AAAS & NSF Envisioning the Future of Undergraduate STEM Education: Research and Practice Symposium, Washington, DC, April 27-29, 2016.

Experiences and Reflections of using Parallel Design Patterns Tools in an Undergraduate Program

Clayton Ferner
University of North Carolina Wilmington
601 S. College Rd.
Wilmington, NC 28403, USA
cferner@uncw.edu

Barry Wilkinson
University of North Carolina Charlotte
9201 University City Blvd.
Charlotte, NC 28223 USA
abw@uncc.edu

Barbara Heath
East Main Evaluation & Consulting, LLC
P.O. Box 12343
Wilmington, NC 28405 USA
bheath@emeconline.com

***Abstract* — In this paper, we discuss the results and lessons learned from a 3-year study of using two tools in the classroom to teach parallel computing. One tool was developed at the University of North Carolina Wilmington (UNCW), and one tool was developed at the University of North Carolina Charlotte (UNCC). The tools are designed to raise the level of abstraction above the low level tools such as MPI and OpenMP. We have used these new tools in the classroom to teach parallel computing in three semesters over three years, developed teach materials based upon these tools, gather feedback from the students to assess the effectiveness of our tools and teaching materials, and discuss the lessons learned and future work.**

Keywords- pattern programming; compiler directives; parallel computing; distributed computing.

I. Introduction

For many years, parallel computing was considered an advanced topic in the computer science curriculum, and only a small percentage of software developers were creating parallel programs. With the movement into multi-core computers, all general purpose computers manufactured nowadays are multi-core. Furthermore, parallel systems are easily created by building a cluster of computers. Parallel software development is now an important topic and skill that most software developers need to embrace. Indeed, ACM and IEEE have included parallel and distributed computing as a Knowledge Area in the Computer Science Curricula 2013 [ACM2013].

The need is growing to train software engineers at a rate that can support the software industry. Only a small percentage of software developers are skilled at developing correct parallel software. Although many computer science programs teach parallel computer as an elective, most do not consider it a required course. In contrast to that, there is a movement to introduce parallel computing into CS 1 and CS 2.

In our experience teaching parallel and distributed computing, the concepts are still at an advanced level making it difficult to teach parallel computing to all computer science students, especially in CS 1 and CS 2. Part of the reason for this is that the tools used for developing parallel software, such as OpenMP and MPI, are low-level tools, analogous to assembly language programming. The programmer still must deal with issues as race conditions and deadlocks.

At the University of North Carolina Wilmington (UNCW) and University of North Carolina Charlotte (UNCC), we have been developing tools that create a high-level of abstraction for creating parallel and distributed software. We have been using these tools in the classroom to teach parallel computing. In addition, we have also been teaching parallel and distributed computing using lower level tools, such as OpenMP, MPI, and CUDA, with which we and the students can compare and contrast the various options. In addition to the tools, we have been developing educational material based upon the various methods we use in the classroom.

The course has been taught jointly between the UNCW and UNCC over three separate semesters using the North Carolina Research and Education televideo network (NCREN), which connects the universities throughout North Carolina. In Fall 2012, we taught this course only between UNCW and UNCC. In Fall 2013, we taught the course to UNCW, UNCC, North Carolina A&T, East Carolina University, University of North Carolina Greensboro, and Winston-Salem State University. In Fall 2014, we included the campuses of UNCW, UNCC, Appalachian State University, East Carolina University, University of North Carolina Greensboro, and Western Carolina University. In Fall 2012 we had 58 students (22 undergraduates and 36 graduates) across the two campuses. In Fall 2013 we had 61 students (26 undergraduates and 35 graduates) across five campuses. In Fall 2014 we had 82 students (52 undergraduates and 30 graduates) across six campuses.

During these offerings, teaching effectiveness data was collected by an external evaluator, co-author Heath, who collected and analyzed the data completely independent of the two instructors Ferner and Wilkinson. The results were not released to the instructors until after the semester was over and grades were completed. Proper Institutional Review Board (IRB) protocols were followed throughout including the use of consent forms and complete confidentiality. Student participation was completely voluntary.

