Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

441

T. Ito R.H. Halstead, Jr. (Eds.)

Parallel Lisp: Languages and Systems

US/Japan Workshop on Parallel Lisp
Sendai, Japan, June 5–8, 1989
Proceedings

Foreword by J. McCarthy

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo Hong Kong

Editors

Takayasu Ito
Department of Information Engineering
Faculty of Engineering, Tohoku University
Sendai, 980, Japan

Robert H. Halstead, Jr.
Cambridge Research Lab
Digital Equipment Corporation
1 Kendall Square, Building 700
Cambridge, MA 02139, USA

CR Subject Classification (1987): C.1.2, C.4, D.1.3, D.3–4

ISBN 3-540-52782-6 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-52782-6 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

Foreword

Since computers were first invented, it has been known that serial computation has limits that can be far exceeded by using parallel computation. Even very early computers used parallelism in carrying out arithmetic operations, and improved hardware has expanded this kind of parallelism.

The first project to build a parallel computer was probably Illiac 4 proposed by the early 1960s. It was over-elaborate, the cellular automaton influenced design made it almost immune to programming, and by the time it was working, it had been over-run by the Cray I, and other ordinary serial computers with added vector facilities and pipelining.

Parallel computing poses a harsh dilemma for the system designer. The largest number of arithmetic operations per second is obtained by designs that offer very limited communication among the processors. If the problem fits such a design, it can run very fast, but for many kinds of problem, effective parallelism cannot be obtained without good communication. Designs offering the best communication, e.g. fully shared full-speed memory, cannot compute as fast as other designs and don't scale easily to very large numbers of processors. Ingenuity sometimes provides unexpected solutions, but sometimes it seems that no amount of ingenuity will substitute for shared memory.

The largest numerical computations are those involving partial differential equations. When these are replaced by difference equations in the most obvious ways, they seem to lend themselves to regular arrays of processors. However, as soon as shock waves require concentrating the computation on dynamically selected parts of space, and radiation propagates influences at the speed of light, the most obvious grids waste computation.

The idea of queue-based multiprocessing arose in the early 1960s, but support was not offered for actually implementing it. The idea is that processes can dynamically generate subprocesses that can be done in parallel, and these subtasks are put in a queue structure from which processors take tasks when they become free. On the one hand, queue based multiprocessing seems to require a shared memory, which is expensive. On the other hand, it offers straightforward ways of programming almost any kind of problem using techniques that aren't far from those used in programming for serial computers. Moreover, the programs produced don't depend on the number of processors, which can even change dynamically. The languages needed are just the usual serial languages augmented by a few constructions for declaring parallelism.

Queue-based multiprocessing is particularly well suited for symbolic computation, where the same recursive process may involve data structures of similar structure but of enormously varied size, and where the data structures are dynamically determined. Lisp can be made into a parallel language in a variety of ways without distorting its character. Moreover, many Lisp programs written for serial machines can be made to take advantage of parallelism of this kind. Putting Lisp programs on parallel machines based on the idea of a cellular automaton is problematical, and if a solution is found for a particular program, it is likely to be strongly configuration dependent.

Projects to build parallel Lisp systems in the form of compilers and interpreters for existing or announced shared memory multiprocessors began in the middle 1980s and have proceeded uneventfully. It seems to be a straightforward task whenever the necessary resources can be assembled and maintained. The initial proposals for parallel constructs were similar to each other. In fact my original idea in proposing the workshop reported in these papers was that it would be a standardization conference, and on the basis of some experience with the parallel constructs, a proposal could be made for the incorporation of parallelism into Common Lisp. Unfortunately, it seems that the field of parallel Lisp is not quite ready for standardization. I hope standardization will be pursued in a future meeting.

The present workshop is about the first in which extensive experience in actually implementing and using the parallel constructs is extensively reported. The approaches taken are adequately introduced in the Preface.

It seems to me that both queue-based multi-processing and systems with weaker communication are destined to survive and will be suitable for different kinds of application. Queue-based multi-procesing will provide general and straightforward facilities of all kinds of work, but some kinds of program will compute faster on more specialized systems.

John McCarthy

Preface

Lisp has been the most popular programming language for artificial intelligence and symbolic computing. Since the early 1980's, parallel Lisp languages and parallel execution of Lisp programs have been studied extensively in response to the needs of AI applications and progress in parallel architecture. Early parallel Lisp projects resulted in parallelized Lisp interpreters and data-flow models for parallel execution of Lisp programs. More recently, parallel Lisp systems that can execute realistic applications with "industrial-strength" performance have been developed.

