

Software and Hardware for High Performance and Low Power Homogeneous and Heterogeneous Multicore Systems

Hironori Kasahara

Professor, Dept. of Computer Science & Engineering

Director, Advanced Multicore Processor Research Institute

Waseda University, Tokyo, Japan

IEEE Computer Society President Elect 2017, President 2018

1980 BS, 82 MS, 85 Ph.D. , Dept. EE, Waseda Univ.

1985 Visiting Scholar: U. of California, Berkeley

1986 Assistant Prof., 1988 Associate Prof., 1997

**Prof. Dept. of EECE, Waseda Univ. Now Dept. of
Computer Sci. & Eng.**

**1989-90 Research Scholar: U. of Illinois, Urbana-
Champaign, Center for Supercomputing R&D**

1987 IFAC World Congress Young Author Prize

1997 IPSJ Sakai Special Research Award

2005 STARC Academia-Industry Research Award

2008 LSI of the Year Second Prize

2008 Intel Asia Academic Forum Best Research Award

2010 IEEE CS Golden Core Member Award

2014 Minister of Edu., Sci. & Tech. Research Prize

2015 IPSJ Fellow

2017 IEEE Fellow

**Reviewed Papers: 214, Invited Talks: 145, Published
Unexamined Patent Application: 59 (Japan, US, GB,
China Granted Patents: 30), Articles in News Papers,
Web News, Medias incl. TV etc.: 572**

Committees in Societies and Government 245

**IEEE Computer Society President 2018, BoG(2009-
14), Multicore STC Chair (2012-), Japan Chair (2005-
07), IPSJ Chair: HG for Mag. & J. Edit, Sig. on ARC.**

**【METI/NEDO】 Project Leaders: Multicore for
Consumer Electronics, Advanced Parallelizing
Compiler, Chair: Computer Strategy Committee
【Cabinet Office】 CSTP Supercomputer Strategic
ICT PT, Japan Prize Selection Committees, etc.**

**【MEXT】 Info. Sci. & Tech. Committee,
Supercomputers (Earth Simulator, HPCI Promo.,
Next Gen. Supercomputer K) Committees, etc.**

Guang R. Gao

University of Delaware

About Awards

[Nominate](#)

[Recipients](#)

[Fellows](#)

[Technical Awards](#)

[Education Awards](#)

[Service Awards and
Certificates](#)

[Other Awards](#)

[Discontinued Awards](#)

[Awards FAQ](#)

Awards Resources

[Nomination site](#)

[Committee](#)

[Handbook](#)

[2017 Awards Brochure](#)

[New!](#)

[2017 Ceremony Photos](#)

[New!](#)

[SC16 Awards Session](#)

[SC16 Awards Session](#)

[Photos](#)

[Press Releases](#)

[Award Videos](#)

[Fellows](#)

2017 B. Ramakrishna Rau Award Recipient

"For contributions to compiler techniques and microarchitectures for instruction-level and thread-level parallel computing."



Guang R. Gao has been a faculty member at the University of Delaware since 1996, and is an endowed distinguished Professor of Electrical and Computer Engineering. He received his Ph.D. degree in 1986 in Computer Science at MIT -- the first from mainland China. Gao is an alumni of Tsinghua University in Beijing. Gao is a ACM Fellow and IEEE Fellow since 2007.

Gao has devoted a majority of his research and academic activities in pursuing R&D in dataflow program execution models and architecture/compiler innovations. The work of Gao and his students have shown that the fundamental value of the dataflow model of computation in addressing the challenges and bottleneck encountered in parallel computer systems under classical von-Neumann architectures model. To this end, Gao has led a series of parallel architecture and system projects where various aspects of dataflow models are explored and realized in high-performance computers -- ranging from innovations in programming paradigms, microarchitecture design, to system software technology (compilers, runtime, etc.) and multicore chip hardware design methodology. Gao has made unique and significant contributions in compiler techniques and microarchitectures for instruction-level and thread-level parallelism.

The impact of Gao's work has also been recognized with his entrepreneur effort to apply his academic R&D results successfully to a significant supercomputing system project in mid-2000s - known as IBM Cyclops-64 Supercomputer - one of the largest supercomputer based on large-scale many-core chip technology at that time and has been in use in real world.





Home News Timeline Our Team Publication Knowledge Base STC Event

Background

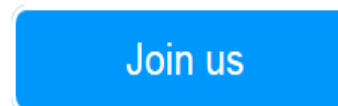
Motivation

Focus

Mission

Background

Since the mid 2000s, with the advent of multicore architectures, parallel computer and systems have witnessed a remarkable comeback from the downturn of the early 90s. Some have called it the “**second spring**” of parallel computing and the trend has continued. It is a good time for this STC’s birth.



Contact us

Chair

Guang R. Gao, IEEE fellow (UD)

Vice Chair

Jean-Luc Gaudiot, IEEE fellow (UCI)

Organizing committee

Advisors (see list...)

Coordinators

Newsletter (Dr. Xiaoming Li)

Tech Activities (Dr. Sunita Chandrasekaran)

Conferences (Dr. Stéphane Zuckerman)

Web (Dr. Long Zheng)

Student Volunteers

Jose M Monsalve (UD)

Siddhisanket Raskar

Motivation

Our STC is motivated by the challenges that are coming with the second-spring of parallel computation — including but not limited to

- Demands for high-performance parallel systems with potentially extreme-scale performance;
- Computation demands from data-intensive workloads with “big data” volumes and real-time requirements, as well as intelligent data analytic processing capability;
- Demands for high energy efficiency and strong system resiliency. This would be particularly the case for embedded mobile systems.

Focus

Our STC focuses on all manners of directions derived from Dataflow, as listed below,

STC Founding Member

These are the people that made possible the creation of this IEEE Special Technical Community.

- **Arvind**

Massachusetts Institute of Technology, USA

- **David Abramson**

The University of Queensland, Australia

- **Georgi Gaydadjiev**

Maxeler, Imperial College London, UK

- **Guang R. Gao**

University of Delaware, USA

- **Hirarki Kei**

The University of Tokyo, Japan

- **Hironori Kasahara**

Waseda University, Japan

- **Jack B. Dennis**

Massachusetts Institute of Technology, USA

- **Jean-Luc Gaudiot**

University of California, Irvine, USA

- **Jin Hai**

Huazhong University of Science and Technology, China

- **Kuan-Ching Li**

Providence University, Taiwan

- **Mateo Valero**

Universitat Politècnica de Catalunya, Spain

- **Mei Hong**

Beijing Institute of technology, China

- **Nahid Emad**

University of Versailles, France

- **Nelson Amaral**

University of Alberta, Canada

- **R. Govindarajan**

Indian Institute of Science, India

- **Roberto Giorgi**

Università di Siena, Italy

- **Skevos Evripidou**

University of Cyprus, Cyprus

- **Stephane Zuckerman**

Michigan Technological University, USA

- **Vivek Sarkar**

Rice University, USA

- **Won Woo Ro**

Yonsei University, South Korea

- **Zheng Weimin**

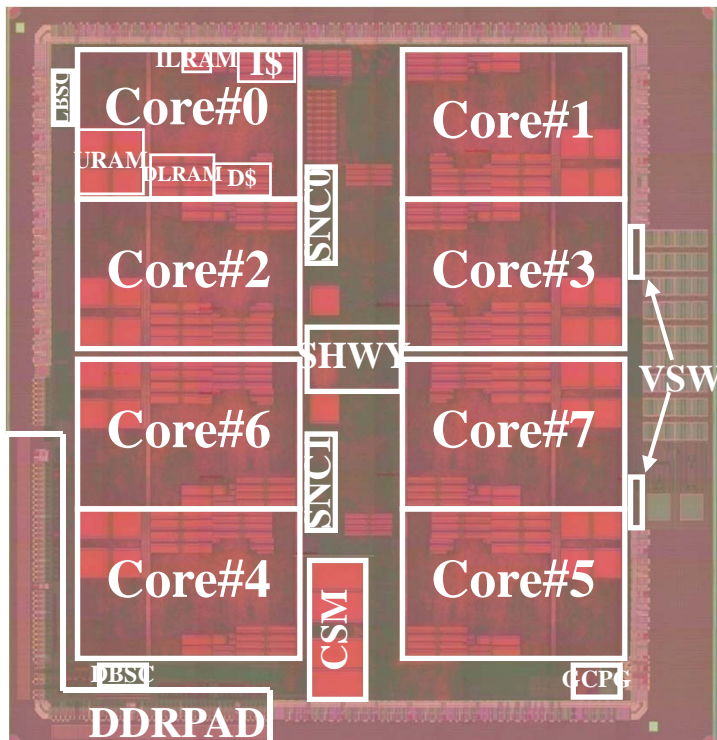
Tsinghua University, China

IEEE Computer Society BoG (Board of Governors) Feb.1, 2017



Multicores for Performance and Low Power

Power consumption is one of the biggest problems for performance scaling from smartphones to cloud servers and supercomputers (“K” more than 10MW) .



IEEE ISSCC08: Paper No. 4.5,
M.Ito, ... and H. Kasahara,
“An 8640 MIPS SoC with
Independent Power-off Control of 8
CPUs and 8 RAMs by an Automatic
Parallelizing Compiler”

$$\text{Power} \propto \text{Frequency} * \text{Voltage}^2$$

(Voltage \propto Frequency)

➔ $\text{Power} \propto \text{Frequency}^3$

If Frequency is reduced to 1/4
(Ex. 4GHz→1GHz),
Power is reduced to 1/64 and
Performance falls down to 1/4 .

<Multicores>

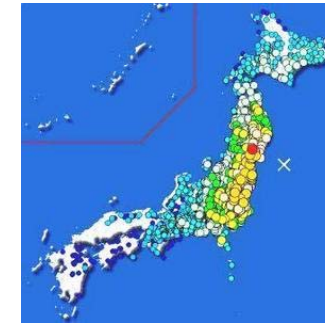
If 8cores are integrated on a chip,
Power is still 1/8 and
Performance becomes 2 times.

Parallel Soft is important for scalable performance of multicore

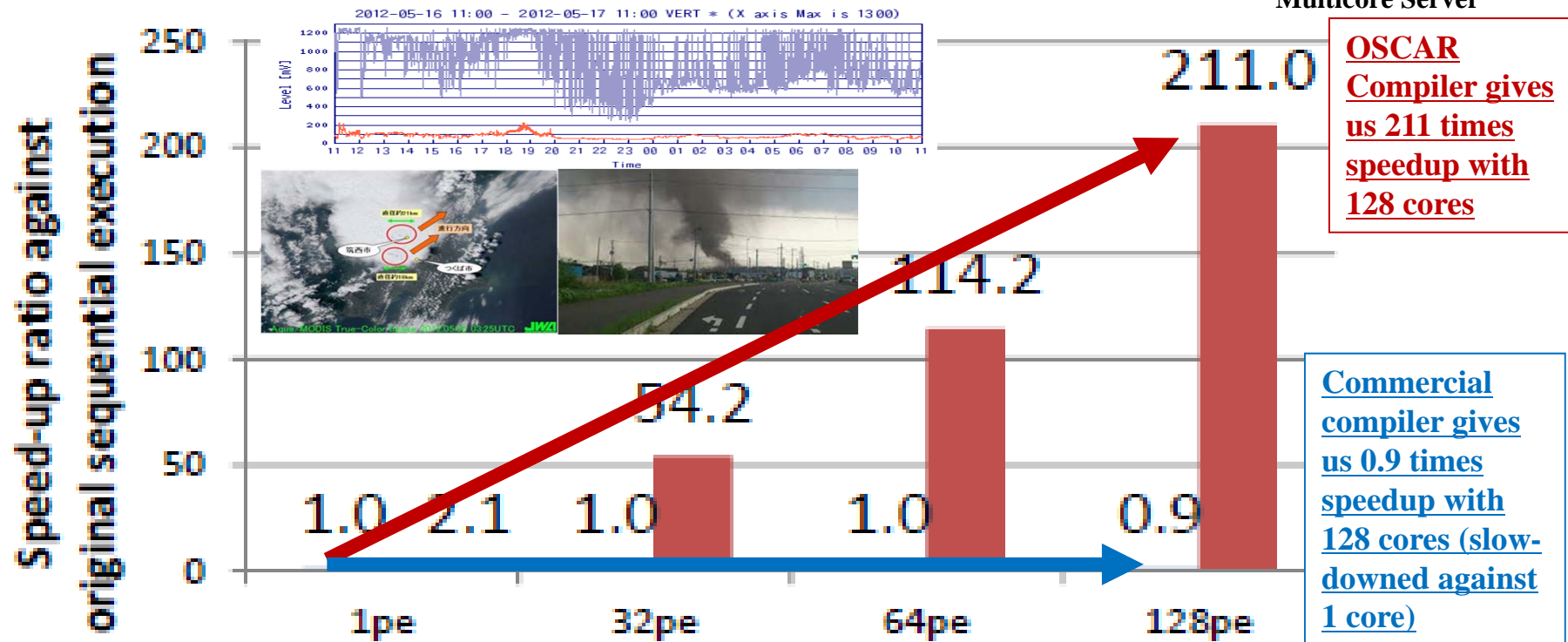
- Just more cores don't give us speedup
- Development cost and period of parallel software are getting a bottleneck of development of embedded systems, eg. IoT, Automobile

