Chapter 1
Automatic Parallelization for MIPSpro Compilers

This chapter discusses automatic parallelization for the 7.2 and later releases of the Silicon Graphics MIPSpro compilers. You can achieve automatic parallelization with these compilers using the MIPSpro Auto−Parallelizing Option (APO), an optional software product for programs written for the N32 and N64 application binary interfaces. See the ABI(5) reference page for information on the N32 and N64 ABIs.

The MIPSpro APO is an extension integrated into the four compilers listed in the left column of Table 1−1. It is not a source−to−source preprocessor as was used prior to the 7.2 release. If the Auto−Parallelizing Option is installed, the compilers are known as *auto−parallelizing compilers* and are referred to by the names in the right column.

Table 1−1 MIPSpro 7.2 (and Later) Compilers and the Auto−Parallelizing Option

Standard Compilers	Compilers With the Auto−Parallelizing Option
MIPSpro Fortran 77	MIPSpro Auto−Parallelizing Fortran 77
MIPSpro Fortran 90	MIPSpro Auto−Parallelizing Fortran 90
MIPSpro C	MIPSpro Auto−Parallelizing C
MIPSpro C++	MIPSpro Auto−Parallelizing C++

This chapter contains these sections:

- "Understanding Automatic Parallelization" introduces parallelization in the MIPSpro compilers.
- "About the MIPSpro Auto−Parallelizing Option" describes the MIPSpro APO's features.
- "Using the MIPSpro Auto−Parallelizing Option" tells how to use the MIPSpro APO to achieve parallelization.

Understanding Automatic Parallelization

Parallelization is the process of analyzing sequential programs for parallelism and restructuring them to run efficiently on multiprocessor systems. The goal is to minimize the overall computation time by distributing the computational workload among the available processors. Parallelization can be automatic or manual.

During *automatic parallelization*, the Auto−Parallelizing Option extension of the compiler analyzes and restructures the program with little or no intervention by you. The MIPSpro APO automatically generates code that splits the processing of loops among multiple processors. An alternative is *manual parallelization*, in which you perform the parallelization using compiler directives and other programming techniques. Manual parallelization is discussed in the documents listed under "Manual Parallelization References". The introduction also contains useful optimization references under "References on Optimization Techniques".

About the MIPSpro Auto−Parallelizing Option

The Auto−Parallelizing Option helps you exploit parallelism in programs to enhance their performance on multiprocessor systems. It is a compiler extension controlled with flags in the command lines that invoke the MIPSpro auto−parallelizing compilers. Although their runtime performance suffers slightly on single−processor systems, parallelized programs can be created and debugged with the MIPSpro auto−parallelizing compilers on any Silicon Graphics system that uses a MIPS processor.

Starting with the 7.2 release, the auto−parallelizing compilers integrate automatic parallelization, provided by the MIPSpro APO, with other compiler optimizations, such as interprocedural analysis (IPA) and loop nest optimization (LNO). Whereas releases prior to 7.2 relied on source−to−source preprocessors, the 7.2 and later versions internalize automatic parallelization into the optimizer of the MIPSpro compilers. As seen in Figure 1−1, the MIPSpro APO works on an intermediate representation generated during the compiling process. This provides several benefits:

- Automatic parallelization is integrated with the optimizations for single processors.
- The options and compiler directives of the MIPSpro APO and the MIPSpro compilers are consistent.
- Support for C++ is now possible.
- Runtime and compile−time performance is improved.

