Retrospective: A Look Back at 20+ Years of Experience in Parallel Computing Education

Joel C. Adams
Department of Computer Science
Calvin University
Grand Rapids, MI 49546 USA
adams@calvin.edu

***Abstract*—The author has been teaching parallel computing since the late 1990s. This paper provides a brief history of some of the key experiences in his journey as a parallel computing educator, including formative faculty development workshops, technology shifts, NSF-sponsored projects, and courses taught. By reading this history, the reader may see how much has changed over the past two decades, and so gain insight into how to prepare for changes to parallel computing education in the future.**

Keywords—beowulf, cluster, computer science, education, high performance, history, multicore, parallel, software, supercomputing

I. Introduction

In 1996-7, the author was a young faculty member in the Department of Computer Science (CS) at Calvin University. At that time, the department had no course offerings in parallel computing and decided that this was a significant gap in Calvin's CS curriculum. The author's background was in theoretical distributed systems, which was closer to parallel computing than anyone else in the department, so the author was tasked with designing a course to fill this gap. With no direct training in parallel computing, the author had no curricular models to draw on, which was disconcerting.

Fortunately, Chris Nevison from Colgate University and Nan Schaller from the Rochester Institute of Technology led a week-long NSF-sponsored faculty development workshop on parallel computing at Colgate during the summer of 1997. This workshop included an overview of different parallel computing hardware platforms, an introduction to parallel algorithms, and hands-on practice using parallel computing software platforms. The workshop leaders also provided the syllabi of their parallel computing courses, sample assignments, PowerPoint presentations, and other useful materials. In short, this workshop provided the author with an excellent introduction to the subject, and a good starting point for developing a new course.

This paper provides a retrospective look at how Calvin's parallel computing curriculum has evolved since that time, in roughly five-year intervals. Section II describes the early versions of the course offered in the late 1990s. Section III describes how the course evolved in the early 2000s. Section IV describes how the course and curriculum changed as unicore processors began to disappear after 2005. Section V describes how the course evolved in the early 2010s to incorporate new technologies, and Section VI describes changes to the course since 2015. Sprinkled throughout are other, related events that affected the author's trajectory as a parallel computing educator.

II. The late 1990s

Calvin follows a "4-1-4" calendar in which an academic year consists of a normal four-month Fall semester, a January term (J-term), and a normal four-month Spring semester. During the J-term, a student takes one course that meets three hours per day for fifteen days. By "compressing" a fifteen-week semester into fifteen days, the J-term provides an ideal opportunity to test-drive experimental courses that cover topics outside of the normal curriculum. If successful, such courses can eventually be turned into regular courses that meet during a normal semester.

A. 1998: Parallel Computing, Iteration 1

Encouraged by his positive experience at the Colgate workshop, the author offered an initial *Parallel Computing* course during Calvin's 1998 J-term. 15 students signed up for the course. The course content was heavily based on material from the Colgate workshop; it included exposure to parallel computing concepts, with experiential learning through hands-on activities and six homework projects. The course met in the CS department's computer lab of 32 Sun workstations running Solaris, connected with Ethernet. The course text was Pacheco's *Parallel Programming with MPI* [19].

The conceptual material included coverage of the Flynn Taxonomy; parallel hardware organization and network topologies; parallel algorithms (e.g., odd-even transposition sort, parallel matrix multiplication, parallel Eratosthenes Sieve); analysis of parallel complexity (speedup, parallel efficiency, scalability and Amdahl's Law); and parallel computing history.

One goal of the course was to let students experience two different models of parallelism: SIMD and MIMD computing. For SIMD computing, the author installed *Parallaxis-III* [11] on each lab workstation. Parallaxis is a machine-independent framework for defining virtual SIMD architectures, specifying parallel algorithms (using Modula-2 syntax), and running a specified algorithm on a defined architecture. To support MIMD computing, the author installed *MPICH* [16] on each workstation. MPICH is a free, open source implementation of the message passing interface (MPI).

Students spent the first half of the course using Parallaxis to explore SIMD computing. Parallaxis includes a simulator that lets a student run their parallel algorithm on their workstation to experience pseudo-parallel execution. Students used Parallaxis to complete three SIMD projects of increasing difficulty levels.

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

In the second half of the course, the students explored MIMD computing using MPICH. By using the lab as a network-of-workstations (NoW) multiprocessor, students could complete hands-on MPI active-learning activities, plus three MPI projects of increasing difficulty levels.

Using the lab as a NoW multiprocessor provided students with hands-on experience using MPI, but it did not provide them with a realistic experience of the benefits of parallel computing. More precisely, students were unable to experience consistent speedup because the remote MPI processes they were launching were competing with the processes launched by other students for the NoW's limited unicore CPU resources. Students who were willing to come to the lab early in the morning (i.e., when no one else was present) could have all of the workstations at their disposal and experience consistent parallel speedup, but this was not a general solution to the problem. The only good solution to this problem was to somehow acquire a dedicated multiprocessor for students to use.

B. 1999: A First Beowulf Cluster

One of the parallel hardware topics covered in the course was the *Beowulf cluster*, created by Thomas Sterling and Donald Becker at NASA [10]. A Beowulf cluster is a dedicated multiprocessor built from commodity off-the-shelf PCs, connected with standard network fabric (e.g., Ethernet), and running open source software (e.g., Linux, MPI, etc.).

