Cursusinformatie

Inhoud

- **[Cursusbeschrijving / Course Description](vu.nl)**
  zoals gepubliceerd op vu.nl / as published at vu.nl

- **GPU Assignment**
  Bijgevoegde bestanden: Download for the assignment (44,925 MB)

Introduction

The goal of this assignment is to learn how to use many-core accelerators, GPUs in this particular case, to parallelize data-intensive code. Starting with provided reference code, the student is not just supposed to write CUDA equivalents of the given algorithms, but to write a fast parallel implementation in CUDA.
You are requested to implement an image processing pipeline: the pipeline takes a color image as its input, convert it to grayscale, use the image's histogram to contrast enhance the grayscale image and, eventually, returns a smoothed grayscale version of the input image.
The program must be benchmarked on the NVIDIA GTX480 [2] GPUs on the DAS-4 [1].

Contacts

You can send all your questions regarding this assignment to Alessio Sclocco (a.sclocco@vu.nl). You can also drop by in my office (science building, room U-421).

The Pipeline

The image processing pipeline used for this assignment is composed by four algorithms: (1) grayscale conversion, (2) histogram computation, (3) contrast enhancement and (4) smoothing. A brief description of the algorithms follows.

Grayscale Conversion

Our input images are RGB images, this means that every color is rendered adding together the three components representing Red, Green and Blue. The gray value of a pixel is given by weighting this three values and then summing them together.
The formula is: gray = 0.3*R + 0.59*G + 0.11*B

Histogram Computation

The histogram measures how often a value of gray is used in an image. To compute the histogram simply count the value of every pixel and increment the corresponding counter. The are 256 possible values of gray.

Contrast Enhancement

The computed histogram is used in this phase to determine which are the darkest and lightest gray values actually used in an image - i.e., the lowest (min) and highest (max) gray values that have "scored" in the histogram above a certain threshold. Thus, pixels whose values are lower than min are set to black, pixels whose values are higher than max are set to white, and pixels whose values are inside the interval are scaled.

Smoothing

Smoothing is the process of removing noise from an image by the means of statistical analysis. To remove the noise, each point is replaced by a weighted average of its neighbours. In this way small-scale structures are removed from the image. We are using a triangular smoothing algorithm, i.e. the maximum weight is for the point in the middle and decreases linearly moving far from the center. As an example, a 5-point triangular smooth filter in one dimension will use the following weights: 1, 2, 3, 2, 1.
In this assignment you will use a two-dimensional 5-point triangular smooth filter.

Accelerated Application

A sequential version of the application is provided, for your convenience, on the Blackboard page of the course.
There are no restrictions on how the accelerated filters should be implemented, as long as the functionality is preserved. However, the recommendation is to start from the implementation provided as an example.
You need to parallelize and offload to the GPU the previously described algorithms. The "Kernel" comment in the code indicates the part that must be parallelized.
There are no assumptions about the size of the input images, thus the code must be capable of running with color images of any size. The output must match the one of the sequential version; a compare utility is provided to test for this.
The output that is verified for compliance is the final output image (named smooth.bmp). You are free to save also the intermediate images, e.g. for debugging, but do not include the time to write this images in the performance measurements.
Although the application's text output is not verified for compliance, please keep a similar structure with the one of the sequential output, as it makes testing a lot easier.

Requirements

Implement a CUDA version of the image processing application. Be sure that your executable is named cuda and is placed in the gpu/bin directory. Makefiles are provided.
In the directory gpu/images 16 different images are provided for testing. The parallel application should produce a correct output for all these images. You are encouraged to test your application on more images.
Measure the total execution time of the application, the execution time of the four kernels and the (introduced) memory transfer overheads. Compute the speedup over the sequential implementation, the achieved GFLOP/s and the utilization.
Write a short report, in English, describing your design, implementation and performance testing of your accelerated application. Focus on the parallelization details and the applied optimizations. Before presenting the results introduce the experimental setup and what is measured. Use tables and graphs when presenting the results.
Last, but very important, please comment the achieved performance of the application, the causes behind the different types of behavior you have observed, and conclude on the scalability and potential for acceleration of the three different kernels.

Performance Requirements

There are two performance requirements about the CUDA implementation: (1) optimize (at least) the histogram computation and the smoothing filter (hint: use shared memory), and (2) the execution times (in milliseconds) measured for the computation of the largest image in the set (gpu/images/image09.bmp), must be below the following thresholds:
- Grayscale Conversion < 5.5 ms
- Histogram < 68 ms
- Contrast Enhancement < 8.7 ms
- Smoothing < 84 ms
- Total execution < 1023 ms

In case your submission does not meet the performance requirements, points will be deducted from your final grade. Note that a too large gap between the expected and achieved performance (e.g., more than 2 times slower than the given thresholds) might lead to a failing grade.

