Coarse-Grain Task Parallel Processing Using the OpenMP Backend of the OSCAR Multigrain Parallelizing Compiler

- K. Ishizaka, M. Obata, H. Kasahara
- Waseda University, Japan

Research Background

- Wide use of SMP
  from single chip multiprocessor to supercomputer
- Increasing the number of processors...
  - Gap between peak performance
    and effective performance
  - Difficulty of programming
    by users to improve performance
- Advanced new automatic parallelizing compiler
  - Cost Performance
  - Ease of use
  - Effective performance

Automatic parallelizing compiler

- Automatic loop parallelization
  - <Analysis & Restructuring>
    - Runtime dependence analysis, Array privatization, Symbolic analysis, Interprocedure analysis, Unimodular transformation and so on...
- Polaris, SUIF, Parafrase2, ...
  - Limitation of loop parallelism
    - Sequential loops and parts of program except loops
    - Fewer loop iterations than the number of processors

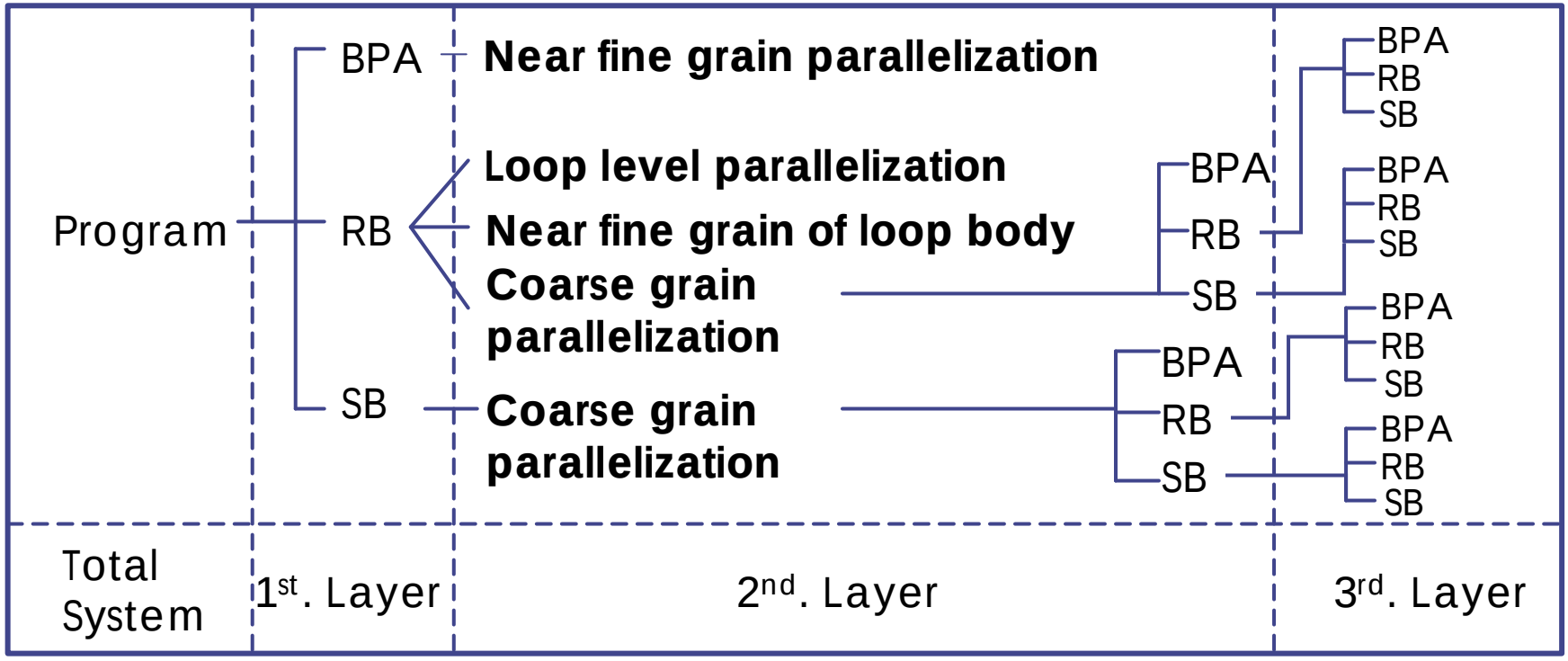
Multigrain parallelization

- Improvement of effective performance
- Scalability
  - Coarse grain parallelism:
    subroutines, loops, basic blocks
  - Near fine grain parallelism: statements
  - in addition to loop parallelism

- Multi level parallelism:
  - PROMIS compiler (Parafrase2 + EVE)
  - NANOS compiler (NthLIB, extended OpenMP)
  - OSCAR Fortran compiler

Generation of coarse grain tasks

- Program is decomposed into macro-tasks (MTs).
  - Block of Pseudo Assignments (BPA): Basic Block (BB)
  - Repetition Block (RB) : natural loop
  - Subroutine Block (SB): subroutine



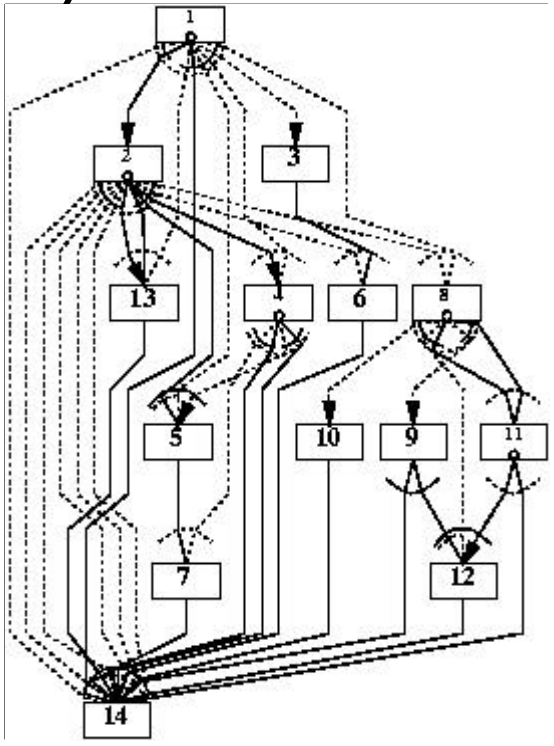
Earliest Executable Condition Analysis

- Conditions on which MT may begin its execution earliest.
  - (Condition for determination of MT execution)
    AND
    (Condition for Data access)

