Cursusdocumenten

Inhoud

-

General PPP Information

Goals

The Parallel programming practical course teaches students several techniques widely used in parallel programs. The first assignment uses message-passing to for an SPMD application; students get familiar with C and MPI. The second assignment focuses on a Master-Worker application, and requires the students to use Java. The third and last assignment focuses on the use of an offload model and the use of graphical processing units (GPUs) as accelerators; thus, students are required to program GPUs using CUDA or OpenCL.

Basic facts and rules

For this practical, students work **on their own,** to complete **three assignments,** one with C and MPI, one with Java, and one with GPU-accelerated code, using OpenCL or CUDA. The practical work consists in implementing and benchmarking parallel applications using the DAS4, as well as writing a brief report for each one of the assignments (in English). The practical officially starts on November 1st, when the first assignment will be published. The other two assignments will be published later.
The deadline for this practical course is February 1st, 2013, 23:59:00 CET.
There is no introductory lecture, and there are no lab sessions for this practical. Students can start working as soon as the assignments are posted online. Questions about the assignments can and should be asked via email, and addressed to the person listed as "contact" in each assignment:

- The MPI assignment: Ana Lucia Varbanescu (analucia@cs.vu.nl , A.L.Varbanescu@vu.nl);
- The Java assignment: Alessio Sclocco (a.sclocco@vu.nl);
- The GPU assignment: Alessio Sclocco (a.sclocco@vu.nl).

For more information about the MPI, Java, and GPU assignments, as well as more details on the DAS4 cluster, benchmarking and reporting your results, check he additional course documents.

Registration and accounts

If you have been enrolled for the course on the Blackboard by 31/10/2012 at 12:00, you should be receiving your DAS4 account information by email directly from the cluster administration. If you did not receive your account information before Monday, November 5th, 2012, or if you experience problems with logging in, please contact Ana Lucia Varbanescu (analucia@cs.vu.nl , A.L.Varbanescu@vu.nl).

Submitting and grading

To pass this practical, a student has to pass all three assignments (i.e., each assignment has to have a grade of 5.5 or higher). Assignments are submitted via the Blackboard. Submissions are sanity-checked for compliance (i.e., correctness of the compilation, running, output and not excessive execution time). For each submission, students receive a "pass/fail" result for this sanity check within a few days.
An assignment that fails the sanity check will NOT be graded, and it is considered failed. After a failed sanity check, the assignment can be corrected and resubmitted. The total number of submissions for each assignment is 3.
An assignment that passes the sanity check is not guaranteed a passing grade. The sanity check it only guarantees that the assignment will be further graded. An assignment that passes the sanity check can still be corrected/improved and resubmitted, provided that this resubmission is within the 3 allowed attempts. Note that in case multiple submissions of the same assignment have passed the sanity check, only the **last** submission (according to the blackboard site) is graded.

Rules for grading

Each assignment has a set of requirements. Meeting all these requirements leads to the grade 8. However, students that improve the design, implementation, or analysis of their solution beyond the assignment requirements can gain up to 2 bonus points per assignment. Each assignment has suggestions for improvements that can count for bonus points, but we strongly encourage other creative attempts.

Important
One can get bonus points for interesting and/or creative ways to improve the parallel application (its implementation, performance, or analysis). However, optimizing the given (reference) sequential algorithm, **does not count** as a bonus. Furthermore, **any added (bonus) features are only graded for solutions that still meet the basic requirements**! The grade for each assignment is a combination of the grades for the design and implementation, the achieved performance, and the report (with a special focus on the results/performance analysis). Therefore, do not neglect any of these three components!
The final grade for the practical is calculated as an average all three assignments, rounded to the closest half-integer (6.3 is 6.5, 6.7 is 6.5, 6.8 is 7).

Special cases

Students who re-take this course because they have failed one of the two/three parts of the assignment in the previous years are allowed to keep their passing grade from the previous attempt. For example, if a student registered in 2010 got a 7 for the MPI assignment and a failed the Java and GPU assignment, he/she is allowed to keep the MPI grade, thus only needing to solve and submit the Java and GPU assignments. Still, students are also allowed to try to improve and resubmit the already passed assignment. For example, in the previous case, the student can opt to re-try the MPI assignment as well.
Students who re-take this practical to improve their grade have to improve and resubmit all assignments (i.e., a student that has passed the course in a previous attempt, cannot choose to keep only one or two of the older grades).

- ## DAS-4 for PPP users

The Distributed ASCI Supercomputer 4

The system that you should use to run and test the parallel applications is called DAS4. It is a wide-area distributed cluster designed by the Advanced School for Computing and Imaging (ASCI). See more details on the DAS4 website.
For this course, you should use the cluster located at the VU (see details here).

Cluster	Nodes	Type	Speed	Memory	Storage	Node HDDs	Network	Accelerators
VU	74	dual quad-core	2.4 GHz	24 GB	30 TB	2*1TB	IB and GbE	16*GTX480 + 2*C2050

Note that, for developing, testing, and debugging, the other clusters in DAS4 may also be used (especially if the VU cluster is very busy). However, the reported performance numbers have to be those obtain for runs on the VU cluster.