Inspired by two parallel Lisp languages, Qlisp and Multilisp, parallel Lisp languages have been proposed and developed by several research groups in the United States and Japan. The U.S./Japan Workshop on Parallel Lisp, held at Tohoku University in Sendai from June 5–8, 1989, brought together many of these researchers to discuss the techniques and conceptual models underlying their research projects. Significant advances in several areas were reported, notably in high-performance parallel Lisp implementation techniques and language constructs for speculative computing. The `future` construct emerged as the most popular approach for introducing parallelism into Lisp programs, and several advances in defining `future` to be compatible with other constructs, such as continuations, were presented.

Workshop participants submitted papers describing their research projects, which were distributed at the workshop. Based on the presentations and discussions at the workshop, participants revised their manuscripts and contributed them to this book. This book is organized into two parts. Part I focuses on parallel Lisp languages and programming models and Part II focuses on parallel Lisp systems and architectures. Contributions in this book are of two kinds: full papers and extended abstracts. Many participants contributed full papers describing the research they presented at the workshop; some participants contributed extended abstracts, because the work had already been published elsewhere or for other reasons. Generally, all the contributions concern how to make parallel computation more practical in Lisp through new approaches to language semantics, system design, or implementation techniques. This preface gives a summary of the workshop activities and the content of the papers included in this book.

The first paper in Part I is by Robert Halstead. It gives an overview of current language design and implementation ideas for parallel Lisp systems, based on his experience with the Multilisp language (an extended version of Scheme) and its implementations. The paper presents three criteria for judging Scheme extensions for parallel computing: compatibility with sequential Scheme, invariance of the result when `future` is introduced into side-effect-free Scheme programs, and modularity. These criteria are used to evaluate proposed mechanisms for continuations, speculative computing, and exception handling. The paper also discusses several other directions for further research in improving the Multilisp language and its implementations. These research topics include data types to facilitate data-parallel computing; techniques to reduce scheduling costs and enhance locality; garbage collection; and tools to assist in the development of parallel programs.

The next paper is by Takayasu Ito and Manabu Matsui. It introduces the parallel Lisp language PaiLisp and its definition in terms of the kernel language PaiLisp-Kernel. It shows how futures, Qlisp's exclusive `qlambda` closures, and many other concepts can be defined in terms of a small set of kernel constructs. The paper also introduces a novel interpretation of continuations in a parallel language, in which invoking a continuation alters the flow of control in the task that originally captured the continuation, instead of in the invoking task.

A presentation at the workshop by Morry Katz described another way to define continuations in a parallel language. This work is represented in this book by an extended abstract by Katz and Daniel Weise and is also discussed in Halstead's paper. The full paper appears in the 1990 ACM Conference on Lisp and Functional Programming.

A paper by Randy Osborne presents an interesting model for speculative computation called the *sponsor model* and gives performance results from an implementation of this model in Multilisp. The sponsor model is a modular framework for providing more control over scheduling than is provided by standard parallel Lisp systems. This control is useful in many applications, especially those that involve search. The sponsor model can also be useful in system contexts where resources must be shared among users, or wherever a user needs control over groups of subtasks.

James Miller and Barbara Epstein write about copying garbage-collection algorithms for parallel Lisp implementations, discussing the issues crucial to achieving good performance. Their parallel garbage-collection algorithm supports speculative computation with the help of a *weak pair* construct and garbage-collection of irrelevant tasks. Their approach to speculative computation via implicit reclamation of irrelevant tasks contrasts with Osborne's sponsor model, in which irrelevant tasks are identified explicitly by program commands.

Ron Goldman, Richard Gabriel, and Carol Sexton give an overview of the Qlisp language, a parallel version of Common Lisp with "industrial-strength" performance. In addition to futures, Qlisp supports *propositional parameters* to help limit excessive process creation and a `qlambda` construct for monitor-like mutual exclusion. Qlisp also includes `kill-process` and `catch/throw` constructs for killing processes, which is useful for some forms of speculative computation. To reduce the need for programmers to address synchronization problems explicitly, the paper introduces two new mechanisms: *heavyweight futures* and *partially, multiply invoked functions.*

At the workshop, Joseph Weening presented an analytical model of the performance of *dynamic partitioning*—a technique for reducing the frequency of process creation by avoiding it when the number of available processes exceeds the number of processors. Joseph D. Pehoushek reported on a Qlisp implementation using dynamic partitioning and other methods for reducing process creation costs. Pehoushek and Weening have contributed a combined paper to this book, presenting both their experimental and analytical results; experimentally, dynamic partitioning gives performance improvements of up to a factor of 2 in the Qlisp implementation.