  - represented on
    Macro-Task-Graph(MTG)

Earliest Executable Condition of MT6

MT2 takes a branch
that guarantees MT4 will execute
OR
MT3 completes execution

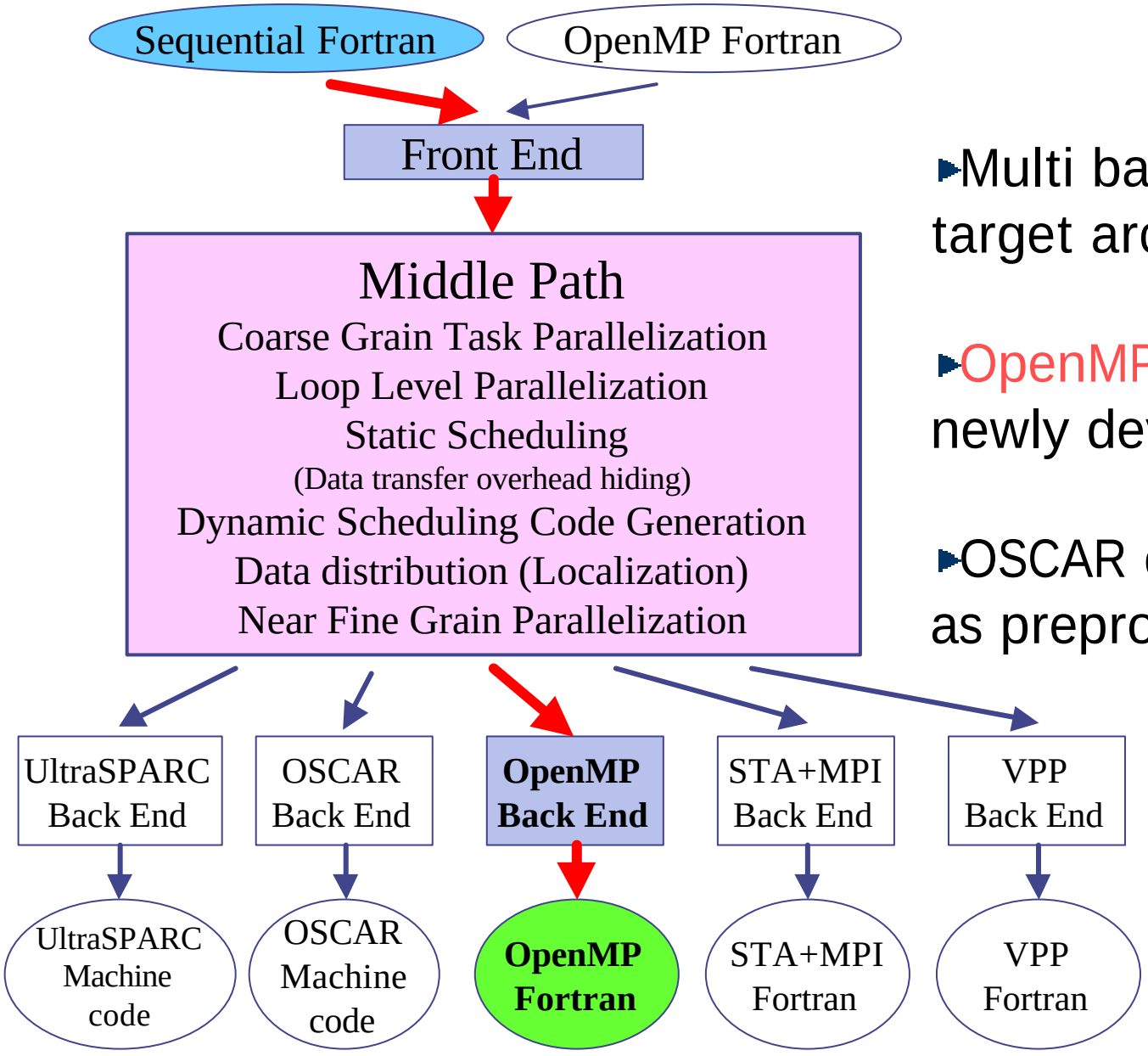


Macro-Task Graph

Coarse grain task scheduling

- Static Scheduling
  - At compilation-time,
    MTs are assigned to thread groups statically.
  - Minimize scheduling overhead
- Centralized/Distributed dynamic scheduling
  - At run-time,
    MTs are assigned dynamically by embedded scheduling code generated by OSCAR compiler.
  - Cope with runtime uncertainty like conditional branchs
- selectable for each layer of MTG

OSCAR Fortran Compiler



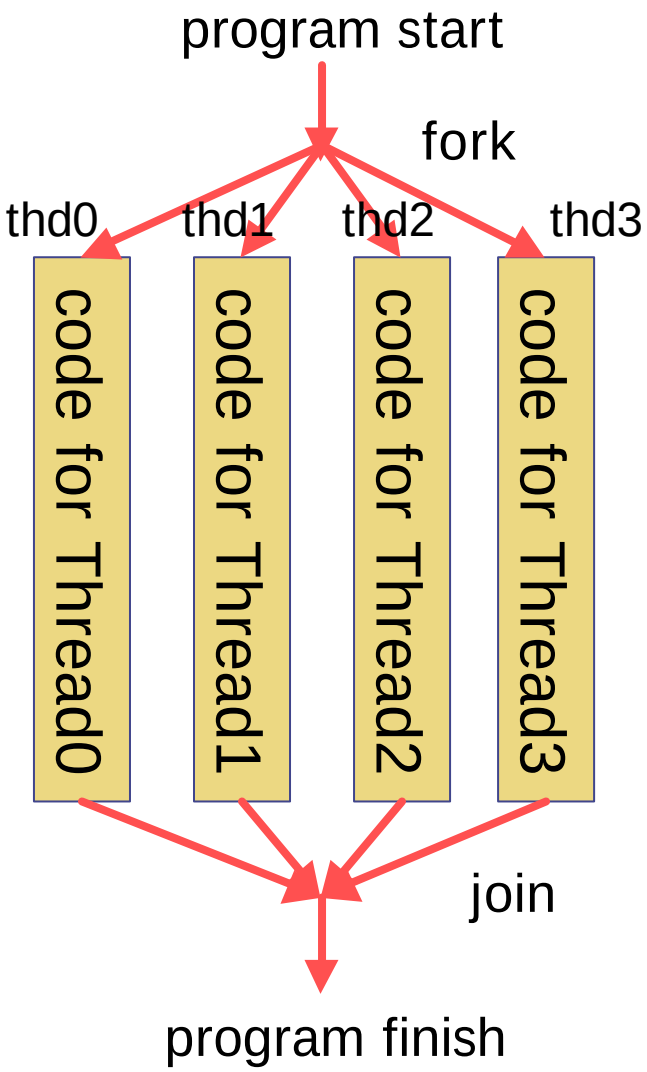
- Multi backends for multi target architectures
- OpenMP backend is newly developed
- OSCAR compiler is used as preprocessor

