Reading List on Parallel Programming Languages

Guy E. Blelloch

July 29, 1994

This reading list gives pointers to papers on various parallel programming languages. It is meant to be representative and is certainly not complete. We have tried to place each language in a category where we think it fits best. Others are welcome to disagree with our choices. For each class we give an annotated list of readings. Three papers that give somewhat of an overview of parallel languages are [Sch93, Per92, BST89, Bal92].

1 Data parallelism

Data-parallel languages have been around for a long time, but interest in them has surged recently because of the availability of massively parallel hardware. These languages are attractive because parallelism is not expressed as a set of processes whose interactions are managed by the user, but rather as parallel operations on aggregate data structures. A classification of such languages is presented in [SB91].

1.1 Flat data-parallel languages

The aggregate data structures of a flat data-parallel language cannot contain members that are themselves aggregates.

1. An examination and comparison of the features found in a number of data-parallel languages [SB91].
2. C*. C extended with domains, poly and mono variable classes, and reduction operations [RS87].
3. Fortran 90. Triplet notation for array sections. Operations and intrinsic functions on array sections [MR92].
4. HPF. Similar to Fortran 90, but includes data layout specifications to help the compiler generate efficient code [Hig93].
5. Implementation of C* for MIMD machines [QH90].
6. APL: the oldest and most influential of data-parallel languages [Ive62].
7. UC: Another C based data-parallel language [BCK90].
8. Apply. Regular communication patterns, mappings for various architectures [HWW89]. Designed for image processing.

1.2 Nested data-parallel languages:

Nested data-parallel languages allow aggregate data structures whose members are also aggregates. This allows parallel evaluation of functions that themselves involve parallel evaluation.

1. NESL. A Functional language with sequences as collections [Ble93].
2. Proteus. Not strictly a data-parallel language, but much of the compiler work has been on a data-parallel subset [MNP+90].
3. CM-Lisp. Common Lisp augmented with nested key-ordered collections, and associated operations [SH86].
4. The Paralation model. A language-independent model of parallel computation. Nested key-ordered collections with notion of shape [Sab88].
5. Divicom. A language based on parallel divide-and-conquer [MH88].

2 Explicit communication models

In this class of languages, the user is completely responsible for the creation and management of processes and their interactions with one another. Languages differ in the organization of processes and communication channels (static, dynamic) and the communication schemes (synchronous, asynchronous).

2.1 Message passing

These languages represent processes that communicate through messages.

1. CSP. Seminal work on formulating parallelism as a network of sequential processes exchanging messages [Hoa78].
2. Occam. Real language based on Hoare's CSP [May83].
3. Ada tasking and rendezvous facilities [HP83].
4. A comprehensive survey of message passing languages, both parallel and distributed. This article has an exhaustive list of languages for distributed systems and further references [BST89].

2.2 Object-oriented languages

These languages represent a computation as a dynamic network of objects exchanging messages.

1. Survey of the ideas behind parallel object-oriented languages [Agh90] (for a deeper coverage, see [Agh91]).
2. Survey and comparison of most existing parallel object-oriented languages [WKH92].
3. Concurrent Smalltalk: methods, distributed data structures [HCD89].
4. Comprehensive treatment of the Actors paradigm [Agh86].
5. Concurrent Aggregates [CD90].

6. Cantor. An Actor-like language for programming fine-grained multi-computers [AS88].
7. Emerald. An object-based language for distributed applications. Abstract data-types supply communication interface to an object; allows process migration [BHJ+87].
8. COOL. Objects with futures [CGH90].
9. Orca. Shared objects using broadcasting [TKB92].

2.3 Coordination languages

These languages provide a framework for coordinating sequential computations, usually written in some other language.

1. A paper that motivates the coordination language approach to parallelism [FKT90].
2. Linda. A language-independent model based on tuple-space with a relatively small number of primitives. A book [GC90] describes the language in full, or a paper [CG89a] overviews the language and suggests three useful programming styles. A C version of Linda is available comercially.
3. Strand. A commercially available language [FT90].