Earthquake wave propagation simulation GMS developed by National Research Institute for Earth Science and Disaster Resilience (NIED)

■ original (sun studio) ■ proposed method

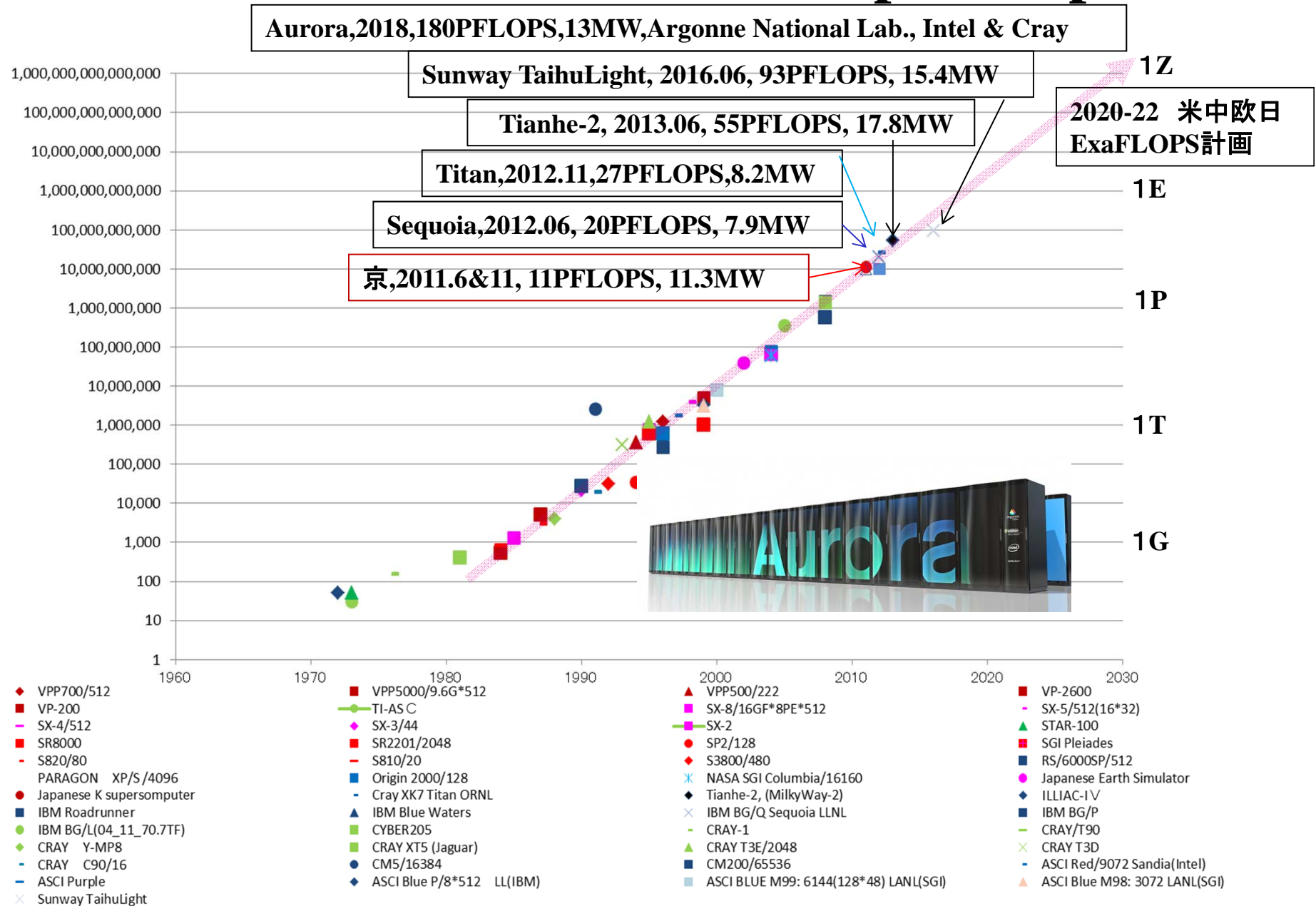


Fjitsu M9000 SPARC Multicore Server



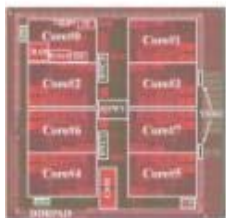
- Automatic parallelizing compiler available on the market gave us no speedup against execution time on 1 core on 64 cores
 - Execution time with 128 cores was slower than 1 core (0.9 times speedup)
- Advanced OSCAR parallelizing compiler gave us 211 times speedup with 128cores against execution time with 1 core using commercial compiler
 - OSCAR compiler gave us 2.1 times speedup on 1 core against commercial compiler by global cache optimization

Trend of Peak Performances of Supercomputers



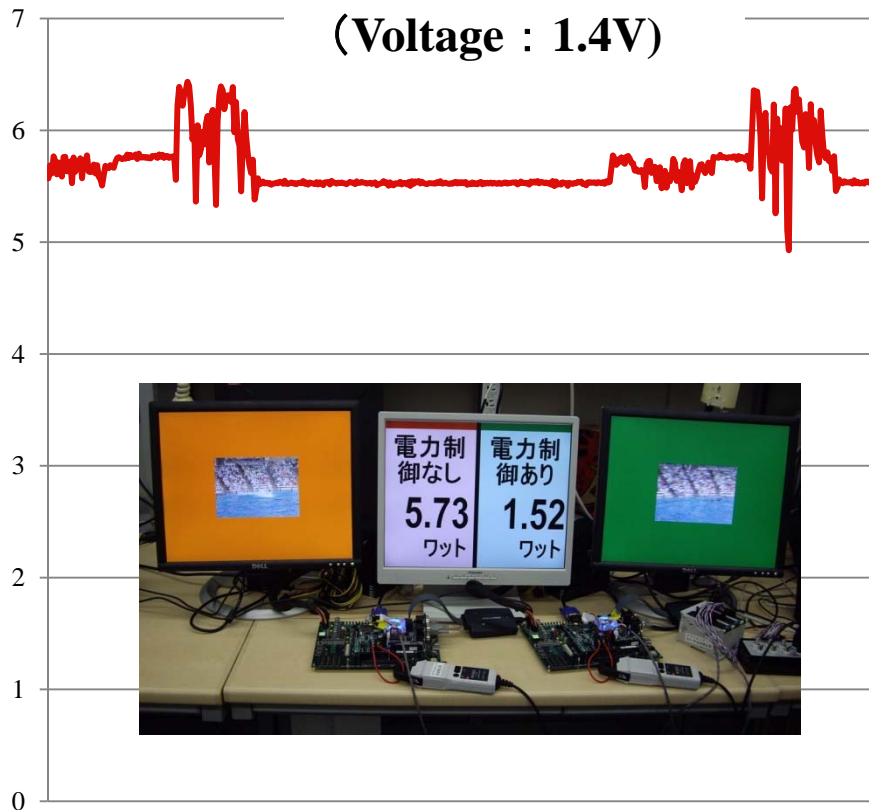
Power Reduction of MPEG2 Decoding to 1/4 on 8 Core Homogeneous Multicore RP-2 by OSCAR Parallelizing Compiler

MPEG2 Decoding with 8 CPU cores



Without Power
Control

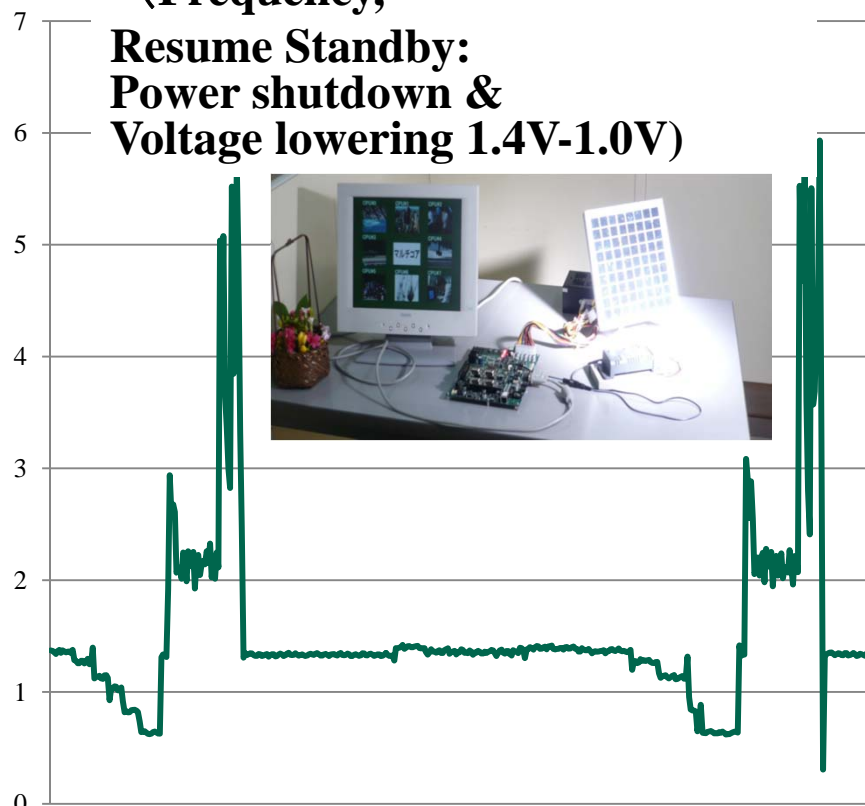
(Voltage : 1.4V)



Avg. Power
5.73 [W]

With Power Control
(Frequency,
Resume Standby:

Power shutdown &
Voltage lowering 1.4V-1.0V)

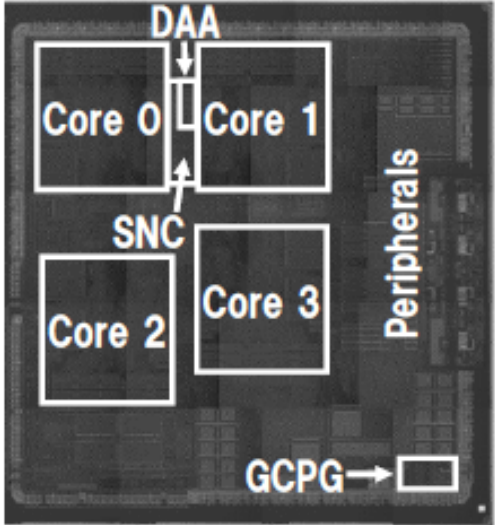
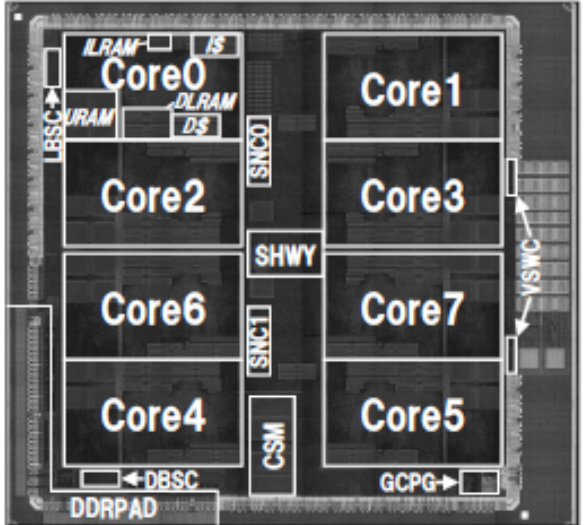
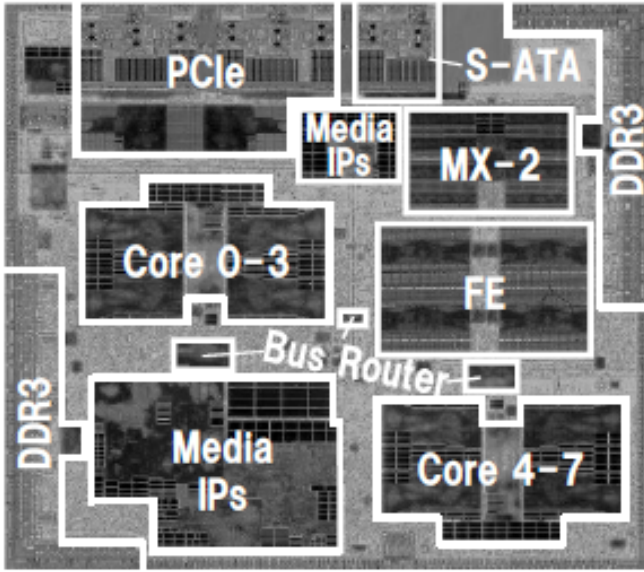


