CME 213

SPRING 2013-2014

Eric Darve

Stanford University

SYLLABUS

People

Instructors:

- Eric Darve, ME, ICME
- Erich Elsen
- NVIDIA engineers

Teaching assistants:

- Christopher Fougner
- Haozhun Jin

Web sites

- Class forum: piazza. You may need to register manually. We will use piazza for all class communication. Make sure you do read important messages from the instructors.
- Class grades: coursework. You are automatically enrolled if you are registered for this class.
- Class material: Bitbucket repository (more on this later).

https://piazza.com/

https://coursework.stanford.edu

https://bitbucket.org

GRADING, HOMEWORK, PROJECT

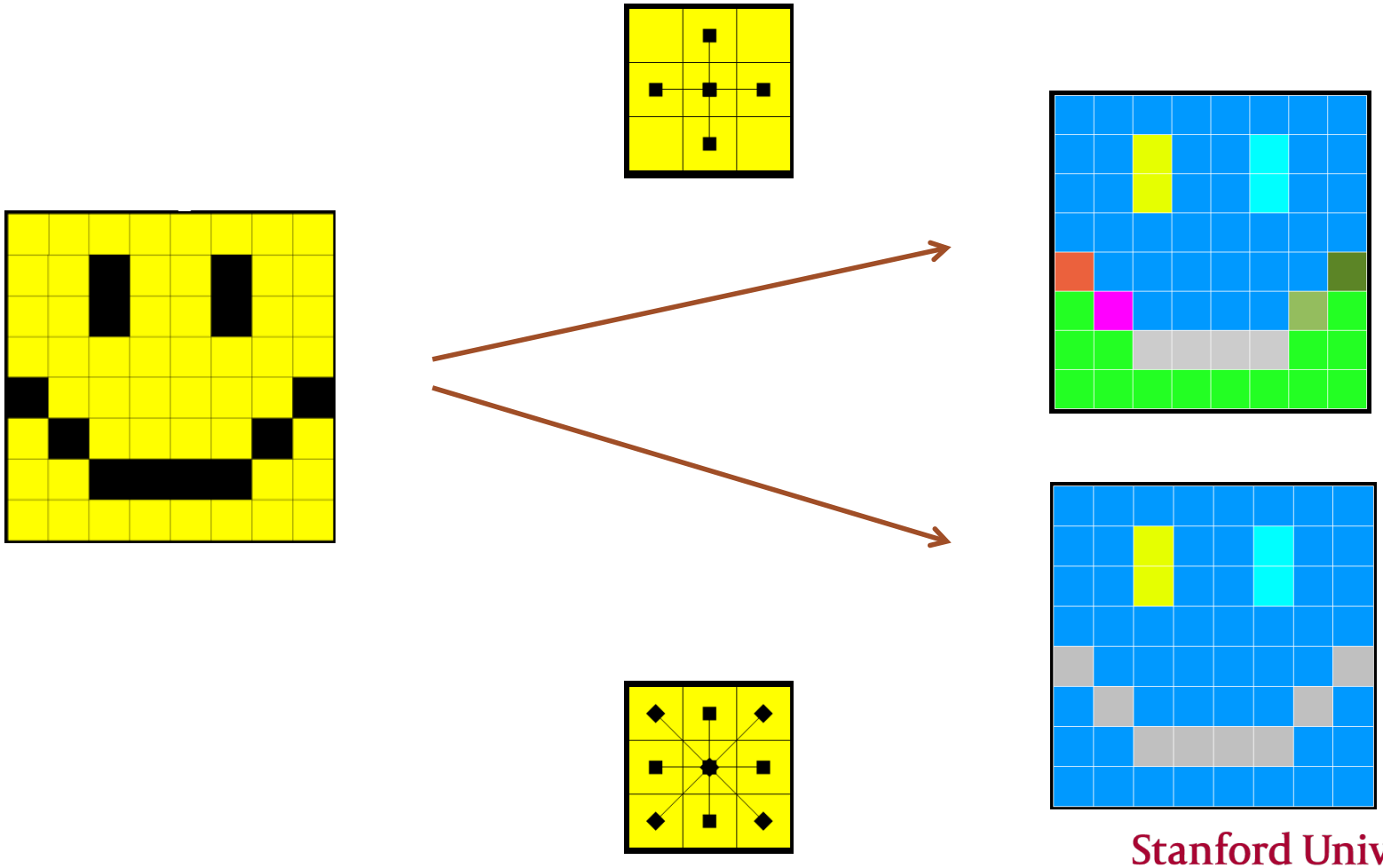
- 5 homework assignments: 70% of grade
- One final project: 30% of grade
- Curving algorithm. This may change at any time!
  - A: 55%
  - B: 40%
  - C: 5%
- Honor code:

