EA: Research-infused teaching of parallel programming concepts for undergraduate Software Engineering students

Nasser Giacaman and Oliver Sinnen
Department of Electrical and Computer Engineering
The University of Auckland
Auckland, New Zealand
n.giacaman@auckland.ac.nz o.sinnen@auckland.ac.nz

***Abstract*—This paper presents experience using a research-infused teaching approach towards an undergraduate parallel programming course. The research-teaching nexus is applied at various levels, first by using research-led teaching of core parallel programming concepts, as well as teaching the latest developments from the affiliated research group. The bulk of the course, however, focuses more on the student-driven research-based and research-tutored teaching approaches, where students actively participate in groups on research projects; students are fully immersed in the learning activity of their respective project, while at the same time participating in discussions of wider parallel programming topics across other groups. This intimate affiliation between the undergraduate course and the research group results in a wide range of benefits for all those involved.**

I. Introduction

Much like Computer Science, the Software Engineering degree teaches students the core elements of hardware and software, while at the same time developing their problem solving skills. However, the Software Engineering degree places a much higher emphasis on the application of that knowledge and takes students through the complete software development process; this includes working in teams to gather requirements, designing, building, testing and deploying the final software product. The objective of Software Engineering is to prepare students for meeting industry needs, by taking responsibility of managing and delivering software projects.

A recent trend in industry has come in the form of ubiquitous multi-core everyday consumer devices, namely laptops, netbooks, tablets and smartphones. Much like traditional parallel computing devices, programmers need to explicitly incorporate multi-threading within their software applications to utilise the underlying hardware. Herein lies a major complication, namely the necessary skill-set required depends on the particular parallel hardware (e.g. shared memory multiprocessor versus distributed memory systems). While some undergraduate courses do teach essential parallel programming concepts, it is usually in the form of a module within a non-dedicated course (for example, an operating systems course). The concepts taught typically include students as recipients of well grounded foundational knowledge, rarely directly involving students in the latest parallel programming research.

With an ever-changing technical world, students are constantly expected to be ever so more flexible in their acquisition and analysis of the changing skill-sets; in this regard, developing a research-skilled undergraduate student is essential for their career, and not merely just to quench their desire for intellectual insight [7]. The level of research integration within the teaching can vary significantly [5]; in *research-led* teaching, students are mere recipients of research (e.g. the instructor's research is referred to within lectures or supplementary reading), while in *research-based* teaching the students take on the active role of advanced learning alongside the instructors by participating in inquiry-based learning. For completeness (but of less interest to us here), *research-oriented* teaching focuses on research ethos (as opposed focusing on the actual research content) [5]. Figure 1 shows this research-teaching nexus model, extended to include *research-tutored* teaching [6], where students work in groups writing and discussing papers.

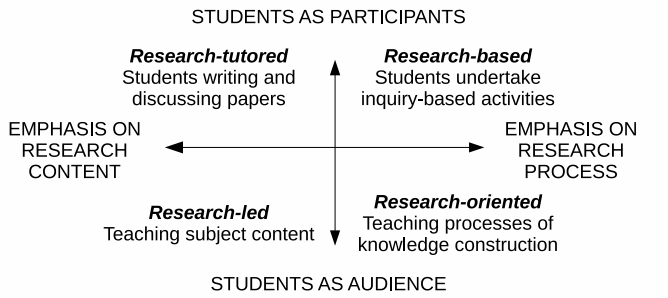

Figure 1. The research-teaching nexus and how it integrates with curriculum design based on content emphasis and student participation in the research (based on Healey [6]).

Contributions: An important way to effectively develop the link between research and teaching is to share discipline-specific case studies [7]; here, we share our experience with SoftEng 751 and discuss our experiences in linking research and teaching specifically for parallel programming tailored in a Software Engineering undergraduate degree. As instructors, we were pleased with the outcomes of this approach; based on student evaluations, we believe students were also satisfied.

We present our approach, which we term as *research-infused* teaching as it incorporates most of the research-teaching nexus categories of Figure 1. The course allows

students to develop core parallel programming skills, while also providing a genuine research experience within a research group. We present our experiences with the course structure and administration that both benefits the undergraduate students as well as postgraduate research students in the research group. Sample projects undertaken are also discussed, to help motivate instructors regarding defining suitably-scoped project topics.