Avg. Power
1.52 [W]

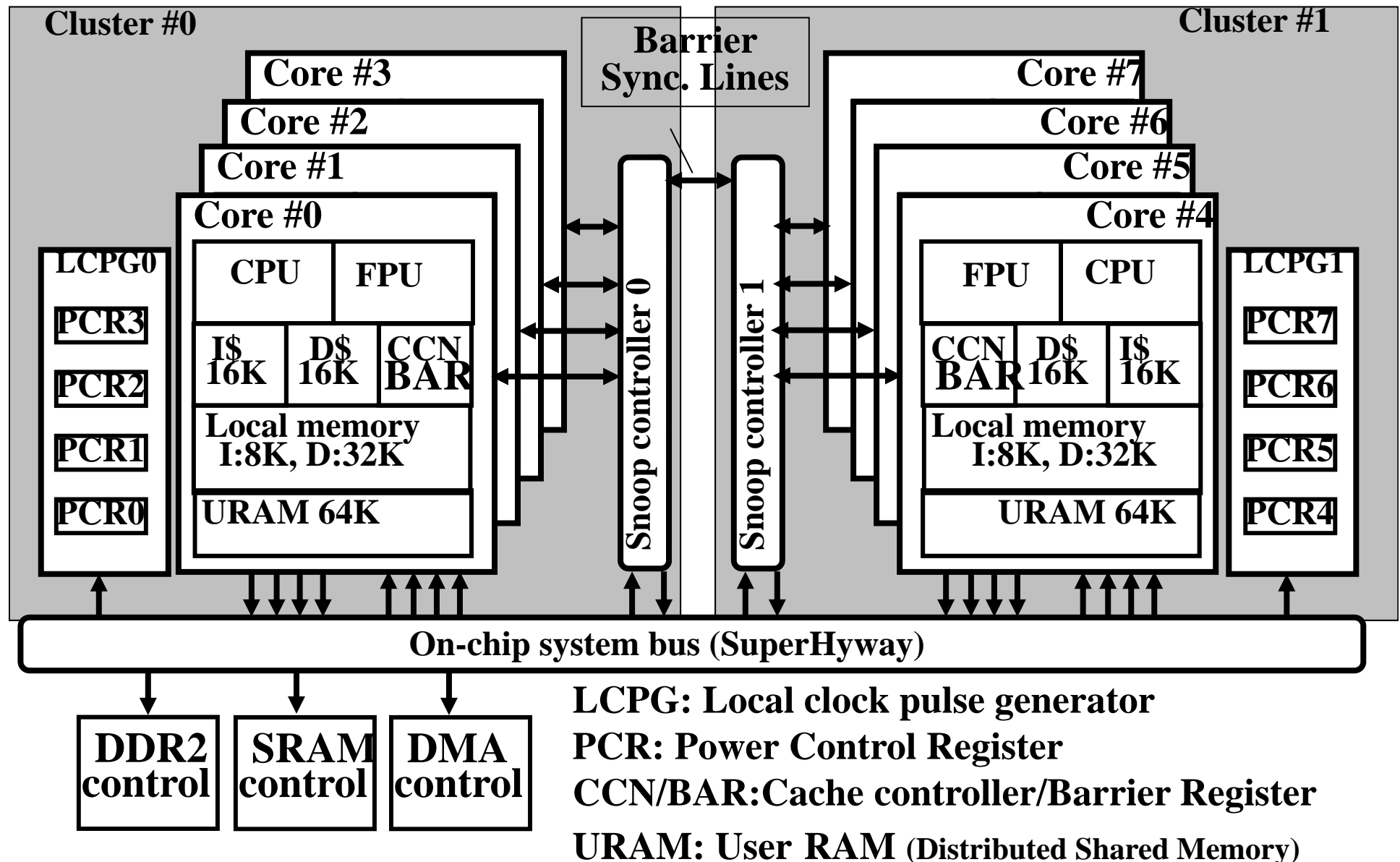
73.5% Power Reduction



4 core multicore RP1 (2007) , 8 core multicore RP2 (2008) and 15 core Heterogeneous multicore RPX (2010) developed in NEDO Projects with Hitachi and Renesas

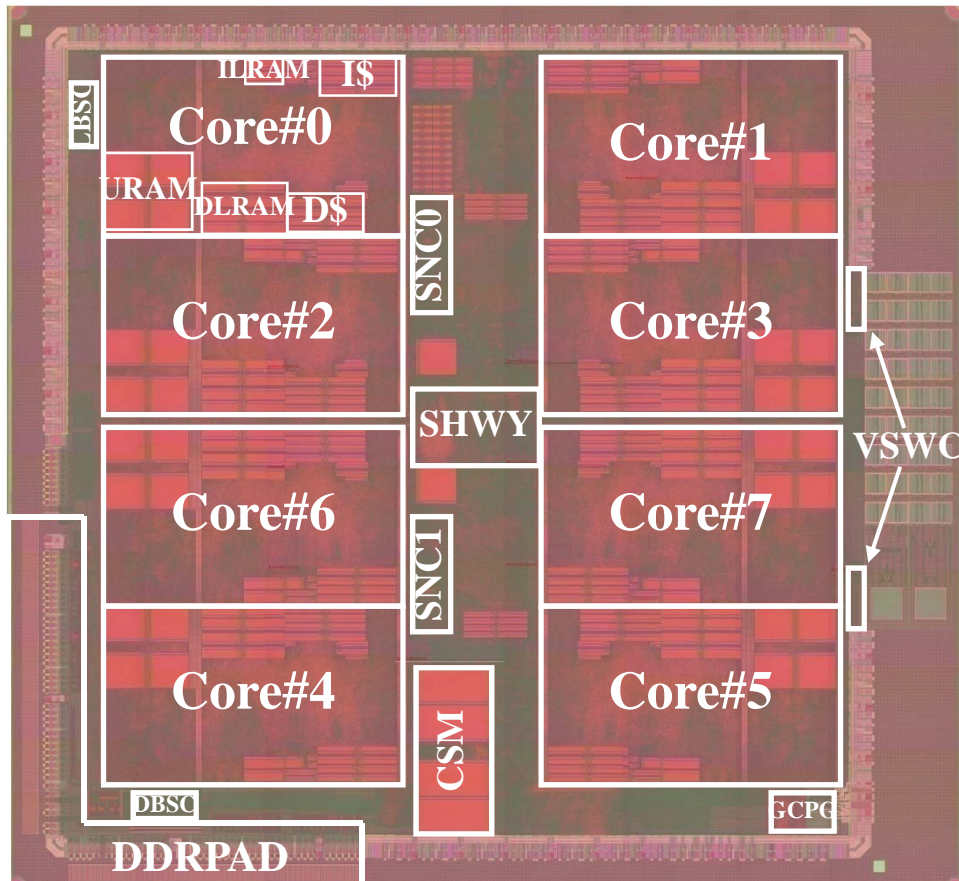
RP-1 (ISSCC2007 #5.3)	RP-2 (ISSCC2008 #4.5)	RP-X (ISSCC2010 #5.3)
		
90nm, 8-layer, triple-Vth, CMOS	90nm, 8-layer, triple-Vth, CMOS	45nm, 8-layer, triple-Vth, CMOS
97.6 mm ² (9.88 x 9.88 mm)	104.8 mm ² (10.61 x 9.88 mm)	153.8 mm ² (12.4 x 12.4 mm)
1.0V (internal), 1.8/3.3V (I/O)	1.0-1.4V (internal), 1.8/3.3V (I/O)	1.0-1.2V (internal), 1.2-3.3V (I/O)
600MHz ,4.32 GIPS,16.8 GFLOPS	600MHz , 8.64 GIPS, 33.6 GFLOPS	648MHz, 13.7GIPS, 115GOPS, 36.2GFLOPS
11.4 GOPS/W (32b換算)	18.3 GOPS/W (32b換算)	37.3 GOPS/W (32b換算)

Compiler Co-designed Multicore RP2



Renesas-Hitachi-Waseda Low Power 8 core RP2

Developed in 2007 in METI/NEDO project

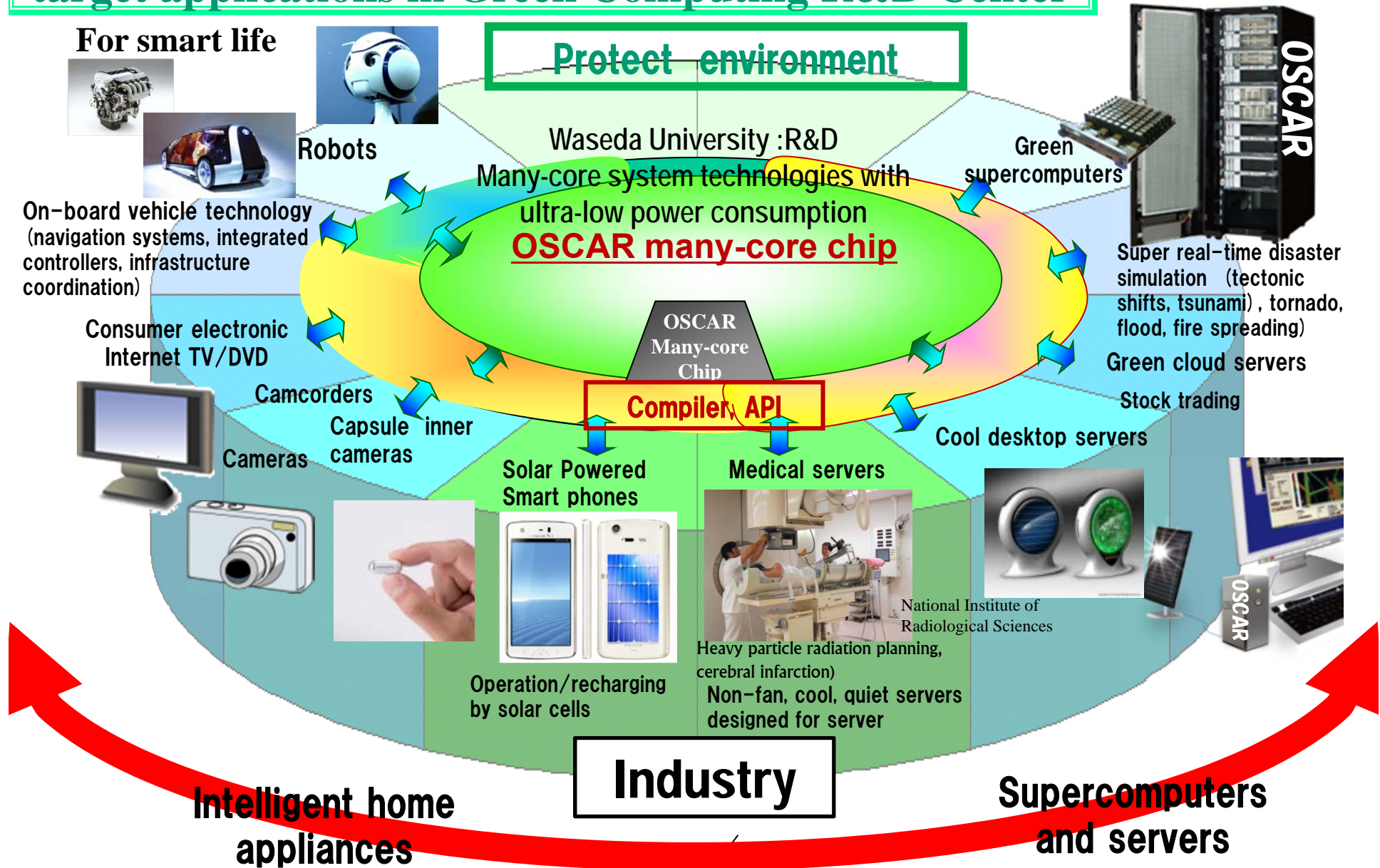


Process Technology	90nm, 8-layer, triple-Vth, CMOS
Chip Size	104.8mm ² (10.61mm x 9.88mm)
CPU Core Size	6.6mm ² (3.36mm x 1.96mm)
Supply Voltage	1.0V–1.4V (internal), 1.8/3.3V (I/O)
Power Domains	17 (8 CPUs, 8 URAMs, common)

IEEE ISSCC08: Paper No. 4.5, M.ITO, ... and H. Kasahara, “An 8640 MIPS SoC with Independent Power-off Control of 8 CPUs and 8 RAMs by an Automatic Parallelizing Compiler”