Code generation of OpenMP BackEnd

- Threads are forked only once at the beginning of a program by OpenMP “PARALLEL SECTIONS” directive
- These threads join only once at the end of program
- Compiler generates codes for each threads in each section

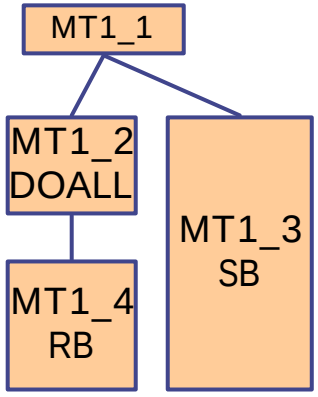
Generation of Threads

```
!$OMP PARALLEL SECTIONS
        code for thread0
!$OMP SECTION
        code for thread1
!$OMP SECTION
        code for thread2
!$OMP SECTION
        code for thread3
!$OMP END PARALLEL SECTIONS
```



Static Scheduling

- Compiler generates
  - different code in each section according to static scheduling result

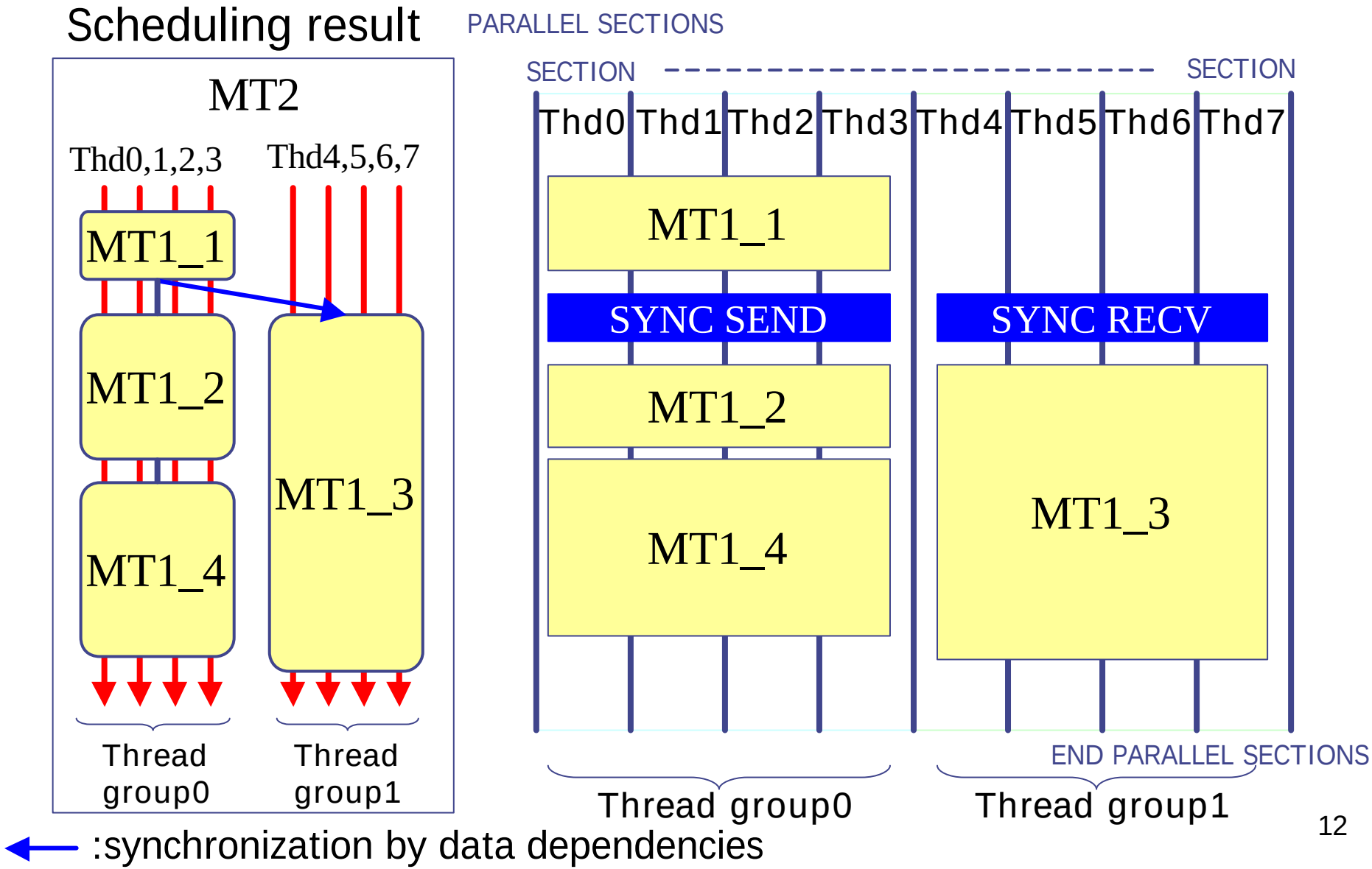


MTG of program (1st layer)

