Appendix A

Source Code for the SOP Environment

This appendix presents the source code for `nonrec`, `tosop`, and the SOP assembler. The first two are in FP, the last is in the `sed` language accepted by the `sed` batch editor available on UNI”X systems.

A.1 nonrec

```
# nonrec.fp
# transform normal ffp for input to tosop
#
# nonrec: <<name1 body1> ... <nameN bodyN>> => new-body,
# where new-body is functionally equivalent to body1, but
# contains no recursion

Def nonrec optcomp o reduce

# optcomp: if there are any instances of
# <compose, ... <compose, ... > >, raise all the elements in
# the sublist. These instances are common because
# reduce adds new compose operators
Def optcomp null -> id;
            atom -> id;
            (bu = compose) o 1 ->
```

```
                (bur = 1) o length -> _id;
                (bur = 2) o length -> optcomp o 1r;
                hasconst -> optcomp o \/clrconst o aa [id];
                hasid -> optcomp o clrid;
                = o [append o aa raisecomp, id] o tl -> aa optcomp;
                optcomp o (bu apndl compose) o append o
                aa raisecomp o tl;
            aa optcomp

# raisecomp: <compose, x, y, z> => <x, y, z>
# raisecomp: <a, b, c> => <<a, b, c>>
Def raisecomp atom -> [id];
              (bu = compose) o 1 -> tl;
              [id]

# hasconst: <constant, x, y, z> => whether x or y (but not z)
# has the form <constant, ...>
Def hasconst \/or o aa (atom -> _F; (bu = constant) o 1) o tlr

# clrconst: <<constant, x, <constant, ...>>, <z>> =>
#  <constant, x, <constant, ...>>
# clrconst: <<constant, x, y>, <z>> => <constant, x, y, z>
Def clrconst atom o 1r o 1 -> append;
             (bu = constant) o 1 o 1r o 1 -> 1;
             append

Def hasid \/or o aa (bu = id)

Def clrid append o aa ((bu = id) -> _<>; [id])

# reduce: <<name1 body1>...<nameN bodyN>> =>
# an expression for name1 that has no recursive invocations and only
# refers to primitive FFP functions.
# <<f <cond, atom, <construct, id>, <compose, append, <aa, f>>>>>
# =>
# <<cond, <compose, atom, <rselect, 1>>,
#         <compose, <construct, id>, <rselect, 1>>,
#         <compose, append, <aa, apply>, distl,
#                   <construct, <lselect, 1>, <rselect, 1>>>>
```

```
Def reduce [recself] o 1 o 1r o recirc o [1 o 1, id]

# recirc: <root, <<name1, body1>, ... <nameN, bodyN>>> =>
#               <root, <<root, body>>>
# where body performs as required but has no recursion.
# root=nameI for some 0 < I <= N.
# recirc first takes body1 (bodyN if root=name1) and places it
# everywhere that <recursive, name1> appears, then runs reduce
# on all the remaining functions, until there is no more
# recursion in bodyI
Def recirc (while (bur > 1) o length o 1r
                    = o [1, 1 o 1 o 1r] ->  # put root last
                            [1, reverse o 1r];
                  [1, aa rreplace o distl o
                      [recself o 1, tl] o 1r])

# rreplace: <<name1, body1>, <namei, bodyi>> =>
# <namei, bodyi>
# with all occurrences of name1 replaced by body1
Def rreplace [1 o 1r, rtreplace o [1 o 1, 1r o 1, 1r o 1r]]

# rtreplace <old, new, tree> => tree' with any occurrence
# of old replaced by new.
Def rtreplace = o [1, 1r] -> 2;
              atom o 1r -> 1r;
              aa rtreplace o aa apndr o distl o [tlr, 1r]

# recself <name, body> => <name, body'> where any self
# recursion (marked by the appearance of 'name' in body) has
# been removed
Def recself hasname -> [1, [recone]]; id

# recone: <name, body> => non-recursive body for name. Any
# calls to other functions are ignored, even though they
# may lead to mutual recursion.
Def recone = -> _apply;
           atom o 1r -> 1r;
           (bu = compose) o 1 o 1r ->
                /rcomp o (bur apndr <>) o distl;
```

```
          member o [1 o 1r, [_cond, _construct, _while]] ->
                  apndl o [1 o 1r, aa rnorm o tl o distl];
          (bu = aa) o 1 o 1r -> recaa o [1, 1r o 1r];
          member o [1 o 1r,
                    [_tree, _insertt, _insertl, _insertr]] ->
               recins;
          aa recone o distl

# recaa: <name, expr> => non-recursive form of the expression.
# The expression always contains 'name'.
# returns <compose, <aa, recone: f>, distl>
Def recaa [_compose, [_aa, recone], _distl]

# recins: <name, <ff, expr>> => non-recursive form of the
# expression.
# The expression always contains 'name'.
# returns <compose, <aa, recone: f>, distl>
Def recins error o _insertrecur

# rnorm: <name, f> => non-recursive form of this part of
# the conditional or constructor
# if f does not contain 'name', returns
#  <compose, f, <rselect, 1>>
# if f contains name, returns <recone: f>
Def rnorm member o [1, flatten o 1r] -> recone;
          [_compose, 1r, [_rselect, _1]]

# rcomp: <<name, f>, rest of compose> => non-recursive
# form of this composition.
# if f does not contain name, then returns
# <f, rest of compose>
# if f contains name, returns
# <recone:f, <construct, <lselect, 1>,
#                <compose, rest of compose, <rselect, 1>>>>
# special case: if rest of compose is nil
# if f contains name, returns "<apply>"
# if f does not contain name, returns <f, <rselect, 1>>
Def rcomp member o [1 o 1, flatten o 1r o 1] ->
                null o 1r -> [recone o 1];
```

```
                [recone o 1, [_construct, [_lselect, _1],
                                (bu apndl compose) o 1r]];
          null o 1r -> [1r o 1, [_rselect, _1]];
          apndl o [1r o 1, 1r]

# hasrec: body => boolean, whether there is a recursive
# self-invocation in the body
Def hasrec (bu member recursive) o flatten

# hasname: <name, body> => boolean, whether the name is used
# in the body.
Def hasname member o [1, flatten o 1r]

# findrec: expr => <name1..namen>, the names of the functions
# that appear as (mutually) recursive
Def findrec uniqsort o selectlm o
            [aa (bu + 1) o (bu indicesl recursive), id] o
            flatten

Def uniqsort null -> id;
             (bu = 1) o length -> id;
             append o
                [uniqsort o uniqbefore, uniqsame,
                 uniqsort o uniqafter] o
                distl o [uniqpivot, id]

Def uniqbefore append o aa ( > -> tl ; _<> )
Def uniqafter  append o aa ( < -> tl ; _<> )
Def uniqsame   [1] o append o aa ( = -> tl ; _<> )
# the element in the middle of the array is
# often a good pivot, so use it
Def uniqpivot selectl o [(bur div 2) o length, id]
```

