Teaching Concurrent and Distributed Programming With Concepts Over Mathematical Proofs

1st David Marchant
Niels Bohr Institute
University of Copenhagen
Copenhagen, Denmark
david.marchant@nbi.ku.dk

2nd Carl-Johannes Johnsen
Neils Bohr Institute
University of Copenhagen
Copenhagen, Denmark
carl.johnsen@nbi.ku.dk

3rd Brian Vinter
Neils Bohr Institute
University of Copenhagen
Copenhagen, Denmark
brian.vinter@nbi.ku.dk

4th Kenneth Skovhede
Neils Bohr Institute
University of Copenhagen
Copenhagen, Denmark
kenneth.skovhede@nbi.ku.dk

***Abstract*—This paper describes how a concept-based approach to teaching was used to update how concurrent and distributed systems were taught at the University of Copenhagen. This approach focuses on discussion to drive student engagement whilst fostering a deeper understanding of the presented topics compared to more traditional displays of crude facts. The course is split into three sections: local concurrency, networked concurrency, and concurrency in hardware. This allows for an easier student journey through the course, as they are introduced to all core concepts in the first section, then have them reinforced in greater detail in the subsequent sections. Finally, the experience gained in updating this course is presented so others attempting to do similar may learn from it.**

***Index Terms*—Concurrent, Parallel, CSP, SME, ZeroMQ, Teaching, Concepts**

I. Introduction

Scientific data processing is a considerable computing task that necessitates the use of High Performance Computing. Despite the presence of various libraries[1] to manage all aspects of parallel programming, knowledge of how a distributed system works is still essential to make full use of parallel hardware. This can present a problem as parallelisation is not a trivial topic, and scientists running experiments may not have an extensive background in computer science or programming.

At The University of Copenhagen, distributed computing courses are taught by the eScience group, part of the Niels Bohr Institute. These courses are mostly taught to physics students, who need this knowledge so that they can set up experiments to make use of parallel computing. A new *Concurrent and Distributed Systems* course has been introduced to replace an older course. The previous course had a theoretical focus, with rigid facts rather than engaging concepts. This would turn students off and give only a surface level of understanding.

This paper proposes replacing a fact-based approach to teaching with a concept-based approach. This will be broken down into 3 linked stages, local concurrent programming, distributed concurrent programming, and concurrent programming at a hardware level. In this paper these areas are demonstrated using PyCSP, PyZMQ and SME, but any similar library could be used in their place as it is the shared concepts beneath them that are important. These underlying concepts, such as deadlock, race conditions, and distributed states are the core of concurrent programming and so are the true learning goals.

Ultimately this was partially successful, with students responding well, though at great time investment on the part of the lecturers. Despite this, the techniques demonstrated their validity and could be applied to other similar courses going forward.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 765604.

[1]As illustrated by, but not limited to: CUDA [1], OpenMP [2], CSP [3], ZeroMQ [4], and MPI [5]

II. Objectives

This paper is an account of, and reflection on, teaching carried out within the *Concurrent and Distributed Systems* course. This is done with 3 goals in mind. In no particular order these are:

1) To record how a new approach to teaching concurrent systems design was put into practice.
2) To evaluate and reflect on how this new methodology worked.
3) To recommend for similar future courses what could be carried forward and what should be changed.

All of these stated objectives will be considered in the context of teaching parallel programming to non-computer scientists. Within this paper 'computer scientists' are those students whose primary area of study falls within computing. Any other students are 'non computer scientists' and are assumed to have no more than a passing familiarity with programming.

III. Background

The Niels Bohr Institute is a research institute specialising in astronomy, biophysics, condensed matter physics, geophysics, quantum physics, and particle physics. In addition to this, the eScience department conducts its own research into scientific methods using computers. As well as maintaining physical hardware and software to support the other departments, it is also responsible for teaching High Performance Computing. A new Masters level course in concurrent and distributed systems was designed to be run in the academic year 2018-2019 with the authors as the teaching team. This course was

titled *Concurrent and Distributed Systems* and was intended as a refreshed version of older courses in similar areas that were now deemed insufficient.

The *Concurrent and Distributed Systems* course used as a basis for this paper was taught at the University of Copenhagen, from the 19th of November 2018, until the 27th of January 2019. It was taught with 2 lecture slots a week, each 1.45 hours in duration. There was a weekly practical session also 1.45 hours in duration. 3 assignments were given out over the course, each lasting roughly 2 weeks and a final examination was given in the form of a take home exam with students having 11 days to complete it.

IV. Facts VS Concepts

The base assumption for the new course was that focusing on the *concepts* of distributed computing was preferable to focusing on *facts*. By *facts*, *mathematical proofs*, *information*, or *technical details* we refer to pieces of knowledge that are (probably[2]) true. It is often fundamental to the subject area, can be easily rote memorised [7], and can easily be expressed in a book, lecture or other one way communication. The syntax of a built in function, the clock speed of different machines, particular communication protocols, would all be examples of *data*, *information* or *technical details*. For the rest of this paper these shall all be referred to as *facts*.

