CS601: Software Development for Scientific Computing

Programming Assignment 1 - Makefiles, Matrix Multiplication, Library Functions, and Valgrind (Cachegrind)
Due: 15/9/2024

The objective of this assignment is to gain a hands-on experience with:

1. A build-tool such as Make
2. Reading the documentation of and using library functions
   (a) Use of BLAS library functions
   (b) Use of `intrinsics` for x86 architecture
3. Using Valgrind to collect metrics pertaining to Cache memory

1 Version Control Systems and `Git`

You will use GitHub and git, a version control system, to get started with and submit all assignments. Please read the guide at Github about version control and git. Another resource is here.

1.1 `Git` - setup and submission instructions

You have provided us your GitHub accounts earlier. This is the account you should use to create and submit all of your assignments this semester. The `Git` setup and submission instructions remain the same for all assignments. Please replace the assignment number in the examples with the number of the assignment you are submitting.

1. Create a Git repository for the assignment.
   (a) Log in to your Github account.
   (b) Visit the Github repository Discussion page to find the link for the assignment cs601PA1. Click the link. This will create a repository on Github for the assignment (you will follow a similar procedure for all future assignments). Make sure that the repository is called 'IITDhCSE/CS601PA1-<your GitHub username here>'.
   (c) Clone the repository to develop your assignment. Cloning a repository creates a local copy. Change your directory to whichever directory you want to create your local copy in, and type:
   `> git clone git@github.com:IITDhCSE/CS601PA1-<your GitHub username here>.git CS601PA1`
   This will create a subdirectory called `CS601PA1`, where you will work on your code.
   In this command: `git clone` copies a repository.
   `git@github.com:IITDhCSE/CS601PA1-<your GitHub username here>.git` tells `git` where the server (remote copy) of your code is.
   `CS601PA1` tells `git` to place the code in a local directory named `CS601PA1`
   If you change to directory `CS601PA1` and list the contents, you should see the files you will need for this assignment:
   ```
   > cd CS601PA1

   > ls
   ```
   And you should see all of the files required to get started with this assignment.

   Sometimes, you may see an error accessing the repository and the clone command may fail to recognize the existance of a repository. One of the reasons could be setting up access credentials:

Setting up SSH key with GitHub Set up a public SSH key in your GitHub account (if you haven't already). To do this, first generate a new ssh key:

```
> ssh-keygen
```

Hit enter three times (to accept the default location, then to set and confirm an empty passphrase). This will create two files: /.ssh/id_rsa (your private key) and /.ssh/id_rsa.pub (your public key) Then print out your public key:

```
> cat  /.ssh/id_rsa.pub
```

And copy it to the clipboard. Then follow steps at: https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account

2. As you develop your code, you can commit a local version of your changes (just to make sure that you can back up if you break something) by typing:

   ```
   > git add <file name that you want to commit>
   ```

   ```
   > git commit -m ''< describe your changes>''
   ```

   `git add <filename>` tells `git` to "stage" a file for committing. Staging files is useful if you want to make changes to several files at once and treat them as one logical change to your code. You need to call `git` add every time you want to commit a file that you have changed.

   `git commit` tells `git` to save a new version of your code including all the changes you staged with `git add`. Note that until you execute `git commit`, none of your changes will have a version associated with them. You can save/commit the changes many times. It is a good habit committing often. It is very reasonable if you commit every ten minutes (or more often).

   Do not type `git add *` because you will likely add unnecessary files to the repository. When your repository has many unnecessary files, committing becomes slower. If the unnecessary files are large (such as executables or core files), committing can take several minutes and your assignments may be considered late.

3. The changes you saved by executing `git commit` in the previous step are local to your development environment i.e. they are saved in a local repository. To copy your changes back to Github (to make sure they are saved if your computer crashes, or if you want to continue developing your code from another machine, or you want to make your code visible to a collaborator), type

   ```
   > git push
   ```

   If you do not push, the teaching staff cannot see your solutions.

4. you will use `git`'s "tagging" functionality to submit assignments. Rather than using any submission system, you will use `git` to *tag* which version of the code you want to grade. To tag the latest version of the code, type:

   ```
   > git tag -a <tagname> -m ''<describe the tag>''
   ```

   This will attach a tag with name <tagname> to your latest commit. Once you have a version of your program that you want to submit, when you run the following commands:

   ```
   > git tag -a cs601pa1submission -m "Turnin PA1"
   > git push --tags
   ```

   it would associate your source code files with a release tag name cs601pa1submission and push it to the remote server. The grading system will check out the source code version with appropriate tag name and grade that. If you want to update your submission (and tell the grading system to ignore any previous submissions), you would type:

   ```
   > git tag -a -f cs601pa1submission -m "Turnin PA1"
   > git push -f --tags
   ```

to indicate:

`> git tag -a -f cs601pa1submission -m "Turnin PA1"` overwrites the tag named "cs601pa1submission" on the local repository.

`git push -f -tags`, pushes the updates and overwrites the tag on the remote repository (on Github).

These commands will overwrite any other tag named `cs601pa1submission` with one for the current commit. Please be careful about the following rules:

(a) For each assignment, you should tag only one version with the name `cs601pa1submission`. It is your responsibility to tag the correct one. You CANNOT request regrading if the grading program retrieves the version that you do not want to submit.

(b) After tagging a version `cs601pa1submission`, any modifications you make to your program WILL NOT BE GRADED (unless you update the tag, as described above).

