CS 378: PfC-HP
The University of Texas at Austin — Project 1
CS 378 Teaching Team — Due: March 1 2024, 9p

Project 1

1 Interaction of Algorithms with Architecture

This is an individual project.

In this project, you will implement a high-performance matrix-matrix multiplication algorithm.

1.1 Preconditions

Before starting this assignment, you should be familiar with

- Cloning, building and installing a pre-existing software library.
- Understand the layout of matrices within memory, and know how to access various elements of the matrix.
- Understand accuracy of floating point calculations, determine if the results of matrix multiplication is correct and write code to implement the same.

1.2 Postconditions

- Determine the theoretical peak performance of a processing core.
- Analyze the cost of moving data between memory and registers.
- Block matrix-matrix multiplication for registers.
- Cast computation in terms of vector intrinsic functions that give access to vector registers and vector instructions.
- Optimize the micro-kernel that will become our unit of computation for future optimizations.
- Identify layers in the memory hierarchy.
- Orchestrate matrix-matrix multiplication in terms of computation with submatrices (blocks) so as to improve the ratio of computation to memory operations.
- Rearrange data to improve how memory is contiguously accessed.
- Organize loops so data is effectively reused at multiple levels of the memory hierarchy.

2 Getting the project code

For this project, the code is made available on my github repository. You will duplicate my repository into your own private repository by following these steps:

- On the github web page, create a new private repository. Your repository must be named

  ```
  <eid>-cs378pfcp-project-mmm
  ```

  Make sure the repository is private.

- On your favorite terminal, create a bare clone of my repository

  ```
  $ git clone --bare https://github.com/utcs378-pfcp/cs378pfcp-
      project-mmm.git
  ```

- Mirror-push to your private repository.

  ```
  $ cd cs378pfcp-project-mmm.git
  $ git push --mirror https://github.com/<username>/<eid>-cs378pfcp-
      project-mmm.git
  ```

- Remove the temporary local repository you created earlier.

  ```
  $ cd ..
  $ rm -rf cs378pfcp-project-mmm.git
  ```

- You should be able to see the starter code in your private repository.

- Now, you must clone your private repository so you can get started.

  ```
  $ git clone git@github.com:<username>/<eid>-cs378pfcp-project2.git
  ```

- `cd` into the your repository on your command line.

  ```
  $ cd <eid>-cs378pfcp-project-mmm
  ```

- Build and run the starter code

  ```
  $ make run
  ```

  This should compile and run your code. Make sure the code runs properly. If you should get an error, read the error, and figure out how to fix it. The error most likely is because of your set up. Understand the errors and fix them accordingly.

3 Starter Code

`driver.c` is the file where `main()` is written. `main()` gets the matrix sizes at compile time through the `PSIZE` variable in the `Makefile`. It sets up the matrices, and fills them with randomized double precision floating point numbers. The code calls to the BLIS API `bli_dgemm` as the reference code, and times this call. It also calls `MyGemm()`, which at the moment is a triple-nested loop implementation of `GEMM`. The code then compares the entries of the resulting C from the reference implementation and `MyGemm()`, and reports the maximum absolute difference of the two implementations. We expect this difference to be lower than 10^{-12}.

To build the code

To build the code without running the executable, use the `driver` target, in other words run,

```
$ make driver
```

on your command line.
Once your code is built, you can run the executable by typing either of the following in your command line:

```
$ make run
```

or

```
$ ./driver_gemm.x
```

4 Things you will implement

There are various tiers of things you will implement for this project. You will be grading based on the features you chose to implement, and how you implement them.

Rules

- You must use row strides and column strides to traverse the matrices.
- You must create a separate function for the ukernel.
- The arguments that you pass in to any function you create must be needed by the function. You may not pass in variables "just in case".
- You must declare any function you may create in the `mygemm.h` header file and not in project2.h
- Your values of `MR, NR, KC, MC, NC` must be defined in `mygemm.h` header file.

- You may chose to create `C` files that contains any extra functions you may chose to write to support your implemenation of GEMM. If you create extra `C` files to support your implementation, the necessary object files must be listed in `objs.mk`. See the example of the file `fiveloops.c`
- Any memory you allocate must be of a size that you need. you may not allocate more memory "just in case".
- Any memory that you allocate must be freed.
- Your code must be well documented, and choices you make for your implementation must be documented.
- If you receive a full score on your project then your submission has passed all the autograder tests. However the converse of this statement is not true.

4.1 Beginning

- You have implemented the five loops Goto's algorithm.
- You have implemented the microkernel using intrinsics.
- You have implemented the packing routines.
- Your implementation of GEMM knows only how to deal with column-stored matrices.

4.2 Developing

- You have determined the values of `MR, NR, KC, MC, NC` based on the architecture you are working on.

4.3 Advanced

- Unroll the loop within the microkernel.
- Your implementation of GEMM is within 98% of the reference implementation.
- Your implementation of GEMM works for any arbitrary matrix size, as well as shape. In other words, you have dealt with the edge cases where the matrix size may not be a "nice" multiple of `MR, NR, KC`.
- Your implementation of GEMM works for matrices where the column stride is greater than the number of rows in the matrix.

4.4 Exemplary

- Your implementation of GEMM works on both column-stored matrices and row-stored matrices, i.e. the input matrices are either all column-stored or row-stored.
- You explore other optimizations that might be possible by researching the available research on this topic.

5 Submission

You will submit your code to gradescope. You will submit:

- `gemm.c`
- `mygemm.h`
- `objs.mk`
- Any other C file that you may have written.

Please give us 2-3 days to set it up and make sure everything is working.

6 Questions to ask while Debugging

- The first thing you should get working is the 5 loops around the microkernel without packing. At this point, set your values of MC, KC, NC to a multiple of 48.
- Implement the packing routine for matrix B. Make the necessary changes in the ukernel as well.
- If this is not working, edit your Makefile or driver.c so that you are testing only one matrix size. Set the $m = MR$, $n = NR$. I would recommend setting $k = KC$. At this stage, are you getting the correct answer when you multiple the matrices? If not, check how you are packing this micropanel of B, and how B is accessed in the microkernel.
- Once you get the above to work, change $n = 2 * NR$, and repeat this process. Once you get this to work, make n a range of arbitrary values that are multiples of 6, and see if your code is still functional.
- Repeat these same steps for the A matrix.
- Once you are confident that your packing routine and microkernel are functional, you can move on to the developing stage. Figure out the close to optimal values of MC, KC, NC. Refer to lecture recordings to figure this out.
- This should get you to within 98% of the reference implementation.
- You are now ready to tackle the Advance stage.
- We want to unroll the loop in the microkernel by a small factor. You have to determine what factor works best for you.
- Next you can deal with any matrix size. Think about what would happen in your current code if your matrix size is not a nice multiple of MR, NR, KC. How does that break your code? How can you fix that?
- Make sure you have not made any assumptions that the column stride is equal to the number of rows in your code.