Ideas or *concepts* refer to pieces of knowledge that are not necessarily verifiable. These are the grand approaches that can emerge from multiple facts and used to explain or guide systems. For example, consider system architectures, design approaches, or algorithm structures. All of these demand engagement from someone to understand and cannot be meaningfully rote memorised. These depend on many facts to be understood, and our understanding of them is constantly morphing and being updated. They can also be used to extrapolate new areas of knowledge and understanding [7]. For the rest of this paper these shall be referred to as *concepts*.

Concepts such as object-orientation are already widely taught and understood within computing, demonstrating their utility within computing education. This is especially important as the difference between sequential and parallel programming is not one of technical details, but one of thought. How you approach a parallel problem is fundamentally different to a sequential one, with a completely different structure and thought process behind it. Put another way, the problems of parallel are concepts, rather than facts, and so they require teaching focused on those concepts rather than on facts [8]. Therefore, even though *Concurrent and Distributed Systems* is aimed at potentially novice computer scientists, the teaching should be concept-based.

We could say that facts are more basic than concepts in a learning context, as facts are specific and cannot broadly be applied. However, facts are still required to act as a base for understanding concepts, and so both concepts and facts should be taught together. This means that when this paper refers to teaching concepts rather than facts, it is meant that concepts should be emphasised over facts, but not that facts should be ignored entirely as they form a necessary part of students learning.

In response to all this it was decided to start from scratch with a new series of lectures. The university format meant that we had to keep to 2 lectures a week with a practical. As concepts are much more difficult to explain than facts, as they tend to require a back and forth discussion in order to teach [8], the lecture format may have presented a problem. However, the class was expected to be small enough that the necessary discussions could take place. Note that despite all that has been said, facts are still extremely important. Concepts without facts become meaningless as they cannot be applied to the external world. It was hoped then, that the resulting slides could communicate the necessary concepts of concurrent and parallel systems, with enough supporting facts so as to be understandable.

V. Course Goal

Concurrent and Distributed Systems was aimed primarily at non computer scientists with limited programming experience. This is justified by the increasing requirement for parallel processing by scientists in all areas of physics studied at the Niels Bohr Institute [9]. In addition, students may be involved in designing scientific instruments or experiments which contain concurrent systems. Although there exist libraries that purport to take care of all parallelisation for the user, such as CUDA [1], MPI [10] or OpenMP [2], these still need a good base of knowledge from the user before they can be fully utilised. It would be possible to run a course that only related the facts of how these systems work, by highlighting specific commands and their expected outcome. However, this would leave students with a very narrow pool of knowledge, specific only to the exact software and problems described in the course. By adopting a concept-based approach, where instead the base concepts of distributed programming are explored and understood, non computer scientists can claim a theoretical understanding of parallel programming. This should suffice for them to effectively use any of these preexisting systems, and potentially even start designing their own custom implementations.

As most undergraduate physicists are not expected to be running big enough experiments to justify the use of high performance systems, *Concurrent and Distributed Systems* was set at a Masters level with classes expected to have between 6 and 12 students enrolled. The sought after learning objective would be some measurable understanding of asynchronous concurrent and distributed systems. That is, a system comprised of multiple processes, where the order of processing is not and cannot be determined at the start of processing. By the end of the course students should be able to design and implement concurrent and parallel systems in both hardware and software. These systems should be robust

[2]At the very least it is expected to be true, even if it is up for debate. Within the field of Epistemology it could be said that these statements are justified beliefs, that are true as far as can currently be determined. Consider this in the context of the works of Edmund Gettier and others. [6]

to common design problems such as deadlock, livelock and race conditions. Students should also be able to demonstrate their systems correctness using diagrams and descriptions.

VI. Selecting Course Content

To introduce and reinforce universal concepts of parallel computing, the decision was taken to break the subject down into three smaller sections. These could then slowly introduce concepts, and illustrate their universality within distributed programming. As these concepts would occur repeatedly through the three sections in increasing depth, it was hoped that they would be further reinforced. Starting with local parallelisation would be logical, as it meant that problems such as networking could be ignored. This allows for the introduction of the base concepts such as determinism, race conditions, deadlock, livelock, compartmentalisation, as well as identifying what sort of tasks are suitable for parallel or not.

All scientific programming within the Niels Bohr Institute is taught in Python[3], where possible. This meant that the underlying language was already set, and that some familiarity with the language could be assumed. To illustrate parallel processing concepts in the first course section, PyCSP[4] [13] was selected as all members of the teaching team were already familiar with it. It is worth noting that PyCSP has been taught in related courses previously, and has been found to be a very good introduction to concurrent and distributed concepts, even for novices [14]. Sticking with what the teaching team were familiar with was seen as important as it meant all members had already built up a body of knowledge designing systems using PyCSP. It was hoped that this would mean that the teaching team could adequately answer questions without having to rely on slides or textbooks to do the heavy lifting. This would be essential if we were to avoid dry lectures of reading technical information to students, but were instead to encourage conversation and interaction.

From a localised system the next logical step was a distributed one. These would still be using the same concepts from before, but now with added challenges such as the impossibility of global memory. This could be done using PyZMQ[5] [15] as again, it is Python based, simple to learn, and the teaching team were already familiar with it. Finally it was felt that students should have some introduction to physical devices that could be used to run a distributed system, such as in an *Internet of Things* device. This was as the problems that would be introduced in the first two sections are just as much problems in hardware as in software. To illustrate this, FPGAs[6] were used, with SME[7] [16] as the code base.

