2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

Formal Methods for Surviving the Jungle of Heterogeneous Parallelism

Ganesh Gopalakrishnan
School of Computing,
University of Utah,
Salt Lake City, UT 84112
Email: ganesh@cs.utah.edu
Research Page: http://www.cs.utah.edu/fv

***Abstract*—The parallel programming community will soon be entering the 'jungle' of heterogeneous hardware and software. Unfortunately, we are not adequately preparing future programmers (today's students) to cope with the many challenges of heterogeneous concurrency, especially in their ability to rigorously specify and verify concurrent systems. Concerted action is urgently needed to create a body of education material supplemented by effective software tools that help gain working knowledge of specification and verification techniques. We suggest funding models and incentives that can help create this material and put them into wide practice.**

***Keywords*-Parallel/concurrent programming, and education; multicore computing; teaching parallelism concepts; formal methods/debugging tools.**

I. The Heterogeneous Parallel Jungle

In his article [1], Sutter describes how the computing industry which began multicore CPU adoption around year 2004 has virtually completed this transition in every conceivable computing domain—phones, game stations, serves, and cloud machines are all now multicore CPU based. He points to the unsettling growth that lies ahead, in which a veritable 'jungle' of heterogeneous hardware and software varieties will play a role at each of these domains. This heterogeneity is also predicted by the Borkar/Chien article [2] where the authors describe the microprocessor roadmap in which commerical CPUs will end up containing six or more core types. The consequences of these predictions are dire. The ability of the computing society to make progress in computational performance is now hinged on designers' ability to deal with the high amounts and sheer variety of concurrency.

This paper is about one crucially important aspect of concurrency: *how do we now show that a heterogeneous software system is operating correctly?* Today's software testing methods—already inadequate for testing concurrent programs—will prove to be even more so in the face of growing heterogeneity. Today's labor-intensive ways of writing tests for sequential programs are untenable for concurrent programs—especially where multiple models of concurrency are involved. While formally based approaches to specification and verification are essential, we must also ensure that we do not take an 'all or nothing' approach to formality. In other words, we must offer designers a spectrum of techniques to choose from, so that they can utilize the highest levels of rigor for the most critical of components. Given the perpetual shortage of trained manpower and verification resources, the less critical components can still be verified through conventional means or through semi-formal methods. This is more or less how the microprocessor industry operates, deploying formal methods in roughly the priority order: (i) floating-point units, (ii) cache coherency engines, (iii) interconnect protocols, and (iv) everything else.

Luckily, many verification techniques are becoming more conducive to supporting the use of different verification 'strengths.' For instance, powerful verification techniques such as symbolic reasoning engines based on BDDs, SAT methods, and decision procedure based methods ("SMT, [3]") can be used on inputs and states that are partially instantiated to yield the best of both worlds (namely, execution based analysis and symbolic analysis). All this of course depends on students being taught the fundamentals of these techniques so that they enter the workforce with the required level of maturity to use the tools well as well as evolve them in interesting ways.

A. Need for Vision and Concerted Action

A vast number of promising avenues for formal specification and verification that can actually be taught to undergraduates actually exist. Yet, these avenues are not being pursued aggressively.

The problem lies with the manner in which we have gone on to include these techniques into our curricula. We do fully back the TCPP curriculum efforts which already address many of the necessary steps to achieve formal training concurrency There are also excellent alternate course proposals [4] that cover many of these topics. We now briefly discuss some of the required priorities, going forward.

1) Formal Methods are Essential: Informal explanations and direct experience with actual hardware and software are indispensible to come to grips with nuances of concurrency. Beyond that, a student must be exposed to judiciously chosen formal explanations. As a simple example, to teach a new student about the exponential growth of the number of threads interleavings, a good approach is to first derive familiar equations that characterize this growth, and then

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.164

introduce them to concepts (*e.g.*, partial order reduction) that help them compute equivalence classes of interleavings. As another example, consider the ongoing efforts on standardizing language level memory models (*e.g.*, C++0x): we believe that having a formal component to teaching is essential in this area. In the fast-moving world of concurrency, the onus is on instructors to equip students with the ability to engage in life-long learning—achieved through formally based education methods. As an example, consider one important topic in data center design: server consolidation and virtualization, assisted by innovative uses of Flash memories to relieve I/O pressures. We were pleasantly surprised to hear, in a recent talk [5] in this area, that even I/O system designers are grappling with similar memory consistency issues. A student who has been taught judiciously chosen and effective formal methods in this area—for example, the ability to write clear state transition style specifications in the Operational Semantic Style—is far ahead in terms of the ability to innovate. A quote from Sutter [6] pertaining to memory model specifications is most apropo:

> My 30 years of experience reasoning about and specifying systems has shown that the only way to understand any non-trivial system is as a state machine.