In January 1999, the author submitted a proposal to NSF's Major Research Instrumentation (MRI) program, requesting funding to build a Beowulf cluster for research and education at Calvin. This proposal was not funded, and a key reason was a reviewer's skepticism that an undergraduate institution could build such a cluster. At the time, every cluster listed on the Beowulf website was either built by PhD researchers working at a government lab or by graduate students in a research lab.

However, one of the students in the author's 1998 course—Mark Ryken—was fascinated by *Loki*, a Beowulf cluster built by Michael Warren at Los Alamos National Labs [14]. Loki consisted of sixteen Pentium-200 PCs connected with Ethernet in a star+hypercube (4D) topology. Ryken decided to build a similar cluster as his CS senior capstone project. Ryken worked part-time in Calvin's Information Technology group and leveraged his connections there to acquire a dozen castoff 25-Mhz Intel-486 PCs. From these, he was able to assemble nine working PCs, providing a head node plus eight compute nodes. An internal Calvin grant provided funds to acquire a Fast Ethernet switch plus enough network interface cards to connect these PCs into a star+hypercube (3D) topology. The result was *MBH'99* that, at various times, stood for either "Mark's Beowulf Hypercube" or "Mark's Big Headache" depending on how the project was going. With 25-MHz I-486s, MBH'99 was too slow to be of practical use in solving real-world problems, but it served to demonstrate for the first time that an undergraduate student could build a working Beowulf cluster [2].

III. The Early 2000s

Calvin discourages faculty members from offering the same J-term course in consecutive years, so it was January 2000 before the author's parallel computing course was offered again.

A. 2000: Parallel Computing, Iteration 2

During the 2000 J-term, the author offered the *Parallel Computing* course a second time, with an enrollment of 16 students. This second offering followed the same syllabus as the 1998 course, with minor tweaks to fix issues that arose in the previous offering of the course.

B. 2000: A High Performance Beowulf Cluster

Also in January 2000, the author submitted a revised NSF-MRI proposal to build a Beowulf cluster at Calvin, using MBH'99 as preliminary work to demonstrate the feasibility of the project. This proposal was approved and about $60,000 was awarded in August 2000. During Fall 2000, the author designed the new cluster, purchased the components, and recruited students to help assemble and configure it. The new cluster was dubbed *Ohm* (for *O*ur *H*ypercube *M*ultiprocessor) and it became operational in 2001 [3]. Ohm consisted of 18 nodes plus a file server node to let all nodes access users' home directories. Each node contained a 1-GHz AMD Athon-64 CPU; the nodes were connected by Gigabit Ethernet in a star+hypercube (4D) topology. Using HP-Linpack, Ohm was benchmarked at 10.4 GFlops (R_{Max}), which was "high performance" at the time.

In 2001, the department was seeking a full-time sysadmin, so a series of upper-level students were hired and trained to administer and maintain Ohm. During summer 2002, the department hired a full-time sysadmin. Responsibility for administering Ohm was included as part of this new hire's job description. Initially, this new sysadmin just supervised the students who performed the day-to-day administration of Ohm, but he took on a more active role as students graduated.

It is worth noting that Ohm's primary purpose was to support multidisciplinary research, but most Calvin research takes place during the summer, providing open time during the academic year to use the cluster for educational purposes.

C. 2002: High Performance Computing, Iteration 1

With the acquisition of a dedicated Beowulf cluster, the 2002 J-term course was renamed *High Performance Computing* and the course was revised to integrate the cluster. 13 students enrolled in the course, all males. The content of this HPC course was similar to the previous courses, except that in the second (MPI) half, after using the CS lab to develop and debug their MPI programs, students now transferred their programs to Ohm and ran them there with different numbers of processes. This provided them with far more accurate timing data, improving the students' abilities to assess the scalability of their programs. By running their programs via a batch queueing system on Ohm, students could experience the speedup of parallel computing, resulting in a much-improved learning experience.

D. 2003: High Performance Computing, Iteration 2

For a reason that is now lost in time, the author offered Calvin's HPC course again during the 2003 J-term—two years in a row. 15 students enrolled. This course differed from the previous courses in two significant ways.

The first difference came in response to student feedback: Students in the 2002 course wanted earlier and more hands-on experience using Ohm, so the segment on MPI and distributed-memory parallelism was moved to the first half of the course.

The second difference was driven by external forces: Symmetric multiprocessors (SMPs) were increasingly affordable and commonplace, so the department had acquired a two-CPU Sun SMP system. Coverage of multithreading and shared-memory parallelism was of higher priority than SIMD computing, so the second half of the HPC course was mostly devoted to SMP computing using OpenMP [18]. Multithreading thus replaced most of the SIMD computing content.

E. 2004: The NCSI Workshop at Oklahoma

In summer 2004, the author attended a workshop sponsored by the *National Computational Science Institute* (NCSI) and *Shodor Education Foundation* [24] at the University of Oklahoma. This workshop focused on using MPI to solve science problems, and let the attendees explore MPI-based solutions to a variety of compelling problems, such as an N-body galaxy-formation simulation, a Monte Carlo forest fire simulation, and others. The author subsequently used these examples to "refresh" the content of his HPC course (see below).

One of the many interesting sessions at this workshop was a presentation of *LittleFe* [21], a six-node Beowulf cluster. Designed by Charlie Peck, Paul Gray, Tom Murphy, and David Joiner, LittleFe was small enough to fit into a checked-luggage suitcase but useful for demonstrating parallel speedup. One could (for example) run the N-body galaxy-formation simulation with one process to watch the galaxy slowly take form, then re-run the simulation with six processes and watch the galaxy form about six times as fast. A portable cluster like LittleFe opens up many possibilities, ranging from taking the cluster to a technical conference to live-demo parallel software, to taking the cluster to a high school science class to demonstrate how parallel computing can accelerate scientific discovery.

