HE VD IG | REDS Institut Reconfigurable and Embedded Digital Systems

Calcul Numérique et accélération Matérielle (CNM)

Prof Marina Zapater
Assistants: Mehdi Akeddar

Stage 3 – Convolution and Prefetching Optimization

lab01 (17.10.2024)

Objectives of the laboratory

The objectives of Lab01 are to exercise the content regarding acceleration using CPU. Lab01 is structured in three sessions. This manual refers to "Session 3" and is devoted to optimizing the convolution operation by using data prefetching. **Some code for this session is c++, make sure to compile with g++ instead of gcc.**

Laboratory validation and grading

- We will grade this laboratory based on the work we ask to be delivered. The parts in *light blue* are what you are expected to reply in the readme file.
- We expect (at least) three different commits to the git repository, one per session, clearly indicating the session submission (for example: git commit -m "submission session 2 lab01")
- This last commit, with all work for all sessions completed, should be done at the latest the night before the next lab (lab2 starts). This means by October 30 2024 at 23h59.

Stage 1 – Prefetching and Processors

In computer architecture, prefetching refers to the retrieving and storing of data into the buffer memory (cache) **before** the processor requires the data. When the processor wants to process the data, it is readily available and can be processed within a very short period of time.
If the data were not stored in the cache, the processor would have to download directly from the memory address - accordingly, there would be delays. Prefetching aims to reduce access speed and is used at several levels of a system, for example, even with DDR, main memories or boot operations of operating systems. Which data will be loaded into the cache is often determined using a branch prediction algorithm.

Stage 2 – Measuring convolution performance with and without prefetching

We will implement a simple convolution function and apply prefetching on this task. The general idea consists of identifying the input to be prefetched and explicitly call the "__builtin_prefetch" (documentation) function to optimize the data call.

Implement a prefetching strategy using the "__builtin_prefetch" function. Record the time taken by the task with and without prefetching, for different matrices/filter sizes and different level of compiler optimization (O0, etc).
What do you observe? Can you explain those results?

Stage 3 – Efficient prefetching

Following the observations in Stage 2, the performance of prefetching highly depends on the task and material setup. Therefore, to showcase this, we provided a C code in file “binary_search_prefetching.c”.
This code implements a simple binary search algorithm in an optimized way. The purpose of binary search is to find the index of the **key** in the array if it exists or return -1 if it's not present. The code includes conditional prefetching (**#ifdef DO_PREFETCH**) to provide hints to the processor about loading future data. Prefetching can improve cache utilization and reduce memory latency.

- Two prefetches are added, each for a different half of the search range.
- The first prefetch hints at loading data around **array[(mid + 1 + high) / 2]**, which is in the higher half of the current search range.
- The second prefetch hints at loading data around **array[(low + mid - 1) / 2]**, which is in the lower half of the current search range.

Compile the code with and without the DO_PREFETCH flag to generate 2 different executables. You can try different levels of compiler optimization.

Stage 4 – perf monitoring tool (REMINDER)

The **perf** tool is presented in ***lab 1-Session 01.*** Here’s a little reminder on how to use it.

Launching perf monitoring:
As you will see in the documentation we linked above, to gather performance statistics for the program you want run, you can use “perf stat”.

```
perf stat -e EVENTS_YOU_WANT_TO_MONITOR ./program-to-monitor
```

You need to provide **perf stat** with a comma-separated list of the events you want to record, as well as the name of the program that you want to monitor.

Run the perf command on the 2 executables (with and without prefetching). Record the L1 cache loads, the L1 cache loads misses, and the total task time. To accomplish this task, you will need to look into the “perf stat” documentation. We recommend that you list the events available, and that you pick the relevant ones you want to record.

Before you leave…

Make sure you have all the requests *in light blue* of this manual, that you have implemented all the functions required and filled-in the answers in the *README* file in the repo. Then commit all files to your own “cnm24-yourname” repository.

We expect (at least) one commit for this session, clearly indicating the session submission (for example: git commit -m “submission session3 lab01”)