

On the Implementation of OpenMP 2.0 Extensions in the Fujitsu PRIMEPOWER compiler

Hidetoshi Iwashita¹, Masanori Kaneko¹, Masaki Aoki¹, Kohichiro Hotta¹, and
Matthijs van Waveren²

¹ Software Technology Development Division
Software Group, Fujitsu Ltd.
140 Miyamoto
Numazu-shi, Shizuoka 410-0396, Japan

² Fujitsu Systems Europe
8, rue Maryse Hilsz
Parc de la Plaine
31500 Toulouse, France
waveren@fujitsu.fr

Abstract. The OpenMP Architecture Review Board has released version 2.0 of the OpenMP Fortran language specification in November 2000, and version 2.0 of the OpenMP C/C++ language specification in March 2002. This paper discusses the implementation of the OpenMP Fortran 2.0 WORKSHARE construct, NUM_THREADS clause, COPYPRIVATE clause, and array REDUCTION clause in the Parallelnavi software package. We focus on the WORKSHARE construct and discuss how we attain parallelization with loop fusion.

1 Introduction

The OpenMP Architecture Review Board has released version 2.0 of the OpenMP Fortran language specification [1] in November 2000, and version 2.0 of the OpenMP C/C++ language specification [2] in March 2002. Currently the OpenMP ARB is working on the development of an integrated OpenMP Fortran/C/C++ language specification.

While OpenMP Fortran 1.1 was aligned with Fortran 77, OpenMP Fortran 2.0 has been updated to be aligned with Fortran 95. The WORKSHARE directive allows parallelization of array expressions in Fortran statements. The new NUM_THREADS clause on parallel regions defines the number of threads to be used to execute that region. The COPYPRIVATE clause has been added on END SINGLE. THREADPRIVATE may now be applied to variables as well as COMMON blocks. REDUCTION is allowed on an array name. COPYIN works on variables as well as COMMON blocks, and reprivatization of variables is now allowed.

While OpenMP C/C++ 1.0 was aligned with OpenMP Fortran 1.0 and C89, OpenMP C/C++ 2.0 has been brought up to the level of C99 and OpenMP Fortran 2.0. This means that some of the extensions mentioned above for OpenMP Fortran 2.0 have been introduced in OpenMP C/C++ 2.0, viz. the `num_threads` clause, the `copyprivate` clause, and the extension on the `threadprivate` directive.

Parallelnavi is a software package for the PRIMEPOWER series server. Parallelnavi Fortran V2.1 supports the OpenMP Fortran API Version 2.0. Parallelnavi C/C++ V2.1 supports the OpenMP C/C++ API Version 2.0. This paper describes the implementation of the OpenMP Fortran 2.0 extensions in the Parallelnavi software package. We focus on the `WORKSHARE` construct and discuss how we attain parallelization with loop fusion.

The Fujitsu PRIMEPOWER HPC2500 [3] is a parallel computation server supporting up to 128 CPUs and 512 gigabytes of memory. The CPU used is SPARC64GP, which conforms to the SPARC International V9 architecture and loads the Solaris 8 operating system. Each cabinet (node) has 8 system boards and each system board has four CPUs, 16 gigabytes of memory, six PCI cards and a first level (L1) crossbar switch. All system boards, within and between nodes, are connected with the second level (L2) crossbar switch.

A distinguishing feature of the PRIMEPOWER HPC2500 is its behavior as a flat Symmetric Multi-processing (SMP) server. Two-layered crossbar networks allow every CPU to access all memory in the system and guarantee coherency.

2 OpenMP API v2.0 Compiler Support

The Fujitsu Solaris compiler supports the OpenMP API V2.0, which includes the following major features:

- `WORKSHARE` construct,
- `NUM_THREADS` clause,
- `COPYPRIVATE` clause, and
- Array `REDUCTION` clause.

The `NUM_THREADS` clause, which specifies the number of threads to execute a parallel region, was implemented in a similar fashion as the `OMP_SET_NUM_THREADS` library call.

The `COPYPRIVATE` clause was implemented to make the threads keep the following order: 1) the `SINGLE` thread specified with the `COPYPRIVATE` clause writes out the value of the variable to a global temporary, 2) all threads perform barrier synchronization, and 3) every thread reads the global temporary into its own private variable.

