Parallelization of Automotive Engine Control Software On Embedded Multi-core Processor Using OSCAR Compiler

Yohei Kanehagi, Dan Umeda, Akihiro Hayashi,
Keiji Kimura and Hironori Kasahara
Graduate School of Fundamental Science and Engineering, Waseda University
3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan.
Tel: +81-3-5286-3018, Email: {ykane, umedan, ahayashi} @kasahara.cs.waseda.ac.jp,
kimura@apal.cs.waseda.ac.jp, kasahara@waseda.jp

(Keywords: multi-core processor, automobile, automatic parallelization, embedded system)

Ⅰ. Introduction

The next-generation automobiles are required to be more safe, comfortable and energy-efficient. These requirements can be realized by integrated control systems with enhanced electric control units, or real-time control system such as engine control and enhanced information system such as human and other cars recognition, navigations considering traffic conditions including the occasions of natural disasters. For example, sophisticating engine control algorithms requires performance enhancement of microprocessors to satisfy real-time constraints. Use of multi-core processors is a promising approach to realize the next-generation automobiles integrated control system. In terms of multi-core processors in the automotive control, the previous works include improvements of reliability by performing redundant calculation [1] and improvements of throughput by functional distribution [2] rather than improvement of response time, or performance by parallel processing. To the best of our knowledge, parallel processing of the automotive control software to reduce response time has not been succeeded on multi-core processors because the program consists of conditional branches and small basic blocks. On the other hand, this paper is the first paper has successfully parallelized the practical automotive engine control software using automatic multigrain parallelizing compiler, or the OSCAR Compiler has been developed by the authors for more than 25 years. The OSCAR compiler parallelizes automotive programs by utilizing coarse grain task parallelism with newly developed parallelism enhanced methods like the branch duplication instead of loop parallelism. Performance of the hand-written engine control programs provided by Toyota Motor Corp. on the RP-X having eight SH4A processor cores developed by Renesas, Hitachi, Tokyo Institute of technology and Waseda University is evaluated. The evaluation shows speedups of 1.54 times with 2 processor cores compared with the case of an ordinary sequential execution.

Ⅱ. OSCAR Automatic Parallelizing Compiler

The OSCAR Complier realizes an automatic parallelization of programs written in Parallelizable C [3], which is very close to MISRA C used in automobile industry for reliability and productivity or Fortran77. The OSCAR Compiler’s input is a sequential program and its output is an executable for a target multi-core. The OSCAR Compiler exploits multigrain parallelism including coarse-grain parallelism, loop level parallelism and fine-grain parallelism [4]. The OSCAR Compiler decomposes a program into coarse grain tasks, namely macro-tasks (MTs), such as basic block (BB), loop (RB), and function call or subroutine call (SB). Macro-tasks can be hierarchically defined inside each sequential loop or function. After generation of macro-tasks, data dependencies and control flow among macro-tasks are analyzed in each nested layer, and hierarchical macro-flow graphs (MFGs) representing control flow and data dependencies among macro-tasks are generated as shown in Fig. 1. Next, to exploit coarse grain task parallelism among macro-tasks, the Earliest Executable Condition analysis [5] is applied to each macro-flow graph. By this analysis, a macro-task graph (MTG) is generated for each macro-flow graph representing coarse grain task parallelism.

Ⅲ. Parallelization of Engine Control Software

Target engine control programs like crankshaft control are composed of conditional branches and small basic blocks without parallelizable loops. For this reason, current product compilers cannot parallelize this kind of control programs. Also, accelerator cannot apply to the application. The traditional loop parallelization technique widely used for multi-core processors can not apply the target engine control code since the program is composed of a series of conditional branches, assignment statements, and subroutine calls. Fig. 2 shows a MTG of the engine control program. As shown in Fig. 2, **sb1** and **sb12** can be executed in parallel with other tasks. However, because the execution times of **sb1** and **sb12** account for just 1% of the whole execution time, an inline expansion is applied to other tasks in order to exploit more parallelism over hierarchies or nested levels. In the proposed method, to improve coarse grain task parallelism the OSCAR Compiler applies selective inline expansion to function calls which have a coarse grain parallelism inside callee functions and also have large execution cost to exploit sufficient parallelism keeping the code size as small as possible. Fig. 3 shows a MTG of the restructured program by which an inline expansion was applied. As shown in Fig. 3, more coarse grain parallelism is exploited than that of Fig. 2. However, the execution time of **block36** accounts for about 70% of the whole execution time and **block36** has coarse grain parallelism inside. Fig. 4 shows a simplified image of the **block36**. There are **sb2**, **sb3** and **sb4** inside a then-clause of the if-statement of the left side of Fig. 4. These subroutine calls are assigned onto same processor core though. These coarse grain parallelisms among them since the if-statement and these subroutine calls are packed into the same MT to minimize scheduling overhead. However, by duplicating if-statement as shown in the right side of Fig. 4, and by packing each if-statement and each subroutine call into a MT, parallelism among those subroutine calls can be efficiently exploited. The duplication applies as a condition that a conditional expression does not change in **sb2**, **sb3** and **sb4**. Fig. 5 shows a MTG of the program after the inline expansion and the conditional branch duplication are applied. The graph's critical path accounts for about 60% of the whole execution time. Selective inline expansion and conditional branch duplication allow us to exploit remarkable coarse grain parallelism.

Ⅳ. Performance Evaluation of Parallelized Engine Control Code on RP-X

This paper uses the embedded multi-core processor RP-X [5] to evaluate the performance of parallelized automotive engine control code. The RP-X processor has eight 648MHz SH-4A general-purpose processor cores, four 324MHz FEGA accelerator cores, two matrix processor "MX-2" and video processing unit "VPU5", as shown in Fig. 6. RP-X can change the clock frequency of processor cores, such as 648MHz, 324MHz, 162MHz and 81MHz. In this paper, we set the clock frequency of processor cores as 81MHz in order to bring close to it actually used automotive control unit. Fig. 7 shows the result of the evaluation of the automotive engine control software parallelized by the OSCAR Compiler. Our proposed method attains speedups of 1.54 times with 2 processor cores compared with the sequential execution.