Example: 8processer system

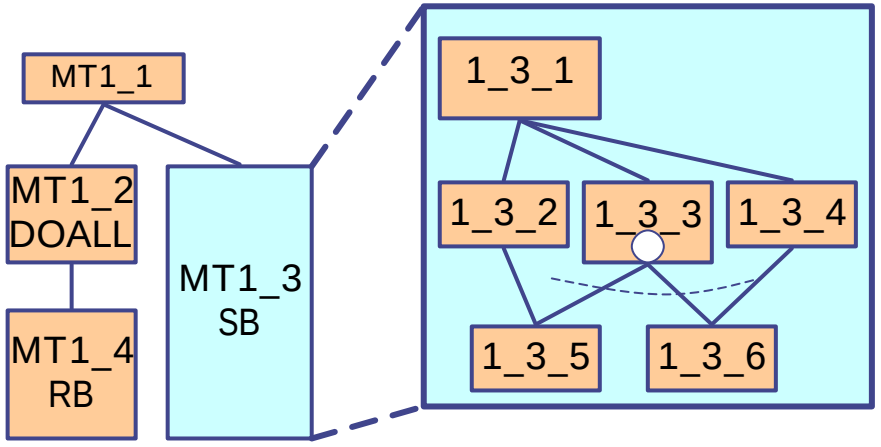
schedule MTs to 2 thread groups composed of 4 threads

Code image of Static Scheduling



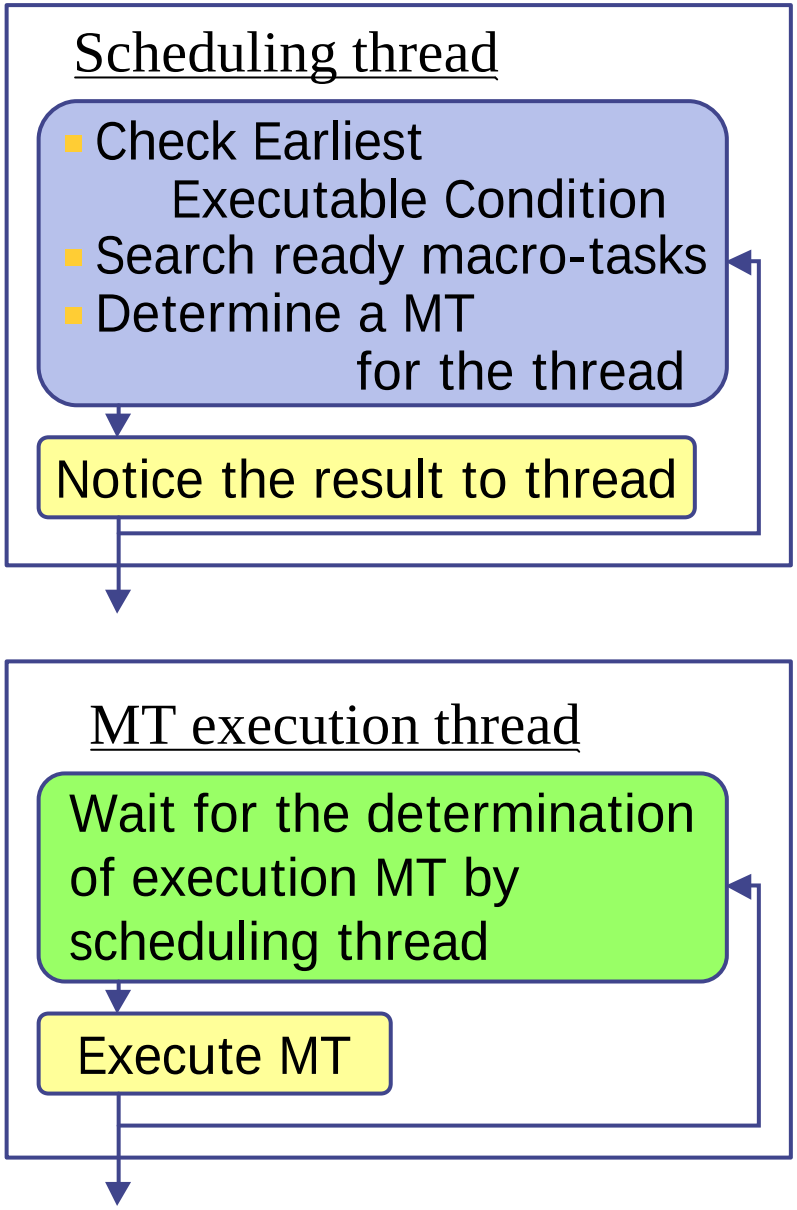
Centralized Dynamic Scheduling

- Compiler generates
  - Scheduling code in the section for a scheduler thread
  - all MTs codes in other sections

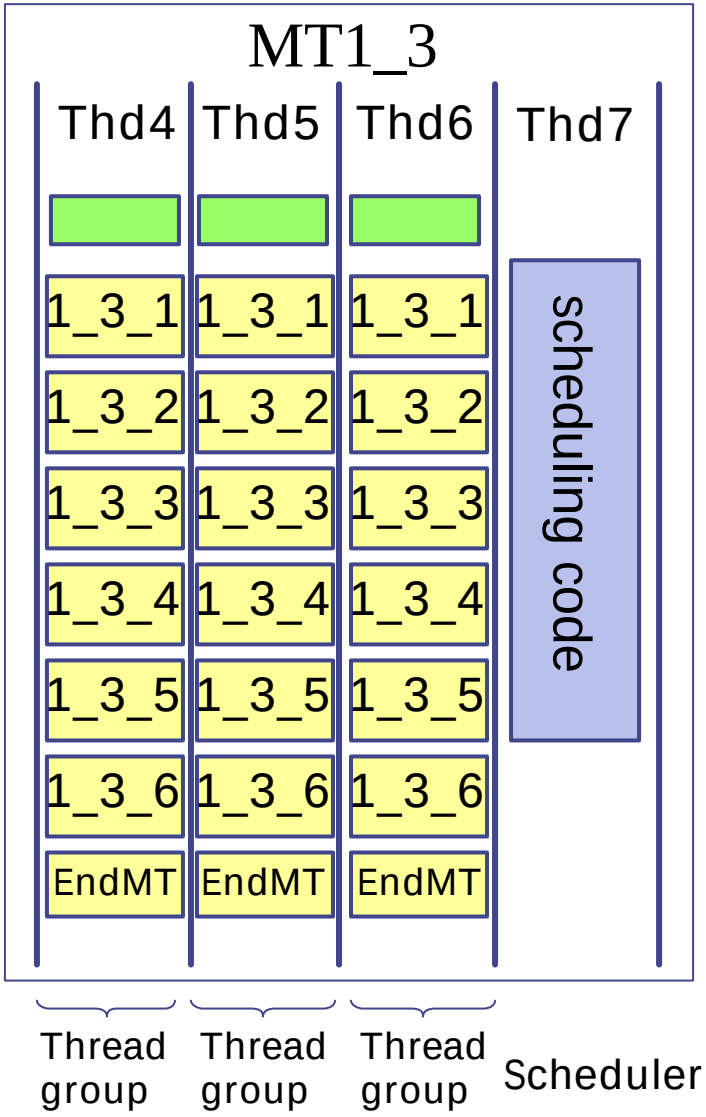


- MT1_3 is executed by 4 threads in group of the 1st layer
- Centralized dynamic scheduling is used in the 2nd layer
- 1 of 4 threads is used as scheduler
- 3 of 4 threads execute MTs

