A. Parallaxis Language Definition

System	=	**SYSTEM** sys_ident ";" Declarations sys_ident ".".
Declarations	=	{ ConstantDecl | TypeDecl | ConfigDecl | ConnectDecl | VariableDecl | ProcedureDecl} block.
ConstantDecl	=	**CONST** { ident "=" ConstExpr ";" }.
TypeDecl	=	**TYPE** { ident "=" type ";" }.
ConfigDecl	=	**CONFIGURATION** config ";" { config ";" }.
config	=	conf_ident IntRange { "," IntRange }.
IntRange	=	"[" range "]".
range	=	int_ConstExpr [".." int_ConstExpr].
ConnectDecl	=	**CONNECTION** ({ TransferFunc ";" } | ";").
TransferFunc	=	out_direction ":" conf_ident "[" source { "," source } "]" ("->" | "<->") destination { "," destination }.
direction	=	ident ["(" integer ")"].
source	=	ident | integer.
destination	=	[discriminant] conf_ident "[" DestExprList "]" "." in_indirection.
discriminant	=	"{" bool_expr "}".
DestExprList	=	DestExpr { "," DestExpr }.
DestExpr	=	expr | int_ConstExpr ".." int_ConstExpr.
VariableDecl	=	(**SCALAR** | **VECTOR**) { ident { "," ident } ":" type ";" }.
ProcedureDecl	=	**PROCEDURE** proc_ident [FormalParams] ";"Declarations proc_ident ";".
FormalParam	=	"(" [parameters { ";" parameters }] ")"[":" (**SCALAR** | **VECTOR**) function_type].
parameters	=	(**SCALAR** | **VECTOR**) [**VAR**] ident { "," ident } ":" type.
block	=	**BEGIN** StatementSeq **END**.
StatementSeq	=	statement { ";" statement }.
statement	=	[assignment | ProcedureCall | IfSelection | CaseSelection | WhileLoop | RepeatLoop | LoopStatement | ForLoop | WithStatement | **EXIT** | **RETURN** [expr] | ParallelExec | Propagate | Send | Receive | Load | Store].
ParallelExec	=	**PARALLEL** selection StatementSeq **ENDPARALLEL**.
selection	=	[conf_ident] ["[" entry "]" { "," "[" entry "]" }].
entry	=	range { "," range } | expr [".." expr] | "*".
Propagate	=	**PROPAGATE** "." out_dirvar ["^" (integer | ident)]["." in_dirvar] "(" [vector_expr ","] vector_designator ")"[**REDUCE** "." operator_i-dent].
dirvar	=	ident ["(" expr ")"].

Load = **LOAD** selection "(" vector_designator "," scalar_designator["," length_designator] ")".

Store = **STORE** selection "(" vector_designator "," scalar_designator["," length_designator] ")".

Send = **SEND** port "(" vector_expr ")" **TO** port "(" vector_ident ")" [**REDUCE** "." operator_ident].

Receive = **RECEIVE** port "(" vector_designator ")" **FROM** port "(" vector_ident ")" [**REDUCE** "." operator_ident].

port = conf_ident "." dirvar.

Reduce = **REDUCE** "." operator_ident selection "(" expr ")".

assignment = designator ":=" expr.

designator = ident { "[" ExprList "]" | "." ident | "^" }.

ProcedureCall = proc_ident ["(" ExprList ")"].

FunctionCall = proc_ident "(" [ExprList] ")".

IfSelection = **IF** bool_expr **THEN** StatementSeq{ **ELSIF** bool_expr **THEN** StatementSeq }[**ELSE** StatementSeq] **END**.

CaseSelection = **CASE** expr **OF** case { "|" case } [**ELSE** StatementSeq] **END**.

case = CaseLabelList ":" StatementSeq.

CaseLabelList = CaseLabels { "," CaseLabels }.

CaseLabels = ConstExpr [".." ConstExpr].

WhileLoop = **WHILE** bool_expr **DO** StatementSeq **END**.

