SGEMM Tiled Stencil Coarsened

PUMPS 2015

OBJECTIVE

The purpose of this lab is to practice the thread coarsening and register tiling optimization techniques using matrix-matrix multiplication as an example.

INSTRUCTIONS

Edit the file code to launch and implement a matrix-matrix multiplication kernel that uses shared memory, thread coarsening, and register tiling optimization techniques. The first input matrix `A` is in column major format while the second matrix `B` is in row major format. This means that the columns and rows read from the datafile for the first input matrix are transposed. Your code is expected to work for varying input dimensions - which may or may not be divisible by your tile size. It is a good idea to test and debug initially with examples where the matrix size is divisible by the tile size, and then try the boundary cases.

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

QUESTIONS

- Assuming and output tile of size (M rows)x(N columns), where M is a multiple of N, how much on chip memory (registers and shared memory) is needed to store each of the input and output tiles. Show your work.
- Try to tune your tile sizes for the two input matrices and output matrix. Report the execution time for various combinations of tile sizes. Comment on the results explaining what factors you think might have influenced the difference (or lack of difference) in performance when the tile sizes change.

SUGGESTIONS

- The system's autosave feature is not an excuse to not backup your code and answers to your questions regularly.
- If you have not done so already, read the tutorial
- Do not modify the template code provided – only insert code where the `//@@` demarcation is placed
- Develop your solution incrementally and test each version thoroughly before moving on to the next version
- Do not wait until the last minute to attempt the lab.
- If you get stuck with boundary conditions, grab a pen and paper. It is much easier to figure out the boundary conditions there.
- Implement the serial CPU version first, this will give you an understanding of the loops
- Get the first dataset working first. The datasets are ordered so the first one is the easiest to handle
- Make sure that your algorithm handles non-regular dimensional inputs (not square or multiples of 2). The slides may present the algorithm with nice inputs, since it minimizes the conditions. The datasets reflect different sizes of input that you are expected to handle
- Make sure that you test your program using all the datasets provided (the datasets can be selected using the dropdown next to the submission button)
- Check for errors: for example, when developing CUDA code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do {                                                \
        cudaError_t err = stmt;                                           \
        if (err != cudaSuccess) {                                         \
            wbLog(ERROR, "Failed to run stmt ", #stmt);                   \
            wbLog(ERROR, "CUDA error ...  ", cudaGetErrorString(err));\
            return -1;                                                    \
        }                                                                 \
    } while(0)
```

An example usage is `wbCheck(cudaMalloc(...))`. A similar macro can be developed while programming OpenCL code.

LOCAL DEVELOPMENT

While not required, the library used throughout the course can be downloaded from Github. The library does not depend on any external library (and should be cross platform), you can use `make` to generate the shared object file (further instructions are found on the Github page). Linking against the library would allow you to get similar behavior to the web interface

(minus the imposed limitations). Once linked against the library, you can launch your program as follows:

```
./program -e <expected_output_file> \
    -i <input_file_1>,<input_file_2> -o <output_file> -t <type>
```

The `<expected_output_file>` and `<input_file_n>` are the input and output files provided in the dataset. The `<output_file>` is the location you'd like to place the output from your program. The `<type>` is the output file type: `vector`, `matrix`, or `image`. If an MP does not expect an input or output, then pass `none` as the parameter.

In case of issues or suggestions with the library, please report them through the issue tracker on Github.