Implementation and Optimization of Data Prefetching Algorithm Based on LLVM Compilation System

Group 4: Yixin Zhang, Jiazhen Zhao, Yuntao Zhang, Yuqing Wang

Yunda Chai et al 2021 J. Phys.: Conf. Ser. 1827 012136
Memory access optimization:

- Why we need memory access optimization?
  - Loading data from memory takes a lot of time.

- Current popular approach: principle of data locality (When loading a data from memory, load its neighbor data as well as a whole block)
  - Drawback: Poor data locality issue (which results in 64% CPU cycles loss)[1], high cost of doing profiling to fix locality issue [2].

Data prefetching

- What is prefetching in memory access optimization?
  - Prefetching is to read the data from memory into the cache before the data is used. (Eg, load instruction is executed in advance)

---

Without **data prefetching:**

```
Execution of program  wait  Execution of program
load A[N]
```

**With data prefetching:** (loading process can be done in parallel with the program execution)

```
Pipeline execution:

The delay caused by taking A[N]
The delay caused by taking A[N+1]
The delay caused by taking A[N+2]
```
Benefits of **data prefetching**
- Parallelize the loading process with the rest of the program
- Reduce memory access latency to improve program performance
- Solve the issues related to data locality

Drawbacks:
- Additional costs to do the analysis of how to do the prefetching

Now, let’s show the algorithm for **data prefetching** ......
Algorithm Overview

➢ What:
  ● Iterate through all load/store instructions. Determine the capability and efficiency to prefetch the instruction.

➢ How:
  ● Traverse the CFG graph using depth-first algorithm.
  ● Iterate through qualified BBs. Each load/store will go through cost check.
Pre-condition Check

Two basic assumptions:

- No prefetching in BB
- Objects that need prefetching exist

Parameter explanation:

- PrefetchDistance: determine whether to analyze
- InstrinsinID = prefetch: the instruction is prefetching
Sub-Algorithm: Prefetch Scheduling

➢ Purpose:
• Schedule ahead to eliminate the overhead of inserting prefetch instructions.
• Hide the delays of memory access. (elaborated in the introduction).

➢ Details:
• Let’s define:
  • \( T_{pref} \) = Delay in number of clocks of issuing the prefetch instructions for a certain basic block.
  • \( T_{loop} \) = Number of clocks for one loop when there is no prefetch operation.
  • \( IterAhead \) = Number of loop iterations ahead to issue the prefetch instructions.
• Then:
  • \( IterAhead = \frac{T_{pref} + T_{loop} - 1}{T_{loop}} \)
  • - 1 here, as we want to let the prefetch happens at least one cpu clock before the data access.
Algorithm for prefetch scheduling

<table>
<thead>
<tr>
<th>Algorithm 2. Basic block instruction cost and Prefetch scheduling distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each $BB$ in $CFG$</td>
</tr>
<tr>
<td>For each $ins$ in $BB$</td>
</tr>
<tr>
<td>if $ins$ is ephemeral value</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>LoopSize += $ins$’ Cost</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>if LoopSize is null</td>
</tr>
<tr>
<td>LoopSize = 1</td>
</tr>
<tr>
<td>ItearsAhead = (PrefetchDistance + LoopSize − 1) / LoopSize</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

* Ephemeral value means just a simple value. If an instruction is an simple value, we will skip’s its cpu clocks count.
Cost Model

- Do not prefetch if constant value or access step is not large enough

---

**Algorithm 3. The analysis of cost model**

```plaintext
For each ins in loop
    /* Get the prefetch analysis information of the loop body*/
    if(The value of prefetch is an invariant || The step of access is not enough to guarantee prefetch || The prefetch operation overlaps)
        continue
    if(The prefetch scheduling distance > prefetch forward iteration value)
        continue
end for
```
Results & Discussion

- LLVM, Shenwei platform, SPEC2006 test set
- average speedup of 6%
Conclusion

- **Prefetch analysis algorithm**
  - Loop analysis, scheduling, cost model
  - eliminate unnecessary prefetch operations
  - reduce instruction overhead
  - improve system performance

- **Future improvement**
  - Support outer-loop prefetch