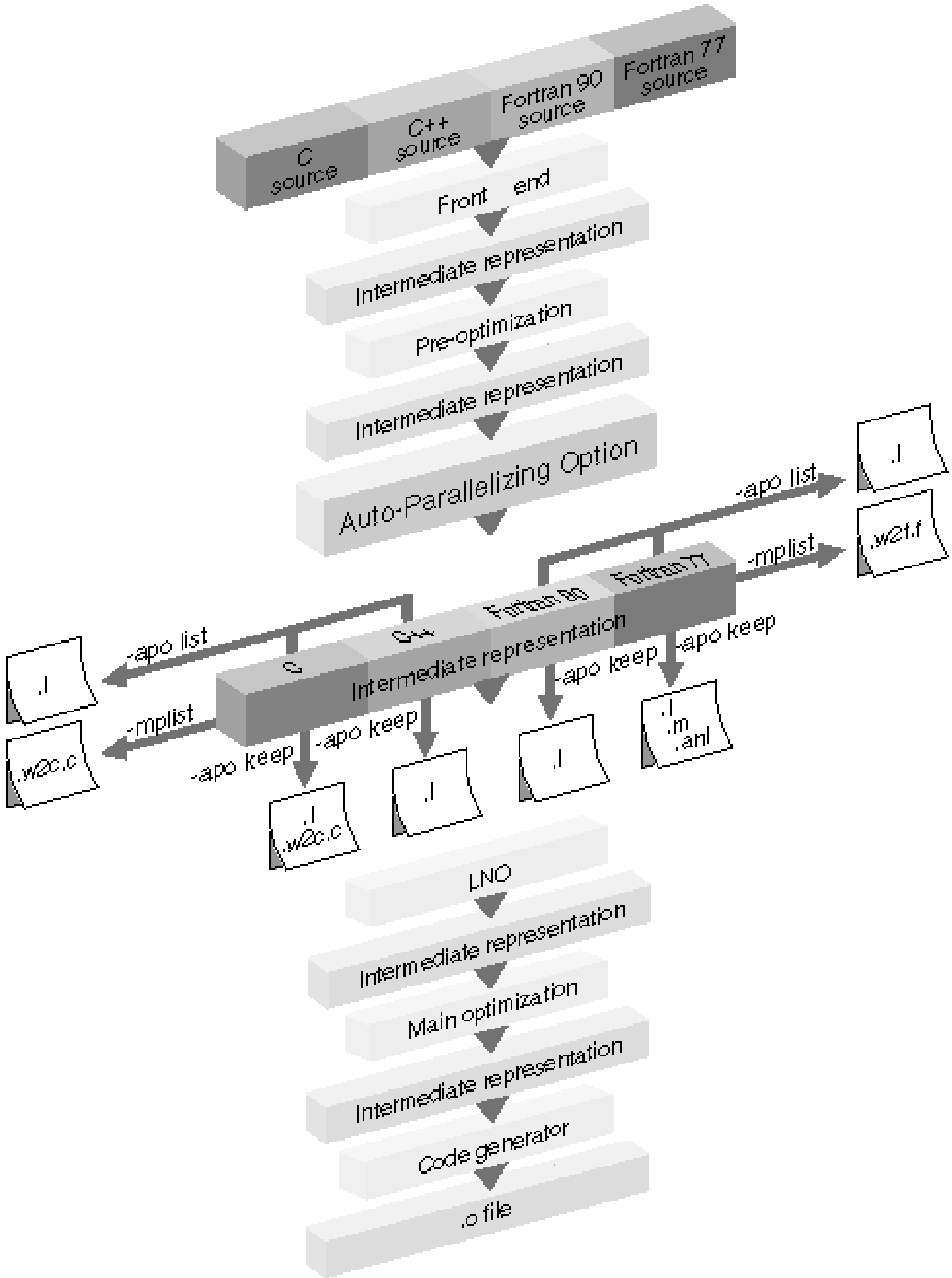

Figure 1−1 Files Generated by the MIPSpro Auto−Parallelizing Option

These benefits were not possible with the earlier MIPSpro compilers, which achieved parallelization by relying on the Power Fortran and Power C preprocessors to provide source−to−source conversions before compilation.

Using the MIPSpro Auto−Parallelizing Option

This section describes how to use the MIPSpro Auto−Parallelizing Option to compile and run parallelized programs.

- "Invoking the Auto−Parallelizing Option" demonstrates how to compile programs with the auto−parallelizing versions of Fortran 77, Fortran 90, C, and C++.

- "MIPSpro Compiler Command−Line Options"describes options relevant to the MIPSpro APO.
- "Understanding the Auto−Parallelizing Option Output Files"explains how to use the listing mechanisms of the MIPSpro APO to see what transformations were performed and what regions of codes were parallelized.
- "Running Your Program" shows how to run your parallelized program.