3 Functional languages

Functional languages are referentially transparent and have no state. This feature has been exploited in various ways to gain parallelism.

3.1 Speculative computation

These languages try to gain parallelism by performing computations in parallel before their results are known to be required.

1. MultiLisp. Lisp with futures. A *future* is a promise to provide the value of a computation if needed; pointers to the future may be freely manipulated. Run-time system decides when futures should create new processes. An introduction to Multilisp [Hal85] and developments in MultiLisp, particularly the difficulties of implementing both futures and explicit continuations (`call/cc`) [Hal90].
2. Qlisp. Provides *qlet* and *qlambda* constructs for spawning parallel processes. User supplies predicates for determining whether these should create new processes [GG88].
3. Mul-T. Similar to Multilisp but code is compiled rather than interpreted; has improved task management/creation [KHM89].

3.2 Graph reduction

A functional program can be represented as a graph of combinators [Tur79]. Reduction of this graph has proven to lead to efficient schemes for sequential evaluation of lazy functional programs. Proponents claim that graph reduction is inherently parallel, but there has been little in the way of efficient parallel implementation.

1. MIMD graph reduction [Pey89].

2. Graphinators and SIMD graph reduction [HM88].

3. Everything you wanted to know about lazy functional languages but were afraid to ask [Pey87].

3.3 Dataflow languages

Dataflow languages are single-assignment languages, the idea being to execute them in a data-driven manner on (usually) special-purpose hardware.

1. Motivation for the dataflow approach [Arv89].

2. Id. A dataflow language based on speculative computation I-structures and M-structures [Nik90].

3. Sisal. Discusses the language, the compiler, the runtime system, and performance numbers [FCO90].

4 Non-determinism

Languages of this class used non-determinism in a fundamental way to gain parallelism. Also central is the notion of the logical variable as a communication channel. Non-determinism is exploited in two different ways: non-deterministic composition similar to Dijkstra's guarded clauses (in PCN [CT90] and UNITY [CM88]), and non-deterministic goal resolution (in parallel logic programming [Sha89]).

1. A comprehensive survey of concurrent logic programming techniques [Sha89].

2. UNITY. Proving the correctness of parallel programs is emphasized [CM88].

3. PCN. Synthesis of ideas from CSP, UNITY and logic programming [CT90].

4. PARLOG. A concurrent version of Prolog exploiting both AND- and OR-parallelism [CG85].

5 Automatic parallelization

The presence of "dusty-deck" programs in production codes makes automatic parallelization of such programs attractive. Automatic parallelization may also ease the burden of programmers by allowing them to use programming languages and paradigms with which they are more familiar.

5.1 Fortran

Initial efforts were targeted at vector machines.

1. Discusses the basics of data dependence analysis and loop transformations [PW86].

2. Vectorization of "dusty-deck" Fortran programs [PKL80].

3. Differentiates between vectorization and vector optimization. Discusses decision mechanisms [Wol88].

4. Detailed discussion of vectorization and parallelization techniques for Fortran programs [Pol88].

5. More on vectorization of Fortran programs [AK87].

6. Still more of the same stuff [Wol89].

7. Mathematical foundations of dependence analysis [Ban88].

5.2 Scheme

Scheme and other dialects of LISP present a number of difficulties for automatic parallelizers: lists (pointers, non-contiguous storage, no random access), higher-order functions, continuations, closures, and dynamic storage management. Work on such languages is much more recent [HP88, LH88], and the "dusty-deck" problem is not as pronounced.

1. Describes the PARCEL system: vectorized list representation, Fortran-style program transformations [HP88].
2. Describes the CURARE system: details of dependence analysis for Scheme [LH88].
3. Harrison's thesis. Lots of semantics and details of optimizations [Har89].
4. Control flow analysis of Scheme. Not directly concerned with parallelism [Shi88].
5. Describes the transformation of a PDE solver written in pure Lisp to a vectorizable Fortran program for execution on the CRAY X-MP [BH90].

6 The real world

This group of readings examines the question of just how useful these various programming paradigms are, and how willing people are to use them for real-world applications.

