Brief Announcement: Open Cilk

Tao B. Schardl
MIT CSAIL
Cambridge, MA, USA
neboat@mit.edu

I-Ting Angelina Lee
Washington University in St. Louis
St. Louis, MO, USA
angelee@wustl.edu

Charles E. Leiserson
MIT CSAIL
Cambridge, MA, USA
cel@mit.edu

ABSTRACT

Open Cilk is a new open-source platform to support Cilk multithreaded programming, especially for researchers and teachers. Open Cilk aims to provide a full-featured implementation of Cilk that is easy to modify and extend. Based on the award-winning Tapir/LLVM compiler, Open Cilk will provide a streamlined runtime system and feature comprehensive static instrumentation for dynamic-analysis tools. As a community-infrastructure project, Open Cilk encourages contributions from researchers in the areas of languages, compilers, runtime systems, tools, libraries, and benchmarks.

CCS CONCEPTS

• **Theory of computation → Parallel algorithms**; • **Computing methodologies → Parallel programming languages**; • **Software and its engineering → Parallel programming languages**; **Compilers**; **Runtime environments**; **Software testing and debugging**;

KEYWORDS

Cilk; multicore programming; parallel algorithms; parallel languages; productivity tools

ACM Reference Format:
Tao B. Schardl, I-Ting Angelina Lee, and Charles E. Leiserson. 2018. Brief Announcement: Open Cilk. In *SPAA '18: 30th ACM Symposium on Parallelism in Algorithms and Architectures, July 16–18, 2018, Vienna, Austria.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3210377.3210658

1. Introduction

For years Cilk [6, 11, 19, 33, 42] has offered software developers a high-performance multithreaded programming environment while providing researchers with a platform for investigating a wide range of topics in multicore computing. When Cilk research began at MIT in 1994, it was conceived [5, 6, 10, 11, 26, 41] as a parallel-programming extension to the C programming language [28]. In 2006, at the advent of the multicore revolution, MIT spun off Cilk into the start-up Cilk Arts, Inc., which produced the open-source Cilk++ platform [33] as an extension to C++ [51]. Intel Corporation acquired Cilk Arts in 2009, added vectorization directives, and rechristened Cilk++ as Cilk Plus [19, 42], making it available commercially in the ICC compiler and in open-source implementations for the GCC [2, 13] and LLVM [21] compilers.

But Intel was unable to maintain a strong internal Cilk-development group. By 2013, attrition among Intel's Cilk team caused Intel to cease active development of the Cilk Plus product and focus solely on maintenance. In August 2017, Intel announced [23] that it would no longer even maintain Cilk Plus. Since Intel also maintained the open-source GCC implementation, GCC announced it was dropping its support for Cilk as well [14].

Cilk Hub and Open Cilk

In January 2018, a group of us decided to address the flagging support for Cilk by creating a community-driven organization called ***Cilk Hub*** (http://cilkhub.org) to maintain and develop the Cilk technology. Cilk Hub provides a technically superior ***Open Cilk*** system to serve the needs of software developers, multicore and parallel-computing researchers, and teachers of parallel computing. Open Cilk features a new compiler based on Tapir/LLVM [45], comprehensive static instrumentation, and a suite of productivity tools. Cilk Hub provides external documentation of the Open Cilk system, offers workshops and tutorials, and coordinates annual meetings. Cilk Hub has formed an Advisory Board of leading researchers in multicore computing to set directions and priorities.

Cilk Hub will be active in developing Open Cilk as an open-source community project. Project enhancements include a lean-and-mean runtime system, new linguistics, new productivity tools, and libraries and benchmarks. Cilk Hub will manage integration of community-proposed enhancements to Open Cilk to ensure that it remains an efficient and malleable platform for multicore-programming research. Cilk Hub will produce a cloud VM that will allow researchers, teachers, and programmers to use Open Cilk technology — including the compiler, runtime system, libraries, and tools — without elaborate installation.