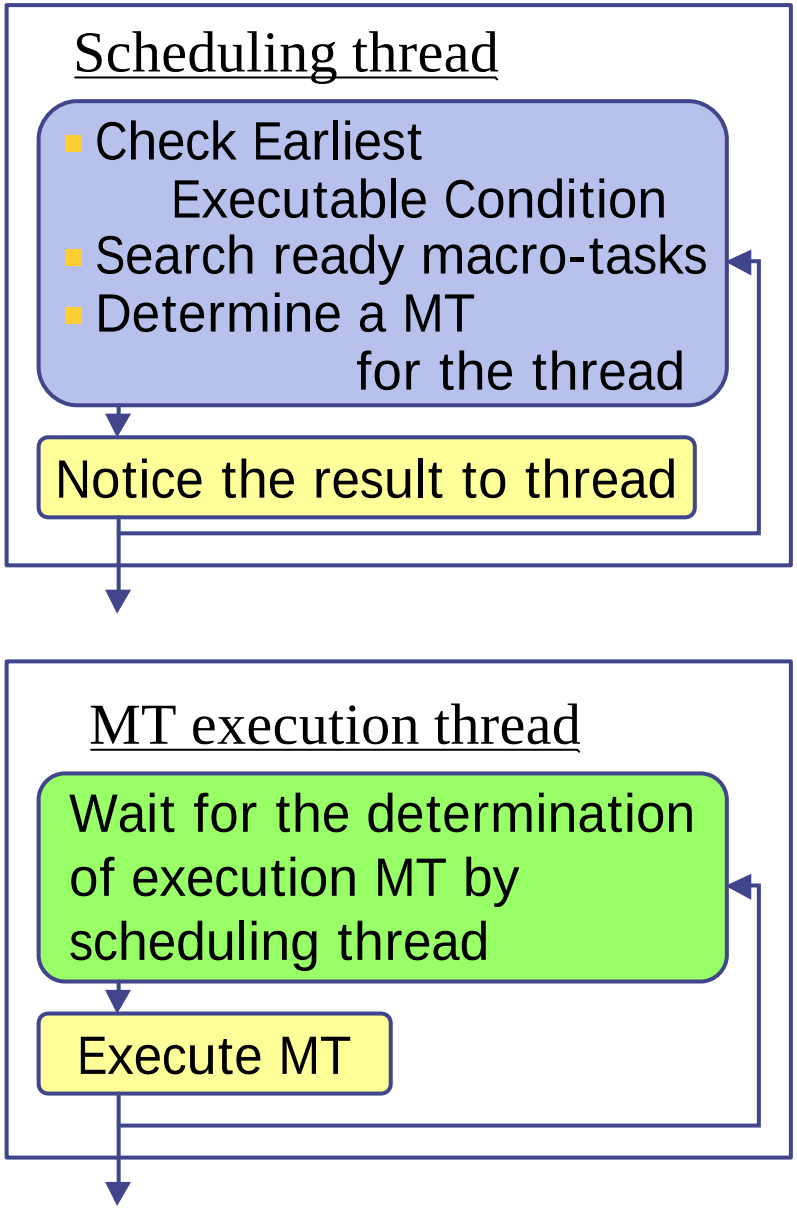
Centralized dynamic scheduling inside MT1_3