Bug Reports

If there are bugs in the reference implementation contact me. In case of confirmed bugs, a corrected version of the reference code will be uploaded on Blackboard.

Compiling and Running Your Application

Your code should be written in C++ and use the provided Makefiles for compiling. Examine carefully the file gpu/Makefile.inc as there are few variables to set.

To compile the code you need to load the CUDA module on the DAS-4 (module load cuda42/toolkit). You can also add this to your .bashrc file.
To run you code on the DAS-4 use prun like this: prun -v -np 1 -native '-l gpu=GTX480' /home/[user]/gpu/bin/cuda /home/[user]/gpu/images/image[XX].[ext]
For more information check the DAS-4 website as it has a dedicated page for GPUs [3]. The image processing library used is CImg [6].

Submission Rules

You have to submit both the code (preferably containing useful comments that illustrate how the application works) and the report. The report must be put inside the gpu/docs directory and be in PDF format.
Create an archive containing your working gpu directory and name it using your VUNet id (e.g. jj400), your full name (e.g. Jan Janssen) and the type of the assignment (e.g. gpu). In this example your archive would be named jj400_JanJanssen_gpu.tar.gz; we use an automated environment to check the assignment, so follow these rules strictly.
The assignment must be submitted using Blackboard. Within few days from your submission you will be notified whether the code has passed or not the compliance check. Note that the compliance test will only verify that the output is correct, and not the performance requirements. In case your code didn't pass the test, you can work on it more and submit a new version for evaluation. You are allowed a total of 3 submissions for this assignment.
In case your code did pass the test, you can still update your submission until the deadline and/or until reaching the maximum number of attempts (3). However, grading will be done only after the deadline has passed.
Please keep in mind that passing the compliance test does not guarantee a passing grade, it only marks the submission as ready for grading.

Grading

A correct implementation of a GPU-accelerated filter pipeline compliant with the requirements above (both for code and documentation) is graded with 8.
Up to 2 bonus points (to grade 10) can be given for extra work on implementation, optimizations, testing and benchmarking. Examples of extra work are: (1) implementing the pipeline with OpenCL and comparing it with CUDA, (2) implementing the pipeline with OpenCL and analyze the performance portability on CPUs, (3) applying different CUDA optimizations (like coalescing, using the shared memory, etc.) and showing their effect on the overall application performance, (4) extending the performance analysis including qualitative estimations of the performance.
Creative attempts at improving the parallel implementation, the performance and/or the analysis beyond the mandatory requirements are encouraged.
You get bonus points if you find interesting and/or creative ways to improve the parallel application (its implementation, performance, or analysis).
Important notes:
(1) Optimizing the sequential reference algorithm does not count as a bonus. However, in case of modifications on the algorithm used for parallelization, the sequential code has to be modified accordingly to allow for a fair speed-up calculation.
(2) Make sure that the basic requirements for the assignment are still met: additional features are only graded for working solutions.
The submission that is graded is the last one available on Blackboard at the time of the deadline, therefore make sure that your last submission is compliant, complete and correct.

References

[1] DAS-4
[2] NVIDIA GTX480
[3] DAS-4 GPU programming
[4] CUDA documentation
[5] OpenCL documentation
[6] CImg

•

JAVA Assignment

Bijgevoegde bestanden: Download for the assignment (5,92 MB)

Rubik's Cube

This assignment for the parallel programming course consists of writing a parallel program in Java using the Ibis system and running it on the DAS-4.
A Rubik's Cube is a puzzle designed by Mr Rubik. Solving it requires twisting the cube until every side is of a single color. Alternative solutions are taking it apart, or taking off all the little stickers, then putting them back in the solved state ;)

- Rubik's Cube (Wikipedia)
- Online solver (not optimal)
- Official Site

The goal of the assignment is to write an application which is able to determine the shortest number of twists possible, as well as the number of ways to solve the cube, given a certain cube. A sequential solver is available in this page. The application includes a build.xml file for building the application with ant, as well as a doc and bin directory to hold any documentation, scripts and external dependencies for your application.
Although the application randomly creates cubes it is deterministic, with a given setting and seed it will always generate and solve the same cube, allowing for easy benchmarking of the application. The different command line parameters drasticly change the application runs. If the default parameters do not work for you, try to adjust them somewhat.

