Parallelization and Power Reduction of Embedded Real-time Applications by OSCAR Compiler on ARM and Intel Multicores

Hironori Kasahara
Professor, Dept. of Computer Science & Engineering
Director, Advanced Multicore Processor Research Institute
Waseda University, Tokyo, Japan
IEEE Computer Society Multicore STC Chair
URL: http://www.kasahara.cs.waseda.ac.jp/

OSCAR Parallelizing Compiler

To improve effective performance, cost-performance and software productivity and reduce power

Multigrain Parallelization

coarse-grain parallelism among loops and subroutines, near fine grain parallelism among statements in addition to loop parallelism

Data Localization

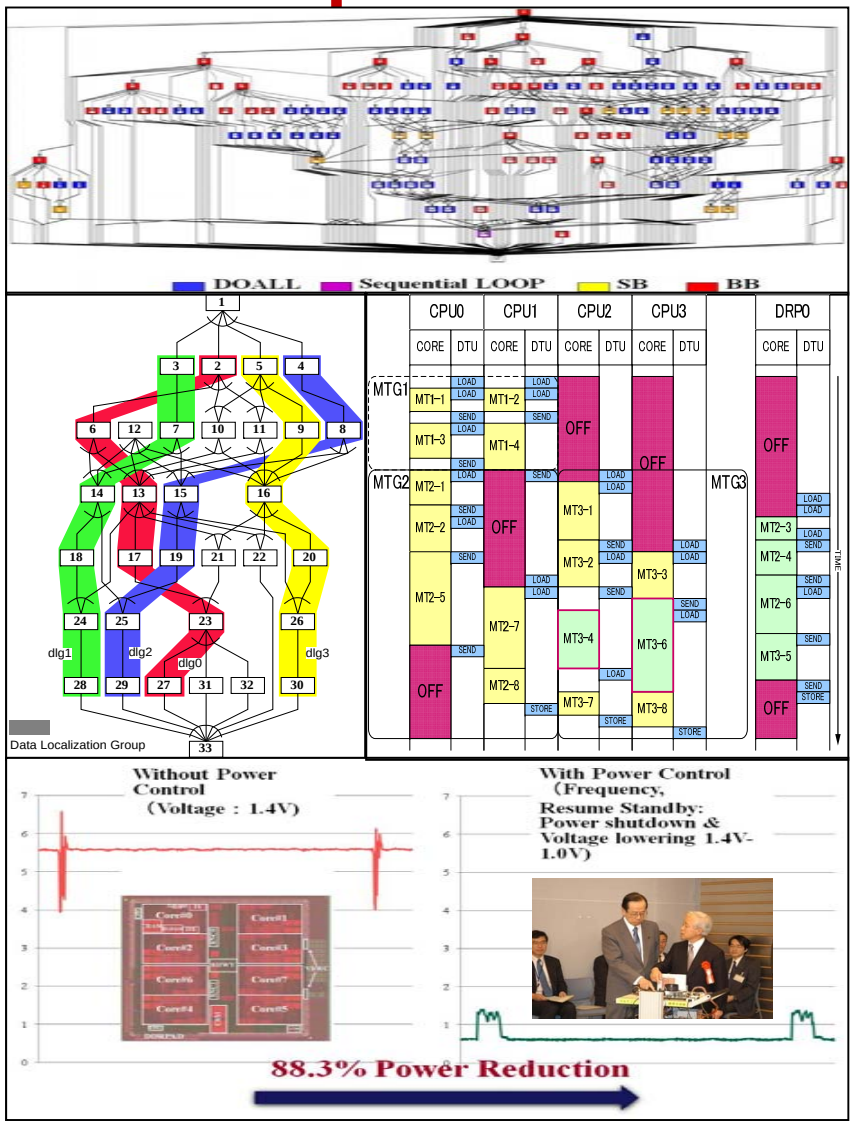
Automatic data management for distributed shared memory, cache and local memory

Data Transfer Overlapping

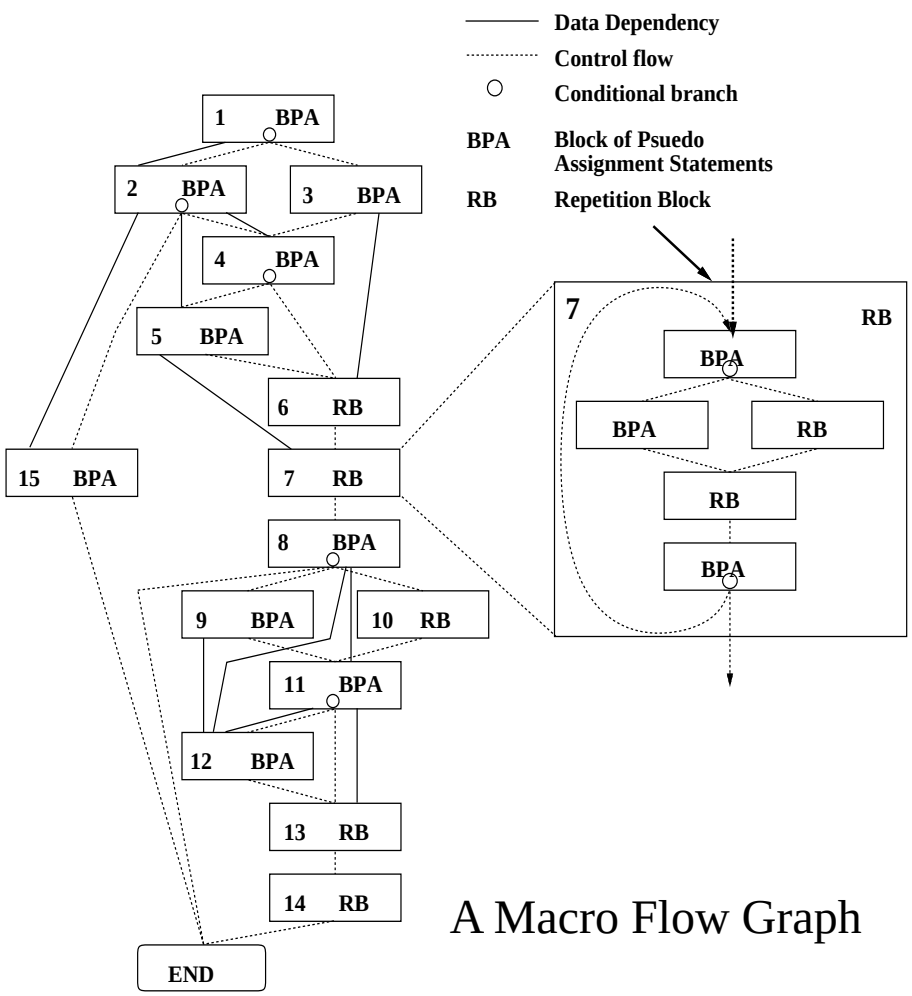
Data transfer overlapping using Data Transfer Controllers (DMAs)

Power Reduction

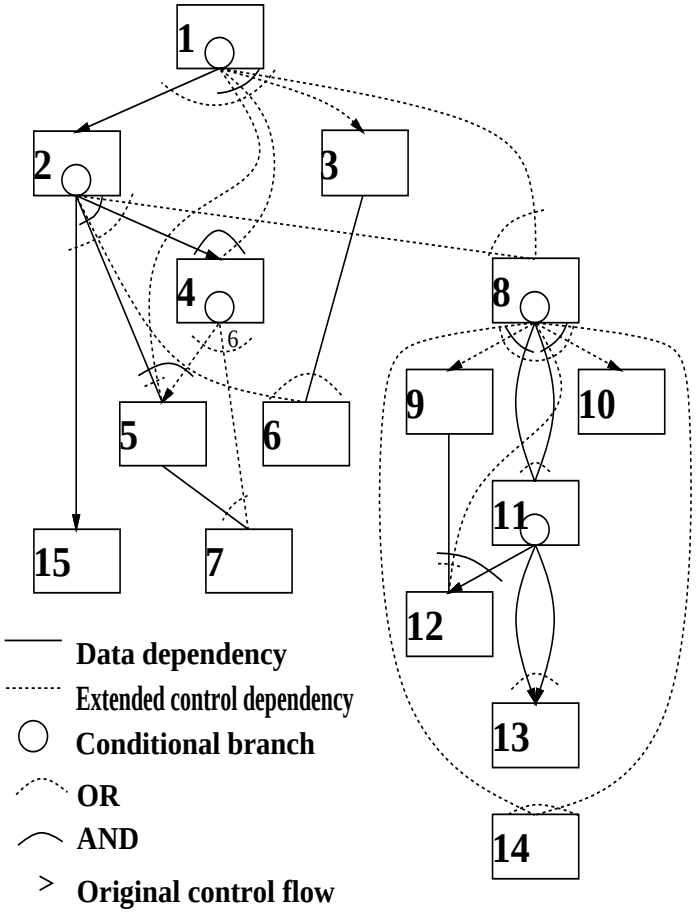
Reduction of consumed power by compiler control DVFS and Power gating with hardware supports.



Earliest Executable Condition Analysis for Coarse Grain Tasks (Macro-tasks)

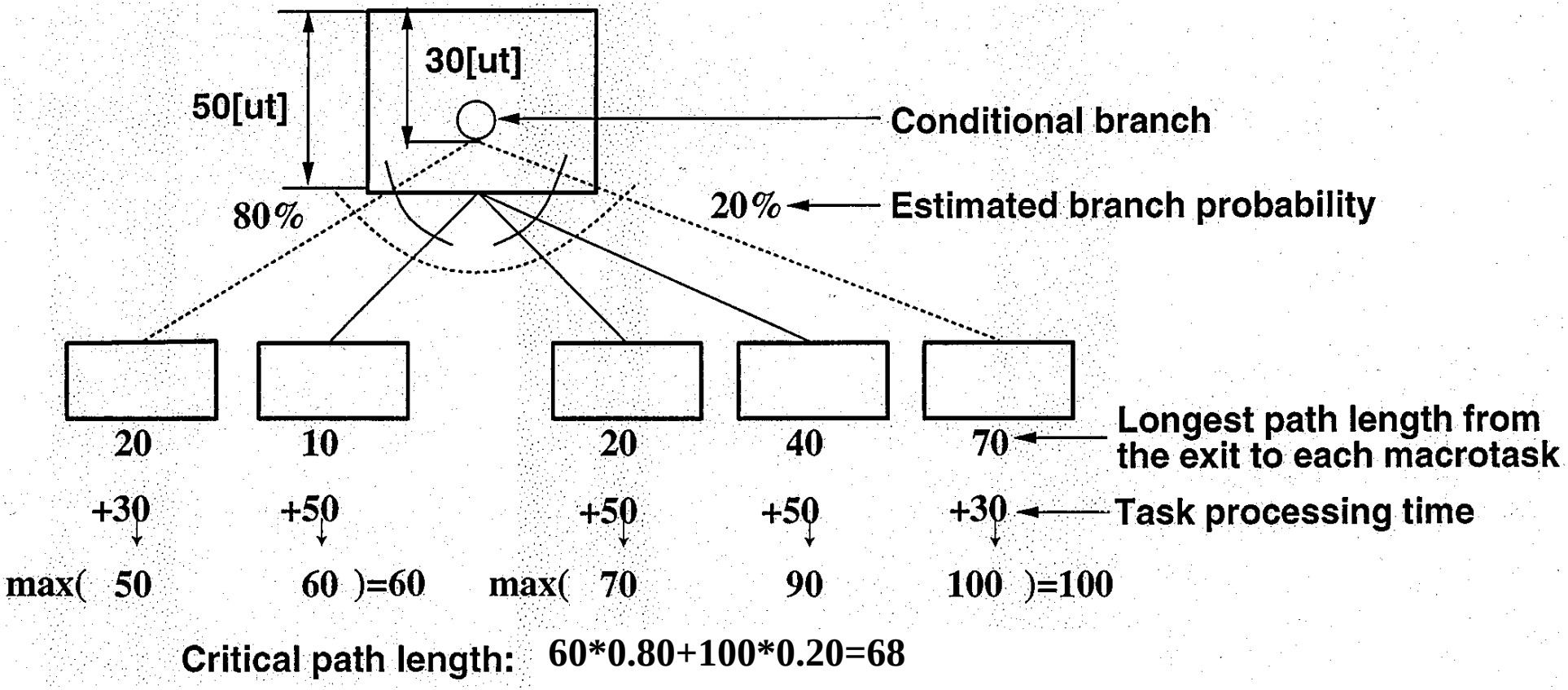


A Macro Flow Graph



A Macro Task Graph

PRIORITY DETERMINATION IN DYNAMIC CP METHOD



Earliest Executable Conditions

EEC: Control dependence + Data Dependence

Control dependences show executions of MTs are decided

Data dependences show data accessed by MTs are ready

MT2 may start execution after MT1 branches to MT2 and MT1 finish execution.

