Comparison of results from auto-vectorization and forced vectorization, overwriting the cost analysis results

- cost model accuracy needs improvement on either platform
- there is no correlation between cost prediction and performance gain.

We therefore seek a cost model that represents the target hardware in all aspects, is no longer restricted to modeling relative cost, but is capable of performance predictions. Besides a more accurate vectorization decision, with such a model, it will be possible to compare code transformations and chose the most beneficial one.

2 Cost Model Analysis

In LLVM, transformation costs are calculated with different algorithms using the same underlying lookup-tables for instruction costs. To show comparable numbers, we therefore restricted our analysis to the cost prediction in the Loop Level Vectorizer (LLV). As a benchmark, we used the Test Suite for Vectorizing Compilers (TSVC) [MGG+11], which contains 151 loop patterns written in C. It was compiled with LLVM, version 4.0 [LA04]. For platforms, we chose an Intel i5-2500K and a Xeon E5-2697, supporting the AVX and AVX2 instruction set extensions, respectively.

2.1 Prediction Correctness

As a first analysis, we compared the results of the automatic vectorization with results from a forced vectorization that overwrites the cost model prediction. The results are listed in table 1. It can be seen that the cost model does prevent loops to be vectorized that would cause a slowdown or show no improvement. However, it also misses the opportunity to vectorize loops with speedups on both platforms. On average, the newly vectorized loops show a speedup of 1.58x on the i5 and 2.43x on the E5 platform.

2.2 Correlation Analysis

The second analysis investigates a correlation between the cost prediction and the speedup after vectorization. In LLVM's loop level vectorizer, a loop body's cost is calculated based on the instruction type and the Vectorization Factor (VF), i.e. the number of elements per vector. All individual instruction costs, which may have a dependency on the VF, are summed up and divided by the VF, which is set to 1 for scalar cost calculations:

$$c_{loop} = \frac{1}{VF} \sum c_{instr,VF}$$

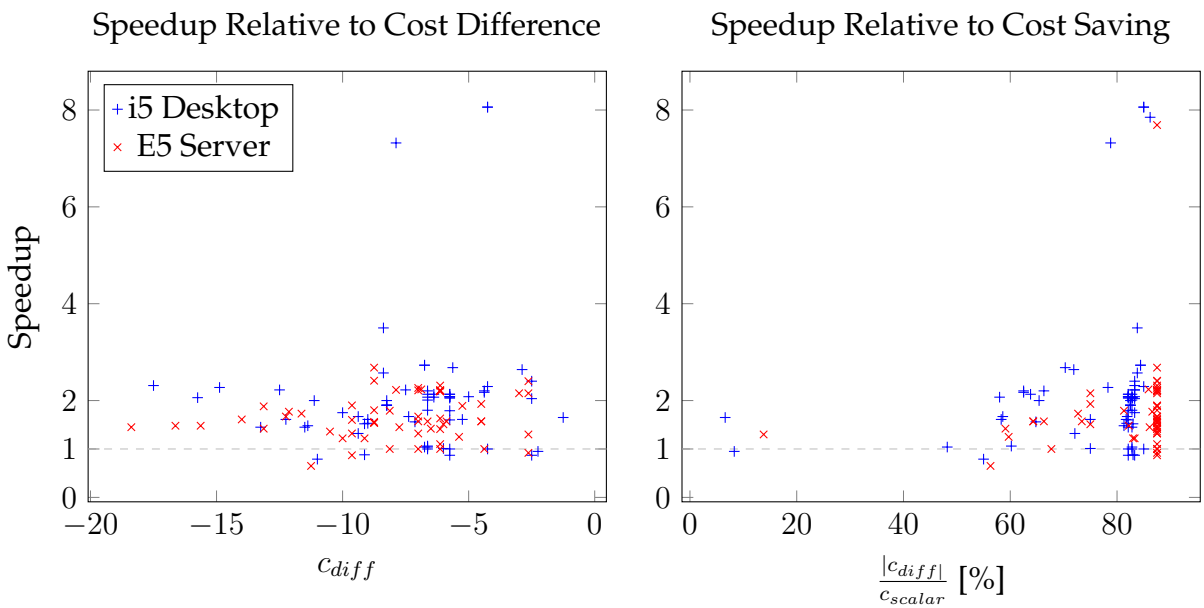

Figure 1: Correlation analysis between modeled cost and performance gain

As mentioned earlier, the instruction costs are taken from hardware specific lookup tables. The compiler will then calculate the cost difference between the scalar and the vectorized loop body by