A.2 tosop

```
# tosop.fp: nonrecursive ffp to sop code translator
# this is a compiler from ffp to sop code. The ffp
# is supposed to have primitives and nothing else; codes
```

```
# are found in the directories srcdir and altdir, defined
# in config.fp

Def tosop optcomp o putcodes o 1r o 1

Def putcodes null -> id;
             (bu = tl) -> putcodes o _tll;
             atom -> getatomdef;
             (bu = cond) o 1 ->
                (bu = 4) o length ->
                    [sopcode o _cond, _".s",
                     putcodes o 2, aa putcodes o tl o tl];
                [sopcode o _cond, _".s",
                     putcodes o 2, [putcodes o 3,
                                    putcodes o (bu apndl cond) o
                                    tl o tl o tl]];
             (bu = lselect) o 1 -> putsel;
             (bu = rselect) o 1 -> putsel;
             (bur member  <bu, bul, bur>) o 1 -> putbu;
             aa putcodes

Def putbu and o [or o [(bu = eq), (bu = equatom)] o 2,
                 (bu = 0) o 1r] ->
                putcodes o _eq0;
          and o [or o [(bu = eq), (bu = equatom)] o 2,
                 (bu = 1) o 1r] ->
                putcodes o _eq1;
          and o [(bu = eq) o 2, atom o 1r] ->
                putcodes o [1, _eqatom, 1r];
          and o [(bu = plus) o 2, (bu = 1) o 1r] ->
                putcodes o _plus1;
          (bu = <bur, minus, 1>) ->
                putcodes o _minus1;
          (bu = bu) o 1 ->
                apndl o [sopcode o _bul, aa putcodes o tl];
          aa putcodes

Def putsel (bu = <rselect, 1>) -> sopcode o _rsel1;
           (bu = <rselect, 2>) -> sopcode o _rsel2;
```

```
           (bu = <lselect, 1>) -> sopcode o _lsel1;
           (bu = <lselect, 2>) -> sopcode o _lsel2;
           (bur < 1) o 1r ->
                error o [_"negative selector: ", 1r];
           error o [_"tosop can't handle selector: ", 1r]

# optcomp: if there are any instances of
# <".c", ... <".c", ... > >, raise all the elements in
# the sublist.
# <".c", ... <constant, ...> ...>, =>
#       <".c", ... <constant, ...>>
# <".c", ... id, ...>, => <".c", ... ...>
# <".c", X>, => X
Def optcomp null -> id;
            atom -> id;
            = o [_".c", 1] ->
                = o [append o aa raisecomp, id] o tl ->
                     aa optcomp;
                optcomp o apndl o
                [_".c", append o aa raisecomp o tl];
            aa optcomp

# raisecomp: <".c", x, y, z> => <x, y, z>
# raisecomp: <a, b, c> => <<a, b, c>>
Def raisecomp atom -> [id];
              = o [_".c", 1] -> tl;
              [id]

Def getatomdef number -> id;
               character -> id;
               (bu = T) -> _"1";
               (bu = F) -> _"0";
               sopcode

Def sopcode member o [id, aa 1 o stddefs] ->
                getstddef;
            goodfile o append o [explode, stdext] ->
                readfile o append o [explode, stdext];
            goodfile o append o [stddir, explode, stdext] ->
```

```
            readfile o append o [stddir, explode, stdext];
        goodfile o append o [altdir, explode, stdext] ->
            readfile o append o [altdir, explode, stdext];
        (bu error "unknown function: ")

# these two functions can be used to speed up the compiler
# by pre-loading all the standard definitions
Def stddefs _<<apteryxfiletype, none>>
Def getstddef error

# file name -> boolean
Def goodfile (bur member <datafile, text>) o filetype
```

