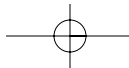
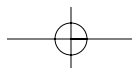


# Contents

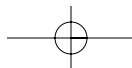
<b>Preface</b>	<b>xi</b>
<b>Acknowledgments</b>	<b>xiii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
Explicitly Parallel Instruction Computing	2
<i>Parallelism</i>	2
<i>Compiler Technology</i>	3
Architecture of the Intel Compiler	4
<i>Profiling a Program</i>	5
<i>Code Generation</i>	5
Overview of Programming Languages	6
<i>Machine Language</i>	6
<i>Assembly Language</i>	7
<i>High-Level Languages</i>	8
Overview of the Compilation Stages of a C Program	9
The Application Program	12
The Software Development Method	12
<i>Specification of the Problem</i>	13
<i>Analysis of the Problem</i>	13
<i>Design of the Algorithm</i>	13
<i>Implementation of the Algorithm</i>	14
<i>Testing of the Algorithm</i>	14
<i>Maintenance of the Program</i>	14
Applying the Software Development Method—A Case Study	15
	 <b>vii</b>

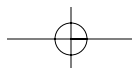




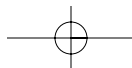
## viii Contents

Modular Programming—C versus Fortran	18
<i>General Structure of a Modular ANSI C/C++ Program</i>	19
<i>General Structure of a C Function</i>	20
<i>Example of a Modular C Program</i>	23
The Structure of a Fortran Program	25
MS Platform SDK	28
<i>Building Applications with the Platform SDK</i>	28
Itanium Processor Software Development Tools	29
<b>Chapter 2 Application Programming Architecture Resources for the Itanium Processor</b>	<b>31</b>
Application Register Set	31
<i>Instruction Pointer</i>	33
<i>General-Purpose Integer Registers</i>	34
<i>Floating-Point Registers</i>	35
<i>Predicate Registers</i>	35
<i>Branch Registers</i>	36
<i>Application Registers</i>	37
Compiler Use of the Architecture	37
<i>Compiler Usage Conventions for Registers</i>	39
Integer Data Types at the Machine Level	42
Data Alignment	45
Storage of C Data Types: Big-Endian vs. Little-Endian	47
<i>Determining the Size of C Data Types</i>	49
Multimedia Data	50
Floating-Point Data	52
<i>IEEE Real Numbers</i>	52
<i>IEEE Real-Number Memory Formats</i>	53
<i>Floating-Point Register Format</i>	54
<i>Storage of Single-, Double-, and Extended-Double Precision         Floating-Point Numbers in Memory</i>	57
<b>Chapter 3 Implementation of High-Level Language Program Structures</b>	<b>63</b>
Itanium Processor Instruction Set	64
Assembly Language Instruction Syntax	64
Instruction Formats	66
<i>Arithmetic Instructions</i>	67
<i>Logic Instructions</i>	68
<i>Shift Instructions</i>	69
<i>Compare Instructions</i>	70
<i>Branch Instructions</i>	73





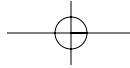
Control-Flow Program Structures	76
<i>IF-THEN-ELSE Constructs</i>	77
<i>Repetition Control Constructs: WHILE/FOR and DO Loops</i>	80
Loads, Stores, and the Locality of Reference	87
<b>Chapter 4 Instruction-Level Parallelism</b>	<b>95</b>
Parallel Processing	95
<i>Types of Dependencies</i>	96
<i>Identifying Instruction Groups</i>	98
Instruction Bundle Format and Templates	99
<i>Automatic and Explicit Bundling</i>	105
Matching the Instruction Stream to Execution Resources	106
An Example Program Organized into Bundles with Templates	109
<b>Chapter 5 Roles of Architectural Features in Implementing Application Programs</b>	<b>113</b>
Speculation	114
<i>Control Dependencies and Control Speculation</i>	116
<i>Implementing Control Speculation</i>	119
<i>Data Dependencies and Data Speculation</i>	120
<i>Implementing Data Speculation</i>	122
Predication	124
<i>Eliminating Conditional Branches with Predication</i>	125
<i>Collapsing Control-Flow Paths</i>	127
<i>Predicated IF-THEN-ELSE Program Structure</i>	127
<i>Implementing SWITCH with Predication</i>	128
Parallel Compares	130
<i>Parallel Branch Implementations</i>	130
<i>Parallel Compare Instructions</i>	131
<i>Optimizing Code with the Parallel Compare Operation</i>	131
Software Pipelining	133
<i>Nonpipelined and Pipelined Loops</i>	133
<i>Resources for Implementing Loop Pipelining</i>	135
<i>Implementing a Loop Program Structure with Pipelining</i>	136
Fused Multiply-Add and Multiply-Subtract Operations	142
<i>Optimizing a Floating-Point Computation Using the Fused Multiply-Add Operation</i>	143
<b>Chapter 6 Creating an Itanium Executable File</b>	<b>145</b>
The Itanium Compiler	145
Compiling for the Itanium Processor on 32- and 64-Bit Systems	146
<i>Selecting the Compiler from within the Microsoft IDE</i>	147
<i>Building for Itanium-Based Platforms</i>	147