RepeatLoop = **REPEAT** StatementSeq **UNTIL** bool_expr.

LoopStatement = **LOOP** StatementSeq **END**.

ForLoop = **FOR** ident ":=" expr **TO** expr [**BY** ConstExpr] **DO** StatementSeq **END**.

WithStatement = **WITH** designator **DO** StatementSeq **END**.

type = SimpleType | ArrayType | RecordType | SetType | PointerType.

SimpleType = type_ident | enumeration | subrange.

enumeration = "(" const_ident { "," const_ident } ")".

subrange = "[" ConstExpr [".." ConstExpr] "]".

ArrayType = **ARRAY** SimpleType { "," SimpleType } **OF** type.

RecordType = **RECORD** FieldListSeq **END**.

FieldListSeq = FieldList { ";" FieldList }.

FieldList = [ident { "," ident } ":" type | **CASE** [ident] ":" SimpleType **OF** Variant { "|" Variant }[**ELSE** FieldListSeq] **END**].

Variant = CaseLabelList ":" FieldListSeq.

SetType = **SET OF** SimpleType.

PointerType = **POINTER TO** type.

ExprList = expr { "," expr }.

expr = SimpleExpr { relation SimpleExpr }.

relation = "=" | "<>" | "#" | "<" | ">" | "<=" | ">=" | **IN**.

SimpleExpr	=	["+" | "-"] term { AddOperator term }.
AddOperator	=	"+" | "-" | **OR**.
term	=	power { MulOperator power }.
MulOperator	=	"*" | "/" | **DIV** | **MOD** | **AND** | "&".
power	=	factor { "**" factor }.
factor	=	FunctionCall | Reduce | string | set | number | designator | array | structure | "(" expr ")" | **NOT** factor.
array	=	array_ident "(" ExprList ")".
structure	=	record_ident "(" ExprList ")".
string	=	" ' " { character | ' " ' } " ' " | ' " ' { character | " ' " } ' " '.
set	=	[type_ident] "{" [element { "," element }] "}".
element	=	ConstExpr [".." ConstExpr].
CExprList	=	[ConstExpr { "," ConstExpr }].
ConstExpr	=	SimpleConstExpr { relation SimpleConstExpr }.
SimpleConstExpr	=	["+" | "-"] ConstTerm { AddOperator ConstTerm }.
ConstTerm	=	ConstPower { MulOperator ConstPower }.
ConstPower	=	ConstFactor { "**" ConstFactor }.
ConstFactor	=	const_ident | number | record_ident "(" CExprList ")" |set | array_ident "(" CExprList ")" | "(" ConstExpr ")" | string | stdfct_ident "(" CExprList ")" | NOT ConstFactor.
number	=	integer | real.
integer	=	digit { digit }.
real	=	digit { digit } "." { digit } exponent | { digit } "." digit { digit } exponent.
exponent	=	["E" ["+" | "-"] digit { digit }].
ident	=	letter { letter | digit }.
character	=	letter | digit | "(* all characters but quotes *)".
digit	=	"0" | "..." | "9".
letter	=	"A" | "..." | "Z" | "a" | "..." | "z" | "_".

B. PARZ Language Definition

IntermediateProgram = **START** integer **PE** integer **PORTS** ControlVarDecl LocalVarDecl { Statement ";" [comment] } **STOP** .

ControlVarDecl = **SCALAR** declaration .

LocalVarDecl = **VECTOR** declaration .

declaration = { type integer | integer "(" declaration ")" | **U** "(" declaration { "," declaration } ")" } .

type = **B** | **C** | **I** | **R** .

