

A PROGRAMMING SYSTEM FOR THE IMAGINE MEDIA PROCESSOR

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Peter Mattson

March 2002

© Copyright by Peter Mattson 2002

All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Prof. William J. Dally
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Prof. Dawson Engler

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Prof. Monica Lam

Approved for the University Committee on Graduate Studies:

Abstract

Media processing applications, such as image-processing, signal processing, and graphics, motivate new processor architectures that place new burdens on the compiler. These applications demand very high arithmetic rates and data bandwidth, but lack data reuse. The Imagine Media Processor (Imagine) is a new architecture that introduces two innovations to meet these demands. To support a large number of arithmetic logic units (ALUs), it uses multiple register files connected to the ALUs by shared buses and register file ports. Instead of a cache, which relies on data reuse, it uses a large on-chip memory called the stream register file (SRF) to hold sequences of records called *streams*. The compiler is responsible for allocating the shared interconnect and SRF.

This thesis describes a programming system that enables efficient application development for Imagine and other architectures that include these innovations. It introduces an implementation of the stream programming model. The stream programming model divides a media processing application into a series of kernels, computation-intensive functions that operate on streams, and a stream program that defines the high-level control- and data flow between kernels. This thesis introduces KernelC, a C-like language used to write kernels. It describes a KernelC compiler that uses a new technique called communication scheduling to allocate the shared buses and register file ports and manage data flow between multiple functional units and register files. This thesis introduces a language extension called StreamC used with C++ to write stream programs. It describes a StreamC compiler that uses a new technique called stream scheduling to allocate the stream register file and determine when to load and store streams.

The Imagine programming system has been used to implement sophisticated, high-performance applications for Imagine such as stereo depth extraction, MPEG2 encoding, and polygon rendering. Experimental results for a set of image processing, signal processing, and graphics benchmarks show that communication scheduling delivers schedule lengths for an Imagine Media Processor with multiple partitioned register files and shared interconnect that are within 1% of the same architecture with one multi-ported register file that is estimated to be eight times larger. For these benchmarks, stream scheduling allocates the SRF as well or better than experienced Imagine programmers can by hand using assembly language.

Acknowledgements

This thesis would not have been possible without the support of many exceptional people.

To them, my thanks:

My advisor, Professor Bill Dally, for a fascinating problem, good advice, constructive criticism, support, and flexibility.

The other members of my committee, Professor Monica Lam and Professor Dawson Engler, for their timely and constructive criticism.

The members of my research group for their persistent use of the Imagine programming system as it was being developed and their many suggestions on how to it could be improved.

The understanding people at Reservoir Labs for providing an interesting place to work, and for being patient.

Professor Yongmin Kim, for wonderful challenges which started me down this path as an undergraduate.

My family, without whom I would not be the person I am.

My wife, Allissa, for being a constant source of happiness and encouragement, and so much more.

Table of Contents

Abstract	v
Acknowledgements	vii
Table of Contents	ix
List of Figures	xiii
1 Introduction	1
1.1 The Problem.....	1
1.2 Contributions	7
1.3 Thesis Roadmap.....	8
2 Background	11
2.1 VLIW Scheduling.....	11
2.2 Streams.....	14
2.3 Imagine Media Processor.....	15
2.3.1 Stream controller/host interface.....	15
2.3.2 Stream register file.....	15
2.3.3 Memory system.....	16
2.3.4 Processor core	17
2.4 Summary.....	17
3 Stream Programming Model	19
3.1 Overview.....	20
3.2 KernelC.....	23
3.2.1 Structured data access.....	23
3.2.2 Limited control flow	26
3.2.3 Additional data types and math operators.....	27
3.2.4 SIMD processing support	28
3.3 StreamC	29
3.3.1 Streams.....	31
3.3.2 Microcontroller variables.....	34
3.3.3 Kernels.....	35
3.3.4 Stream copies and transfers	36
3.3.5 Data-dependence annotations	37
3.4 Summary.....	39
4 Communication Scheduling	41
4.1 Motivation.....	42
4.2 Overview.....	45