Ⅴ. Conclusions

This paper has described parallelization of the automotive engine control software by use the OSCAR Compiler. The original hand-written sequential engine control code is restructured with the conditional branch duplication and selective inline expansion in order to exploit coarse grain task parallelism. The parallelized program is evaluated on the embedded multi-core RP-X. As a result, we use 2 cores on RP-X, and attain 1.54 times speed-up compared with sequential execution. The result shows that the OSCAR Compiler can exploit parallelism from the automotive engine control software, which is composed of a series of conditional branches, assignment statements and subroutine calls. Currently, the authors are evaluating the performance of the proposed method using engine control processors.

Ⅵ. Acknowledgements

This work was supported by Toyota Motor Corporation. I would like to express appreciation to Mr. Mitsuo Sawada from Toyota Motor Corporation.

References

[1] Kyungil Seo, Taeyoung Chung, Hyundong Heo, Kyongsu Yi, and Naehyuck Chang, "An Investigation into Multi-Core Architectures to Improve a Processing Performance of the Unified Chassis Control Algorithms," SAE Int.J.Passeng.Cars-Electron.Electr.Syst., pp. 53-62, 2010.

[2] Dinesh Padole, and Preeti Bajaj, "FUZZY ARBITER BASED MULTI CORE SYSTEM-ON-CHIP INTEGRATED CONTROLLER FOR AUTOMOTIVE SYSTEMS: A DESIGN APPROACH," CCECE, pp. 1937-1940, 2008.

[3] M. Mase, Y. Onozaki, K. Kimura, and H. Kasahara, "Parallelizable c and its performance on low power high performance multicore processors," In Proc. of 15th Workshop on Compilers for Parallel Computing, Jul. 2010.

[4] H. Kasahara, H. Honda, A. Mogi, A. Ogura, K. Fujiwara, and S. Narita, "A multi-grain parallelizing compilation scheme for OSCAR(Optimally scheduled advanced multiprocessor," In Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing, pp. 283-297, August 1991.

[5] H. Honda, M. Iwata, and H. Kasahara, "Coarse grain parallelism detection scheme of a Fortran program," Trans. of IEICE, Vol.J73-D-1, No.12, pp. 951-960, Dec. 1990.

[6] Y. Yuyama, M. Ito, Y. Kiyoshige, Y. Nitta, S. Matsui, O. Nishii, A. Hasegawa, M. Ishikawa, T. Yamada, J. Miyakoshi, K. Terada, T. Nojiri, M. Satoh, H. Mizuno, K. Uchiyama, Y. Wada, K. Kimura, H. Kasahara, and H. Maejima, “A 45nm 37.3gops/w heterogeneous multi-core soc,” IEEE International Solid-State Circuits Conference, ISSCC, pp. 100-101, Feb. 2010.

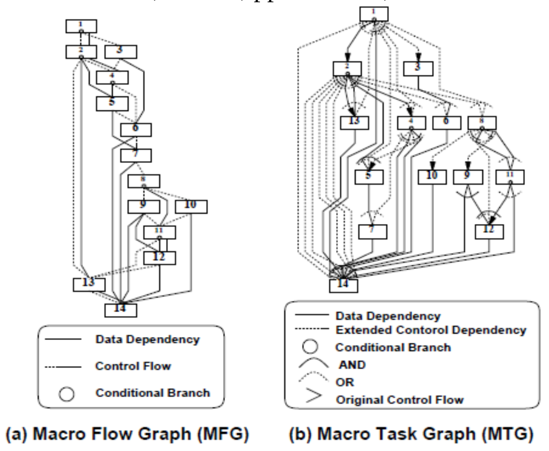
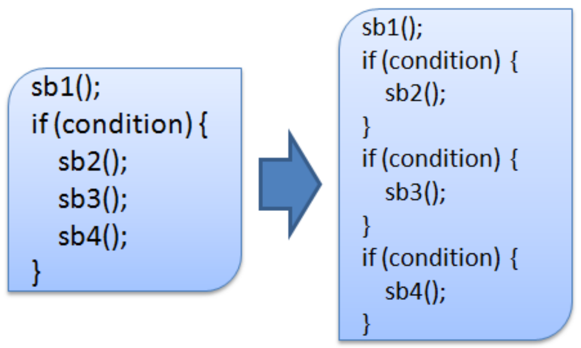


Fig. 1. Macro-flow graph and macro-task graph



```
sb1();
if (condition) {
    sb2();
    sb3();
    sb4();
}
```

```
sb1();
if (condition) {
    sb2();
}
if (condition) {
    sb3();
}
if (condition) {
    sb4();
}
```

Fig. 4. Conditional branch duplication

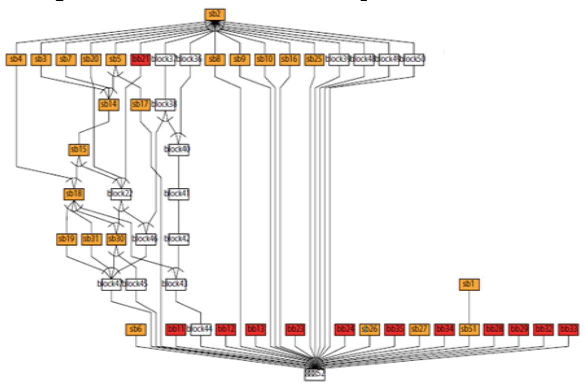


Fig. 5. Macro-task graph of the automotive engine control software after inline expansion and conditional branch duplication

