Cost Modelling for Vectorization on ARM

Angela Pohl, Biagio Cosenza and Ben Juurlink

ARM Research Summit 2018
Challenges of Auto-Vectorization in Compilers

1. Is it **possible** to vectorize the code?
   - Passes: Loop Level Vectorization (LLV), Superword-Level Parallelism (SLP) …
   - Algorithms: loop transformations, partial vectorization, SLP padding …
   - Challenges: dependences, missing instructions …

2. Is it **beneficial** to vectorize the code?
   - Transformations might add overhead
   - Code can run slower afterwards

Cost Modelling for Vectorization on ARM, A. Pohl et al.
Short Introduction to Cost Modelling

C Source

```c
int s121() {
    for (int i = 0; i < LEN; i++) {
        j = i + 1;
        a[i] = a[j] + b[i];
    }
    return 0;
}
```

LLVM's Intermediate Representation

```llvm
%indvars.iv.next = add nuw nsw i64 %indvars.iv, 1, !dbg !55
%1 = load float, float* %arrayidx, align 4, !dbg !56, !tbaa !22
%2 = load float, float* %arrayidx7, align 4, !dbg !57, !tbaa !22
%add8 = fadd fast float %2, %1, !dbg !58
store float %add8, float* %arrayidx10, align 4, !dbg !60, !tbaa !22
%exitcond = icmp eq i64 %indvars.iv.next, 31999, !dbg !61
```
State of the Art Analysis

- LLV pass of LLVM 6.0 on ARMv8
- TSVC Benchmark: 151 basic loop patterns
- Vectorization with overwritten cost model
- No unrolling, no interleaving

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorized loops</td>
<td>61</td>
</tr>
<tr>
<td>$f\ominus$</td>
<td>14</td>
</tr>
<tr>
<td>$f\oplus$</td>
<td>0</td>
</tr>
<tr>
<td>$t_{norm}$</td>
<td>117.2</td>
</tr>
</tbody>
</table>
Linear Modelling Approach

• Formulate basic block as linear equation for each test pattern

\[ c_{vec} = \sum n_i c_i \]

• Determine basic block target cost based on measurement

\[ c_{target} = \frac{c_{scalar}}{S_{meas}} = \sum n_i c_i' \]

• Apply mathematical fitting to set of linear equations
Linear Modelling: Example

```c
for (int i = 0; i < LEN; i++){
    a[i] = b[k] + 1.0;
}
```

$$c_{scalar} = 8 \quad S_{meas} = 2.76 \Rightarrow c_{target} = 2.89$$

```c
for (int i = 0; i < LEN; i++){
    prod *= a[i];
}
```

$$c_{scalar} = 6 \quad S_{meas} = 2.30 \Rightarrow c_{target} = 2.61$$

$$2.89 = 1 \times c_{load} + 1 \times c_{fadd} + 0 \times c_{fmult} + \cdots$$

$$2.61 = 1 \times c_{load} + 0 \times c_{fadd} + 1 \times c_{fmult} + \cdots$$
Modelling Speedup Instead of Cost

Problem:
- Target costs vary significantly, i.e. data has to be fitted across large interval
- But: fitting benefits from smaller target intervals

Idea:
- Model speedup instead of cost
- Limits interval to (0, vectorization factor)
- Linear equations become \( S_{meas} = \sum n_i w_i \)
  with \( n_i = \) numer of instructions of same type, \( w_i = \) corresponding weight
Results: Fitted for Speedup

- L2: Least Squares (minimizes Euclidian L2 Norm)
- NNLS: Non-Negative Least Squares (all coefficients > 0)
Adding Block Composition as Feature

Problem:
• Current cost model only cares about individual instructions
• Arithmetic intensity can have major impact on speedup, e.g. if code is memory bound

Idea:
• Replace count of instruction type with overall percentage, e.g. 20% load, 10% cmp, …

\[ S_{meas} = \sum \frac{n_i}{n_{total}} w_i \]
Results: Fitted with Rated Instruction Count

\begin{align*}
\text{L}^2 \quad \text{Speedup} & & \quad \text{NNLS} \quad \text{Speedup} \\
S_{\text{meas}} & \times \rho = 0.65 & S_{\text{meas}} & \times \rho = 0.58 \\
\end{align*}

\begin{table}
\begin{tabular}{|c|c|c|c|}
\hline
\text{Metric} & \text{Baseline} & \text{L}^2 & \text{NNLS} \\
\hline
\text{Vectorized loops} & 61 & 76 & 80 \\
\text{f} & 14 & 2 & 0 \\
\text{f} & 0 & 1 & 3 \\
\text{tnorm} & 117.2 & 113.2 & 113.4 \\
\hline
\end{tabular}
\end{table}
Leave One Out Cross Validation: NNLS

- Fitting is done with whole training data set except for one kernel
- Kernel is then predicted with that model
Conclusion

- Compilers need more accurate cost models to avoid mispredictions

- Aligned cost models enable comparison of different transformation options

- With our refined cost model we
  - Increase the correlation between estimated and measured speedup
  - Decrease the number of false predictions
  - Lower execution times

- Next steps: add more code features and tests to cover all instruction types
THANK YOU
BACKUP
Why a More Accurate Cost Model?

1. Improve classification results to decide whether code should be vectorized or not.

2. Enable comparison of different transformations with each other, not just to baseline.

```c
for (int i = 0; i < LEN/2; i++){
    k = j + 1;
    a[i] = b[k] - d[i];
    j = k + 1
    b[k] = a[i] + c[k];
}
```

<table>
<thead>
<tr>
<th></th>
<th>LLV</th>
<th>SLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Speedup</td>
<td>0.96</td>
<td>1.00</td>
</tr>
<tr>
<td>Measured Speedup</td>
<td>1.13</td>
<td>1.20</td>
</tr>
</tbody>
</table>

Example code run on Intel i5
L2- LOOCV Validation Results

![Leave One Out Cross Validation](image)

- l2
- nnls
- baseline
State of the Art Analysis – x86

- TSVC benchmark: 151 basic loop patterns
- SLP vectorization applied after loop unrolling
- Results shown for Intel Xeon E5 with AVX2

Cost Modelling for Vectorization on ARM, A. Pohl et al.
Results: Fitted for Cost – x86

- Correlation between estimated and measured speedup in LLVM's LLV pass after fitting
- L2: Least Squares (minimizes Euclidian L2 Norm)
- NNLS: Non-Negative Least Squares (all coefficients > 0)
- SVR: Support Vector Regression
Results: Fitted for Speedup – x86

- All three approaches improve correlation further
- False negatives reduced (L2) or eliminated (NNLS, SVR)
- But: small increase in false positives