THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE

OFFICE OF NAVAL RESEARCH
Advanced Book Series
Consulting Editor
André M. van Tilborg

Other titles in the series:

FOUNDATIONS OF KNOWLEDGE ACQUISITION: Cognitive Models of Complex Learning, edited by Susan Chipman and Alan L. Meyrowitz
ISBN: 0-7923-9277-9

FOUNDATIONS OF KNOWLEDGE ACQUISITION: Machine Learning, edited by Alan L. Meyrowitz and Susan Chipman
ISBN: 0-7923-9278-7

FOUNDATIONS OF REAL-TIME COMPUTING: Formal Specifications and Methods, edited by André M. van Tilborg and Gary M. Koob

FOUNDATIONS OF REAL-TIME COMPUTING: Scheduling and Resource Management, edited by André M. van Tilborg and Gary M. Koob
ISBN: 0-7923-9166-7
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendees</td>
<td>vii</td>
</tr>
<tr>
<td>Speakers</td>
<td>xi</td>
</tr>
<tr>
<td>System Demonstrations</td>
<td>xiii</td>
</tr>
<tr>
<td>Preface</td>
<td>xv</td>
</tr>
<tr>
<td>1. Deductive Derivation of Parallel Programs</td>
<td>1</td>
</tr>
<tr>
<td>Peter Pepper, <em>Technical University of Berlin</em></td>
<td></td>
</tr>
<tr>
<td>2. Derivation of Parallel Sorting Algorithms</td>
<td>55</td>
</tr>
<tr>
<td>Douglas R. Smith, <em>Kestrel Institute</em></td>
<td></td>
</tr>
<tr>
<td>3. Some Experiments in Transforming Towards Parallel Executability</td>
<td>71</td>
</tr>
<tr>
<td>Helmut Partsch, <em>University of Nijmegen</em></td>
<td></td>
</tr>
<tr>
<td>4. The Use of the Tupling Strategy in the Development of Parallel Programs</td>
<td>111</td>
</tr>
<tr>
<td>Alberto Pettorossi, Enrico Pietropoli &amp; Maurizio Proietti, <em>University of Rome II &amp; IASI CNR</em></td>
<td></td>
</tr>
<tr>
<td>5. Scheduling Program Task Graphs on MIMD Architectures</td>
<td>153</td>
</tr>
<tr>
<td>Apostolos Gerasoulis &amp; Tao Yang, <em>Rutgers University</em></td>
<td></td>
</tr>
<tr>
<td>6. Derivation of Randomized Sorting and Selection Algorithms</td>
<td>187</td>
</tr>
<tr>
<td>Sanguthevar Rajasekaran &amp; John H. Reif, <em>University of Pennsylvania &amp; Duke University</em></td>
<td></td>
</tr>
<tr>
<td>7. Time-Space Optimal Parallel Computation</td>
<td>207</td>
</tr>
<tr>
<td>Michael A. Langston, <em>University of Tennessee</em></td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>225</td>
</tr>
</tbody>
</table>
LIST OF ATTENDEES AT THE ONR WORKSHOP ON PARALLEL ALGORITHM DERIVATION AND PROGRAM TRANSFORMATION

Sandeep Bhatt
Computer Science Dept.
Yale University
New Haven, CT 06520
bhatt-sandeep@cs.yale.edu

Thomas Cheatham
Dept. of Computer Science
Harvard University
Cambridge, MA 02138
cheatham@harvard.harvard.edu

James M. Boyle
Math & Computer Science Div.
Argonne National Laboratory
Building 221
9700 South Cass Ave.
Argonne, IL 60439-4844
boyle@mcs.anl.gov

Xushen Chen
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012

Jiazhen Cai
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
cai@cs.nyu.edu

Marina Chen
Dept. of Computer Science
Yale University
P.O. Box 2158 Yale Station
New Haven, Conn. 06520
chen-marina@cs.yale.edu