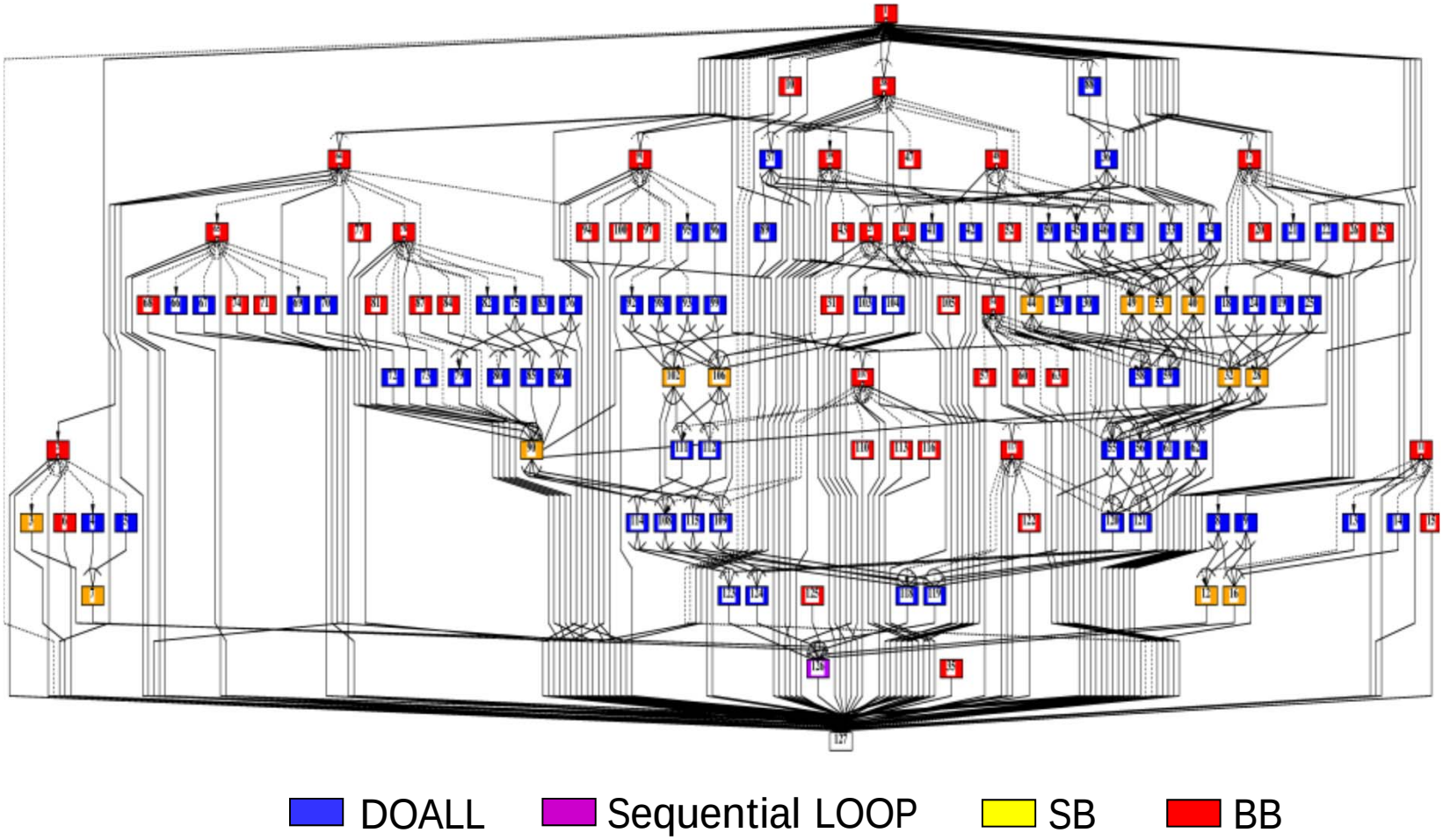
Macrotask No.	Earliest Executable Condition
1	
2	1_2
3	$(1)_3$
4	2_4 OR $(1)_3$
5	$(4)_5$ AND [2_4 OR $(1)_3$]
6	3 OR $(2)_4$
7	5 OR $(4)_6$
8	$(2)_4$ OR $(1)_3$
9	$(8)_9$
10	$(8)_{10}$
11	8_9 OR 8_{10}
12	11_{12} AND [9 OR $(8)_{10}$]
13	11_{13} OR 11_{12}
14	$(8)_9$ OR $(8)_{10}$
15	2_{15}

MT3 may start execution after MT1 branches to MT3.

MT6 may start execution after MT3 finish execution or MT2 branches to MT4.

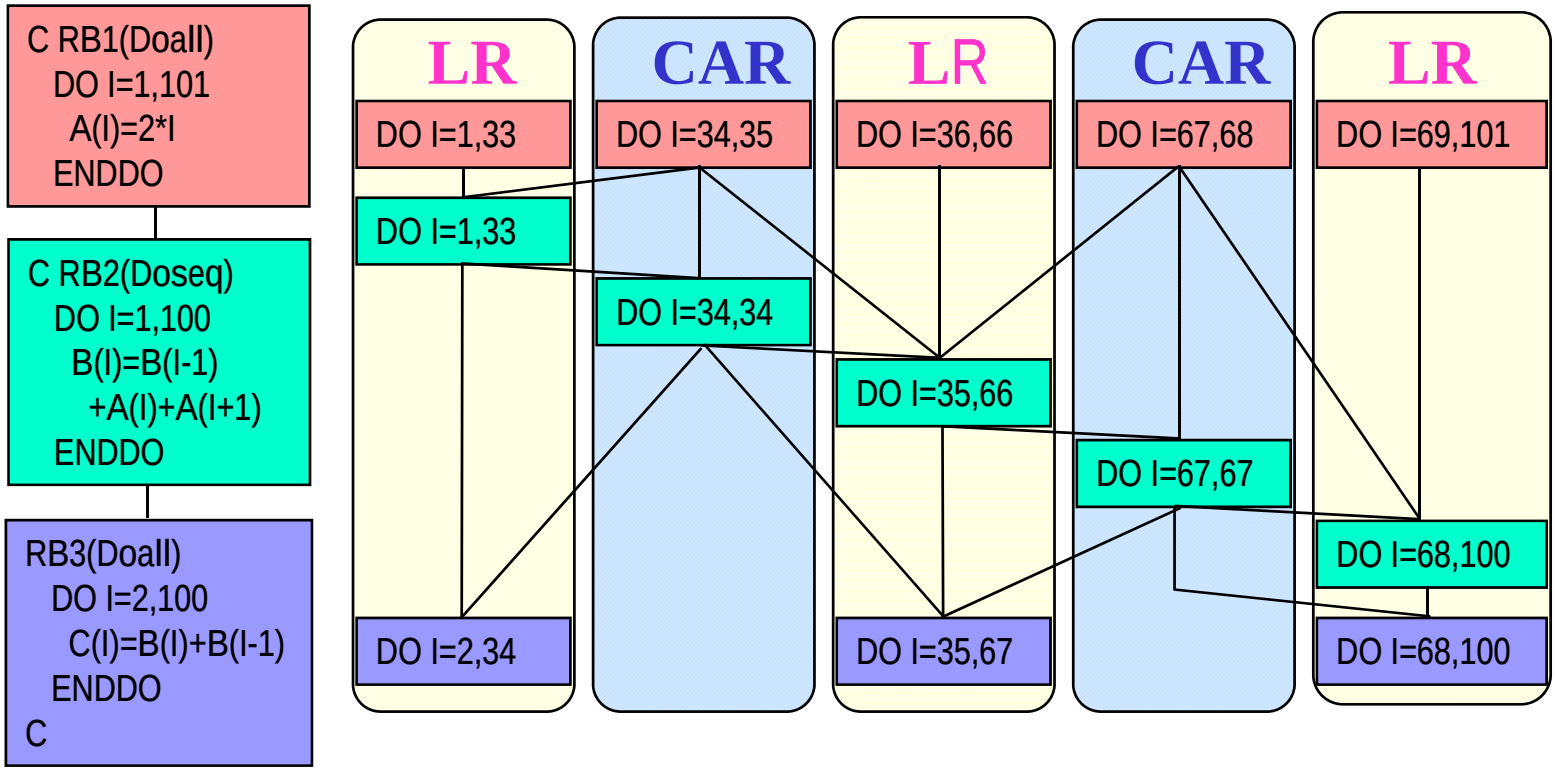
MTG of Su2cor-LOOPS-DO400

- Coarse grain parallelism PARA_ALD = 4.3



Data-Localization: Loop Aligned Decomposition

- **Decompose multiple loop (Doall and Seq) into CARs and LRs considering inter-loop data dependence.**
  - **Most data in LR can be passed through LM.**
  - **LR: Localizable Region, CAR: Commonly Accessed Region**



Multicore Program Development Using OSCAR API V2.0

Sequential Application Program in Fortran or C
(Consumer Electronics, Automobiles, Medical, Scientific computation, etc.)