Following this workshop, conversations between the author and Charlie Peck subsequently catalyzed a series of *LittleFe Buildouts* at the annual SIGCSE and Supercomputing conferences. At these day-long events, each participant would arrive, receive a box of LittleFe hardware components, spend the morning assembling their own LittleFe, spend the early afternoon installing and configuring software (including Linux, MPI and MPI-based examples), and then spend the rest of the day learning the basics of MPI. Thanks to Intel sponsorship, at the end of the day, each participant could take home the LittleFe s/he had assembled. Through these Buildout events over the next several years, hundreds of institutions received a free Beowulf cluster suitable for teaching students about parallel computing.

F. 2005: Distributed Computing in Iceland

The author spent the first six months of 2005 as a Fulbright scholar in Iceland, at what is now the School of Engineering of Reykjavik University. While there, he was asked to teach a course on distributed-memory parallel computing using MPI. The university had no distributed-memory multiprocessor, but provided funds to build one, so the author and his 14 students spent the first few weeks of the course designing and building a six-node Beowulf cluster. Each node contained a 3.8-GHz Intel Pentium-4 CPU, and the nodes were connected with Gigabit Ethernet. The students named their new cluster *Sleipnir*, after Odin's eight-legged horse. Using HP-Linpack, Sleipnir was benchmarked at 20.25 Gflops (R_{Max}), which was quite impressive for such a modest cluster.

Some of the students in this course were from eastern Europe and did not understand spoken English, making it a challenge to explain the behavior of the various MPI commands. However, these students were adept at reading C code. After discovering this, the author replaced most of his Powerpoint lectures with live-demoes of a set of minimalist programs, each containing just enough code to illustrate the behavior of a particular MPI command: send-receive, broadcast, reduce, scatter, gather, and so on. By distilling each program down to only what was needed to illustrate a given MPI behavior, running the program, and then relating its output-behavior to the source code that produced that behavior, the author was able to teach these students the basics of distributed-memory multiprocessing using MPI, despite the communication challenges.

These minimalist programs were so useful for explaining MPI commands, that upon returning to Calvin, the author continued to use them in his courses, and to create additional minimalist programs to illustrate various parallel design patterns [15] in MPI, OpenMP, and other parallel platforms.

G. 2005: High Performance Computing, Iteration 3

In 2005, the author offered the HPC course for the first time during the regular Fall semester instead of the J-term. With CS enrollments declining nationwide, six students enrolled. This course was a significant revision in several ways.

One change was that a new course text was adopted: Quinn's *Parallel Programming in C with MPI and OpenMP* [23]. The 2003 course syllabus was expanded for this course to match the presentation of material in the new text.

Another change was that science problems from the 2004 NCSI workshop (e.g., the forest fire simulation) were incorporated into the course. These replaced less-compelling examples, and so did not alter the structure of the course.

Another change was that the author replaced some of his Powerpoint presentations with live-demos of the minimalist MPI programs he had developed in Iceland, as well as new minimalist OpenMP programs created for the course to illustrate OpenMP construct behaviors. The resulting set of programs formed the basis for the *patternlets*—minimalist programs illustrating parallel design patterns—that have been shown to improve student understanding of parallel concepts [6].

With the expanded focus on HPC, another change was the addition of a lecture on how to improve program performance via compiler optimizations, introducing students to different optimization techniques and discussing how to implement them by hand in "cutting edge" languages whose compilers (or interpreters) did not support optimization.

The author noticed that in the 15-day J-term course, students were only taking that one course, allowing them to focus on it exclusively; however, the J-term's compressed schedule did not leave much outside-the-classroom time for students to digest and really internalize the parallel way of thinking. Moving the course to the regular semester expanded its duration to 15 weeks, which gave students more outside-of-class time to digest this new way of thinking, and thus gain significantly more skill.

One final observation from this course was that students noticed they could get faster performance testing and debugging

their MPI programs in our lab than they could on Ohm. The workstations in our lab were replaced every three years; thanks to Dennard Scaling, they now contained 3.2-GHz AMD Athlon-64 CPUs, which were much speedier than the 1.0-GHz Athlon-64 CPUs in Ohm. Just four years old, Ohm was starting to show its age.

IV. 2006-2010: The Multicore Era Begins

Early in 2006, the author submitted a follow-up proposal to the NSF-MRI program for funding to replace Ohm with a new cluster. This proposal was not funded, but the program officer encouraged the author to revise and resubmit the next year.

In 2005, AMD had released their dual-core Athlon-64 X2 CPU, and in 2006, Intel followed with their Core-2 Duo CPU, signaling the beginning of the multicore era. Also, Gigabit Ethernet (1000Mbps) adaptors replaced "Fast" Ethernet (100Mbps) as the standard on-board adaptor on most computer motherboards. These two changes—multicore CPUs and standard Gigabit Ethernet—opened up the possibility of building a portable Beowulf cluster with much greater computational density (and so higher performance) than was previously possible.

