
Contents

Figures xi

Tables xiii

About This Guide xv

What This Guide Contains xv

What You Should Know Before Reading This Guide xvi

Suggestions for Further Reading xvi

Conventions Used in This Guide xviii

1. **About the MIPSpro Compiler System** 1
2. **Using the MIPSpro Compiler System** 7
 - Selecting a Compiler 8
 - Using a Defaults Specification File 8
 - Using Command-Line Options 10
 - Setting an Environment Variable 10
 - Object File Format and Dynamic Linking 10
 - Executable and Linking Format 11
 - Dynamic Shared Objects 11
 - Position-Independent Code 12

- Source File Considerations 12
 - Source File Naming Conventions 13
 - Header Files 13
 - Specifying a Header File 14
 - Creating a Header File for Multiple Languages 14
 - Using Precompiled Headers in C and C++ 15
 - About Precompiled Headers 16
 - Automatic Precompiled Header Processing 16
 - Other Ways to Control Precompiled Headers 20
 - PCH Performance Issues 21
- Compiler Drivers 22
 - Default Behavior for Compiler Drivers 22
 - General Options for Compiler Drivers 23
- Linking 27
 - Invoking the Linker Manually 28
 - Linker Syntax 28
 - Linker Example 30
 - Linking Assembly Language Programs 30
 - Linking Libraries 30
 - Specifying Libraries and DSOs 31
 - Examples of Linking DSOs 32
 - Linking to Previously Built Dynamic Shared Objects 32
 - Linking Multilanguage Programs 33
 - Finding an Unresolved Symbol With *ld* 35
- Debugging 35

Getting Information About Object Files	36
Disassembling Object Files with <i>dis</i>	37
<i>dis</i> Syntax	37
<i>dis</i> Options	37
Listing Parts of DWARF Object Files With <i>dwarfdump</i>	38
<i>dwarfdump</i> Syntax	38
<i>dwarfdump</i> Options	39
Listing Parts of ELF Object Files and Libraries with <i>elfdump</i>	40
<i>elfdump</i> Syntax	40
<i>elfump</i> Options	40
Determining File Type with <i>file</i>	42
<i>file</i> Syntax	43
<i>file</i> Example	43
Listing Symbol Table Information: <i>nm</i>	43
<i>nm</i> Syntax	43
<i>nm</i> Symbol Table Options	44
<i>nm</i> Example of Obtaining a Symbol Table Listing	46
Determining Section Sizes with <i>size</i>	47
<i>size</i> Syntax	47
<i>size</i> Options	47
<i>size</i> Example	48
Removing Symbol Table and Relocation Bits with <i>strip</i>	48
<i>strip</i> Syntax	49
Using the Archiver to Create Libraries	49
<i>ar</i> Syntax	50
<i>ar</i> Options	51
<i>ar</i> Examples	52
3. Using Dynamic Shared Objects	55
Benefits of Using DSOs	55

- Using DSOs 57
 - DSOs vs. Archive Libraries 57
 - Using QuickStart 58
 - Guidelines for Using Shared Libraries 58
 - Choosing Library Members 59
 - Tuning Shared Library Code 60
- Taking Advantage of QuickStart 62
- Building DSOs 65
 - Creating DSOs 65
 - Making DSOs Self-Contained 65
 - Controlling Symbols to Be Exported or Loaded 66
 - Using DSOs With C++ 67
 - Using Registry Files 68
 - Registry File Format 69
 - Directive Lines 69
 - Shared Object Specification Lines 70
- Run-Time Linking 71
 - Searching for DSOs at Run Time 71
 - Run-Time Symbol Resolution 72
 - Compiling with `-Bsymbolic` 72
 - Converting Libraries to DSOs 73
- Dynamic Loading Under Program Control 75
- Versioning of DSOs 77
 - The Versioning Mechanism 77
 - What Is a Version? 78
 - Building a Shared Library Using Versioning 78
 - Example of Versioning 79
- 4. Optimizing Program Performance 83**
 - Optimization Overview 84
 - Benefits of Optimization 84
 - Optimization and Debugging 84
 - Using the Optimization Options 84