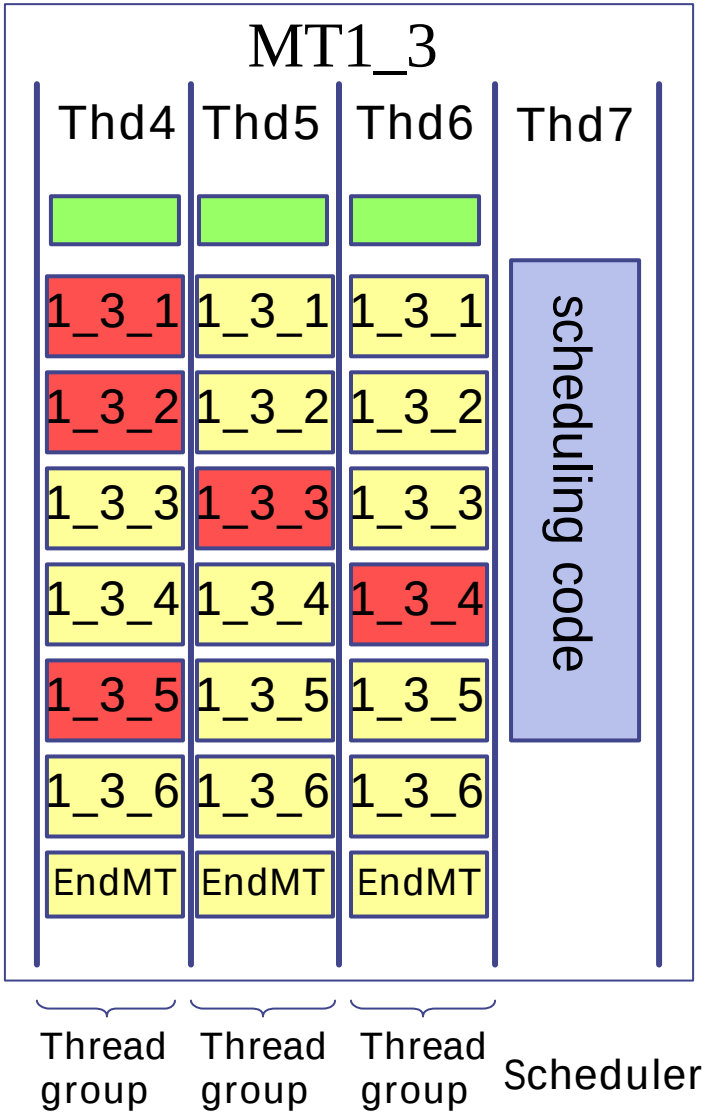
< Code image for MT1_3 >



Centralized dynamic scheduling inside MT1_3

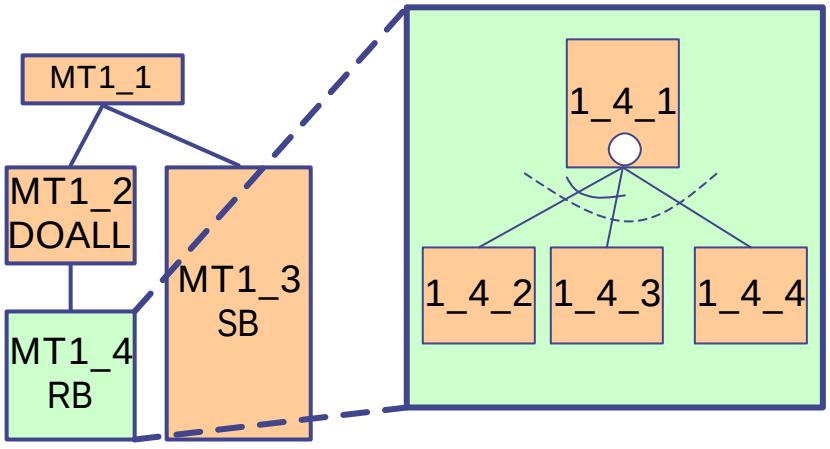


< Code image for MT1_3 >



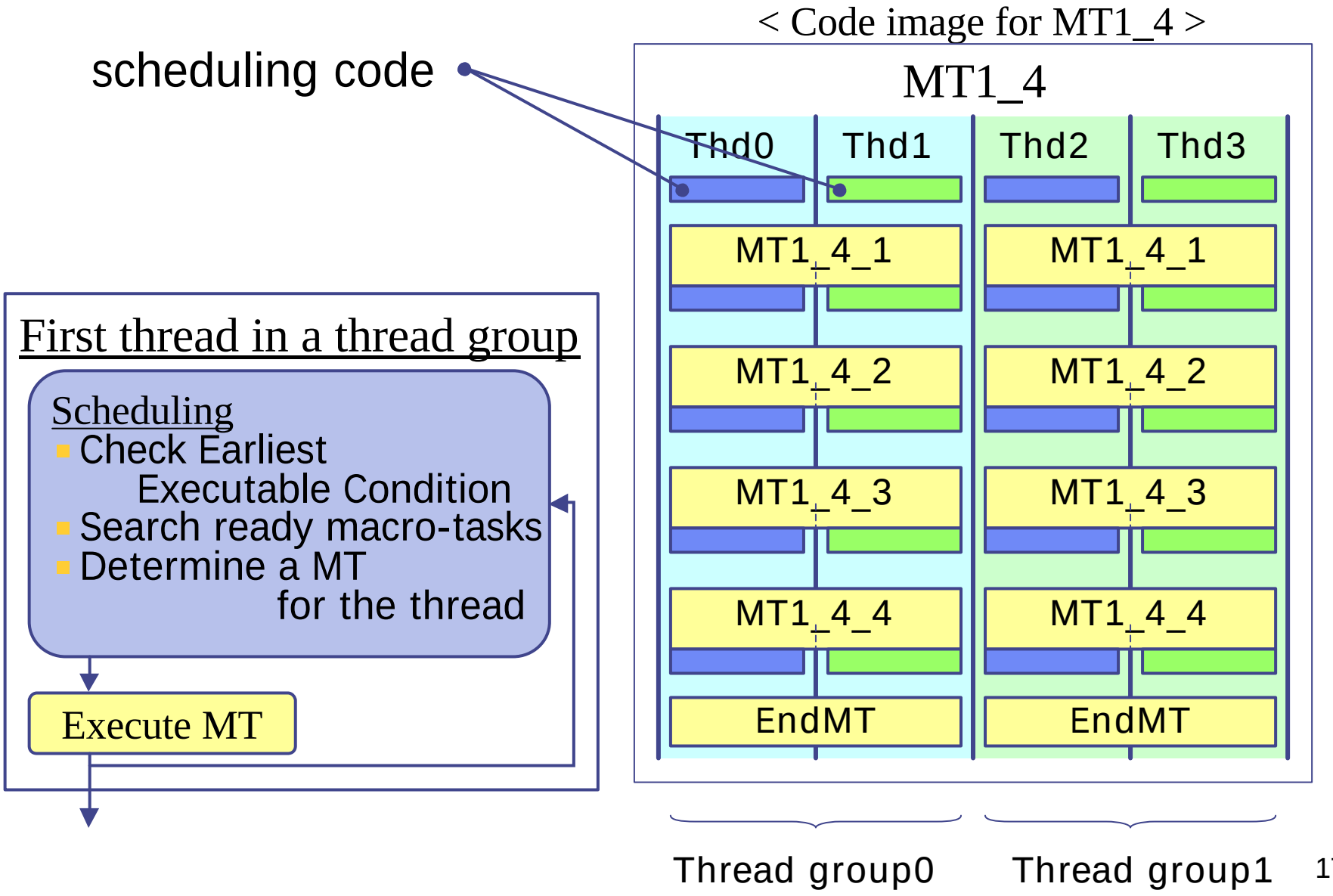
Distributed Dynamic Scheduling

- Compiler generates
  - Scheduling code and MTs code in all sections

