



A GENERALIZED EXECUTION MODEL FOR NESTED  
DATA-PARALLEL COMPUTING

By  
Dean Engelhardt  
April 18, 1997

A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN THE DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF ADELAIDE

# Contents

Preface	xii
Declaration	xiii
Acknowledgments	xiv
<b>1 Introduction</b>	<b>1</b>
1.1 Hardware Implementations of Data Parallel Execution . . . . .	3
1.1.1 SIMD Hardware Solutions . . . . .	4
1.1.2 SPMD Hardware Solutions . . . . .	6
1.2 Data-Parallel Languages . . . . .	8
1.2.1 Historical Development of DP Languages . . . . .	9
1.3 Limits of Traditional Data Parallelism . . . . .	17
1.4 Thesis Overview . . . . .	18
<b>2 Nested Data-Parallelism</b>	<b>21</b>
2.1 Concepts of Nested Data-Parallelism . . . . .	22
2.1.1 Nested Data-Parallel Programming Languages . . . . .	23
2.2 NDP Implementations via Serialization . . . . .	28
2.2.1 Perils of Serialization . . . . .	30
2.3 Structure Flattening . . . . .	34
2.3.1 Conditional Execution . . . . .	37
2.4 NDP as an Efficient Basis for Irregular Scientific Computing . . . . .	38
<b>3 Designing a New Paradigm for NDP</b>	<b>42</b>
3.1 Basic Concepts . . . . .	43
3.1.1 Vectors . . . . .	43

3.1.2	The Distributed Memory Machine . . . . .	44
3.1.3	Partitioning Vectors Across the Distributed Machine . . . . .	45
3.2	Modelling Data-Parallelism . . . . .	47
3.2.1	The DP Operator as a Specialized Co-operative Computation . . . . .	47
3.2.2	Executing a DP Operation on a Distributed Memory Machine . . . . .	50
3.2.3	A Synchronizing Distributed Execution for DP Operators . . . . .	59
3.2.4	Avoiding Deadlock by Responsibility Aggregation . . . . .	65
3.3	Extending to the Nested Data-Parallel Case . . . . .	71
3.3.1	Modelling NDP Operations . . . . .	71
3.3.2	Macro-Responsibilities for NDP . . . . .	74
3.4	A Multi-threaded Alternative . . . . .	81
3.4.1	Modelling Multi-Threaded Execution . . . . .	82
3.4.2	Multi-Threaded Execution of Flat DP Operations . . . . .	83
3.4.3	Multi-Threaded Execution of NDP Operations . . . . .	84
3.5	Summarizing the Mathematical Analysis . . . . .	86
<b>4</b>	<b>Applying the Mathematical Analysis</b>	<b>88</b>
4.1	Modelling Existing DP Paradigms . . . . .	89
4.1.1	Modelling SIMD Hardware: the CM-2 . . . . .	89
4.1.2	Modelling SPMD Hardware: the CM-5 . . . . .	90
4.2	Reasoning About Performance for Abstract Model Implementations . . . . .	93
4.2.1	Principal Cost Factors . . . . .	94
4.2.2	Implementations on a SPMD machine . . . . .	96
<b>5</b>	<b>A Framework for Multi-threaded Expression</b>	<b>101</b>
5.1	A Threaded Abstract Machine . . . . .	102
5.1.1	Execution Model and the Task Pool . . . . .	103
5.1.2	The Result Pool . . . . .	104
5.1.3	Data Types, Aggregates and Partitioning . . . . .	105
5.1.4	Communication . . . . .	105
5.1.5	Abstract Machine Instruction Set . . . . .	106
5.1.6	Some Useful Set Operations . . . . .	108
5.1.7	An Example $\Omega$ -Execution . . . . .	111
5.2	Threaded Specification of Data-Parallel Operations . . . . .	116
5.2.1	Issues in the Specification of Data-Parallel Threads . . . . .	117

5.2.2	Auxiliary Threads . . . . .	122
5.2.3	The Map Thread . . . . .	126
5.2.4	The Reduce Thread . . . . .	131
5.2.5	The Scan Thread . . . . .	136
5.3	Summarizing the Multi-threaded Framework . . . . .	142
<b>6</b>	<b>AMAM: An Implementation of the Model</b>	<b>144</b>
6.1	A Multi-Threading Environment for the CM-5 . . . . .	145
6.1.1	AMAM Threads and Scheduling . . . . .	147
6.1.2	Storage Model for Partitioned Vectors . . . . .	154
6.1.3	The Communications Subsystem . . . . .	157
6.1.4	A Hybrid Model for Keys . . . . .	158
6.1.5	AMAM System Calls . . . . .	159
6.2	Optimizations for NDP use of the AMAM . . . . .	161
6.2.1	Multiple-Direct Keys . . . . .	162
6.2.2	An Alternative To Barrier Synchronization . . . . .	164
6.2.3	Working With Participant Sets . . . . .	169
6.2.4	Inverting Partitioning Functions . . . . .	170
6.3	Overview of Related Architectures . . . . .	171
6.3.1	P-RISC and TAM . . . . .	171
6.3.2	Charm . . . . .	177
6.3.3	Pebbles . . . . .	178
6.3.4	Nomadic Threads and the I-Structure Software Cache . . . . .	178
6.3.5	Multi-Threaded Hardware Architectures . . . . .	180
6.4	$\Omega$ -Threads in AMAM . . . . .	181
6.4.1	Vector Dereference . . . . .	182
6.4.2	Nestable Data-Parallel MAP . . . . .	182
6.5	Summary of the Architecture . . . . .	189
<b>7</b>	<b>Case Study: Implementing a Simple Nested Data-Parallel Language on the AMAM</b>	<b>192</b>
7.1	An Overview of the Language Adl . . . . .	193
7.1.1	Program Structure . . . . .	193
7.1.2	Expressions in the Language . . . . .	195
7.1.3	Data Parallel Operations . . . . .	196