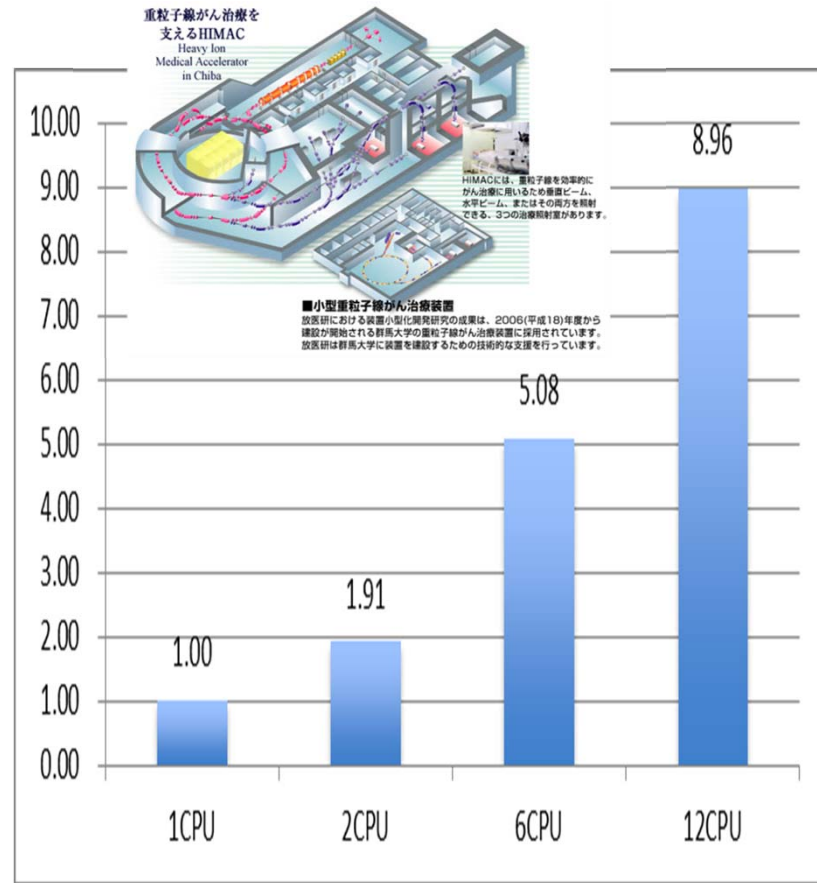
Industry-government-academia collaboration and target applications in Green Computing R&D Center

Protect lives



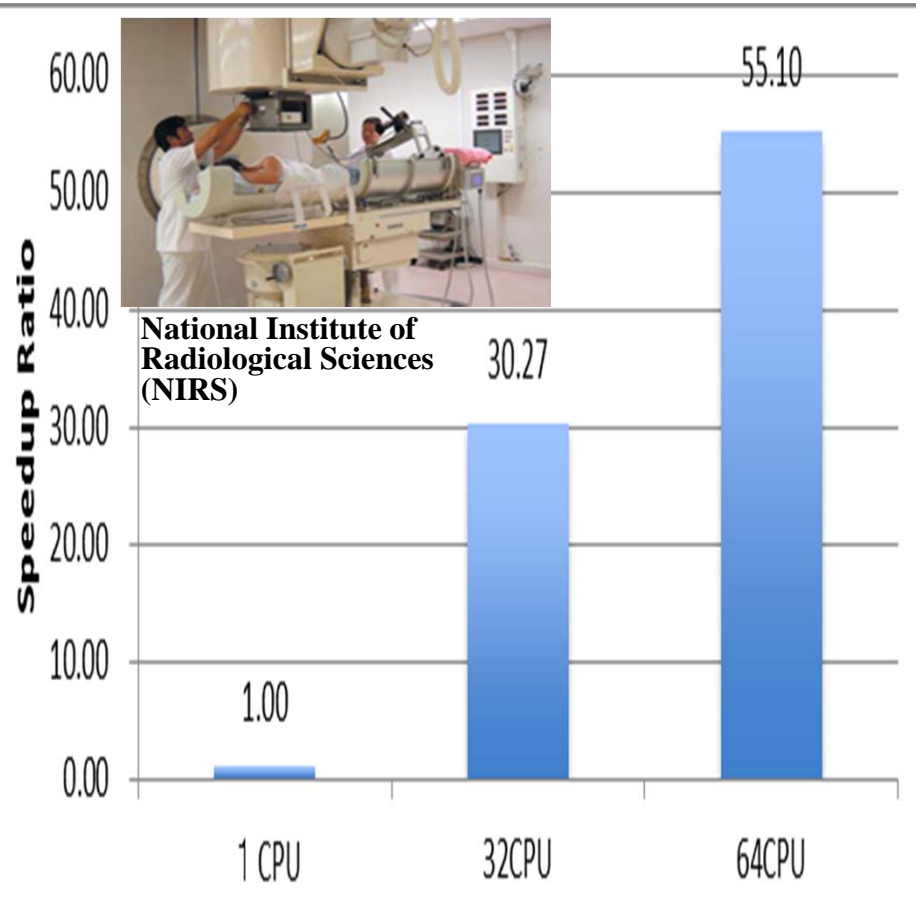
Cancer Treatment Carbon Ion Radiotherapy

(Previous best was 2.5 times speedup on 16 processors with hand optimization)



8.9times speedup by 12 processors

**Intel Xeon X5670 2.93GHz 12
core SMP (Hitachi HA8000)**



55 times speedup by 64 processors

**IBM Power 7 64 core SMP
(Hitachi SR16000)**

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

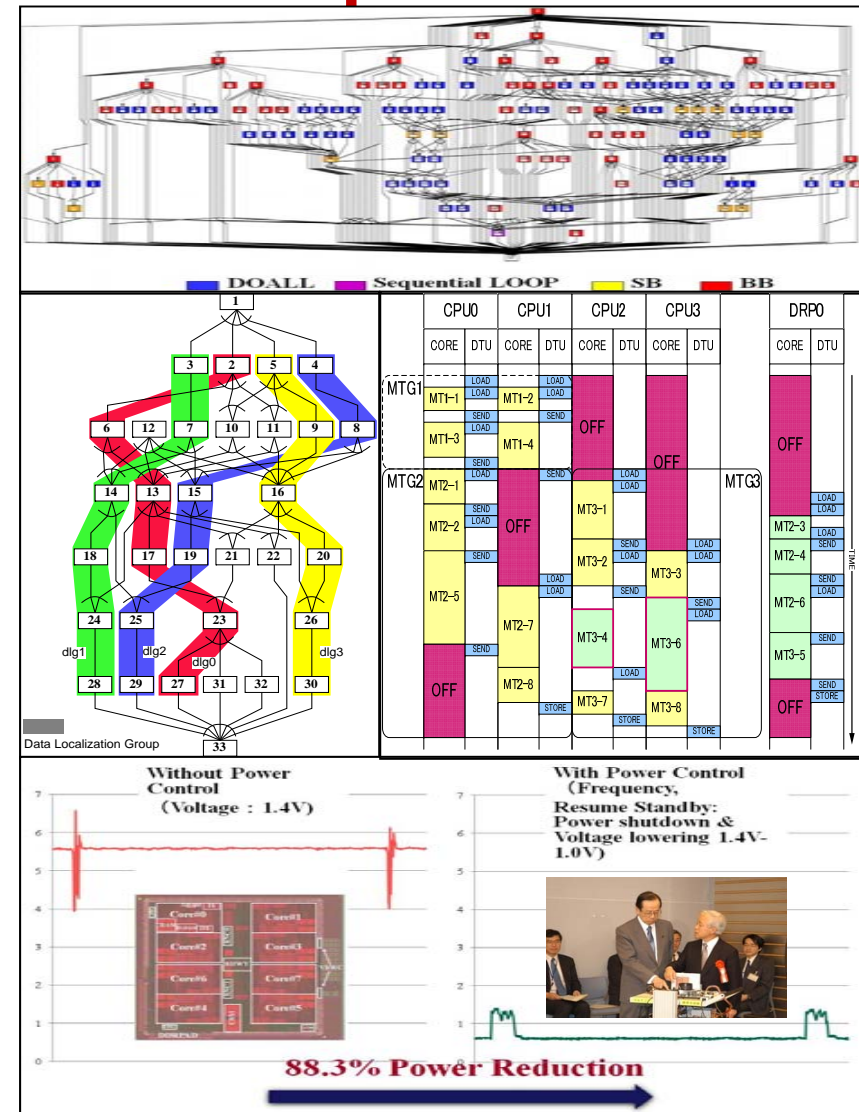
Automatic data management for distributed shared memory, cache and local memory

Data Transfer Overlapping

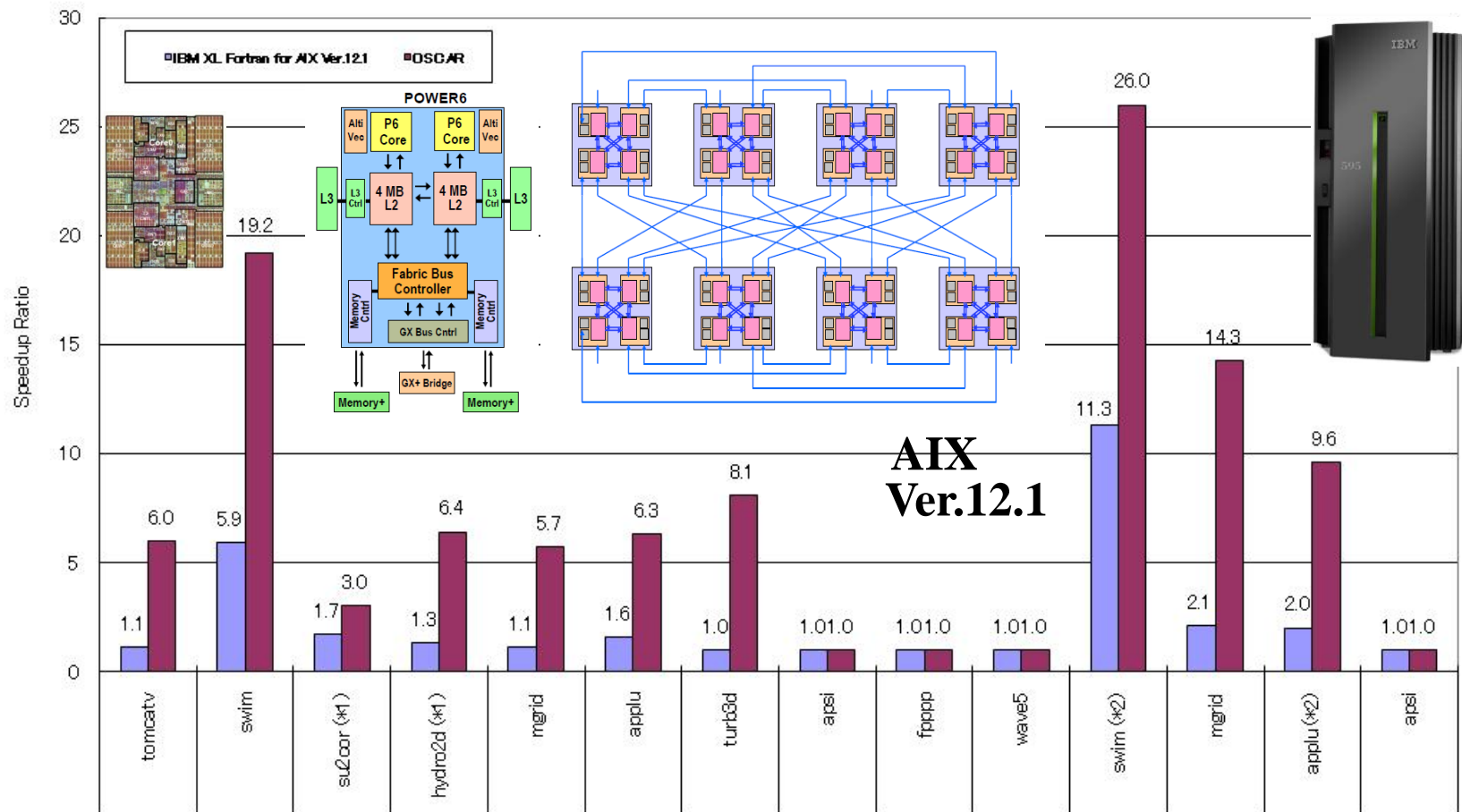
Data transfer overlapping using Data Transfer Controllers (DMAs)

Power Reduction

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



Performance of OSCAR Compiler on IBM p6 595 Power6 (4.2GHz) based 32-core SMP Server



Compi

(*1) Sequential: -O3 -qarch=pwr6, XLF: -O3 -qarch=pwr6 -qsmp=auto, OSCAR: -O3 -qarch=pwr6 -qsmp=noauto