It was decided that it might help students to keep motivated and interested in the material if they could relate it to their own interests or research [7]. As at this point it was known that the FPGA boards were to be used, and that the students would build a system on the board, it followed that this system could be something scientific. A simple sound locator system was decided upon. This could be used in all course sections, so that all the assessments are tied together by a common thread. In the first section the students design a PyCSP system to process multiple microphones listening for sounds to determine the direction of a sounds source. In the second section they design a networked system, where they each link together their individual systems. In the third section they then program an individual microphone.

The hope is that these three sections will cover all essential areas of knowledge for the students, and assumes relatively little background knowledge. By starting on local systems and working up to more and more decoupled examples it is also intended that students are slowly introduced to the topic without them being hit with incomprehensible topics all at once. The students journey through the course should be simplified greatly as in the first section performance and efficiency are secondary concerns to robustness and ease of understanding. As the students continue through the course they are introduced more and more to requirements of performance and working within the already introduced concepts to get more processing done in less time.

VII. Software and Hardware

Software and hardware infrastructure was needed to run the course effectively. Students would need Python, PyCSP, ZeroMQ, and SME. These libraries and their dependencies could be time consuming to set up per individual, setting them up would not be particularly informative to the students. To get around this, JupyterLab Notebooks were used as a learning environment. JupyterLab Notebooks are documents accessible online, capable of displaying and running live code. They are centrally stored and so can have all dependencies pre-loaded onto them, meaning all students can easily start from the same point, with a complete system. For hardware, the PyNQ [17] board was selected as they contained a FPGA chip, had the necessary hardware to run Python scripts, and were reasonably priced.

VIII. Previous Teaching Material

For most of the course there already existed relevant lecture slides from previous similar courses. Naturally, these needed slight editing to fit with the new course, but in the case of section one, on PyCSP, a complete rework was required.

[3]This is due to the utility of Python for scientific analysis [11] and research, as well as its wide adoption throughout the scientific community. [12]

[4]PyCSP is a Python specific implementation of Communicating Sequential Processes (CSP) [3], a formal definition of how a system could be split up into several independent sub-sections and how those sub-sections would communicate. Other implementations exist in other languages, in varying states of completeness. The underlying principles between each are shared however.

[5]PyZMQ is a python specific implementation of ZeroMQ (also known as ØMQ) [4]. It is a library for easy asynchronous communication over a network.

[6]Field Programmable Gate Arrays (FPGA) are essentially programmable circuit boards, allowing for the implementation of many different hardware circuits using only one device, as opposed to expensive, custom made chips.

[7]Synchronous Message Exchange (SME) is a CSP derived language for programming FPGA boards. It compiles into VHDL and is designed to be more user friendly and quicker to program.

Plenty of teaching material was available [14], in the form of slides, books and workbooks. Each of these were considered in turn but rejected for a variety of reasons. These materials were often for a different length of course, meaning serious cutting or padding would be needed. As the different resources available were also from disparate sources they were all designed inconsistently, so even more editing would be needed to bring together any slides into a common visual language.

Aside from these small, practical considerations, it was also felt that the available material relied far too much on reading complex information off of slides as a method of teaching. Slides would be either extensive blocks of code, complex diagrams, or paragraphs of text. Often times, mathematical proofs of correctness were included and run through, proving the validity of a certain approach. This may be correct, but that level of detailed understanding is unnecessary for most students, especially non computer scientists. The material did not support interaction beyond asking simple memory recall questions rather than discussion and could be said to be entirely fact-based, as discuss in section IV.

It was felt that a better approach would be a more discussion based one [18], with a focus on the ideas and concepts behind CSP rather than on the technical details [19]. This should be especially possible given that the course was set at a Masters level and so should have students who can engage in a topic more in terms of ideas rather just simple facts. The expected small enrolment also meant that facilitating informal discussions rather than a strict lecture should be possible. Finally, a conceptual understanding would be preferable for non computer scientists as numerous libraries and systems exist to automate the generation of parallel code. It is the theoretical understanding behind these libraries that the non-computer scientists need.

IX. Goals for the New Material

To design the new material, objectives had to be set against which it could be designed, and success judged. These goals were devised with the aim of using an inductive approach to teaching [20]. These are presented below, with higher priority goals being at the top of the table. The material should:

	Goal
G1	Facilitate the teaching of concurrent and parallel concepts.
G2	Support the presented concepts with facts.
G3	Encourage student engagement in the class through exercises and discussion.
G4	Provoke questions and discussion from the students.
G5	Enable the teacher to explain in their own words, rather than relying on technical definitions.
G6	Be clear and easy to understand.
G7	Be reusable in subsequent courses, even by others not on the current teaching team.

Most of these goals should be self explanatory and so will not explained at length. G3 and G4 may require clarification though. Ultimately they both are the same idea, to focus more on conversation about a topic, rather than a direct lecture on it. It has been split into two goals to show that this is a two part process. In G4 we need to make sure that the teacher is accepting of this style in their teaching, whilst in G3 we should encourage the students to be interacting with the lesson. After all, if only one person is trying to start a conversation, it will not happen easily. Note the use of the words 'teacher' and 'lesson' rather than 'lecturer' and 'lecture'. This is done to suggest that the person standing at the front is not merely talking *at* the students, but *with* them [19], and does not denote any further difference.