Homogeneous

Hetero

Manual parallelization / power reduction

Accelerator Compiler/ User
Add "hint" directives before a loop or a function to specify it is executable by the accelerator with how many clocks

Waseda OSCAR Parallelizing Compiler
- Coarse grain task parallelization
- Data Localization
- DMAC data transfer
- Power reduction using DVFS, Clock/ Power gating

OSCAR API for Homogeneous and/or Heterogeneous Multicores and manycores
Directives for thread generation, memory, data transfer using DMA, power managements

Parallelized API F or C program

Proc0
Code with directives
Thread 0

Proc1
Code with directives
Thread 1

Accelerator 1
Code

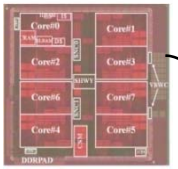
Accelerator 2
Code

Generation of parallel machine codes using sequential compilers

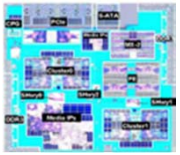
Low Power Homogeneous Multicore Code Generation
API Analyzer | Existing sequential compiler

Low Power Heterogeneous Multicore Code Generation
API Analyzer (Available from Waseda) | Existing sequential compiler

Server Code Generation
OpenMP Compiler



Homegeneous Multicore s from Vendor A (SMP servers)



Heterogeneous Multicores from Vendor B

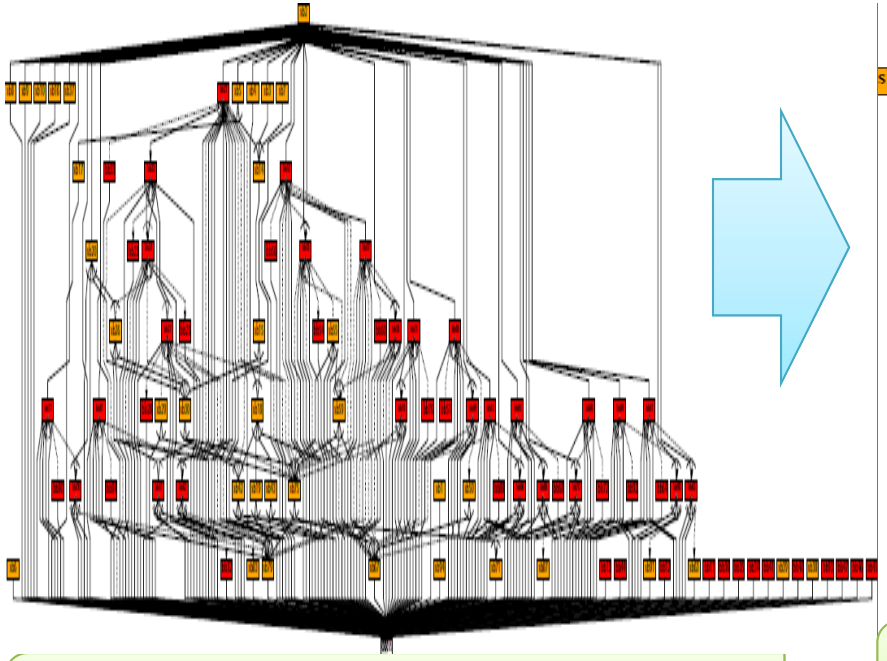
Shred memory servers

Executable on various multicores

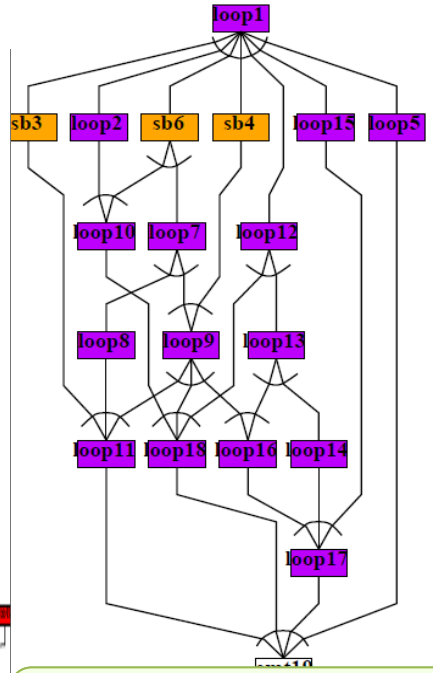
Hitachi, Renesas, NEC, Fujitsu, Toshiba, Denso, Olympus, Mitsubishi, Esol, Cats, Gaio, 3 univ.

OSCAR: Optimally Scheduled Advanced Multiprocessor
API : Application Program Interface

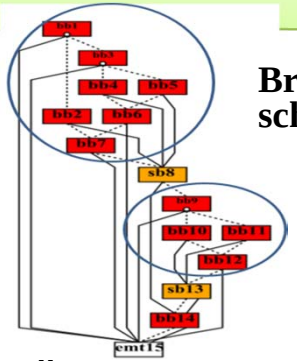
Speedup with 2cores for Engine Crankshaft Handwritten Program on Renesas RPX Multi-core Processor



Macrotask graph with a lot of conditional branches

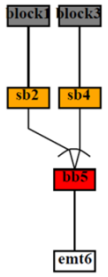


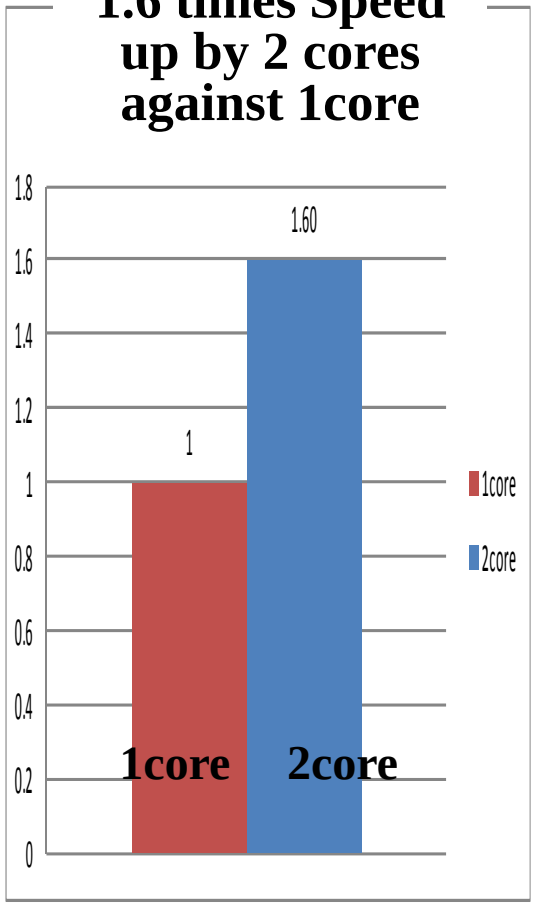
Macrotask graph after task fusion



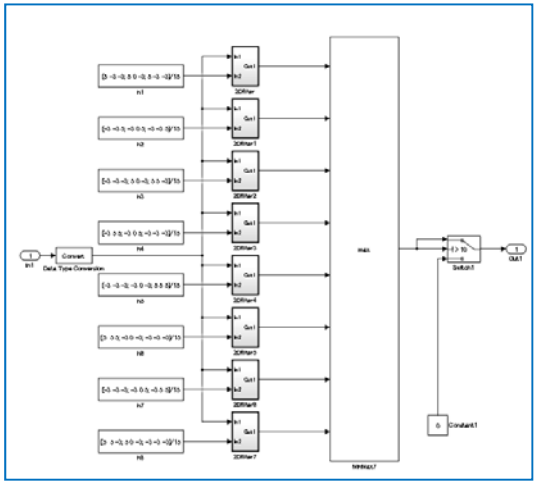
Branches are fused to macrotasks for static scheduling

Grain is too fine (us) for dynamic scheduling.





OSCAR Compile Flow for Simulink Applications



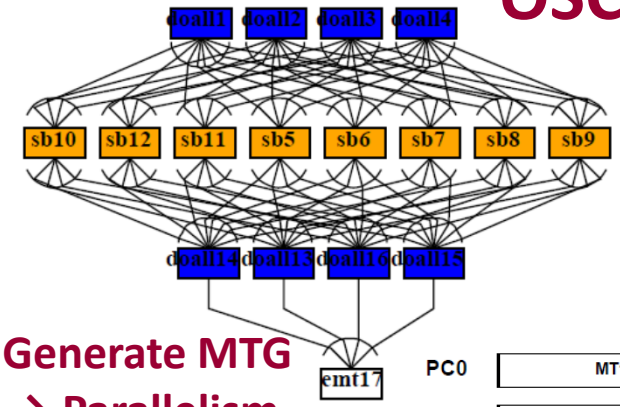
Simulink model

Generate C code
using Embedded Coder

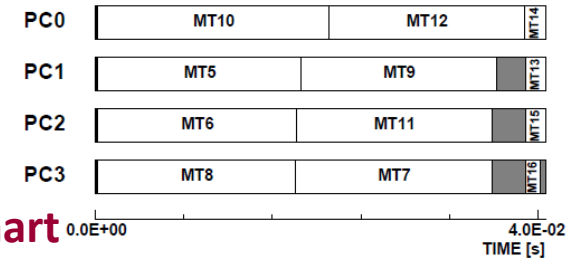
```
/* Model step function */
void VesselExtraction_step(void)
{
  int32_T i;
  real_T u0;

  /* DataTypeConversion: '<S1>/Data Type Conversion' incorporates:
   *  Inport: '<Root>/In1'
   */
  for (i = 0; i < 16384; i++) {
    VesselExtraction_B.DataTypeConversion[i] = VesselExtraction_U.In1[i];
  }

  /* End of DataTypeConversion: '<S1>/Data Type Conversion' */

  /* Outputs for Atomic SubSystem: '<S1>/2Dfilter' */

  /* Constant: '<S1>/h1' */
  VesselExtraction_Dfilter(VesselExtraction_B.DataTypeConversion,
    VesselExtraction_P.h1_Value, &VesselExtraction_B.Dfilter,
    (P_Dfilter_VesselExtraction_T *)&VesselExtraction_P.Dfilter);

  /* End of Outputs for SubSystem: '<S1>/2Dfilter' */

  /* Outputs for Atomic SubSystem: '<S1>/2Dfilter1' */

  /* Constant: '<S1>/h2' */
  VesselExtraction_Dfilter(VesselExtraction_B.DataTypeConversion,
    VesselExtraction_P.h2_Value, &VesselExtraction_B.Dfilter1,
    (P_Dfilter_VesselExtraction_T *)&VesselExtraction_P.Dfilter1);
```

C code

OSCAR Compiler



(1) Generate MTG
→ Parallelism



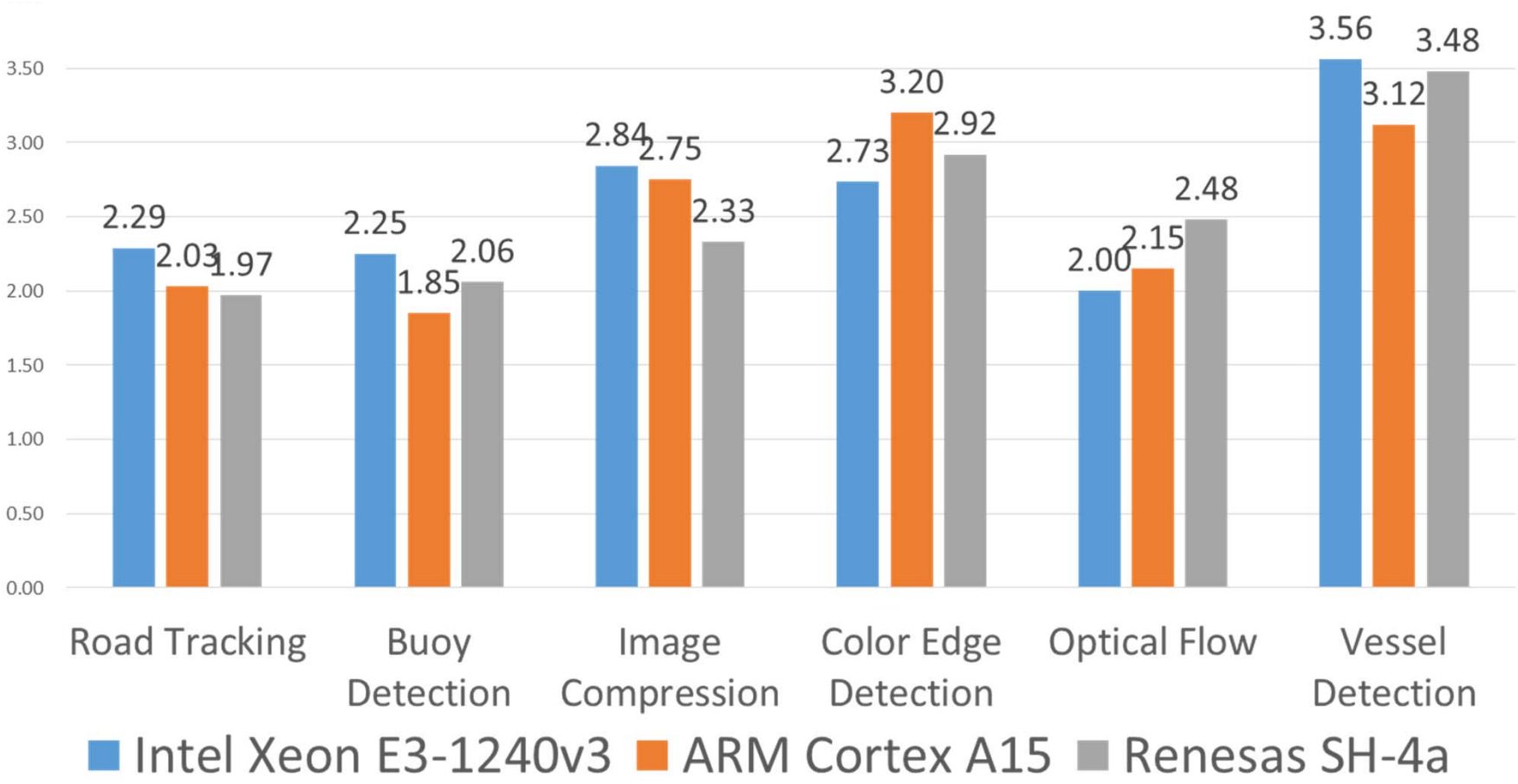
(2) Generate gantt chart
→ Scheduling in a multicore

```
void VesselExtraction_step ( )
{
    int thr1 ;
    int thr2 ;
    int thr3 ;
    {
        oscar_thread_create ( & thr1 ,
            thread_function_001 , (void*)1 ) ;
        oscar_thread_create ( & thr2 ,
            thread_function_002 , (void*)2 ) ;
        oscar_thread_create ( & thr3 ,
            thread_function_003 , (void*)3 ) ;

            VesselExtraction_step_PE0 ( ) ;

        oscar_thread_join ( thr1 ) ;
        oscar_thread_join ( thr2 ) ;
        oscar_thread_join ( thr3 ) ;
    }
}

void thread_function_001 ( void )
{
    VesselExtraction_step_PE1 ( ) ;
}
```