The implementation of the array `REDUCTION` is a simple and natural expansion of the scalar `REDUCTION` in the OpenMP API V1.1.

The following part of this section introduces the implementation of the `WORKSHARE` construct, which is the most important new feature of the OpenMP API V2.0.

2.1 Compilation of WORKSHARE Construct

The configuration of the Fujitsu Solaris compiler is shown in Figure 1, focusing on the treatment of OpenMP code. The OMP module, a part of Middle Pass A, interprets all OpenMP directives except WORKSHARE and generates corresponding execution code and runtime system (RTS) calls.

The OMP2 module was added to support the WORKSHARE construct. It converts the following statements and constructs into code corresponding to DO loops enclosed with OpenMP DO directives.

- Array assignment statement
- FORALL statement/construct
- WHERE statement/construct

The OMP2 module follows the MPV module, which converts array assignment statements, FORALL and WHERE constructs into loops. As a result, the implementation of the WORKSHARE block is reduced to a problem of loop parallelization.

The OMP2 module regards a sequence of scalar assignment statements, as if they were enclosed with the SINGLE construct.

Some Fortran 90 array intrinsic functions are parallelized in the WORKSHARE construct, as described in Sect. 2.3.

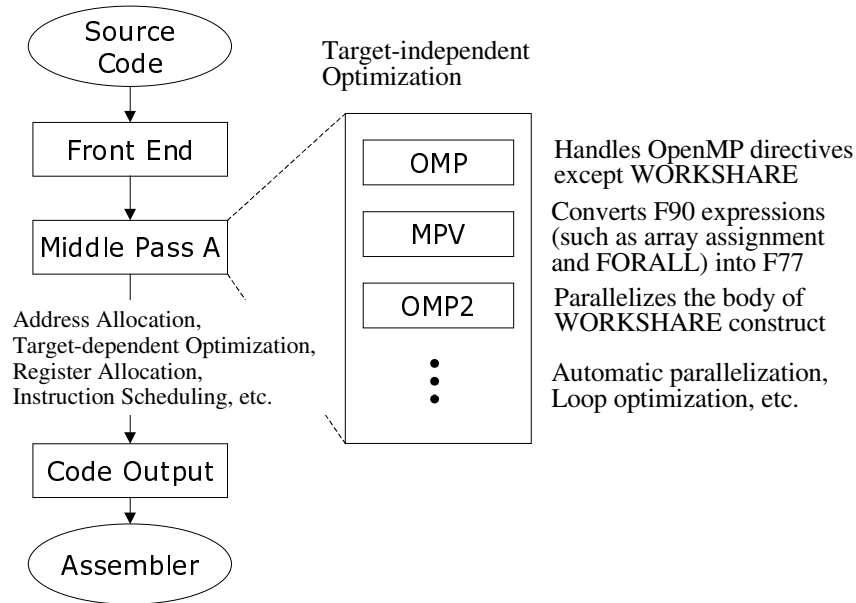


Fig. 1. Configuration of the Fujitsu Solaris compiler that supports the OpenMP API V. 2.0

2.2 Loop Fusion in WORKSHARE Construct

The Fortran 95 language specification [4] defines an array assignment statement as an execution sequence, which includes several DO loops. For efficiency of the generated code, the loops must be fused as much as the data dependence allows. In our implementation, the MPV module not only expands the Fortran 90 conventions into loops, but also fuses the loops using data dependence analysis by taking into account the characteristics of the original statements.

An example of the WORKSHARE construct is shown in (a) of Fig. 2. In this program fragment, the following data dependences can be found with data dependency analysis [7].