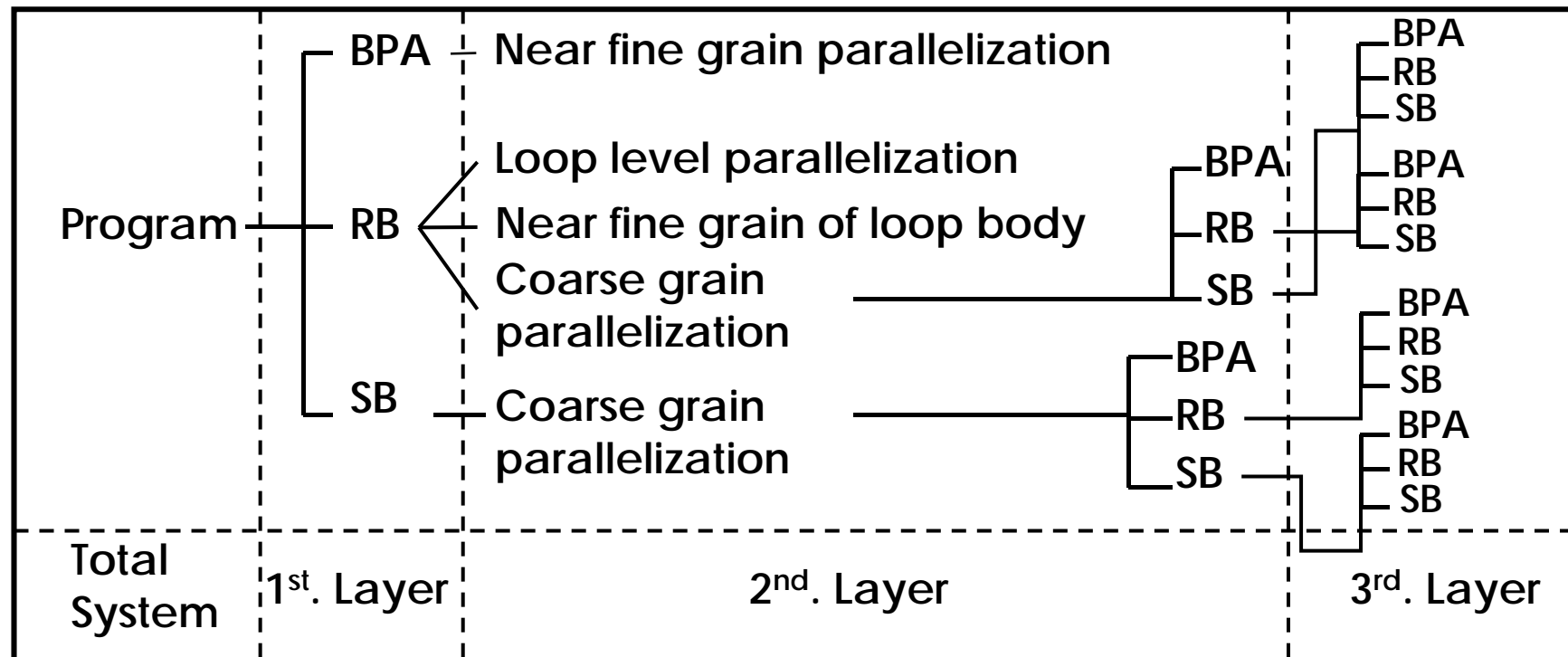
(*2) Sequential: -O5 -q64 -qarch=pwr6, XLF: -O5 -q64 -qarch=pwr6 -qsmp=auto, OSCAR: -O5 -q64 -qarch=pwr6 -qsmp=noauto

(Others) Sequential: -O5 -qarch=pwr6, XLF: -O5 -qarch=pwr6 -qsmp=auto, OSCAR: -O5 -qarch=pwr6 -qsmp=noauto

Generation of Coarse Grain Tasks

■ Macro-tasks (MTs)

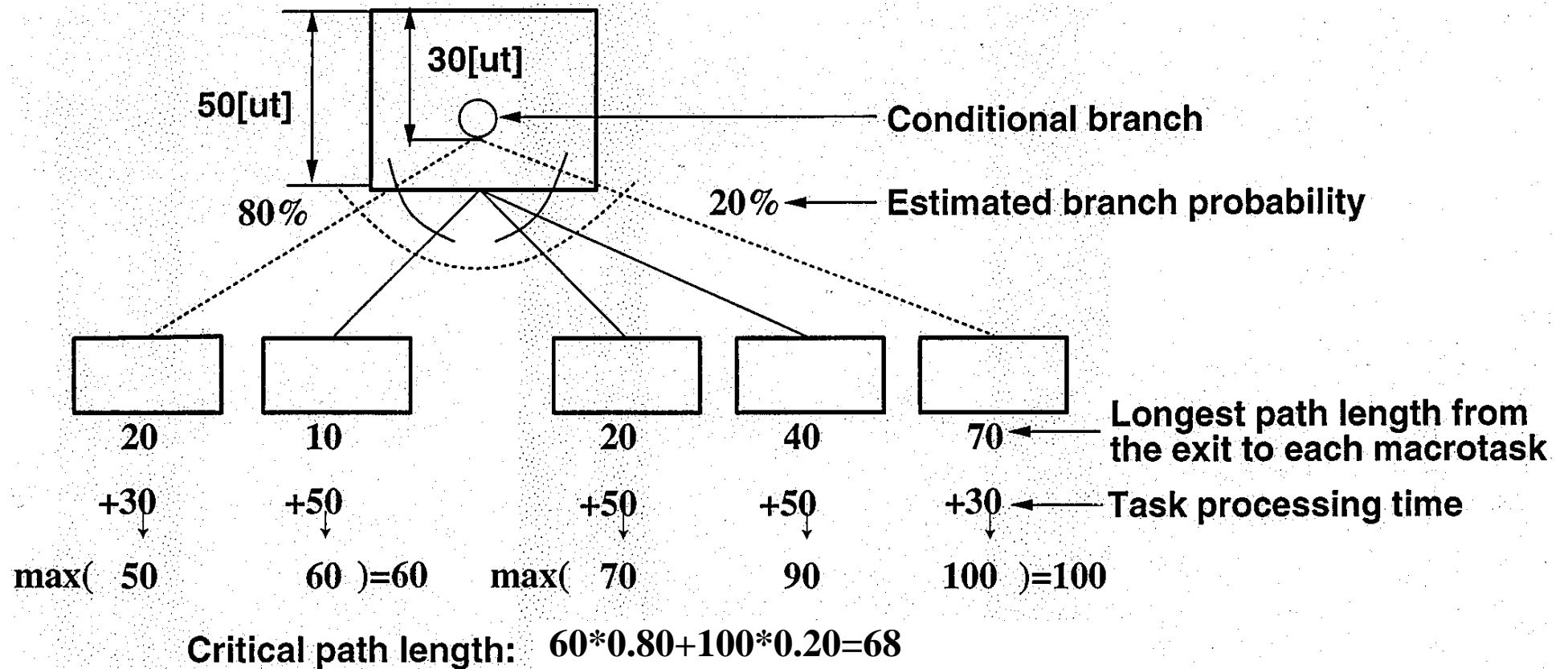
- **Block of Pseudo Assignments (BPA): Basic Block (BB)**
- **Repetition Block (RB) : natural loop**
- **Subroutine Block (SB): subroutine**



Grain Tasks (Macro-tasks)



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

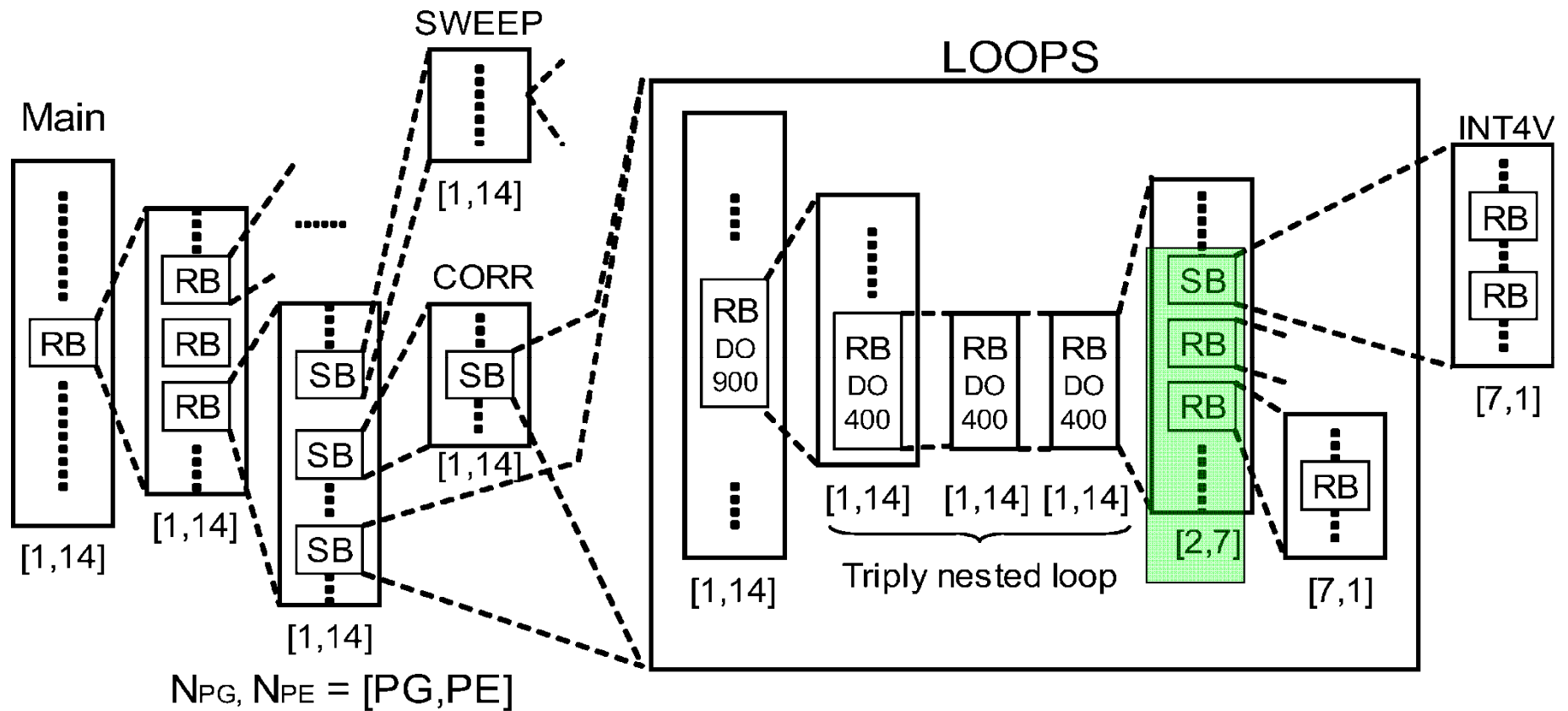
MT3 may start execution after MT1 branches to MT3.