The rest of this paper is organised as follows: section II presents some background on the context of the course, while section III discusses the overview of the course philosophy. Section IV more specifically addresses the integration of the course projects within a research group, while section V presents an evaluation from students and instructors. We conclude in section VI.

II. Background

This paper presents experience teaching a parallel computing course, SoftEng 751 High Performance Computing, in the Department of Electrical and Computer Engineering at the University of Auckland. As it is part of the Software Engineering degree (available for final year students), the emphasis in this course is on developing larger scale projects in teams. Since this course is also taken by a number of Masters-taught students, it also needs to incorporate a research component in the form of an independent and challenging hands-on project.

There is no earlier course in the Software Engineering undergraduate degree that introduces parallel programming concepts. However, since SoftEng 751 is dedicated to parallel programming, students are taught the essential core concepts while also allowing sufficient time to endeavour on an larger independent research team project. The selection of these core concepts supports those programming topics proposed by the NSF/IEEE-TCPP Curriculum Initiative on Parallel & Distributed Computing [2] as being most vital.

This has been put forward as part of the Fall 2012 Early Adopter program [9]. In line with the Curriculum Initiative's goal, SoftEng 751 aims to motivate students by embracing parallel and distributed computing as an integral component of their professional career; this is especially important in an Engineering discipline such as Software Engineering. The focus of the course is on *applying* parallel programming skills to an everyday software engineering domain that most directly affects people in society; in this regards, the focus is on applications for multi-core desktop and mobile devices, as opposed to distributed parallel programming.

The Parallel and Reconfigurable Computing (PARC) lab was founded in order to support parallel computing research at the University of Auckland. The lab primarily focuses on the shared memory aspects of parallel computing, with facilities such as 64-core, 16-core, 8-core systems and numerous multi-core Android mobile devices (smartphones and tablets). The lab focus is on the development of desktop and mobile applications, targeting the current and future wave of multi-core systems. SoftEng 751 largely includes research projects closely linked to the latest development from the PARC lab, as will be discussed in this paper.

Week	Lecture purpose *[and notes]*	
1	IT	Concurrency, threads, thread-safety, synchronisation
2	IT	Shared memory (task and data) parallelism, OpenMP
3	IT	PARC research tools, *[release project topics]*
4	IT	Parallel system architecture
5	IT	Dependence analysis, scheduling
6	IT,A	Discuss project topics, test, *[assign projects to groups]*
Study break	P	*[students commence projects]*
	P	
7	ST	Group presentations
8	ST	Group presentations
9	ST	Group presentations
10	ST	Group presentations
11	A, P	Test, *[project work]*
12	P, A	*[Project work, final project submission]*

Figure 2. SoftEng 751 course structure. The second column represents how the respective week was used, either instructor-led teaching (IT), assessment (A), "free-time" for project work (P) or student-led teaching (ST) in the form of group presentations.

III. The research-teaching nexus of SoftEng 751

An important goal of SoftEng 751 is to develop practical parallel programming experience while also infusing the latest research from the PARC lab. To achieve these goals, it is important to first provide students with the necessary background knowledge regarding the project topics. Once students have a firm grounding in these concepts, they commence their research project.

A. Course structure and content

SoftEng 751 is a semester-long course that includes 6 weeks of teaching, followed by a 2-week study break, then concluding with another 6 weeks of teaching (the standard at the University of Auckland). Students were taught the essentials of shared-memory parallel programming (as per the Early Adopter proposal [3]) within the first 5 weeks (see figure 2). The 6^{th} week of teaching was dedicated to testing the material covered in the first 5 weeks, as well as discussions on the proposed project topics (discussed more in section III-D). The remainder of the teaching weeks (i.e. after the study break), involved students presenting their selected topic to the rest of the class; the contents of these presentations were all examinable, with a final test in the following week concluding the presentations.

B. Parallel systems available

To assist students in their projects, a number of shared memory parallel systems were made accessible to them. In addition to students having their own multi-core laptops, the following systems were also available:

- 64-core server (4x 16-core AMD Opteron 6272 processors @ 2.1 GHz)
- 16-core workstation (4x quad-core Intel Xeon E7340 processors @ 2.4 GHz)

- 8-core workstation (2x quad-core Intel Xeon E5320 processors @ 1.86GHz)
- Various departmental lab workstations (mostly quad-core)
- Various Android tablets and smartphones (quad-core)