A.3 SOP Assembler

```
# this sed program is an assembler for the
# sop language
# a line has the syntax
#
#   OPCODE  ; comments
# or
#   OPCODE d=n ls=n ; comments
# or
#   SUBST d=n ls=n p r, p r ; comments
#
s/;.*$//
s/#.*$//
# eliminate blanks and tabs
s/[  ]//g
# lengthen replacements from 1 to two characters,
# then remove the comma
s/SUBST.*,..$/&_/
s/SUBSTd=..s....,/&_/
s/,//
# assemble opcodes
s/SUBST/.u/g
s/BRACE/.b/g
s/SPLIT/.i/g
```

```
s/COMP/.c/g
s/SEL/.s/g
s/COPY/.y/g
s/NCELLS/.n/g
s/PLUS/.p/g
s/MINUS/.m/g
s/TIMES/.t/g
s/DIV/.d/g
s/GREATER/.g/g
s/SHOW/.w/g
# assemble parameter definitions
s/d=//g
s/ls=/=+/g
s/rs=/=-/g
s/ls~/n+/g
s/rs~/n-/g
# put processed line at beginning of pattern space,
# unless it is empty
x
H
# eliminate this line if it is empty
g
s/^\n//
h
# if at last line, print resulting assembly
$g
$s/\n$//
# add a compose if more than one input line
$s/^.*\n/< .c &/
$s/\n.*$/& >/
$p
# do not print line; wait until end of file
d
```