In this paper, we discuss the results and lessons from the three-year study of using our tools in the classroom to teach parallel computing. We briefly describe patterns and the two tools that have been develop at UNCW and UNCC to raise the level of abstraction for creating parallel programs. The rest of this paper is organized as follows. Existing work is briefly reviewed in Section II. In Section III, we describe the two new approaches. In Section IV, we briefly discuss patterns and our tools. In Section V, we describe our teaching materials. In Section VI, we describe the survey instruments. In Section VII, we present and discuss the results. Finally, Sections VI0 and IXX conclude and discuss future work.

II. Background

Parallel programming is traditionally taught beginning with MPI for message passing and OpenMP for shared-memory programming interspersed with parallel algorithms and most course textbooks such as Pacheco [11] follow this approach. The principal object is to have student learn how to program with MPI and OpenMP.

Some courses, especially in engineering-oriented programs, choose to use Pthreads, an even lower level API. We followed a similar conventional approach up to 2012. Our early textbook, Wilkinson and Allen [15], tried to de-focus on the actual tools and focus on parallel algorithms; however, one has to learn how to write parallel programs even with an algorithms-oriented course and we succumbed to teaching MPI and OpenMP first. It is well-established that writing parallel program is difficult. Apart from the usual factors found in sequential programming one has to deal with multiple programs executing concurrently. Figuring out all possible scenarios is a real challenge. One has to consider possible pitfalls such as race conditions and potential deadlocks. In 2012, we obtained NSF funding to continue our research work on using parallel design patterns, specifically to use design patterns within an undergraduate parallel programming course. We had developed several tools that are able to create executable parallel code based upon parallel design patterns without requiring the programmer to write MPI or OpenMP directly. The purpose of the project was to bring design pattern tools into the undergraduate parallel programming classes. Through several incarnations between 2012 and 2015, we tried different ways to do this starting with a top-down approach introducing design pattern tools first, and finally choosing a bottom up approach where students started with OpenMP, MPI and then our higher level tools. In this paper we report on the three year study.

III. Existing Work

The idea of design patterns in software engineering has been around for many years [7] and applied to undergraduate teaching [1]. The pattern approach to parallel programming has been explored in the influential textbook by Mattson et al. [9] written for software developers, and also in several research projects including at University of Illinois at Urbana-Champaign, University of California, Berkeley [8], and University of Torino/Università di Pisa Italy [6]. Industrial efforts in this direction include Intel [10] and Microsoft [14]. These projects do not focus on teaching parallel programming at the undergraduate level; rather they promote higher-level tools for programming style, scalability, and productivity. We extend such goals to undergraduate parallel programming teaching.