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

Automatic processor assignment in 103.su2cor

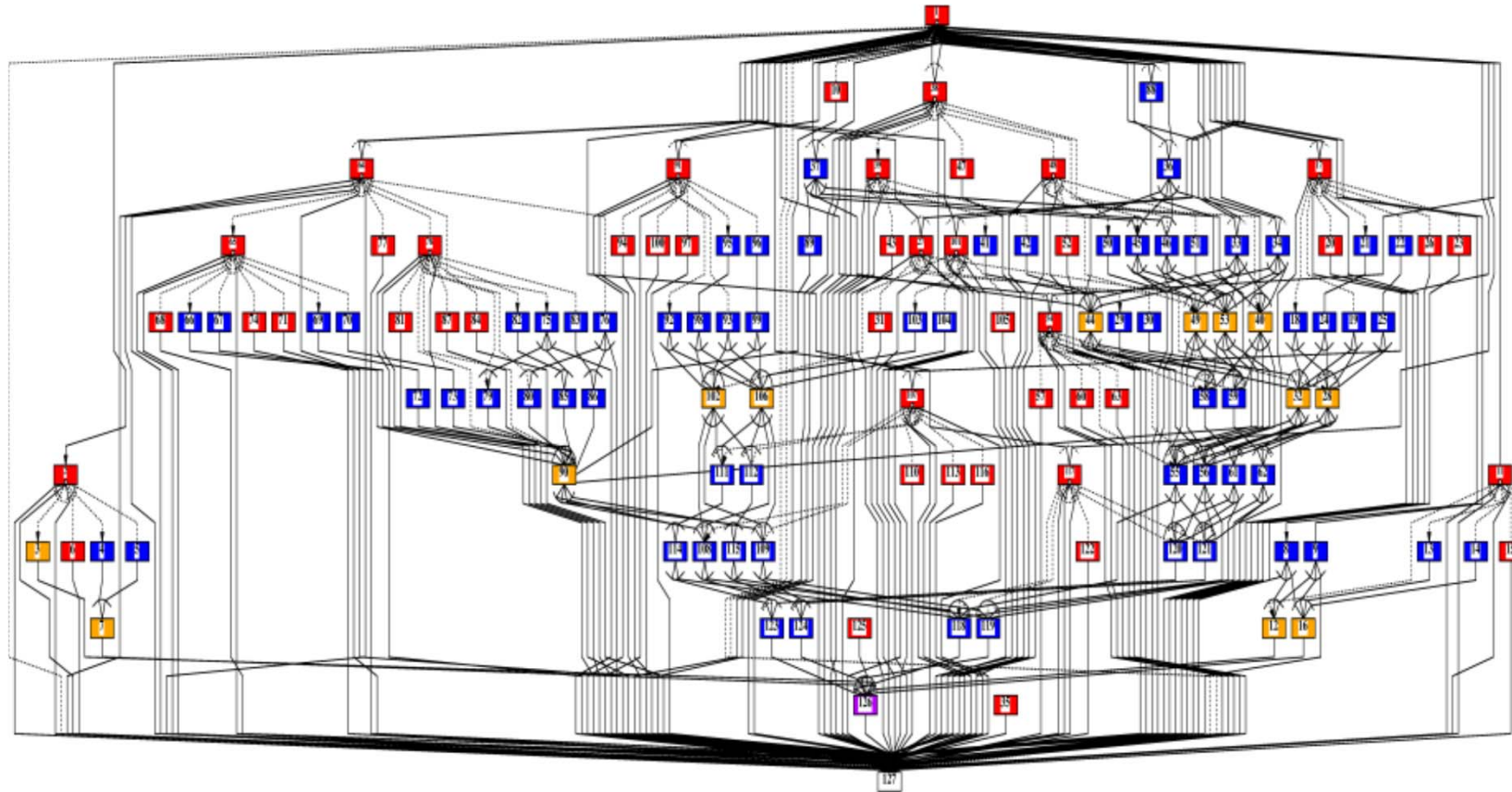
- **Using 14 processors**

Coarse grain parallelization within DO400



MTG of Su2cor-LOOPS-DO400

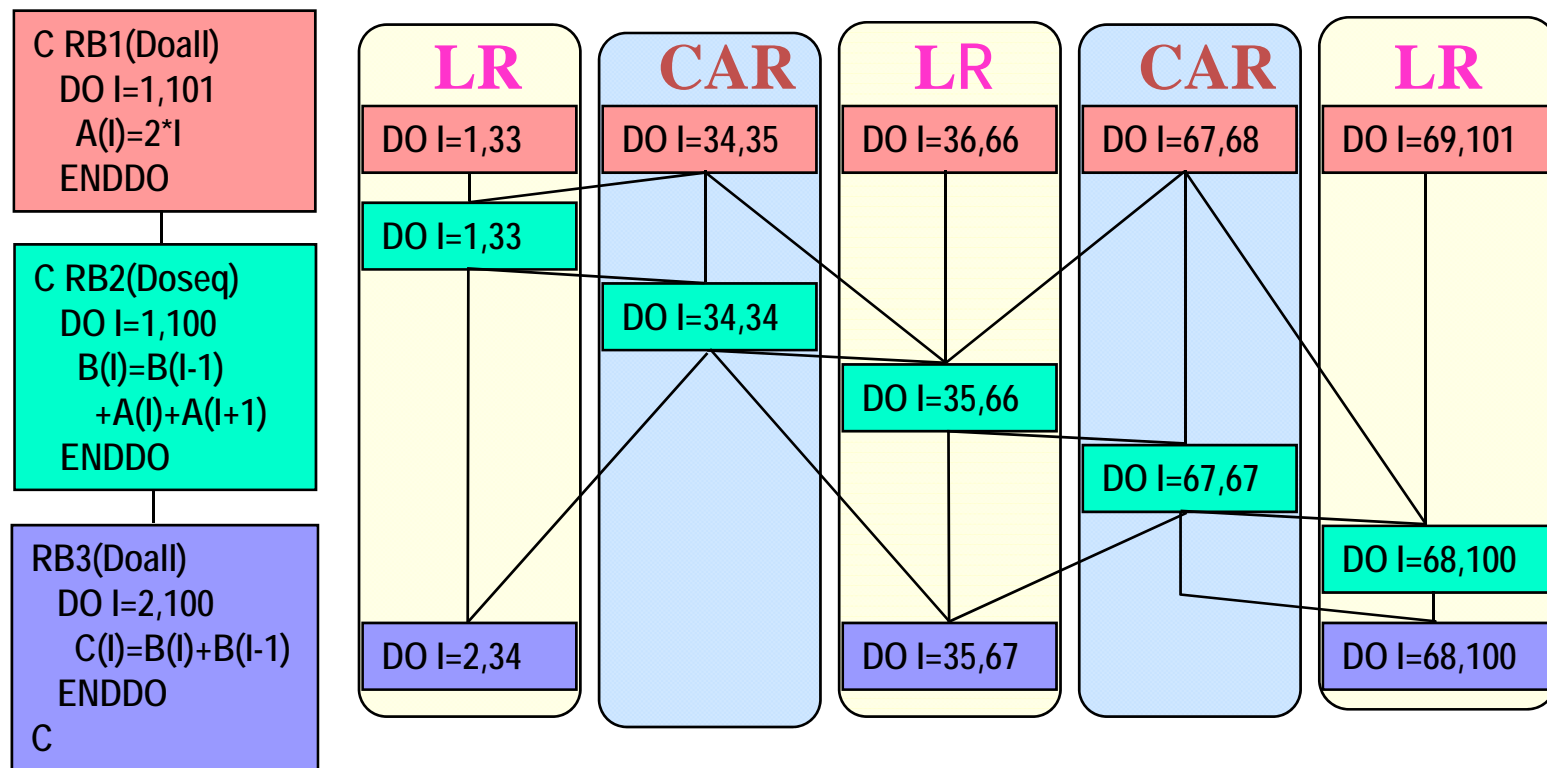
- Coarse grain parallelism $\text{PARA_ALD} = 4.3$



■ DOALL ■ Sequential LOOP ■ SB ■ BB

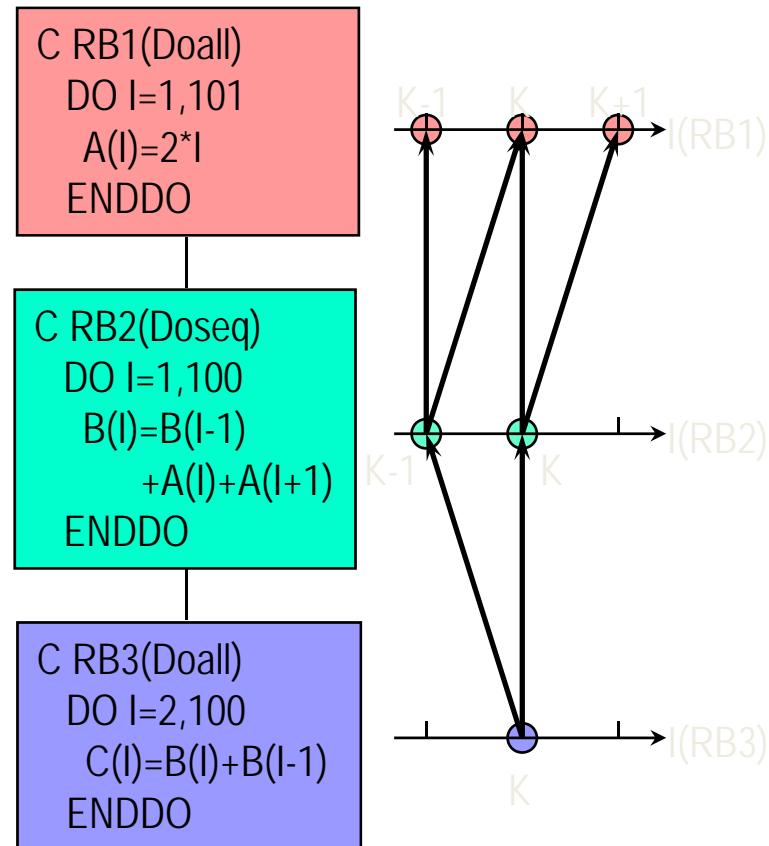
Data-Localization: Loop Aligned Decomposition

- 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



Inter-loop data dependence analysis in TLG

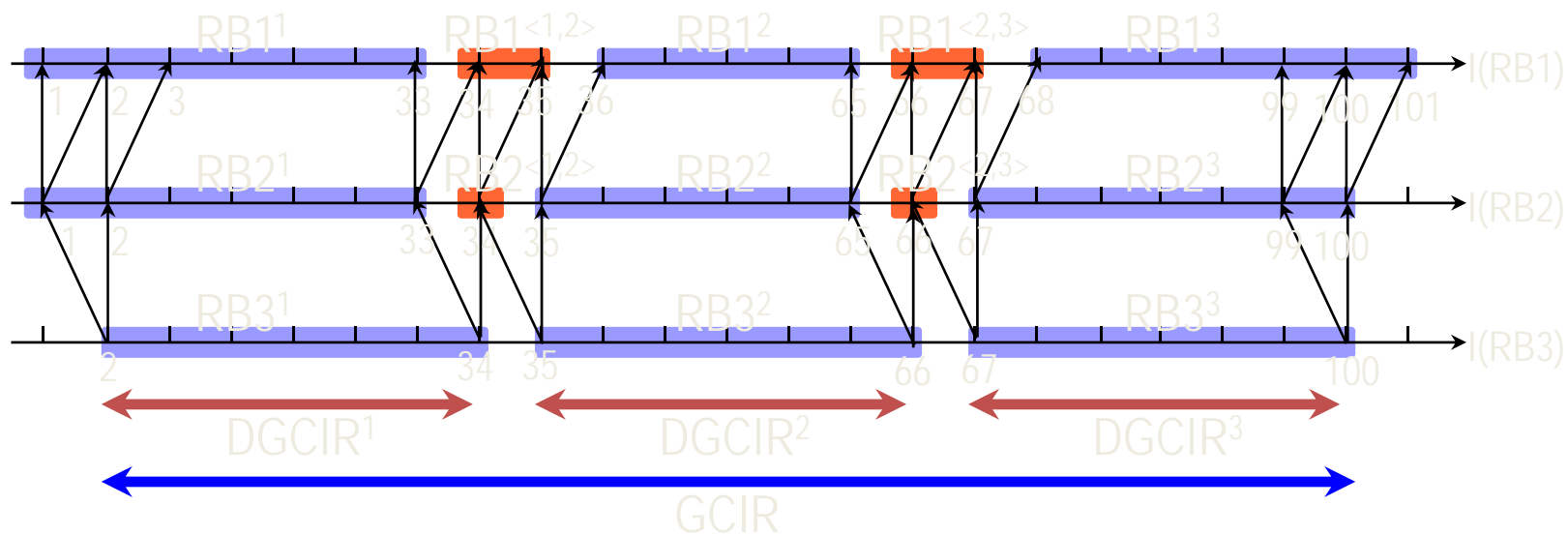
- Define exit-RB in TLG as Standard-Loop
- Find iterations on which a iteration of Standard-Loop is data dependent
 - e.g. K_{th} of RB3 is data-dep on $K-1_{th}, K_{th}$ of RB2, on $K-1_{th}, K_{th}, K+1_{th}$ of RB1 **indirectly**.