Remote login

To run and test parallel applications on the DAS4, you have to login on the fs0.das4.cs.vu.nl (the DAS4 fileserver/"head-node") first. You may use:
localhost prompt> ssh fs0.das4.cs.vu.nl
ssh (Secure Shell) automatically sets your display environment variables. Note that you are only able to log on the fs0 from within the faculty network.

Loading modules

To use the DAS4 you must setup your environment using the module command. Add the following lines to the .bashrc script:
  module load openmpi/gcc   module load java module load prun

Compiling and running parallel applications

Compiling your parallel program is different for each programming language, so this is described per assignment. There are some differences when running your specific assignments as well, which are also discussed fora each assignment.
In general, to execute a program in parallel on the DAS (i.e., "to run a job"), students will use use the prun command. This allows you to reserve nodes and use them "exclusively". The prun syntax is:
prun [*prun options*] *-processes-per-node* -np *cpus* [*specifics for application-type*] [*application command-line options*]

So, for example, to run your MPI program on 4 DAS nodes, using a single process per node, you should use:

```
DAS prompt> prun -v -1 -np 4 ./your_program
```

To run the same program on 2 DAS nodes, using 2 processes per node, you should use:

```
DAS prompt> prun -v -2 -np 2 ./your_program
```

The prun option -v allows you to see some information about the nodes you are running your application on.
The -1 / -2 / -processes-per-node option allows you to decide how many instances of your program you want to use for each node. In other words, the -1 option is used to reserve a whole machine, and not just a processor (remember that the DAS consists of SMP nodes). If the -1 flag is omitted, prun might run your job on two machines, using two processors per machine. For the performance measurements in this course, you should always use the -1 option.
The [specifics for application type] are discussed separately for each assignment.
Whenever you want to terminate a running application, you can simply press *ctrl*-c. The command pkill allows one to kill zombie processes. It is probably best to run pkillevery once in a while to make sure no leftover processes are running on the DAS, which may hurt performance measurements of other DAS users. The preserve command can also be used to kill jobs.
You can also use the command

```
DAS prompt> preserve -list
```

in order to obtain additional information on the current status of the DAS processors. To learn more about prun, preserve and pkill, examine their manual pages with man.

System abuse

Running your applications without prun will start your application on the DAS4 fileserver (fs0.das4), and will **NOT** run the application in parallel. Also, since many other DAS users are working on fs0.das4, running a heavy program there will hurt general system performance and is considered to be an **abuse** of the system! When testing a strictly sequential program (i.e.. your reference sequential code), you should run it with prun -v -1 -np 1 ./a.out for the same reason. More importantly, the fs0.das4 fileserver simply cannot be compared to one individual DAS node because they are completely different architectures. Thus, any performance results (like speed-up, for example) comparing timings on the fileserver (i.e., without prun), with timings on the nodes will be incorrect.
Please comply to the DAS4 Usage Policy.

-

Benchmarking your application

How to measure application performance

If you are ready to make performance measurements, you should keep the following points in mind:

- Times should always be measured by using ***'wall clock time'***, which covers the absolute total time, including idle time. Simply store the clock time in a temporary variable at the start of the measured application kernel/region/section, and determine the difference between the clock time at the end of the measured region and the 'start time' variable.
- Do not include application (de-)***initialisation*** in your measurements. You may measure that as well, of course, but do not add them to the application's kernel timings. Data distribution and data gathering (as being part of the application's initialisation and de-initialisation) are not considered to be part of the application's kernel. Only when they occur multiple times in-between calculations, inside the kernel, you have to include them in your timing measurements.
- Do not print (debugging-)***text on the screen*** when measuring because it can have a big impact on the execution time.
- Try to minimize ***data copying and data transferring*** in your application kernel. Since data copying is a relatively slow process, and storing data multiple times can have a negative impact on caching performance, the application's overall performance will decrease.
- Remember to ***compile and run your application with all possible optimisations on***.
- When running your application several times, you will notice that it will not execute exactly as fast in all cases. Therefore, to make accurate measurements, ***run your application multiple times*** (at least 3 times, even more times in case you intend to leave out extreme values) and determine the ***mean*** timings.
- If you decide to take measurements for a parallel program on all CPUs individually, use the one CPU that takes the ***longest*** time to complete. For example, when you find that for a dual-processor application, CPU1 takes 3.21 seconds to finish its operations, while CPU2 had already finished in 2.89 seconds, you must take the timing measurements for the first processor since that one specifies the absolute application execution time.
- When you are computing speedups, make sure you are comparing the multi-processor application agains the **provided sequential** program. Also, remember that***you must always use prun*** to run your programs. Even the application variants running on just 1 CPU should be started using prun -v -1 ./a.out 1 or something similar.