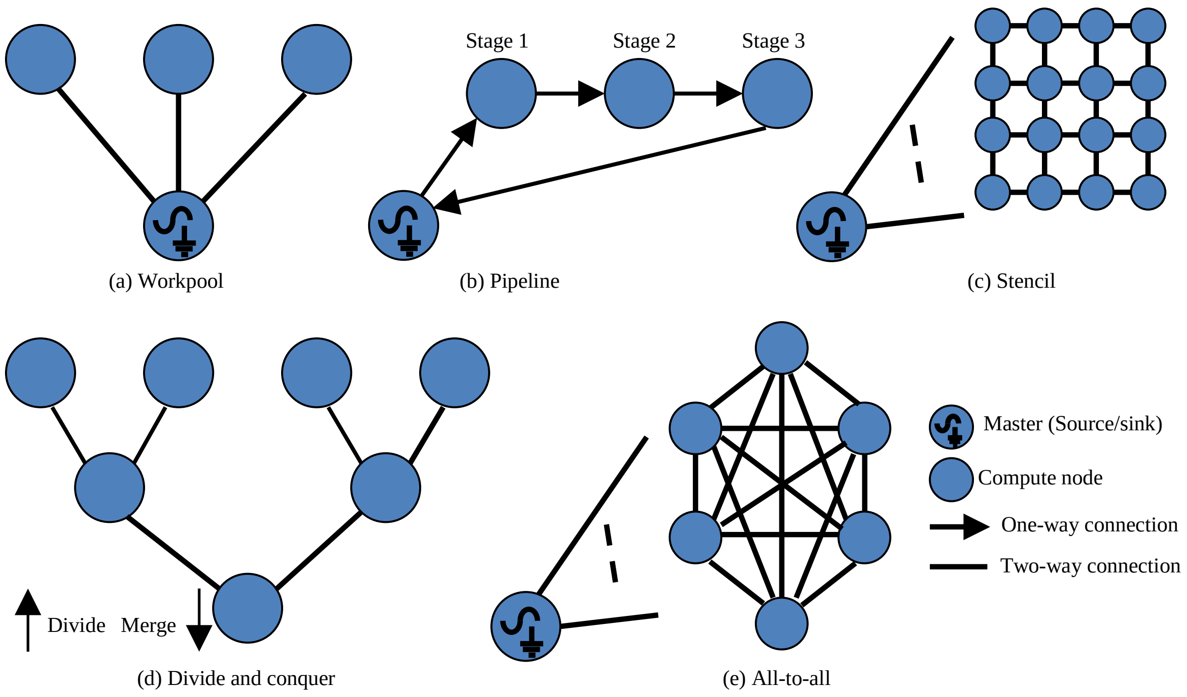

Figure 1: Parallel Patterns

Renault and Parrot [12] have created a pre-processors that can automatically generated MPI derived datatypes from the C data types. The goal is to assist the programmer with creating the complex MPI datatypes needed for the transmission of user-defined datatypes. Their work, however, does not generate the code to parallelize an algorithm. The closest work to our work on compiler directives is the llCoMP compiler for the llc language [13]. The llc language allows the programmer to specify parallel constructs for both MPI and OpenMP using llc and OpenMP pragma statements. In their work, Reyes, Dorta, Almeida, and Sande discuss the use of “static” and “dynamic patterns”. However, it appears that these patterns are compiler optimizations designed to improve the efficiency of the communication rather than user defined patterns related to the algorithm structure. Our compiler also has directives to specify parallel constructs to use both MPI and OpenMP. We plan to introduce new directives specifically to describe an algorithm through a pattern.

IV. Patterns

There are low-level and high-level patterns. Examples of low-level patterns are the fork-join, client-server, and scatter-gather. The patterns we mostly focus upon are the high-level patterns, such as workpool, pipeline, stencil, divide-and-conquer, and all-to-all, as depicted in Figure 1. The workpool pattern is similar to scatter-gather, except that it uses a workpool to assign work to workers. This is probably the most universal pattern as it can be applied to many algorithms.

The stencil pattern is used for Jacobi iteration type of problems, such as the heat distribution problem, the game of life problem, and the Gauss-Seidel algorithm. The divide-and-conquer pattern is applicable to many sorting algorithms. The all-to-all pattern is used for problems such as the n-body problem.

The first tool that was developed is called the *Seeds* framework, created by Jeremy Villalobos in his Ph.D. dissertation project. The Seeds framework requires the programmer only to implement four methods in Java: a diffuse method, a gather method, a compute method, and a bootstrap method. The Seeds framework will self-deploy a parallel program on a distributed-system implementing the specified communication pattern.

The second tool is the *Paraguin* compiler, a compiler-directive approach. The user specifies the pattern as well as the functions and data used in the computation using #pragma statements. The compiler generates MPI code to implement the specified pattern using the given functionality and data.

We do not go into great detail describing these tools here because they have been described in more detail in previous work. We refer the reader to [2], [3], and [16] for more detail on these tools. We have also developed a new tool called Suzaku, which uses macros to generate MPI code. This tool is new and was not included in this study.

