Semantically Sequential, Parallel Execution of Programs on Multiprocessors

Gagan Gupta
Department of Computer Sciences, University of Wisconsin-Madison
Email: gagang@cs.wisc.edu. **Address**: 6083 Viroqua Drive, #D, Fitchburg, WI 53719.
Advisor: Gurindar S. Sohi; **ACM Membership #** 3813893.
Category: Graduate.

A program's execution consistent with the user-specified program order simplifies many aspects of computer design, e.g., programming and resource management. This has greatly benefited uniprocessor system adoption. Parallel systems do not enforce an ordered program execution, complicating system design. Hence, I ask, can parallel execution of programs on multiprocessors be made program-order consistent? I explore one approach to do so, and its impact on performance and system design. Preliminary work shows that the approach is promising.

Emerging Challenges

Due to evolving technology trends, a program's automatic performance scaling or its completion is no longer assured. Today, hardware comprises multiple computational resources, is more unreliable, and is energy constrained. The onus is on programmers to develop parallel programs and achieve their efficient execution on such platforms.

To conserve energy and scale performance further, new techniques are emerging, such as hardware energy management [21, 22, 41], dynamic resource management [1, 8], and approximate computing [4, 28, 38]. Unfortunately these techniques give rise to *discretionary exceptions*: user-permitted events that can interrupt a program's execution. Such exceptions can arise from computation scheduling, hardware emergencies and erroneous computations.

These trends will likely make parallel programs and frequent exceptions during their execution the norm in the future. Parallel programming, "a gigantic challenge facing the computer science community" [32], is already onerous. Handling frequent exceptions complicates it further.

Current Approaches

Designers today program multiprocessors using the decades-old nondeterministic parallel programming model, which was originally developed for supercomputers. Although the model's shortcomings are well documented [27, 40], practitioners believe that it maximizes parallelism [10, 24, 31]. Hence, to surmount the emerging parallel programming challenges, most current proposals embrace nondeterminism, but mitigate its shortcomings.

Some proposals simplify nondeterministic program expression, without addressing nondeterminism itself [10, 14, 23, 24, 34, 36]. Others overcome nondeterminism by making the execution deterministic at run-time, but penalize performance, and may hamper portability [3, 5, 6, 15, 16, 25, 29, 31, 33]. A few past proposals have explored deterministic parallel programs, but they limit the amount of exposed parallelism [2, 7, 9, 18, 35, 37].

To handle exceptions in nondeterministic parallel programs, designers use checkpoint-and-recovery (CPR) [11, 12, 17, 30]. My analysis shows that due to the overheads arising from nondeterminism, CPR may not scale to handle the expected frequent discretionary exceptions [20].

In summary, present approaches cure the symptoms, but do not cure nondeterminism itself. Hence an efficient and practical approach to parallel programming remains elusive.

Proposed Research

I seek a fresh and holistic view of the parallel programming challenges. I draw inspiration from successful concepts over the history of computing, specifically, sequential programs and superscalar processors. I envision that developers will design parallel algorithms, but express them as *statically sequential* programs. A parallel system will exploit the algorithmic parallelism using *globally precise-restartable*, dataflow execution of the tasks therein, while maintaining the program's sequential semantics, analogous to precise-interruptible, instruction-level parallel execution in superscalar processors. Sequential semantics simplifies programming and enables efficient exception handling.

I explore the approach's impact on performance and system design. Initial results, followed by future directions are presented here.

Programming and Execution Model. As a first step, I have developed a model based on statically-sequential, object-oriented C++ imperative programs [19]. The model leverages programmers' domain expertise to develop parallel algorithms but eases the burden of explicitly orchestrating, and hence, reasoning about their correct parallel execution. The algorithm's parallelism is exploited using the dataflow principle since it naturally exposes concurrency.

Analogous to superscalar processors, a runtime implementation of the model sequences through the program, and attempts to execute user-designated tasks concurrently. It dynamically establishes data dependences between the tasks. Independent tasks are executed concurrently, while depen-

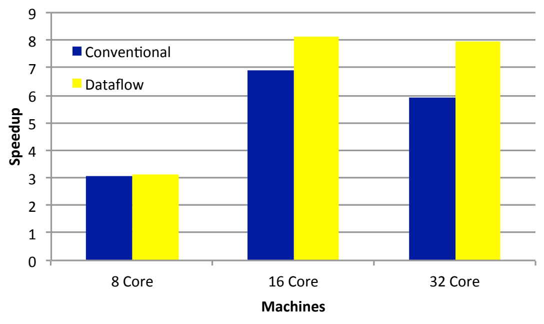

Figure 1. Harmonic mean of speedups achieved for standard parallel programs on an 8-core, a 16-core and a 32-core machine using conventional (Pthreads) and dataflow model. Dataflow achieves similar or better speedups.