A. 2006: Microwulf, a Portable HPC Cluster

During the 2006-2007 academic year, the author designed and helped senior Tim Brom assemble Microwulf, a personal, portable Beowulf cluster [4]. Unlike LittleFe, the goal of the Microwulf design was to achieve as much computational performance as possible within a $2500 budget, while maintaining portability. By using microATX motherboards, 3.8-GHz AMD Athlon-64 X2 (dual core) CPUs in each of four nodes, and connecting those nodes with Gigabit Ethernet, Microwulf achieved 26.25 Gflops (R_{MAX}) on the HP-Linpack benchmark for just $2,470, making it the first Beowulf cluster to break the $100/Gflop price barrier. (By August 2007, the price of Microwulf's components had decreased to $1,255, improving its price/performance ratio to less than $48/Gflop.) At 11x12x17 inches and weighing just 31 pounds, Microwulf was small and light enough to fit on a desktop or in a checked-luggage suitcase.

Early in 2007, the author submitted a revised proposal to the NSF-MRI program for funding to replace Ohm. This proposal was funded late in the summer of 2007, providing about $206,000 for a new Beowulf cluster for multidisciplinary research (and education) at Calvin.

B. 2007: High Performance Computing, Iteration 4

In Fall 2007, the HPC course was offered again, with 12 students enrolling. The funding for the new cluster was awarded too late to be useful for the Fall 2007 semester. However, with four-node Microwulf being more than 2.5 times faster than 18-node Ohm, the author revised his Fall-2007 HPC course to use Microwulf instead of Ohm for the MPI segment of the course, and to use one of Microwulf's dual-core nodes instead of the department's comparatively slow SMP system for the OpenMP segment.

Another difference was that discussions of new MPI-2 and OpenMP 2.0 features were incorporated into the MPI and OpenMP segments of the course. For example, since sequential I/O has historically been a key bottleneck in many parallel computations, MPI-2 contained a new parallel I/O mechanism, so coverage of this new mechanism was incorporated into the HPC course. Likewise, OpenMP-2.0 contained a new mechanism for user-defined reduction operations, so coverage of that was added to the course. Both MPI and OpenMP continue to evolve, so these standards must be regularly reviewed and course materials updated as appropriate.

To make room for these changes, the coverage of the history of parallel computing was slightly compressed/reduced.

By replacing Ohm with Microwulf, the lab workstations were no longer faster than the course Beowulf cluster, but with just eight cores, Microwulf's scalability was limited; it was faster than the lab's NoW multiprocessor using 1-8 MPI processes, but the two were about the same with 16 processes. If a student had the lab to herself, the NoW's slower CPUs could outperform Microwulf's faster CPUs at 32 or more processes.

Ohm continued to be used as a research multiprocessor by Calvin's scientists, but with funding acquired for a new cluster, its time was clearly coming to an end.

C. 2008: Dahl, a new Beowulf Cluster

Components for the new cluster were purchased in late 2007. It was assembled, installed, and configured in early 2008, and was fully operational by May of that year, at which point Ohm was decommissioned. Since this new cluster would be heavily used for running simulations, it was named *Dahl* in honor of Ole Johan Dahl, the Norwegian computer scientist who co-invented the Simula programming language.

Dahl consisted of a full rack of 45 nodes, connected using a 20Gbps Infiniband network. Each node contained at least two Intel 2.2-GHz Xeon quad core CPUs—over 360 total cores—providing 3.7 Tflops (R_{PEAK}) performance [5]. As a Tflops-scale machine, Dahl was a major upgrade to both Ohm and Microwulf; it greatly accelerated several Calvin scientists' research programs, resulting in over 20 publications.

D. 2008: Parallel Computing in CS2

By 2008, dual- and quad-core CPUs were ubiquitous, and it was apparent that in the near future, virtually all software would be running on multicore processors. In order for the performance of software to improve as it was ported from devices with fewer cores to devices with more cores, the software would have to designed and implemented with parallelism in mind. To the author, it was evident that all CS majors needed to gain experience with parallel computing, specifically shared-memory parallelism via multithreading.

The author saw two options for ensuring that all CS majors gained experience with parallelism:

1. Add a new parallelism core course to the CS curriculum.
2. Incorporate parallel topics in each existing core course (as appropriate) in the CS curriculum.

The CS curriculum is quite full, and there are other topics competing for coverage (e.g., databases, security), making option #1 problematic. For option #2, getting faculty to agree to incorporate parallelism into their courses was a problem; some faculty hoped it was a fad that would just go away.

The author regularly taught Calvin's CS2 course (*Data Structures*, implemented in C++), so as a first step toward option #2, the author added a week on parallelism and multithreading to his CS2 course during the 2008-9 academic year. To make room for this addition, a week's worth of coverage of graphs was shifted from CS2 to CS3.

Thanks to the use of C++ in CS2, it was fairly easy to introduce multithreading using OpenMP. The author gave three lectures, in which he used OpenMP patternlets to live-demo fork-join multithreading, race conditions, synchronization, and reductions. He also created a lab exercise in which students timed sequential Matrix addition and transpose operations, used OpenMP to parallelize those operations, timed the parallel versions using 1, 2, 4, 6, and 8 threads, and then used a spreadsheet line-chart to visualize the change in performance. The author also had the students parallelize the other Matrix operations as a homework project, and updated the final exam with four questions related to multithreading.

The author also regularly taught Calvin's *Operating Systems & Networking* course, a core CS course in which all majors are introduced to shared-memory and distributed-memory concurrency. The shared-memory content was easy to extend with material on thread performance, software issues specific to multicore cache performance (e.g., false sharing), and so on.