> "that they will not **give** or **receive** aid in examinations; that they will not **give** or **receive** unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading"

- You cannot copy someone's computer code.
- The best way to make sure there is no honor code violation is to never show your code or paper to other students, and to not look at the code or paper of other students.

THE FINAL PROJECT

The final project will be an implementation of the connected component algorithm using CUDA and GPUs.



Books

Good news: all books are available electronically from the Stanford Library. Just go to:

http://searchworks.stanford.edu/

OpenMP, MPI, Parallel programming

- *Parallel Programming for Multicore and Cluster Systems*, Rauber and Rünger. Applications focus mostly on linear algebra.
- *Introduction to Parallel Computing*, Grama, Gupta, Karypis, Kumar. Wide range of applications from sort to FFT, linear algebra and tree search.
- *An introduction to parallel programming*, Pacheco. More examples and less theoretical. Applications include n-body codes and tree search.

OpenMP and Multicore Books

- *Using OpenMP: portable shared memory parallel programming*, Chapman, Jost, van der Pas. Advanced coverage of OpenMP.
- *Parallel Programming in OpenMP*, Chandra, Menon, Dagum, Kohr, Maydan, McDonald; a bit outdated.
- *The art of multiprocessor programming*, Herlihy, Shavit. Specializes on advanced multicore programming.

CUDA BOOKS

- *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Sanders, Kandrot
- *CUDA Handbook: A Comprehensive Guide to GPU Programming*, Wilt
- *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, Cook

- CUDA online documentation:

http://docs.nvidia.com/cuda/index.html

https://developer.nvidia.com/cuda-education-training

Uploaded reading material:

- CUDA_C_Best_Practices_Guide.pdf
- CUDA_C_Programming_Guide.pdf

WHAT THIS CLASS IS ABOUT

- We will focus on how to program:
  - Multicore processors, e.g., desktop processors: Pthreads, OpenMP.
  - NVIDIA graphics processors using CUDA.
  - Computer clusters using MPI.
- We will cover some numerical algorithms for illustration: sort, linear algebra, basic parallel primitives.

WHAT THIS CLASS IS NOT ABOUT

- Parallel computer architecture
- Parallel design patterns and programming models
- Parallel numerical algorithms. See *CME 342: Parallel Methods in Numerical Analysis*

What this class requires

- Some basic knowledge of UNIX (ssh, makefile, etc)
- Good knowledge of C and C++ (including pointers, templates)
- Proficiency in scientific programming, including debugging and testing

SCHEDULE

- Pthreads, OpenMP – Eric Darve
- CUDA – Erich Elsen
- CUDA – NVIDIA engineers
- MPI – Eric Darve

Homework

- Homework submission: use git to send us your work.
- All submissions will consist of:
  - Source code
  - Figures
  - Markdown file with text answers

