⑧ Interprocedural Multigrain Parallelization

Yuichiro AOKI [1, 2]

[1] APC Technology Group [2] Hitachi, Ltd.

Contents

1. Introduction

- **Background**

  Exploiting thread-level parallelism of a program is indispensable for parallelizing compilers on SMPs.

- **Problem**

  Most of the commercial parallelizing compilers exploit loop parallelism only.

⬇

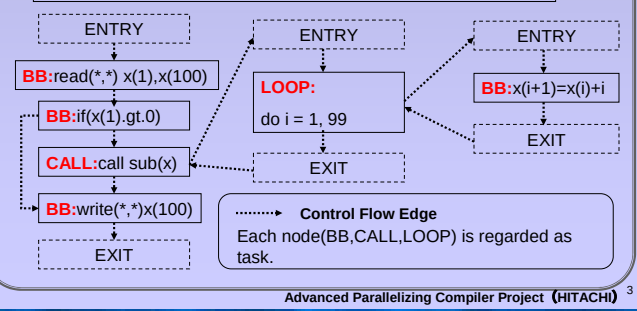
- **Purpose**

  To develop a compiler module that exploits interprocedural task parallelism of the program on top of loop parallelism.

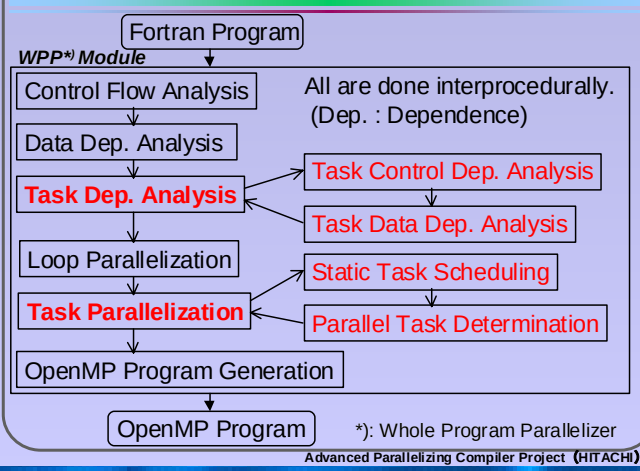
2. Macro FLow Graph(MFG)

Example：

```
program main                    subroutine sub(x)
read(*,*) x(1),x(100)           do i = 1, 99
if(x(1).gt.0) call sub(x)         x(i+1)=x(i)+i
write(*,*) x(100)               enddo
```



3.1 Structure



3.2 Data Dependence Information

- Definition -

mod ：The set of variables possibly defined. ⊆ Var
euse：The set of variables possibly used before definition. ⊆ Var
rmod：The set of pairs of reaching-mod variables and tasks including their mod refs. ⊆ Var×Task
ruse ：The set of pairs of reaching-use variables and tasks including their use refs. ⊆ Var×Task

Var： The set of all variables
Task：The set of all tasks

3.3 Task Data Dependence Analysis

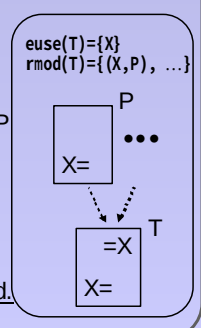
Analyze task data dependence for each layer of MFG. (task:node of MFG)

Ex. The case of flow dependence

P, T： Task
pr_P：Var×Task→Var；Projection to Task P

$euse(T) \cap pr_P(rmod(T)) \neq \phi$
⇒Flow Dependence of Task T to P.

Then macro-flow graph(MFG) is generated.



3.4 Task Parallelization(1/5)

- Policy -

The task parallelization phase

- **applies static scheduling.**
  (We assume that the number of processors is given at compile time.)
- **doesn't exploit nested parallelism.**
- gives higher priority to loop parallelism over task parallelism.
- assigns each thread team associated with a certain parallel loop to all processors.

3.4 Task Parallelization(2/5)

- Algorithm -

(1) Apply CP/MISF-based static scheduling to tasks in each layer of MFG. Scheduled tasks must be under the same conditions.
(2) Evaluate predicted grain size (PGS) of each layer.

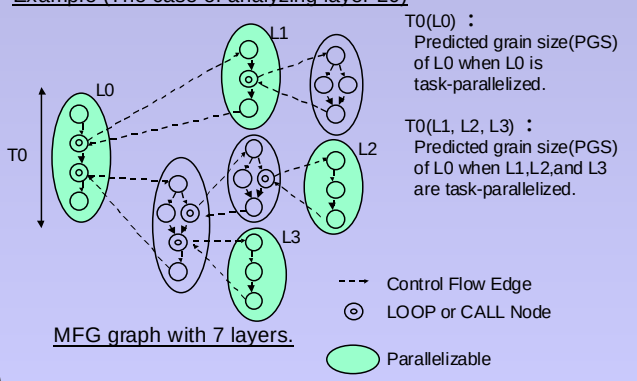
The following is done traversing MFG in reverse DFN order.

