Chapter 3

Assisting the MIPSpro Auto−Parallelizing Option

This chapter discusses actions you can take to enhance the performance of the MIPSpro Auto−Parallelizing Option.

- "Strategies for Assisting the Auto−Parallelizing Option"gives strategies for optimizing your code for the MIPSpro APO.
- "Compiler Directives for Automatic Parallelization" discusses the compiler directives the MIPSpro APO recognizes.

Strategies for Assisting the Auto−Parallelizing Option

There are circumstances that interfere with the Auto−Parallelizing Option's ability to optimize programs. As shown in Chapter 2, "Understanding Incomplete Optimization," problems are sometimes caused by coding practices. Other times, the MIPSpro APO does not have enough information to make good parallelization decisions. You can pursue three strategies to attack these problems and to achieve better results with the MIPSpro APO.

- The first approach is to modify your code to avoid coding practices that the MIPSpro APO cannot analyze well. Specific problems and solutions are discussed in Chapter 2, "Understanding Incomplete Optimization."
- The second strategy is to assist the MIPSpro APO with the manual parallelization directives. They are described in the *MIPSpro Compiling and Performance Tuning Guide*, and require the **−mp**compiler option. The MIPSpro APO is designed to recognize and coexist with manual parallelism. You can use manual directives with some loop nests, while leaving others to the MIPSpro APO. This approach has both positive and negative aspects.

  Positive: The manual parallelization directives are well defined and deterministic. If you use a manual directive, the specified loop is run in parallel.

  Note: This last statement assumes that the trip count is greater than one and that the specified loop is not nested in another parallel loop.

  Negative: You must carefully analyze the code to determine that parallelism is safe. Also, you must mark all variables that need to be localized.

- The third alternative is to use the automatic parallelization compiler directives to give the MIPSpro APO more information about your code. The automatic directives are described in "Compiler Directives for Automatic Parallelization". Like the manual directives, they have positive and negative features.

  Positive: The automatic directives are easier to use: They allow you to express the information you know without your having to be certain that all the conditions for parallelization are met.

  Negative: The automatic directives are hints and thus do not impose parallelism. In addition, as with the manual directives, you must ensure that you are using them safely. Because they require less information than the manual directives, automatic directives can have subtle meanings.

Compiler Directives for Automatic Parallelization

The Auto−Parallelizing Option recognizes three types of compiler directives:

- Fortran directives, which enable, disable, or modify features of the MIPSpro APO
- Fortran assertions, which assist the MIPSpro APO by providing it with additional information about the source program
- Pragmas, the C and C++ counterparts to Fortran directives and assertions

In practice, the MIPSpro APO makes little distinction between Fortran assertions and Fortran directives. The automatic parallelization compiler directives do not impose parallelism; they give hints and assertions to the MIPSpro APO to assist it in choosing the right loops. The section "Invoking the Auto−Parallelizing Option"gives more details on compiling with the MIPSpro APO. Table 3−1lists the directives, assertions, and pragmas that the MIPSpro APO recognizes.

Table 3−1 Auto−Parallelizing Option Directives, Assertions, and Pragmas

Compiler Directive	Meaning and Notes
C*$* NO CONCURRENTIZE #pragma no concurrentize	Varies with placement. Either do not parallelize any loops in a subroutine, or do not parallelize any loops in a file.
C*$* CONCURRENTIZE #pragma concurrentize	Override C*$* NO CONCURRENTIZE.
C*$* ASSERT DO (CONCURRENT) #pragma concurrent	Do not let perceived dependences between two references to the same array inhibit parallelizing. Does not require **−apo**
C*$* ASSERT DO (SERIAL) #pragma serial	Do not parallelize the following loop.
C*$* ASSERT CONCURRENT CALL #pragma concurrent call	Ignore dependences in subroutine calls that would inhibit parallelizing. Does not require **−apo**
C*$* ASSERT PERMUTATION (*array_name*) #pragma permutation (*array_name*)	Array *array_name* is a permutation array. Does not require **−apo**
C*$* ASSERT DO PREFER (CONCURRENT) #pragma prefer concurrent	Parallelize the following loop if it is safe.
C*$* ASSERT DO PREFER (SERIAL) #pragma prefer serial	Do not parallelize the following loop.