Invoking the Auto−Parallelizing Option

You invoke the Auto−Parallelizing Option by including the **−apo** flag on the command line that starts a MIPSpro auto−parallelizing compiler. Additional flags allow you to generate reports to aid in debugging. The syntax for compiling programs with the MIPSpro APO is as follows:

- Auto−Parallelizing Fortran 77, Auto−Parallelizing Fortran 90, Auto−Parallelizing C, and Auto−Parallelizing C++ are invoked using **−apo**:

```
f77 options −apo[{list|keep}] [−mplist] filename
f90 options −apo[{list|keep}]           filename
cc  options −apo[{list|keep}] [−mplist] filename
CC  options −apo[{list|keep}]           filename
```

- Alternatively, the auto−parallelizing compilers may be invoked using the **−pfa** or **−pca** flags instead of **−apo**. These options are provided for backward compatibility and their use is deprecated.

The command−line entries are defined as follows:

*option*s	The MIPSpro compiler command−line options. **−O3** is recommended for using the MIPSpro APO. For details, see "MIPSpro Compiler Command−Line Options", and the documentation for your MIPSpro compiler.
−apo	Invoke the Auto−Parallelizing Option.
−apo list	Invoke the MIPSpro APO and produce a *.l* file, a listing of those parts of the program that can run in parallel and those that cannot. This file is discussed in "About the .l File".
−apo keep	Invoke the MIPSpro APO and generate *.l*, *.w2c.c*, *.m*, and *.anl* files as shown in Table 1−2. Because of data conflicts, do not use with **−mplist** or the LNO options **−FLIST** and **−CLIST**. See "Loop Nest Optimizer Options".
−mplist	Generate the equivalent parallelized program for Fortran 77, in a *.w2f.f* file, or for C, in a *.w2c.c* file. These files are discussed in the section "About the .w2f.f and .w2c.c Files". Do not use with **−apo keep**, **−FLIST**, or **−CLIST**.
filename	The name of the file containing the source code.

Note: Starting with the 7.2.1 release of the MIPSpro compilers, the **−apo keep** and **−mplist** flags cause Auto−Parallelizing Fortran 77 to generate *.m* and *w2f.f* files based on OpenMP directives (see "OpenMP and the Auto−Parallelizing Fortran 77 Output Files"). To have the compiler emit the pre−OpenMP directives, use the Fortran 77 option **−FLIST:emit_pcf** instead of these flags.

The files generated by **−apo keep** with the various compilers are shown in Table 1−2 The *.l* file is the same as that generated using **−apo list**, and the *.w2c.c* file is the same as that generated using **−mplist**. The other two files are for Fortran 77 only. They are the *.m* file, a parallelized equivalent program based on OpenMP, and the *.anl* file, a file for use with WorkShop Pro MPF. These files are explained in the section "About the .m and .anl Files".

Table 1−2 Files Generated by −apo keep

File Suffix	f77	f90	cc	CC
.l	Yes	Yes	Yes	Yes
.w2c.c	No	No	Yes	No
.m	Yes	No	No	No
.anl	Yes	No	No	No

Consider a typical command line:

```
f77 −O3 −n32 −mips4 −c −apo −mplist myProg.f
```

This command uses Auto−Parallelizing Fortran 77 (**f77**...**−apo**) to compile (**−c**) the file *myProg.f* with the MIPSpro compiler options **−O3**, **−n32**, and **−mips4**. Using **−O3**, which requests aggressive optimization, is recommended for using the MIPSpro APO. It is covered in "Optimization Options". The option **−n32** requests an object with an N32 ABI; **−mips4** requests that the code be generated with the MIPS IV instruction set. You can find out more about these options in the *MIPSpro Compiling and Performance Tuning Guide.* Using **−mplist** requests that a parallelized Fortran 77 program be created in the file *myProg.w2f.f*. If you are using WorkShop Pro MPF, you may want to use **−apo keep** instead of **−mplist** to get a *.anl* file.

To use the Auto−Parallelizing Option correctly, remember these points:

- The MIPSpro APO can be used only with **−n32** or **−64** compiles. With **−o32** compiles, the **−pfa** and the **−pca** flags invoke the older, Power parallelizers, and the **−apo** flag is not supported.
- If you link separately, you must have one of the following in the link line:
  - the **−apo** flag
  - the **−mp** option (See the *MIPSpro Fortran 77 Programmer's Guide.*)
- Because of data set conflicts, you can use only one of the following in a compilation:
  - **−apo keep**
  - **−mplist**
  - **−FLIST** or **−CLIST** (See "Loop Nest Optimizer Options".)