Robert Kessler and Mark Swanson describe Concurrent Scheme, a language for parallel programming on a distributed-memory architecture. The central and novel concept of Concurrent Scheme is the *domain*, which is an entity containing mutable data. At most one thread of execution can be active in each domain at any time. A domain thus enforces mutual exclusion on accesses to its contents in a manner similar to Qlisp's `qlambda` or Hoare's monitors. Though threads cannot execute concurrently in the same domain, threads can execute concurrently in separate domains. Concurrent Scheme has been prototyped on the Bolt, Beranek, and Newman GP1000 multiprocessor, but is ultimately targeted for the Hewlett-Packard Mayfly architecture.

W. Ludwell Harrison gave a workshop presentation describing techniques used in Parcel, a system for interprocedural analysis and restructuring of sequential Scheme programs for parallel execution. Good speedups have been achieved by applying his parallelizing compiler to several programs, including the Boyer benchmark. A detailed paper about his methods appears in *Lisp and Symbolic Computation 2:3/4* (1989, pp. 179–396). Zahira Ammarguellat presented her work on *control-flow normalization*—simplifying the control flow of Scheme programs to facilitate parallelizing transformations. A joint paper in this book by Harrison and Ammarguellat gives an overview and thoughtful critique of Parcel's design. It then outlines the design principles of Miprac, a successor to Parcel that extends Parcel's techniques and applies them to a broad range of procedural languages from FORTRAN to Scheme.

A presentation at the workshop by Akinori Yonezawa discussed reflection in the object-oriented concurrent language ABCL/R. More detail on this subject appears in a collection of papers edited by Yonezawa—*ABCL: An Object-Oriented Concurrent System* (MIT Press, 1990). A brief summary also appears as an extended abstract in this book.

In his presentation, Etsuya Shibayama discussed optimistic and pessimistic synchronization policies in the context of the "car washing problem." A short article in this book by Shibayama and Yonezawa gives the highlights; fuller details appear in Yonezawa's book, *ABCL: An Object-Oriented Concurrent System.*

Mario Tokoro presented *MD-based computing*, which introduces notions of "distance" and "mass" and proposes an approach that models objects as being under the influence of a computational "gravitational field." He summarizes these ideas in an article in this book.

Part II of the book, focusing on parallel Lisp systems and architectures, begins with a paper by Ikuo Takeuchi based on practical experience with TAO, a Lisp dialect that fuses the procedural, object-oriented, and logic-programming paradigms into one language. TAO runs on the ELIS Lisp machine and is in production use by many users. The paper discusses TAO's approaches to process management, sharing Lisp programs among processes and users, name-space problems associated with symbol packages, concurrent primitives, and concurrent program debugging. Most of the key primitives of TAO are implemented in ELIS microcode, leading to good performance, even in interpreted mode—even a TCP/IP network system was run under the interpreter!

A short article by Ken-ichiro Murakami describes the MacELIS multiprocessor, designed to be compatible with single-processor TAO/ELIS systems. Interprocessor communication is supported by an *in-core pseudo-network*, which uses standard network protocols to transmit messages through a shared memory.

An extended abstract by David Kranz, Robert Halstead, and Eric Mohr describes Mul-T, a parallel Lisp system with "industrial-strength" performance. Mul-T uses an optimizing compiler to generate code for an Encore Multimax multiprocessor and offers real speed-ups over good sequential implementations. Additional information about Mul-T appears in the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation and in the paper by Halstead in this book.

Dan Pierson outlines the issues in integrating parallel Lisp systems with modern operating systems. He discusses the application of services provided by the Mach and Unix operating systems to the problems of process management, scheduling, and exception handling in a Qlisp-based parallel Lisp system.

Hideya Iwasaki gives a brief description of mUtilisp, a parallel dialect of Utilisp (University of Tokyo Interactive Lisp). mUtilisp programs are composed of processes that communicate by message-passing; shared objects are not supported. An implementation of mUtilisp that simulates parallelism by time-slicing is available.

A paper by Taiichi Yuasa and Takafumi Kawana describes an experimental parallel computer (the PM1) and the PMLisp language. The PM1 is built around an 8-bit microprocessor (the Z-80) and a butterfly network for interprocessor communication. It is a first prototype of a massively parallel "P-machine." PMLisp is a Scheme-like language with explicit constructs for interprocessor communication. Examples are given showing how to express a shortest-path algorithm and models of various network topologies in PMLisp.

Hiroshi Yasui, Toshikazu Sakaguchi, Kohichi Kudo, and Nobuyuki Hironishi write a short article about the EVLIS machine, a multiprocessor composed of EVAL II processors. *Regulation lists* are introduced as a mechanism for controlling the execution of parallel processes and the performance of the "List-Tarai-4" benchmark (using regulation lists) on EVLIS is analyzed. A multi-port memory system for EVLIS is also proposed as a way of increasing performance.