2) Needless Variety: Constructs found in various heterogeneous APIs often have significant overlap. For example, message passing constructs occur in different APIs in different-looking syntaxes, and unfortunately are likely to be taught as if they are different. Formal methods must be developed to help bridge these apparent differences, by building briding abstractions based upon a parsimonious primitive basis.

3) What was once hard may no longer be so: As an example, previously proposed curricula may have shied away from topics involving mathematical logic or programming semantics. The reality is that teaching such topics is becoming easier, thanks to the availability of an increasing number of tools. Some of our experiences with these tools are provided in § II-E.

4) It may require "industrial-strength" ideas/tools: Just because we are "only" discussing undergraduate education, there is a temptation to think that approaches known for decades (hence known to many likely teachers of a concurrency-relevant subject) are enough. For example, manually checking for adherence to best practices is the message we glean from many excellent books (*e.g.*, [7]); the trouble is that even experts don't have a handle on best practices for emerging heterogeneous APIs. Also, while students must be taught the basics of model checking, that alone does not suffice. We must also expose them to more practical tools and techniques (§ II-E).

5) Consolidating Teaching Resources: Literally hundreds of excellent projects on creating educational material and software (including some of the author's own) have been proposed and implemented; yet, many have not taken root. Without a powerful set of incentives, these ideas and resources face the uphill battle of vying for attention and commitment from teachers who are already overworked with respect to their basic survival modes of either heavy teaching or heavy research grant proposal writing.

Suitable incentives must be in place so that formal methods researchers engaged in cutting-edge concurrency research are likely to invest time/effort in curriculum development. There must also be incentives to measure the impact of these curricula through monitoring and quality assurance mechanisms that don't have to be created anew (something that is less likely to happen). We propose the use of existing conference tutorial channels as a vehicle for making this happen. In particular,

- Those funded under special initiatives on formal methods for concurrency must be required to submit tutorials to major conferences. It is quite timely to point out that *this very forum—TCPP Workshops associated with IPDPS—could evolve into one such workshop, consisting of talks, tutorials on education, and posters.*
- Funding must also be devoted to making sure that students interested in taking these tutorials compete for travel/registration awards, and the winners obtain generous travel and registration support to attend these tutorials.

II. A Sampling of Our Efforts

We now present a list of our own past efforts, which among them have had an impact, and why most of them have not had the level of adoption they have the potential for. We then detail our thoughts on how educators as well as funding agencies might approach some of the challenges, and work toward demonstrating measurable impact.

A. Formal Methods for MPI Programming

The PI was one of the tutorial instructors of a Supercomputing 2010 (half-day) and a Supercomputing 2011 (full-day) tutorial in which he taught formal methods for dynamic analysis of MPI programs. Some of these approaches have been summarized in PI's CACM December 2011 article [8]. As reported in this article, our dynamic analyzer ISP is now available with a framework called Graphical Explorer of MPI programs (GEM) which, in turn, is part of official releases of the Eclipse Parallel Tools Platform (PTP) since November 2009. During the 2011 Supercomputing PTP tutorial, ISP and GEM were remote-enabled on NCSA cluster machines, and used by nearly 30 tutorial attendees on a large (10,000+) line example. These attendees detected bugs (memory leaks) in this example during this tutorial. PI's student Gibson has helped incorporate all the texbook examples from a popular MPI programming book [9] into ISP/GEM [10] as easily accessible projects! This does make ISP/GEM available as an excellent tool using which to teach

formal methods for MPI programming, especially because it goes with examples from an actual popular MPI textbook.

More importantly, students can, for the first time, study many of the MPI concepts and behaviors through formal dynamic analysis using MPI—not relying solely on informal explanations found in books. We have demonstrated that scenarios introduced in MPI manuals can be *calculated* within ISP through its happens-before model of MPI. Such *predictive power* is what formal methods are all about—one can dispense with case-specific explanations that first of all make the material look disjoint, and second of all do not aid the student to engage in life-long learning.