2. The Open Cilk system

Open Cilk is a full-featured, open-source implementation of Cilk designed to run on Linux and other Unix-like systems. Open Cilk provides a platform onto which researchers and teachers currently using Cilk Plus can migrate their ongoing research and educational materials with minimal effort. Open Cilk also serves as a vehicle for future research on Cilk programming and parallel-language technology.

Language

Open Cilk is fully compatible with Cilk Plus, minus its vector notation. Open Cilk also enhances Cilk Plus by, for example, providing support for spawning statement blocks, instead of just functions. Going forward, we plan to improve the Cilk language based on input

This research was supported in part by NSF Grants 1314547, 1527692, 1533644, and 1733873.

SPAA '18, July 16–18, 2018, Vienna, Austria

ACM ISBN 978-1-4503-5799-9/18/07.
https://doi.org/10.1145/3210377.3210658

from the community. As we develop Cilk, we will strive to maintain its simplicity and ensure that all new language features work seamlessly with existing features. Some of the language features to be considered include simple linguistics for reducer hyperobjects [12] and a more efficient implementation of reducer hyperobjects in the Open Cilk runtime system [31], as well as linguistic and runtime support for on-the-fly pipeline parallelism [32], speculative parallelism [8], async-finish parallelism as in X10 [7], amorphous parallelism [39], and vectorization and data parallelism [17].

Compiler

Open Cilk's compiler is based on the Tapir/LLVM compiler [45] and its Cilk/Clang front end. Tapir/LLVM embeds fork-join parallelism into LLVM's intermediate representation (IR). The resulting simple extension to LLVM's IR, called ***Tapir***, allows compiler optimizations for serial code to operate on parallel code with minimal changes. Integrating Tapir into LLVM required only adding or modifying about 6000 source lines of code, compared to LLVM's 4-million-line codebase. These few changes enable the Open Cilk compiler to produce more efficient parallel executables than other compilers for parallel languages.

Runtime system

Although the Open Cilk system currently uses the Cilk Plus runtime system, we plan to develop a new open-source work-stealing runtime system for Open Cilk to suit the needs of researchers studying parallel runtime systems. The Open Cilk runtime system will implement the core runtime functionality of the existing Cilk Plus runtime system, but feature a smaller, simpler codebase and extensive documentation. We plan to enhance the runtime system with new capabilities. Among the considerations are support for long-latency operations such as file I/O [37] and interoperability with the system environment, including debuggers [1] and Pthreads [18].

Comprehensive static instrumentation

To support productivity tools for parallelism, the Open Cilk compiler provides comprehensive compiler instrumentation based on the CSI framework [44]. CSI provides a simple application program interface (API) consisting of hooks (functions) that are automatically called at salient points throughout the program-under-test, such as function calls, basic blocks, and memory operations. A tool writer can implement a "CSI-tool" as a library, without modifying the compiler, by simply defining the semantics of the various API hooks. Undefined hooks are elided, minimizing the overhead of the CSI-tool. Whereas the original CSI framework operated on serial programs, the Open Cilk compiler extends CSI to instrument Tapir, giving tool writers additional hooks that correspond to `cilk_spawn`, `cilk_sync`, and other important points in parallel programs.

Productivity tools

The Open Cilk system includes two open-source CSI-tools: the Cilksan nondeterminacy detector, and the Cilkscale scalability analyzer. Cilksan implements the core functionality of Intel's closed-source Cilkscreen tool [20], and Cilkscale implements the core functionality of Intel's closed-source Cilkview tool [16]. We plan to enhance Open Cilk's suite of productivity tools with performance and functionality enhancements [38, 40, 54, 56] to existing tools, as well as with implementations of additional productivity tools, such as the Cilkprof scalability profiler [43] and a tool to support record and replay [30, 55].