V. Materials and Assignments

In Fall 2012, we tried a top-down approach to teaching the course. We started with the topic of patterns: low and high level. We introduced the Seeds framework, followed by the Paraguin compiler. Next, we taught them message-passing in a distributed-memory environment using MPI, followed by a shared-memory environment using OpenMP. After more discussion of patterns and algorithms, we finished the semester on parallel computing on GPUs using CUDA.

What we found is that computer science students prefer a bottom-up approach rather than top-down. This is based upon direct feedback from students.

In Fall 2013, we reorganized the material to be more bottom-up. We started with share-memory programming and OpenMP, followed by distributed-memory and MPI. We then introduce a few patterns and proceeded into using the Paraguin compiler. Next we introduced more patterns and taught them about the Seeds framework. We also added at this point the Suzaku framework, which is a new tool that automatically creates MPI code based upon macros. The Suzaku framework is very new and was not part of this study. The second half of the semester proceeded similar to Fall 2012. After more discussion of patterns and algorithms, we finished the semester on parallel computing on GPUs using CUDA.

A. *Fall 2014 course*

During the three study we tried both top-down and bottom-up approaches when it comes to introducing the programming tools. Originally we had five quite large assignments top-down, starting with Seeds, then Paraguin, then OpenMP, and MPI. We discovered two issues that students had with this approach. First they were challenged with the top-down approach and preferred and learned better by starting with low level tools such as OpenMP and MPI and then working upwards towards the pattern based tools. Secondly we found the large assignments were challenging to students. The assignments would begin with a simple tutorial section running provided code and then later in the assignment, students would have to write code themselves, but many students would wait to the last moment to look at the assignment at all. In Fall 2014, we divided the assignments into smaller parts, each with a one or two week deadline. The first “pre-assignment” was simply installing the virtual machine we also introduced in Fall 2014, and then seven main assignments. The first assignment was a tutorial on OpenMP with most of the code given and a one week deadline. The second assignment required students to write OpenMP code (heat distribution problem with X-11 graphics). Then we moved onto an MPI tutorial with most the code given and an MPI assignment where students had to write most of the code themselves (Monte Carlo Pi calculation). Then we had assignments on Paraguin, Seeds, and CUDA. We found this approach worked better and would recommend it. Students gained more confident by doing the tutorial assignments and having short deadlines was also better to keep students on track. This is especially important with our televideo classes where remote students do not have physical access to the instructor or TAs. After the Fall 2014 class we have taught the class in Spring 2015 and Fall 2015 as an online class at UNC-Charlotte with the same structure and again shorter but more assignments kept students on track.

The virtual machine we also introduced in Fall 2014 significantly helped reduce software and cluster issues. Students installed our VM on their own computer to do most of the code development before finally testing their code on our parallel programming clusters. In fact nowadays with all computers being multicore, students can do everything on their own computer and possibly still demonstrate speedup. Parallel programming can now be taught without the need for a remote cluster (except for CUDA programming).

VI. Survey Instruments

During the three semesters in which the course was taught, students were invited to provide feedback that would help us assess the effectiveness of our tools and teaching materials. Feedback was collected by the external evaluator via three surveys: a pre-, mid-, and post-course survey. Students who provided consent and completed each of the three surveys were entered in a drawing for one of eight $25 Amazon gift cards. Table 1 shows the response rate for each survey for each course offering.

The purpose of the pre- and post-semester surveys was to assess the degree to which the students learned the material taught during this offering. A set of seven pre-course items were developed for this purpose. The items were presented with a six-point Likert scale from “strongly disagree” (1) through “strongly agree” (6). Table 2 shows the questions that were on these surveys. There were additional questions on the post-semester survey related to students’ feedback of the course, but the results of those questions are not discussed in this paper, because they are focused upon the delivery, helpfulness of the instructors, required learning outside the classroom, and suggestions for improvement. The information garnered from that data did not focus on the effectiveness of our approaches.