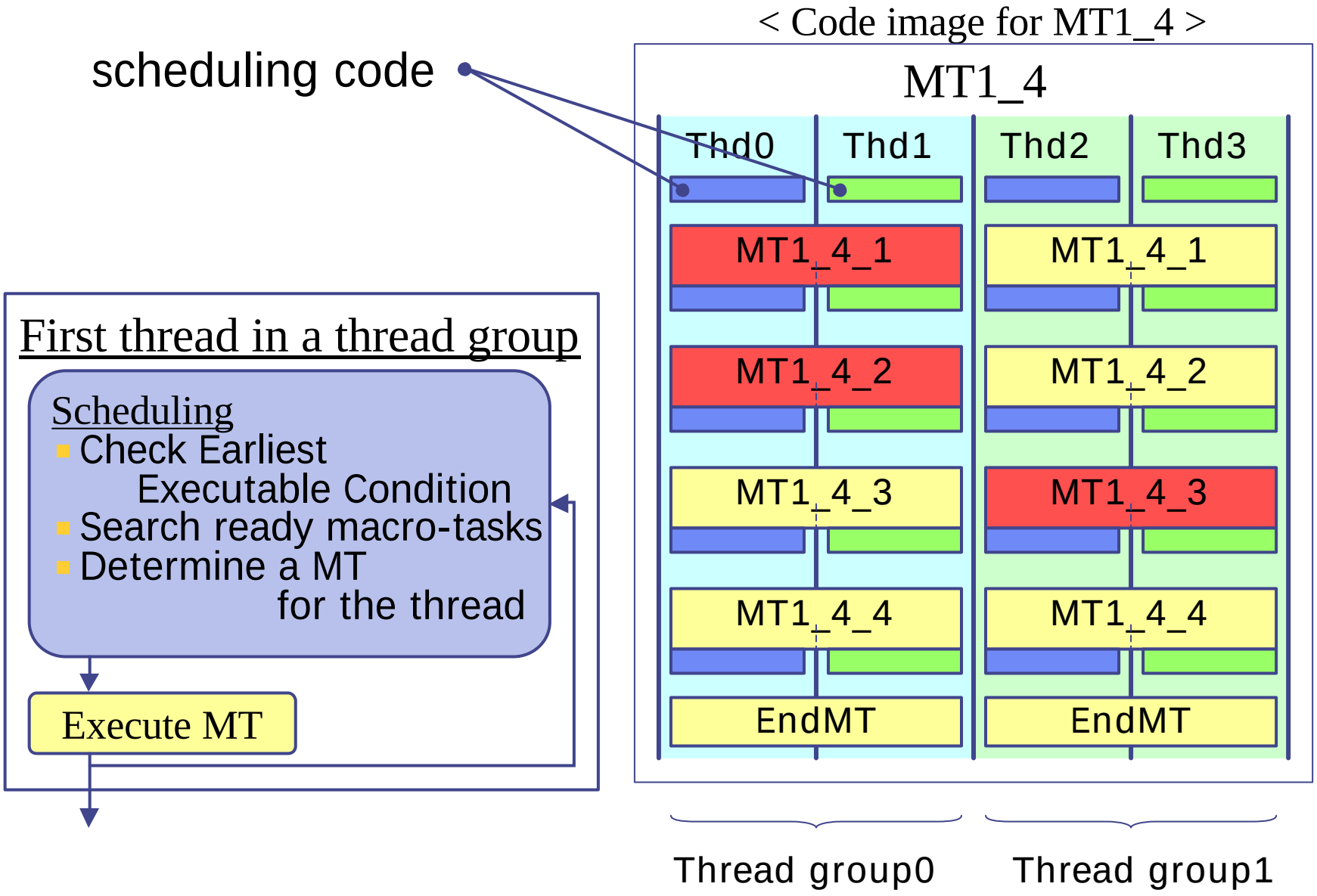


- MT1_4 is executed by 4 threads in group of the 1st layer
- Distributed dynamic scheduling is used in the 2nd layer
- MTs are scheduled to 2 thread groups each of which has 2 threads

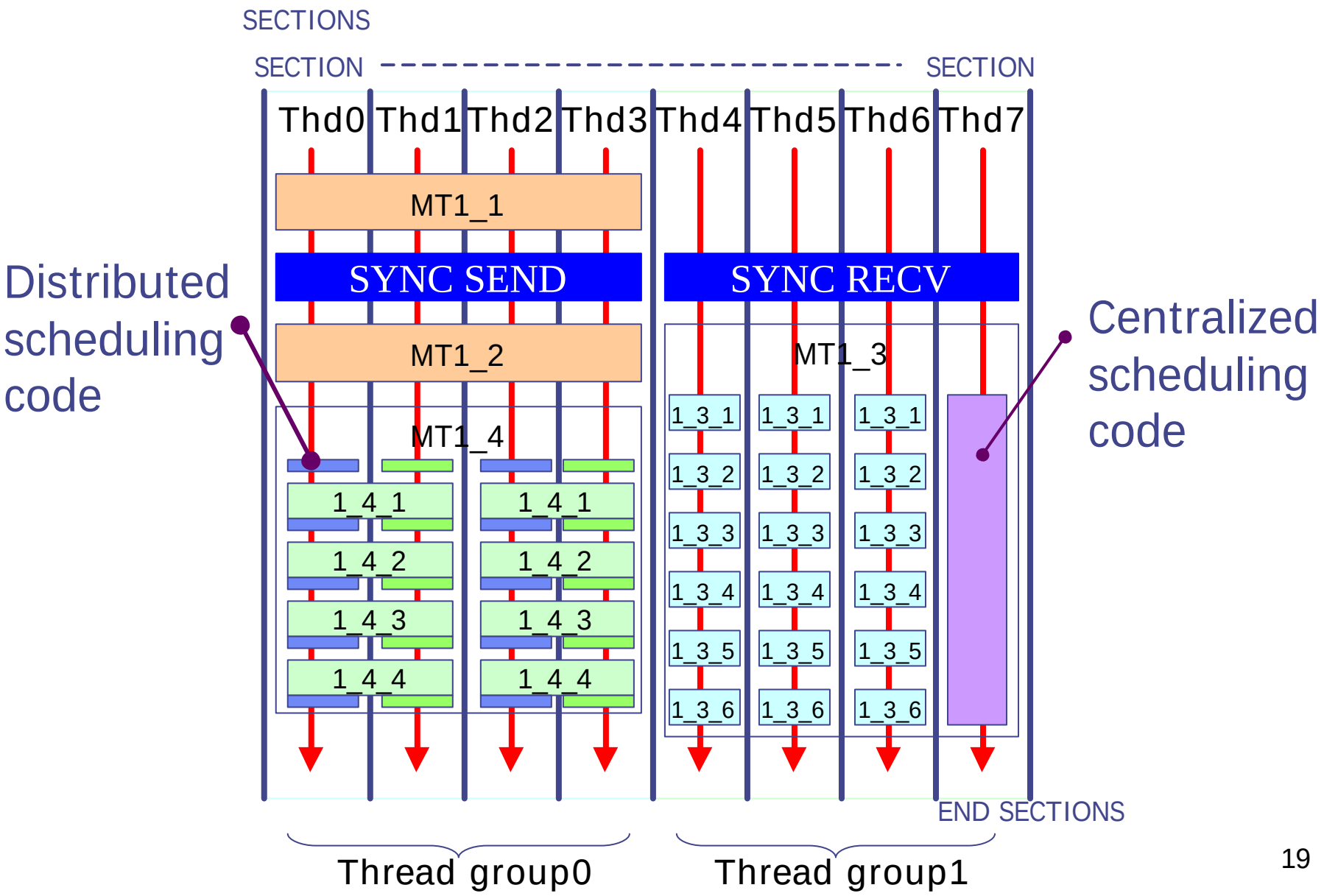
Distributed dynamic scheduling inside MT1_4