4.2.1	Role in a VLIW scheduler	48
4.3	Algorithm.....	50
4.4	Implementation	59
4.4.1	Determining valid stubs for a communication.....	59
4.4.2	Finding a permutation of stubs	61
4.4.3	Scheduling copy operations	64
4.5	Performance	64
4.6	Summary	65
5	KernelC Compiler	67
5.1	Pre-scheduling	68
5.1.1	Parsing	68
5.1.2	Control flow analysis	68
5.1.3	Data flow analysis.....	69
5.1.4	Dependency analysis.....	71
5.1.5	Stream input/output ordering.....	73
5.1.6	Scratch pad access ordering.....	75
5.2	Scheduling	75
5.2.1	Basic block ordering	76
5.2.2	Scheduling algorithm.....	77
5.2.3	Operation prioritization.....	81
5.2.4	Functional unit assignment	85
5.2.5	Randomization	88
5.3	Post-scheduling.....	88
5.3.1	Register allocation	88
5.3.2	Machine code generation	91
5.4	Summary.....	92
6	Stream Scheduling	93
6.1	Motivation.....	94
6.2	Overview.....	98
6.3	Algorithm.....	105
6.3.1	Completion.....	117
6.4	Special Cases	117
6.4.1	Variable length and variable bounds streams	119
6.4.2	Indexed Streams.....	119
6.4.3	Other Stream Operations	120
6.5	Summary.....	120
7	StreamC Compiler and Dispatcher	123
7.1	StreamC Compiler	125
7.1.1	Profile compilation	125
7.1.2	Resource allocation.....	129
7.1.3	Operation translation.....	135
7.1.4	Issue slot assignment and dependency analysis.....	138
7.2	Run-time Dispatcher	141
7.2.1	Imagine operation dispatching.....	141
7.2.2	Host/Imagine data transfers	144

7.2.3 Double-buffering	145
7.3 Optimizations.....	147
7.3.1 Strip-mining.....	147
7.3.2 Software-pipelining	149
7.4 Summary.....	152
8 Evaluation	153
8.1 KernelC Compiler.....	153
8.1.1 Evaluation Architectures.....	153
8.1.2 Evaluation Benchmarks	155
8.1.3 Results.....	155
8.1.4 General analysis.....	162
8.1.5 Benchmark analysis	165
8.2 StreamC Compiler	168
8.2.1 Methodology	168
8.2.2 Benchmarks	171
8.2.3 Results.....	175
8.2.4 General analysis.....	176
8.2.5 Benchmark analysis	180
8.3 Summary.....	184
9 Conclusion	185
9.1 Summary.....	185
9.2 Future Work.....	186
9.2.1 Communication scheduling with register pressure.....	186
9.2.2 Multiprocessor systems.....	187
9.3 Epilogue.....	187
Appendix A: Detailed Results	189
Bibliography	191

List of Figures

1-1	Simplified diagram of a conventional processor	2
1-2	Simplified diagram of an Imagine processor	3
1-3	Simplified polygon rendering using stream programming model	4
1-4	Communication scheduling assigns each communication to a route	5
1-5	Stream scheduling assigns each stream access to a buffer in the SRF	6
2-1	Imagine Media Processor.....	16
3-1	Stream programming model	20
3-2	Simplified polygon rendering using stream programming model	21
3-3	Code transformation to stream programming model	22
3-4	Abbreviated definition of KernelC	25
3-5	Example with data access highlighted	26
3-6	Example with limited control flow highlighted	27
3-7	Kernel to brighten an 8-bit grayscale image	28
3-8	Kernel to sum a stream of integers	29
3-9	C++ declarations for StreamC components	31
3-10	A basic stream is an array of records	31
3-11	A derived stream is a subset of the records in a basic stream defined by a start, end, and access pattern	32
3-12	Sequential access pattern includes every record	33
3-13	Strided access pattern includes every strideth record	33
3-14	Indexed access pattern includes records with positions given by index stream ...	33
3-15	Example with streams highlighted.....	33
3-16	Derived streams mapped to the underlying basic stream	35
3-17	Extended example with microcontroller variables highlighted	36
3-18	Kernel call in example	36
3-19	Copies and transfers.....	37
3-20	Simple example of stream copies and transfers.....	37
3-21	Data dependent control flow annotations	38
3-22	Example of data dependence annotations	39
4-1	Example code fragment	43
4-2	Single register file architecture	43
4-3	Schedule for single register file architecture	43
4-4	Shared interconnect architecture	44
4-5	Schedule for shared interconnect architecture	45
4-6	Communication scheduling assigns each communication to a route	46
4-7	Communications in motivating example	46