C. Assessment

Group work is a vital component of Software Engineering, so a large component of the SoftEng 751 grade reflected this. To also emphasis the importance of research, a large proportion of the final grade was directed towards the projects themselves. In fact, only 25% of the grade targeted individual understanding of the lecture-style material covered in the first few weeks:

- **Test 1 (25%):** A test in week 6 concluded the first section of the course, by assessing students on their understanding of the core parallel programming concepts taught in weeks 1-5. Having the test immediately before students select and commence their projects also encouraged them to review these vital concepts.
- **Group seminar (20%):** From weeks 7 to 10, groups presented their projects to the remainder of the class during standard lecture times. Each lecture slot allowed for two presentations, each 20 minutes (+5 minutes for questions). A first-in first-served doodle poll was used for students to self–schedule a time slot. All members were expected to contribute equally in the seminar, and were assessed individually. Of course, groups presenting earlier would not have progressed as much as those presenting later in the course; for this reason, they were not assessed on their progress but rather on identifying and conveying the important aspects relevant to their topic. Following their presentation, groups were required to upload their slides to the university's learning management system for the remainder of the students to review.
- **Test 2 (10%):** To conclude the group presentations, a test was held in week 11. The purpose of this test was to encourage student participation and interest across all the project topics, and essentially increasing awareness of research in general.
- **Project implementation (25%) and report (20%):** The remainder of the course was dedicated for students to focus on their project, which included their final implementation and group report. For implementation, subversion logs were assessed to gauge individual member contributions. Students were also required to submit peer evaluations discussing the contributions made by each member; in most cases, students within a team were awarded equal marks. Both the report and implementation were due in the final week.

D. Determining and assigning project topics

As would any research group, the PARC group has a number of ongoing project interests being undertaken. The most stable of those projects are available online[1] for students and the public alike to download and use freely. The PARC lab maintains a wish-list of "todo" items that have been identified as suitable nugget-sized projects that would be perfect for an undergraduate student to investigate or implement as part of the SoftEng 751 course.

In determining the suitability of these projects, an important factor to consider is the time-frame; SoftEng 751 constitutes one-quarter of a full-time student's workload, while students will have 8 weeks of development time dedicated for the project. Another vital factor to take into consideration relates to groupwork, particularly the group sizes and how well is a project equally divisible amongst group members (necessary for assessment purposes). Finally, a project is especially likely to be on the list when it is considered to be an "independent nugget" that is complimentary to the PARC lab's work, but allows students to start development early on without too much time need to delve into the lab's larger projects.

For a class enrolment of almost 60 students in the last semester and a size of 3 students per group, the project list was narrowed down to the 10 most suitable and interesting projects from the PARC lab's perspective. This meant each of the 10 topics was undertaken by two groups; this in particular worked well as different groups on the same topic still produced considerably different (but excellent) results. Abstract-style descriptions were released to students regarding the topics via email, and then the topics were discussed in a lecture allowing students to ask further questions.

Some project topics had higher preference than others. To make the allocation of project topics fair, a doodle poll was released for groups to select which of the 10 topics they wanted. The doodle poll was set up to allow only two groups per topic, and each group could only make one selection. This worked extremely well, by minimising administration involvement from the instructor side; at the same time, the fair first-in first-served nature of the process was appreciated by students.

Before releasing the doodle poll, it was ensured that all students were allocated to a group. This ensured that not only were students given the chance to discuss with their group their prioritised list of topics, but that no student was disadvantaged when the doodle poll was released. In fact, students were even informed a priori as to the exact time the doodle poll would be released.

E. Research-infused teaching

Section I discussed the benefits of integrating research within undergraduate studies; even if students do not wish to pursue an academic career, being exposed to research allows them to better adapt to the ever-changing technical world that they will face in industry. Since the focus of Software Engineering is meeting industry needs, and with parallel programming increasingly becoming important in the software industry, it only makes sense to infuse research into parallel programming courses. There needs to be a focus on doing or building something.

For SoftEng 751, various categories of the research-teaching nexus (Figure 1) were taken into account. It was believed that a research-led approach alone was insufficient in developing

[1] www.parallelit.org

the student's research skills. From the students point of view, this form of teaching equates to a passive learning approach; information is transmitted one way with students expected to absorb the information [1]. To improve student learning and motivation [8], [1], the focus within these lectures includes a component of students *doing* something; in most cases, this entails a programming exercise.