(c) The grading program starts retrieving soon after the submission deadline of each assignment. If your repository has no version tagged `cs601pa1submission`, it is considered that you are late.

(d) The grading program checks every student's repository 120 hours after the submission deadline. If a version tagged `submission` is found, the grading program retrieves and grades that version.

(e) The grading program uses only the version tagged `cs601pa1submission`. It does NOT choose the higher score before and after the submission deadline. If a later version has the `cs601pa1submission` tag, this later version will be graded with the late discount. Thus, you should tag a late version with `cs601pa1submission` only if you are confident that the new score, with the late discount, is higher.

(f) The time of submission is the time when you push the code to the repository, not the time when the grading program retrieves your code. If you push the code after the deadline, it is late. Even though you push before the grading program starts retrieving your program, it is still considered late.

(g) You should push at least fifteen minutes before the deadline. Give yourself some time to accommodate unexpected situations (such as slow networks).

(h) You are encouraged to tag partially working programs for submission early. In case anything occurs (for example, your computer is broken), you may receive some points. Please remember to tag improved version as you make progress.

Do not submit any binaries. Your git repo should only contain source files; no products of compilation (.o, .exe, a.out etc.).

Do not send your code for grading. The only acceptable way for grading is to tag your repository.

Under absolutely no circumstance will the teaching staff (instructors and teaching assistants) debug your programs without your presence. Such email is ALWAYS ignored. If you need help, go to office hours, or post on the discussion forum.

1.2 Intel vector extensions

Refer to overview and Intel compiler specific documentation[1]. With g++ compiler, you may use the following compiler flags `-O2 -ftree-vectorize -fopt-info-vec-missed` to know a detailed report about all the for loops that were not vectorized.

Example: `__m256 m2 = _mm256_load_ps((b+k*n+j));`, this line defines a variable `m2` of type `__m256`. The variable's value is loaded from memory at address `b+k*n+j` (read: b[k][j], where b is a `const float [][]` of size `n`). Furthermore, 256bits or 8 float numbers are loaded into the `__m2` variable. Suffix `__ps` denotes

[1]The concepts and API naming conventions are generic

packed, **s**ingle-precision. You can have **s**calar, **d**ouble-precision and other options as well. For complete list, refer to the documentation.

Credits to Marcus Holm(marcus.holm@it.uu.se) for an excellent overview here (Read Sections 1 to 3)

2 Problem Statements

1. The starter file for this part of the assignment contains `matmul.cpp` that implements a C++ program for computing the product of two matrices as follows: `C=C+A*B`, where `C`, `A`, and `B` are square matrices of size `N`. The product `A*B` is computed and updated to matrix `C`, which is zero initialized. A simple matrix-multiplication consists of 3 nested loops (referred to as `ijk` loop) shown in the below pseudocode:

   ```
   for i=0 to N-1
    for j=0 to N-1
     for k=0 to N-1
      c_ij = c_ij + a_ik * b_kj
   ```

   a_{xy}, b_{xy}, and c_{xy} are elements of matrices `A`, `B`, and `C` respectively with x and y are suitable indices in the range $[0, N-1]$. You must reorganize the source code as per the folder-based arrangement (`inc`, `src` etc.) discussed in class. You may add new files.

   (a) Implement different versions of matrix multiplication function corresponding to every possible loop ordering and tabulate in your report the execution times and last layer cache data read misses. Edit the `Makefile` given to you so that a single command such as `make build_matmul_single` builds separate targets for each of the loop orderings (and supporting single-precision matrix multiplication). A command such as `make run_matmul_single` executes each of the targets built previously with a matrix size of 4096x4096 using single-precision data. Similar commands should be included in your `makefile` for target building and execution with double-precision data (i.e. `make build_matmul_double` and `make run_matmul_double` should be supported). You may edit the files provided as per your wish. However, your code must include flags for conditional compilation corresponding to compiling different versions of matmul implementations. Your implementation must be able to support `float` and `double` types with build-time flags `-DSINGLE` or `-DDOUBLE` corresponding to single-precision floating point multiplication or double-precision floating point multiplication.

   (b) Read about and use the library functions from BLAS library for performing matmul. You must implement two versions: a) using the available function for performing dot-product b) using `sgemm`. Tabulate in your report the execution times (with same matrix input size as earlier. Use single-precision float data only.) and last layer cache data read misses. *Discuss your observations.*

2. The starter files for this part of the assignment are `matvec.cpp`, `timeutils.cpp`, and `timeutils.h`. Read about and use the function calls / intrinsics for SSE3 (128-bit vector register extension) on x86 architecture. You will have to implement matrix-vector product using intrinsics. Hint: think of loop unrolling code that we discussed in class. Your implementation in the `matvec_intrinsics()` function should make calls to functions that look like e.g., `_mm_load_ps() etc.`. The resulting implementation would be equivalent to unrolling the inner loop 4 times. Edit the `matvec.cpp` file to implement this part of the assignment. Discuss your observations. Also edit `Makefile` so that after building the required target, you execute the target.

you need to make sure that the code compiles fine and executes on the ***hip*** *server.*

2.1 What you need to submit

Reorganized source code structure.

- (for problem statement 1:) Modified `matmul.cpp` and additional source code files that you may have.
- (for problem statement 2:) Modified `matvec.cpp` that contains different versions of matvec.

- (for problem statement 1 and 2:) A brief report that contains the details as described previously (include screenshots of your executions).

  You must tag your source code and submit as described earlier. The tag name to be used is: `cs601pa1submission`. *All tag names are case-sensitive..*