MIPSpro Compiler Command−Line Options

Prior to MIPSpro 7.2, parallelization was done by the Power Fortran and Power C preprocessors, which had their own set of flags. Starting with MIPSpro 7.2, the Auto−Parallelizing Option does the parallelization and recognizes the same options as the compilers. This has reduced the number of options you need to know and has simplified their use. For example, suppose you are using Auto−Parallelizing Fortran 77 and want to turn off round−off changing transformations in all phases of compiling. In MIPSpro 7.2, specifying **−OPT:roundoff=0** does this. Previously, you also needed to add the option **−pfa,−r=0** to turn off round−off changing transformations in the Power Fortran preprocessor.

The next sections cover the compiler command−line options most commonly needed with the Auto−Parallelizing Option:

- "Optimization Options"
- "Interprocedural Analysis"
- "Loop Nest Optimizer Options"
- "Miscellaneous Optimization Options"

For more extensive information about compiler command−line options, see the *MIPSpro Compiling and Performance Tuning Guide* and the guide for your compiler.

Optimization Options

Optimization option **−O3** performs aggressive optimization and its use is recommended to run the MIPSpro APO. The optimization at this level maximizes code quality even if it requires extensive compile time or relaxing language rules. The **−O3** option uses transformations that are usually beneficial but can hurt performance in pathological cases. This level may cause noticeable changes in floating−point results due to the relaxation of operation−ordering rules. Floating−point optimization is discussed further in "Miscellaneous Optimization Options".

Interprocedural Analysis

Interprocedural analysis (IPA), invoked by the **−IPA** command−line option, performs program optimizations that can be done only with knowledge of the whole program. Typical IPA optimizations are

- procedure inlining
- identification of global constants
- dead function elimination
- dead variable elimination
- dead call elimination
- interprocedural alias analysis
- interprocedural constant propagation

More information about these optimizations can be found in the books listed under "References on Optimization Techniques".

As of the MIPSpro 7.2.1 release, the Auto−Parallelizing Option with IPA is able to optimize only those loops whose function calls were inlined by the MIPSpro APO. This is further described in "Function Calls in Loops".

Note: If IPA expands subroutines inline in a calling routine, the subroutines are compiled with the options of the calling routine. If the calling routine is not compiled with **−apo**, none of its inlined subroutines are parallelized. This is true even if the subroutines are compiled separately with **−apo**, because with IPA automatic parallelization is deferred until link time.

Loop Nest Optimizer Options

The loop nest optimizer (LNO) performs loop optimizations that better exploit caches and instruction−level parallelism. Some of the optimizations of the LNO are

- loop interchange
- loop fusion
- loop fission
- cache blocking and outer loop unrolling

The LNO runs when you use the **−O3** option. It is an integrated part of the compiler back end, not a preprocessor. As is true with the Auto−Parallelizing Option, the same optimizations and control options can be used with Fortran, C, or C++ programs. The *MIPSpro Compiling and Performance Tuning Guide* describes three LNO options of particular interest to users of the MIPSpro APO:

- **−LNO:parallel_overhead=***n*. This option controls the auto−parallelizing compiler's estimate of the overhead incurred by invoking parallel loops. The default value for *n* varies on different systems, but is typically in the low thousands of processor cycles.
- **−LNO:auto_dist=on** This option requests that the MIPSpro APO insert data distribution directives to provide the best memory utilization on the S2MP (Scalable Shared−Memory Parallel) architecture of the Origin2000 platform. There is more information about this option in your compiler's reference pages.
- **−LNO:ignore_pragmas** This option causes the MIPSpro APO to ignore all of the directives, assertions, and pragmas covered in "Compiler Directives for Automatic Parallelization". This includes the directive **C*$* NO CONCURRENTIZE**.