Although the in-class exercises helped develop student enthusiasm, a deeper student participation was vital in order to better engage them in the research; this was the purpose of the group projects. The project allows students to fully engage in the course material, providing for a deep approach to learning [1]. The project itself is an example of research-based teaching, where students fully engage in the learning activity. At the same time, the group presentations (and resulting discussions) allow for research-tutored teaching, as does the report assessment; here, students lead the discussion on existing solutions and potential risks.

Based on the model of Figure 1, the one thing really missing in SoftEng 751 is some explicit emphasis on the research methodology (i.e. research-oriented teaching). However, we believe this is of less concern here since:

1) this contains no direct relevance to parallel programming content and distracts from the focus of the SoftEng 751 course,
2) students attain this knowledge anyway in other Engineering professional development courses, and
3) as shown in the model, this involves little student involvement as a lot of it places students as audience.

With that said, students were still indirectly exposed to the ethos of the research group, in particular to the group's software development etiquette (more in section IV-A).

An interesting point to note regarding the effectiveness of the group presentations and class discussions (i.e. research-tutored teaching), is that students need to know that the purpose of the discussions is to *collaboratively discuss* the material (rather than test their understanding), otherwise they will take on a surface approach to learning [7].

Since the selected project topics were directly extracted from the PARC research lab, it meant that students were not only being taught the latest research, but they were also directly engaged and contributing to its continued development. To support this, many of the groups were put in direct contact with postgraduate students (i.e. Masters and PhD) that acted as representatives to the PARC group. This worked extremely well, allowing both the undergraduate students to get more guidance while at the same time providing the postgraduate students with supervision experience. In some cases, the postgraduate students also directly benefit by having outcomes from the undergraduate student projects integrated with their own research projects.

IV. Integration with research group

There are a few considerations in engaging the SoftEng 751 students in the research group's activities. The aim was to welcome students into the group, to make them all feel like they are making a real contribution to the research group rather than just be working on a project towards their course.

A. Welcoming students with PARC protocols

The first way that students were made to feel a true part of PARC was to require they use the same software development tools as the research group. A major component of this was providing students with access to some version control (in this case subversion). Groups would create a new project directory for their group; the idea here was that students receive the message that they are part of the community of the research group, and therefore follow practices of the group. Using this approach, the instructors were able to view the development history for each group. This was powerful not only for assessment of the group as a whole, but also in regards to individual student contributions. Direct write access was not granted to the main PARC repository, however this approach built confidence for subsequent future projects where students continued on their research project and required integration into the main PARC repository.

Now that a larger number of projects were being undertaken by a now larger PARC community, it was vital that certain protocols were being followed. This was especially important regarding code etiquette when using the group's subversion; the students were provided with documentation regarding good hygiene in the directory structure for their project. This included information such as separating their source code from tests and benchmarks, what files to exclude from the subversion server, and so on. Also part of this protocol was that projects were all expected to work on Linux, since all PARC systems run on Linux. While students were free to develop on other operating systems, it was their responsibility to ensure all committed code worked on the PARC systems. This not only included script files, but also taking minor differences such as file separators and new lines into consideration.

Students were especially made to feel part of the research group by being put in direct contact of a more senior and existing research student from the PARC group. This allowed the SoftEng 751 students to gain more support, especially in a technical manner when they come across subtle issues. The Masters or PhD student would also typically help clarify the project, or provide background information where needed. This not only provided supervision experience for the postgraduate student, but it also directly helped their own research project in the case where SoftEng 751 students were investigating nugget projects for the Masters/PhD student.

Finally, possibly one of the most important aspects of the projects, was to ensure that all groups felt a strong sense of independence and ownership towards their project. Students were constantly reminded they have high flexibility in the direction of the project, and are able to pursue whatever direction interested them. This worked very well, with many groups developing projects that exceeded the instructor expectations. The additional advantage of regularly promoting ownership and independence meant that groups with the same topic developed different solutions and felt less pressured as they saw they ultimately had a different project; this also helped emphasise the research nature of the project, as there was no predetermined or known result.

B. Current PARC projects

To put some of the example projects that students undertake in SoftEng 751 into context, this section will provide some background for the relevant PARC projects. Students are first directed to the research website where all relevant documentations and downloads are made available. The bigger of the projects include Parallel Task [4] and Pyjama [10]. Both of these tools were developed to assist object-oriented programmers in developing parallel applications, in way of extended compilers and supporting runtime libraries. In the case of Parallel Task, Java is extended to introduce task parallelism and task dependences by introducing a few keywords. In the case of Pyjama, the OpenMP philosophy is introduced into an object-oriented paradigm to allow incremental parallelism on existing Java applications.