Through these changes, the author took first steps toward ensuring that all Calvin CS majors learned about multithreading.

E. 2009: High Performance Computing, Iteration 5

In Fall 2009, the HPC course was offered again, with an enrollment of 7 students. The course content was stable: once again, the first half covered distributed-memory parallelism using MPI and the second half covered shared-memory parallelism using OpenMP, but with Dahl replacing Microwulf. Since each of Dahl's nodes now had at least 8 cores (some had 16), students could use it to better test program scalability in both the MPI and OpenMP course segments.

As in previous HPC offerings, students developed each program using the department's lab workstations and then moved that program to Dahl to empirically measure the program's scalability. The 2009 lab workstations contained 3.6-GHz Intel i7 (quad core) CPUs that were over 50% faster than the dual 2.2-GHz Intel Xeon (quad core) CPUs in each of Dahl's compute nodes. Because of this clock-speed difference, the lab NoW multiprocessor would typically outperform Dahl for MPI computations using 1, 2, 4, and 8 processes. Beyond that point, the performance of lab NoW multiprocessor would plateau, but Dahl's performance would continue to improve through 16, 32, 64, 128, and 256 processes. The performance cross-over point where Dahl would match the lab NoW was usually about 16 processes. At 512 processes, Dahl's performance would plateau, letting the students directly experience the relationship how hardware constraints ultimately limit program scalability.

Similarly, for OpenMP computations, the lab workstations would generally outperform Dahl when using 1, 2, and 4 threads; but Dahl's nodes would prevail using 8 and 16 threads. The 2009 HPC course thus let students directly experience the strengths and weaknesses of NoW multiprocessors vs. Beowulf clusters, shared-memory vs. distributed-memory, and so on.

V. 2010-2015: Accelerating Parallelism

In 2010, the author used the last of the NSF-MRI grant funds to add an "accelerator" node to Dahl, containing an eight-core Intel Xeon CPU, an Nvidia GeForce GTX 470 GPU, CUDA, and OpenCL. This and other developments set the stage for expanding the coverage of parallel topics at Calvin.

A. 2010: The TCPP Early Adoptor Program

In 2010, the NSF/IEEE TCPP group began their *Early Adopter Program* [17], which offered mini-grants for CS faculty or departments to adopt their Curriculum Recommendations [22] and incorporate parallelism into CS courses. As department chair, the author applied for and received one of these mini-grants, which he then used as to provide financial incentives for his faculty members to incorporate a week's worth of parallel computing content into CS core courses, specifically:

- In the *Algorithms and Advanced Data Structures* course, the instructor added coverage of parallel algorithm design, specific parallel algorithms, distributed graph algorithms, and parallel asymptotic analysis.
- In the *Programming Language Concepts* course, the instructor expanded coverage of shared memory parallelism, including a new lab exercise in which students explored the parallel mechanisms of Ada, C++, and Ruby, measuring and comparing each program's scalability.
- In the *Intro to Computer Architecture* course, the instructor expanded the coverage of multicore processor and cache organization, and the implications for bus bandwidth, main memory, memory controllers, and so on.
- In the *Software Engineering* course, the instructor expanded the coverage of how to create distributed computing apps that use cloud services via those services' APIs.

All of these are core CS courses at Calvin, so the TCPP Early Adopter program played a key role in providing the incentives needed to get other Calvin faculty to incorporate parallelism into their courses. Through these changes, Calvin CS majors receive exposure to parallel computing in most of their core courses.

B. 2010: CSinParallel, Iteration 1

In summer 2010, the author attended the ITiCSE conference in Ankara, Turkey. There he participated in a working group organized by Dick Brown and Libby Shoop who had launched *CSinParallel.org*, an NSF-funded web-repository for modular parallel computing course materials that had been tested at multiple institutions [12]. One of the outcomes of this working group was an influential paper on how to incorporate parallel computing topics into the CS curriculum [13]. Another outcome was that the author was invited to join the CSinParallel project, laying the groundwork for a follow-up NSF proposal.

C. 2011: High Performance Computing, Iteration 6

The author offered Calvin's HPC course again in Fall 2011, with 12 students enrolling in the course. The major change in this iteration was the addition of a new segment on accelerators, specifically CUDA and OpenCL on GPUs. Several lectures about CUDA and OpenCL were added, plus lab exercises and projects for both of these frameworks.

To make room for this material, some of the OpenMP and shared-memory parallel content was trimmed from the course. This was not a problem because these topics were now receiving significant coverage in the core CS courses, making the coverage in the HPC course redundant.

Most students had no problems completing the CUDA material, but the majority found OpenCL to be very challenging. OpenCL's added complexity (e.g., discovering platforms and devices, setting up queues for each, compiling the kernel for each, and so on) seemed to cross a cognitive threshold for these students; many just could not implement OpenCL solutions.

D. 2012: CSinParallel, Iteration 2

In 2012, the CSinParallel group applied for and received NSF TUES-2 funding to develop new parallel computing modules, expand the *CSinParallel.org* website with new interfaces, and hold faculty development workshops to help CS faculty start incorporating parallel topics into existing CS courses. Over the next few years, this group was highly productive, developing 15 new course modules; creating new search-interfaces for *CSinParallel.org* based on course, software platform, or hardware platform; hosting 19 conference or summer faculty development workshops; organizing five conference special sessions or panels; and giving numerous presentations, all promoting parallel computing education.