If you are using Fortran 77 or C, you can view the transformed code in the original source language after the LNO performs its transformations. Two translators, integrated into the back end, convert the compiler's internal representation into the original source language. You can invoke the desired translator by using the *f77* option **−FLIST:=on** or the *cc* option **−CLIST:=on** For example,

```
f77 −O3 −FLIST:=on test.f
```

creates an *a.out* object file and the Fortran file *test.w2f.f*. Because it is generated at a later stage of the compilation, this *.w2f.f* file differs somewhat from the *.w2f.f* file generated by the **−mplist** option (see "About the .w2f.f and .w2c.c Files"). You can read the *.w2f.f* file, which is a compilable Fortran representation of the original program after the LNO phase. Because the LNO is not a preprocessor, recompiling the *.w2f.f* file may result in an executable that differs from the original compilation of the *.f* file.

Miscellaneous Optimization Options

Miscellaneous optimizations, controlled by the **−OPT** command−line option, are those not associated with a distinct compiler phase. Two of these optimizations are particularly relevant to the MIPSpro APO:

- **−OPT:alias=***name*
- **−OPT:roundoff=***n*

The **−OPT:alias=***name* option has several variations. One of interest to users of the MIPSpro APO is **−OPT:alias=restrict** Under this option, the compiler assumes a very restrictive model of aliasing: Memory operations dereferencing different named pointers are assumed neither to alias with each other, nor to alias with any named scalar variable. Even explicit assignments, such as the one below, are forbidden:

```
int *i, *j;
i = j;
```

Consider this code:

```
void dbl (int *i, int *j){
   *i = *i + *i;
   *j = *j + *j;
}
```

The compiler assumes that *i* and *j* point to different memory locations, and produces an overlapped schedule for the two calculations. See also "Aliased Parameter Information" in Chapter 2 about using **__restrict** as an alternative approach to aliasing.

A related option is **−OPT:alias=disjoint** Under this option, the compiler assumes memory operations dereferencing different named pointers do not alias with each other, and different dereferencing depths of the same pointer do not alias with each other. For example, if *p* and *q* are pointers, **p* does not alias with **q*, ***p*, or ***q*.

The **−OPT:roundoff=***n* option controls floating−point accuracy and the behavior of overflow and underflow exceptions relative to the source language rules. The default for **−O3** optimization is **−OPT:roundoff=2** This setting allows transformations with extensive effects on floating−point results. It allows associative rearrangement across loop iterations, and the distribution of multiplication over addition and subtraction. It disallows only transformations known to cause overflow, underflow, or cumulative round−off errors for a wide range of floating−point operands.

At the **−OPT:roundoff=2** or **3** level of optimization, the MIPSpro APO may change the sequence of a loop's floating−point operations in order to parallelize it. Because floating−point operations have finite precision, this change may cause slightly different results. If you want to avoid these differences by not having such loops parallelized, you must compile with the **−OPT:roundoff=0** or **1** command−line option. Consider this example:

```
REAL A, B(100)
DO I = 1, 100
  A = A + B(I)
END DO
```

At the default setting of **−OPT:roundoff=2** for the **−O3** level of optimization, the MIPSpro APO parallelizes this loop. At the start of the loop, each processor gets a private copy of *A* in which to hold a partial sum. At the end of the loop, the partial sum in each processor's copy is added to the total in the original, global copy. This value of *A* may be different from the value generated by a version of the loop that is not parallelized.

Understanding the Auto−Parallelizing Option Output Files

Processing a program with the Auto−Parallelizing Option often results in excellent parallelization with no further effort. But, as described in Chapter 2, "Understanding Incomplete Optimization," there are cases that cannot be effectively parallelized automatically. To help analyze these cases, the MIPSpro APO provides a number of options to generate listings that describe where parallelization failed and where it succeeded. By understanding these listings, you may be able to identify small problems that prevent a loop from being made parallel. With a little work, you can often remove these data dependences, dramatically improving the program's performance.

Tip: When looking for loops to run in parallel, focus on the areas of the code that use most of the execution time. Optimizing a routine that uses only one percent of the execution time cannot significantly improve the overall performance of your program. To determine where the program spends its execution time, you can use tools such as SpeedShop and the WorkShop Pro MPF Parallel Analyzer View. More information about these tools can be found in "About the .m and .anl Files".

About the .l File

The **−apo list**and **−apo keep**options generate files, whose names end with *.l*, that list the original loops in the program along with messages telling whether or not the loops were parallelized. For loops that were not parallelized, an explanation is given.