There are two important points to remember regarding the compiler directives:

- Three compiler directives affect the compiling process even if **−apo**is not specified.
  - **C*$* ASSERT DO (CONCURRENT)** and **#pragma concurrent** may affect optimizations such as loop interchange.
  - **C*$* ASSERT CONCURRENT CALL** and **#pragma concurrent call** also may affect optimizations such as loop interchange.
  - **C*$* ASSERT PERMUTATION** and **#pragma permutation** may affect any optimization that requires permutation arrays.
- The general compiler option **−LNO:ignore_pragmas**causes the MIPSpro APO to ignore all of the directives, assertions, and pragmas in this section.

C*$* NO CONCURRENTIZE and #pragma no concurrentize

The **C*$* NO CONCURRENTIZE** directive prevents parallelization. Its effect depends on its placement.

- When placed inside subroutines and functions, the directive prevents their parallelization. In the following example, no loops inside *SUB1()* are parallelized.

```
       SUBROUTINE SUB1
C*$* NO CONCURRENTIZE
         ...
       END
```

- When placed outside of a subroutine, **C*$* NO CONCURRENTIZE** prevents the parallelization of all subroutines in the file, even those that appear ahead of it in the file. Loops inside subroutines *SUB2()* and *SUB3()* are not parallelized in this example:

```
       SUBROUTINE SUB2
         ...
       END
C*$* NO CONCURRENTIZE
       SUBROUTINE SUB3
         ...
       END
```

C*$* CONCURRENTIZE and #pragma concurrentize

Placing the **C*$* CONCURRENTIZE** directive inside a subroutine overrides a **C*$* NO CONCURRENTIZE** directive placed outside it. In other words, this directive allows you to selectively parallelize subroutines in a file that has been made sequential with **C*$* NO CONCURRENTIZE**.

C*$* ASSERT DO (CONCURRENT) and #pragma concurrent

C*$* ASSERT DO (CONCURRENT) instructs the MIPSpro APO, when analyzing the loop immediately following this assertion, to ignore all dependences between two references to the same array. Be aware that if there are real dependences between array references, **C*$* ASSERT DO (CONCURRENT)** may cause the MIPSpro APO to generate incorrect code. The following example is a correct use of the assertion when $M > N$:

```
C*$* ASSERT DO (CONCURRENT)
       DO I = 1, N
         A(I) = A(I+M)
```

There are six facts to be aware of when using this assertion:

- If multiple loops in a nest can be parallelized, **C*$* ASSERT DO (CONCURRENT)** causes the MIPSpro APO to prefer the loop immediately following the assertion.
- Applying this directive to an inner loop may cause the loop to be made outermost by the MIPSpro APO's loop interchange operations.

- The assertion does not affect how the MIPSpro APO analyzes **CALL** statements. See "C*$* ASSERT CONCURRENT CALL and #pragma concurrent call".
- The assertion does not affect how the MIPSpro APO analyzes dependences between two potentially aliased pointers. See "Aliased Parameter Information" for a discussion of aliased pointers.
- This assertion affects the compilation even when **−apo**is not specified.
- The compiler may find some obvious real dependences. If it does so, it ignores this assertion.

C*$* ASSERT DO (SERIAL) and #pragma serial

C*$* ASSERT DO (SERIAL) instructs the Auto−Parallelizing Option not to parallelize the loop following the assertion. However, the MIPSpro APO may parallelize another loop in the same nest. The parallelized loop may be either inside or outside the designated sequential loop.

C*$* ASSERT CONCURRENT CALL and #pragma concurrent call

The **C*$* ASSERT CONCURRENT CALL** assertion instructs the MIPSpro APO to ignore the dependences of subroutine and function calls contained in the loop that follows the assertion. Other points to be aware of are the following:

- The assertion applies to the loop that immediately follows it and to all loops nested inside that loop.
- The assertion affects the compilation even when **−apo**is not specified.