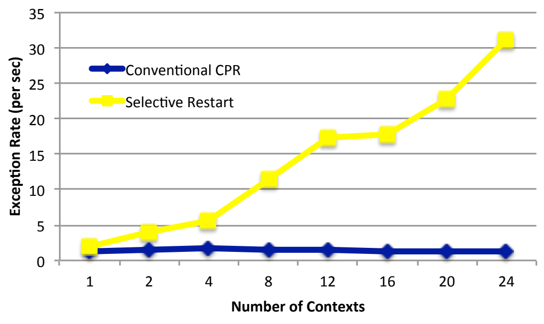

Figure 2. Exception resiliency of conventional CPR and selective restart for the nondeterministic implementation of the Pbzip2 program running on a 24-context machine [20]. Selective restart can handle more exceptions as the system size grows, whereas conventional CPR does not.

dent tasks are serialized. The resulting execution is naturally deterministic. Despite the sequential semantics, the model exploits the available parallelism. Dataflow execution can look deep into the program for parallelism, which conventional user-orchestrated execution cannot without considerable efforts. On popular parallel benchmarks the runtime's performance is comparable to the conventional method (Figure 1).

Exception Handling. Implicit ordering of a program's task can also simplify exception handling.

Recently I have shown that if a parallel programs execution is made deterministic, the program can be made *globally precise-restartable*, analogous to precise-interruptible sequential programs [20]. This approach is scalable, unlike conventional CPR. I am extending this work to the above execution model, which is already deterministic.

Briefly, the runtime system tracks (i) the order of the program's currently executing tasks, (ii) the objects they may modify and (iii) the state of those objects before they are modified. When a task excepts, objects modified by it and by those "younger" to it can be restored to their pre-modified state, causing the program state to reflect precise, ordered execution up to the exception. The program may restart using this state, and is hence *globally precise-restartable*.

Dataflow execution can be exploited to selectively re-execute only the excepted task, without impacting the rest of the program, since only independent computations execute concurrently. This selective restart makes the approach scalable (Figure 2), making it well-suited for the highly exception-prone future systems. I expect similar benefits for ordered programs. More signficantly, it can potentially enable new capabilities, as described next.

Future Work

Although the approach shows promise, it is unclear how well it performs for programs with irregular parallelism. I propose to study the model's applicability to programs from the STAMP [13] and Lonestar [26] suites. Initial analysis shows that computations in these programs spend time identifying the data to be computed. This can hamper dataflow execution due to unknown depdendences, limiting the parallelism if sequential semantics is to be maintained. To overcome this limitation, I propose to execute computations speculatively. If the sequential semantics is violated, it can be treated as an exception and handled using global precise-restartability. This approach is similar to misspeculation handling in out-of-order superscalar processors [39]. My work will study its utility in parallel systems.

In addition to speculation, I propose to apply precise restart to fault tolerance in ordered parallel programs. This study will explore the benefits of ordered execution as compared with nondeterminsitic and deterministic execution of conventional parallel programs.

Conclusion

I have explored program-order consistent execution of a parallel program on multiprocessors. Doing so imparts sequential semantics to the execution, without sacrifycing performance. The approach potentially simplifies several aspects of parallel system design, such as programming, resource management, fault tolerance, security, etc. I am exploring ways to broaden the approach's applicability to a wider range of programs and applications.

If successful, I believe that the approach will take a significant stride in meeting the grand challenges of parallel computing.

References

[1] Amazon EC2 spot instances. 2009.

[2] M. D. Allen, S. Sridharan, and G. S. Sohi. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '09, pages 85–96, New York, NY, USA, 2009. ACM.

[3] A. Aviram, S.-C. Weng, S. Hu, and B. Ford. Efficient system-enforced deterministic parallelism. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[4] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM.

[5] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman. Coredet: A compiler and runtime system for deterministic multithreaded execution. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 53–64, New York, NY, USA, 2010. ACM.

[6] T. Bergan, N. Hunt, L. Ceze, and S. D. Gribble. Deterministic process groups in dos. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[7] E. D. Berger, T. Yang, T. Liu, and G. Novark. Grace: Safe multithreaded programming for c/c++. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 81–96, New York, NY, USA, 2009. ACM.

[8] F. Blagojevic, C. Iancu, K. Yelick, M. Curtis-Maury, D. S. Nikolopoulos, and B. Rose. Scheduling dynamic parallelism on accelerators. In *Proceedings of the 6th ACM Conference on Computing Frontiers*, CF '09, pages 161–170, New York, NY, USA, 2009. ACM.

[9] R. L. Bocchino, Jr., V. S. Adve, D. Dig, S. V. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian. A type and effect system for deterministic parallel java. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 97–116, New York, NY, USA, 2009. ACM.