Example 1−1shows a simple Fortran 77 program. The subroutine is contained in a file named *testl.f.*

Example 1−1 Subroutine in File testl.f

```
SUBROUTINE sub(arr, n)
  REAL*8 arr(n)
  DO i = 1, n
    arr(i) = arr(i) + arr(i−1)
  END DO
  DO i = 1, n
    arr(i) = arr(i) + 7.0
    CALL foo(a)
  END DO
  DO i = 1, n
    arr(i) = arr(i) + 7.0
  END DO
END
```

When *testl.f* is compiled with

```
f77 −O3 −n32 −mips4 −apo list testl.f −c.
```

the Auto−Parallelizing Option produces the file*testl.l*, shown in Example 1−2

Example 1−2 Listing in File testl.l

```
Parallelization Log for Subprogram sub_
3: Not Parallel
         Array dependence from arr on line 4 to arr on line 4.
6: Not Parallel
         Call foo on line 8.
```

```
10: PARALLEL (Auto) __mpdo_sub_1
```

The last line (10) is important to understand. Whenever a loop is run in parallel, the parallel version of the loop is put in its own subroutine. The MIPSpro profiling tools attribute all the time spent in the loop to this subroutine. The last line indicates that the name of the subroutine is *__mpdo_sub_1*. For more information about interpreting this file, you can refer to Chapter 2, "Understanding Incomplete Optimization," or "Additional Reading".

OpenMP and the Auto−Parallelizing Fortran 77 Output Files

The 7.2.1 release of the MIPSpro compilers is the first to incorporate OpenMP, a cross−vendor API for shared−memory parallel programming in Fortran and, eventually, C and C++. OpenMP— a collection of directives, library routines, and environment variables—is used to specify shared−memory parallelism in source code. Additionally, OpenMP is intended to enhance your ability to implement the coarse−grained parallelism of large code sections. On Silicon Graphics platforms, OpenMP replaces the older Parallel Computing Forum (PCF) and SGI DOACROSS directives for Fortran. More information about the specification can be found at the OpenMP Web site: http://www.openmp.org/.

The MIPSpro APO interoperates with OpenMP as well as with the older directives. This means that an Auto−Parallelizing Fortran 77 or Auto−Parallelizing Fortran 90 file may use a mixture of directives from each source. As of the 7.2.1 release, the only OpenMP−related changes that most MIPSpro APO users see are in the Auto−Parallelizing Fortran 77 *w2f.f* and *.m* files, generated using the **−mplist** and **−apo keep** flags, respectively. The parallelized source programs contained in these files now contain OpenMP directives. None of the other MIPSpro auto−parallelizing compilers generate source programs based on OpenMP.

About the .w2f.f and .w2c.c Files

The *.w2f.f* and *.w2c.c* files contain Fortran 77 and C code, respectively, that mimics the behavior of programs after they undergo automatic parallelization. The representations in these files are designed to be readable so that you can see what portions of the original code were not parallelized. You can use the information in these files to change the original programs to aid their parallelization.

The MIPSpro auto−parallelizing compilers create the *.w2f.f* and *.w2c.c* files by invoking the appropriate translator to turn the compilers' internal representations into Fortran 77 or C. In most cases, the files contain valid code that can be recompiled, although compiling a *.w2f.f* or *.w2c.c* file with a standard MIPSpro compiler does not produce object code that is exactly the same as that generated by an auto−parallelizing compiler processing the original source. This is because the MIPSpro APO is an internal phase of the MIPSpro auto−parallelizing compilers, not a source−to−source preprocessor, and does not use a *w2f.f* or *.w2c.c* source file to generate the object file.

The **−mplist** option tells Auto−Parallelizing Fortran 77 to compile a program and generate a *.w2f.f* file. Because it is generated at an earlier stage of the compilation, this *.w2f.f* file is much more easily understood than the *.w2f.f* file generated using the **−FLIST:=on** option (see "Loop Nest Optimizer Options"). The parallelized program in the *.w2f.f* file uses OpenMP directives (see "OpenMP and the Auto−Parallelizing Fortran 77 Output Files"). A *.w2f.f* program that uses PCF instead of OpenMP can be generated by adding the option **−FLIST:emit_pcf** to the *f77* command line. There is no *.w2f.f* file for Auto−Parallelizing Fortran 90.