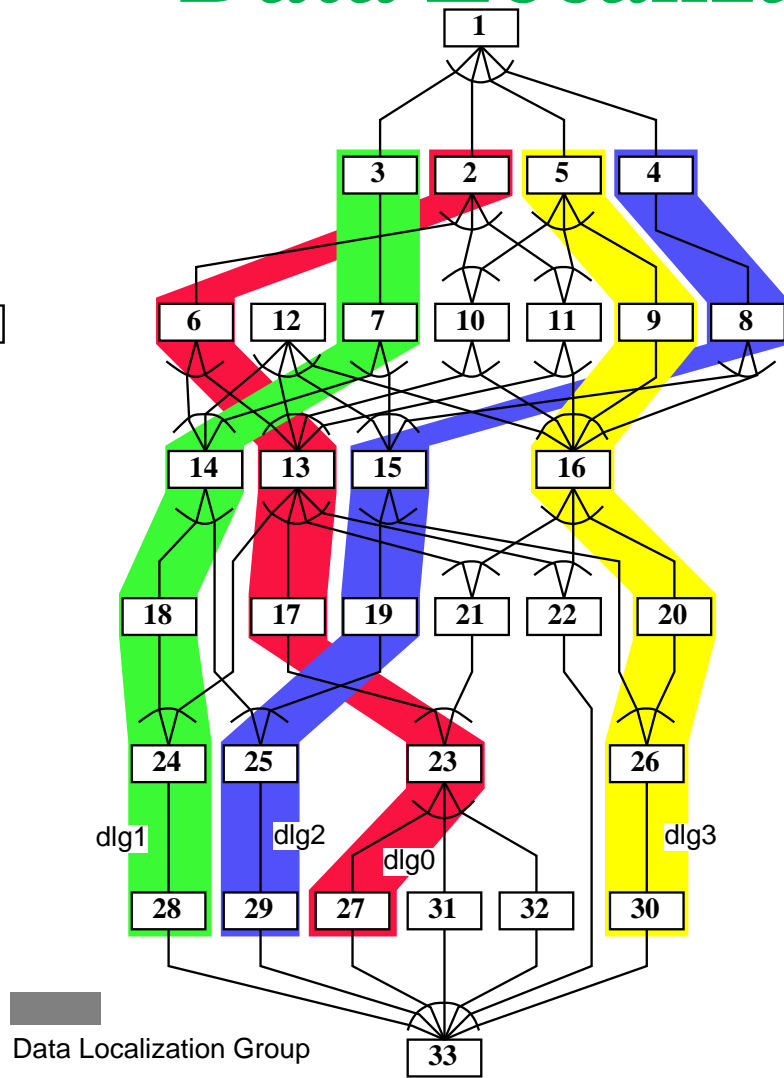
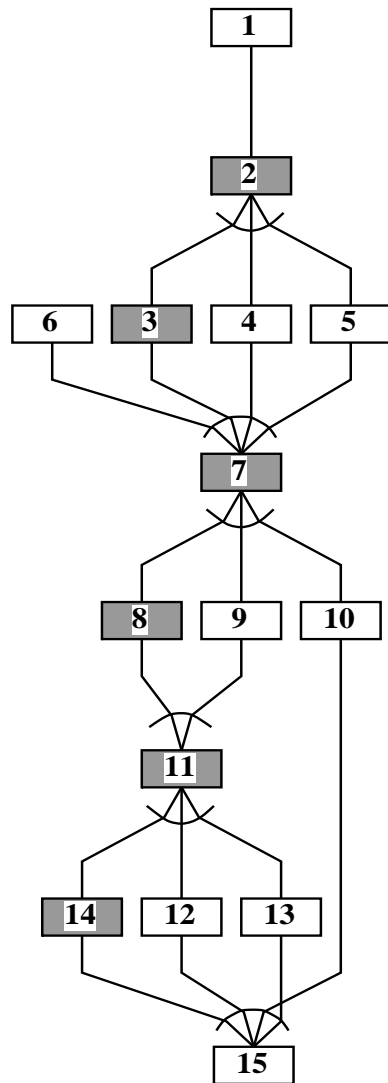
Example of TLG

Decomposition of RBs in TLG

- Decompose GCIR into $DGCIR^p (1 \leq p \leq n)$
 - n: (multiple) num of PCs, DGCIR: Decomposed GCIR
- Generate CAR on which $DGCIR^p \& DGCIR^{p+1}$ are data-dep.
- Generate LR on which $DGCIR^p$ is data-dep.



Data Localization



PE0

12
2
6
4
8
15
19
25
29
13
17
22
21
23
27

PE1

1
3
7
14
18
5
9
11
10
16
20
26
30
24
28
32
31

A schedule for two processors

An Example of Data Localization for Spec95 Swim

```

DO 200 J=1,N
DO 200 I=1,M
  UNEW(I+1,J) = UOLD(I+1,J)+
1  TDT8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV(I,J+1)+CV(I,J)
2  +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
  VNEW(I,J+1) = VOLD(I,J+1)-TDT8*(Z(I+1,J+1)+Z(I,J+1))
1  *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
2  -TDTSDY*(H(I,J+1)-H(I,J))
  PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
1  -TDTSDY*(CV(I,J+1)-CV(I,J))
200 CONTINUE

```

```

DO 210 J=1,N
  UNEW(1,J) = UNEW(M+1,J)
  VNEW(M+1,J+1) = VNEW(1,J+1)
  PNEW(M+1,J) = PNEW(1,J)
210 CONTINUE

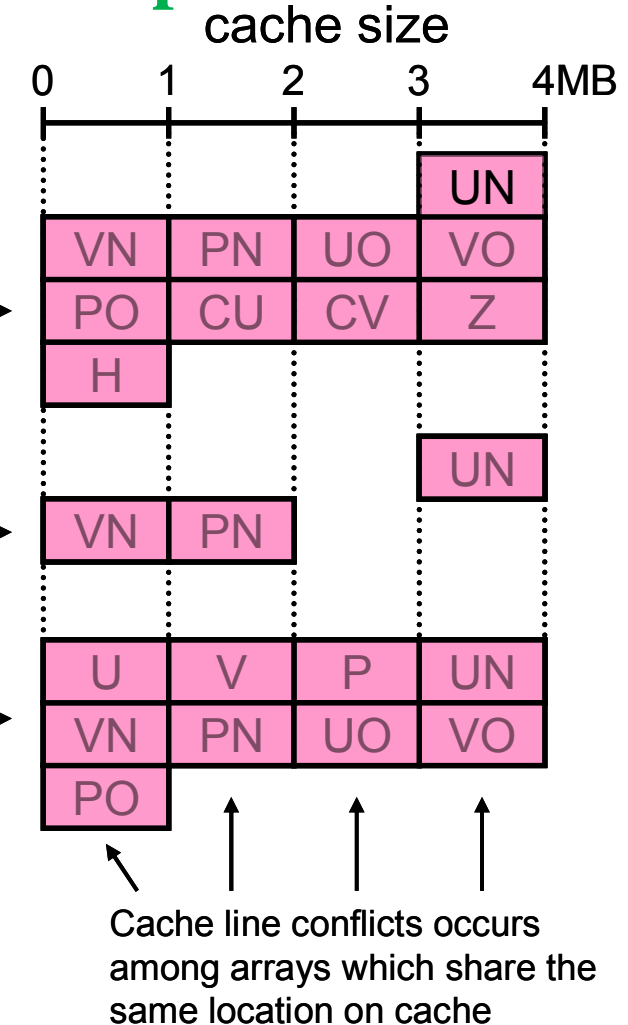
```

```

DO 300 J=1,N
DO 300 I=1,M
  UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
  VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
  POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
300 CONTINUE

```

(a) An example of target loop group for data localization



(b) Image of alignment of arrays on cache accessed by target loops

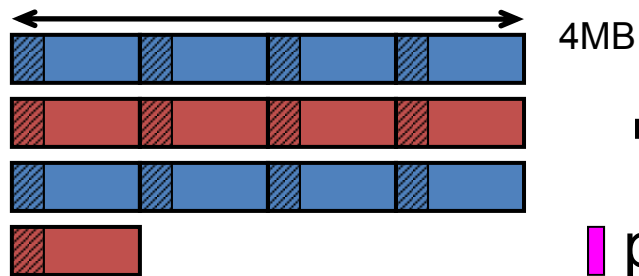
Data Layout for Removing Line Conflict Misses by Array Dimension Padding

Declaration part of arrays in spec95 swim

before padding

PARAMETER (N1=513, N2=513)

COMMON U(N1,N2), V(N1,N2), P(N1,N2),
* UNEW(N1,N2), VNEW(N1,N2),
1 PNEW(N1,N2), UOLD(N1,N2),
* VOLD(N1,N2), POLD(N1,N2),
2 CU(N1,N2), CV(N1,N2),
* Z(N1,N2), H(N1,N2)

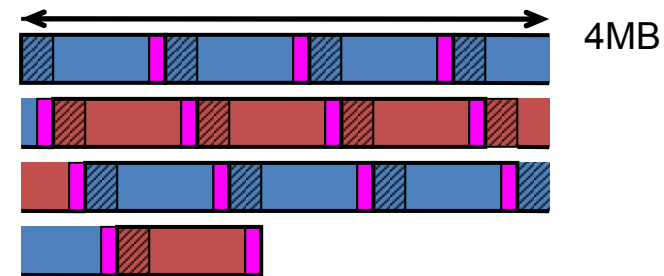


Box: Access range of DLG0

after padding

PARAMETER (N1=513, N2=544)

COMMON U(N1,N2), V(N1,N2), P(N1,N2),
* UNEW(N1,N2), VNEW(N1,N2),
1 PNEW(N1,N2), UOLD(N1,N2),
* VOLD(N1,N2), POLD(N1,N2),
2 CU(N1,N2), CV(N1,N2),
* Z(N1,N2), H(N1,N2)



Statement Level Near Fine Grain Task

<<LU Decomposition>>

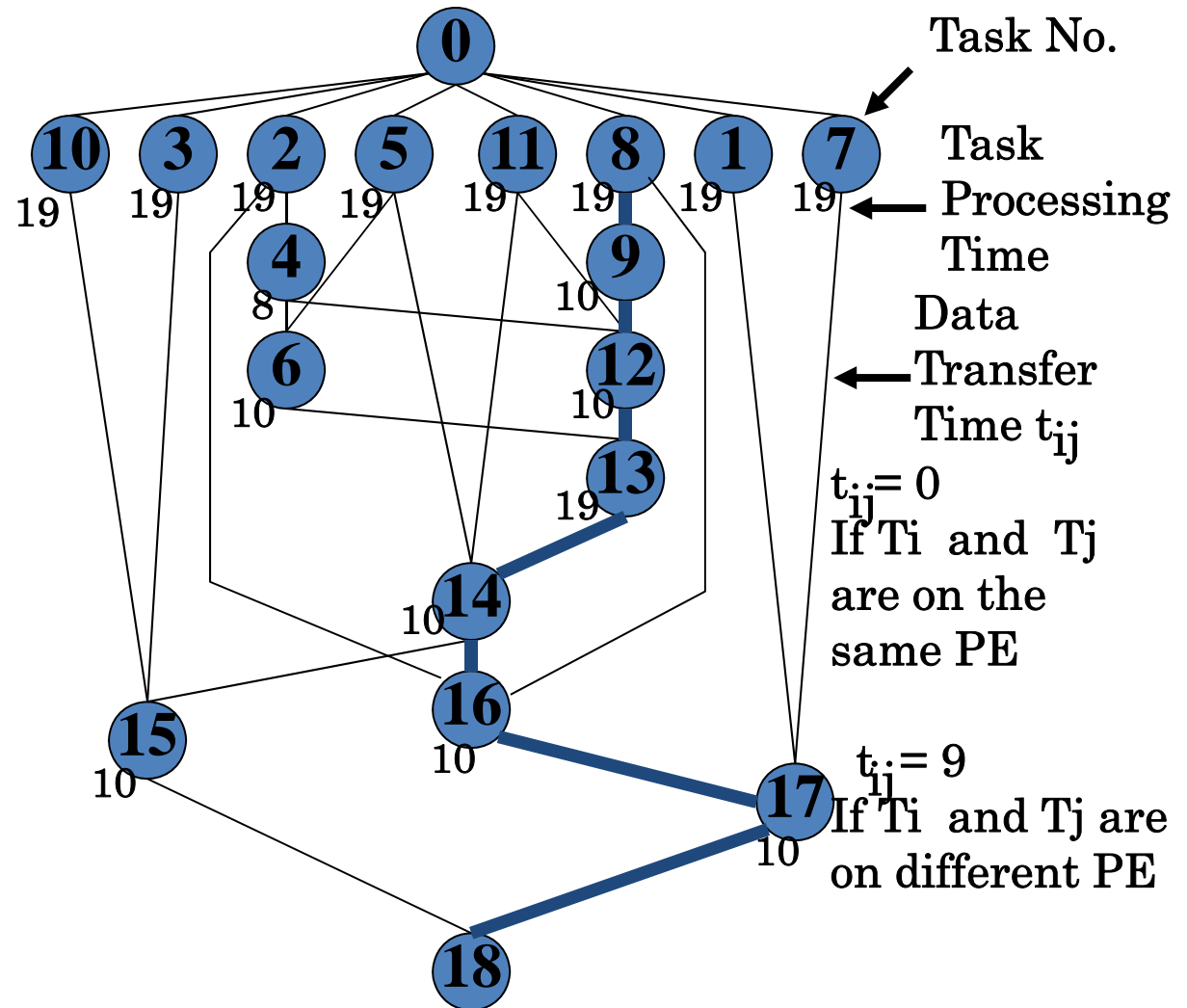
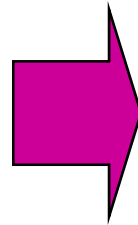
- 1) $u_{12} = a_{12}/l_{11}$
- 2) $u_{24} = a_{24}/l_{22}$
- 3) $u_{34} = a_{34}/l_{33}$
- 4) $l_{54} = -l_{52} * u_{24}$
- 5) $u_{45} = a_{45}/l_{44}$
- 6) $l_{55} = u_{55} - l_{54} * u_{45}$