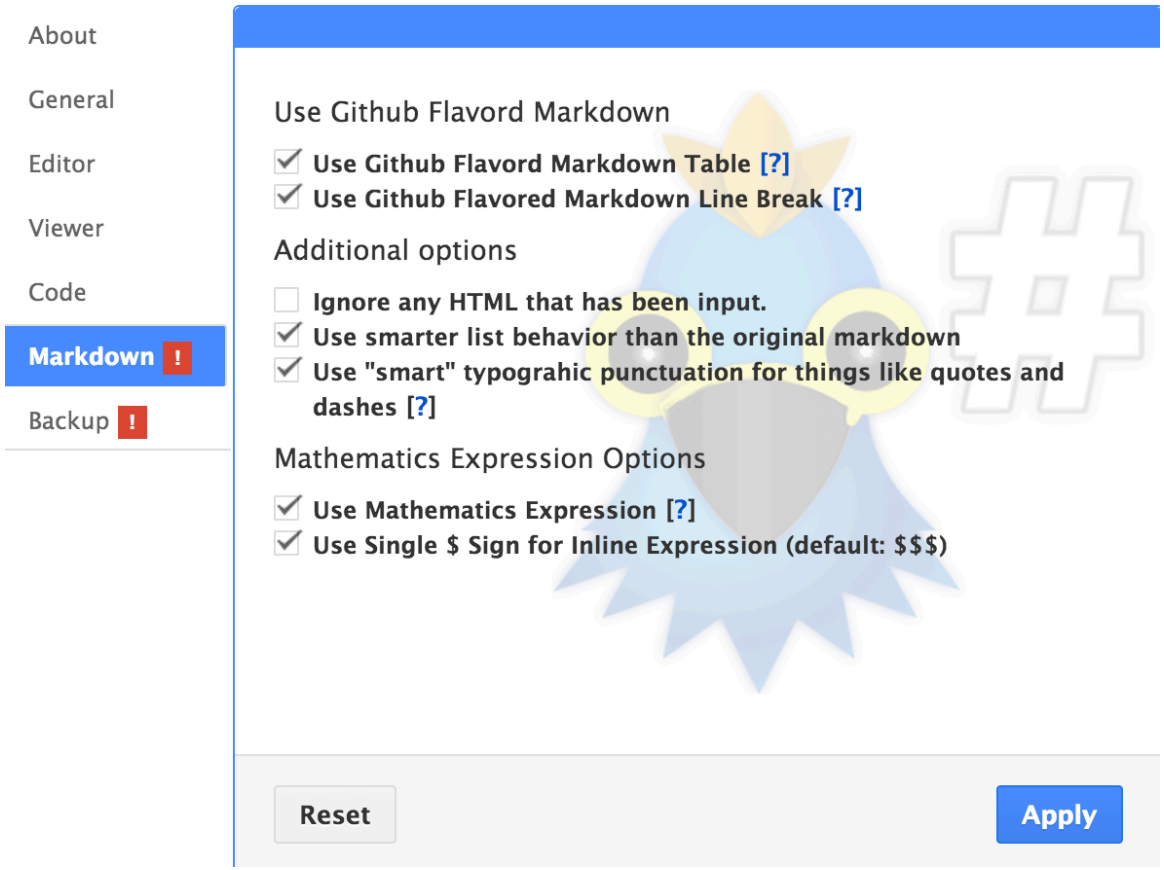
- Graders will add the grades and comments in your submission files.

- You use git again to get the updated files.

MARKDOWN FILES

- Most of the homework will be code and figures.
- For the text part, we will use Markdown.
- A markdown file is a text file with some special syntax for formatting.
- The recommended software to edit markdown files is stackedit.
- You can use any software you want as long as you produce a valid markdown file that we can open.

- Stackedit:
  https://stackedit.io/#
  Data is saved in your browser local files. You need to download the file to save it.
- Haroopad
  http://pad.haroopress.com/
  This software is less mature and not as easy to use as stackedit but it works directly with local files on your hard-drive. You need to change the options in Preference to get the math to work correctly.



Haroopad configuration

GIT TUTORIAL

- Homework will require you to use git.
- What is git useful for?
  - Collaborate on a project with other people, e.g., make collaborative changes to the same documents.
  - Keep versions of your work as you make changes. This allows reverting to a previous version, and maintaining different versions when necessary.