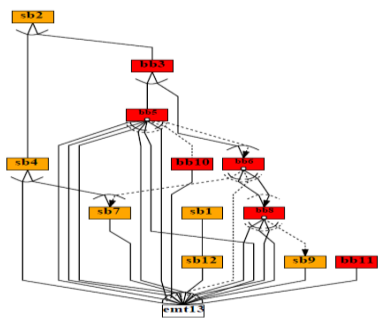


Fig. 2. Macro-task graph of the automotive engine control software

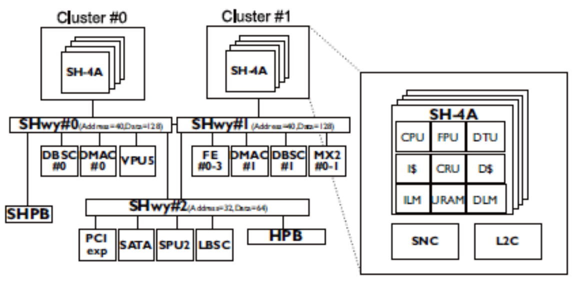


Fig. 6. The embedded multi-core processor RP-X

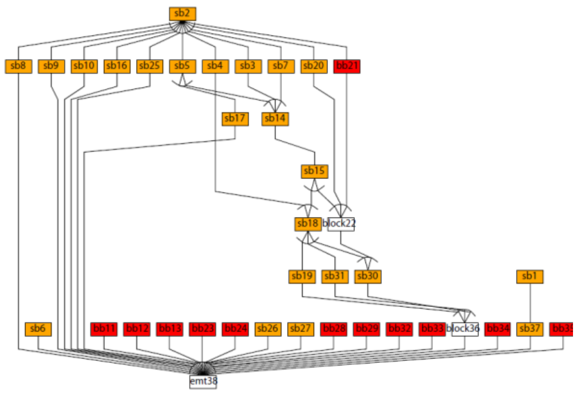


Fig. 3. Macro-task graph of the automotive engine control software after inline expansion

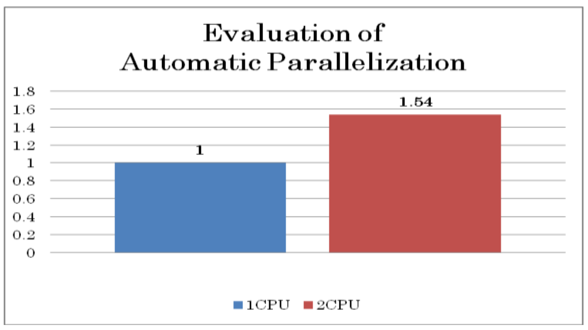


Fig. 7. The evaluation of automatic parallelization of the automotive engine control software on RP-X

Parallelization of Automotive Engine Control Software On Embedded Multi-core Processor Using OSCAR Compiler

Yohei Kanehagi, Dan Umeda,
Akihiro Hayashi, Keiji Kimura
and Hironori Kasahara

Faster Processing of Engine Control Programs for Future Automotives

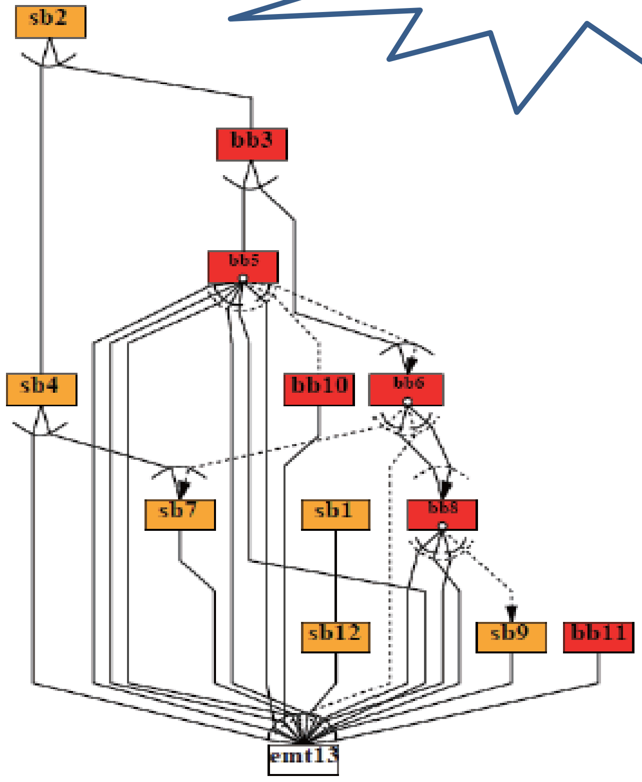
- Safety, comfort and energy-efficiency are more necessary for future automobile
  - Faster processing of Engine Control Programs is required
- **New sophisticated control function will be used**

- Parallel processing of engine control programs on multi-core processors is required
  - Performance with single core processor is limited by power and heat

Motivation : Parallelizing Crankshaft Program

```
void main()
{
   sb1();
   sb2();
   bb3;
   sb4();
   if (cond1)
   {
      bb6;
      if (cond2)
      {
         sb7();
      }
      if (cond3)
      {
         sb9();
      }
   }
   sb12();
}
```

No loops



Parallelizing Automotive Control Programs on Multi-core processors

- Automotive crankshaft program
  - Two Challenges
    1. Real-time constraints
       - minimizing overhead is important
    2. It is hard to parallelize since there are many conditional branches, assign statements, but few loops inside them.
       - Current product compilers such as Intel and IBM can not parallelize this program
       - Current accelerators as GPU and FEGA is applicable to this program
  - No one has succeeded to parallelize this program

We explore the possibility of utilization of multi-core processor for next-generation automobiles

- As the first step, this work uses 2 cores
  - Since the next-generation automobiles are going to use 2 cores

OSCAR Compiler for Crankshaft Control Programs

OSCAR Compiler

- **Effective Automatic Parallelization**
- **Utilization of coarse grain parallelism in this case**
  - Our approach improves coarse grain parallelism by aggregating fine grain statements and duplicating conditional branches

Related Works

- Few works are working on using a multi-core processor for automotive software
  1. Improving Reliability of Control Software (SAE'10)
     - by performing redundant calculation with a multi-core processor
  2. Improving Throughput of Control Software (ECE'08, SIES'11)
     - by performing **functional distribution** with a multi-core processor