David Callahan
Tera Computer
david@tera.com

Young-il Choo
Dept. of Computer Science
Yale University
P.O. Box 2158 Yale Station
New Haven, Conn. 06520
choo-young-il@cs.yale.edu

Johnny Chang
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
chang@cs.nyu.edu

Richard Cole
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
cole@cs.nyu.edu
Martin S. Feather
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90291
feather@vaxa.isi.edu

Apostolos Gerasoulis
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903
gerasoulis@cs.rutgers.edu

Ben Goldberg
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
goldberg@cs.nyu.edu

Allen Goldberg
Kestrel Institute
3620 Hillview Ave.
Palo Alto, CA 94304
goldberg@kestrel.edu

Fritz Henglein
DIKU
Universitetsparken 1
DK-2100 København 0
Denmark
henglein@diku.dk

Donald B. Johnson
Dept. of Math and CS
Dartmouth College
Hanover, NH 03755
djohnson@dartmouth.edu

Neil Jones
DIKU
Universitetsparken 1
DK-2100 København 0
Denmark
neil@diku.dk

Zvi Kedem
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
kedem@nyu.edu

S. Rao Kosaraju
Dept. of Computer Science
Johns Hopkins University
Baltimore, MD 21218
kosaraju@cs.jhu.edu

Mike Langston
Dept. of Computer Science
University of Tennessee
Knoxville, TN 37996
langston@cs.utk.edu

David Lillie
Dept. of Computing
Imperial College
Queens Gate
London SW7 2AZ, UK
djl@doc.ic.ac.uk

Fillia Makedon
Dept. of Math and CS
Dartmouth College
Hanover, NH 03755
makedon@dartmouth.edu
Lambert Meertens
Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
lambert@cwi.nl

Gary L. Miller
Carnegie Mellon University
School of Computer Science
Schenley Park
Pittsburgh, PA 15213-3890
glmiller@cs.cmu.edu

Bud Mishra
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
mishra@cs.nyu.edu

Bob Paige
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
paige@cs.nyu.edu

Krishna Palem
IBM T. J. Watson Research Ctr.
P.O. Box 218
Yorktown Heights, N.Y. 10598
palem@cs.nyu.edu

Helmut Partsch
Dept. of Informatics
Catholic University of Nijmegen
NL-6525 ED Nijmegen, The Netherlands
helmut@cs.kun.nl

Peter Pepper
Tech. Universität Berlin
FB 20, Institut fuer Angewandte
Informatik, Sekr. SWT FR 5-6
Franlinstrasse 28-29
D-1000 Berlin 10, Germany
pepper@cs.tu-berlin.de

Alberto Pettorossi
University of Rome II
c/o IASI CNR
Viale Manzoni 30
I-00185 Roma, Italy
adp@iasi.rm.cnr.it

Jan Prins
Computer Science Dept.
University of North Carolina
Chapel Hill, NC
prins@cs.unc.edu

John Reif
Computer Science Dept.
Duke University
Durham, N.C. 27706
reif@cs.duke.edu

Larry Rudolph
IBM T. J. Watson Research Ctr.
P.O. Box 218
Yorktown Heights, N.Y. 10598
rudolph@watson.ibm.com

Jack Schwartz
Courant Institute
Computer Science Dept.
New York University
New York, New York 10012
schwartz@cs.nyu.edu
Doug Smith  
Kestrel Institute  
3620 Hillview Ave.  
Palo Alto, CA 94304  
smith@kestrel.edu

Rob Strom  
IBM T. J. Watson Research Ctr.  
P.O. Box 218  
Yorktown Heights, N.Y. 10598  
strom@watson.ibm.com

Carolyn Talcott  
Dept. of Computer Science  
Stanford University  
Stanford, CA 94305  
talcott@cs.stanford.edu

Thanasis Tsantilas  
Dept. of Computer Science  
Columbia University  
New York, New York 10027  
thanasis@cs.columbia.edu