Statement = [labeldef] stat .

labeldef = label ["!" [integer]] ["R"] ":" .

label = integer .

stat = assignment | computation | movestat | equalstat | **HALT** | **END** | **NOP** | connect | disconnect | gotostat | ifstat | whilestat | callstat | proc | **RETURN** | pushstat | popstat | parallel | propagate | send | receive | load | store | read | write | errorcall | openinput | **CLOSEINPUT** | openoutput | **CLOSEOUTPUT** | GraphFunc | GraphProc | debug | trace | notrace .

assignment = vardesc ":=" ["-" | **NOT**] VarConst | Sca_vardesc ":=" **STATUS** | vardesc ":=" **RANDOM** | vardesc ":=" **NEW** declaration | vardesc ":=" **INITSET** bits .

computation = vardesc ":=" VarConst ArithOperator VarConst | vardesc ":=" VarConst RelOperator VarConst | Sca_vardesc ":=" reduce | Vec_vardesc ":=" connected | vardesc ":=" FuncIdent VarConst | vardesc ":=" **ARCTANT** VarConst VarConst | vardesc ":=" **STRCMP** VarConst_string VarConst_string .

ArithOperator = "+" | "-" | "*" | "/" | "^" | **MOD** | **AND** | **OR** .

RelOperator = "=" | "<>" | "<" | "<=" | ">" | ">=" .

FuncIdent = **ABS** | **SQRT** | **EXP** | **LN** | **SIN** | **COS** | **TAN** | **ARCSIN** | **ARCCOS** | **ARCTAN** .

movestat = **MOVE** vardesc **TO** vardesc **AS** declaration .

equalstat = **EQUAL** vardesc vardesc **AS** declaration .

connect = **CONNECT** port **TO** port **AT** VarConst | **BICONNECT** port **TO** port **AT** VarConst .

disconnect = **DISCONNECT** [port] .

gotostat = **GOTO** label .

ifstat = **IF** VarConst [RelOperator VarConst] **CALL** label | **IF** Sca_VarConst [RelOperator Sca_VarConst] **GOTO** label .

whilestat = **WHILE** VarConst [RelOperator VarConst] **CALL** label .

callstat = **CALL** label .

proc = **PROC** integer [ControlVarDecl] [LocalVarDecl] .

pushstat = **PUSHS** Sca_VarConst | **PUSHV** VarConst .

popstat = **POPS** vardesc | **POPV** Vec_vardesc .

parallel	= **PARALLEL** (bits | Sca_VarConst) .
connected	= **CONNECTED IN** port [**OUT** port] | **CONNECTED OUT** port [IN port] .
propagate	= **PROPAGATE** Vec_vardesc **OUT** port **IN** port | **PROPAGATE** VarConst **TO** Vec_vardesc [**REDUCE** (RFuncIdent | label)] .
send	= **SEND** VarConst **TO** Vec_vardesc [**REDUCE** (RFuncIdent | label)] .
receive	= **RECEIVE** Vec_vardesc **FROM** VarConst [**REDUCE** (RFuncIdent | label)] .
load	= **LOAD** Vec_vardesc **WITH** Sca_vardesc [(**PE** integer) | (**AS** declaration)] .
store	= **STORE** Vec_vardesc **TO** Sca_vardesc [(**PE** integer) | (**AS** declaration)] .
reduce	= **REDUCE** (RFuncIdent | label) **OF** Vec_VarConst .
RFuncIdent	= **FIRST** | **LAST** | **SUM** | **PRODUCT** | **MAX** | **MIN** | **AND** | **OR** .
read	= **READ** vardesc [Sca_VarConst] .
write	= (**WRITE** | **WRITELN**) VarConst [Sca_VarConst [Sca_VarConst]] .
errorcall	= **ERROR** string .
openinput	= **OPENINPUT** Sca_Var Const_string .
openoutput	= **OPENOUTPUT** Sca_Var Const_string .
GraphFunc	= Sca_vardesc ":=" **GETPIXEL** Sca_VarConst Sca_VarConst | Sca_vardesc ":=" **OPENW** Sca_Var Const Sca_Var Const .
GraphProc	= **MOVETO** Sca_VarConst Sca_VarConst | **LINETO** Sca_VarConst Sca_Var Const | **SETPIXEL** Sca_VarConst Sca_Var Const | **WSIZE** Sca_vardesc Sca_vardesc | **SETCOLOR** Sca_vardesc | **SELECTW** Sca_VarConst | **CLOSEW** Sca_Var Const .
debug	= **DEBUG** vardesc string **AS** declaration .
trace	= **TRACE** vardesc string **AS** declaration .
notrace	= **NOTRACE** [vardesc] .
VarConst	= vardesc | readable_predec | constant | **ADDR** variable .
Sca_VarConst	= Sca_vardesc | Sca_readable_predec | Sca_constant | **ADDR** ScaVar .
Vec_VarConst	= Vec_vardesc | Vec_readable_predec | Vec_constant .
VarConst_string	= variable | string .
Sca_VarConst_string	= ScaVar | string .
port	= ScaVar | integer .
vardesc	= variable | indirect | writeable_predec .
Sca_vardesc	= ScaVar | Sca_indirect | writeable_predec .
Vec_vardesc	= VecVar | Vec_indirect .
indirect	= Sca_indirect | Vec_indirect .
Sca_indirect	= **S** type [integer ":"] "[" ScaVar "]" .