4-8	Routes for communications in motivating example	47
4-9	Composition of a route	48
4-10	Composition of route for communication of a from operation 1 to operation 4 ..	49
4-11	Flowgraph for a simple scheduler with communication scheduling	50
4-12	Incremental composition of a route (left to right).....	51
4-13	Valid write stubs	52
4-14	Valid read stubs	53
4-15	Permutation of write stubs when scheduling operation 1	54
4-16	Permutation of write stubs when scheduling operation 2	54
4-17	Operation 3 cannot be scheduled due to stub conflicts.....	54
4-18	Route for communication of b from operation 2 to operation 4.....	55
4-19	Copy operation code transformation.....	56
4-20	A copy operation effectively splits original communication into two communications	56
4-21	Copy ranges based on location of read operation	57
4-22	Route for communication of a from operation 1 to operation 4	58
4-23	Copy connected architecture constraint.....	60
4-24	Pseudocode for stub permutation search	63
5-1	Example kernel in KernelC and corresponding primitive operations.....	69
5-2	Control flow graph.....	70
5-3	Communication graph.....	72
5-4	Dependency graph	73
5-5	Dependency graph before and after stream input/output ordering.	75
5-6	Example architecture	76
5-7	Scheduling algorithm flowchart.....	78
5-8	Cycle-driven schedule.....	79
5-9	Operation 5 can't be scheduled due to communication conflict.....	79
5-10	Operation-driven schedule.....	80
5-11	Operation 5 can be scheduled without conflict.....	80
5-12	Optimal schedule	81
5-13	Possible communication conflict with a two-phase scheduler	82
5-14	Functional unit usage calculation	84
5-15	Communications to and from operation 4	87
5-16	Communications with routes through left register file of adder 0.....	89
5-17	Routes through left register file of adder 0	90
5-18	Cycle 3 of final schedule	91
5-19	Driver encoding	92
5-20	Resource encoding.....	92
6-1	Available bandwidth in Imagine.....	94
6-2	Example stream program	96
6-3	SRF allocation graph for stream caching.....	97
6-4	SRF allocation graph for stream scheduling.....	99
6-5	Stream scheduling assigns each stream access to a buffer in SRF	99
6-6	Profile of all stream accesses for the example program	100
6-7	Buffers for each stream access in the example program.....	100
6-8	Double-buffered stream read	101

6-9	Double buffered stream write	101
6-10	Stream operations with accesses assigned to different buffers	102
6-11	Stream operations with accesses assigned to the same buffer	102
6-12	Buffers arranged to allow parallel execution and memory accesses	103
6-13	Buffers in two-dimensions for the example program	104
6-14	Example program with double-buffering	106
6-15	Example program with initial buffers	108
6-16	Write to sx does not reach read of s	109
6-17	Write to sx reaches read of s	109
6-18	Example program with required memory loads	110
6-19	Buffer divided in space	111
6-20	Buffer divided in time	111
6-21	Buffers that can be divided in the example program	111
6-22	Example program with divided buffers	112
6-23	Example program with load and store shadows	113
6-24	Order of positioning for buffers in the example program	114
6-25	First arrangement of buffers for the example program	115
6-26	Example program showing the buffer that is divided in time	117
6-27	Revised order of positioning for buffers in the example program	118
6-28	Final arrangement of buffers for the example program	118
6-29	write to sv reaches read of s1	119
6-30	sv doesn't cover s1 so write to s also reaches read of s1	119
6-31	if s is produced closer, streamCopy(s, t) saves s as t	121
6-32	if t is used closer, streamCopy(s, t) loads s as t	121
7-1	Roles of the StreamC compiler and run-time dispatcher	124
7-2	Stream program EyeMatch	127
7-3	Profile of EyeMatch(--, 200, 200, --, 100, 100)	128
7-4	Microcode store allocation for EyeMatch	131
7-5	SRF allocation for EyeMatch	132
7-6	Memory allocation for EyeMatch	133
7-7	Control register allocation for EyeMatch	135
7-8	Imagine operations by purpose	136
7-9	Stream operation specific Imagine operations	137
7-10	Imagine operations for first four stream operations in EyeMatch	138
7-11	RAW masks for first fifteen Imagine operations in EyeMatch	142
7-12	Imagine operations for double-buffered stream write to image	146
7-13	Basic and strip-mined data flow	148
7-14	Strip-mined loop	149
7-15	Loop with sequential memory access	150
7-16	Software-pipelined loop	150
7-17	Software-pipelining groups	151
7-18	Moving an operation between groups to reduce run time	151
7-19	Input to StreamC compiler software-pipelining	152
7-20	Output from StreamC compiler software-pipelining	152
8-1	Single register file architecture	154
8-2	Distributed register file architecture	154

8-3	Representative latencies.....	155
8-4	KernelC benchmarks.....	157
8-5	Relative increase in schedule length for DRF over single RF	158
8-6	Increase in schedule length for DRF over single RF in cycles.....	159
8-7	Relative difference between DRF schedule length and CLB	159
8-8	Relative increase in operation count for DRF over single RF due to copies.....	160
8-9	Relative increase in register demand due to duplication	161
8-10	Relative increase in register demand due to load imbalance	161
8-11	Relative increase in register demand	162
8-12	Single register file schedule for convfx7x7 (29 cycles).....	167
8-13	Distributed register file schedule for convfx7x7 (30 cycles).....	167
8-14	Single register file schedule for spansprep (64 cycles).....	169
8-15	DRF schedule for spansprep (70 cycles)	170
8-16	Equivalent StreamC and Macrocode	171
8-17	StreamC benchmark run times.....	176
8-18	Strip size	177
8-19	Memory traffic.....	177
8-20	Occupancy	178