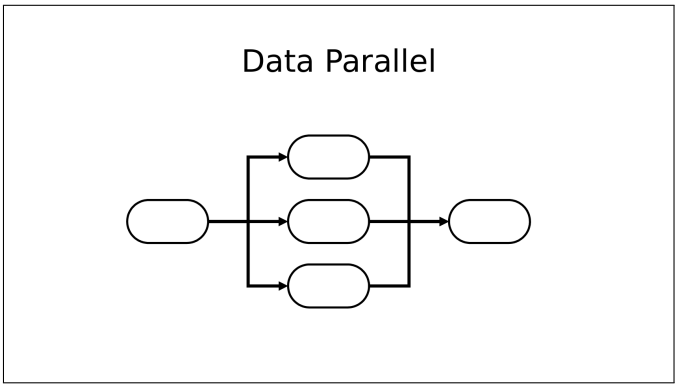

Fig. 1. Lesson 1, slide 23. This demonstrates the simple, clean design for the slides with the bare minimum of information. This diagram is intended as a conversation aid, rather than an explanation on its own.

X. Designing New Material

With these goals in mind, eight sub-topics were selected for section one of *Concurrent and Distributed Systems*[8]. These could then roughly align to the 4 lectures given in this section, with each lecture split by a small break, forming 8 half-lecture slots of roughly 45 minutes. In that time new concepts should be introduced, the facts to support them presented, and the resulting discussion engaged in. This is a lot to do, so presentations were kept to around 25 slides. Sentences on slides were kept short and well spaced. Diagrams were plain and presented without accompanying text on the slide. For examples, see figures 1 and 2.

Both of these slides are typical of the newly made slides for this course, as both of them provide one or two key facts, and very little else. This was done deliberately with the aim of fostering conversation. By having so little information on the slides the lesson could not turn into a session of just reading information from slides, as there simply is not enough information to fill the time by doing so. This approach would also be coupled with regular questions from the teacher so as to foster more dialogue than in a more traditional lecture.

[8]These were *parallel design problems*, *an introduction to PyCSP*, *deadlock and livelock*, *parallel system design principles*, *determinism and race conditions*, *compartmentalisation*, *additional CSP concepts*, and *network communication*. For a complete course description and to see the contents of each section, all course materials are available at [21]

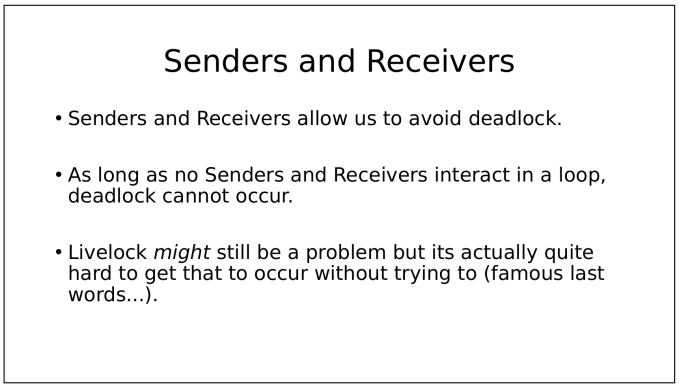

Fig. 2. Lesson 3, slide 15. Where text must be used it is kept to a minimum. As before, these sentences are intended as aides to what is currently being discussed rather than a lesson on their own. Even the written text is written conversationally.

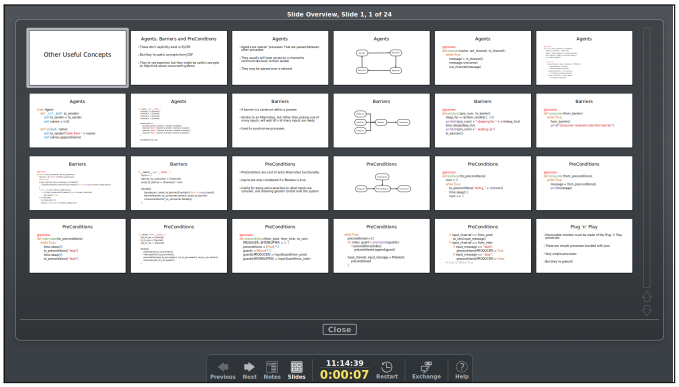

Fig. 3. Lesson 7, tiled view. This is demonstrated in LibreOffice Impress, but many other slideshow programs have similar features

The design of the slideshow itself also changed, with lessons being divided into sub-categories and where possible, not depending on a specific ordering to make sense. When displaying the slides in a lecture a tiled slide selector could be used to jump from slide to slide in a non-deterministic manner, and so follow the current direction of discussion. This is illustrated in figure 3. The simple design of the slides also helped here as it meant the correct slide is still readable on a laptop screen when in tiled view and so can be selected without difficulty.