E. 2013: ACM/IEEE CS Curriculum 2013

In 2013, the *ACM/IEEE CS Curriculum Recommendations* (CS 2013) were released [1]. Thanks to the active involvement of NSF/IEEE TCPP representatives, CS 2013 included parallel and distributed computing (PDC) as a new knowledge area and recommended all CS majors receive significant exposure to PDC, especially the shared-memory parallel techniques needed to make efficient use of the multicore CPUs in most devices.

F. 2012-2013: Coprocessors

In 2012, Adapteva released the Parallela, a single-board computer (SBC) the size of a credit card that provided a 1-GHz dual-core ARM processor plus a 16-core coprocessor, for $99. (Note: With 18 1-GHz cores, the Parallela's R_{PEAK} value was similar to that of Ohm, Calvin's first Beowulf cluster!) The same year, Intel released the Xeon Phi, a family of coprocessors offering 57 or 61 x86 cores, depending on the model purchased; later models offered 64, 68, and 72 cores. In 2013, China's Tianhe-2 system burst onto the scene as the world's fastest supercomputer by using 48,000 Xeon Phi coprocessors. Coprocessors seemed poised to be the future of HPC.

During summer 2013, Calvin acquired a Parallela and a Xeon Phi 3120 coprocessor. The Parallela operated as a standalone computer; the Xeon Phi was a PCI card added to Dahl's accelerator node. Each of the Phi's 57 1.1-GHz cores contained four hardware threads, providing significant computational power. To support communication between the cores, the Phi contained a sophisticated internal network that allowed it to be used as a cluster-on-a-chip with MPI, or as a shared-memory manycore processor using OpenMP.

G. 2013: High Performance Computing, Iteration 7

In Fall 2013, the author offered the HPC course again, with 23 students enrolling. This iteration was a significant overhaul of the course, with (i) the adoption of a new textbook, Pacheco's *An Introduction to Parallel Programming* [20], and (ii) the use of parallel design patterns [15] as a unifying theme throughout the course. The general structure of the course remained similar to the 2011 course: the first half used MPI to explore distributed-memory parallelism; a segment that used OpenMP to explore shared-memory parallelism; and a segment on accelerators. However, parallel design patterns were used to provide a thematic structure within each segment, and snippets of parallel computing history were interwoven throughout.

The new textbook included a unit on Pthreads, so the author added a new week on Pthreads between the MPI and the OpenMP segments. Having to solve a problem using Pthreads' explicit multithreading made them far more appreciative of OpenMP's implicit multithreading!

Another change was the addition of a week on heterogeneous architectures between the OpenMP segment and the accelerator segment, with lectures plus an MPI+OpenMP lab exercise and project.

A final change is that, where the 2011 course had only looked at GPUs in its accelerator segment, the 2013 course added a week's worth of content on coprocessors. This included lectures on the Parallela and Xeon Phi architectures, plus a lab exercise and assignment to provide hands-on experience using the Xeon Phi. To make room for this content, the OpenCL content was reduced to a single lecture, leaving the CUDA content as the primary GPU component.

H. 2014-15: Seeing Parallelism

As part of his work with the CSinParallel group, the author began work in 2014 on real-time visualizations of shared-memory parallel algorithms. Shared-memory parallelism was a logical starting point for visualization, since parallel entities would need to draw on a shared canvas. This work turned into a multiyear project that has continued to the present.

Since existing graphics libraries were not thread-safe, the first step was to create such a library. The result was the *Thread Safe Graphics Library* (TSGL), an object-oriented C++11 library that allowed multiple threads to safely draw to the same canvas in near real-time. In Spring 2015, the author ran an experiment, in which half of his CS2 (*Data Structures*) students did the traditional lab exercise in which they parallelized Matrix operations, while the other half of the students did an experimental lab exercise in which they parallelized an image-processing operation. Thanks to TSGL, the latter group could watch the parallelization occurring in real-time. The results of this controlled experiment provided strong evidence that the visualization improved student understanding of a key parallel abstraction [7]. As a result, the Matrix-processing exercise was replaced with the visual image-processing exercise in all subsequent offerings of the author's CS2 course.

I. 2015: High Performance Computing, Iteration 8

In Fall 2015, the author offered Calvin's HPC course again, with 28 students enrolling. One change in this iteration was that the parallel visualizations created using TSGL were incorporated into the course, allowing students to see the behavior of specific parallel patterns (e.g., parallel loop).

Another change was catalyzed by CSinParallel: By summer 2015, this group had authored several modules using *Apache Hadoop* [8] and its *Hadoop Distributed File System* (HDFS) to explore the *MapReduce* framework. With HDFS providing a distributed file system to spread large data sets (too big to fit into a single computer's memory) across a cluster, plus Hadoop for processing those data sets in parallel, this technology was interesting and useful for students to experience. To introduce students to this technology, a lecture was added explaining the MapReduce conceptual framework, plus a lab exercise to provide hands-on experience using HDFS and Hadoop. To make room for this content, the coverage of HPC history was compressed to a single lecture.

One other observation from this iteration was that Dahl was now 7 years old; the CS lab machines had been replaced twice since Dahl was created, making it increasingly hard for Dahl to surpass the CS lab NoW multiprocessor's performance. Dahl's nodes also began failing, indicating it was nearing its end.

VI. 2016-Present: Always Reforming

Parallel technology continues to evolve. Even single board computers (e.g., the Raspberry Pi) now have multicore CPUs, new languages are appearing with parallel computing support, and some older languages are enhancing their parallel support. As the parallel computing landscape continues to evolve, parallel computing courses must also continue to adapt.