Distributed dynamic scheduling inside MT1_4



Hierarchical MT code image

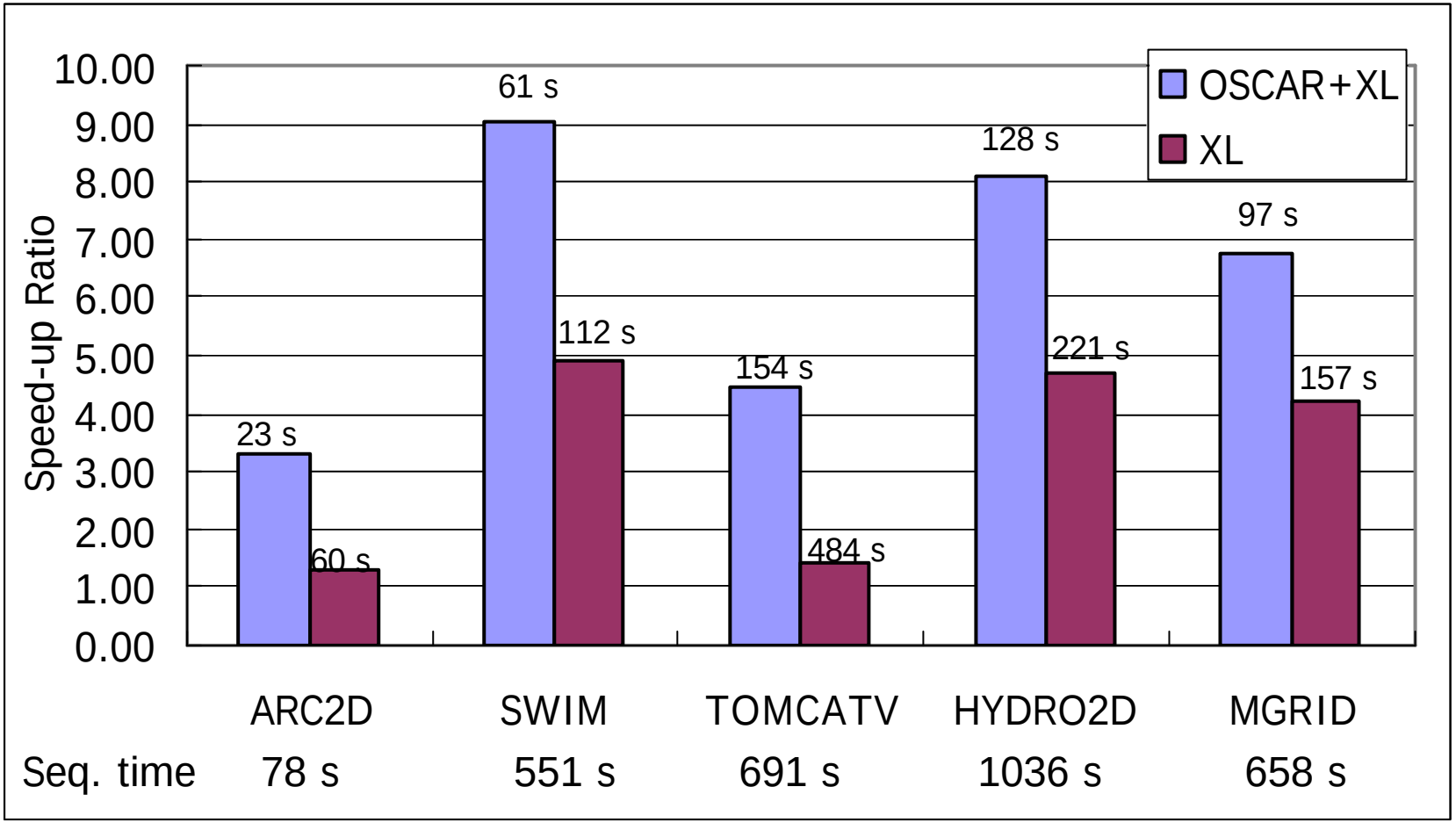


Performance evaluation

- Multiprocessor system(SMP)
  - IBM RS6000 SP 604e High Node
    - PowerPC 604e 200MHz x 8 processors
    - L1 Cache 32kbyte(Instruction/Data respectively)
    - L2 Cache 1Mbyte/processor
- Native parallelizing compiler
  - IBM XL Fortran Compiler Ver. 5.1
- Compiler options
  - sequential time
    - -O3 –qmaxmem=-1 –qhot
  - automatic parallelization
    - -O3 –qmaxmem=-1 –qsmp=auto –qhot
  - with OSCAR Compiler
    - -O2 –qmaxmem=-1 –qsmp=noauto –qhot

Performance on IBM RS6000

- Evaluation on 8 processors
- Speed-up Ratio = Sequential Time / Parallelization Time



Conclusions

- Proposal of an efficient implementation method by using OpenMP API for a hierarchical coarse grain task parallel processing on SMP
- In the evaluation on RS6000, OSCAR Fortran compiler could boost up the performance of XL Fortran compiler 1.5 to 3 times on 8 processors

- Future works
  - More evaluation on various SMP machines
  - Development of data localization techniques to use cache more efficiently