In both cases, Parallel Task and Pyjama shine by providing essential support necessary for graphical user interface (GUI) applications. It is for this reason that Java is used as the target language, since most graphical toolkits are object-oriented. Java also has the advantage that it is cross platform, as well as being the language used in developing Android applications. In this regards, due to the interactive nature of these target applications, responsiveness and user-perceived performance is of high importance. The PARC research distinguishes this as *concurrency* (for user-perceived performance), which differs from *parallelism* (used to explicitly improve wall-clock performance by utilising the multiple processing cores).

C. Sample SoftEng 751 projects

This section discusses some of the projects that were made available to students from the 2013 SoftEng 751 undertaking. Where noted, some projects had the option of being undertaken on the Android platform; in this case, it was required that the group members were already familiar with Android development, as they were mainly assessed on the parallelisation aspects rather than the actual app development (also, because Android development was not taught in SoftEng 751).

Since the course is offered once a year, it allows adequate time for a suitable collection of topics to be generated during the year. These topics are maintained as a shared document by the instructors, and are reviewed at the start of the course to determine the top-ten. Some of the project ideas are proposed by the instructors and graduate students, while others are recycled from previous years. The reuse of some project topics is possible, due to their research nature.

As shown below, most of these projects focus on shared memory parallel systems, particularly for desktop or mobile systems. It is worthwhile noting that, at face value, some of the topics do not appear to be of a research nature; however, it is the integration of those topics with the research tools that form the research component.

1) *Thumbnails of images in a folder*: (also available for Android) Using Parallel Task, this involved writing a small GUI application in which the user could open a folder of images with thumbnails being displayed for each image. The students could use existing functions/libraries to scale the images to a certain size (which should be adjustable). The main goal for this project was that the resizing of the images be done in parallel and that the GUI remains fully responsive; for example, the user could scroll up or down while the thumbnails were being rendered. One of the groups developed a desktop version, comparing the performance across a number of Java parallelisation strategies (using Parallel Task, Java threads, SwingWorker), while also investigating different ways to schedule the workload, and using different image input sizes. The second group with this topic investigated on Android, comparing Parallel Task to Android's AsyncTask and handlers/loopers.
2) *Parallel quicksort*: This involved developing parallel implementations of the classical quicksort algorithm, to sort a large array of numbers. Although theoretically a relatively simple recursive algorithm, the students had to implement three versions using object-oriented language support (using Parallel Task, Pyjama and standard Java threads and concurrency classes). The research component of this typically-parallelised algorithm is the use of the research tools in a way that was never done before; here, using the object-oriented approach of Parallel Task and Pyjama.
3) *Parallelisation of simple computational kernels*: Using Pyjama, this involved implementing basic algorithms (usually in the form of some nested loops) in parallel using Java and Pyjama. The implementations of the algorithms in C were provided to the students. Examples of the computations that were performed by these algorithms include FFT, molecular dynamics, graph processing and linear algebra. The groups compared Pyjama to parallelisation using standard Java concurrency libraries. Again, the research aspect of this project involved merging OpenMP concepts (developed for imperative languages) using an object-oriented approach.
4) *Search for a string in text files of a folder*: (also available for Android) Using Parallel Task, this involved writing a small GUI application in which the user would specify a search string (or even a regular expression), which is then searched in the text files of a folder and its subfolders. The goal was to do the search in parallel without blocking the user interface. Students were then expected to taking user interactivity into consideration also, where encountered strings were also displayed as file and line number pairs while the search was still in progress.
5) *Reductions in Pyjama*: The concept of a reduction is an important aspect of parallel programming since it allows for an efficient solution to sharing variables. The OpenMP API specifies a number of reductions that may be applied on a limited set of data types. In the case of an object-oriented language, such as Java, there are a larger wealth of reductions that one may wish to perform on a larger amount of data types (for example merging collections). Using Pyjama as the case study, this project involved the development of a number of reductions in an object-oriented language.
6) *Task-aware libraries for Parallel Task*: In a tasking model, as opposed to a threading model, programmers

need to think in terms of tasks rather than threads. Java provides various thread-safe classes in its concurrency package; but in a tasking model, such as Parallel Task, using a "thread-safe" class in this environment does not necessarily equate to a correct solution. The aim of this project was to get students thinking about possible implications this may have, getting students to implement various Parallel Task task-safe classes; these can be thought of as counterparts to the thread-safe classes provided in Java.