(3) Compare the PGS of current working layer with those of its parallelizable descendant layers.
(4) Choose the layer with the shortest PGS as the parallel candidate.

(DFN : Depth-First Numbering)

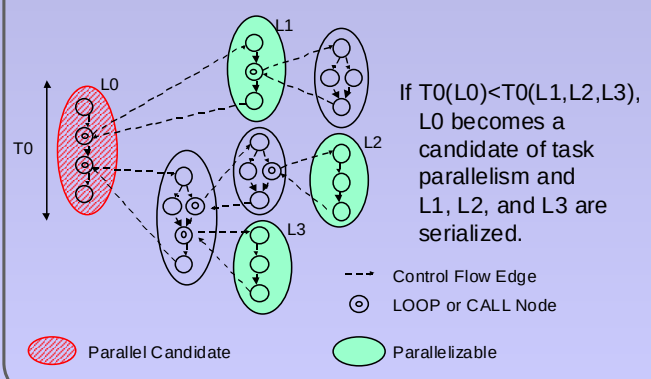
3.4 Task Parallelization(3/5)

Example (The case of analyzing layer L0)



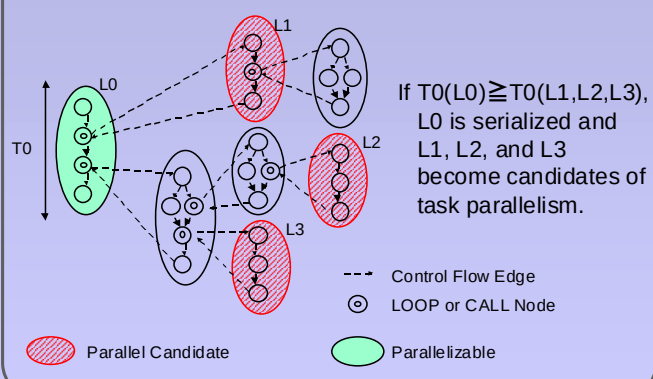
3.4 Task Parallelization(4/5)

Example (The case of analyzing layer L0)



3.4 Task Parallelization(5/5)

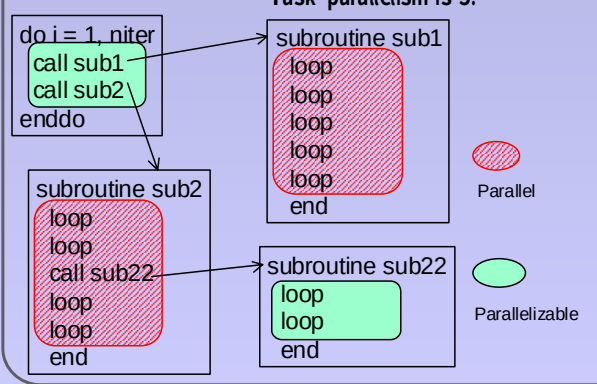
Example (The case of analyzing layer L0)



4. Evaluation

Input program

All loops are unparallelizable.
Task-parallelism is 5.



4. Evaluation

Output Source Image of Parallel Task :

```
Initialize task start flags.
Initialize task end flags.
!$omp parallel private(...)
  Get thread number.
  Initialize lock variables.
  while(! (All task end flags
        are set)) {
    <Task1>
    <Task2>
      :
  } end while
!$omp end parallel
```

```
<Task1>
C Task1[starts]----------
    <Critical Section Starts>
    if (Task1 start flag is initial value) then
      Task1 start flag = thread number
    endif
    <Critical Section Ends>
    if (Task1 start flag is thread number) then
      <Task1 Body>
      Set task1 end flag.
    endif
C Task1[ends]------------
```


4. Evaluation

[Environment]

SGI® Origin® 2000 (R10000® 195MHz x 32, Memory 11GB)
Compiler：MIPSpro™ Fortran90 Ver7.30
- Options：
  Serial： -Ofast=ip27 -LNO:fusion=2
  Parallel (Conventional)： (Serial) ＋ -apo (automatic parallelization)
  Parallel (This Study)： (Serial) ＋ -mp (OpenMP parallelization)

[Performance]

Task parallel program ran 2.75 times faster than the conventional one in 5 processors.

Num of PEs	Serial	1	2	3	4	5	6	7	8
Conventional[sec.]	9.85	9.77	9.78	9.78	9.78	9.79	9.77	9.63	9.77
This Study[sec.]	—	8.80	7.46	4.98	4.35	3.55	3.81	3.81	3.80
Speedup	—	1.11	1.31	1.96	2.25	**2.75**	2.56	2.56	2.57

cf.) Speedup = (The fastest result of the conventional case)/(This study).

5. Conclusion

1. We have developed interprocedural multigrain parallelization technique in our module.
   This technique has the following features :

   + Interprocedural Task Dependence Analysis.
   + CP/MISF-based Static Task Scheduling.

2. Initial evaluation shows effectiveness of our technique.