The MIPSpro APO ignores the dependences in subroutine *FRED()* when it analyzes the following loop:

```
C*$* ASSERT CONCURRENT CALL
       DO I = 1, N
         CALL FRED
         ...
       END DO
       SUBROUTINE FRED
         ...
       END
```

To prevent incorrect parallelization, make sure the following conditions are met when using **C*$* ASSERT CONCURRENT CALL**:

- A subroutine inside the loop cannot read from a location that is written to during another iteration. This rule does not apply to a location that is a local variable declared inside the subroutine.
- A subroutine inside the loop cannot write to a location that is read from or written to during another iteration. This rule does not apply to a location that is a local variable declared inside the subroutine.

The following code shows an illegal use of the assertion. Subroutine *FRED()* writes to variable *T,* which is also read from by *WILMA()* during other iterations.

```
C*$* ASSERT CONCURRENT CALL
       DO I = 1,M
         CALL FRED(B, I, T)
         CALL WILMA(A, I, T)
       END DO
       SUBROUTINE FRED(B, I, T)
         REAL B(*)
         T = B(I)
       END
       SUBROUTINE WILMA(A, I, T)
         REAL A(*)
         A(I) = T
       END
```

By localizing the variable *T*, you can manually parallelize the above example safely. But the MIPSpro APO does not know to localize *T*, and it illegally parallelizes the loop because of the assertion.

C*$* ASSERT PERMUTATION and #pragma permutation

When placed inside a subroutine, **C*$* ASSERT PERMUTATION (***array_name***)** tells the MIPSpro APO that *array_name* is a *permutation* array: Every element of the array has a distinct value. The assertion does not require the permutation array to be *dense*. In other words, while every *IB(I)* must have a distinct value, there can be gaps between those values, such as *IB(1)* = 1, *IB(2)* = 4, *IB(3)* = 9, and so on.

Array *IB* is asserted to be a permutation array for both loops in *SUB1()* in this example.

```
             SUBROUTINE SUB1
               DO I = 1, N
                 A(IB(I)) = ...
               END DO
      C*$* ASSERT PERMUTATION (IB)
               DO I = 1, N
                 A(IB(I)) = ...
               END DO
             END
```

There are three points to be made about this assertion:

- As shown in the example, you can use this assertion to parallelize loops that use arrays for indirect addressing. Without this assertion, the MIPSpro APO cannot determine that the array elements used as indexes are distinct.
- **C*$* ASSERT PERMUTATION** (*array_name*) affects every loop in a subroutine, even those that appear ahead it.
- The assertion affects compilation even when **−apo**is not specified.

C*$* ASSERT DO PREFER (CONCURRENT) and #pragma prefer concurrent

C*$* ASSERT DO PREFER (CONCURRENT) instructs the Auto−Parallelizing Option to parallelize the loop immediately following the assertion, if it is safe to do so. This assertion is always safe to use. Unless it can determine the loop is safe, the MIPSpro APO does not parallelize a loop because of this assertion.

The following code encourages the MIPSpro APO to run the *I* loop in parallel:

```
C*$* ASSERT DO PREFER (CONCURRENT)
       DO I = 1, M
         DO J = 1, N
           A(I,J) = B(I,J)
         END DO
         ...
       END DO
```

When dealing with nested loops, the Auto−Parallelizing Option follows these guidelines:

- If the loop specified by this assertion is safe to parallelize, the MIPSpro APO chooses it to parallelize, even if other loops in the nest are safe.
- If the specified loop is not safe to parallelize, the MIPSpro APO uses its heuristics to choose among loops that are safe.
- If this directive is applied to an inner loop, the MIPSpro APO may make it the outermost loop.
- If this assertion is applied to more than one loop in a nest, the MIPSpro APO uses its heuristics to choose one of the specified loops.

C*$* ASSERT DO PREFER (SERIAL) and #pragma prefer serial

The *C*$* ASSERT DO PREFER (SERIAL)* assertion instructs the Auto−Parallelizing Option not to parallelize the loop that immediately follows. It is essentially the same as *C*$* ASSERT DO (SERIAL)*. In the following case, the assertion requests that the *J* loop be run serially:

```
       DO I = 1, M
C*$* ASSERT DO PREFER (SERIAL)
         DO J = 1, N
           A(I,J) = B(I,J)
         END DO
         ...
       END DO
```

The assertion applies only to the loop directly after the assertion. For example, the MIPSpro APO still tries to parallelize the *I* loop in the code shown above. The assertion is used in cases like this when the value of *N* is very small.