Table 1: Survey Response Rates

Project Year	Date	Survey	Response Rate (%)
1	9/4/12 – 9/18/12	Pre	36
	10/11/12 – 10/26/12	Mid	29
	11/26/12 – 12/3/12	Post	28
2	8/30/13 – 9/13/13	Pre	55
	10/22/13 – 11/4/13	Mid	32
	11/25/13 – 12/9/13	Post	45
3	9/2/2014 – 9/16/14	Pre	28
	11/5/14 – 11/19/14	Mid	22
	12/3/14 – 12/17/14	Post	16

Table 2: Pre- and Post-Semester Survey Questions

Item
I am familiar with the topic of parallel patterns for structured parallel programming.
I am able to use the pattern programming framework to create a parallel implementation of an algorithm.
I am familiar with the CUDA parallel computing architecture.
I am able to use CUDA parallel computing architecture.
I am able to use MPI to create a parallel implementation of an algorithm.
I am able to use OpenMP to create a parallel implementation of an algorithm.
I am able to use the Paraguin compiler (with compiler directives) to create a parallel implementation of an algorithm.

In addition to these questions, students were also asked to rate the relative difficulty of using pattern programming, MPI, the Seeds Framework, and the Paraguin compiler directives using a six-point Likert scale from “verydifficult” (1) through “very easy” (6).

VII. Results

Table 3, Table 4, and Table 5 give the student responses to the questions in Table 2 for each of the 3 course offerings. The scores included are only for the students who completed all of the surveys. With each semester, at the outset of the course, students responded in the “disagree” to “mildly disagree” range for all items presented indicating that students were not familiar with the topics or methods. However, by the conclusion of the course, students responded in the “mildly agree” to “agree” range to the same survey items, indicating that they learned the topics and how to use the methods. This result is expected.

In the Fall 2012 (Table 3), we see that the students indicated a greater comfort in developing parallel programs with the lower level tools than with the Seeds framework or the Paraguin compiler. This comfort level reversed in Fall 2013 (Table 4). In Fall 2014 (Table 5) the relative comfort level between the tools was fairly flat. Since each semester included a different cohort of students, it is difficult to compare the results directly between semesters. Clearly, the modifications made to the new tools and the teaching materials between 2012 and 2013 had a positive impact on the students’ comprehension of the new tools. The only significant changes between 2013 and 2014 was the order of the material, not the content of the material. We would like students to feel more confident using our tools than the low level tools, but our objective is for the students to feel confident creating correct parallel programs using a variety of tools at high and low levels of abstraction.

Table 3: Survey Results From Fall 2012

Item	**Pre Mean (SD) N=16**	**Post Mean (SD) N=16**
I am familiar with the topic of parallel patterns for structured parallel programming.	2.56 (1.59)	4.44 (1.09)
I am able to use the pattern programming framework to create a parallel implementation of an algorithm.	2.25 (1.61)	4.25 (0.86)
I am able to use the Paraguin compiler (with compiler directives) to create a parallel implementation of an algorithm.	1.69 (0.95)	4.13 (1.15)
I am able to use MPI to create a parallel implementation of an algorithm.	2.31 (1.40)	4.88 (0.81)
I am able to use OpenMP to create a parallel implementation of an algorithm.	2.19 (1.17)	5.06 (1.24)
I am familiar with the CUDA parallel computing architecture.	2.06 (1.57)	4.63 (0.72)
I am able to use CUDA parallel computing architecture.	1.94 (1.61)	4.44 (0.89)