$$c_{diff} = c_{vec} - c_{scalar}$$

A negative result ensures that the transformation is applied. Figure 1 shows the speedups of the loops vectorized by LLVM relative to their predicted cost differences. It can be seen that there is no distinct correlation between the two, as one would expect a higher speedup with increasing differences.

Besides this absolute measure, we furthermore calculated the predicted cost saving, i.e. the percentage of instruction costs saved by vectorization, via

$$saving = \frac{|c_{diff}|}{c_{scalar}} = \frac{|c_{vec} - c_{scalar}|}{c_{scalar}}$$

When plotting the speedups against this metric, it can be seen in Figure 1 that the cost model predicts similar savings on each platform for the majority of loops. In this case, the average predicted saving is 76% on the desktop processor and 82% on the server processor, which is close to a $\frac{7}{8}$ reduction, i.e. the ideal saving for the maximum vectorization factor of 8. Nonetheless, resulting speedups vary significantly between between small slowdowns and speedups of 3x and higher, although the same saving is predicted. It again proves that the current cost modeling is not able to foresee the actual performance impact of a compiler's code transformations.

3 Correlating Cost with Performance

Based on these insights, we therefore propose a more accurate cost modeling approach. As the results show, the current cost calculation tends to predict a transformation cost close to

the ideal $\frac{1}{VF}$. In other words, the vectorization overhead is not modeled accurately. We are also able to observe a difference in speedups based on the target hardware, although both platforms use the same vector instructions for the majority of the loops. Further hardware specific side-effects hence need to be considered when predicting resulting performance, which is not the case as of today.

As a first step, the underlying instruction cost lookup-tables need to be rendered more precisely. Since modern processors support a plethora of runtime optimization techniques, a static cost model will not be able to cover all these features and characteristics, however. It is therefore desirable to accurately analyze or benchmark an application's target system, create an exact cost lookup-table, and add this table to the compiler without having to re-built it. Such an exact lookup-table could be based on already existing performance tables, obtained via micro-benchmarking, heuristics, or even machine learning approaches. Creating such accurate lookup-tables will be the next step of this work.

Secondly, cost modeling approaches within the compiler should be aligned. With each compiler pass using its own cost analysis algorithm, transformation benefits cannot be compared. For example, there are loops that can be either vectorized via loop or straight line code vectorization.

4 Summary

When optimizing code, compilers need to determine if a code transformation is indeed beneficial in terms of performance. For this purpose, cost calculations are performed, and results of vectorized and scalar code versions are compared. If the cost of the vectorized code is lower, the transformation is applied.

Our analysis shows, however, that the cost prediction does not correlate with the final performance gain. Instead, vectorization overhead and further hardware-specific side-effects are not taken into account. We therefore propose a pluggable cost model, which will enable target system specific instruction cost lookup tables. These tables can be generated offline, via modelling, heuristics, benchmarking or machine learning. With this approach, it is possible to correlate cost prediction with the performance gain of an optimization. Furthermore, it will allow the comparison of different optimization techniques based on their calculated benefit during compilation.

References

[LA04] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis and transformation. pages 75–88, San Jose, CA, USA, Mar 2004.

[MGG+11] Saeed Maleki, Yaoqing Gao, Maria J. Garzarán, Tommy Wong, and David A. Padua. An Evaluation of Vectorizing Compilers. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, PACT '11, pages 372–382, Washington, DC, USA, 2011. IEEE Computer Society.