Our Lack of Success, and Reasons: We really have not met the degree of success we were hoping for. Our Supercomputing 2011 experience of teaching the tutorial attendees suggests that the material can have a tangible impact. However we must now take these efforts to the next level of practicality:

• We must build formal tools that fail (or degrade) gracefully, yielding bug-reports and insights proportional to the amount of time that they are run on an example. At present, formal methods groups (including our own) have not created tools that have this feature.

• We must build open-analysis tools. It is widely known that there can't be a truly 'push-button' verification tool. Users must instead be given the ability to *script* a certain analysis "cocktail"—for instance (1) deploy symbolic verification methods to compute a general symbolic internal state, (2) concretize this state and run random testing. This will allow designers to trust and learn to use verification tools more effectively. Present-day tools do not support this usage model.

• A basic change of mindset is essential. As one example, our efforts within our campus to encourage local HPC training facilities to adopt our formal MPI analysis tools has gone nowhere. The mindset of those who teach MPI seems to be geared toward the question "how to quickly get group of students running their programs on a cluster" and not "how to ensure that they are able to achieve reliable computations."

B. Formal Methods for GPU Programming

Similar to what we did with ISP/GEM, we have begun an ambitious research program addressing formal methods for CUDA. The basis for this work was laid over two tools we have built, published about, and released. The first of these tools is PUG [11], and the second, and much more capable tool is a concolic testing and verification approach for CUDA programs embodied in a brand-new tool called GKLEE [12]. Similarly to our efforts with ISP/GEM and Pacheco's book, we are systematically working out examples from the popular CUDA book [13], and hope to have this compendium released along with GKLEE.

C. *Courses on* Practical Pallel and Concurrent Programming

The author helped run a pilot version of the PPCP course [4] at the University of Utah. This course has, since then, been also offered at the University of Washington Seattle by Musuvathi, one of the co-authors of the PPCP course. Clearly, there is enormous potential to build on PPCP—but one that again needs resources and concerted action.

D. *TV'06, PADTAD'09, EC^2*

The PI has also helped run other workshops/tutorials:

- A workshop on Shared Memory Consistency Models, Specification and Verification in 2000 - [14]
- A workshop on concurrency verification in 2000 - [15]
- "Thread Verification" (TV, [16]) in 2006
- A tutorial-day at PADTAD 2009 [17]
- The successful workshop series EC^2 [18].

There is a huge potential to collect material from these workshops and make it available to students, especially to trace the evolution of topics over this rapidly changing area.

E. Course Modules

Thanks to the bounty of tools and infrastructures others are creating, the PI has experienced encouraging success in creating very promising course curricula. Some examples are now discussed.

Python-based Course Modules: It has been our experience that by employing Python as an easily teachable and powerful programming language, we can make considerable headway into teaching formal methods even within the existing framework of undergraduate course. Here are some examples (a URL that houses our material will be maintained; a preliminary URL is [19]).

• We have incorporated much of a traditional "Models of Computation" class into this approach. Students can very concretely experiment with NFA, DFA, and regular expressions, and also experiment with other classical machines within Python. Using declarative subsets of Python (*e.g.*, using set comprehensions supported by Python), we can make Python code very close to mathematical notation! We also had luck showing both (math and Python) in juxtaposition, thus helping remove students' apprehension with math.

• We have built manipulation routines for Binary Decision Diagrams in Python. We can teach BDDs as being a special form of DFA. We can teach them mathematical logic using BDDs. Our python routines are very easy to read and understand.

• We have built Boolean Satisfiability routines in Python. These are being used to teach SAT methods.

• We have given students access to SMT solvers through a Python interface. Undergraduate students have been, with ease, able to formulate and solve familiar puzzles. Here is one original solution that a student developed The student was taught the simple board game of Ken-Ken (similar to

Sudoku) In Ken-Ken, one solves for missing numbers that satisfy arithmetic constraints. The student was asked to even solve for missing operators (missing $+$, $-$, $*$, or $/$) that might satisfy the constraints. The student had no difficulty arriving at suitable assertions with which to model and solve the problem. Things such as this would have not even been imaginable, or found feasible merely five years ago. The rapid pace of advances in solver technology bode well in terms of truly enabling high impact in formal methods education! Of course, *all these directly relate to concurrency education also*: our GKLEE tool for instance heavily relies on SMT technology.