To achieve the goal of more concepts and more discussion, regular exercises were introduced [19]. This is perhaps best exemplified by lecture 4, on designing a concurrent and parallel system that is only 5 slides long. The first slide is a title slide, while the second sets the exercise of designing a system. The third illustrates some discussion points that might occur as the students solve the problem and acts as an initial guide to students if they don't know how to start. The fourth introduces more exercise as it expands the initial problem. The final slide acts as a reinforcement to the core concept of this exercise, explicitly stating some supporting facts to ideas hopefully encountered. These slides would last no more than a few minutes, and exemplify every lectures role more as support

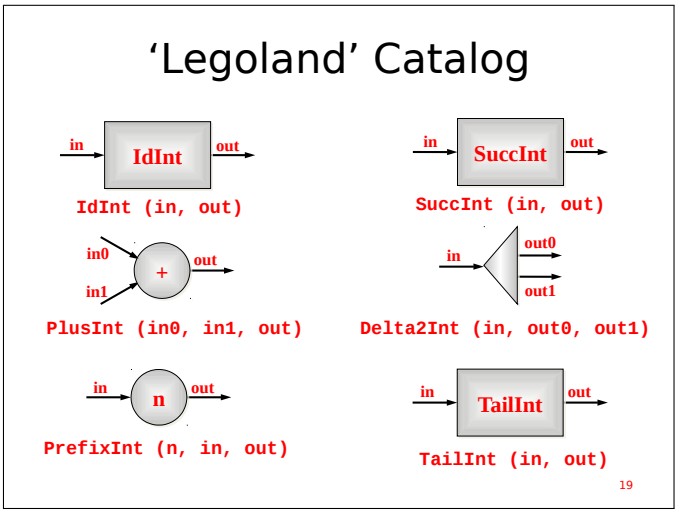

Fig. 4. An example slide from the previously used material for similar courses. This example has been picked as it is superficially similar to the newly designed slides, yet would foster a different style of teaching.

to a discussion rather than as the lesson itself.

The overall design of the slides compares favourably with the previous materials as the new ones are clearer, more condensed displays of relevant information. They act as effective notes for students as the minimalist design helps ensure that the information that is left makes effective, if brief notes as to the key supporting facts for the lessons concepts. As a comparison a sample slide from the older material is shown in figure 4. This slide appears superficially similar the a new ones, but it only displays a variety of available inbuilt CSP cookie-cutter processes[9]. These are rarely used in actual practice and so would be an example of teaching facts for facts sake as they do not lead to any wider conceptual understanding. As a result of this, the slide fosters little discussion beyond a description of the displayed information

One final brief note on the design of the new slides is that they each use the same visual language from the very beginning to the very end. Diagrams were expressed in the same way consistently, meaning that students only needed to decipher one way of reading diagrams. As there is no formal UML definition for network diagrams, previous materials visual languages could change dramatically between diagrams. Focusing on concepts over ideas may be hard enough for some students to follow, so these additional complications should be minimised by using the same style throughout. The designed slides, along with all other course material is available at a public Git repository [21].

XI. Lectures and Practicals

When teaching the lectures, to foster an environment of discussion, affairs were kept fairly informal. For instance, a

[9]These are provided processes that each perform some very basic functionality such as adding together two input numbers. They can be combined together to form more advanced functionality. In practice this is not done as the overhead from having so many processes makes for a bloated and slow system.

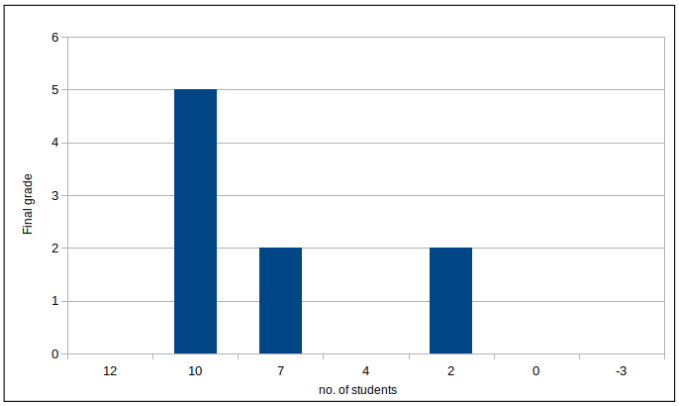

Fig. 5. Final results. Note that only nine students are shown as one is currently in the process of re-sitting the final assignment and so has not yet received a final grade. Initially they received the grade '0'.

conversational tone would be adopted throughout the lesson, with regular comments and observations. Questions from the students should be allowed as soon as they occurred to the students, rather than waiting for the end or some gap in explanations. It is also important that the teacher is open to admitting what they don't know as part of encouraging a conversation [19].

Practicals were relatively unplanned compared to lectures as again they were expected to be more conversational by their very nature. As students should already expect this from a practical rather than a lecture, less effort was needed to foster this specific atmosphere. There was no specific plan set for individual practicals and they were intended mostly as support sessions for help with assignments and troubleshooting any technical issues that emerged.

XII. Results

The course ran as expected with no major issues. Ten students enrolled on the course, though unexpectedly, eight were computer science students. This meant that a higher standard of background computing knowledge could be assumed. There was a reasonable spread of grades after the final exam, as shown in figure 5. These results are certainly higher than the expected bell curve, but with a small sample size and most students having a higher than expected familiarity with the topic, is not surprising or concerning.

The PyNQ boards caused minor software trouble throughout the course. These technical difficulties were never sufficient to derail the course and have now been ironed out so should not provide difficulty in future courses. However, the true problem with them was that they served as a distraction in students reports on their system. When asked to comment on the shortcomings of their designs, most would default to listing simple technical problems, rather than engaging critically with the conceptual problems. Most were able to engage with concepts once verbally prompted. The inability to do so in their report is potentially down to a failure to set expectations at the beginning of the assignment, rather than a fundamental problem of approach. Greater care should be taken in future to communicate what is expected from assignments, with explicit instructions in the assignment handouts.