Appendix B

FP Benchmarks

B.1 Integer Sum

```
Def isum \/xplus

Def xplus +
```

B.2 n Queens Problem

```
# nqueens.fp: gives all solutions for placing n queens on an nxn
# chess board in such a way that they do not threaten each other
# Typical call:
# nqueens 8

# nqueens : n => vector of solutions, or nil
Def nqueens nmqueens o [id, id]

# nmqueens : <n, m> => list of n safe row positions for n queens
# on an n-column by m-row chess board. Precondition: n <= m
# Typical call:
# nmqueens : <2, 3> => <<1, 3>, <3, 1>>
Def nmqueens (bu = 1) o 1 -> aa [id] o iota o 2;
             append o aa (null -> id; [id]) o aa safe o
                append o aa distl o distr o
                [iota o 2, nmqueens o [(bur - 1) o 1, 2]]
```

```
# safe : <row, rowpositions> => <row | rowpositions> if safe,
# <> otherwise
# e.g. safe : <3, <1, 4, 7>> => <3, 1, 4, 7>,
#      safe : <3, <4, 1, 7>> => <>
Def safe \/and o aa saferow o aa apndl o pairpos o distl -> apndl ;
         _<>

# saferow : <col, row@col1, row@col> => whether a queen placed at
# (row@col1, 1) is safe from one at (row@col, col)
Def saferow \/and o aa (not o eqatom) o
            [tl, [1, - o tl], [1, neg o - o tl]]

# pairpos : <x1..xn> ==> <<1 x1>..<n xn>>
# Def pairpos null -> _<>; trans o [iota o length, id]
# pairpos.fp: a sop-efficient definition of pairpos

# pairpos : <a, b, c> => <<1, a>, <2, b>, <3, c>>
Def pairpos null -> id; \/cmppair o aa [[_1, id]]

# cmppair : <<<1, a>, <2, b>>, <<1, c>, <2, d>, <3, e>>> =>
#                <<1, a>, <2, b>, <3, c>, <4, d>, <5, e>>
Def cmppair append o
            [1,
             aa [+ o [1, 1 o 1r], 1r o 1r] o
# passing up <2, <1, c>>, <2, <2, d>>, <3, <5, e>>>
             distl o [1 o 1r o 1, 1r]]
```

B.3 Primes

```
# parprimes.fp
# primes number generator; prime x is removed from the
# list of candidates in parallel
Def parprimes sieve o tl o iota

Def sieve null -> id;
          apndl o [1, sieve o prfilter o [1, tl]]
```

```
Def prfilter null o 2 -> 2;
             append o aa elim o distl

Def elim (bu = 0) o mod o reverse -> _<>; [2]
```

Bibliography

[1] G. M. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. *Dokl. Akad. Nauk SSSR, Mathemat*, 146(2):263–266, 1962.

[2] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings, 1989.

[3] Marco Annaratone, Emmanuel Arnould, Robert Cohn, Thomas Gross, H. T. Kung, Monica Lam, Onat Menzilcioglu, Ken Sarocky, John Senko, and Jon A. Webb. Architecture of Warp. In *Spring COMPCON '87*, pages 264–267. IEEE, February 1987.

[4] John Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 16(8):613–641, August 1978. Turing Award Lecture.

[5] Scott B. Baden. *Run-Time Partitioning of Scientific Continuum Calculations Running on Multiprocessors*. PhD thesis, University of California at Berkeley, June 1987.

[6] Scott B. Baden. Programming abstractions for dynamically partitioning and coordinating localized scientific calculations running on multiprocessors. *SIAM Journal of Statistics and Scientific Computation*, January 1991.

[7] George H. Barnes, Richard M. Brown, Maso Kato, David J. Kuck, Daniel L. Slotnick, and Richard A. Stokes. The ILLIAC IV computer. *IEEE Transactions on Computers*, C-17(8):746–757, August 1968.

[8] Marsha J. Berger and Shahid H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

[9] S. Bettayeb, Z. Miller, and I. H. Sudborough. Embedding Grids into hypercubes. In *Proceedings of the AWOCS*, 1988. (must check).

[10] Edoardo S. Biagioni. FPC: A translator for FP. Technical Report TR88-027, Dept. of Computer Science, UNC-Chapel Hill, Chapel Hill, North Carolina, 27599-3175, May 1988.

[11] Edoardo S. Biagioni and Jan F. Prins. Scan directed load balancing for highly-parallel mesh-connected computers. In *Unstructured Scientific Computation on Scalable Multiprocessors*. MIT Press, 1991. (to appear).

[12] Guy E. Blelloch. *Scan Primitives and Parallel Vector Models*. PhD thesis, Electrical Engineering and Computer Science, MIT, November 1988.