A. 2016-18: A Beowulf Cluster for Data Science

In 2016-17, the author chaired a Calvin committee to design a new B.S. program in Data Science. This program consisted of roughly 40% CS coursework, 40% mathematics or statistics coursework, and 20% data-specific topics not covered in either CS or statistics courses.

To support the needs of this new program (i.e., to store and process large data sets), and because Dahl was deteriorating, the author wrote an NSF-MRI proposal for a new multidisciplinary research cluster costing about $260,000. This proposal was funded late in the summer of 2017. The author spent the Fall of 2017 designing and then purchasing the new cluster, which was named *Borg* in honor of computer scientist Anita Borg. To store data, Borg was equipped with 100TB of tiered storage, using NVMe for parallel access. To process data, Borg was configured with a head node with two 3.6-GHz Xeon Gold (4-core) CPUs and 96 GB of RAM; twenty compute nodes, each with two 3.2-GHz Xeon Gold (8-core) CPUs and 96GB of RAM; a virtualization node for running the web services needed by different research projects (configured identically with the head node); and a GPU/high-memory node containing the same CPUs as the compute nodes, 768 GB of RAM, and four Nvidia Titan V graphics cards. Each of these cards has 5120 CUDA cores for GPU computations plus 640 Tensor cores for machine learning projects. The nodes were connected with a 100Gbps Intel Omnipath data network and a 40Gbps Ethernet administrative network. Borg became operational in May 2018 and was immediately put to heavy use by Calvin researchers.

B. 2017: High Performance Computing, Iteration 9

In Fall 2017, the author offered his HPC course again, with 21 students enrolling. Because Borg was still in the design stages during this semester, Dahl was used a final time. This course was thus almost the same as the 2015 course; one minor difference was that some of the lecture material on MapReduce and Hadoop was revised to explore the newer, better-performing *Apache Spark* [9] technology.

C. 2018: CSinParallel, Iteration 3

In 2018, the CSinParallel group applied for and received an NSF-DUE Level 2 grant to fund a project titled "*Seeing, Hearing, and Touching Parallel Computing*." Building on the previous NSF-TUES 2 work, the author continues to work on TSGL to explore new ways to help students to *see* parallel algorithmic behavior. However, since visualization is of limited benefit for students with visual impairments, the author is also exploring audialization—the sonic equivalent of visualization—in order for students to *hear* parallel algorithmic behavior. Others in the CSinParallel group are exploring ways to let students who are tactile learners experience parallel algorithmic behavior by *touch*, including "unplugged" parallel computing activities, the tactile construction of mini-Beowulf clusters using single board computers, and similar touch-related tactics.

D. 2019: High Performance Computing, Iteration 10

In Fall 2019, the author offered his HPC course again, with 33 students enrolling. This iteration was similar to the 2017 offering, aside from two changes.

One change was that Borg was integrated into the course, replacing Dahl. At that time, the CS lab machines had 3.6-GHz Intel i7 CPUs compared to the 3.2-GHz Xeon Gold CPUs in Borg, but with 33 students taking the course (i.e., bogging down the lab), Borg's scalability and faster network usually gave it the edge, especially on communication-intensive computations.

The other change stemmed from two 2017 developments: (i) Intel's announcement that it was halting further work on its "Knights Hill" Xeon Phi family, and (ii) Adapteva's halting its production of the Parallela SBC. Together, these cast doubt on the future of coprocessor-based parallel computing. As a result, the coprocessor content and Xeon Phi materials from the 2017 course were replaced by expanded coverage of GPU computing and CUDA, to provide students with additional hands-on experience that delved deeper into CUDA.

VII. Discussion and Conclusions

The parallel hardware changes over the past 20+ years have been breathtaking. When the author began his journey in 1996, a Cray T3D-256 supercomputer costing millions of dollars could achieve 25.3 Gflops. In 1997, a sub-$2500 Microwulf could achieve 26.25 Gflops. Today, one graphics card can outperform either one; multicore processors are everywhere and manycore processors offer dozens of cores on one chip. GPUs provide hundreds or thousands of cores, depending on one's budget.

In the high-performance world, heterogeneous clusters of nodes with manycore CPUs and GPUs are common; software developers must create equally heterogeneous solutions. Challenges abound, however: Moore's Law appears to be reaching its limits, and energy consumption is a grand challenge that must be overcome if we are to achieve exascale computing.

The changes in parallel software systems have been less dramatic but are still substantial. The MPI-1.0 standard was finalized in 1994; MPI-4.0 is currently in development. The OpenMP-1.0 standard was finalized over 1997 and 1998; OpenMP-5.0 was finalized in 2018. OpenCL appears to be losing its battle with CUDA for use of the GPU. OpenACC and OpenMP both seek to make low-level CUDA unnecessary, but so far offer poor efficiency. For HPC, the MPI+CUDA, MPI+OpenMP, or MPI+OpenMP+CUDA software hybrids seem to be the top choices.

On the language front: Chapel, Julia, Scala, and others seek to supplant MPI and OpenMP as high performance computing languages, but it is not clear if any of them will succeed. C++11, Go, Rust, and others seek to provide improved multithreading models for shared-memory multiprocessors; given the ubiquity of multicore systems, these languages may well find niches.