7.2	Building an Execution Environment for Adl . . . . .	199
7.2.1	A Thread Library to Support NDP . . . . .	200
7.3	Compiling Adl into Threads . . . . .	204
7.3.1	Type Checking and Simple Optimizations . . . . .	206
7.3.2	Data Partitioning . . . . .	206
7.3.3	Thread Production . . . . .	207
7.3.4	Code Generation . . . . .	211
7.4	Summary of the Language Implementation . . . . .	218
<b>8</b>	<b>Experimental Evaluation of the AMAM</b>	<b>220</b>
8.1	Qualitative Visual Analysis of AMAM’s NDP Execution . . . . .	220
8.1.1	Instrumentation of the AMAM . . . . .	221
8.1.2	Experimental Suite . . . . .	223
8.1.3	Space-Time Visualization of Communication . . . . .	224
8.1.4	Visualization of Aggregate Communications Volume . . . . .	229
8.1.5	Visualization of Processor Utilization . . . . .	230
8.1.6	Related Work . . . . .	232
8.1.7	Summary of Qualitative Results . . . . .	232
8.2	Quantitative Analysis of Adl Performance . . . . .	234
8.2.1	Basic Data-Parallel Performance . . . . .	234
8.2.2	Flat Irregular Program Performance . . . . .	239
8.2.3	Simple Nested DP Performance . . . . .	240
8.2.4	A Simple Finite Element Mesh Computation . . . . .	246
8.2.5	A Molecular Chemistry Kernel . . . . .	250
8.2.6	Summary of Quantitative Results . . . . .	252
<b>9</b>	<b>Conclusions and Scope for Future Work</b>	<b>254</b>
9.1	Directions for Future Research . . . . .	256
9.1.1	AMAM . . . . .	256
9.1.2	The Adl Compiler . . . . .	257
9.1.3	Performance Evaluation and Visualization . . . . .	259
9.2	Contributions . . . . .	259
9.3	Final Observations . . . . .	260

<b>A</b>	<b>Visual Partitioning Specification</b>	<b>262</b>
A.1	Representing Data Placement in the Adl System . . . . .	263
A.1.1	Partitioning a Vector Nest . . . . .	266
A.2	Visual Specification . . . . .	271
A.2.1	A Conceptual Framework for Abstract Specification . . . . .	271
A.2.2	Defining of Simple Partitioning: A Sample PFN Session . . . . .	275
A.2.3	Defining a Partitioning Scheme . . . . .	279
A.3	Generating Partitioning Functions . . . . .	286
A.3.1	Generating Partitioning Functions by Simple Template . . . . .	286
A.3.2	Generating Partitioning Functions by Global Template . . . . .	289
A.3.3	Generating Partitioning Schemes . . . . .	293
A.3.4	Extending to the Arbitrary Nesting Case . . . . .	294
A.3.5	Extended Code Generation Example . . . . .	295
A.4	Related and Future Work . . . . .	300
<b>B</b>	<b>Source Code for Evaluated Programs</b>	<b>302</b>
B.1	Flat Data-Parallel Codes . . . . .	302
B.1.1	Flat map Program . . . . .	302
B.1.2	Flat scan Program . . . . .	304
B.1.3	Irregular Vector Index Program . . . . .	305
B.2	Nested Data-Parallel Codes . . . . .	307
B.2.1	Nested map-scan Program . . . . .	307
B.2.2	meshcomp: A Simple Finite Element Mesh-Based Computation	309
B.2.3	forces: A Molecular Chemistry Kernel . . . . .	327
	<b>Bibliography</b>	<b>332</b>

# Preface

Of the many attempts to simplify the task of programming today's parallel multiprocessor architectures, the most successful is the paradigm of Data-Parallelism. Much of the appeal of the model lies in its high-level view of the parallel machine coupled with its efficient mapping to a large class of real-world architectures.

While the Data-Parallel model has been very effective at allowing highly parallel specification of operations across rectangular arrays, its applicability to programs which use less regular data structures is very limited. This factor compromises the utility of the model: many important scientific problems based heavily upon irregular data structures (e.g., sparse matrix computations, finite element irregular mesh codes) cannot be efficiently parallelized using existing Data-Parallel techniques.

The purpose of this thesis is to explore extensions to the Data-Parallel model which make it more amenable to these kinds of irregular problems. In particular we consider the paradigm of Nested Data-Parallelism proposed by Blelloch, investigating techniques for efficiently mapping programs with such nested parallelism onto traditional multiprocessor architectures. Through mathematical analysis we derive a novel implementation of such features which makes use of a multi-threaded model of computation upon each processor of a multiprocessor machine. We describe a realization of this execution model which we have constructed for the Thinking Machines CM-5 and detail how we have used this system as the basis for the implementation of a simple language with Nested Data-Parallel features. To demonstrate the validity of our approach we benchmark the performance of several programs compiled using this language system, comparing the figures obtained with those for equivalent programs compiled in the CM Fortran and NESL systems. For the real-world irregular codes we examine, our unoptimized compiler output running on the CM-5 threading system delivers performance that is competitive with both existing systems, surpassing each in certain situations.