Students responded positively in their end of course surveys [21]. Most seemed happy with the quality of teaching, but thought that the workload was too light. This may be due to the students being more familiar with the course contents than was anticipated. Students also felt that better use could have been made of the practical sessions. They were intended as informal help-sessions, and so were only lightly attended. This was expected, but a more defined structure with some set exercises may have helped give non-computer scientists more hands on time to get familiar with programming. It also would have filled out the workload. Students in related courses have also reported that having mandatory, short, defined workshop exercises every week helped them learn the topics. Workshops would also particularly suit the presented style of teaching as concepts are developed through practice and playing around with the presented facts. The workshop materials and any work produced in them would also inherently produce good revision and reference material for students if they needed to revise.

Preparing the teaching materials took roughly two weeks of work time spent on just 4 hours of lessons. The sheer length of time taken may be a function of the teachers relative inexperience, but it nevertheless illustrates the extensive planning and preparation required for such an involved teaching style. Naturally, this will reduce with practice but should be kept in mind by any new teaching teams attempting to replicate this approach.

The teaching team also feels that too much material was covered, particularly in the second section. Several types of communication protocols were discussed at length but never used or recommended. More time spent on FPGAs would have better served the students. This would help with the final point of criticism for the course. Throughout the course, but particularly in relation to hardware, the teaching team overestimated the students familiarity with programming concepts. During hardware setup they could be asked to define their board's IP address but no specific instructions on how to do this were provided at first as it was simply assumed that anyone could do this. Again, this can easily be addressed in future iterations of the course through increased support in practical sessions and more explicit instructions/guides in assignment hand outs.

XIII. Reflections by the Teaching Team

It is reasonable to conclude that the course went well. An expected number of students passed, the core topics have been taught, and students responded positively in their end of course surveys. To further evaluate the success of the course, the goals presented in section IX should be considered. These evaluations will rely heavily on the personal reflection by the primary author, who taught this section of the course. Further from student questionnaires and colleagues has also been added where possible and appropriate. Therefore, this section should not be considered scientific and unbiased. However, it was felt that the personal experience of an honest attempt at

trying to implement this style of teaching may be illuminating to others considering doing the same, especially as scientific guidance does not, and cannot, meaningfully exist for this [20].

A. G1: Facilitate the teaching of concurrent and parallel concepts

As the most important objective, G1 was kept in mind through the whole process, and was the reason that most previous teaching materials were replaced. Most sections worked extremely well, such as lesson 5 on Determinism and Race Conditions. Almost all of the slides in this lesson are prompts to student interaction, with students guessing at the outcome of some simple programs. These examples illustrate what determinism is so that hopefully by the time a quick slide explaining it through facts is shown, they already have formed the core concept, that can then be reinforced by the necessary facts.

However, several slides through the course turned out more full of explicit information than was first expected, and so became the primary teaching tool within their lessons. This is suspected to have led to less conceptual understanding from the students as the concepts communicated in those lessons did not appear in the final assessment submissions. Greater discipline is needed in limiting facts on slides, and a rule of thumb such as 'only 3 facts per slide' would help keep slides short and force discussion to the fore.

The failure of some slides illustrates the success of others. As mentioned in sections IV and X, concepts require facts to support them and so all facts on the slide should support a concept, rather than being a fact in and of itself. As most slides fostered conversation and with them conceptual understanding (as evidenced by the concepts being well understood in assignments) it is demonstrated that the presented teaching techniques are suitable for this goal.

B. G2: Support the Presented Concepts with Facts

Where the presented concepts fell down however was in the application to parallel programming as a whole. Once the course moved to topics other than CSP it seemed that several of these core concepts were forgotten, or it was not realised by the students that these were broadly applicable concepts rather than just relating to CSP. This is perhaps similar to the shape of the earth problem encountered by Vosniadou and Brewer [22]. The key similarity here is that the children and students appeared to have a complete understanding of the topic when first asked, but actually did not upon closer inspection. Vosniadou and Brewer suggest that the children did not have enough supporting facts for them to form an accurate conception of what the earth looks like when it is said to be round. It could be that a similar problem has occurred here, with insufficient examples provided illustrating the broader applications of the core concepts. Surprisingly, this issue was not limited to the physicists and even appeared with the computer scientists, who were expected to have an existing broad understanding of computing and so be able to apply concepts more accurately. Care should be taken in future to use a wider range of examples to demonstrate that these concepts are bigger than they might otherwise appear.

C. G3: Encourage student engagement in the class through exercises and discussion

Student engagement was fostered throughout the course through the use of exercises, questions, and prompts, rather than as a defined stage within the teaching cycle [7][10]. At certain points this was hard to keep up, and the teaching fell back on explaining things at the students. lesson 7 in particular suffers from this, as it is just an explanation of some common methodologies that exist in other CSP implementations. This was definitely the least successful lesson with very little seemingly being learned from it, judging on the taught material not being present in any of the students submissions. These topics are not notably more difficult than other lectures, nor were they explained worse than other topics. The lack of interaction was the only notable difference, leading to the conclusion that this is why it stuck less in the student's memory than other lessons.