As the preceding discussion illustrates, there is a great deal of foment in the parallel computing world, and there is no end in sight. This turbulence makes it challenging to teach parallel and distributed computing, as one can spend time and energy mastering a new parallel technology, only to have it suddenly become obsolete. Curated teaching-resource sites such as *CSinParallel.org* can help, but must be constantly maintained.

Over his many years of teaching parallel computing, some of the insights this author has gained include:

1. As parallel computing educators, our first job is to serve our students. This includes helping them acquire useful knowledge and skills, and not teaching them ephemera. For example, *parallel design patterns* represent a stable body of knowledge amidst the turbulence of parallel computing. As the best practices identified by professional parallel software developers over decades of experience, they are well worth mastering, and have excellent potential to remain useful for the foreseeable future.
2. "Cutting edge" technologies are often overhyped and may be ephemeral. Coprocessors are seemingly an example.
3. Official standards (e.g., MPI, OpenMP) are wonderful. If a technology does not have a standard, it may be wise to delay incorporating it into one's teaching, or the time one invests may be wasted. (CUDA may be an exception.)
4. Students have amazing energy, creativity, and enthusiasm. Enabling and channeling their interests sometimes opens interesting doorways, as MBH'99 did over 20 years ago.
5. Summer workshops can be excellent faculty development opportunities. Such a workshop started this author's PDC journey; one never knows where a workshop may lead!

The author hopes the information in this paper has been interesting and useful to the reader, as we collectively look to the future of parallel and distributed computing education.

Acknowledgments

The author gratefully acknowledges the support of the National Science Foundation, specifically DUE-IUSE #1822486, MRI#1726260, DUE#1225739, MRI#0722819, and MRI#0079739; and the Calvin University Science Division.

References

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Computer Science Curricula 2013*. ACM Press and IEEE Computer Society Press, 2013.

[2] J. Adams, W. D. Laverell and M. Ryken*, "MBH'99: A Beowulf Cluster Capstone Project", *14th Annual Midwest Computer Conference*, Whitewater, WI, March 2000.

[3] J. Adams and D. Vos, "Small College Supercomputing: Building a Beowulf Cluster at a Comprehensive College", *33rd SIGCSE Technical Symposium on Computer Science Education*, Covington, KY, February 2002, pp. 411- 415.

[4] J. Adams and T. Brom, "Microwulf: A Beowulf Cluster For Every Desk", *39th SIGCSE Technical Symposium on Computer Science Education*, March 2008, pp. 121-125.

[5] J. Adams, K. Hoogeboom, J. Walz, "A Cluster for CS Education in the Multicore Era", *42nd SIGCSE Technical Symposium on Computer Science Education*, March 2011, p. 27-31.

[6] J. Adams, "Patternlets: A Teaching Tool for Introducing Students to Parallel Design Patterns", *Journal of Parallel and Distributed Computing*, (105) July 2017, p. 31-41.

[7] J. Adams, et al, "TSGL: A Tool for Visualizing Multithreaded Behavior", *Journal of Parallel and Distributed Computing*, Volume 118, Issue P1, Aug 2018, pp. 233-246.

[8] Apache Software Foundation, *Apache Hadoop*. Online: https://hadoop.apache.org. Accessed 2020-02-09.

[9] Apache Software Foundation, *Apache Spark: lightning fast unified analytics engine*. Online: https://spark.apache.org. Accessed 2020-02-09.

[10] D. J. Becker, J. Salmon, D. F. Sevarese, and T. Sterling, *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters,* MIT Press, 1999.

[11] T. Braunl, Parallaxis-III: A Structured Data-Parallel Programming Language, *Algorithms and Architectures for Parallel Processing* (ICA3P-95), May, 1995, p. 43-52.

[12] R. Brown, E. Shoop. *CSinParallel: Parallel Computing in the Computer Science Curriculum*. Online: https://csinparallel.org/. Accessed 2020-02-09.

[13] R. Brown, et al, "Strategies for Preparing Computer Science Students for the Multicore World", *Proceedings of the 2010 ITiCSE Working Group Reports*, Ankara, Turkey, pp. 97-115.

[14] J. Hill, M. Warren, and P. Goda, "I'm Not Going to Pay a Lot for This Supercomputer!", *Linux Journal*, January 1998.

[15] T. Mattson, B. Sanders, and B. Massengill, *Patterns for Parallel Programming*, Addison-Wesley, 2004.

[16] *MPICH: High Performance Portable MPI*. Online: https://www.mpich.org. Accessed 2020-02-09.

[17] NSF/IEEE TCPP Curriculum Initiative, *Early Adopter Program*. Online: http://tcpp.cs.gsu.edu/curriculum/?q=the-early-adopter-program.html. Accessed 2020-02-09.

[18] *OpenMP: Enabling HPC Since 1997*. Online: https://www.openmp.org. Accessed 2020-02-09.

[19] P. Pacheco, *Parallel Programming with MPI*, Morgan-Kaufmann, 1996.

[20] P. Pacheco, *An Introduction to Parallel Programming*, Morgan-Kaufmann, 2011.

[21] C. Peck, et al. *LittleFe: Parallel and Cluster Computing On The Move*. Online, http://littlefe.net. Accessed 2020-02-09.

[22] S. Prasad, et al, *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates*. Online: http://tcpp.cs.gsu.edu/curriculum/?q=system/files/NSF-TCPP-curriculum-version1.pdf. Accessed 2020-02-09.

[23] M. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2003.

[24] Shodor Education Foundation, *National Computational Science Institute*. Online: http://computationalscience.org. Accessed 2020-02-09.