[10] R. L. Bocchino, Jr., S. Heumann, N. Honarmand, S. V. Adve, V. S. Adve, A. Welc, and T. Shpeisman. Safe nondeterminism in a deterministic-by-default parallel language. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 535–548, New York, NY, USA, 2011. ACM.

[11] G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill. Recent advances in checkpoint/recovery systems. In *IPDPS*. IEEE, 2006.

[12] G. Bronevetsky, D. Marques, K. Pingali, P. Szwed, and M. Schulz. Application-level checkpointing for shared memory programs. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, pages 235–247, New York, NY, USA, 2004. ACM.

[13] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization*, September 2008.

[14] V. Cavé, J. Zhao, J. Shirako, and V. Sarkar. Habanero-java: The new adventures of old x10. In *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java*, PPPJ '11, pages 51–61, New York, NY, USA, 2011. ACM.

[15] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: Deterministic shared memory multiprocessing. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 85–96, New York, NY, USA, 2009. ACM.

[16] J. Devietti, J. Nelson, T. Bergan, L. Ceze, and D. Grossman. RCDC: A relaxed consistency deterministic computer. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 67–78, New York, NY, USA, 2011. ACM.

[17] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, Sept. 2002.

[18] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, pages 212–223, New York, NY, USA, 1998. ACM.

[19] G. Gupta and G. S. Sohi. Dataflow execution of sequential imperative programs on multicore architectures. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 59–70, New York, NY, USA, 2011. ACM.

[20] G. Gupta, S. Sridharan, and G. S. Sohi. Globally precise-restartable execution of parallel programs. To appear in PLDI, 2014.

[21] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. M. Brooks. DeCoR: A delayed commit and rollback mechanism for handling inductive noise in processors. In *HPCA*, pages 381–392. IEEE Computer Society, 2008.

[22] M. S. Gupta, J. A. Rivers, P. Bose, G.-Y. Wei, and D. Brooks. Tribeca: Design for pvt variations with local recovery and fine-grained adaptation. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 435–446, New York, NY, USA, 2009. ACM.

[23] T. Harris, J. Larus, and R. Rajwar. Transactional memory, 2nd edition. *Synthesis Lectures on Computer Architecture*, 5(1):1–263, 2010.

[24] S. T. Heumann, V. S. Adve, and S. Wang. The tasks with effects model for safe concurrency. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '13, pages 239–250, New York, NY, USA, 2013. ACM.

[25] D. R. Hower, P. Dudnik, M. D. Hill, and D. A. Wood. Calvin: Deterministic or not? free will to choose. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 333–334, Washington, DC, USA, 2011. IEEE Computer Society.

[26] M. Kulkarni, M. Burtscher, C. Casçaval, and K. Pingali. Lonestar: A suite of parallel irregular programs. In *ISPASS '09: IEEE International Symposium on Performance Analysis of Systems and Software*, 2009.

[27] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, May 2006.

[28] X. Li and D. Yeung. Exploiting application-level correctness for low-cost fault tolerance. *J. Instruction-Level Parallelism*, 10, 2008.

[29] T. Liu, C. Curtsinger, and E. D. Berger. Dthreads: Efficient deterministic multithreading. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 327–336, New York, NY, USA, 2011. ACM.

[30] D. Marques, G. Bronevetsky, R. Fernandes, K. Pingali, and P. Stodghil. Optimizing checkpoint sizes in the c3 system. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 10 - Volume 11*, IPDPS '05, pages 226.1–, Washington, DC, USA, 2005. IEEE Computer Society.

[31] D. Nguyen, A. Lenharth, and K. Pingali. Deterministic galois: On-demand, portable and parameterless. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 499–512, New York, NY, USA, 2014. ACM.

[32] C. O'Hanlon. A conversation with john hennessy and david patterson. *Queue*, 4(10):14–22, Dec. 2006.

[33] M. Olszewski, J. Ansel, and S. Amarasinghe. Kendo: Efficient deterministic multithreading in software. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 97–108, New York, NY, USA, 2009. ACM.

[34] 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 *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 12–25, 2011.

[35] J. M. Prez, R. M. Badia, and J. Labarta. A dependency-aware task-based programming environment for multi-core architectures. In *CLUSTER*, pages 142–151. IEEE, 2008.

[36] J. Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[37] M. C. Rinard and M. S. Lam. The design, implementation, and evaluation of jade. *ACM Trans. Program. Lang. Syst.*, 20(3):483–545, May 1998.

[38] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[39] J. E. Smith and G. S. Sohi. The microarchitecture of superscalar processors. *Proceedings of the IEEE*, 83(12):1609–1624, Dec. 1995.

[40] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, Sept. 2005.

[41] G. Yan, X. Liang, Y. Han, and X. Li. Leveraging the core-level complementary effects of PVT variations to reduce timing emergencies in multi-core processors. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 485–496, New York, NY, USA, 2010. ACM.