D. G4: Provoke questions and discussion from the students

Students engaged willingly and consistently in discussion of the presented topics, with engagement from the whole cohort. A good way to foster conversation between the students was to set exercises and put them in groups of 2 or 3. This meant that to complete the exercise students were forced to exchange ideas, especially as the tasks where conceptual in nature, such as to design a system.

Better use could have been made of the practical sessions. As they were mainly conceived as trouble shooting sessions, attendance was low and those that did attend mostly did not interact with each other beyond to socialise. Something more structured could have given more of the students a reason to talk about the subject together, and allow for more time to reinforce how to apply the learnt concepts as discussed in section XIII-B.

It is worth noting that the exercise for the second section required the students to come to a mutual agreement on a communication protocol, so that each of their systems could communicate with each others. This section failed as no common agreement was made by the students despite repeated prompting by the teaching team. It may be that this failure was due to insufficient background being presented, and so students did not feel they had an understanding of where to begin. It may also have been that no student wanted to be the one to suggest a protocol that everyone else would have to follow. This vagueness of problem means it is hard to meaningfully reflect on the issue, and so perhaps this style of assignment is simply best avoided in future.

[10]It is worth noting that the models that present very separate, defined stages do not necessarily intend for them to be implemented as such, and often will explicitly state as such.

E. G5: Enable the teacher to explain in their own words, rather than relying on technical definitions

Similar to section XIII-D, this goal was mostly achieved. The slides were bare-bones so that they could not simply be read out, and most topics were explained ahead of displaying the relevant slide. This meant that a personal explanation, usually delivered in plain English acted as the primary introduction to a topic, with the defined points of a slide only introduced at the end to reinforce what was already said. The text that was on the slides also acted as memory prompts so that once an effectively ad-libbed explanation was complete, the slides could be checked to confirm that all essential points had been hit.

F. G6: Be clear and easy to understand

This informal approach meant that explanations or slides could be rather opaque. However, students seemed to follow along at the expected rate, and understood what was being said. This may be down to most student being computer scientists however, and so would potentially more familiar with this subject matter. I would expect that this problem could mostly be addressed by further practice at explaining the subject, and is affected mostly by practice at teaching.

G. G7: Be reusable in subsequent courses, even potentially by others not on the current teaching team

Re-usability was mostly forgotten through material creation, and in hindsight would not have been included as a goal. Much of the ambition of this style of teaching was in improvisation and discussion, with pre-made slides potentially discouraging that. The slides were only ever intended as a visual support to what was expected to be said, which may differ considerably from others lecturers. These slides may be a useful guide or starting point for another lecturers slides, but are not expected to be entirely usable by another lecturer without work.

This leads into the major downside of this approach. That being the length of time taken to produce these lectures. Whilst this process should speed up with experience, it will still need to be repeated for each course, making this a very time intensive form of teaching. The ideas presented within this paper about a concept focus are not new[11], yet it is perhaps this time commitment that limits their wider adoption.

XIV. Future Recommendations

It is recommended that *Concurrent and Distributed Systems* continues in future, and that the ideas put forward in its teaching style are iterated upon. The conceptual basis of the course worked well, and PyCSP, PyZMQ, and SME acted as good illustrators of these concepts. The use of established libraries meant that relatively little time could be spent on simple facts and allowed for discussions both broad and deep about the theoretical underpinnings of distributed systems. One of the primary goals was to make a course for physicists who needed to understand parallel programming, and yet only 20% of the eventual enrolment were physicists. The course description for students should better reflect who it is intended for. Additionally, it may be worth advertising the course directly to physics students, perhaps by making sure supervisors within the various departments at the Niels Bohr Institute are aware of its existence, and are mentioning it to those who may benefit from it.

Masters students can still need considerable prompting to engage critically with material rather than just rote learning. Being very clear about this from the beginning is essential. Even greater emphasis should also be taken on student engagement in lectures. This can be done with exercises and should lead to better learning outcomes. In addition it will foster more conversations by their very nature. In particular, small group exercises during class are extremely good at this, especially when the work is conceptual in nature.

In contrast to the success of the small group exercises, the only assignment that required agreement from the whole cohort did not demonstrate any. Each individual solved the problem in their own way. Although some guesses were made as to what caused this failure, no definitive problem could be found. This might demonstrate that long form group work may not be as effective as the short class exercises. It could also demonstrate that a group of 10 is too big to solve a small problem, and so should have been broken up.

Greater use should be made of the practical sessions by including short programming exercises. For example, after a lesson on deadlock, a short exercise to purposefully implement a deadlocking system would be good. These small exercises could be done in the practical sessions further away from assignment hand-ins to keep the students engaged through the quieter parts of the course. It would also allow them to build up their practical experience with facts they themselves have discovered, and to reinforce the concepts they have just been exposed to.

The PyNQ boards proved to be something of a distraction. They had constant minor technical difficulties, even with the JupyterLab Notebooks. With more experience these issues could be ironed out. It may also be that a more abstract series of assignments that did not have to run on a particular board would be better, as any technical problem proved too tempting students to focus on when reflecting on their own solution, rather than reflections on the concepts within their system. Eliminating the physical element may help address this.