Vec_indirect = **V** type [integer ":"] "[" variable "]" .

variable = ScaVar | VecVar .

ScaVar = **S** type integer ":" integer .

VecVar = **V** type integer ":" integer .

writeable_predec = **MAXTRANS** .

readable_predec = Sca_readable_predec | Vec_readable_predec .

Sca_readable_predec = **ACTTRANS** | **DONE** | **TERMCH** | **SRESULT** .

Vec_readable_predec = **VRESULT** .

constant = Sca_constant | Vec_constant .

Sca_constant = **TRUE** | **FALSE** | char | **TERMS** | **EOL** | **CHR** "(" integer ")" |integer | **NIL** | **SIZE** "(" declaration ")" | real | string .

Vec_constant = **ID** .

comment = < any character sequence until end-of-line > .

char = " ' " (character | " ' ' ") " ' " .

integer = digit { digit } .

real = (integer "." | ([integer] "." integer)) [exponent] | integer exponent .

exponent = ("E" | "e") ["+" | "-"] integer .

string = ' " ' { character \ ' " ' | ' " " ' } ' " ' .

bits = { "0" | "1" | " " | < end-of-line > } .

character = < any character > .

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .

C. Execution Recording Structure

Execution recording to a named file may be set by calling the simulator with command-line option -r, or from command level within the simulator.

The record file contains the following information:
program filename:
    program: <name>
change of recording mode:
    mode: <number>
`<number>` =0 : no recording
    1 : short recording
    2 : extended recording

The output for each executed command will be:

1) short recording: `C <num> ;` or
   `Cu <num> ;` during execution of a `REDUCE` operation
   `C` : character identifying command class
   `num` : number of active PEs (for vector commands only)

2) extended recording: `<full_command> <activation> ;`
   `activation`: bitstring (0 and 1) indicating activation of each declared PE, may extend over several lines (for vector commands only)

<u>Summary of Command Characters:</u> (short recording only)

scalar commands:		*vector commands*	
a :	arithmetic command	A :	arithmetic command
b :	move, equal, status, string ass.	B :	move, equal, string assignment
c :	call	-	
d :	debug and trace commands	-	
e :	error-call	-	
f :	procedure / function	-	
g :	goto	-	
h :	halt, end	-	
i :	if_call, if_goto	I :	if_call
-		L :	load, store
-		M :	parallel
n :	new	N :	new
o :	I/O commands	O :	vector I/O
-		P :	propagate
r :	return	R :	return
s :	stack commands (push, pop)	S :	stack commands (push, pop)
-		T :	connect,biconnect,disconnect
-		U :	reduce
v :	comparison	V :	comparison
w :	while_call	W :	while_call
x :	nop (no operation)	-	
z :	assignment	Z :	assignment

D. Literature

[Babb 88] Robert Babb
Programming Parallel Processors
Addison-Wesley, 1988

[Barth 90a] Ingo Barth
Entwicklung eines Compilers für die parallele Programmiersprache Parallaxis
Studienarbeit Nr. 835, Universität Stuttgart, March 1990