Table 1.

u1 to u3	loop carried (+1)	anti-dependence
u2 to u3	loop independent	anti-dependence
u3 to u4	loop independent	flow-dependence
u3 to u5	loop carried (+1)	flow-dependence

If the two OpenMP directive lines were absent, the MPV module would perfectly fuse the three array assignment statements into one loop as shown in (b), taking into account avoidance of the problem of data dependence. As the result, the generated loop could not be parallelized however because it would have loop-carried data dependence.

```

!$omp workshare
1  A(1:N)=0.5*(A(2:N+1)+A(1:N))
2  C(1:N)=A(1:N)+B(1:N)
3  D(1:N)=A(0:N-1)+B(1:N)
!$omp end workshare

```

(a) An example of WORKSHARE construct

```

do i=1,N
1  A(i)=0.5*(A(i+1)+A(i))
2  C(i)=A(i)+B(i)
3  D(i)=A(i-1)+B(i)
end do

```

(b) Loop expansion and fusion in Fortran context

```

!$omp do
do i=1,N
11 tmp(i)=0.5*(A(i+1)+A(i))
end do
!$omp do
do i=1,N
12 A(i)=tmp(i)
end do
!$omp do
do i=1,N
2 C(i)=A(i)+B(i)
3 D(i)=A(i-1)+B(i)
end do

```

(c) WORKSHARE conversion with loop fusion

Fig. 2. Example of WORKSHARE conversion with loop fusion

In order to have parallelization instead of fusion, we modified the MPV module and changed the condition of fusion for the body of WORKSHARE as follows:

- If an array assignment statement contains loop-carried data dependence, separated loops connected with temporary arrays will be generated. See lines 11 and 12 of (c) corresponding to line 1 of (a) for example.
- If two array assignment statements have a mutual loop-carried data dependence, fusion of the loops generated from them will be suppressed. As shown in (c) of Fig. 2, lines 12 and 3 must be separated into different loops because of their loop-carried data dependence.

As the result of the modification, the MPV module still fuses many loops that have no loop-carried data dependence and the OMP2 module can parallelize generated loops, as shown in (c).

2.3 Parallelization of Array Intrinsic Functions

The compiler parallelizes the execution of the Fortran 90 transformational functions MATMUL, DOT_PRODUCT and TRANSPOSE in the body of WORKSHARE in a manner selected from the following, according to the values and the attributes of the arguments:

- Either the Front End or the MPV modules perform inline-expansion of the function call. Subsequently the OMP2 module receives the result just like DO loops written in the WORKSHARE construct and parallelizes them as much as possible.
- The OMP2 module replaces the function call with a parallel version if it is in the context to be able to be parallelized. The parallel version library assumes that it is invoked in a team of threads concurrently in a redundant region. The OMP2 module also generates barrier synchronization if data dependence with the neighboring statements requires it.

3 Conclusion

This paper discusses the implementation of the OpenMP 2.0 WORKSHARE construct, NUM_THREADS clause, COPYPRIVATE clause and array REDUCTION clause. We focus on the WORKSHARE construct, and show how parallelization with loop fusion can be attained.

References

1. OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface Version 2.0, November 2000. (<http://www.openmp.org/specs/mp-documents/fspec20.pdf>)

2. OpenMP Architecture Review Board. OpenMP C/C++ Application Program Interface Version 2.0, March 2002. (<http://www.openmp.org/specs/mp-documents/cspeg20.pdf>)
3. Iwashita H., Yamanaka E., Sueyasu N., Waveren M. van, and Miura K: The SPEC OMP2001 Benchmark on the Fujitsu PRIMEPOWER System. In Proc. of EWOMP2001.
4. ISO/IEC 1539-1:1997, Information Technology - Languages - Fortran
5. Iwashita H., Okada S., Nakanishi M., Shindo T., and Nagakura H: VPP Fortran and Parallel Programming on the VPP500 Supercomputer. In Proceedings of the 1994 International Symposium on Parallel Architectures, Algorithms and Networks (poster session papers), pages 165-172, Kanazawa, Japan, December 1994.
6. Iwashita H., Sueyasu N., Kamiya S., and Waveren, M. van. VPP Fortran and the Design of HPF/JA Extensions, Concurrency and Computation: Practice and Experience.14:575-588,2002
7. Chandra R., Dagum L., Kohr D., Maydan D., MacDonald J., and Menon R.. Parallel Programming in OpenMP. Academic Press, San Diego, CA, USA, 2001.
8. Almasi G., Gottlieb A.. Highly Parallel Computing. The Benjamin/Cummings Publ. Company, Inc, Redwood City, CA, USA, 1989.