The **−mplist**and **−apo keep**options instruct Auto−Parallelizing C to generate a*.w2c.c* file. The *.w2c.c* file contains a parallelized program based on directives similar to the Fortran ones of PCF. There is no *.w2c.c* file for the auto−parallelizing version of C++.

Consider the subroutine in Example 1−3, contained in a file named *testw2.f*.

Example 1−3 Subroutine in File testw2.f

```
SUBROUTINE trivial(a)
  REAL a(10000)
  DO i = 1,10000
    a(i) = 0.0
  END DO
END
```

After compiling *testw2.f* using

```
f77 −O3 −n32 −mips4 −c −apo −mplist testw2.f
```

you get an object file, *testw2.o*, and a file, *testw2.w2f.f*, that contains the code shown in Example 1−4

Example 1−4 Listing in File testw2.w2f.f

```
C ***********************************************************
C Fortran file translated from WHIRL Sun Dec  7 16:53:44 1997
C ***********************************************************


        SUBROUTINE trivial(a)
        IMPLICIT NONE
        REAL*4 a(10000_8)
C
C       **** Variables and functions ****
C
        INTEGER*4 i
C
C       **** statements ****
C
C       PARALLEL DO will be converted to SUBROUTINE __mpdo_trivial_1
C$OMP PARALLEL DO private(i), shared(a)
        DO i = 1, 10000, 1
          a(i) = 0.0
        END DO
        RETURN
        END ! trivial
```

Note: WHIRL is the name for the compiler’s intermediate representation.

As explained in "About the .l File", parallel versions of loops are put in their own subroutines. In this example, that subroutine is *__mpdo_trivial_1*. **C$OMP PARALLEL DO** is an OpenMP directive that specifies a parallel region containing a single DO directive. See "OpenMP and the

Auto−Parallelizing Fortran 77 Output Files"for more information on OpenMP.

About the .m and .anl Files

For Auto−Parallelizing Fortran 77, the**−apo keep**option generates two files in addition to a *.l* file:

- A *.m* file, which is similar to the *.w2f.f* file. It is based on OpenMP (see "OpenMP and the Auto−Parallelizing Fortran 77 Output Files") and mimics the behavior of the program after automatic parallelization. It is also annotated with information that is used by Workshop ProMPF. A parallelized Fortran 77 program based on the PCF directives instead of OpenMP can be created by using the option **−FLIST:emit_pcf**in addition to the **−apo keep**flag.
- A *.anl* file, which is used by Workshop ProMPF.

Silicon Graphics provides a separate product, WorkShop Pro MPF, that provides a graphical interface to aid in both automatic and manual parallelization for Fortran 77. In particular, the WorkShop Pro MPF Parallel Analyzer View helps you understand the structure and parallelization of multiprocessing applications by providing an interactive, visual comparison of their original source with transformed, parallelized code. Refer to the *Developer Magic: WorkShop Pro MPF User's Guide* and the *Developer Magic: Performance Analyzer User's Guide* for details.

SpeedShop, another Silicon Graphics product, allows you to run experiments and generate reports to track down the sources of performance problems. SpeedShop consists of an API, a set of commands that can be run in a shell, and a number of libraries to support the commands. For more information, see the *SpeedShop User's Guide*.

Running Your Program

Running a parallelized version of your program is no different from running a sequential one. The same binary can be executed on various numbers of processors. The default is to have the run−time environment select the number of processors to use based on how many are available.

You can change the default behavior by setting the environment variable OMP_NUM_THREADS, which tells the system to use an explicit number of processors. The statement

```
setenv OMP_NUM_THREADS 2
```

causes the program to create two threads regardless of the number of processors available. Using OMP_NUM_THREADS is preferable to using MP_SET_NUMTHREADS and its older synonym NUM_THREADS, which preceded the release of the MIPSpro APO with OpenMP.

The environment variable OMP_DYNAMIC allows you to control whether the run−time environment should dynamically adjust the number of threads available for executing parallel regions to optimize system resources. The default value is TRUE. If OMP_DYNAMIC is set to FALSE,

```
setenv OMP_DYNAMIC FALSE
```

dynamic adjustment is disabled.