Valentin Turchin  
CUNY  
turcc@cunyvm.cuny.edu

Uzi Vishkin  
University of Maryland  
UMIACS, A. V.  
Williams Building  
College Park, MD 20742-3251  
vishkin@umiacs.umd.edu

Tao Yang  
Dept. of Computer Science  
Rutgers University  
New Brunswick, NJ 08903  
yang@cs.rutgers.edu

Allan Yang  
Dept. of Computer Science  
Yale University  
P.O. Box 2158 Yale Station  
New Haven, Conn. 06520  
yang-allan@cs.yale.edu

Chee Yap  
Courant Institute  
Computer Science Dept.  
New York University  
New York, New York 10012  
yap@cs.nyu.edu

David S. Wile  
Information Sciences Institute  
4676 Admiralty Way  
Marina del Rey, CA 90291  
wile@vaxa.isi.edu
SPEAKERS

Uzi Vishkin  
*University of Maryland & University of Tel Aviv*
An Introduction to Parallel Algorithms

Mike Langston  
*University of Tennessee*
Resource-Bounded Parallel Computation

Lambert Meertens  
*Mathematisch Centrum*
Deriving Parallel Programs from Specifications

Doug Smith  
*Kestrel Institute*
Automating the Design of Algorithms

S. Rao Kosaraju  
*Johns Hopkins University*
Tree-based Parallel Algorithms

Peter Pepper  
*Tech. Universitaet Berlin*
Deductive Derivation of Parallel Programs

Alberto Pettorossi  
*University of Rome II IASI CNR*
The Use of the Tupling Strategy in the Development of Parallel Programs

Jan Prins  
*University of North Carolina*
Derivation of Efficient Bitonic Sorting Algorithms

James M. Boyle  
*Argonne National Lab.*
Programming Models to Support Transformational Derivation of Parallel Programs

Sandeep Bhatt  
*Yale University*
Mapping Computations onto Machines

David Callahan  
*Tera Computer Co.*
Recognizing and Parallelizing Bounded Recurrences

Marina Chen  
*Young-il Choo Yale University*
Generating Parallel Programs from Algorithmic Specifications

Neil Jones  
*DIKU*
Partial Evaluations
Valentin Turchin  
*CUNY*  
Supercompilation Compared with Partial Evaluation

Gary L. Miller  
*Carnegie Mellon University*  
Tree Contraction Parallel Algorithms

Larry Rudolph  
*IBM Watson Research Ctr.*  
Search for the Right Model

Zvi Kedem  
*New York University*  
Parallel Program Transformations for Resilient Computation

Tom Cheatham  
*Harvard University*  
Using the E-L Kernel Based Compiler System

David Lillie  
*Imperial College*  
Synthesis of a Parallel Quicksort

Helmut Partsch  
*Catholic University of Nijmegen*  
Experiments in Transforming Towards Parallel Execution

Krishna Palen  
*IBM TJ Watson*  
Algorithms for Instruction Scheduling on RISC Machines

Apostolos Gerasoulis  
*Rutgers University*  
A Fast Static Scheduling Algorithm

Rob Strom  
*IBM Watson Research*  
Optimistic Program Transformations

Martin S. Feather  
*USC/ISI*  
Constraints, Cooperation, and Communication

Fritz Henglein  
*DIKU*  
A Parallel Semantics for SETL

Carolyn Talcott  
*Stanford University*  
An Operational Approach to Program Equivalence

David S. Wile  
*Information Sciences Inst.*  
Integrating Syntaxes and Their Associated Semantics

Allen Goldberg  
*Kestrel Institute*  
Refining Parallel Algorithms in Proteus
SYSTEM DEMONSTRATIONS