1. Examines the differences in what computer scientists provide in parallel languages, and what scientific researchers want in them [PB90].
2. Discusses the limitations of automatic vectorizers and parallelizers, and the complications of program annotations for a variety of commercial Fortran dialects [KB88].
3. A comparison of Linda, parallel object-oriented languages, functional programming, and concurrent logic programming. Guess who wins! This article generated a great deal of debate in letters to the editor and on Usenet [CG89b].

Acknowledgements

This reading list is based on a version that was originally compiled in 1991 by Siddhartha Chatterjee, Jay Sipelstein and myself for a class on Parallel languages. It was updated in the spring of 1994 for the same class.

References

[Agh86] Gul A. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. The MIT Press, Cambridge, MA, 1986.

[Agh90] Gul Agha. Concurrent Object-Oriented Programming. *Communications of the ACM*, 33(9):125–141, September 1990.

[Agh91] Gul Agha. The Structure and Semantics of Actor Language. In *Proceedings of the School/Workshop on Foundations of Object Oriented Languages*, Lecture Notes in Computer Science. Springer-Verlag, 1991.

[AK87] Randy Allen and Ken Kennedy. Automatic Translation of FORTRAN Programs to Vector Form. *ACM Transactions on Programming Languages and Systems*, 9(4):491–542, October 1987.

[Arv89] Arvind and Rishiyur S. Nikhil. A Dataflow Approach to General-purpose Parallel Computing. Computation Structures Group Memo 302, Laboratory for Computer Science, Massachusetts Institute of Technology, July 1989.

[AS88] W. C. Athas and C. L. Seitz. Multicomputers: Message-passing Concurrent Computers. *Computer*, 21(8):9–24, August 1988.

[Bal92] Henri E. Bal. A comparitive study of five parallel programming languages. *Future Generation Computer Systems*, 8:121–135, July 1992.

[Ban88] Utpal Banerjee. *Dependence Analysis for Supercomputing*. Kluwer Academic Publishers, Norwell, MA, 1988.

[BCK90] R. Bagrodia, K. M. Chandy, and E. Kwan. Uc: a language for the Connection Machine. In *Proceedings of Supercomputing '90*, pages 525–534. IEEE Computer Society Press, November 1990.

[BH90] James M. Boyle and Terence J. Harmer. A Practical Functional Program for the CRAY X-MP. PREPRINT MCS-P159-0690, Mathematics and Computer Science Division, Argonne National Laboratory, July 1990.

[BHJ+87] A. Black, N. Hutchinson, E. Jul, H. Levy, and L. Carter. Distribution and Abstract Data Types in Emerald. *IEEE Transactions on Software Engineering*, 13(1):65–76, January 1987.

[Ble93] Guy E. Blelloch. NESL: A nested data-parallel language (version 2.6). Technical Report CMU-CS-93-129, School of Computer Science, Carnegie Mellon University, April 1993.

[BST89] Henri E. Bal, Jennifer G. Steiner, and Andrew S. Tanenbaum. Programming Languages for Distributed Computing Systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[CD90] Andrew A. Chien and William J. Dally. Experience with Concurrent Aggregates (CA): Implementation and Programming. In *Proceedings of the Fifth Distributed Memory Computers Conference*, Charleston, South Carolina, April 8–12 1990. SIAM.

[CG85] K. L. Clarke and S. Gregory. PARLOG: Parallel Programming in Logic. *ACM Transactions on Programming Languages and Systems*, 8(1):1–49, January 1985.

[CG89a] Nicholas Carriero and David Gelernter. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Computing Surveys*, 21(3):323–357, September 1989.

[CG89b] Nicholas Carriero and David Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, April 1989.

[CGH90] Rohit Chandra, Anoop Gupta, and John Hennessy. COOL: A Language for Parallel Programming. In David Padua, David Gelernter, and Alexandru Nicolau, editors, *Languages and Compilers for Parallel Computing*, Research Monographs in Parallel and Distributed Computing, pages ??–?? The MIT Press, 1990.