- Use various problem sizes. Measure the performance for small, medium, and large problem sizes, and comment on the performance results (we recommend you to also consider different border-line cases, and make sure your application executes correctly). When choosing the ***maximum problem size*** you might want to keep in mind the parameters of the DAS nodes, for example the amount of memory or cache; to find these out, you can use commands like :

```
cat /proc/cpuinfo          vmstat        free
```

- When reporting your performance results, do explain the experimental set-up: the number of nodes and/or cores you have used, the problem sizes, and the what has been measured exactly. Do not forget to report both the numbers for the execution times and the speed-up graphs; include all other relevant measurements you have made in additional tables/graphs. However, do not omit to explain the relevance and the results for each of the included graphs and/or tables.

Writing a report.

How to write a report

For each assignment, your report should contain an analysis of your parallel solution for the given problem. This is a list of questions that should help you to write a good report.

- What was the problem to solve and which algorithm(s) you used?
- In case you changed the sequential program, what were the changes?
- What is the theoretical estimation of the performance of the parallel program? Note that the assignments might need different techniques to estimate the expected parallel performance.
- What are is the experimental set-up? Always create a complete experimental set-up, explain how you decided what measurements to take and why, and what is the relevance of each experiment or set of experiments.
- What was the procedure of taking measurements? Explain what you have measured, why, and how.
- What were the results of your measurements? Include both numbers and graphs. Execution times and speed-ups must be included in your report; however, do include (in separate tables and/or graphs) any additional relevant measurements that you've taken and analyzed.
- What do the numbers say about program performance? Explain all the results you include in the report; any unexplained graph and/or table are basically useless for the reader, and therefore will not be considered for grading your report.
- How do the results compare to the theoretical estimation? Focus on explaining the (eventual) gaps: why they appear and how can they be alleviated.
- Is there any slow-down? Why? Is there any superlinear speedup? Why? Is there a flattening of the speed-up curve? Why?
- In cases when reporting utilization is important/interesting (see GPUs, for example), what are the reasons for low utilization? Can utilization be improved? How?
- How does the performance scale in case there were more processors?
- What are the conclusions of your work? What is your experience with the particular language/programming model that the assignment targets? Any generic difficulties you have encountered? Any guidelines you've extracted from your work?

Note that this list is not meant to be exhaustive, but it provides a basic template for you to use when building your report. Note that the report should not exceed 5 pages (graphs and tables not counted). Brevity is appreciated, but do not omit the interesting details about your implementation and results.
Practical Notes: Only reports in **PDF** format will be accepted. To create speedup graphs you can use any tool but keep the axes scaled evenly. We recommend Gnuplot.

Frequently asked questions (FAQ)

General

I found a bug! What do I do ?
Please send an email to the person listed as "contact" in the assignment you found the bug in.

The sequential algorithm used in the assignment seems to be less than optimal. Can/Should I optimize it in my parallel version?
No, there is no need to optimize the sequential algorithm. This course tries to teach how to parallelize an application, not how to implement applications from scratch. Moreover, if you do optimize the application, please calculate speedups compared to the optimized algorithm, not the original. Otherwise, it wouldn't be fair.

Can I use another algorithm than the suggested one to parallelize the sequential application?
Yes, it's possible, as long the your solution is correct, efficient, and well described in your report. It would be interesting if you implemented both algorithms, compared them and analyzed why one of them performs better. Look in the literature as well. Using a different algorithm is possible, but you are not expected to do it: implementing algorithms suggested in the webpage (and/or discussed in the class) is enough.

I've measured the speedup of my program and it is superlinear. Is this correct?
It depends. Verify first that you measure time and compute the speedup correctly and use large enough problem size. (Hint: this webpage and lectures explain it all). Analyze and explain in your report why there would be superlinear speedup, try to estimate it theoretically.

I've measured the speedup of my program and it seems to not be good enough. What should I do?
Create a performance breakdown of your program. Understand what processors do, why they are idle. When you know the answers, you might redesign and recode your solution. If inefficiencies are inevitable, explain and estimate the slowdown in your report.

I've measured the speedup of my program and it seems ok, close to linear. Is that it?
Well, almost. We also require you to analyze the performance of the program, estimate it theoretically, understand and explain why the speedup is good. Does your speedup scale?

Does the speedup of my program count in the grade?
Yes, it does. However, it is not the only thing that matters ...

Does my code have to work for one node?
Yes!

Does my code have to work for a number of nodes which is not a power of two?
Yes!

I already started this practical last year or earlier, can I do one of the old assignments?
No, you can only do the current assignments. However, any completed assignments - i.e., previous years' assignments for which you have already gotten grades - can be still considered passed, and the old grade can be imported. Thus, if you already have a passing grade for any of the assignments (GPU, Java, or MPI), you do not have to re-do those assignments.

MPI

How do I abort my program? What happens if one processor exits and the others don't?
Use MPI_Abort(...) to abort the execution of the parallel MPI program.

Java

Can I use the java.util.concurrent package?
Yes, you can. However, it is discouraged to use these classes as this package usually only leads to overly complex code. The standard synchronized mechanism of Java should be enough to implement the assignment.