Doug Smith  
Kestrel Institute  

Allan Yang  
Yale University  

Neil Jones  

Fritz Henglein  
DIKU  

Valentin Turchin  
CUNY  

Jiazhen Cai  
Bob Paige  
New York University

KIDS

Crystal and Metacrystal

SIMILIX

The Refal Supercompiler: Demonstration of Function

RAPTS
Preface

This book contains selected papers from the ONR Workshop on Parallel Algorithm Design and Program Transformation that took place at New York University, Courant Institute, from Aug. 30 to Sept. 1, 1991. The aim of the workshop was to bring together computer scientists in transformational programming and parallel algorithm design in order to encourage a sharing of ideas that might benefit both communities. It was hoped that exposure to algorithm design methods developed within the algorithm community would stimulate progress in software development for parallel architectures within the transformational community. It was also hoped that exposure to syntax directed methods and pragmatic programming concerns developed within the transformational community would encourage more realistic theoretical models of parallel architectures and more systematic and algebraic approaches to parallel algorithm design within the algorithm community.

The workshop Organizers were Robert Paige, John Reif, and Ralph Wachter. The workshop was sponsored by the Office of Naval Research under grant number N00014-90-J-1421. There were 44 attendees, 28 presentations, and 5 system demonstrations. All attendees were invited to submit a paper for publication in the book. Each submitted paper was refereed by participants from the Workshop. The final decision on publication was made by the editors.

There were several motivations for holding the workshop and for publishing papers contributed by its participants. Transformational programming and parallel computation are two emerging fields that may ultimately depend on each other for success. Perhaps, because ad hoc programming on sequential machines is so straightforward, sequential programming methodology has had little impact outside the academic community, and transformational methodology has had little impact at all. However, because ad hoc programming for parallel machines is so hard, and because progress in software construction has lagged behind architectural advances for such machines, there is a much greater need to develop parallel programming and transformational methodologies. This book seeks to stimulate investigation of formal ways to overcome problems of parallel computation - with respect to both software development and algorithm design. It represents perspectives from two different communities - transformational programming and parallel algorithm design - to discuss programming, transformational, and compiler methodologies for parallel architectures, and algorithmic paradigms, techniques, and tools for parallel machine models.
Computer Science is a young field with many distinct areas. Some of these areas overlap in their aims, differ distinctly in their approaches, and only rarely have constituents in common. Throughout the workshop the two (mostly nonoverlapping) communities in algorithms and in transformations reached for understanding, but also argued tenaciously in a way that reflected misunderstanding. It is not clear whether a bridge was formed between algorithms people and their counterparts in programming science. But the editors are optimistic. The chapters of this book sometimes show evidence of entrenchment, but they also reveal a synthesis of thinking from two different perspectives.

Certainly, there must be differences in the activities of researchers in algorithm design and program development, because these areas have different goals. In some respects the chapters of Rajasekaran and Reif, and also Langston are prototypical algorithms papers. Their goal is to compute a mathematical function. Their approach is to form this function by composition, parameter substitution, and iteration from other functions that are either already known to be computable within some time bound, or are shown in the paper to be computable within some bound. The functions being manipulated by algorithm designers are often thought not to be conveniently rendered with formal notation. For example, there may not be a single algorithm paper that makes explicit use of a higher order function (as we see in Pepper's chapter), and yet algorithms are invented every day in the algorithms community that are more complicated than any program conceived by researchers in functional programming. Because the algorithms community does not normally feel responsible for implementations, formal notation (which is so important when communicating with a machine) can be dispensed with. One cannot expect this approach to yield a reasonable program easily, but it may stimulate interest in producing such a program.