Requirements

Convert the given sequential version of the solver into a parallel application that it is capable of running on multiple DAS-4 machines in parallel. To communicate, use the Ibis Portability Layer(IPL). Implement a work queue to handle passing work to the machines. You can use a single work queue, located on one of the machines. You can, for instance, elect one of the machines as "master" using the election mechanism of the IPL.
Benchmark your application by running it on the DAS-4 system. Try to get as close to perfect speedups as you can. Also, try to explain the reasons why you get lower than perfect performance. Test your application on 1, 2, 4, 8 and 16 nodes. Make sure you use prun to submit your job, even for jobs using on only one machine. If you run a job on the frontend instead of a node, you will both overload the frontend, which is used by a lot of people, and get useless performance measurements! You can use the timers provided by Java (System.currentTimeMillis or evenSystem.currentTimeNanos) to measure the performance of sections of your code.
Optimize your program such that the grain size of the leaf jobs is controlled to improve performance (i.e., do not put very small jobs in the job queue, but calculate them directly).
Write a small report (in English, about five pages) describing your experiences with Java, the difficulties you encountered and how you solved them. Include measurements of your program. Report elapsed **execution times** as well as **speedups**. Don't just show numbers, but present a **speedup graph** as well. If the program does not achieve linear speedup, give an explanation **including a performance breakdown** for the slowdown. The document should also describe how you implemented the programs and *how* you measured their performance. The programs themselves should contain useful comments that illustrate how they work.

Compiling an Running your Java programs

You can compile your Java programs with javac or with ant using the provided build.xml. **Be sure to use the "module" command on the DAS-4 to set the applications used.** For more information see the DAS-4 section of this website.

To run a parallel program on the DAS nodes, you need to make use of the prun script (see the "DAS-4 for PPP users" in the Course Documents, for more information on the DAS-4 and prun). Furthermore, to run your application, you can simply use java or use the provided java-run script (which adds all jars to the classpath automatically). It expects to find a lib directory in the current directory.
sequential on one machine: ppp@fs0 ida $ prun -v -1 -np 1 bin/java-run rubiks.sequential.Rubiks
ipl version on 8 machines: ppp@fs0 ida $ prun -v -1 -np 8 bin/java-run rubiks.ipl.Rubiks
bonus assignment, 8 machines: ppp@fs0 ida $ prun -v -1 -np 8 bin/java-run rubiks.bonus.Rubiks
For more information on compiling and running IPL applications, see the user's guide and programmer's manual of the IPL included in the distribution (and the assignment template).

Submitting

You have to submit your code (preferably containing useful comments that illustrate how the application works) and the report.
Important: Because we use automatic test scripts to test and benchmark your submissions, you must strictly follow the instructions below.

5. Make sure that your submission has the **exact same** directory structure as the provided template in /home/ppp/pub/ on the DAS-4. The main function of your program should be in the Rubiks.java file of the rubiks.ipl package.
6. Also, make sure that your parallel programs give the **exact same output** as the sequential program. We compare your application's output with the correct output with the *diff* command. This also includes the destinction between printing to standard out and standard error. If there is any difference (except for the run time you print), your submission will be rejected!
7. If you reimplement (parts of) your program for the bonus assignment, please put your new program in the ida.bonus package. Additionally, if you add extra command line options to the application, make sure that you use reasonable default values, because our automated test scripts do not know your command line options.
8. To help you with these checks a sanity-check script is provided in the bin directory of the template. Please do not submit anything which does not pass this test. The script has a single parameter, the address of the ipl server (for testing the ipl assignment). Note that the script does not check whether your application is behaving correctly, it just checks whether the output is formatted correctly.
9. Please structure your code and scripts so that your submission includes all needed dependencies (like the IPL), and simply running "ant" in the root of your applications leads to the compiling of your application. The standard "java-run" script should also work. One way to ensure all this is to not change the template :)
10. Create the report as a PDF file, and make sure you place the file in the pre-created docs directory.
11. Edit the build.xml file and fill in your name and your VU net-ID in the appropriate variables at the top of the build file. Then create a "distribution" of your code suitable for submission by using the **"ant dist"** command in the root of your applications. **Submissions not created with the "ant dist"**

command will be rejected Make sure that the entire distribution compiles correctly, including the bonus assignment. If you tried, but did not finish the bonus assignment, please **do not submit unfinished code** that lead to compilation errors.

12. You are allowed a total of 3 submission attempts for the Java assignment. We run a check on your submission(s), and notify you of the results (i.e., pass/fail for the sanity checks) within a few days. Note that passing the sanity check does not guarantee a passing grade, but guarantees the assignment can be considered ready for grading.

Documentation

- Java 1.6 documentation
- The ipl section at the Ibis website
- The reader
- The code for the sequential implementation, available on this page

Grading

A correct implementation of all the requirements above is graded with 8. Up to 2 bonus points (to grade 10) can ben given for extra work on implementation, optimizations, testing and benchmarking. For example:

17. Replace the sequential workers of the default assignment by multithreaded workers that can make use of multi-core processors of the DAS-4.
18. Implement work stealing instead of the single centralized work queue.

