CCRG OpenMP: Experiments and Improvements

Huang Chun & Yang Xuejun

National Laboratory for Parallel and Distributed Processing, P.R. China
Outline

• Motivations
• CCRG OpenMP Compiler
• Optimized STATIC Schedule Implementation
• Inter-Procedural Optimization
• Conclusion and Future Work
Motivations

• Provide an open source OpenMP compiler infrastructure
  – Portable
  – Productive
  – Robust

• Provide a platform for building
  – Performance analysis system and debug tool for OpenMP applications
  – Static analyzer to help user to correct OpenMP applications
Main Contributions

- CCRG OpenMP Fortran95 Compiler
- Performance evaluation & analysis
- Source-level optimization
  - Static schedule implementation
  - Inter-procedural optimization
CCRG OpenMP Compiler

• Source-to-Source Compiler
• CCRG OpenMP Compiler is consist of
  – Translator
    • Transform OpenMP program to equivalent Fortran code
  – Runtime library
  – Native Compiler
    • GNU GCC
    • Commercial Compilers, such as Intel Compiler

• Features
  – Support Fortran95 languages
  – Use ENTRY statement to reduce the size of the code
  – Portable implementation of OpenMP for SMPs and SDSM
Translator

- Based on Sage++*
  - Fortran OpenMP syntax parser – f2dep
    - Add syntax description for OpenMP directives

```fortran
omp_directive:
  omp_parallel
  | omp_paralleldo
  | omp_parallelworkshare
  | ......
omp_parallel:
  PARALLEL end_spec needkeyword omp_clause_opt keywordoff
  {
    omp_binding_rules (OMP_PARALLEL_NODE);
    $$ = get_bfnd (fi, OMP_PARALLEL_NODE, SMNULL,
    $4, LLNULL, LLNULL);
  }
```

- Translator – omp2f

* See www.extreme.indiana.edu/sage for more information.
SUBROUTINE test()
    DIMENSION a(100)
!$OMP PARALLEL DO NUM_THREADS(4)
    DO 100 k = 1, 100
100 a(k) = 0.9
!$OMP PARALLEL ...
!$OMP END PARALLEL
END

SUBROUTINE test()
    DIMENSION a(100)
    EXTERNAL test_$1, test_$2
    CALL comp_runtime_init()
    CALL comp_parallel(test_$1, 4, 1, a)
    CALL comp_parallel(test_$2, ......)
    CALL comp_exit()
END
SUBROUTINE test_$0(a)
    DIMENSION a(100)
    INTEGER lc_k
    INTEGER _omp_dolo, _omp_dohi, comp_static_more
    ENTRY test_$1(a)
        CALL comp_static_setdo (1, 100, 1, 0)
        DO WHILE (comp_static_more(_omp_dolo, &
                        _omp_dohi, 1).eq.1)
            DO 100 lc_k = _omp_dolo, _omp_dohi, 1
                a(lc_k) = 0.9
            END DO
        CALL comp_barrier()
        RETURN
    ENTRY test_$2(a)
        .......
    RETURN
END
Performance Results

Base Ratios of CCRG and Intel without IPO*  

Base Ratios of CCRG and Intel without IPO*

Intel Fortran Compiler 8.0 is used as the naïve compile of CCRG OpenMP Compiler
* “-ipo” option enables inter-procedural optimization(IPO) across files.
Performance Results

• Most of SPEC OMP Fortran Programs show good performance as Intel Compiler
• Why mgrid & wupwise perform poorly?
  – mgrid
  – wupwise
Profile of mgrid

- "-O3"
- "TRAIN" input set is used
Implementation of \textbf{DO} Directive

\begin{Verbatim}
CALL \texttt{comp\_type\_setdo} (lo, hi, in, chunk)
DO \textbf{WHILE} (\texttt{comp\_type\_more(\_omp\_dolo, \_omp\_dohi, in)} \(\&\) eq. 1)
  DO 100 lc\_k = \_omp\_dolo, \_omp\_dohi, in
  100 \hspace{1cm} a(lc\_k) = 0.9
END DO
\end{Verbatim}

- \textbf{DO WHILE} loop is introduced to implement the schedule clause of OpenMP
- All of the schedule types of OpenMP use the same approach
- \textit{type} is one of the following:
  - static
  - dynamic
  - guided
  - runtime
Optimized STATIC Schedule