Libraries and benchmarks

Open Cilk will incorporate an expansive collection of libraries and benchmark programs. We plan to build this collection by drawing from the extensive body of community-developed parallel libraries and benchmarks [3, 4, 9, 11, 15, 22, 24, 25, 27, 29, 34–36, 46–50, 52, 53, 57].

3. The architecture of Open Cilk

Open Cilk is architected to provide a robust platform for professional application development and for next-generation multicore-computing research and education. The design and implementation of Open Cilk exhibits five central software qualities: compatibility, open source, componentization, integration, and reliability.

Compatibility

Open Cilk provides support for modern C++ features and existing Cilk Plus applications and libraries. To ensure that Cilk can parallelize large applications, Open Cilk supports crucial features, such as using C++ exceptions in Cilk programs. Open Cilk provides support for existing Cilk Plus libraries, including reducer hyperobjects [12] and deterministic parallel random-number generators [35].

Open source

The entire Open Cilk implementation is open source. Open Cilk thus allows researchers to develop enhancements to all parts of the system: compiler, runtime system, libraries, and productivity tools. Cilk Hub manages the process of integrating community-proposed innovations into Open Cilk so that applications researchers and professional developers can exploit the enhancements.

Componentization

The Open Cilk system is divided into distinct software components with well-defined interfaces. This design allows researchers to experiment with alternative implementations of different individual components.

Integration

The components of Open Cilk are integrated to ensure that they all work together correctly. Cilk Hub aims for consistency and compatibility among all components with each system release. Cilk Hub will produce a cloud VM so that researchers, teachers, and programmers can use Open Cilk technology in the cloud without elaborate installation, confident that all components will interoperate.

Reliability

To ensure that Open Cilk releases are stable, fast, and free of serious bugs, Open Cilk includes an extensive suite of unit tests, regression tests, and benchmarks, all of which it will continue to develop. We also plan to develop a cloud-based testing infrastructure for continuous testing and integration of changes to Open Cilk.

4. Conclusion

Cilk Hub welcomes contributions to Open Cilk. To stay abreast of further developments, please visit `http://cilkhub.org`.

REFERENCES

[1] P. Alves, J. Brobecker, D. Evans, T. Tromey, and E. Zaretskii. GDB: The GNU project debugger. http://www.gnu.org/software/gdb/, Oct. 2014. Viewed Nov 3, 2014.

[2] B. Ayer. Intel Cilk Plus is now available in open-source and for GCC 4.7! http://www.cilkplus.org, 2011. The source code for the compiler and its associated runtime is available at http://gcc.gnu.org/svn/gcc/branches/cilkplus.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, pages 72–81, 2008.

[4] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and J. Shun. Internally deterministic parallel algorithms can be fast. In *PPoPP*, pages 181–192, 2012.

[5] R. D. Blumofe. *Executing Multithreaded Programs Efficiently*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, Sept. 1995. Available as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-677.

[6] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, 1996.

[7] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: An object-oriented approach to non-uniform cluster computing. In *OOPSLA*, 2005.

[8] D. Dailey and C. E. Leiserson. Using Cilk to write multiprocessor chess programs. *The Journal of the International Computer Chess Association*, pages 25–52, 2002.

[9] L. Dhulipala, G. Blelloch, and J. Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *SPAA*, pages 293–304, 2017. ISBN 978-1-4503-4593-4.

[10] M. Frigo. A fast Fourier transform compiler. *ACM SIGPLAN Notices*, 34(5):169–180, May 1999.

[11] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In *PLDI*, pages 212–223, 1998.

[12] M. Frigo, P. Halpern, C. E. Leiserson, and S. Lewin-Berlin. Reducers and other Cilk++ hyperobjects. In *SPAA*, pages 79–90, 2009.

[13] GCC Team. GCC 4.9 release series changes, new features, and fixes. Available at https://gcc.gnu.org/gcc-4.9/changes.html, 2014.

[14] GGC Team. GCC 7 release series: Changes, new features, and fixes. Available at https://gcc.gnu.org/gcc-7/changes.html, 2017.