- How will we use git?
  1. Submit and grade homework
  2. Distribute material for the class

GIT CLONE

```
$ git clone https://bitbucket.org/edarve/cme_213_spring_2014.git
```

- This command fetches the directory **`cme_213_spring_2014`** and its content.

- **`edarve`** is the Bitbucket user to which this repository belongs.

- Online site:
- https://bitbucket.org/edarve/cme_213_spring_2014

GIT ADD

- Files need to be added so they can become part of the repository.
- Create README.md
- Add to repository.
- Important note: never add files to the repository that can be generated using files in the repository, e.g., ***.o** and **executables**.
- Why?
  - This is pointless.
  - In case of a dreaded merge conflict (which we won't cover in this short tutorial), there may be no way to fix it.
- For this class, we won't run into any of these problems, but this is a good practice you should learn from the beginning.

GIT COMMIT

- When we are happy with our changes, they need to be committed to the repository.

```
$ git commit -a -m "A message is needed for every commit"
```

GIT PUSH

- So far everything was local.
- We now push our updated repository to the cloud on Bitbucket.

```
$ git push
```

- Perform **`git pull`** on the instructor side.

DOING YOUR HOMEWORK

- Edit file
- `$ git commit -a -m "My update"`
- `$ git push`
- Work some more
- Add a new file
- `$ git add my_new_file`
- Keep working
- Almost at the end of my homework
- Done!
- `$ git commit -a -m "This is it; CME 213 is great"`
- `$ git push`

- Wait (graders are working with your files) …

- `$ git pull`
- Great! I got an A+ in this class!

CHANGING YOUR FILES AFTER THE DEADLINE

Don't do it!

- We will pull from the repository the last commit before the deadline.
- If you make some commits after the deadline, your changes will be merged with the changes made by the instructors (e.g., their comments and grades). That will be confusing for you since the comments will refer to the code before the deadline, which is no longer the current code.
- So don't do it.

DID I FORGET SOME FILES?

- Make sure you check that all your files have been committed and pushed before the deadline.
- How can you tell whether a file is missing or not?
- You can go to the web site but it does not refresh immediately.
- Is there a better way?
- `$ git status`
- Search for files in red under “Untracked files”

- If you know a file does not need to be in the repository, add their name to the file `.gitignore` in the directory.
- Demo with `missing_file.txt`.

GIT EXTRAS

- Git offers many more functionalities.
- One useful feature is the ability to revert.
- Sometimes, you start working on a new implementation and you realize that you went in a totally wrong direction.
- You want to go back to a previously working file and restart from there.

- WARNING! Some of these commands will overwrite some of your changes!!

- `$ git reset --hard HEAD`
- What it does: resets everything to the way things were **at the last commit.**

- `$ git reset --hard HEAD^`
- What it does: resets everything to the way it was **one commit before last.**

- Syntax: `HEAD HEAD^ HEAD^^ HEAD^^^`

GIT AMEND

For the *perfectionist*:

Let's say you did a commit but there is some horrible typo in your message:

```
$ git commit -a -m "No waay"
```

Do git log to see your shameful commit message.

```
$ git log
```

Not all hope is lost.

You can amend the commit.

```
$ git commit --amend -a -m "No way"
$ git log
```

Breathe a sigh of relief.

Git amend can make other changes to your commit as well: add a file, etc.

Merge

- This shows you why git is a powerful collaborative tool.
- People can work on the same file independently. Changes from many contributors can be merged into a single working file!
- Consider an instructor and student making changes to two different places in a file. The instructor can push his changes.
- Then the student can pull those changes and merge them with its own copy of the file.

FOR MORE INFORMATION

- We only scratched the surface of what git can do.
- See the git cheat sheet online.
- See the many tutorials on the web.
- Listing of several tutorials:

https://help.github.com/articles/what-are-other-good-resources-for-learning-git-and-github