[13] Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Transaction on Computers*, 38(11), November 1989.

[14] Guy E. Blelloch and Gary W. Sabot. Compiling collection-oriented languages onto massively parallel computers. In *Frontiers of Massively Parallel Computation (Frontiers '88)*, pages 575–585, November 1988.

[15] Bruce M. Boghosian. A survey of techniques for simulating partial differential equations with lattice gases. Technical Report CA 89-1, Thinking Machines Corporation, September 1989.

[16] Shekhar Borkar, Robert Cohn, George Cox, Sha Gleason, Thomas Gross, H. T. Kung, Monica Lam, Brian Moore, Craig Peterson, John Pieper, Linda Rankin, P. S. Tseng, Jim Sutton, John Urbanski, and Jon Webb. iWARP: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing'88*, pages 330–339, Orlando, Florida, November 1988.

[17] James M. Boyle and Terence J. Harmer. A practical functional program for the CRAY X-MP. Technical Report MCS-P159-0690, Mathematics and Computer Science Division, Argonne National Laboratory, July 1990.

[18] Mark Bromley, Steven Heller, Tim McNerney, and Guy L. Steele Jr. Fortran at ten gigaflops: The Connection Machine convolution compiler. In *ACM Sigplan '91 Conference on Programming Language Design and Implementation*, pages 145–156, June 1991.

[19] Philip M. Campbell, Edward A. Carmona, and David W. Walker. Hierarchical domain decomposition with unitary load balacing for electromagnetic particle-in-cell codes. In *Proceedings of the Fifth Distributed Memory Computing Conference – DMCC 1990*, pages 943–950, Charleston, South Carolina, April 1990.

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

[21] Shyamal Chowdhury. The greedy load sharing algorithm. *Journal of Parallel and Distributed Computing*, 9(1):93–99, June 1990.

[22] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, NJ, 1941.

[23] D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[24] Robert Cromartie and Stephen M. Pizer. Edge-affected context for adaptive contrast enhancement. In *12th International Conference on Information Processing in Medical Imaging (IPMI '91)*, July 1991.

[25] Haskell B. Curry. *Foundations of Mathematical Logic*. McGraw-Hill, 1963.

[26] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6), June 1990.

[27] Richard de Moliner. Kantenerhaltende Glättung. Semesterarbeit, ETH, Zurich, Switzerland, 1989.

[28] Karen M. Dragon and John L. Gustafson. A low-cost hypercube load-balance algorithm. In *Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 583–589, Monterey, CA, March 1989.

[29] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[30] American National Standard for Information Systems. Fortran 8X draft May 1989. *ACM SIGPLAN Fortran Forum*, December 1989.

[31] G. Fox. A tentative design for the 1024 node Caltech concurrent processor (CCP). Technical Report Hm 6, California Institute of Technology, Pasadena, CA, August 1982.

[32] Geoffrey C. Fox, Mark A. Johnson, Gregory A. Lyzenga, Steve W. Otto, John K. Salmon, and David W. Walker. *Solving Problems on Concurrent Processors. Volume I: General Techniques and Regular Problems.* Prentice-Hall, Englewood Cliffs, NY, 1988.

[33] Marc Garbey and David Levine. Massively parallel computation of conservation laws. Technical Report MCS-P85-0689, Mathematics and Computer Science Division, Argonne National Laboratory, June 1989.

[34] Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970.

[35] Martin Gardner. *Wheels, Life, and Other Mathematical Amusements.* W. H. Freeman, New York, 1983.

[36] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[37] Eddy Goldberg. Massively parallel processor introduced. *Computerworld*, 20(18), May 5 1986.

[38] Peter G. Harrison and Michael J. Reeve. The parallel graph reduction machine, ALICE. In Joseph H. Fasel and Robert M. Keller, editors, *Proceedings of the Workshop on Graph Reduction*, volume 279 of *Lecture Notes in Computer Science*, pages 181–202, Santa Fe, NM, 1986. Springer-Verlag.

[39] John L. Hennessy and David A. Patterson. *Computer Architecture — A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[40] W. Daniel Hillis. *The Connection Machine*. MIT Press, 1985.