[Barth 90b] Ingo Barth
Entwicklung eines Compilers für Parallaxis mit dynamischen Verbindungsstrukturen
Diplomarbeit Nr. 705, Universität Stuttgart, Nov. 1990

[Barth Bräunl Sembach 90] Barth, Bräunl, Sembach
Parallaxis User Manual
Computer Science Report, no. 3/90, Universität Stuttgart, March 1990

[Bräunl 89a] Thomas Bräunl
Parallaxis: A Flexible Parallel Programming Environment for AI Applications
Applications of Artificial Intelligence VII, Orlando Florida, March 1989, pp. 275 (11)

[Bräunl 89b] Thomas Bräunl
A Specification Language for Parallel Architectures and Algorithms
Fifth International Workshop on Software Specification and Design, Pittsburgh Pennsylvania, May 1989, pp. 49 (3)

[Bräunl 89c] Thomas Bräunl
Structured SIMD Programming in Parallaxis
Structured Programming, vol. 10, no. 3, July 1989, pp. 121 (12)

[Bräunl 89d] Thomas Bräunl
Ein Modell der Parallelen Programmierung mit funktionaler Spezifikation der Netzwerk-Topologie
Ph.D. Thesis, Universität Stuttgart, Sep. 1989, pp. (185)
Identical to:
Massiv parallele Programmierung mit dem Parallaxis-Modell
Informatik-Fachberichte Nr. 246, Springer-Verlag 1990.

[Bräunl 90] Thomas Braunl
Transparent Massively Parallel Programming with Parallaxis
ISSM International Conference on Parallel and Distributed Computing and Systems, New York NY, Oct. 1990, pp. 40 (2)

[Bräunl 91a] Thomas Bräunl
Parallaxis Modula massiv parallel
c't Magazin für Computertechnik, Heft 6, June 1991, pp. 34-39 (5)

[Bräunl 91b] Thomas Bräunl
Designing Massively Parallel Algorithms with Parallaxis
The 15th Ann. International Computer Software and Applications Conference, compsac91, Kogakuin University, Tokyo Japan, Sep. 1991, pp. 612-617 (6)

[Engelhardt 91] Stefan Engelhardt
Automatische Übersetzung einer massiv parallelen Programmiersprache für sequentielle und parallele Rechnerarchitekturen
Diplomarbeit Nr. 791, Universität Stuttgart, June 1991

[Krauskopf 90] Karsten Krauskopf
Ein massiv paralleles Verfahren zur Stereobildauswertung
Diplomarbeit Nr. 707, Universität Stuttgart, Nov. 1990

[Liebelt 91] Sabine Liebelt
Entwicklung und Untersuchung von massiv parallelen Hidden-Surface- und Raytracing-Algorithmen
Diplomarbeit Nr. 769, Universität Stuttgart, Jan. 1991

[Inmos 84] Inmos Limited
Occam Programming Manual
Prentice Hall International, 1984

[Rose Steele 87] Rose, Steele
C: An Extended C Language for Data Parallel Programming*
Thinking Machines Corporation, Technical Report, PL87-5, 1987

[Schulze Christ 90] Schulze, Christ
Ein Tool zur Visualisierung von Parallaxis Prozessor-Strukturen
Softwarepraktikum 2, Universität Stuttgart, Aug. 1990

[Sembach 90a] Frank Sembach
Entwicklung eines Simulators für die parallele Zwischensprache PARZ
Studienarbeit Nr. 834, Universität Stuttgart, March 1990