All of this leads to following key recommendations for others seeking to introduce an inductive, concept-based approach to teaching parallel systems. In no particular order:

- The student journey of local parallelisation, distributed parallelisation and finally parallel in hardware works well. This is exemplified by PyCSP, to PyZMQ and then SME, and is even an effective introduction for non-computer scientists.
- Student engagement can be fostered through bare-bones teaching materials which force both the lecturer and the students to actively participate throughout the lecture.

[11] Consider that several references on this paper are over a decade old at this point

- Care must be taken when designing materials that they support active discussion. A non-linear slide show that can be adapted to the flow of conversation is a good example of this.
- Group work and practical assignments are essential for allowing concepts to develop and cement in students mind and should be utilised as much as possible.
- Student may be unused to this style of course and assessment, so the expectations on them should be repeated during course descriptions, introduction lectures and assignment handouts.

None of the techniques suggested in this paper (group activities, discussion, bare-bones slides) are unique to parallel computing and could in theory be applied to any subject matter. However, it does take considerable setup time. Despite this, it is hoped that with practice this preparation time will dramatically reduce, making it a more feasible teaching method.

XV. Conclusions

This paper has presented an account of how the *Concurrent and Distributed Systems* course at the Niels Bohr Institute eScience department has modernised the teaching of distributed systems to physicists. This was done because previous courses were insufficient. They were dry courses, full of facts, and with very little student interaction. By focusing instead on concepts supported by facts, discussion and student engagement it was hoped that learning outcomes could be better achieved, with non-computer scientists achieving a deeper level of understanding in the area of parallel systems. Students were introduced to local concurrent programming, followed by networked concurrent programming, and finally concurrent programming on hardware. This allowed for the core concepts to be introduced early, and then applied to different situations, demonstrating their universality.

This course was judged to be a success and so should be refined and repeated in future years. Lessons learned from it could also be applied to other similar courses. In particular, the bare-bones slides and the regular group activities were helpful in fostering an atmosphere of conversation, and elevated the education beyond a discussion of mere facts. However, there was a failure in setting a correct expectation amongst students at the beginning, and the time taken to prepare these lessons was extensive. Regardless, it is still recommended that the experience gained in teaching this course is carried forward, both in this course as it continues next and in other similar courses.

References

[1] "CUDA," https://developer.nvidia.com/cuda-zone, 2019.

[2] "OpenMP," https://www.openmp.org, 2019.

[3] C. A. R. Hoare, *Communicating Sequential Processes*. http://usingcsp.com/cspbook.pdf: Prentice Hall International, 2015.

[4] "ZeroMq," http://zeromq.org, 2019.

[5] "MPI for Python," https://mpi4py.readthedocs.io/en/stable/, 2019.

[6] E. L. Gettier, "Is Justified True Belief Knowledge?" *Analysis*, vol. 23, no. 6, pp. 121–123, 1963.

[7] N. Entwistle, *Teaching for Understanding at University*. Basingstoke: Palgrave Macmillan, 2009.

[8] J. Biggs and C. Tang, *Teaching for Quality Learning at University*. Maidenhead: Open University Press, 2007.

[9] R. Munk, "Teaching Parallel and Distributed Techniques at UCPH through JupyterLab," 7 2019.

[10] B. Barney, "Message Passing Interface (MPI)," https://computing.llnl.gov/tutorials/mpi/, 2019.

[11] J. M. Perkel, "Programming: Pick up Python," https://www.nature.com/news/programming-pick-up-python-1.16833, 2015.

[12] D. Robinson, "Why is Python Growing So Quickly?" https://stackoverflow.blog/2017/09/14/python-growing-quickly/, 2017.

[13] "PyCSP," https://pypi.org/project/pycsp/, 2016.

[14] B. Vinter and M. O. Larsen, "Teaching Concurrency: 10 years of Programming Projects at UCPH," in *Communicating Process Architecture 2017*, K. Chalmers and J. B. Pedersen, Eds. IOS Press, 2017, pp. 135–156.

[15] "PyZMQ," https://github.com/zeromq/pyzmq, 2019.

[16] B. Vinter and K. Skovhede, "Synchronous Message Exchange for Hardware Designs," in *Communicating Process Architectures 2014*, P. H. W. et al, Ed. Open Channel Publishing, 8 2014, pp. 169–179.

[17] "PYNQ: Python Productivity for ZYNQ," http://www.pynq.io, 2019.

[18] M. Prince, "Does Active Learning Work? A Review of the Research," *The Research Journal for Engineering Education*, vol. 93, no. 3, pp. 223–231, 2004.

[19] P. H. Scott, E. F. Mortimer, and O. G. Aguiar, "The Tension Between Authoritative and Dialogic Discourse: A Fundamental Characteristic of Meaning Making Interactions in High School Science Lessons," *The Research Journal for Engineering Education*, vol. 95, no. 2, pp. 123–138, 2006.

[20] M. J. Prince and R. M. Felder, "Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases," *The Research Journal for Engineering Education*, vol. 95, no. 2, pp. 123–138, 2006.

[21] "Concurrent and Distributed Systems course material," https://github.com/PatchOfScotland/ConcurrentAndDistributedCourseMaterial, 2019.

[22] S. Vosniadou and W. F. Brewer, "Mental Models of the Earth: A Study of Conceptual Change in Childhood," *Cognitive Psychology*, vol. 24, pp. 535–585, 1992.