7) *PDF searching*: (also available for Android) Using Parallel Task, this project involved implementing an application that searches a number of PDF files (stored locally on a tablet or laptop/desktop) for a given query. The project involved investigating various granularity and parameters to the parallelisation process (for example, searching per page, per file, number of threads, etc). The project also aimed to demonstrate aspects of Parallel Task, such as intermittent updates as results are found, a responsive GUI, as well as having good (parallel) performance.
8) *Understanding and coping with the Java memory model for multi-threaded programs*: This project involved understanding the Java memory model and how it affects multi-threaded programs. This project served more as an educational purpose, in hope that the outcomes could be useful for future teaching purposes. The students developed a number of code snippets that demonstrate how typical parallelisation problems can occur (for example, forcing a race condition). This topic also involved writing how such problems can be avoided, outlining what options are available and discussing what their respective pros/cons are (for example, simplicity, performance cost, etc).
9) *Parallel use of collections*: When more than one thread accesses a collection in parallel, synchronisation mechanisms are necessary to guarantee correct behaviour. The Java SDK provides special thread-safe collections in `java.util.concurrent`. This project involved analysing their advantage over standard Java collections which are used with locks and other forms of synchronisation. The students implemented test programs to read/write in parallel to/from a collection, comparing the performance of the different approaches. This included the consideration of different locking mechanisms, such as `synchronized`, atomic variables, locks (fair/unfair) and different types of collections (LinkedLists, DeQueues, Sets, etc).
10) *Fast web access through concurrent connections*: (also available for Android) Due to the latency of network connections, it is sometimes meaningful to open several connections at the same time, rather than accessing one after the other. Parallel Task can be a perfect fit for this asynchronous communication approach; however, the question arises how many connections should be opened at the same time. In this project, students implemented a simple program that needs to access a large number of web-pages and used Parallel Task to download these pages as quickly as possible.

V. Evaluation

In evaluating the experience teaching parallel programming in a teaching-infused approach, we discuss in terms of student evaluations and reflections from the instructors perspective.

A. Student evaluations

Based on the end of course summative evaluation, students consistently showed satisfaction with the course and teaching approach. The evaluation consisted of a set of questions using the Likert scale (Strong Agree, Agree, Neutral, Disagree, Strongly Disagree), followed by an open comments section. Taking into consideration the non-traditional teaching approach, it was especially interesting to see if students felt comfortable with this style; fortunately, 95% of the students either agreed or strongly agreed that "The objectives of the lectures were clearly explained" and "The lecturer stimulated my engagement in the learning process".

In regards to the class discussions, 92% of the students either agreed or strongly agreed that "The class discussions were effective in helping me learn". In the open comments section, when asked "What was most helpful for your learning?", one student commended the presentations:

> *The presentations were good practice and watching them was informative*

while another commended the resulting group discussions:

> *Keep up the interaction with all of the groups*

Other students showed their gratitude to the project itself:

> *The project that was part of the course was very helpful*

while another student comments:

> *This course was full of project work. It helped me to learn and explore the concepts in Java. It also helped me to develop my presentation skills.*

It was encouraging to see that some students also acknowledged the research aspect of the course, and in fact desired more emphasis on this when asked "What improvement would you like to see?":

> *Individual meeting time can be extended so that more research oriented discussion can be done. I personally feel this course is very good to perform research hence more time should be devoted by the lecturer during individual meeting.*

B. Outcomes

There were many positive outcomes in the research-infused approach of SoftEng 751. From the students point of view, they were fully immersed within an existing research community. While undertaking independent projects, they got support from senior postgraduate students. At the same time, the undergraduate students were contributing back to the research group. To make this effective, students were expected to follow the research group's etiquette especially in regards to version control; even the final submission involved submission on

the subversion repository. This not only contributed to the research immersion, but it also allowed the instructors to better administer progress and view project history logs.

The community essence of the research group was especially effective in the Masters-taught arena; most Masters-taught students decide to undertake a project course (as an optional component of the Masters-taught programme), and many of those completing SoftEng 751 decide to complete such a project with PARC the following semester. They contribute even more to the research group and have an even closer involvement. Such students are now regarded as experienced members of the research group, purely based on their experience in SoftEng 751; they provide support and mentoring for the newer students. This overlap of experienced and new Masters-taught project students provides a constant stream of mentoring amongst the group. In effect, this also increased the awareness of the research group and its projects due to recommendations from previous students.