[15] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson. Ordering heuristics for parallel graph coloring. In *SPAA*, pages 166–177, 2014.

[16] Y. He, C. E. Leiserson, and W. M. Leiserson. The Cilkview scalability analyzer. In *SPAA*, pages 145–156, 2010.

[17] W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *Commun. ACM*, 29(12): 1170–1183, Dec. 1986.

[18] Institute of Electrical and Electronic Engineers. Information technology — Portable Operating System Interface (POSIX) — Part 1: System application program interface (API) [C language]. IEEE Standard 1003.1, 1996 Edition, 1996.

[19] Intel Corporation. *Intel Cilk Plus Language Specification*, 2010. Document Number: 324396-001US. Available from http://software.intel.com/sites/products/cilk-plus/cilk_plus_language_specification.pdf.

[20] Intel Corporation. Intel Cilk Plus software development kit. Available from http://software.intel.com/en-us/articles/intel-cilk-plus-software-development-kit/, Dec. 2011.

[21] Intel Corporation. Cilk Plus/LLVM: an implementation of the Intel Cilk Plus C/C++ language extensions in LLVM. Available from http://cilkplus.github.io/, 2016.

[22] Intel Corporation. Intel Cilk Plus samples. Available from https://software.intel.com/en-us/code-samples/intel-compiler/intel-compiler-features/intelcilkplus, 2016.

[23] Intel Corporation. Intel Cilk Plus. Available at https://www.cilkplus.org, sep 2017.

[24] Intel Corporation. *Reference Manual for Intel Math Kernel Library 2018 — C*, 2017. URL https://software.intel.com/en-us/mkl-reference-manual-for-c.

[25] S. Itzhaky, R. Singh, A. Solar-Lezama, K. Yessenov, Y. Lu, C. Leiserson, and R. Chowdhury. Deriving divide-and-conquer dynamic programming algorithms using solver-aided transformations. In *OOPSLA*, pages 145–164, 2016. ISBN 978-1-4503-4444-9.

[26] C. F. Joerg. *The Cilk System for Parallel Multithreaded Computing*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, Jan. 1996. Available as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-701.

[27] T. Kaler, W. Hasenplaugh, T. B. Schardl, and C. E. Leiserson. Executing dynamic data-graph computations deterministically using chromatic scheduling. *Transactions on Parallel Computing*, 3(1):2:1–2:31, 2016. ISSN 2329-4949.

[28] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Inc., second edition, 1988.

[29] M. Kulkarni, M. Burtscher, C. Cascaval, and K. Pingali. Lonestar: A suite of parallel irregular programs. In *ISPASS*, pages 65–76, 2009.

[30] T. J. LeBlanc and J. M. Mellor-Crummey. Debugging parallel programs with Instant Replay. *IEEE Transactions on Computers*, C-36(4):471–482, Apr. 1987.

[31] I.-T. A. Lee, A. Shafi, and C. E. Leiserson. Memory-mapping support for reducer hyperobjects. In *SPAA*, pages 287–297, 2012.

[32] I.-T. A. Lee, C. E. Leiserson, T. B. Schardl, Z. Zhang, and J. Sukha. On-the-fly pipeline parallelism. *ACM TOPC*, 2(3):17:1–17:42, 2015.

[33] C. E. Leiserson. The Cilk++ concurrency platform. *Journal of Supercomputing*, 51(3):244–257, 2010.

[34] C. E. Leiserson and T. B. Schardl. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In *SPAA*, pages 303–314. ACM, June 2010.

[35] C. E. Leiserson, T. B. Schardl, and J. Sukha. Deterministic parallel random-number generation for dynamic-multithreading platforms. In *PPoPP*, 2012.

[36] M. M. Maza and Y. Xie. Balanced dense polynomial multiplication on multi-cores. *ACM Communications on Computer Algebra*, 43(3/4):85–87, June 2010. ISSN 1932-2240.