Can not improve response time

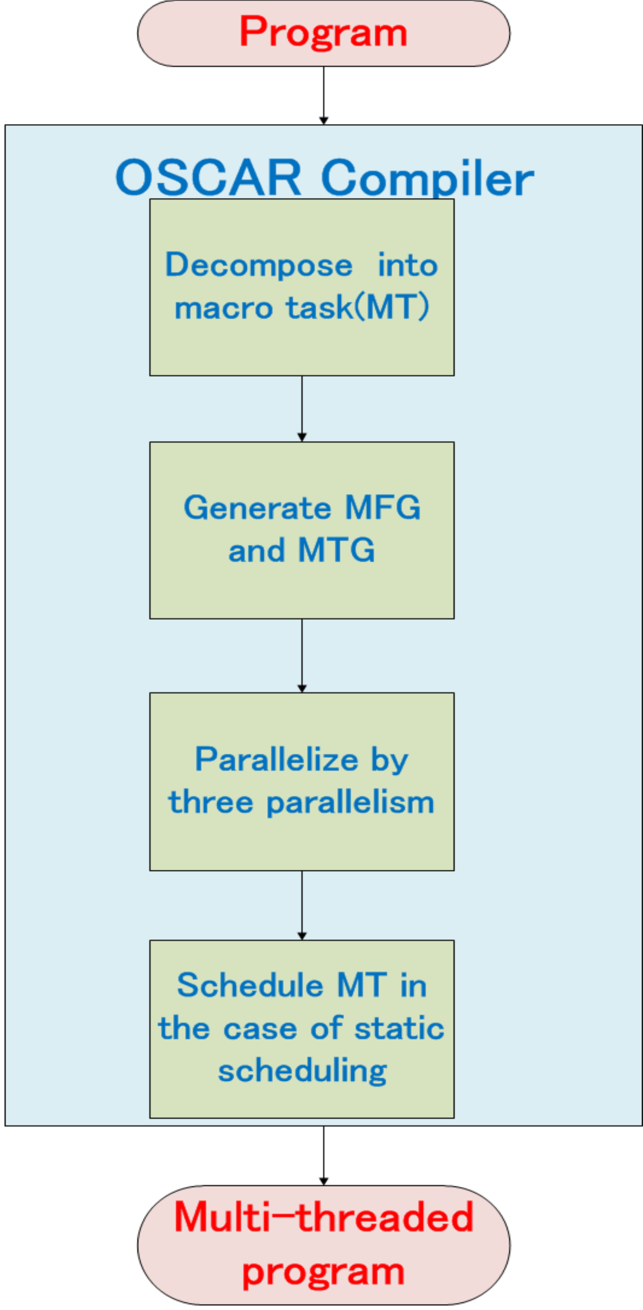
Our approach is parallelizing with OSCAR compiler

OSCAR allows us to perform **workload distribution automatically**

Can improve response time

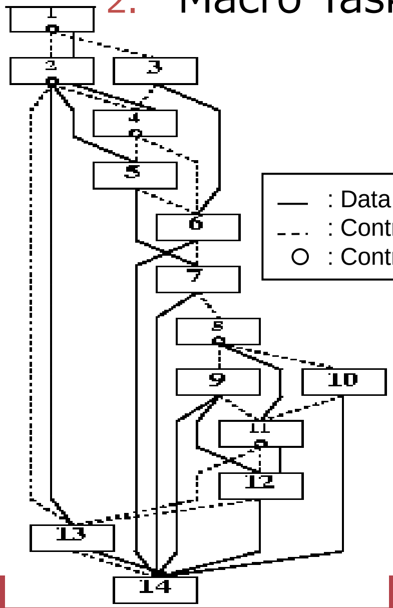
OSCAR Parallel Compiler

- Generate a Multi-threaded program from a sequential C or Fortran
- Multi-grain parallelization
  1. **Coarse grain parallelization**
  2. Loop level parallelization
  3. Fine grain parallelization
- Task scheduling
  - **Static task scheduling**
    - Schedule tasks to CPUs at compiler time
  - Dynamic task scheduling
    - Schedule tasks to CPUs at runtime
    - OSCAR generates dynamic scheduling routines


Program
OSCAR Compiler
Decompose into macro task(MT)
Generate MFG and MTG
Parallelize by three parallelism
Schedule MT in the case of static scheduling
Multi-threaded program

OSCAR Compiler Exploit Parallelism

- Decomposes a program into coarse grain tasks, or macro tasks(MTs)
  1. BB (Basic Block)
  2. RB (Repetition Block, or loop)
  3. SB (Subroutine Block, or function)
- Generates MTG from MFG
  1. Macro Flow Graph (MFG): control-flow and data dependency
  2. Macro Task Graph (MTG): coarse grain task parallelism



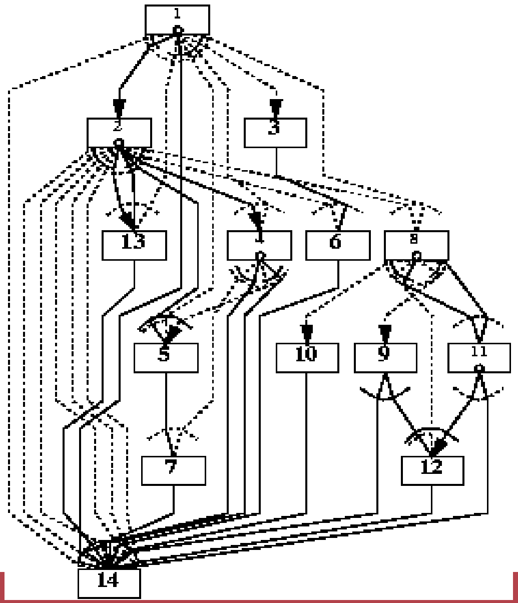
— : Data Dependency
- - - : Control flow
○ : Control Branch

Earliest Executable Condition

(Condition for determination of MT Execution)
AND
(Condition for Data access)

Ex. Earliest Executable Condition of MT6

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



Macro-Flow Graph

Macro-Task Graph

Proposed Parallel Processing Method for Engine Control Programs

- Decreasing overhead of task scheduling and improvement of coarse grain parallelism are necessary