At the workshop, Norihisa Suzuki described the architecture of the TOP-1 multiprocessor and an ongoing project on parallel Common Lisp based on futures. A novel feature of this Lisp is that it supports the use of futures for expressions that return multiple values. Plans for the installation of a real-time garbage collector were also presented. In this book, Suzuki writes about the architecture of TOP-1 and describes parallel processing research projects that have been performed on it.

Eiichi Goto was invited to the workshop to present his work on high-speed computer architectures based on Josephson-junction computing and cyclic pipeline architectures. Such architectures should complement techniques for using parallelism to achieve high-performance symbolic computing. He gave an interesting talk based on his published article in *IEEE Transactions on Computers* (June 1989).

In addition to the formal presentations, the workshop included three discussion sessions in which the participants exchanged views on applications, benchmarks, architectures for parallel Lisp, and the relationships between parallel Lisp and parallel logic programming. The first two discussion sessions were devoted to informal discussion of language design, performance, scheduling, and architecture for parallel Lisp.

Kazunori Ueda began the final discussion session by giving an overview of the GHC guarded-Horn-clause language for parallel logic programming, developed at the Institute for New Generation Computer Technologies (ICOT). He also explained the relationship between GHC and KL1, an extension of GHC also developed at ICOT, and discussed the need for meta-level operations (reflection) in future logic-programming languages. Akikazu Takeuchi then commented on the close relationship between futures and logic variables in parallel logic languages.

Discussion then turned to the question of applications for parallel symbolic computing. Hiroshi Okuno discussed his experience in parallelizing two large AI systems—the production-system language OPS-5, and the truth-maintenance system ATMS: these programs are of the scale that should be available as benchmark programs. W. Ludwell Harrison commented on the value of the "real" programs in the set of FORTRAN benchmarks that have been collected at the University of Illinois and suggested that a similar collection of Lisp benchmarks should be created. Among the benchmarks proposed by various workshop participants were sorting, the Gabriel Lisp benchmark set, the Japan Lisp benchmark set, a symbolic algebra system such as REDUCE, N-body simulations, fast Fourier transforms, robotics, animation, and graphics. At the close of the workshop, all agreed that a standard set of realistic benchmarks for parallel symbolic computing would be very valuable in advancing parallel symbolic computing toward practical utility and making it easier to compare the merits of different approaches.

Workshop participants had several opportunities for informal discussion while absorbing Japanese culture and technology through a series of events outside of the formal agenda: two receptions, a Japanese banquet, and a day of "extracurricular activities" organized by the workshop's hosts. The day of activities included a tour of the "superclean room" for VLSI fabrication at Tohoku University (a "Class 0.0001" clean room where the level of dust is so low as to be unmeasurable), the Golden Temple at Hiraizumi, the temples and gardens at Mohtsu-ji, picturesque Matsushima Bay, and another delicious Japanese banquet at Taritsu-an Restaurant overlooking Matsushima Bay.

The papers in this book describe advances in language design and system architecture for parallel Lisp, but few of them discuss theory or applications of realistic size. In the theoretical domain, we would like to see more work done on the semantics of parallel Lisp languages. Specifically, a sound semantic understanding of combining futures (for parallelism) and continuations (for control) would be valuable. Incorporating speculative computation into such a semantic theory is another important challenge.

Experiments with several small- and medium-sized parallel Lisp applications have been conducted (see, for example, Section 3 of Halstead's paper), but it will be very important to have experience with realistic, large-scale applications too. We hope that the availability of parallel Lisp systems with "industrial-strength" performance will enable and encourage the development of such parallel Lisp applications. It is through experience with such applications that we will learn how to make parallel Lisp systems that are effective for programming in the large as well as in the small. This in turn is a vital step to making parallel Lisp a valuable, general-purpose computing technology for a wide range of users.

The idea of holding a U.S./Japan workshop on parallel Lisp was first suggested to one of us (Ito) by John McCarthy in January, 1987. Halstead was recruited as a co-organizer later that year. Dick Gabriel was later recruited to help, and did most of the organizing work on the U.S. side; unfortunately, he was unable to attend the workshop itself. We also regret the absence from the workshop of other parallel Lisp researchers who were unable to attend, and hope to see all parallel Lisp researchers united at a future workshop.

The proposal to publish this book as part of the *Lecture Notes in Computer Science* series was made to Juris Hartmanis (and accepted) in the fall of 1989. We thank him and all of the above people for their vital role in bringing about the workshop and the publication of this book. We also thank Hans Wossner of Springer-Verlag for his assistance in this book's publication. Finally, we thank all those who helped organize, and participated in, the workshop for their invaluable contributions.

Takayasu Ito
Robert H. Halstead, Jr.
Cambridge, Massachusetts
March, 1990

Table of Contents