[37] S. K. Muller and U. A. Acar. Latency-hiding work stealing: Scheduling interacting parallel computations with work stealing. In *SPAA*, pages 71–82. ACM, 2016. ISBN 978-1-4503-4210-0.

[38] D. Oktay and S. Kamali, May 2017. Private communication.

[39] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Méndez-Lojo, D. Prountzos, and X. Sui. The Tao of parallelism in algorithms. In *PLDI*, pages 12–25, 2011.

[40] R. Raman, J. Zhao, V. Sarkar, M. Vechev, and E. Yahav. Efficient data race detection for async-finish parallelism. *Formal Methods in Systems Design*, 41(3):321–347, Dec. 2012.

[41] K. H. Randall. *Cilk: Efficient Multithreaded Computing*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1998.

[42] A. D. Robison and C. E. Leiserson. Cilk Plus. In P. Balaji, editor, *Programming Models for Parallel Computing*, chapter 13, pages 323–352. The MIT Press, Cambridge, MA, 2015.

[43] T. B. Schardl, B. C. Kuszmaul, I.-T. A. Lee, W. M. Leiserson, and C. E. Leiserson. The Cilkprof scalability profiler. In *SPAA*, pages 89–100, 2015.

[44] T. B. Schardl, T. Denniston, D. Doucet, B. C. Kuszmaul, I.-T. A. Lee, and C. E. Leiserson. The CSI framework for compiler-inserted program instrumentation. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(2):43:1–43:25, Dec. 2017. ISSN 2476-1249.

[45] T. B. Schardl, W. S. Moses, and C. E. Leiserson. Tapir: Embedding fork-join parallelism into LLVM's intermediate representation. In *PPoPP*, pages 249–265, 2017. ISBN 978-1-4503-4493-7.

[46] J. Shun. *Shared-Memory Parallelism Can be Simple, Fast, and Scalable*. Morgan & Claypool, 2017.

[47] J. Shun and G. E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. In *PPoPP*, pages 135–146, 2013.

[48] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, A. Kyrola, H. V. Simhadri, and K. Tangwongsan. Brief announcement: the Problem Based Benchmark Suite. In *SPAA*, pages 68–70, 2012.

[49] J. Shun, L. Dhulipala, and G. E. Blelloch. Smaller and faster: Parallel processing of compressed graphs with Ligra+. In *DCC*, pages 403–412, 2015.

[50] G. L. Steele Jr., D. Lea, and C. H. Flood. Fast splittable pseudorandom number generators. In *OOPSLA*, pages 453–472, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2585-1.

[51] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Boston, MA, third edition, 2000.

[52] Y. Tang, R. You, H. Kan, J. J. Tithi, P. Ganapathi, and R. A. Chowdhury. Cache-oblivious wavefront: Improving parallelism of recursive dynamic programming algorithms without losing cache-efficiency. In *PPoPP*, pages 205–214, 2015. ISBN 978-1-4503-3205-7.

[53] S. Toledo. TAUCS: A library of sparse linear solvers. Available on the Web from http://www.cs.tau.ac.il/~stoledo/taucs/, 2003.

[54] R. Utterback, K. Agrawal, J. T. Fineman, and I.-T. A. Lee. Provably good and practically efficient parallel race detection for fork-join programs. In *SPAA*, pages 83–94, 2016.

[55] R. Utterback, K. Agrawal, I.-T. A. Lee, and M. Kulkarni. Processor-oblivious record and replay. In *PPoPP*, pages 145–161, 2017.

[56] Y. Xu, I.-T. A. Lee, and K. Agrawal. Efficient parallel determinacy race detection for two-dimensional dags. In *PPoPP*, 2018. To appear.

[57] B. Zhang, J. Huang, N. P. Pitsianis, and X. Sun. RECFMM: recursive parallelization of the adaptive fast multipole method for Coulomb and screened Coulomb interactions. *Communications in Computational Physics*, 20(2):534–550, 2016.