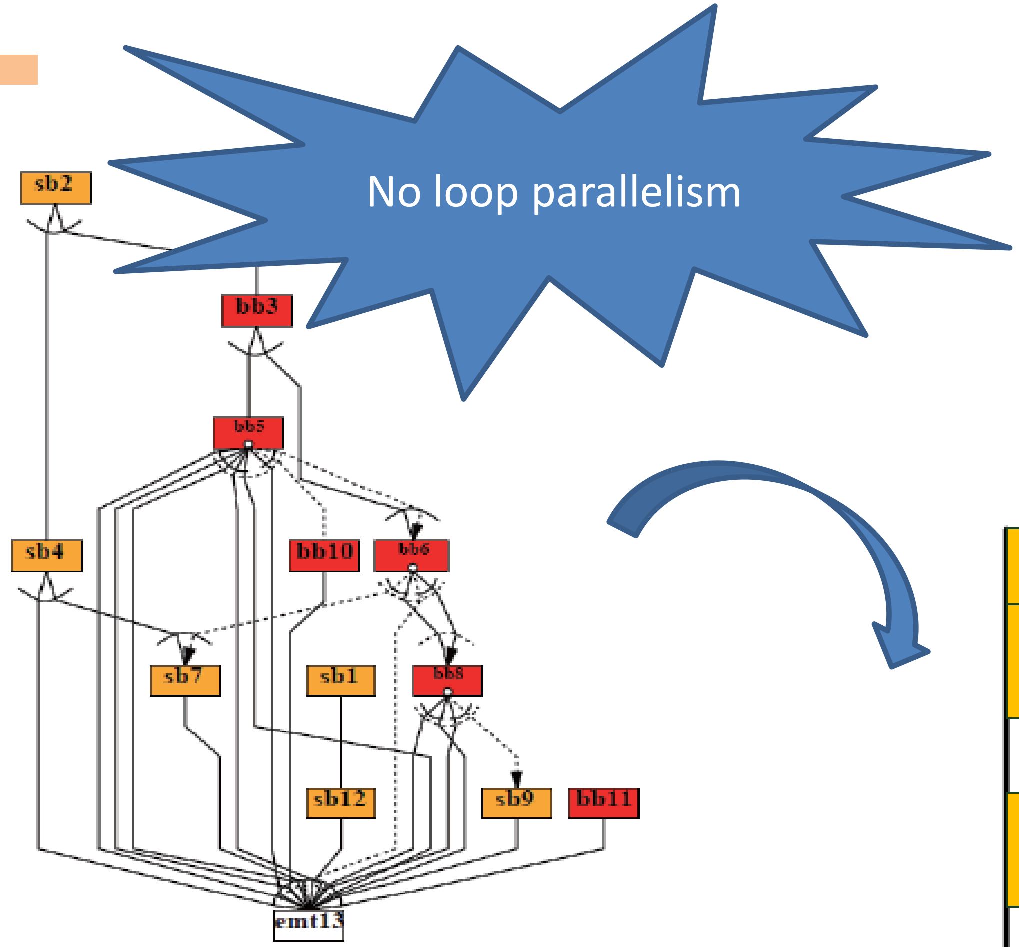
1. Coarse grain parallelization
   - **To detect parallelism among conditional branches and assign statements from the programs and not loops**

2. Static task scheduling
   - **To allow us to guarantee real-time constraints and reduce run-time overhead**

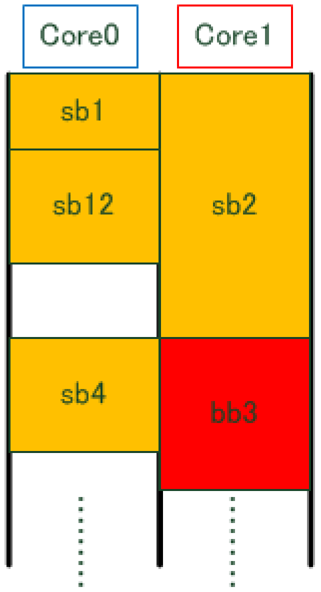
3. Restructuring of inline expansion and duplicating if-statements
   - **To extract more parallelism among coarse grain tasks over conditional branches**

1 Coarse Grain Parallelization

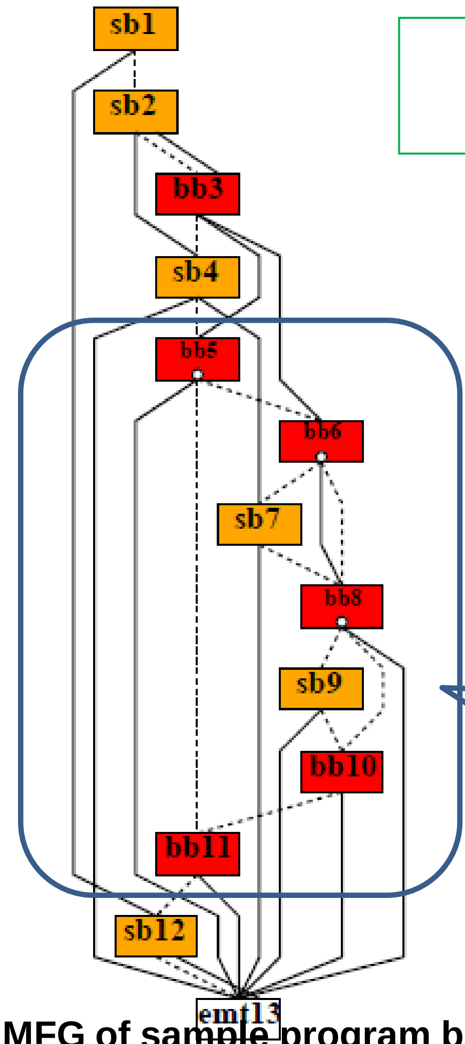
```
void main()
{
   sb1();
   sb2();
   bb3;
   sb4();
   if (cond1)
   {
      bb6;
      if (cond2)
      {
         sb7();
      }
      if (cond3)
      {
         sb9();
      }
   }
   sb12();
}
```