(3) Generate parallelized C code
using the OSCAR API
→ Multiplatform execution
(Intel, ARM and SH etc)

Speedups of MATLAB/Simulink Image Processing on Various 4core Multicores

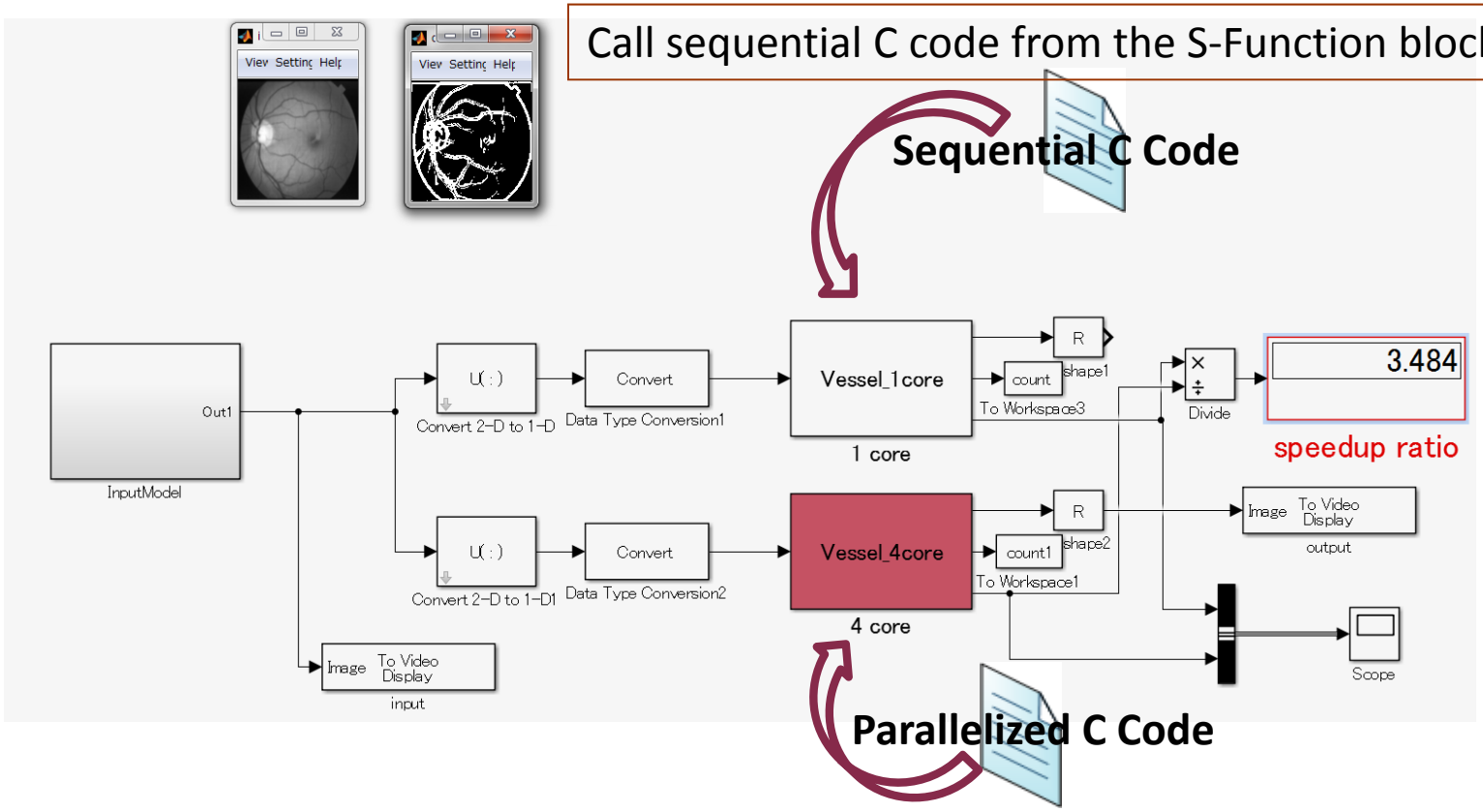
(Intel Xeon, ARM Cortex A15 and Renesas SH4A)



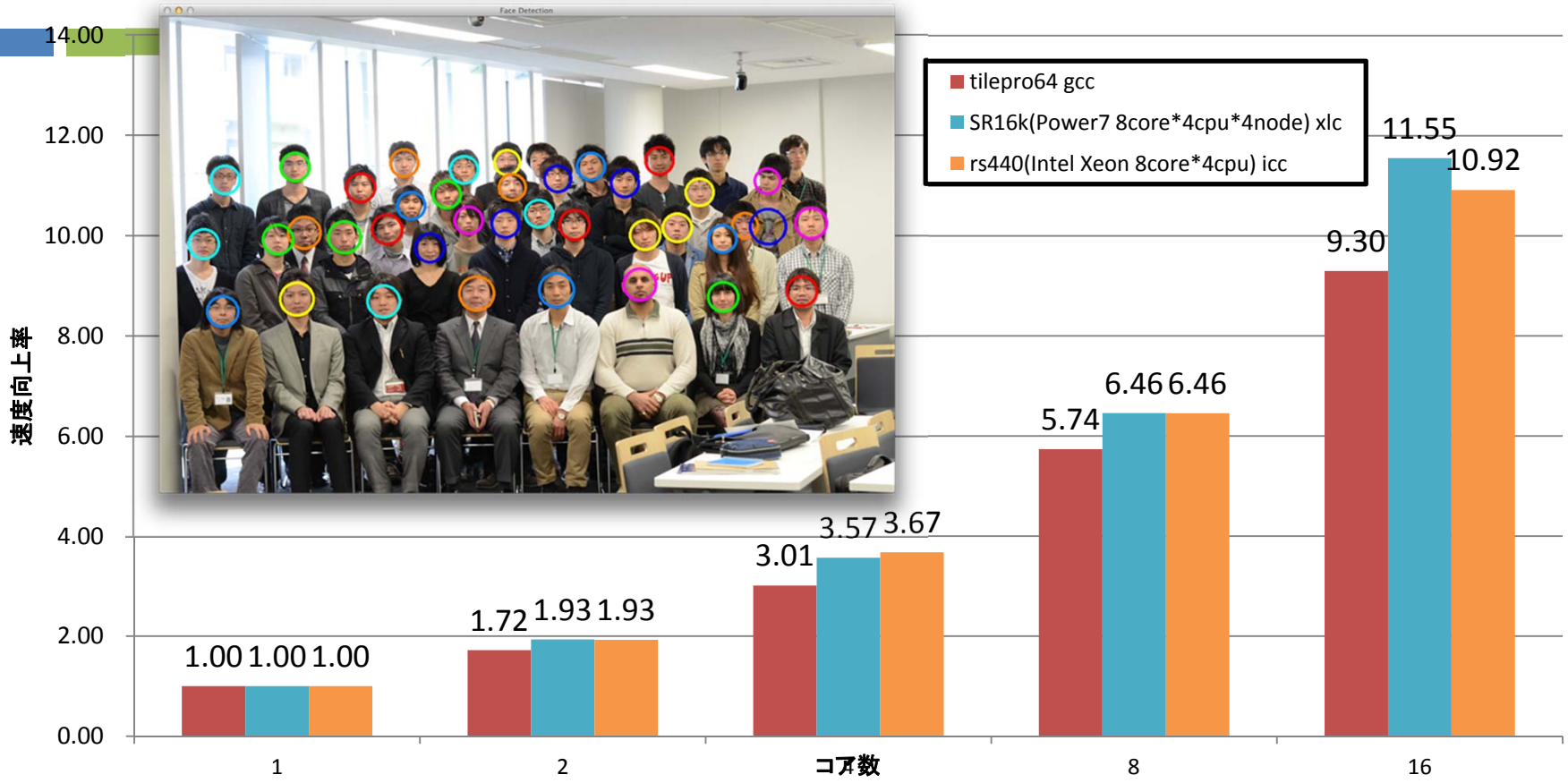
Road Tracking, Image Compression : http://www.mathworks.co.jp/jp/help/vision/examples
Buoy Detection : http://www.mathworks.co.jp/matlabcentral/fileexchange/44706-buoy-detection-using-simulink
Color Edge Detection : http://www.mathworks.co.jp/matlabcentral/fileexchange/28114-fast-edges-of-a-color-image--actual-color--not-converting-to-grayscale-/
Vessel Detection : http://www.mathworks.co.jp/matlabcentral/fileexchange/24990-retinal-blood-vessel-extraction/

Parallel Processing on Simulink Model

- The parallelized C code can be embedded to Simulink using C mex API for HILS and SILS implementation.