Tools for Operational Semantics: There is enormous potential to place powerful tools (*e.g.*, [20]) for conveniently writing Operational Semantic definitions in the hands of undergraduates and graduate students. Again, these developments are (relatively speaking) quite recent, and greatly empower us to teach practically useful and impactful material in a class setting. The author is engaged in such a course [21] at the time of writing, and plans to compile his experiences in due course.

III. Concluding Remarks

In this paper, we have attempted to impress upon the reader the seriousness and urgency of equipping students with the intellectual resources necessary to deal with the upcoming chaos ("jungle") in concurrency. We argued that formal methods are a foundational pillar upon which much of a student's knowledge ought to rest. We have shown instances where these tools are becoming much more powerful, usable, and teachable—especially when coupled with declarative styles of specification afforded by modern functional languages, scripting languages, and domain-specific languages. Funding agencies must have a coherent plan of action to make things happen by channeling resources, suitably incentivizing the developments, and ensuring widespread adoption; some ideas to help them achieve these goals are also listed.

References

[1] Herb Sutter. Welcome to the jungle (on hardware/software concurrency), 2011. http://herbsutter.com/2011/12/29/welcome-to-the-jungle/.

[2] S. Borkar and A. Chien. The future of microprocessors. *Communications of the ACM*, May 2011.

[3] Satisfiability Modulo Theories Competition (SMT-COMP). http://www.smtcomp.org/2009.

[4] Caitlin Sadowski, Thomas Ball, Judith Bishop, Sebastian Burckhardt, Ganesh Gopalakrishnan, Joseph Mayo, Madanlal Musuvathi, Shaz Qadeer, and Stephen Toub. Practical parallel and concurrent programming. In *SIGCSE*, March 2011.

[5] Vikram Joshi and Prashant Radhakrishnan. Getting a million iops through code you don't own, January 2011. Colloquium, School of Computing, University of Utah.

[6] Herb Sutter. The prism memory model specification. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2197.pdf.

[7] Timothy G. Mattson, Beverly Sanders, and Berna Massingill. *Patterns for Parallel Programming*. Addison Wesley, 2005.

[8] Ganesh Gopalakrishnan, Robert M. Kirby, Stephen Siegel, Rajeev Thakur, William Gropp, Ewing Lusk, Bronis R. de Supinski, Martin Schulz, , and Greg Bronevetsky. Formal analysis of mpi-based parallel programs: Present and future. *Communications of ACM*, December 2011.

[9] Peter Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, 1996. ISBN 1-55860-339-5.

[10] Brandon Gibson. Supercomputing 2011 ACM Poster Competition Poster on integrating Pacheco's textbook examples into ISP/GEM.

[11] Guodong Li and Ganesh Gopalakrishnan. Scalable SMT-based verification of GPU kernel functions. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering (SIGSOFT FSE)*, 2010. www.cs.utah.edu/fv/PUG.

[12] Guodong Li, Peng Li, Geof Sawaga, Ganesh Gopalakrishnan, Indradeep Ghosh, and Sreeranga P. Rajan. GKLEE: Concolic verification and test generation for GPUs. In *17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2012. www.cs.utah.edu/fv/GKLEE.

[13] Jason Sanders and Edward Kandrot. *CUDA by Example*. Addison-Wesley, 2011.

[14] Tutorial and workshop on formal specification and verification methods for shared memory systems. http://www.cs.utah.edu/mpv.

[15] Workshop on advances in verification. http://www.cs.utah.edu/wave.

[16] Ganesh Gopalakrishnan and John W. O'Leary. Thread verification (tv), 2006. http://www.cs.utah.edu/tv06.

[17] Ganesh Gopalakrishnan and Eric Mercer. Tutorials day at padtad 2009. https://www.research.ibm.com/haifa/Workshops/padtad2009/tutorial.shtml.

[18] Ganesh Gopalakrishnan. Co-organizing the ec^2 workshop series. http://www.cs.utah.edu/ec2/, http://www.cse.psu.edu/~swarat/ec2/, and http://www.cse.psu.edu/~swarat/ec2-2010/.

[19] Website of "models of computation, fall 2011". http://www.eng.utah.edu/~cs3100.

[20] K and matching logic. http://fsl.cs.uiuc.edu/index.php/K_and_Matching_Logic.

[21] Website of "foundations of cs, spring 2012". http://www.eng.utah.edu/~cs6100.

[22] Jason F. Cantin, Mikko H. Lipasti, and James E. Smith. The complexity of verifying memory coherence and consistency. *IEEE Trans. Parallel Distrib. Syst.*, 16(7):663–671, 2005.