MTG of crankshaft programs



2 Static Scheduling



MFG of sample program before task fusion

To allow us to guarantee real-time constraints and reduce run-time overhead

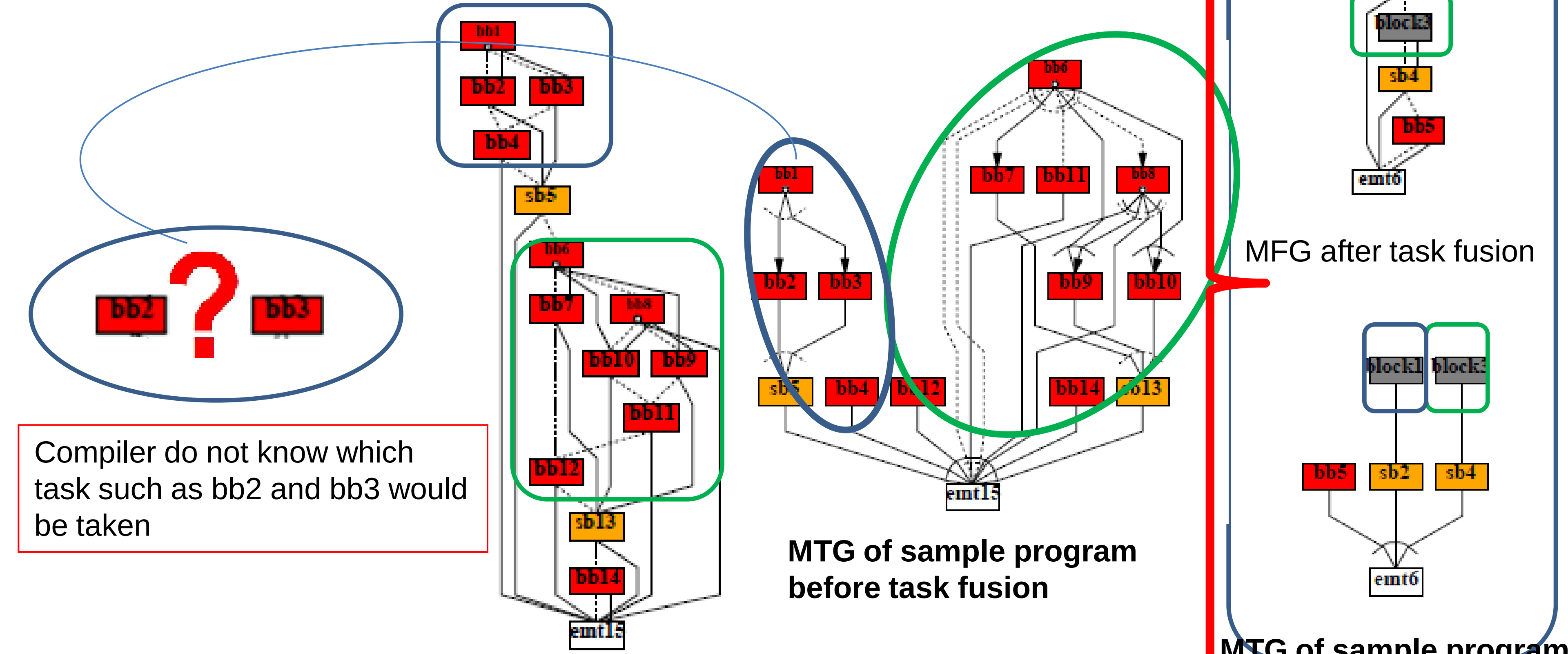
However

Unable to assign tasks such as sb7 statically due to conditional branches.
The compiler cannot see if the branch is taken or not statically.

Fusing tasks with hiding all control-flow edges in MTG to avoid dynamic scheduling.

Task Fusion for Static Scheduling

Fuse branches and succeed task



Compiler do not know which task such as bb2 and bb3 would be taken

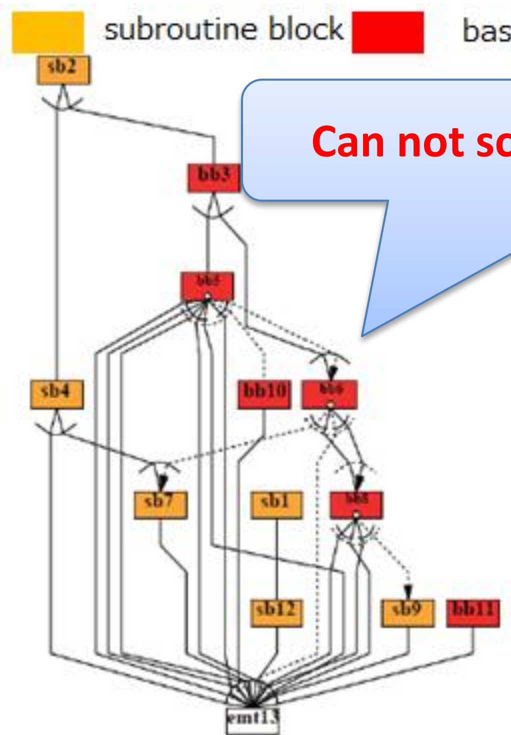
MTG of sample program before task fusion

MFG after task fusion

MTG of sample program after task fusion

MFG of sample program before task fusion

MTG of Crankshaft Program Before and After Task Fusion for Static Scheduling

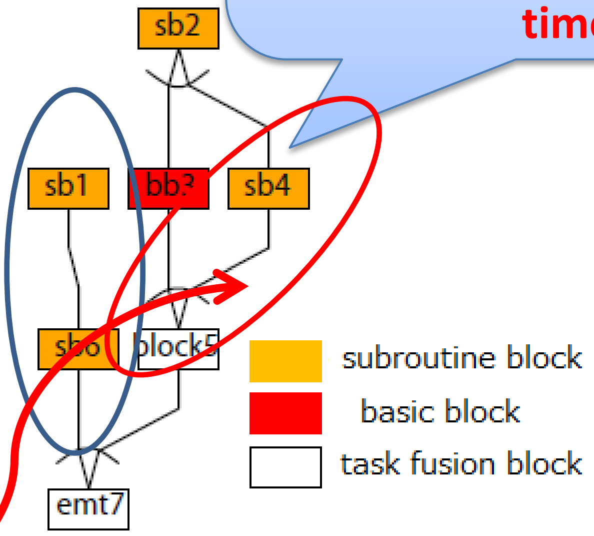


Can not schedule MT at compile time

MTG of crankshaft program before task fusion

sb1 and sb6 account for just 1% of whole execution time.

sb4 and block5 account for over 90% of whole execution time.