[41] W. Daniel Hillis and Guy L. Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[42] Klaus Hinrichs. Implementation of the grid file: Design concepts and experience. *BIT*, 25:569–592, 1985.

[43] Klaus Hinrichs, Jürg Nievergelt, and Peter Schorn. Plane-sweep solves the closest pair problem elegantly. *Information Processing Letters*, 26:255–261, January 1988.

[44] Klaus H. Hinrichs and J. Nievergelt. The grid file: a data structure designed to support proximity queries on spatial objects. In M. Nagl and J. Perl, editors, *Proceedings of the International Workshop on Graph-theoretic Concepts in Computer Science (WG '83)*, pages 100–113. Trauner Verlag, Linz, 1983.

[45] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computation*, 8(4):325–339, 1990.

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

[47] Jia-Wei Hong. *Computation: Computability, Similarity and Duality*. Pitman, 1986.

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

[49] John Hughes. Why functional programming matters. Technical report, Dept. of Computer Science, Chalmers University of Technology and University of Göteborg, S-412 96 Göteborg, Sweden, November 1984.

[50] Oscar H. Ibarra, Ting-Chuen Pong, and Stephen M. Sohn. Parallel recognition and parsing on the hypercube. *IEEE Transactions on Computers*, 40(6), June 1991.

[51] INMOS, Ltd. *Occam 2 Reference Manual*. Prentice Hall International, 1988.

[52] Kenneth E. Iverson. *A Programming Language*. John Wiley and Sons, 1962.

[53] L. V. Kale. Comparing the performance of two dynamic load distribution methods. In *1988 International Conference on Parallel Processing, Volume I, Architecture*, pages 8–12, August 1988.

[54] H. T. Kung and C. E. Leiserson. Algorithms for vlsi processor arrays. In Carver Mead and Lynn Conway, editors, *Introduction to VLSI Systems*, pages 271–292. Addison-Wesley, 1980.

[55] Sun-Yuan Kung, K. S. Arun, Ron J. Gal-Ezer, and D. V. Bhaskar Rao. Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers*, C-31(11):1054–1066, November 1982.

[56] Bradley Clair Kuszmaul. Simulating applicative architectures on the Connection Machine. Master's thesis, Massachusetts Institute of Technology, May 1986.

[57] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. In *Proceedings of the 1977 International Conference on Parallel Processing*, pages 218–223, August 1977.

[58] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[59] Frank C. H. Lin and Robert M. Keller. The gradient model load balancing method. *IEEE Transactions on Software Engineering*, SE-13(1):32–38, January 1987.

[60] Thinking Machines. Connection Machine model CM-2 technical summary. Technical Report HA87-4, Thinking Machines Corp., April 1987.

[61] Gyula A. Magó. A cellular computer architecture for functional programming. In *VLSI, New Architectural Horizons. Digest of Papers. 20th COMPCON, Spring 80*, pages 179–187, San Francisco, CA, February 1980.

[62] Gyula A. Magó and Donald F. Stanat. The FFP Machine. In Veljko M. Milutinović, editor, *High-Level Language Computer Architectures*, pages 430–468. Computer Science Press, 1989.

[63] MasPar Computer Corporation, Sunnyvale, California. *MasPar Parallel Application Language (MPL) Reference Manual*, 1990.

[64] John McCarthy. *The LISP 1.5 Programmers' Manual.* MIT Press, 1962.

[65] Henk Meijer and Selim G. Akl. Optimal computation of prefix sums on a binary tree of processors. *International Journal of Parallel Programming*, 16(2):127–136, April 1988.

[66] Michael Metcalf and John Reid. *Fortran 90 Explained.* Oxford University Press, 1990.

[67] David M. Nicol. Rectilinear partitioning of irregular Data parallel computations. Technical Report 91-55, ICASE (Institute for Computer Applications in Science and Engineering), Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225, July 1991.

[68] D. Parkinson. The ICL distributed array processor DAP. In M. Hooper, editor, *Computational Methods in Classical and Quantum Physics*. Adv. Pub. Ltd., 1976.

[69] Dennis Parkinson. Experience in highly parallel processing using DAP. In Kai Hwang, Steven M. Jacobs, and Earl E. Swartzlander, editors, *International Conference on Parallel Processing (ICPP)*, pages 205–208, University Park, PA, August 19–22 1986. Penn State University, IEEE Computer Society.