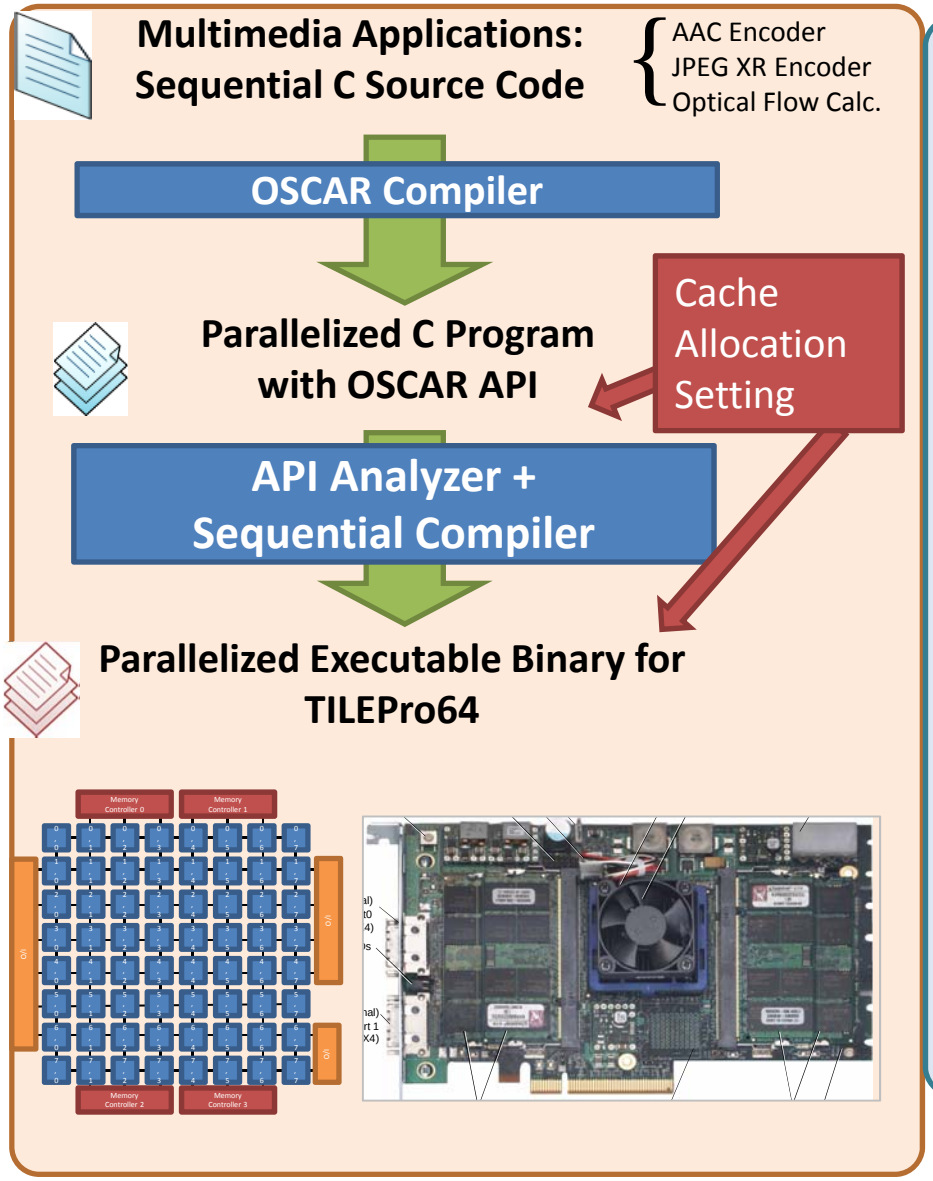


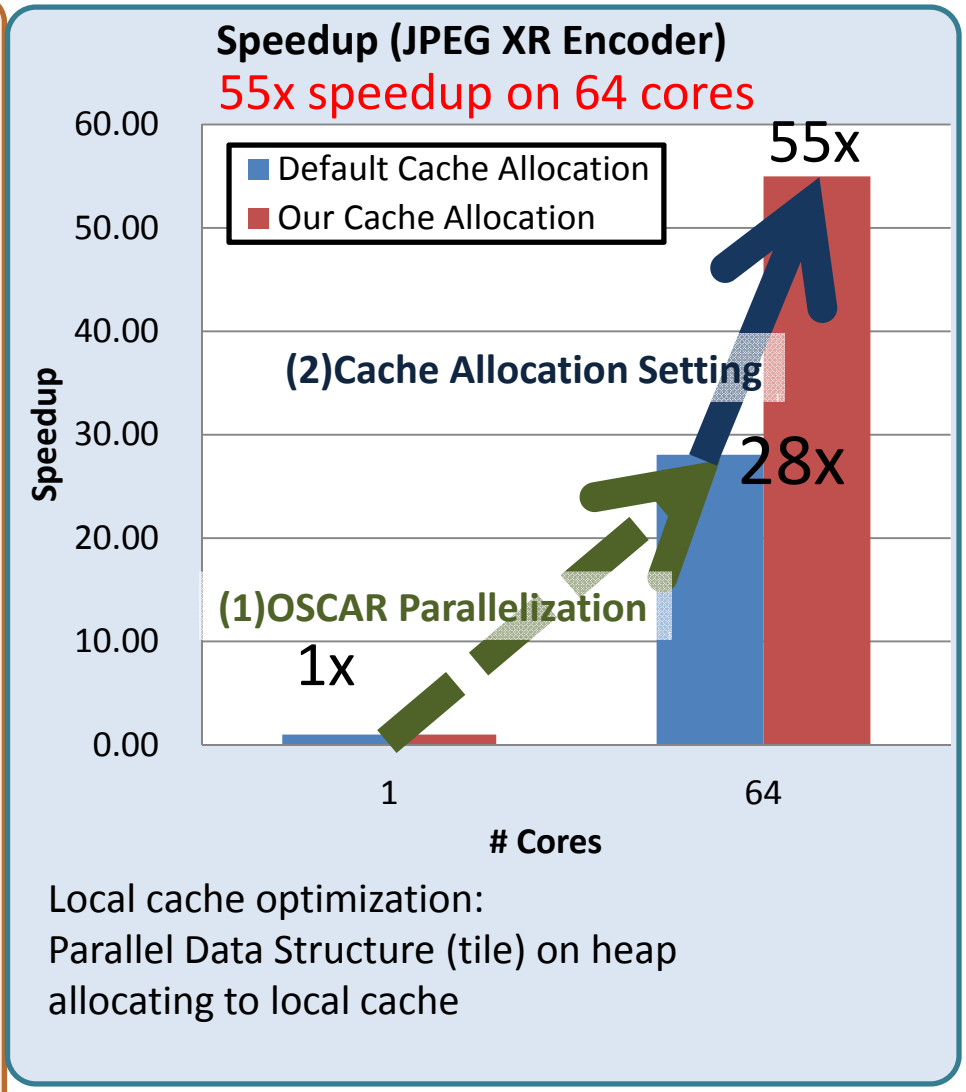
Parallel Processing of Face Detection on Manycore, Highend and PC Server



- OSCAR compiler gives us 11.55 times speedup for 16 cores against 1 core on SR16000 Power7 highend server.

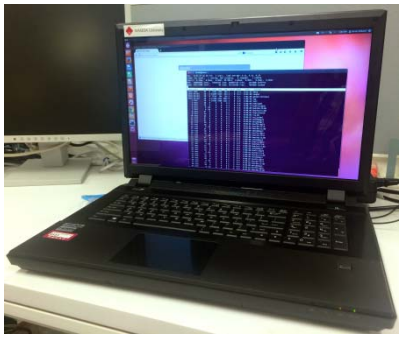
Parallel Processing of JPEG XR Encoder on TILEPro64

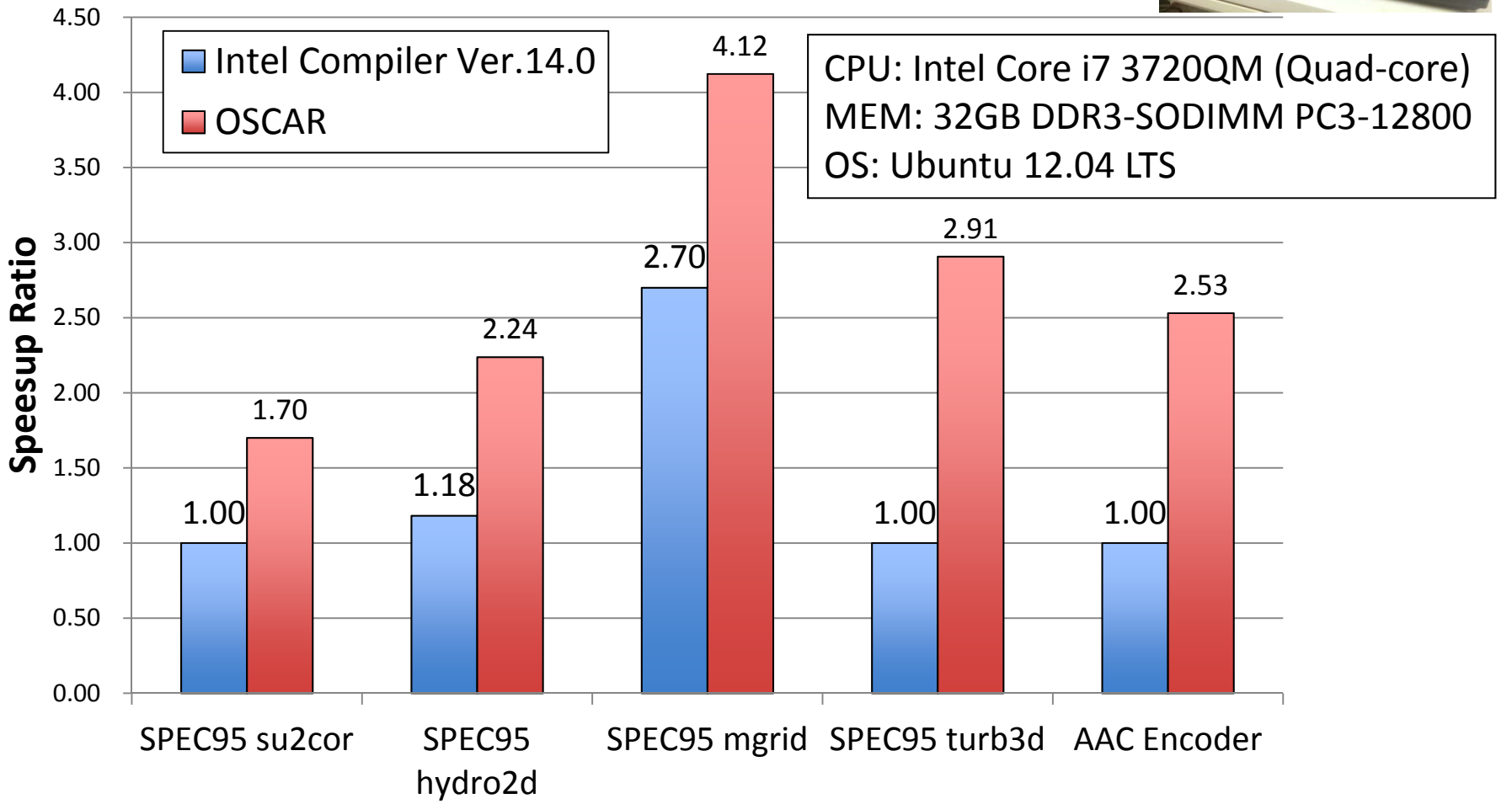




Local cache optimization:
Parallel Data Structure (tile) on heap allocating to local cache