It was also the research postgraduate students (Masters and PhD) that also benefited from the SoftEng 751 projects. Involving the undergraduate students directly with the postgraduate team meant faster progress on the respective projects. This was attributed to the increased user base of the latest research tools, meaning more bugs were identified and resolved, while also providing more valuable feedback. This is of course a double-edged sword using the latest research tools; in some cases, certain bugs delayed progress on the SoftEng 751 projects. However, the undergraduate students were aware of the volatile nature of research and exercised patience while the identified issues were being solved. In some cases, potential misunderstandings were identified (e.g. how to use a particular tool), so this resulted in necessary fine-tuning of the underlying concepts.

To complete the picture of mutual benefit towards the research, and not just teaching, a number of specific and positive outcomes have resulted for the research group. For example, Pyjama is based on concepts from OpenMP, which specifically targets C (and by extension, C++) and Fortran (both from the imperative programming paradigm). Having students apply OpenMP concepts in an object-oriented paradigm (Java) allowed the identification and consequent adaption of the OpenMP semantics in an object-oriented language; for example, it was decided that the OpenMP `private` data clause was a source of confusion for Java developers, and it in fact diverged from good programming practices (e.g. not initialising variables at declaration and reducing variable scope).

Other contributions to the research ideas included the conception of parallel programming patterns using Parallel Task; in one project, students took advantage of fundamental inheritance and encapsulation features of object-oriented languages, allowing the programmer to elegantly alternate between parallel and sequential functionality. Another benefit to the research worth noting was the performance observations of the research tools under various applications. Finally, there were pedagogical contributions in the form of interactive webpages that helped explain typical race conditions and other parallel programming pitfalls.

VI. Conclusions

Affiliating an undergraduate course with a research project has resulted in many positive outcomes, in this case specifically for a parallel programming Software Engineering course. By infusing research into the course, undergraduate students are immersed in a larger research community and develop a better understanding of what research entails. More importantly, students are actively contributing back to the research group rather than only being recipients of the latest research. The interaction between undergraduate and postgraduate students provides additional supervision support, while at the same time assisting postgraduate students in accomplish specific nuggets towards their own research. The evaluations from students highlight an appreciation towards the research infusion, with many students electing to continue their studies with the affiliated research group.

Acknowledgement

We would like to thank the NSF/TCPP CDER Center Early Adopter Awards, for their support and feedback regarding the SoftEng 751 curriculum. Their financial contribution has also funded the purchase of a multi-core Android smartphone and tablet for students to use in the course.

References

[1] John Biggs and Catherine Tang. *Teaching for quality learning at university*. Open university press, 3rd edition, 2007.

[2] Georgia State University. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing. http://www.cs.gsu.edu/~tcpp/curriculum, 2010.

[3] Georgia State University. List of Fall 2012 Early Adopters. http://www.cs.gsu.edu/~tcpp/curriculum/?q=list-of-early-adopters-fall-2012.html, 2012.

[4] Nasser Giacaman and Oliver Sinnen. Parallel Task for parallelizing object-oriented desktop applications. *International Journal of Parallel Programming*, 41(5):621–681, 2013.

[5] Ron Griffiths. Knowledge production and the research–teaching nexus: The case of the built environment disciplines. *Studies in Higher education*, 29(6):709–726, 2004.

[6] Mick Healey. Linking research and teaching exploring disciplinary spaces and the role of inquiry-based learning. *Reshaping the university: new relationships between research, scholarship and teaching*, pages 67–78, 2005.

[7] Alan Jenkins, Mick Healey, and Roger Zetter. *Linking teaching and research in disciplines and departments*. Higher Education Academy York, 2007.

[8] Tony Jenkins. Teaching programming – a journey from teacher to motivator. In *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*, 2001.

[9] NSF/IEEE-TCPP Curriculum Initiative. List of fall 2012 early adopters. http://www.cs.gsu.edu/~tcpp/curriculum/?q=list-of-early-adopters-fall-2012.html.

[10] Vikas, Nasser Giacaman, and Oliver Sinnen. Multiprocessing with GUI-awareness using OpenMP-like directives in Java. *Parallel Computing*, (accepted for publication), 2013.