[70] William Daniel Partain. *Graph Reduction Without Pointers*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC 27599-3175, December 1989.

[71] Douglas M. Pase and Allan R. Larrabee. The intel iPSC concurrent computer. In Robert G. Babb II, editor, *Programming Parallel Processors*, pages 105–124. Addison-Wesley, Reading, MA, 1988.

[72] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. Technical Report UCB/CSD 88/483, Computer Science Division, University of California, Berkeley, California, December 1988.

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

[74] Simon L. Peyton Jones, Chris Clack, Jon Salkild, and Mark Hardie. GRIP—a high performance architecture for parallel graph reduction. In Gilles Kahn, editor, *Functional Programming Languages and Computer Architecture*, number 274 in LNCS, pages 98–112, Portland, OR, September 1987. Springer-Verlag.

[75] Franco P. Preparata and Jean Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, May 1981.

[76] Hamilton Richards Jr. Applicative programming. *Systems Review*, 2(4):299–306, 1985.

[77] Gary Wayne Sabot. *The Paralation Model*. MIT Press, 1988.

[78] Vikram A. Saletore. A distributed and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks. In *Fifth Distributed Memory Computing Conference*, pages 994–999, April 8-12 1990.

[79] Joel Saltz, S. Petiton, H. Berryman, and A. Rifkin. Performance effects of irregular communication patterns on massively parallel multiprocessors. Technical Report 91-12, ICASE, Nasa Langley Research Center, MS 132C, Hampton, VA 23665, 1991. To appear in Journal of Parallel and Distributed Computing.

[80] Isaac D. Scherson and Peter F. Corbett. Communications overhead and the expected speedup of multidimensional mesh-connected parallel

processors. *Journal of Parallel and Distributed Computing*, 11:86–96, 1991.

[81] Charles L. Seitz, Jakov Seizovic, and Wen-King Su. The C programmer's abbreviated guide to multicomputer programming. Technical Report CS-TR-88-1, California Institute of Technology, Pasadena, CA 91126, January 1988.

[82] Sargur N. Srihari. Representation of 3D digital images. *ACM Computing Surveys*, 13:399–424, 1981.

[83] Guy L. Steele, Jr. and W. D. Hillis. Connection Machine Lisp: Fine-grained symbolic processing. In *Proceedings of the 1986 Symposium on Lisp and Functional Programming*, pages 279–297, 1986.

[84] Lixin Tao and Eva Ma. Simulating parallel neighboring communications among square meshes and square toruses. *The Journal of Supercomputing*, 5:57–71, 1991.

[85] Sheryl Tomboulian and Matthew Pappas. Indirect addressing and load balancing for faster solution to the mandelbrot set on simd architectures. In *Frontiers90*, October 1990.

[86] James J. Tuccillo. Numerical weather prediction on the Connection Machine. In *Proceedings of the Supercomputing '89 Supercomputer Applications*, volume II, pages 334–342, St. Petersburg, FL, April–May 1989. Fourth International Conference on Supercomputing (ICS '89).

[87] David A. Turner. A new implementation technique for applicative languages. *Software—Practice and Experience*, pages 31–49, January 1979.

[88] Clifford Walinsky and Deb Banerjee. A functional programming language compiler for massively parallel comp uters. In *Proceedings of the 1990 ACM conference on Lisp and Functional Programming*, 1990.

[89] David W. Walker. Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code. *Concurrency: Practice and Experience*, 2(4):257–288, December 1990.

[90] Yung-Terng Wang and Robert J.T. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217, March 1985.

[91] Ian Watson, Paul Watson, and Viv Woods. Parallel data-driven graph reduction. In J. V. Woods, editor, *Fifth Generation Computer Architectures*, pages 203–220, Amsterdam, 1986. North-Holland.

[92] C. Thomas White, Raj K. Singh, Peter B. Reintjes, Jordan Lampe, Bruce W. Erickson, Wayne D. Dettloff, Vernon L. Chi, and Stephen F. Altschul. BioSCAN: A VLSI-Based system for biosequence analysis. In *1991 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, MA, October 1991.