Performance of OSCAR Compiler on Intel Core i7 Notebook PC

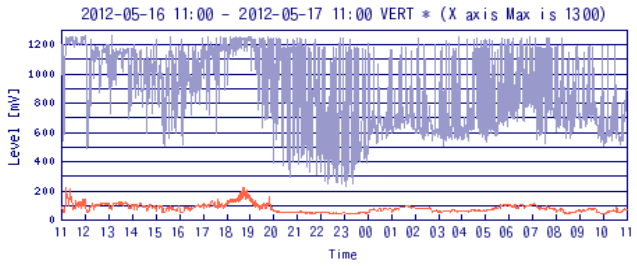


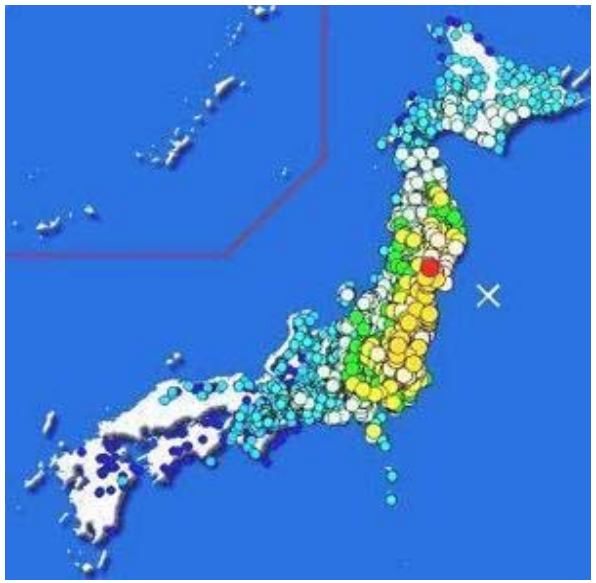


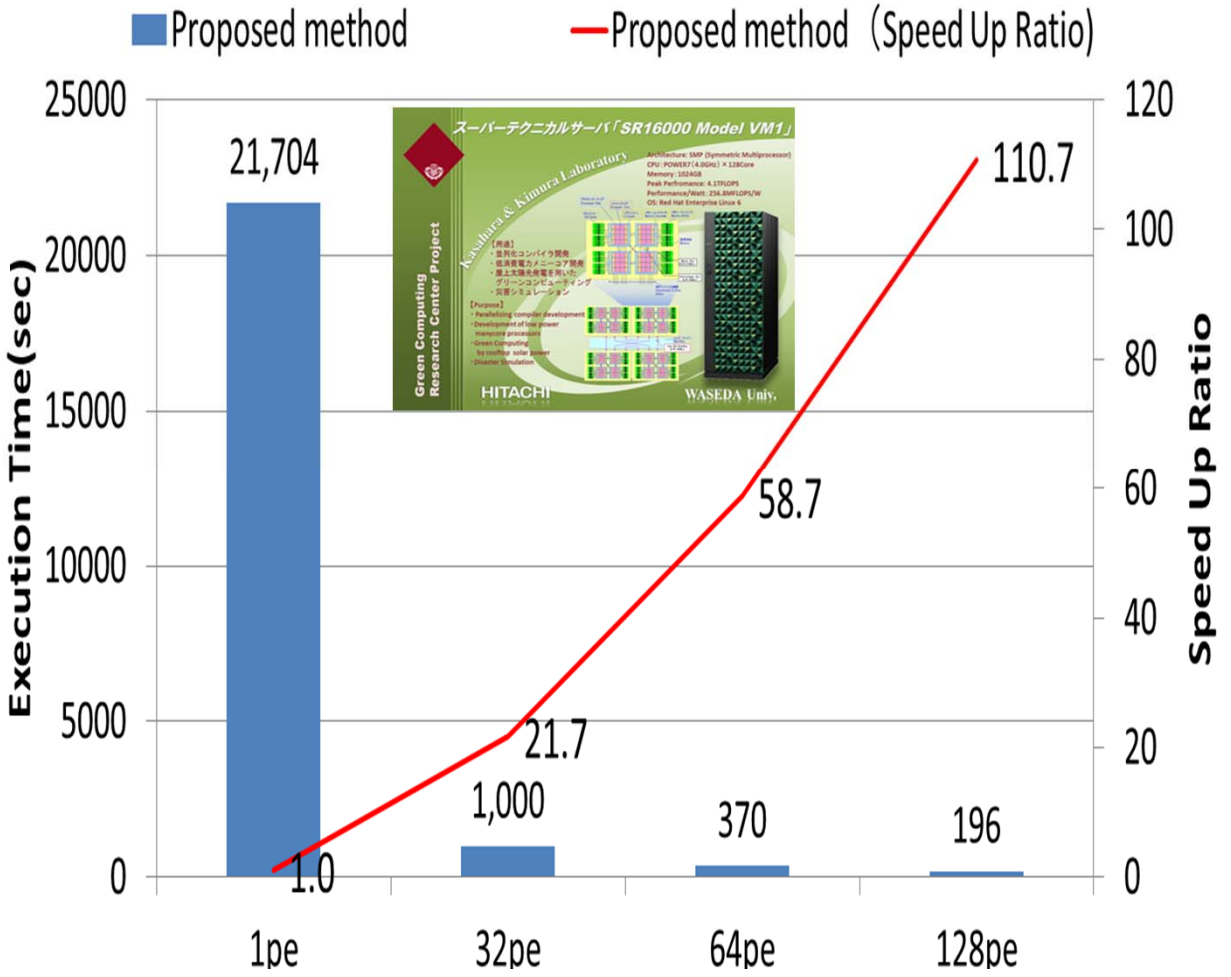
- OSCAR Compiler accelerate Intel Compiler about **2.0 times** on average

110 Times Speedup against the Sequential Processing for GMS Earthquake Wave Propagation Simulation on Hitachi SR16000

（Power7 Based 128 Core Linux SMP）

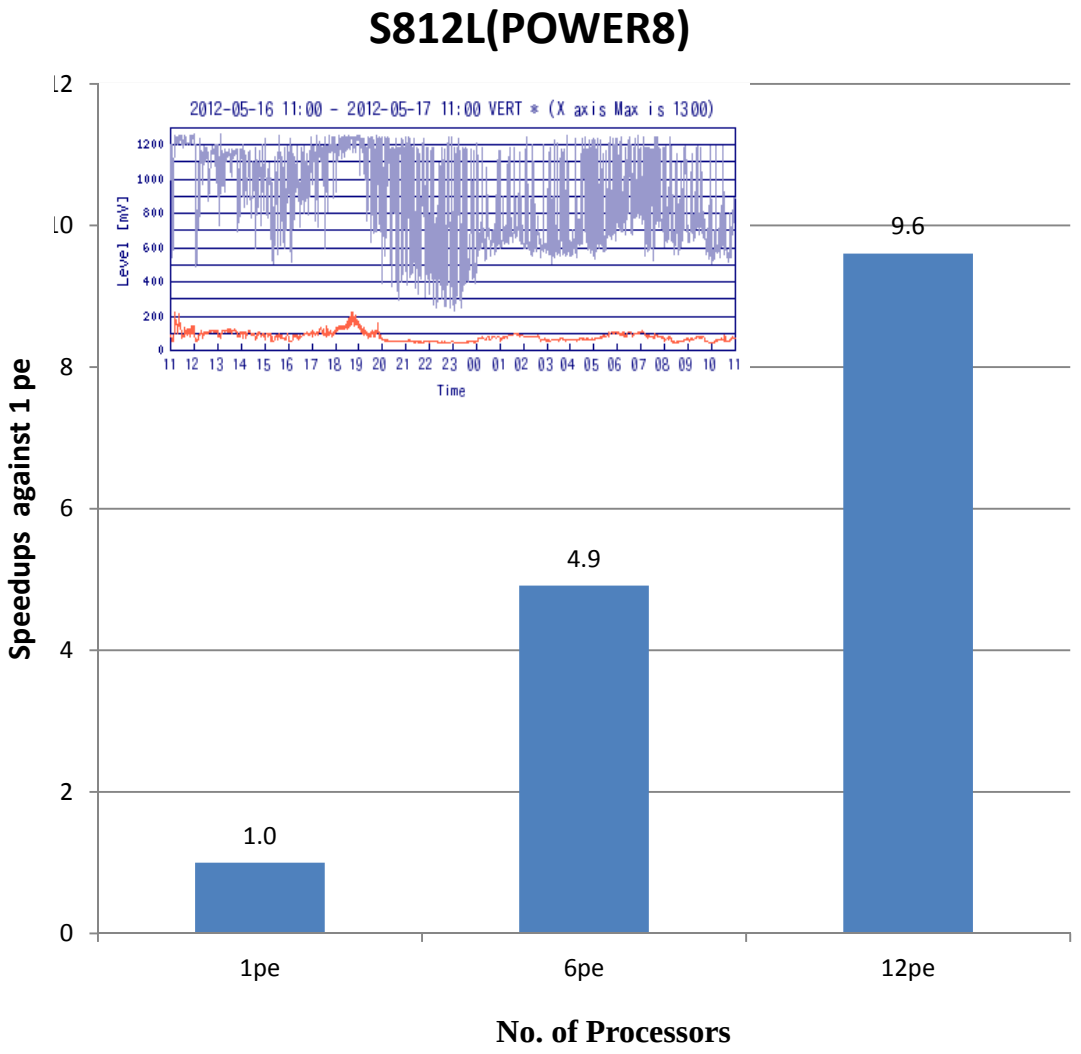






9.6 Times Speedup on 12 cores Power 8 against the Sequential Processing for GMS Earthquake Wave Propagation Simulation

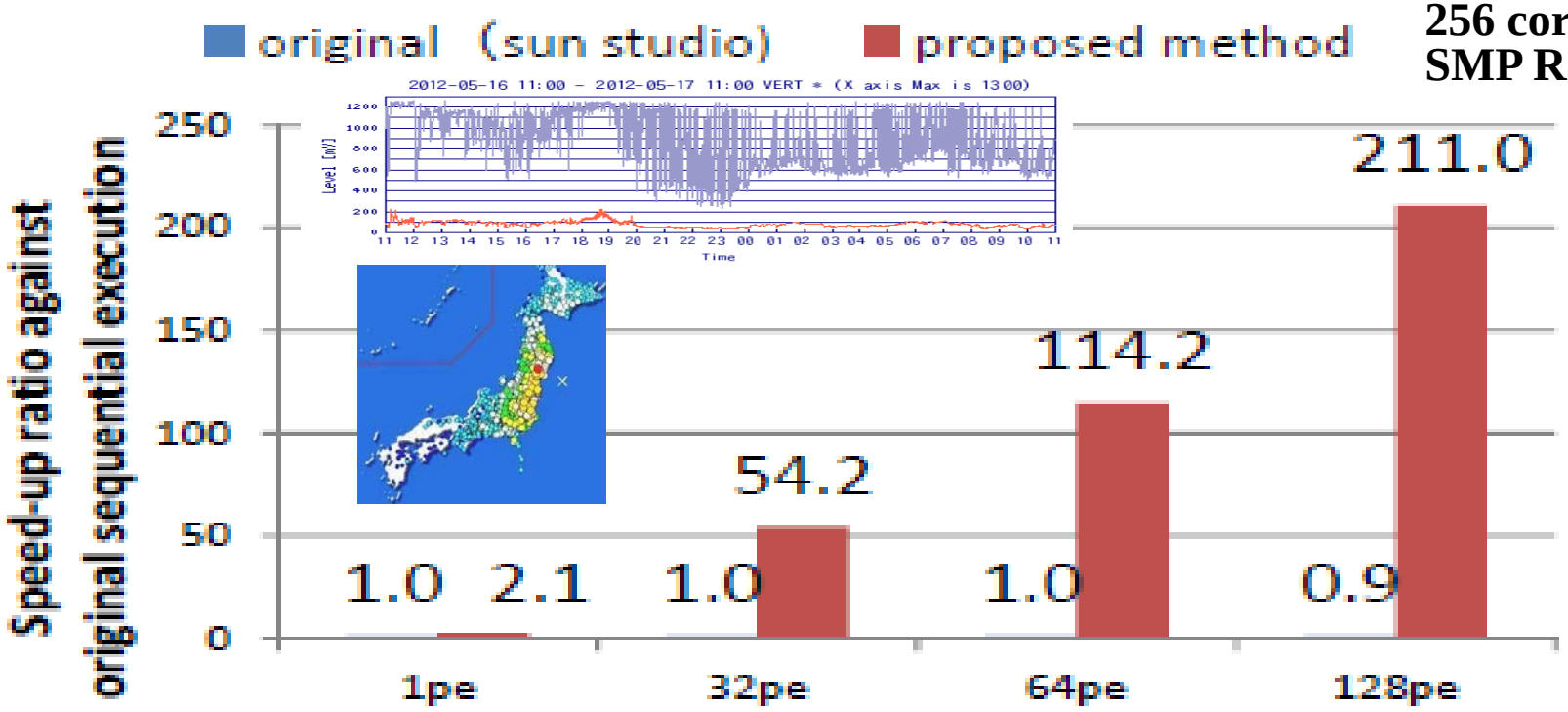
- IBM S812L
  - CPU:POWER8
    - 12 cores
    - Clock Frequency 3.026GHz
  - Memory:60GB
  - OS:Redhat Linux 7.1
  - Backend Fortran compiler: IBM xlf 15.1.1
  - Evaluation using medium size input data(12GB)