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, MA, 1988.

[CT90] K. Mani Chandy and Stephen Taylor. A Primer for Program Composition Notation. Available from Caltech, June 1990.

[FCO90] John T. Feo, David C. Cann, and Rodney R. Oldehoeft. A Report on the Sisal Language Project. *Journal of Parallel and Distributed Computing*, 10(4):349–366, December 1990.

[FKT90] Ian Foster, Carl Kesselman, and Stephen Taylor. Concurrency: Simple Concepts and Powerful Tools. *The Computer Journal*, 33(6):501–507, December 1990.

[FT90] Ian Foster and Stephen Taylor. *Strand: New Concepts in Parallel Programming*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[GC90] David Gelernter and Nicholas Carreiro. *How To Write Parallel Programs: A First Course*. The MIT Press, Cambridge, Massachusetts, 1990.

[GG88] Ron Goldman and Richard P. Gabriel. Qlisp: Experience and New Directions. In *Proceedings of the ACM/SIGPLAN PPEALS 1988, New Haven, CT*, pages 111–123, 1988.

[Hal85] R. H. Halstead. MultiLisp: A Language for Concurrent Symbolic Computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–538, October 1985.

[Hal90] R. H. Halstead. New Ideas in Parallel Lisp: Language Design, Implementation, and Programming Tools. In G. Goos and J. Hartmanis, editors, *Parallel Lisp: Languages and Systems, Proceedings of the U.S./Japan Workshop on Parallel Lisp*, volume 441, pages 2–57. Springer-Verlag, 1990.

[Har89] Williams Ludwell Harrison III. The Interprocedural Analysis and Automatic Parallelization of Scheme Programs. *Lisp and Symbolic Computation: An International Journal*, 2(3/4), October 1989.

[HCD89] Waldemar Horwat, Andrew Chien, and William J. Dally. Experience with CST: Programming and Implementation. In *Proceedings of the ACM SIGPLAN 89 Conference on Programming Language Design and Implementation*, 1989.

[Hig93] High Performance Fortran Forum. *High Performance Fortran Language Specification*, May 1993.

[HM88] Paul Hudak and Eric Mohr. Graphinators and the Duality of SIMD and MIMD. In *ACM Conference on Lisp and Functional Programming*, pages 224–234, July 1988.

[Hoa78] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[HP83] A. Nico Habermann and Dewayne E. Perry. *Ada for Experienced Programmers*, chapter 11. Addison-Welsey, Reading, MA, 1983.

[HP88] Luddy Harrison and David A. Padua. PARCEL: Project for the Automatic Restructuring and Concurrent Evaluation of Lisp. In *Proceedings of the 1988 International Conference on Supercomputing, St. Malo, France*, pages 527–538, 1988.

[HWW89] Leonard G. C. Hamey, Jon A. Webb, and I-Chen Wu. An Architecture Independent Programming Language for Low-Level Vision. *Computer Vision, Graphics, and Image Processing*, 48:246–264, 1989.

[Ive62] Kenneth E. Iverson. *A Programming Language*. Wiley, New York, 1962.

[KB88] Alan H. Karp and Robert G. Babb II. A Comparison of 12 Fortran Dialects. *IEEE Software*, 5(5):52–67, September 1988.

[KHM89] David A. Krantz, Robert H. Halstead, Jr., and Eric Mohr. Mul-T: A High-Performance Parallel Lisp. In *Proceedings of the SIGPLAN'89 Conference on Programming Language Design and Implementation*, pages 81–90, 1989.

[LH88] James R. Larus and Paul N. Hilfinger. Restructuring Lisp Programs for Concurrent Execution. In *Proceedings of the ACM/SIGPLAN PPEALS 1988, New Haven, CT*, pages 100–110, 1988.

[May83] D. May. Occam. *ACM SIGPLAN Notices*, 18(4):69–79, April 1983.

[MH88] Z. G. Mou and P. Hudak. An algebraic model of divide-and-conquer and its parallelism. *Journal of Supercomputing*, 2(3):257–278, November 1988.