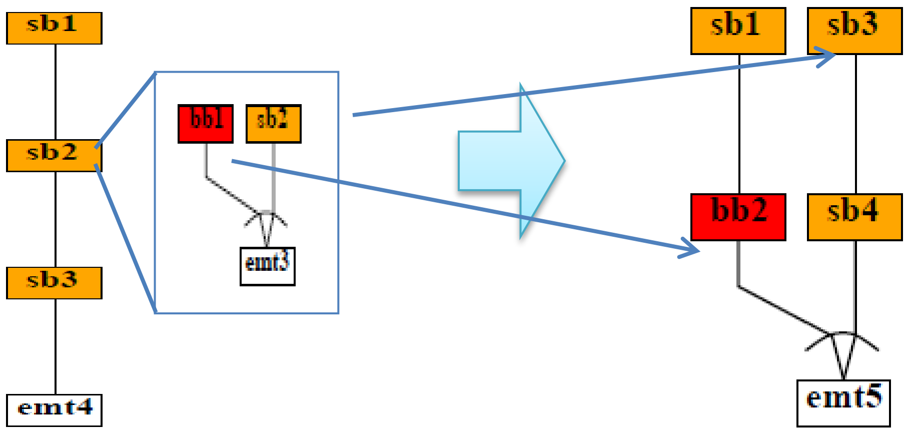
MTG of crankshaft program after task fusion

There are not enough parallelism

3.1 Restructuring : Inline Expansion

- Inline expansion is effective
  - To increase coarse grain parallelism
- Expands functions having inner parallelism

Improve coarse grain parallelism



bb1 inside sb2 is data-dependent on sb1

sb3 is data-dependent on sb2 inside sb2

MTG of sample program before inline expansion

MTG of sample program after inline expansion

3.2 Restructuring: Duplicating If-statements

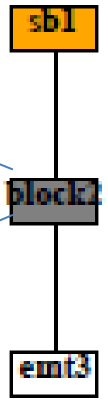
- Duplicating if-statements is effective
  - To increase coarse grain parallelism
- Duplicate fused tasks having inner parallelism

Improve coarse grain parallelism

Sb1 depends sb5

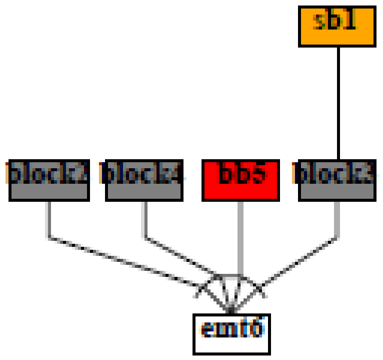
```
func1();
if (condition) {
    func2();
    func3();
    func4();
}
```

No dependemce



Copying if-conditions each functions

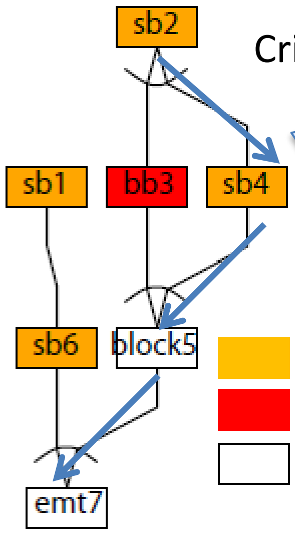
```
func1();
if (condition) {
    func2();
}
if (condition) {
    func3();
}
if (condition) {
    func4();
}
```



MTG of sample program before duplicating if-statements

MTG of sample program after duplicating if-statements

MTG of Crankshaft Program Before and After Inline Expansion and Duplicating If-statements

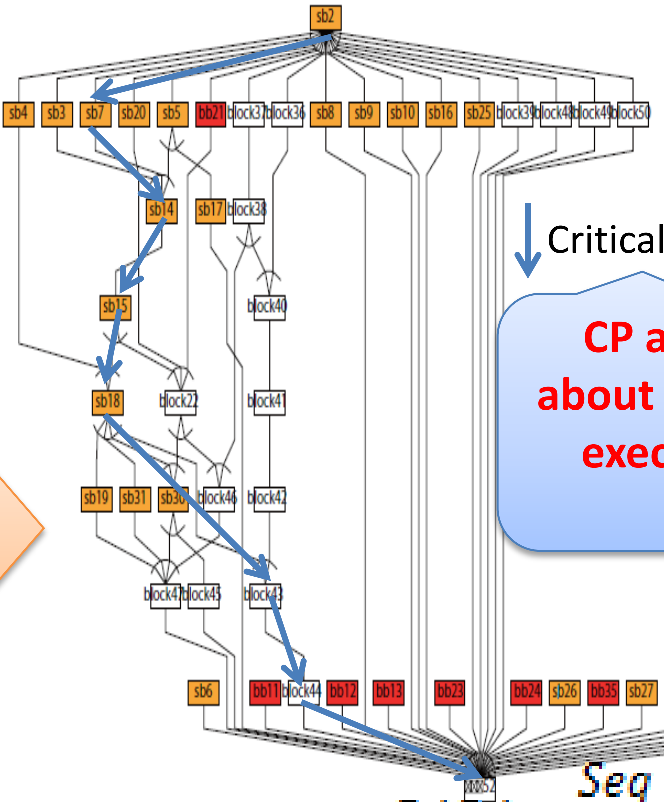


Critical Path(CP)

CP accounts for over 90% of whole execution time.