Performance Tuning with Interprocedural Analysis	85
Inlining	88
Benefits of Inlining	88
Inlining Options for Routines	89
Options To Control Inlining Heuristics	91
Common Block Padding	92
Alias and Address Taken Analysis	93
The -IPA:alias=ON Option	93
The -IPA:addressing=ON Option	94
The -IPA:opt_alias=ON Option	94
Controlling Loop Nest Optimizations	94
Running LNO	94
LNO Optimizations	97
Loop Interchange	97
Blocking and Outer Loop Unrolling	98
Loop Fusion	99
Loop Fission/Distribution	100
Prefetching	102
Gather-Scatter Optimization	102
Compiler Options for LNO	103
Controlling LNO Optimization Levels	104
Controlling Fission and Fusion	104
Controlling Gather-Scatter	105
Controlling Cache Parameters	105
Controlling Blocking and Permutation Transformations	107
Controlling Prefetch	108
Dependence Analysis	109
Pragmas and Directives for LNO	109
Fission/Fusion	110
Blocking and Permutation Transformations	111
Prefetch	114
Dependence Analysis	116

- Controlling Floating Point Optimization 117
 - OPT:roundoff=n 118
 - OPT:IEEE_arithmetic=n 119
 - Other Options to Control Floating Point Behavior 121
 - Debugging Floating-Point Problems 122
- Controlling Miscellaneous Optimizations With the -OPT Option 123
 - Using the -OPT:space Option 123
 - Using the -OPT:Olimit=n Option 123
 - Using the -OPT:alias Option 124
 - Simplifying Code With the -OPT Option 125
- The Code Generator 126
 - Overview of the Code Generator 126
 - Code Generator and Optimization Levels 127
 - An Example of Local Optimization for Fortran 127
 - Code Generator and Optimization Levels -O2 and -O3 128
 - If Conversion 128
 - Cross-Iteration Optimizations 130
 - Read-Read Elimination 130
 - Read-Write Elimination 130
 - Write-Write Elimination 131
 - Common Sub-expression Elimination 131
 - Loop Unrolling 131
 - Recurrence Breaking 132
 - Software Pipelining 133
 - Steps Performed By the Code Generator at Levels -O2 and -O3 133
 - Modifying Code Generator Defaults 134
 - Miscellaneous Code Generator Performance Topics 135
 - Prefetch and Load Latency 135
 - Frequency and Feedback 136
- Controlling the Target Architecture 136
- Controlling the Target Environment 137

- Improving Global Optimization 138
 - Overview of the Global Optimizer 138
 - Optimizing C, C++, and Fortran Programs 139
 - Optimizing C and C++ Programs 139
 - Example of Pointer Placement and Aliasing 140
 - Improving Other Optimization 141
 - C, C++, and Fortran Programs 142
 - C and C++ Programs 142
 - C++ Programs Only 143
 - Register Allocation 143
 - Using SpeedShop 144
- 5. **Coding for 64-Bit Programs** 147
 - Coding Assumptions to Avoid 147
 - sizeof(int) == sizeof(void *) 148
 - sizeof(int) == sizeof(long) 148
 - sizeof(long) == 4 148
 - sizeof(void *) == 4 149
 - Implicitly Declared Functions 149
 - Constants With the High-Order Bit Set 149
 - Arithmetic with **long** Types 149
 - Solving Porting Problems 150
- Guidelines for Writing Code for 64-Bit Silicon Graphics Platforms 150
- 6. **Porting Code to N32 and 64-Bit Silicon Graphics Systems** 155
 - Compatibility 155
 - N32 Porting Guidelines 157
 - Porting Environment 158
 - Source Code Changes 158
 - Build Procedure 158
 - Runtime Issues 159

- Porting Code to 64-Bit Silicon Graphics Systems 159
 - Using Data Types 160
 - Using Predefined Types 161
 - Using Typedefs 162
 - Maximum Memory Allocation 163
 - Arrays Larger Than 2 Gigabytes 163
 - Example of Arrays Larger Than 2 Gigabytes 163
 - Using Large Files With XFS 165
- Index** 167
- Important Note 181