Save lives form natural disasters

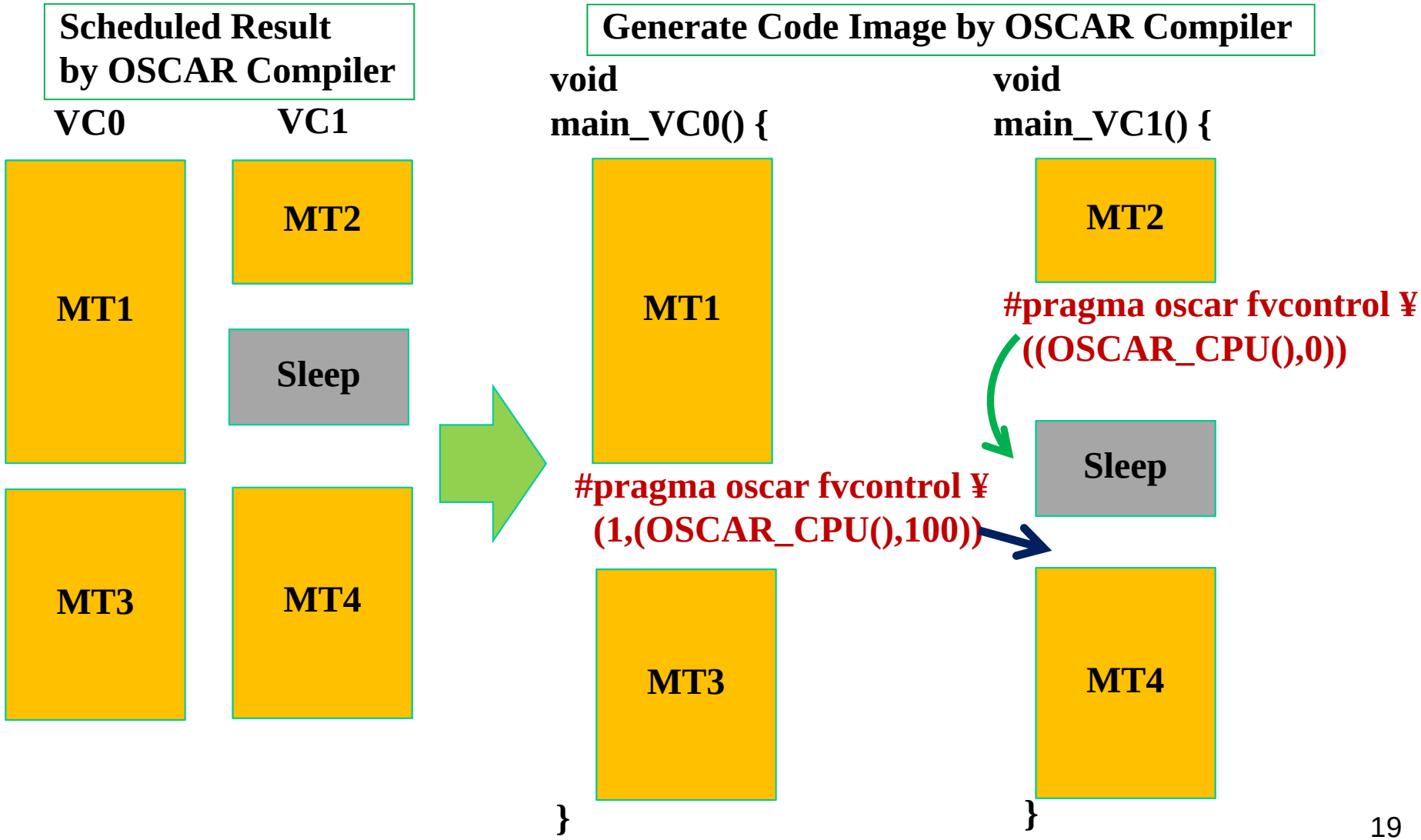
211 Times Speedup against the Sequential Processing using Sun Studio Compiler for GMS Earthquake Wave Propagation Simulation on Fujitsu M9000 Sparc 128 core SMP

Fujitsu M9000 Sparc 256 core SMP Racks



- 100 times speedup on 128 cores against one core using OSCAR compiler
- 211 times speedup on 128 cores against original GMS program on one core using OSCAR Sun Studio Compiler

Low-Power Optimization with OSCAR API



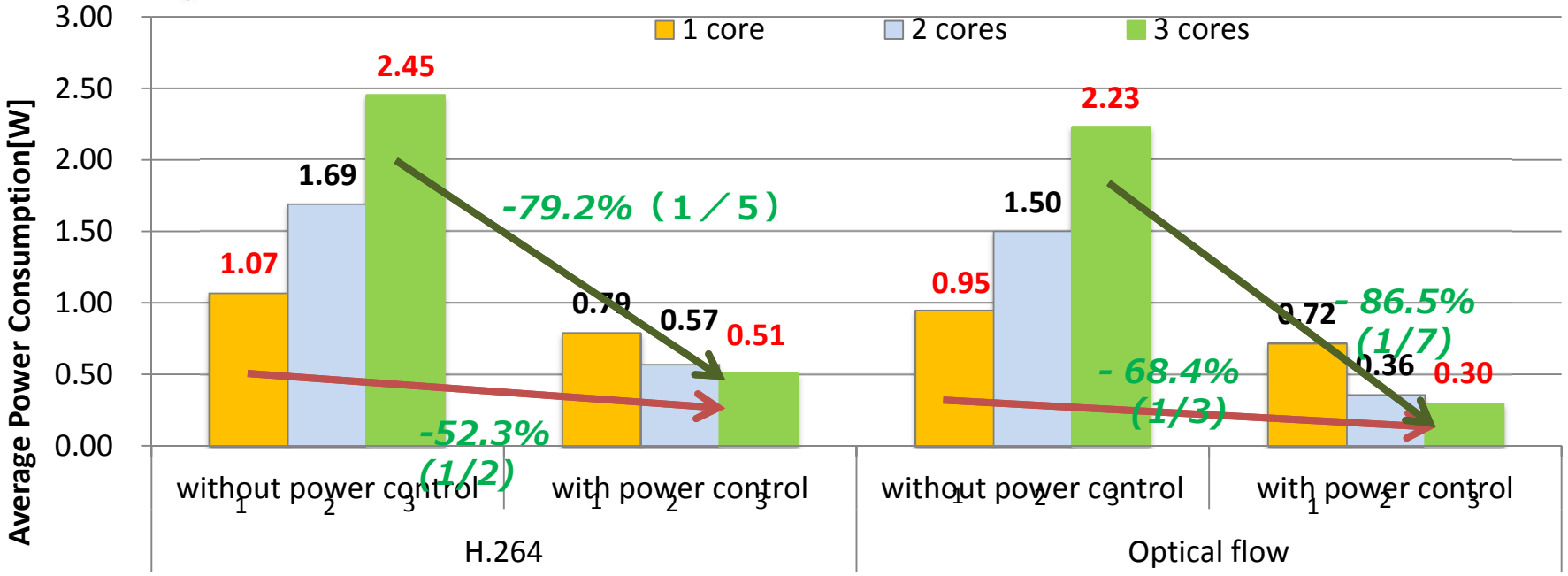
Power on 4 cores ARM CortexA9 with Android

http://www.youtube.com/channel/UCS43lNYEIkC8i_KIgFZYQBQ

H.264 decoder & Optical Flow (Using 3 cores)

ODROID X2

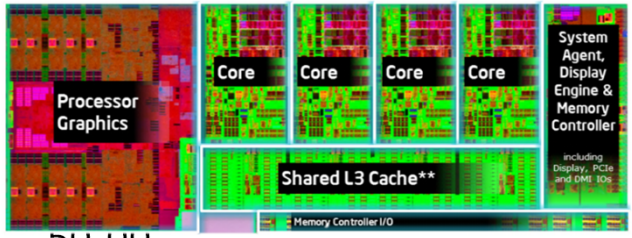
Samsung Exynos4412 Prime, ARM Cortex-A9 Quad core
1.7GHz～0.2GHz, used by Samsung's Galaxy S3



- On the same 3 cores, the power control reduced the power to 1/5～1/7 against no power control.
- The power control reduced the power to 1/2～1/3 compared with the ordinary sequential execution on 1 core without power control.

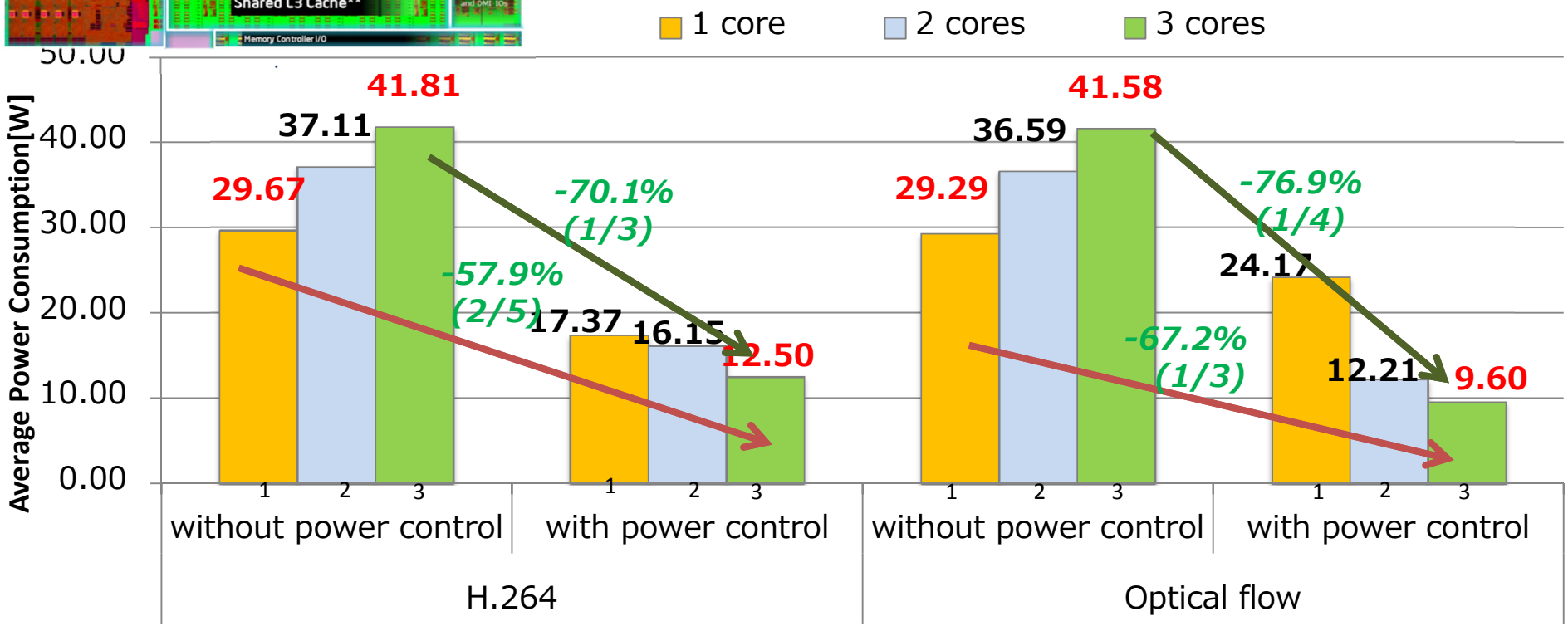
Power Reduction on Intel Haswell 3cores

H.264 decoder & Optical Flow



H81M-A, Intel Core i7 4770k

Quad core, 3.5GHz～0.8GHz



- **The power consumption was reduced to 1/3～1/4 by OSCAR power control against no power control on the same 3 processor cores.**
- **The power reduced to 2/5～1/3 by the compiler power control against 1core no power control.**