**x Contents**

<i>Important Files for Compilation of Itanium-based Applications</i>	149
<i>Invoking the Compiler and Executing the Program</i>	150
Compilation Options and the Compiler Switches	155
<i>Creating Different Output Files</i>	158
<i>Suppressing Linking</i>	159
<i>Linking Object Files</i>	159
<i>Obtaining an Assembly Language Listing</i>	160
<i>Preparing for Debugging</i>	161
<b>Chapter 7 Testing and Debugging Applications</b>	<b>175</b>
Overview of the Intel Itanium Enhanced Debugger	175
Establishing a Remote Debugging Session Using Named Pipes	176
Initiating a Debug Session for edb	178
<i>Invoking edb</i>	179
<i>Remote Upload of a Program</i>	180
<i>Customizing the Debugger Desktop</i>	182
Debugging a C Program	184
<i>Selective Program Execution</i>	185
<i>Breakpoint and Execution Icons</i>	186
<i>Executing the Program</i>	187
<i>Source Level Debugging Example</i>	192
<b>Chapter 8 Applying Libraries in an Application</b>	<b>199</b>
Standard and User Libraries	199
The Standard C Library	200
<i>Contents of the Standard C Library</i>	200
Dynamic-Link Libraries	215
The Intel Integrated Performance Libraries	215
<i>Intel Math Kernel Library 5.0 (MKL v. 5.0)</i>	216
<b>Chapter 9 The Register Stack and Main Memory</b>	<b>221</b>
The Itanium Processor Register Stack	221
Organization of the Register Stack	222
<i>Operation of the Function Call</i>	223
<i>The Current Frame Marker and Previous Frame Marker</i>	224
The Register Stack Engine and Backing Store	225
Configuring the Register Stack Engine	228
Register Stack Instructions	229
Organization of the Application Memory Address Space	230
<i>Static Application Memory Map</i>	230
<i>Structure of a Main Memory Stack Frame</i>	233
<i>Stack Alignment</i>	237
<i>Text and Data Section Alignment</i>	238
Example Program—Creation and Use of the Heap	239

<b>Chapter 10</b>	<b>Porting of Existing Applications to the Itanium Processor</b>	<b>243</b>
Porting		243
<i>I/O Considerations</i>	244	
<i>Driver/Hardware Considerations</i>	245	
Application Porting Issues		245
<i>Memory Model and Data-Type Changes</i>	246	
<i>Size-Variant and Fixed-Size Data Types</i>	248	
<i>Constants</i>	249	
<i>Memory Allocation and sizeof</i>	250	
<i>Offsets into Arrays and Structures</i>	252	
<i>Unions—Implicit and Explicit</i>	252	
<i>printf Format Specifications</i>	253	
Operating System API Issues		253
<i>Changed API Impacts</i>	255	
<i>Compiler and Linker Issues</i>	256	
Obstacles to an Easy Port		258
<i>Warning Level and Lint Copious Results</i>	258	
<i>Limitations of Compiler Issue Detection</i>	259	
<i>Application Interdependence and Change Impact Scope</i>	260	
The Porting Process		260
<i>Inventory—Source Code, Libraries, Drivers, Tools, Hardware, and Skills</i>	260	
<i>Assessment—Quantifying the Issues and Measuring their Scope and Magnitude</i>	260	
<i>Planning—The Blueprint</i>	263	
<i>Migration—Executing the Plan</i>	265	
<i>Formal Test and Debug—Verifying the Results</i>	266	
Migration Project Example		266
<i>Inventory</i>	267	
<i>Assessment and Planning</i>	269	
<i>Migration—Porting the Code</i>	270	
<b>Chapter 11</b>	<b>Optimization and Tuning of Itanium Processor Applications</b>	<b>279</b>
Optimization versus Tuning		279
A Few Important Principles		280
General Principles of Optimization		280
<i>Algorithm Choice</i>	280	
<i>Loop Invariants</i>	281	
<i>Unrolling Loops</i>	281	
<i>Branch Avoidance</i>	282	

**xii Contents**

Interpreting Compiled Code	283
<i>Locating Code in an Assembly Language Listing</i>	283
<i>Reading an Inner Loop</i>	287
Instruction Latencies	291
What the Compiler Can Do	294
Where the Compiler Needs Help	297
<i>Avoiding Aliasing</i>	297
<i>Interprocedural Optimization</i>	300
<i>Profile-Guided Optimization</i>	301
Compiler Options and Switches	302
Code Experimentation	304
Memory Matters	307
Using Intrinsic	309
Writing in Assembly Language	312
VTune and EMon	313
Conclusions	313
<b>Chapter 12 An Optimization Case Study</b>	<b>315</b>
Test Function	315
Test Cases	316
Results First	317
Analyzing the Simple Case	319
Analyzing the Reordered Case	323
Analyzing the ASCII Collapsed Case	323
Analyzing the ASCII Signed Case	325
Analyzing the ASCII Gone Case	326
Analyzing the Bit Logic Case	327
Analyzing the Count += Case	328
Analyzing the Count Doubled Case	329
Taking It to Assembly Language	330
Multimedia: Bringing in the Big Guns	333
Shaving Off One Last Instruction	336
A Final Assembly Language Assault	337
Conclusions	341
<b>Appendix Itanium eel Compiler Switches</b>	<b>343</b>
<b>Index</b>	<b>347</b>