Table 4: Survey Results from Fall 2013

Item	**Pre Mean (SD) N=21**	**Post Mean (SD) N=21**
I am familiar with the topic of parallel patterns for structured parallel programming.	3.14 (1.42)	4.95 (1.07)
I am able to use the pattern programming framework to create a parallel implementation of an algorithm.	3.00 (1.52)	5.05 (0.50)
I am able to use the Paraguin compiler (with compiler directives) to create a parallel implementation of an algorithm.	2.43 (1.50)	5.14 (0.79)
I am able to use MPI to create a parallel implementation of an algorithm.	2.14 (1.24)	4.95 (0.74)
I am able to use OpenMP to create a parallel implementation of an algorithm.	2.33 (1.46)	5.19 (0.75)
I am familiar with the CUDA parallel computing architecture.	2.81 (1.63)	4.71 (0.96)
I am able to use CUDA parallel computing architecture.	2.43 (1.72)	4.95 (0.67)

Table 5: Survey Results from Fall 2014

Item	Pre Mean (SD) N=8	Post Mean (SD) N = 8
I am familiar with the topic of parallel patterns for structured parallel programming.	3.25 (1.16)	4.88 (0.64)
I am able to use the pattern programming framework to create a parallel implementation of an algorithm.	2.88 (1.46)	4.75 (1.28)
I am able to use the Paraguin compiler (with compiler directives) to create a parallel implementation of an algorithm.	2.25 (1.28)	4.88 (0.64)
I am able to use MPI to create a parallel implementation of an algorithm.	3.25 (1.49)	4.75 (1.28)
I am able to use OpenMP to create a parallel implementation of an algorithm.	4.25 (1.39)	4.75 (1.67)
I am familiar with the CUDA parallel computing architecture.	2.63 (1.06)	5.00 (0.53)
I am able to use CUDA parallel computing architecture.	2.00 (0.76)	4.88 (1.25)

Table 6: Perceived Level of Difficulty with the Tools

Method	Mean(SD) Fall 2012	Mean(SD) Fall 2013	Mean(SD) Fall 2014
Pattern Programming (with Seeds)	3.63 (0.89)	3.95 (1.19)	2.88 (0.93)
MPI	3.25 (1.13)	2.05 (0.94)	2.94 (1.20)
Paraguin Compiler Directives	2.56 (1.26)	3.37 (1.26)	3.59 (1.42)

*Lower mean translates to higher difficulty on the scale provided.

The improvement of familiarity of OpenMP did not improve significantly in Fall 2014 between the pre-survey and post-survey. We attributed this to the rearrangement of the material. Introduction to OpenMP was moved earlier in the semester, and coincided with the deployment of the first survey. Clearly, some of the respondents had already seen some of the lecture materials on OpenMP.

Table 6 gives the results of the students' responses to the question about relative difficulty of the tools using a six-point Likert scale from "very difficult" (1) through "very easy" (6). In the 1st semester, students felt that the Paraguin compiler was the most difficult to use. In the second offering, students' responses indicated an improvement in the relative ease with using the Paraguin compiler, which we attribute to the improvements made to the compiler. By the final semester, the relative difficulty of these three tools reversed. Since there were no changes to the Seeds Framework or the material related to the Seeds Framework, this would suggest an improvement in the understanding of MPI, which we attribute to moving the material on MPI earlier in the semester.

VIII. Conclusions

In this three-year study, we used two pattern programming tools, Seeds and Paraguin, both of which implement patterns and hide the message passing code but using different techniques and user interface. In the Java-based Seeds, the programming does not have any access to the underlying code but the code will execute on any platform without any prior software installed (except Java). Paraguin creates MPI code that is available for the programmer to dissect. MPI must be installed to execute the code, but the framework could be integrated better and used before or after learning MPI. Both Seeds and Paraguin reduce the control that students have in how they write the code.

We have found that student often prefer to have complete control over writing the code to let them write it any way they like. Unfortunately that approach is not conducive to good programming. We attribute student's preference to the lack of training in software techniques. However we do see an advantage to having increasing level of control available.