[MNP+90] Peter H. Mills, Lars S. Nyland, Jan F. Prins, John H. Reif, and Robert A. Wagner. Prototyping parallel and distributed programs in Proteus. Technical Report UNC-CH TR90-041, Computer Science Department, University of North Carolina, 1990.

[MR92] Michael Metcalf and John Reid. *Fortran 90 Explained*. Oxford University Press, New York, 1992.

[Nik90] Rishiyur S. Nikhil. ID Version 90.0 Reference Manual. Computation Structures Group Memo 284-1, Laboratory for Computer Science, Massachusetts Institute of Technology, July 1990.

[PB90] Cherri M. Pancake and Donna Bergmark. Do Parallel Languages Respond to the Needs of Scientific Programmers? *Computer*, 23(12):13–23, December 1990.

[Per92] R. H. Perrott. Parallel language developments in Europe: an overview. *Concurrency: Practice and Experience*, 4(8):589–617, December 1992.

[Pey87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, London, UK, 1987.

[Pey89] S. L. Peyton Jones. Parallel Implementations of Functional Programming Languages. *The Computer Journal*, 32(2):175–186, 1989.

[PKL80] David Padua, David Kuck, and Duncan Lawrie. High-Speed Multiprocessors and Compilation Techniques. *IEEE Transactions on Computers*, C-29(9):763–776, September 1980.

[Pol88] Constantine D. Polychronopoulos. *Parallel Programming and Compilers*. Kluwer Academic Publishers, Norwell, MA, 1988.

[PW86] D. A. Padua and M. Wolfe. Advanced Compiler Optimizations for Supercomputers. *Communications of the ACM*, 29(12):1184–1201, December 1986.

[QH90] Michael J. Quinn and Philip J. Hatcher. Data-Parallel Programming on Multicomputers. *IEEE Software*, 7(5):69–76, September 1990.

[RS87] J. R. Rose and G. L. Steele Jr. C*: An Extended C Language for Data Parallel Programming. In *Proceedings Second International Conference on Supercomputing, Vol. 2*, pages 2–16, San Francisco, CA, May 1987.

[Sab88] Gary W. Sabot. *The Paralation Model: Architecture-Independent Parallel Programming*. The MIT Press, Cambridge, Massachusetts, 1988.

[SB91] Jay Sipelstein and Guy E. Blelloch. Collection-oriented languages. *Proceedings of the IEEE*, 79(4):504–523, April 1991.

[Sch93] Wolfgang Schreiner. Parallel functional programming: An annotated bibliography. Available from author (schreine@risc.uni-linz.ac.at), December 1993.

[SH86] Guy L. Steele Jr. and W. Daniel Hillis. Connection Machine Lisp: Fine-Grained Parallel Symbolic Processing. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 279–297, 1986.

[Sha89] Ehud Shapiro. The Family of Concurrent Logic Programming Languages. *ACM Computing Surveys*, 21(3):412–510, September 1989. Except Section 3.

[Shi88] Olin Shivers. Control Flow Analysis in Scheme. In *Proceedings of the SIGPLAN 1988 Conference on Programming Language Design and Implementation*, pages 164–174, June 1988.

[TKB92] A. S. Tannenbaum, M. F. Kaashoek, and H. E. Bal. Parallel programming using shared objects and broadcasting. *Computer*, 25(8):10–19, 1992.

[Tur79] David A. Turner. A New Implementation Technique for Applicative Languages. *Software—Practice and Experience*, 9:31–49, 1979.

[WKH92] Barbara B. Wyatt, Krishna Kavi, and Steve Hufnagel. Parallellism in object-oriented languages: A survey. *IEEE Software*, 9(6):56–66, November 1992.

[Wol88] Michael Wolfe. Vector Optimization vs Vectorization. *Journal of Parallel and Distributed Computing*, 5:551–567, 1988.

[Wol89] Michael Wolfe. *Optimizing Supercompilers for Supercomputers*. The MIT Press, Cambridge, MA, 1989.