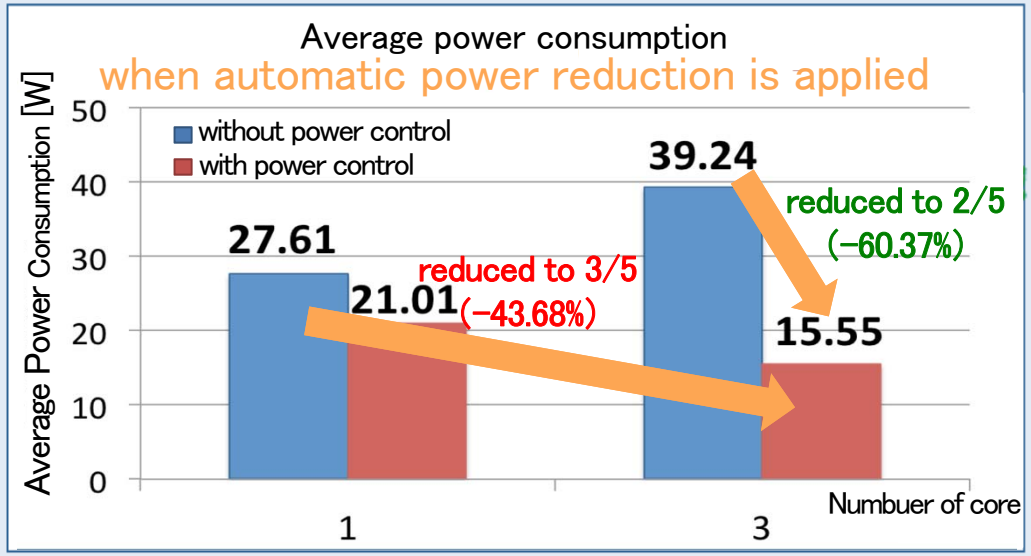
Automatic Power Reduction by OSCAR Compiler on Intel Haswell 4 Core Multicore

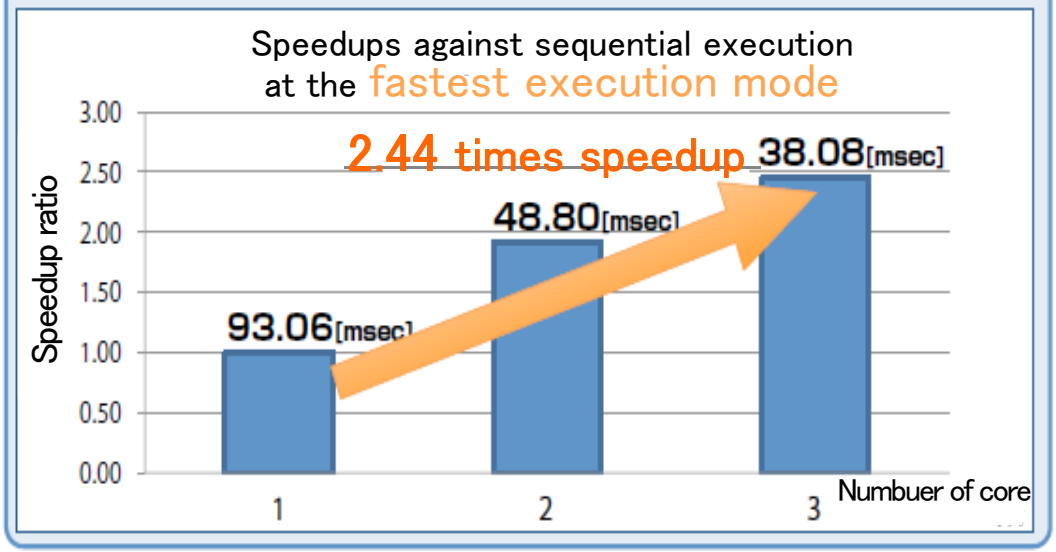
- OSCAR Compiler
- Intel Haswell
- Power Reduction

- Power Consumption for real-time face detection was reduced to 2/5 -

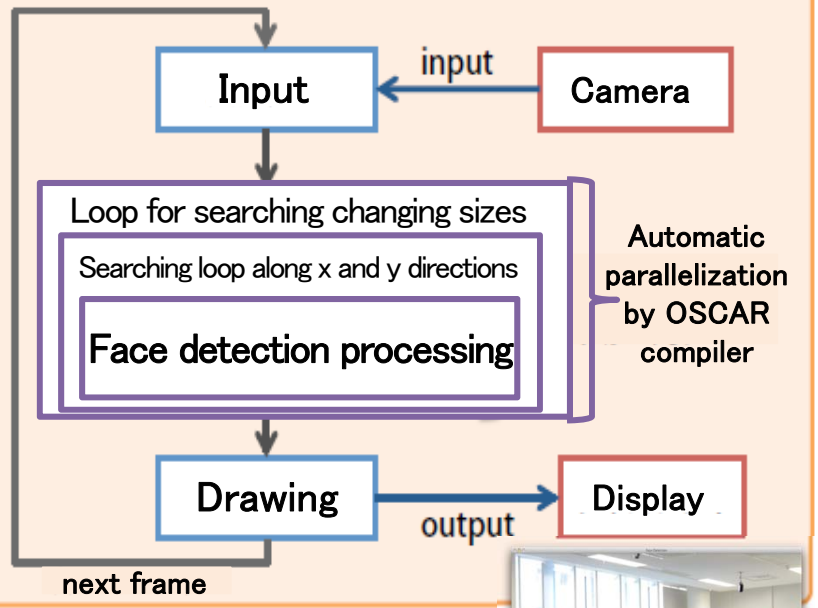
Waseda University

Parallel processing of face detection program on Intel Haswell 4cores





Parallelization flow of OpenCV face detection program

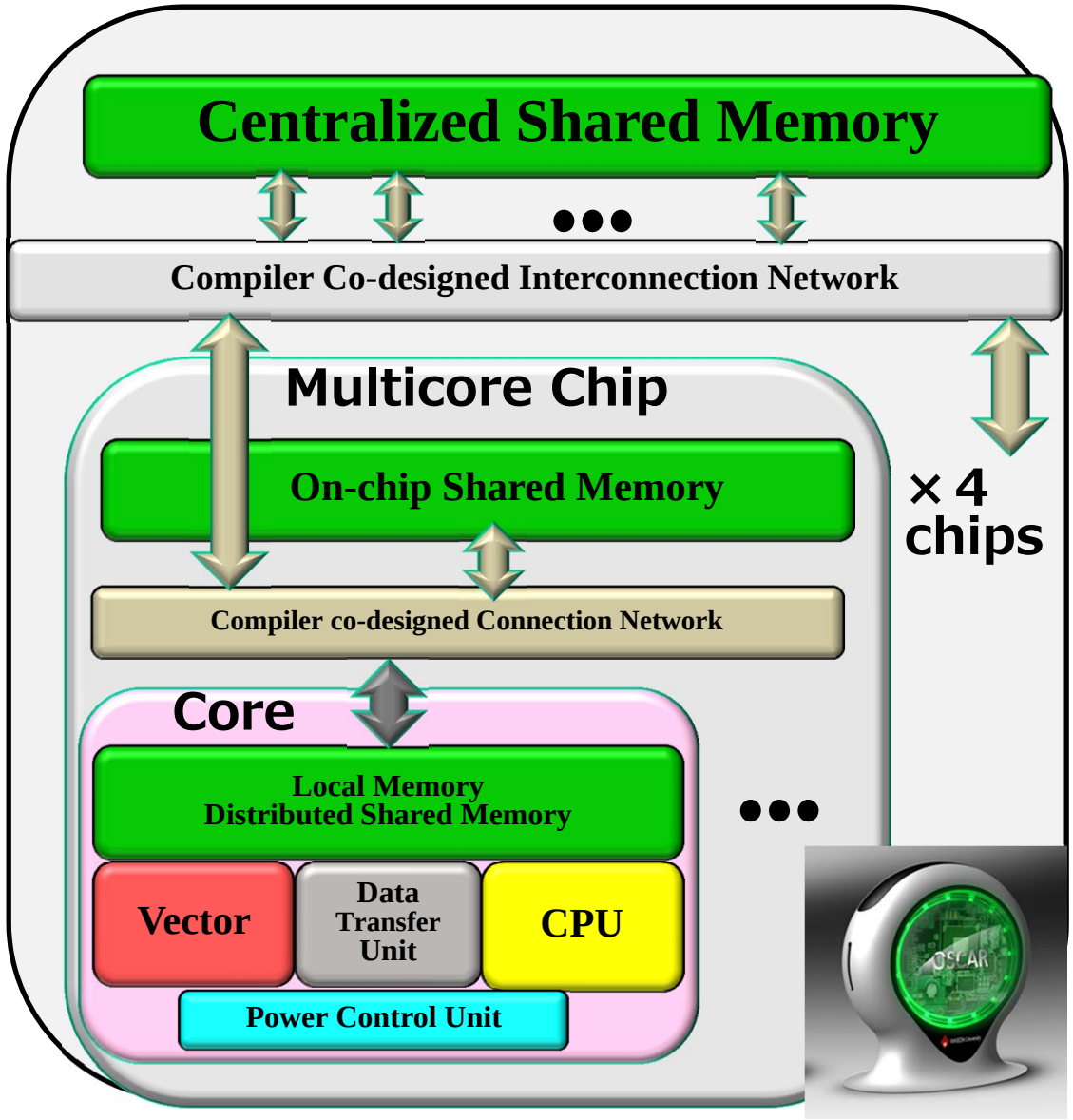


Power measurement on Intel Haswell board

CPU : Inel Core i7 4770K
Number of core : 4
Clock frequency : 3.5GHz～0.8GHz
Mother board : ASUS H81M-A

Inserting power measurement circuit between PMIC and CPU

OSCAR Vector Multicore and Compiler for Embedded to Severs with OSCAR Technology



Target:

- **Solar Powered with compiler power reduction.**
- **Fully automatic parallelization and vectorization including local memory management and data transfer.**

Summary

- OSCAR Automatic Parallelizing and Power Reducing Compiler has succeeded speedup and/or power reduction of scientific applications including, medical applications including "Cancer Treatment Using Carbon Ion", and "Drinkable Inner Camera", industry application including "Automobile Engine Control", and "Smartphone", on various multicores from different vendors including Intel, ARM, Renesas and Fujitsu.
- In automatic parallelization, 110 times speedup for "Earthquake Wave Propagation Simulation" on 128 cores of IBM Power 7 against 1 core, 1.60 times for handwritten "Automobile Engine Control" on Renesas 2 cores using SH4A, 55 times for "JPEG-XR Encoding for Capsule Inner Cameras" on Tilera 64 cores Tile64 manycore, 3.56, 3.12 and 3.45 times for "MATLAB/Simulink Vessel Detection program " on 4 cores Intel Xeon, ARM Cortex A15 and Renesas SH4A respectively.
- In automatic power reduction, consumed powers for H.264 and optical flow were reduced to 1/2 or 1/3 using 3 cores of ARM Cortex A9 and Intel Haswell against ordinary single core execution and real-time Human face detection to 3/5 using 3 cores of Haswell.