IX. Current and Future Work

During the project we developed a third pattern programming framework called Suzaku. Although not yet fully tested in the classroom, we believe this framework might be attractive in some situations. It still requires students to write in a controlled way using patterns but the implementation is very simple and student can readily see the underlying code. Suzaku creates MPI message passing programs using C macros and routines and hides all the complexities of MPI. As such, one can start with patterns and Suzaku and study MPI later (or not even learn MPI at all) in a top-down approach or consider MPI first and patterns and Suzaku afterwards in a bottom-up approach. The computational model of Suzaku is purposely similar to OpenMP but using processes instead of threads. Parallel sections can be created just as in OpenMP. Patterns are created within a parallel section. Various patterns have been implemented including workpool, pipeline, stencil, etc.

The programmer's interface has been modeled on Seeds. Only four routines need to be implemented by the programmer for any particular pattern, initialize, diffuse, compute, and gather. Interesting we have found that all the patterns need just these routines, making it easy to learn the approach.

Acknowledgment

This material is based upon work supported by the National Science Foundation under the collaborative grant #1141005/1141006. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Astrachan, O. 1998. Design Patterns: An Essential Component of CS Curricula, *SIGCSE Bulletin and Proceedings*. 30, 1, 153-160.

[2] Ferner, C., Wilkinson B., and Heath B. Using Patterns to Teach Parallel Computing. *Fourth NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-14)*, Phoenix, AR, May 19, 2014.

[3] Ferner, C., Wilkinson, B., Heath, B. 2013 Toward using higher-level abstractions to teach Parallel Computing. *Third NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-13*), Boston, MA, May 20,.

[4] Ferner, C.S. 2006. Revisiting communication code generation algorithms for message-passing systems, *International Journal of Parallel, Emergent and Distributed Systems (JPEDS) 21(5)*, 323-344.

[5] Ferner, C. S. 2002. The Paraguin compiler---Message-passing code generation using SUIF, in *Proceedings of the IEEE SoutheastCon 2002*, Columbia, SC, 1-6.

[6] Fastflow. University of Torino, Italy /Università di Pisa. http://calvados.di.unipi.it/dokuwiki/doku.php?id=ffnamespace:about

[7] Gamma, E., Helm., R., Johnson, R., and Vlissides, V. 1995. *Design Patterns.* Addison-Wesley, New York.

[8] Keutzer, K., and Mattson, T. Our Pattern Language (OPL): A Design Pattern Language for Engineering (Parallel) Software. http://parlab.eecs.berkeley.edu/wiki/_media/patterns/paraplop_g1_1.pdf.

[9] Mattson, T. G., Sanders, B. A., and Massingill, B. L. 2004. *Patterns for Parallel Programming.* Addison Wesley.

[10] McCool, M., Reinders, J., and Robison, A. 2012. *Structured Parallel Programming: Patterns for Efficient Computation.* Morgan Kaufmann.

[11] Pacheco, P., 2011. An Introduction to Parallel Programming, Morgan Kaufmann,.

[12] Renault, E., Parrot, C. 2006. MPI Pre-processor: Generating MPI Derived Datatypes from C Datatypes Automatically, in *Proceedings of the2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, Columbus, OH, August 14-18.

[13] Reyes, R., Dorta, A.J., Almeida, F., Sande, F. 2009 Automatic hybrid MPI+OpenMP code generation with llc, in *Proceedings of Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10,.

[14] Toub, S. Patterns of Parallel Programming Understanding and Applying Parallel Patterns with the .Net Framework 4 and Visual C#. Microsoft. http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=19222

[15] Barry Wilkinson and Michael Allen, Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice Hall, Upper Saddle River, New Jersey, 1999.

[16] Wilkinson, B., Villalobos, J., and Ferner, C. 2013. Pattern Programming Approach for Teaching Parallel and Distributed Computing.*SIGCSE 2013 Technical Symposium on Computer Science Education*. Denver, Colorado. To appear.

[17] Villalobos, J. 2011. Running Parallel Applications on a Heterogeneous Environment with Accessible Development Practices and Automatic Scalability. PhD diss. University of North Carolina Charlotte.

[18] Villalobos, J. Parallel Grid Application Framework. http://coit-grid01.uncc.edu/seeds/