In the first four chapters, the transformational programming community is represented. The aim of this community is to make contributions to the science of programming. Particular concerns are with how to specify perspicuous mathematical programs, how to map these programs by meaning-preserving source transformations into efficient implementations, how to analyze the resource utilization of these implementations, and how to prove the specification, the transformations, and the implementation correct. Formal reasoning using notation systems is used so that programs can be manipulated systematically using calculi, and especially calculi that admit to computer mechanization. The approach in the transformational programming community is also genetic or top-down in the sense that the design, analysis, and correctness proof are integrated and developed together (as opposed to the classical verification approach). The stress on a correct implementation makes formalism and notation necessary - machines require precision and don't usually self-correct.
There are also common themes expressed among these chapters. Certainly, each contributor shows concern for practical results, and for theories rich in applications. The transformational chapters describe common goals for further mechanization of the transformational methodology, for transformations that capture algorithm design principles, and for transformational systems to be powerful and convenient enough to facilitate the simultaneous design of new algorithms and the development of their implementations. Within the two algorithm chapters we see an interest in notations for specifying algorithm schema and for the use of standard set theoretic notations for communicating problem specifications. Both communities strive for improvement at the meta-level. The stress on algorithm design principles within the algorithm community corresponds closely to the emphasis on transformational methodology within the transformational community. Consequently, the best solution is not the one most specialized to the particular problem, but one that is general enough to provide new meta-level thinking that might help solve other related problems.

Peter Pepper's provocative opening chapter argues sharply in favor of a transformational perspective for building provably correct parallel programs "by construction" on a realistic parallel architecture - namely the SFMD (single function instead of single instruction, multiple data, distributed memory). Based on practical considerations, he assumes that the data vastly exceeds the number of processors. Consequently, the problem of data distribution is the crucial part of program development. His 'deductive' approach is eclectic in the sense that some backwards reasoning in the style of formal verification is admitted. This approach begins with a high level functional sequential specification that is transformed into an equivalent high level parallel form. The use of infinite stream processing and other generic transformations (justified by associative and distributive laws) are used to implement the parallel specification efficiently. The method is illustrated with well selected examples (such as the fundamental prefix sum problem, and the more elusive problem of context free language recognition) that have also stimulated the algorithms community.

In Pepper's approach, the meta-level reasoning to produce datatype theories that support transformations and the reasoning behind the selection of these transformations are placed within a programming methodology (i.e., is part of a manual process) that utilizes the Bird-Meertens formalism. Pepper envisions an interactive system that supports the deductive approach in order to produce a specification at a low enough level of abstraction to be compiled by an automatic system (such as the one proposed by Gerasoulis and Yang) into efficient nets of communicating processes.

Doug Smith's chapter, which is methodologically similar to Pepper's, makes Pepper's proposal more credible by illustrating formal transformational
derivations using KIDS, a working transformational programming system. Taking the derivation of Batcher’s even-odd sort as an extended case study, Smith provides compelling examples of meta-level reasoning in the formation and manipulation of theories used to derive transformations. These theories are both specific to the domain of sorting, and generic in the way they capture the algorithmic principle of divide and conquer. Although meta-level reasoning is only partially implemented in KIDS, it is intriguing to consider the possibilities of a transformational system, which like LCF, has theories as objects.

Helmut Partsch addresses another important pragmatic concern - the reuse and portability of derivations to different parallel architectures. Like the previous two authors Partsch starts out with a functional specification. However, Partsch’s method seeks to transform this specification into a lower level functional form defined in terms of a parameterized abstract parallel machine whose primitives are higher order library functions (called skeletons by Darlington et al. at Imperial College) that can be mechanically turned into implementations on a variety of specific parallel architectures. Partsch illustrates his methodology with a derivation of a parallel implementation of the Cocke, Kasami, Younger nodal span parser, which is precisely the kind of problem likely to catch the attention of the algorithms community. Parallel parsing is one of the big open algorithmic problems, and the so-called nodal span methods seem to offer greater opportunities for parallelism than other more popular sequential methods.

