Department of Information and Computer Science
Utrecht University

INFOB3CC: Parallelism

Trevor L. McDonell

December 15, 2022

Introduction

Use these tasks to practice the topics of the lectures. You may have to do some research to read up on terms or topics not (yet) covered in the lectures.

Questions

1. We discussed Moore's curve and the difficulties in increasing processor performance, specifically the limitations in power dissipation, memory speed, and instruction-level parallelism. To gain more understanding on these topics read the following articles and answer the questions:

   - The Free Lunch is Over: A Fundamental Turn Towards Concurrency in Software
     `https://edu.nl/yrk68`
   - The Future of Computers (Part 1): Multicore and the Memory Wall
     `https://edu.nl/tvcyd`
   - The Future of Computers (Part 2): The Power Wall
     `https://edu.nl/9khdt`

   (a) Why is it unlikely that we will ever get a 10GHz CPU?

   (b) Sandia Labs reports that as the number of cores increases, CPU power *decreases* at an exponential rate. Why?

   (c) What does "embarrassingly parallel" mean?

   (d) According to the article, caches are reaching practical limits. Instead of using caches, why do we not directly increase memory bandwidth; for example, by increasing memory frequency?

   (e) In what way(s) does temperature affect performance?

   (f) What is Dennard Scaling?

   (g) What is Dynamic Voltage Scaling?

2. Definitions of concurrency and parallelism.

   (a) What is concurrency? Give an example of a problem which can be solved using concurrency.

   (b) What is parallelism? Give an example of a problem which can be solved using parallelism.

   (c) What does it mean for an application to be concurrent but not parallel?

   (d) What does it mean for an application to be parallel but not concurrent?

3. Improving applications through the use of parallelism.

   (a) In what ways can parallelism be used to improve the performance of an application?

   (b) What does Amdahl say about the maximum performance improvement which can be achieved by a parallel application?

   (c) What does the Gustafson-Barsis formula say about the maximum performance improvement of a parallel application?

(d) Suppose we have a program which has a serial portion of 5%. What is the maximum possible speedup when running this program on 20 processors according to Gustafson? What is the speedup according to Amdahl? What definitions of the word "speedup" are used here? What are the assumptions that each of these approximations make?

4. Methods of parallelisation.
   (a) What is task parallelism? Give an example.
   (b) What is data parallelism? Give an example.
   (c) What is the difference between task parallelism and data parallelism?
   (d) What is the difference between parallelism and concurrency? Give an example of a problem/computation which is parallel but not concurrent.

5. Applications of parallelism.
   (a) Amdahl's law considers the execution time of a program as a combination of (a) time spent doing serial work and (b) time spent doing parallel work (and nothing in between). Explain what this implies for the maximum parallel scalability of a program.
   (b) Can you think of an example of a (part of a) parallel program that does get faster when more processors are added, but not linearly (i.e. it does not parallelise perfectly)? In other words, can you think of a program for which the assumptions of Amdahl's formula are not satisfied?
   (c) What does "embarrassingly parallel" mean? Give an example of a problem/pattern which is embarrassingly parallel.
   (d) What is the difference between throughput and latency? Consider throughput and latency both for an entire application, and for execution of instructions in a processor. Give examples of how parallelism can be used to address each of these concerns.
   (e) How does a "divide-and-conquer" algorithm work? Give an example of an algorithm which can be implemented in this way.

6. Hardware design of CPUs and GPUs.
   (a) In what ways do the designs of CPU and GPU hardware differ?
   (b) What are the performance considerations relating from the differences in CPU and GPU hardware?
   (c) What kinds of computations are best suited for the CPU?
   (d) What kinds of computations are best suited for the GPU?

7. We discussed GPU hardware and its programming model (CUDA, OpenCL, etc.)
   (a) How does the GPU programming model map to the underlying hardware?
   (b) Give an example of how code in a sequential loop could be reimplemented to run on the GPU.
   (c) What are the consequences of having deeply nested conditionals in kernel code? Compare the case of (nested) `if` statements and a `while` look, such as that which can be used to compute elements of the Mandelbrot set.
   (d) What methods exist for communication and synchronisation between GPU threads? What are the restrictions on these and why is this?
   (e) What is thread occupancy and when is it an important consideration?

8. The canonical implementation of a spin-lock has the following form:

```
do {
  old = atomic_exchange(&lock[i], 1);
} while (old == 1);
```

```
/* critical section */

atomic_exchange(&lock[i], 0);
```

Here `lock` is an array of integers where a value of zero at index `i` indicates that the element at that index is unlocked, and a value of one means that another thread currently has the lock on that element. The initial loop repeatedly attempts to take the lock at index `i` by writing `1` into the slot at that index. Once the `old` state of the lock returns `0`, this thread has acquired the lock and continue to the critical section, after which it releases the lock by writing `0` back into the slot.

Will this code execute correctly when run on the GPU? Motivate your response.

9. What is the difference between correlation and causation?