🟧 subroutine block
🟥 basic block
⬜ task fusion block

$$PARA = \frac{Seq\ cost}{CP\ cost} \cong 1.1$$



🟧 subroutine block
🟥 basic block
⬜ task fusion block

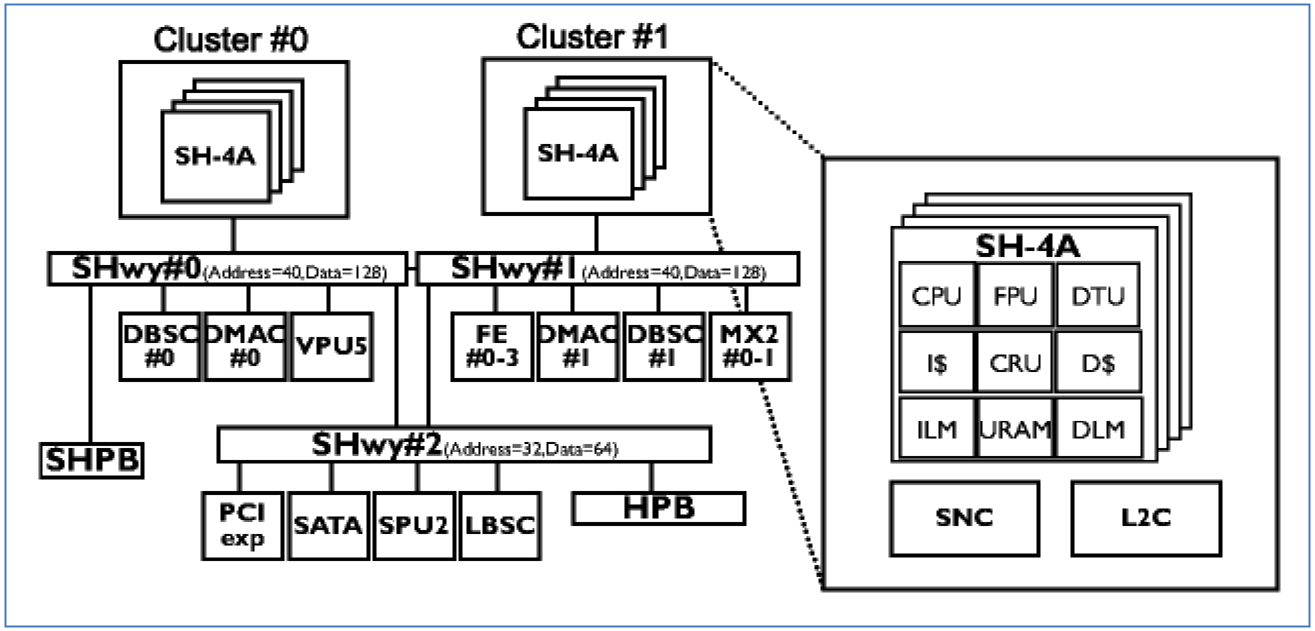
Critical Path(CP)

CP accounts for about 60% of whole execution time.

$$PARA = \frac{Seq\ cost}{CP\ cost} \cong 1.6$$

Successfully increased coarse grain parallelism

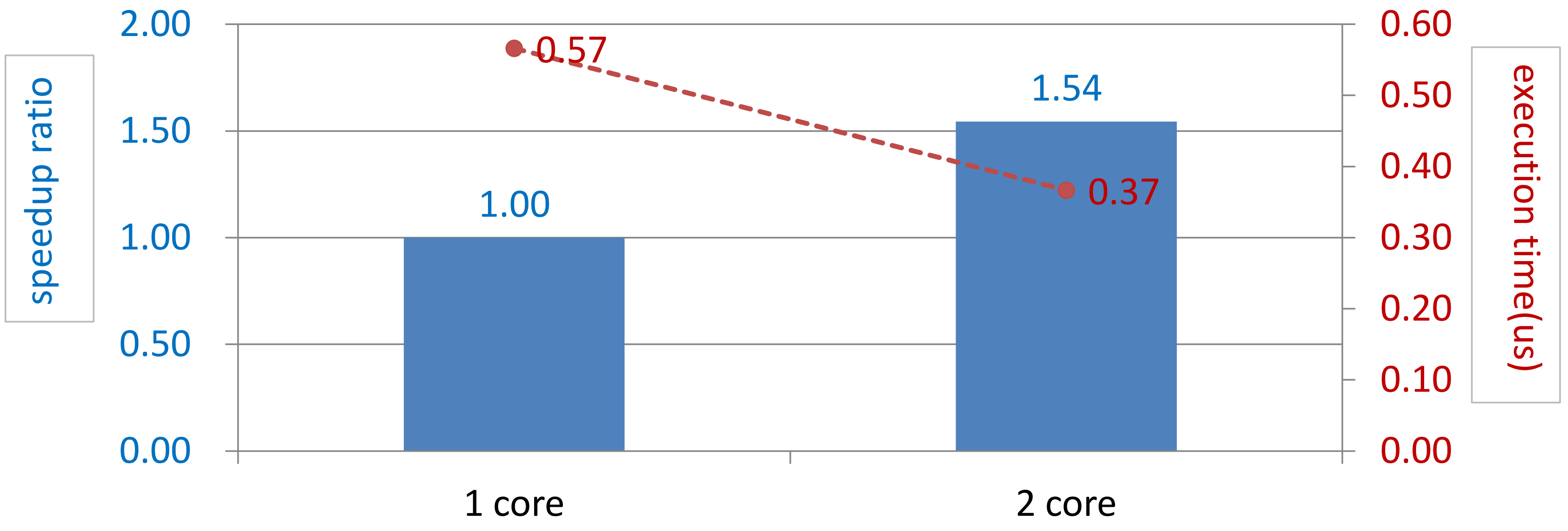
Embedded Multi-core Processor RPX



- SH-4A 648MHz * 8
- FE-GA 324MHz * 4
- Instruction cache: 32KB/core
- Data cache: 32KB/core
- ILM, DLM: 16KB/core
- URAM: 64KB/core

- Change the clock frequency of processors core
  - 648MHz, 324MHz, 162MHz, 81MHz
  - Set 81MHz in order to bring close to it actually used automotive control unit

Evaluation of Crankshaft Program with Multi-core Processors



- Attain 1.54 times speedup on RPX running at 81MHz
  - Theoretical speedup ratio is around 1.6
  - due to execution and thread overheads

Conclusion

- Parallelization of automotive control program with OSCAR compiler
  - Current compilers and accelerators are not applicable
    - Program has no loop, but has conditional branches and assign statements
  - Utilization of coarse grain parallelism and static scheduling in OSCAR compiler
  - The first technique which exploits coarse grain parallelism of automotive control programs
    - Inline expansion
    - Duplicating if-statements
  - We attained 1.54x speedup on embedded multi-core RPX running at 81MHz for the first time

Thank you for your attention!