Speculative Dynamic Vectorization for HW/SW Co-designed Processors

Rakesh Kumar
Alejandro Martínez
Antonio González

Introduction/Motivation

Static Vectorization

- Effective for traditional array-based applications
- Conservative to ensure correctness
- Pointers complicate dependence analysis
- It limits vectorization opportunities

The Proposal

- Vectorize at run time
- Speculatively reorder ambiguous memory accesses
  - It will uncover more vectorization opportunities
- Hardware checks for any memory violation
  - Interpret the code in case of violation
- Algorithm
  - Consider ambiguous memory references as independent.
    - Convert them to speculative instructions if reordered
  - Vectorize consecutive stores
  - Vectorize remaining arithmetic instructions

An Example

The following loop cannot be vectorized if the two arrays cannot be proved non overlapping statically.

```c
void example(double *a, double *b)
{
    int i;
    for (i = 0; i < NUM_ITR; i++)
        a[i] += b[i] * CONST;
}
```

Vectorization using the proposed algorithm:

```
Vectorization continues
Loop:  Pack  v12, xmm1, xmm0
       mulpd  v10, [ebx+eax*8]
       addpd  v9, [edx+eax*8]
```

Speculation and Recovery

Software

- Label instructions in the program order
- If a pair of memory references:
  - May alias and
  - Is reordered
- Convert original load/store instructions to speculative instructions

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ld_64</td>
<td>v1, M[x]</td>
</tr>
<tr>
<td>2</td>
<td>st_64</td>
<td>v2, M[y]</td>
</tr>
<tr>
<td>3</td>
<td>ld_64_spec</td>
<td>v1, M[x]</td>
</tr>
<tr>
<td>4</td>
<td>st_64_spec</td>
<td>v2, M[y]</td>
</tr>
</tbody>
</table>

Hardware

- Save register and memory state before executing speculative code

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PC:</td>
<td>2</td>
<td>st_64_spec</td>
</tr>
<tr>
<td>PC:</td>
<td>1</td>
<td>ld_64_spec</td>
</tr>
</tbody>
</table>

- If violation, restore saved state and restart execution without speculation

Results

The proposed algorithm outperforms GCC by 25%, 22% and 3% for LINPACK, Physicsbench and SPECFP2006 respectively.