The chapter of Alberto Pettorossi, Enrico Pietropoli and Maurizio Proietti combines the interests of both the transformational and algorithm community by using program and transformation schemata and formal complexity analysis to prove optimality of their transformations. They investigate the difficult problem of directly implementing program schema containing first order nonlinear recursion on a synchronous parallel machine. The idea is to avoid (or at least bound) redundant computation of function calls and of subexpressions in general. The so-called tupling strategy (a generalization of the well known method of pairing in logic) of Burstall, Darlington, and Pettorossi is one of the main techniques. This results in what Gerasoulis and Yang call coarse-grain parallelism. Their Theorem 2 for general recursive programs proves that their rewriting scheme supports greater parallelism than the one described in Manna’s book of 1974. The technical theorems in section 5 prove the correctness of an optimal parallel translation (the Eureka procedure) for a more particular recursive program scheme (which still represents a large class of functions). This scheme explains the earlier work of Norman Cohen in a more general setting, and also yields more efficient parallel implementations than does Cohen’s transformations.

The transformational community is concerned with mechanical support for a largely intuitive and manual process of designing and applying
transformations to obtain the highest level of program specifications that can be usefully compiled automatically into an executable form. The corresponding goal of the compiler community is to elevate the abstract level at which programs can be effectively translated into efficient executable forms. Thus, the results of the compiler community improve the fully automatic back-end of the full program development process envisioned by transformational researchers.

The chapter by Gerasoulis and Yang, which shows how to statically schedule (i.e., compile) partially ordered tasks in parallel, fits well into the current book. These authors are concerned with the problems of recognizing parallelism, partitioning the data and the program among processors, and scheduling and coordinating tasks within a distributed memory architecture with an asynchronous message passing paradigm for communication. Like so many important scheduling problems, these problems are NP-hard, so heuristic solutions are needed. The heuristics, which take communication complexity into account, favor coarse grain parallelism. The scheduling methods are illustrated with the problem of Gaussian Elimination. The authors give experimental results obtained using their PYRROS system.

The final two chapters are contributions from the algorithm design community. Both papers are unusual in being motivated by pragmatic concerns. Rajasekaran and Reif illustrate how randomization can make algorithms simpler (and hence simpler to implement) and faster using parallel algorithms derived for selection and sorting. They make use of notation to describe parameterized program schemas, where the choice of parameters is based on solutions to recurrences analyzing divide and conquer strategies. The notation is informative and enhances readability, but is not essential to the paper. The authors' methods are generally applicable to various parallel architectures, and not limited to PRAM models.

Langston surveys his own recent algorithms for merging, sorting, and selection in optimal SPACE as well as time per processor. These basic solutions are then used to provide optimal space/time solutions to set operations. The weaker but also more realistic PRAM model known as EREW (exclusive read, exclusive write) is used. Langston considers communication complexity, where a constant amount of space per processor implies optimal communication complexity. Langston also reports how an implementation of his merging algorithm on a Sequent Symmetry machine with 6 processors is faster than a sequential implementation even with small input instances.

During the workshop Donald Johnson asked whether transformational calculi can make the teaching of algorithms easier. Can language abstraction and formal transformational derivation help explain difficult algorithms?
Rajasekaran and Reif show how even a modest use of notation can improve readability considerably. In responding to Johnson's question at the workshop, Doug Smith thought that the overhead required to teach transformational methodology might be too costly for use in a traditional algorithm course. Chee Yap questioned whether pictures might be more useful than notation and algebraic manipulation for explaining some complicated algorithms in computational geometry. Johnson also raised the big question in many of our minds whether a transformational methodology or its implementation in a system such as KIDS could help in the discovery of an as-yet unknown algorithm? Smith reported that new algorithms have been discovered using KIDS, although KIDS is not primarily used for that purpose. Pepper seems to confront this issue with his parsing example. Pettorossi uses the compiler paradigm to obtain new results with program schemata in another attempt. However, these questions remain open.

The primary audience for this book includes graduate students and researchers in parallel programming and transformational methodology. Each chapter contains a few initial sections in the style of a first year graduate text-book with lots of illustrative examples. It could be used for a graduate seminar course or as a reference book for a course in (1) Software Engineering, (2) Parallel Programming, or (3) Formal Methods in Program Development.

Bob Paige, New York University
John Reif, Duke University
Ralph Wachter, Office of Naval Research