• **DO WHILE** loop can be omitted if
  – No **SCHEDULE** clause.
  – **SCHEDULE(STATIC)** is specified.
  – Static schedule
    • Both chunk size, number of iteration and number of threads are known during compile time
    • \((\text{chunk size} \times \text{number of threads}) \leq \text{number of iteration}\).

```plaintext
CALL comp_static_setdo(1,100,1,0)
CALL comp_static_once(_omp_dolo, _omp_dohi, 1)
DO 100 lc_k = _omp_dolo, _omp_dohi, 1
100   a(lc_k) = 0.9
```

• **SCHEDULE** clause is not specified in most of OpenMP programs
Profile of mgrid after Optimization

![Bar graph showing execution time for different tasks with labels: resid_p1, psinv_p1, rprj3_p1, interp_p1, interp_p2, comm3_p1. Y-axis represents execution time in seconds, ranging from 0 to 50. The tasks are compared across three categories: CCRG, CCRG-Opt, and Intel.]
• Inter-Procedural Optimization (IPO)
• Source-to-source transformation cannot keep the information about the caller-callee relationship between the original procedures.

<table>
<thead>
<tr>
<th></th>
<th>CCRG</th>
<th></th>
<th>Intel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>subroutine</td>
<td>execution time</td>
<td>subroutine</td>
<td>execution time</td>
</tr>
<tr>
<td>1</td>
<td><strong>zgemm</strong></td>
<td>82.10</td>
<td>dlaran</td>
<td>9.77</td>
</tr>
<tr>
<td>2</td>
<td>gammul</td>
<td>10.77</td>
<td>zaxpy</td>
<td>8.57</td>
</tr>
<tr>
<td>3</td>
<td>zaxpy</td>
<td>7.74</td>
<td><strong>zgemm</strong></td>
<td>7.91</td>
</tr>
<tr>
<td>4</td>
<td>dlaran</td>
<td>7.35</td>
<td>lsame</td>
<td>1.87</td>
</tr>
</tbody>
</table>
SU3MUL & ZGEMM

```
SUBROUTINE SU3MUL(U,TRANSU,X,RESULT)

...........
    CALL ZGEMM(TRANSU, 'NO TRANSPOSE', 3, 4, 3,
    & ONE,U,3,X,3,ZERO,RESULT,3)
RETURN
END

SUBROUTINE ZGEMM (TRANSA,TRANSB,M,N,K,
& ALPHA,A,LDA,B,LDB,BETA,C,LDC )

...........
END
```

- **M, N, K** are loop control variables in ZGEMM
- **M, N, K** are used in the logical expression of IF statement
- The values of M,N,K have not been propagated to zgemm when using CCRG
Inter-Procedural Optimization

First Pass
Parser: f2dep → tmp_filename.i

Second Pass
Inter-Procedural Analysis
Parser: f2dep → dep
Translator: omp2f
filename_omp.f
Intermediate file in IPO

tmp_su3mul.i :

{ SUBROUTINE "SU3MUL"
  (FORMAL ("U" COMPLEX*16 DIMENSION(2 3 *))
   ("TRANSU" CHARACTER*1 SCALAR)
   ("X" COMPLEX*16 DIMENSION(1 *))
   ("RESULT" COMPLEX*16 DIMENSION(1 *))
  )

(SUBROUTINE "ZGEMM"
  (ACTUAL (TRANSU, 'NO TRANSPOSE', 3,4,3,
           ONE,U,3,X,3,ZERO,RESULT,3)
  )
}

COMP
CCRG Open MP
SUBROUTINE ZGEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
! Variables Declaration Statements...........
! Assignment to Formal parameters
M = 3
N = 4
K = 3
!Other Executable Statements
END
Performance of wupwise after IPO

![Bar chart showing execution time in seconds for different systems.]

- CCRG
- CCRG-OPT
- Intel
Conclusions

• With CCRG OpenMP compiler, all of SPEC OMP programs can be compiled and executed on SMP machines efficiently.
• To improve the performance, it is necessary and feasible for OpenMP compilers to optimize programs at the source level.
Future Work

• Fortran 95 Syntax
  – `KIND`
    ```fortran
    integer, parameter:: b8 = selected_real_kind(14)
    real(b8) a
    The value of b8 should be calculated by the translator.
    ```

• Source-level Optimization
  – Data Privatization
    • e.g., `FIRSTPRIVATE`
  – More Classic Optimization