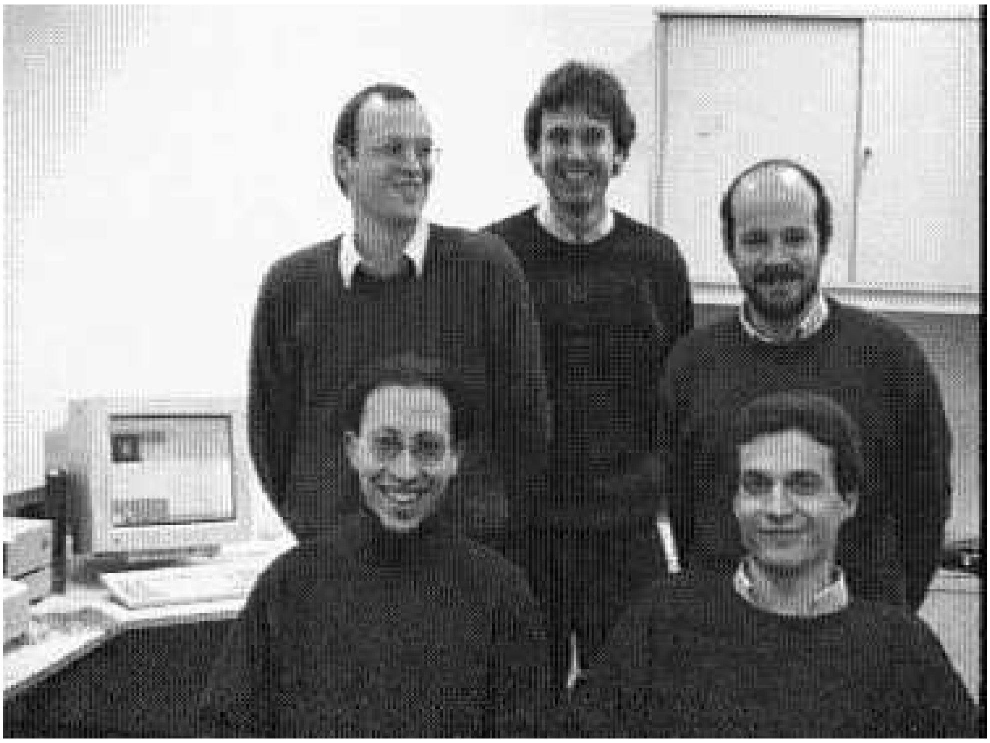
[Sembach 90b] Frank Sembach
Entwicklung eines symbolischen Debuggers für das parallele Sprachensystem Parallaxis/PARZ
Diplomarbeit Nr. 706, Universität Stuttgart, Nov. 1991

[Verba 90] Reinhard Verba
Massiv-parallele Algorithmen zur Lösung von Problemen der linearen Algebra
Studienarbeit Nr. 904, Universität Stuttgart, Nov. 1990

[Walter 91] Volker Walter
Entwurf von massiv parallelen Simulated Annealing Algorithmen
Studienarbeit Nr. 925, Universität Stuttgart, Jan. 1991

[Wirth 83] Niklaus Wirth
Programming in Modula-2
Springer-Verlag, 1983

E. About the Team



back: Ingo Barth, Thomas Bräunl, Frank Sembach
front: Michael Ancutici, Stefan Engelhardt

Dr. Thomas Bräunl studied Computer Science at the Universität Kaiserslautern, Germany (Diplom 1986), and at the University of Southern California (USC), Los Angeles, USA (M.S. 1987). He received a Ph. D. in Computer Science in 1989 from the Universität Stuttgart, in the area of programming models for parallel computation, where he is now lecturing. He initiated and leads the Parallaxis Parallel Programming project. His areas of interest include parallelism, AI, robotics, and computer graphics.

Ingo Barth studied Coputer Science at the Universität Stuttgart, where he is now an assistant. He received his Diplom degree in 1990 and implemented the Parallaxis compiler. His areas of interest are parallel languages, compilers, micro processors, and distributed computing.

Frank Sembach studied Computer Science at the Universität Stuttgart, where he is now an assistant. He received his Diplom in 1990 and implemented the PARZ simulator and debugger. His areas of interest are micro processors, programming languages, compilers, parallel computing, and distributed computing.

Stefan Engelhardt studied Computer Science at the Universität Stuttgart, where he is now an assistant. He received his Diplom in 1991 and implemented the PARZ-to-C and PARZ-to-MPL compilers. His areas of interest are compilers, programming languages, data bases, and parallel computing.

INDEX

V