To get the bonus points, it is up to your fantasy on how you improve the parallel application or your evaluation of the application. Optimizing the sequential algorithm however, does not count as a parallel optimization for the bonus assignment. If you reimplement (parts of) your program for the bonus assignment, please put your new program in the ida.bonus package. This way, it is easier for us to grade it, and additionally you do not break the entire assignment when you make a mistake in the extra code.

TODO list

In order to complete this part of the practical, please follow the following receipt:

- Get the sequential version of the IDA* Puzzle solver from this page.
- Create a parallel application using the IPL. The IPL is already provided in the template.
- Test your application on the DAS, using prun (see the DAS-4 site for more info on the DAS-4 and prun).
- Benchmark your parallel application
- Write a report about the practical. Include the difficulties you encountered, how you solved them, and the performance of the resulting application. Include both run-time and speedups graphs and listings
- Optionally you can try to get the bonus for this assignment. Include a description of your extra work for the bonus in your report.
- Test your assignment with the sanity-check script
- Create a "distribution" containing your code and documentation using the provided "dist" ant target (set the "name" property at the top of the build.xml file to your name!) and submit it using blackboard.
- Wait/check for the pass/fail notification of your assignment, update the code and/or report if wanted/needed, and re-submit. Note that you have a total of 3 attempts to submit your final solution. Only the last submission found on the site at the submission deadline will be graded.

•

MPI Assignment

Bijgevoegde bestanden: mpi_2k12-13.tar.gz (555,926 kB)

Context

The Ministry of Transportation of The Kingdom of Far Far Away has come up with a revolutionary idea: implementing a centralized information system for all the roads in the country. The system requires live updates for the following information:
-- the total distance on all practicable roads in the country
-- the "on-the-road" diameter of the country
-- travel directions for the shortest path between any two cities in the country

The roads are registered as a list of direct, uninterrupted roads between cities (i.e, if the way from city A to city B passes through city X, the list will record (A,X,20km) and (X,B,15km), and not (A,B,35km)). The distance of each such direct road is also stored. The roads are not directed - i.e., the distance from A to B is the same as the distance from B to A.

The system has to run in real-time: due to accidents or road-work, the list of practicable roads keeps changing, but the information service should always be up to date. Thus, a competition has been started for the fastest solution one can provide for such a on-line system.

Requirements

The contest is managed by the King's Informatics Commision (KIC), which lists the following requirements:
-- the solutions should calculate the correct up-to-date information for the road distances, total road distance, and
-- the solutions should be parallel and executable on a cluster
-- implementation should be based on C and MPI.
-- the performance benchmarking has to be done on the DAS4 cluster.
-- at least five road lists, provided by KIC, should be used for benchmarking.

Submission

The solutions to be submitted to the KIC should contain the following:
-- source-code for the MPI implementation and the required scripts for building the program
-- source-code for the sequential implementation and the required scripts for building the program
-- executing any of these two solutions should provide timing information in a given format.
-- a brief report explaining the algorithm, the important technical details of the implementation, and the benchmarking performance results. In this context, the report should answer the following questions:
* How long does a complete update of the information system take (for each road-list)?
* How does the update time change on larger/smaller clusters (for each road-list)?
* How does the update time compare, for different cluster sizes, for all road-lists?
* Does the cluster-based solution always work better (for each road-list) compared with the sequential one? How much better?
* How large should the cluster be for the update to be done in real-time (i.e., the time for the update should not exceed a threshold of 1s) ? Is this the same for each road-list?

Bonus features Interesting solutions and/or additional features can obtain bonus points, which will position the solution higher in the rankings. Examples include (but are not limited to) the following:
-- non-complete updates: receiving an update of the practicable road status will only trigger a partial update, and not a complete update.
-- generalize the solution for directed roads (i.e., the distance from A to B might not be the same as the distance from B to A).
-- more detailed analysis of the performance
-- answering road planning questions such as "how should the roads be built such that the information system is very fast?"

Resources

KIC provides each competitor the following resources (packed in an archive):
-- a skeleton for the application
-- a brief tutorial on using DAS4
-- five road networks for benchmarking
-- an open-source generator of road lists, for eventual additional testing

Furthermore KIC offers each participant several services:

-- the opportunity for one consultancy session: if a proposed design of the solution is submitted before January 10th, 2013, KIC will provide feedback and suggestions on the design. A design proposal should include pseudo-code for the solution and a brief explanation on how the system would work.
-- three chances to submit a final solution. Each submission will be checked for compliance against the correctness requirements, and the author will be informed of the outcome. However, no grading/ranking will be made. The grading/ranking will be provided after the deadline of the competition, set on 01/02/2012, at 23:59 CET.
-- the latest one of the submissions of each author will be considered for the final ranking.

Ranking

The final result for each author will be a grade which will take into account:
-- if the submission complies with all the requirements.
-- the achieved performance.
-- the completeness of the report.
-- the ranking of the achieved performance in the list of correct submissions.
-- any additional features that the authors have included.