<<Forward Substitution>>

- 7) $y_1 = b_1 / l_{11}$
- 8) $y_2 = b_2 / l_{22}$
- 9) $b_5 = b_5 - l_{52} * y_2$
- 10) $y_3 = b_3 / l_{33}$
- 11) $y_4 = b_4 / l_{44}$
- 12) $b_5 = b_5 - l_{54} * y_4$
- 13) $y_5 = b_5 / l_{55}$

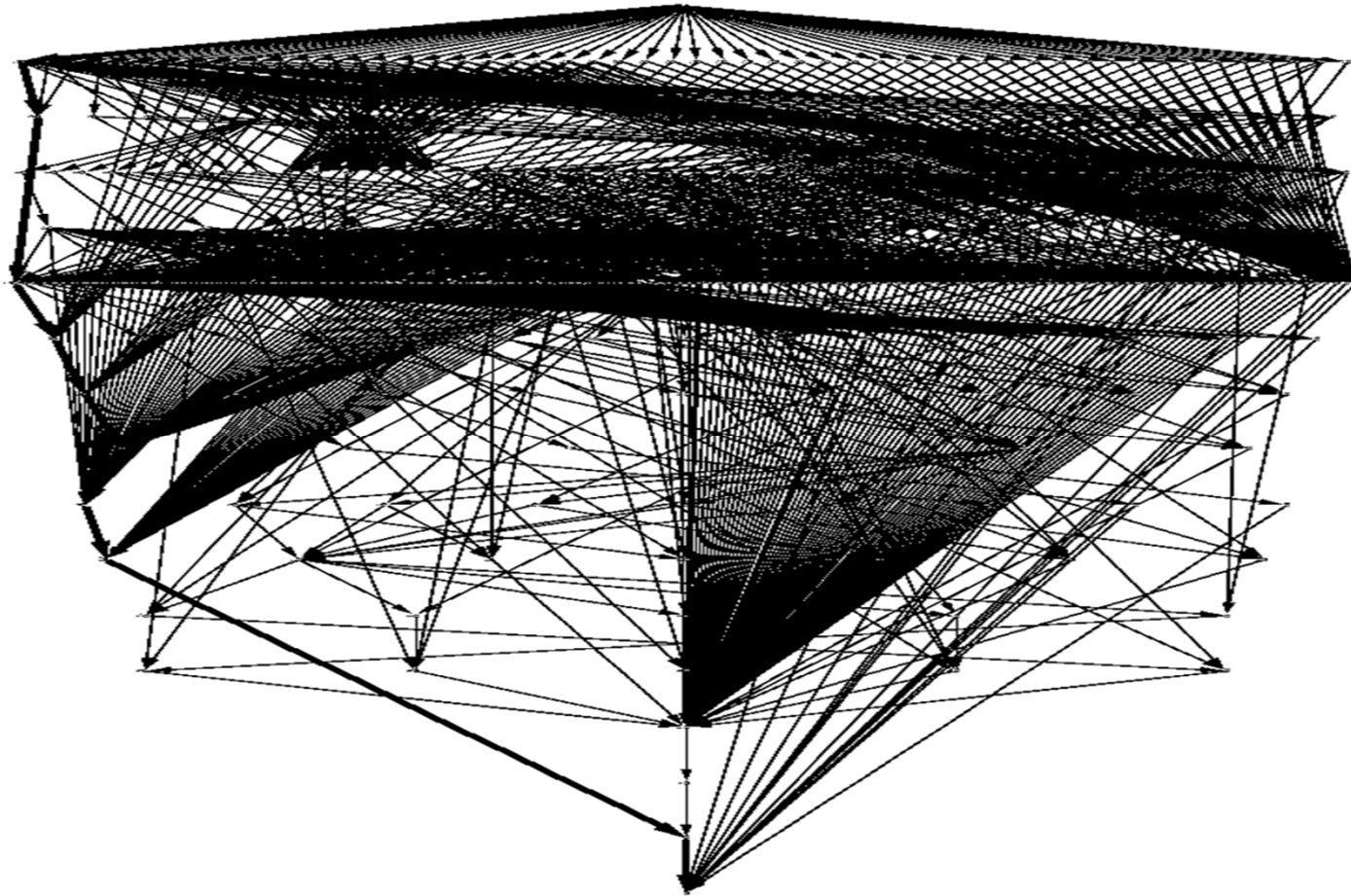
<<Backward Substitution>>

- 14) $x_4 = y_4 - u_{45} * y_5$
- 15) $x_3 = y_3 - u_{34} * x_4$
- 16) $x_2 = y_2 - u_{24} * x_4$
- 17) $x_1 = y_1 - u_{12} * x_2$

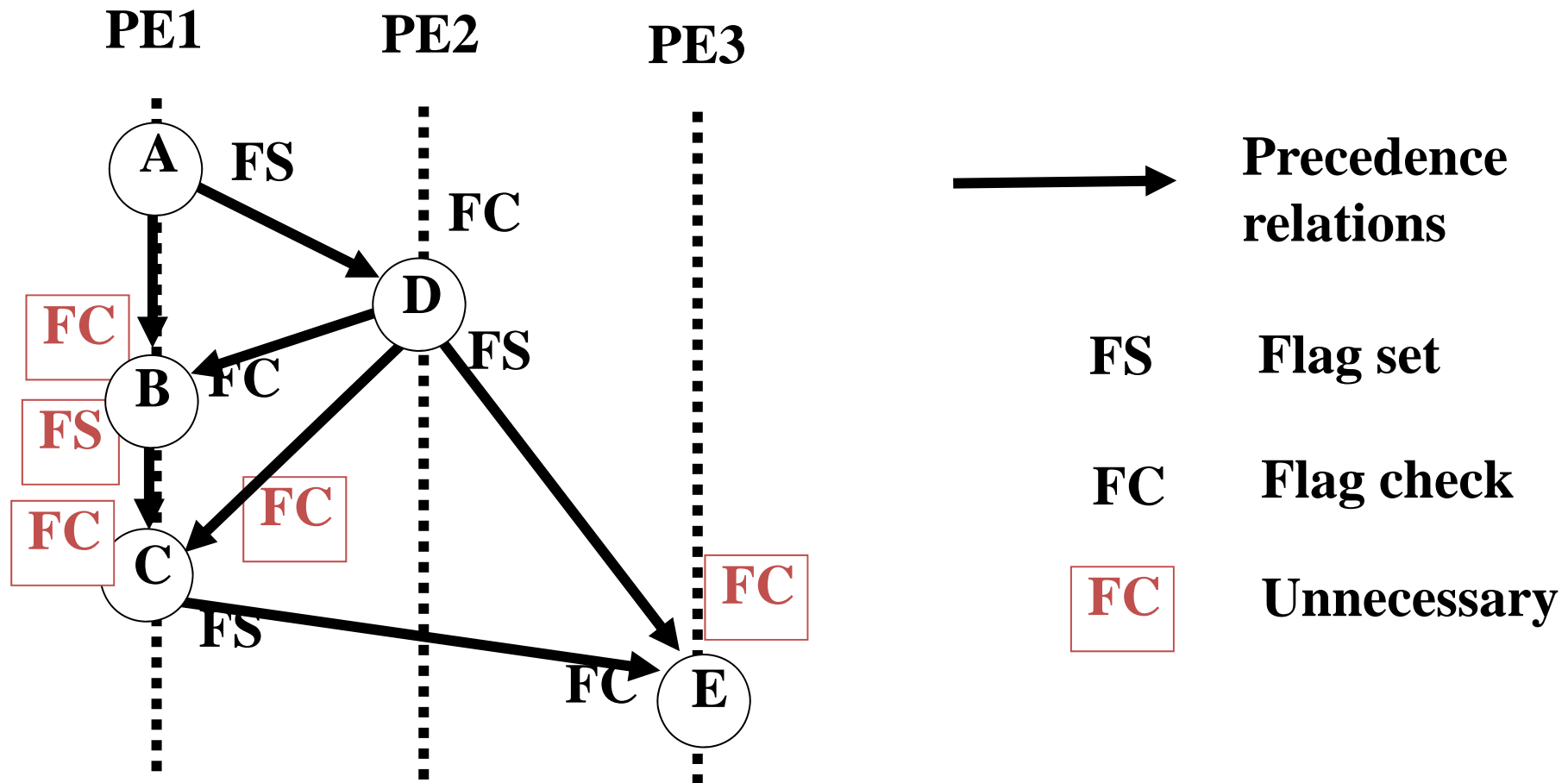


Task Graph for FPPPP

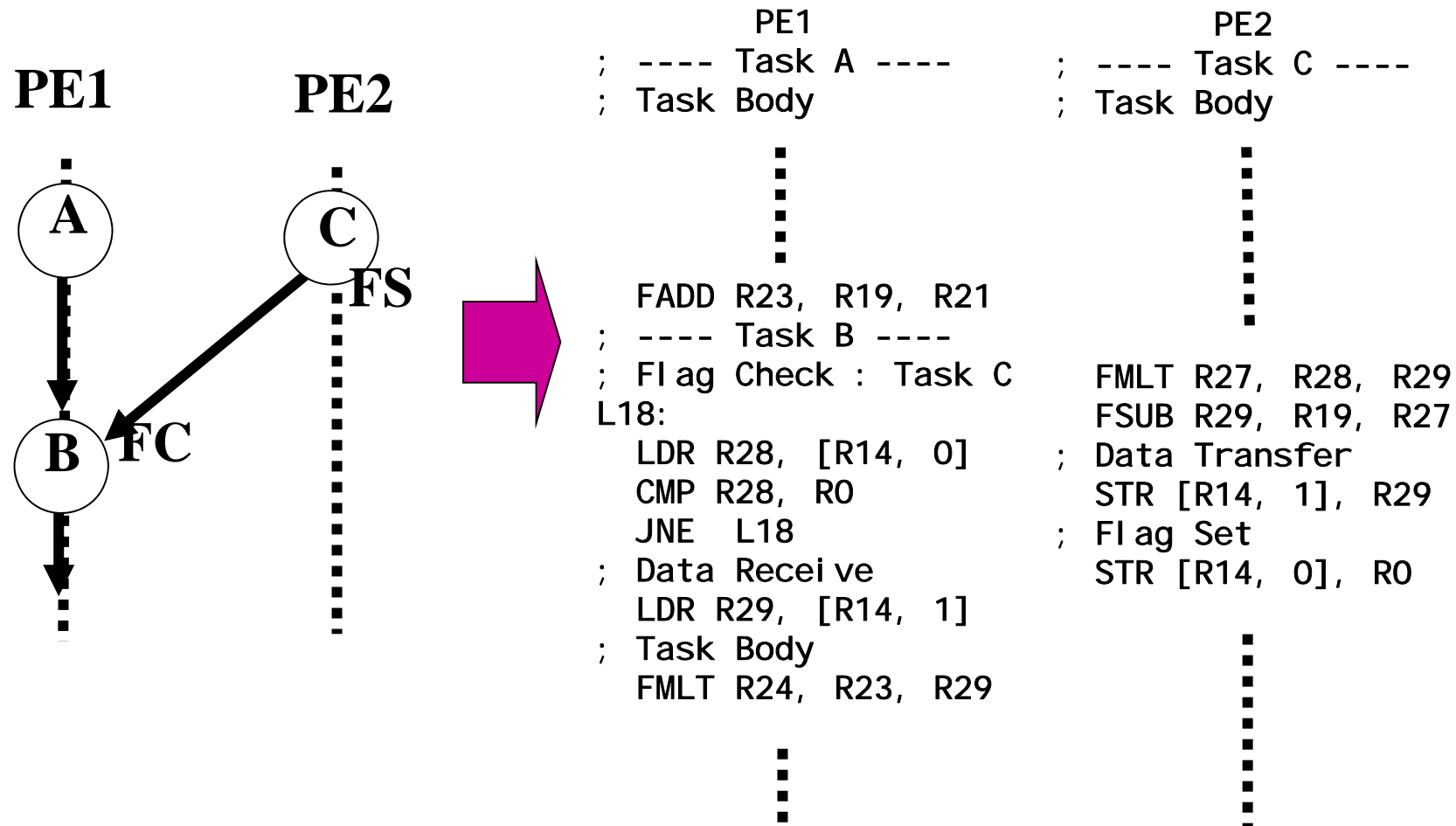
Statement level near fine grain parallelism



Elimination of Redundant Synchronization for Shared Data on Centralized Shared Memory after Static Task Scheduling



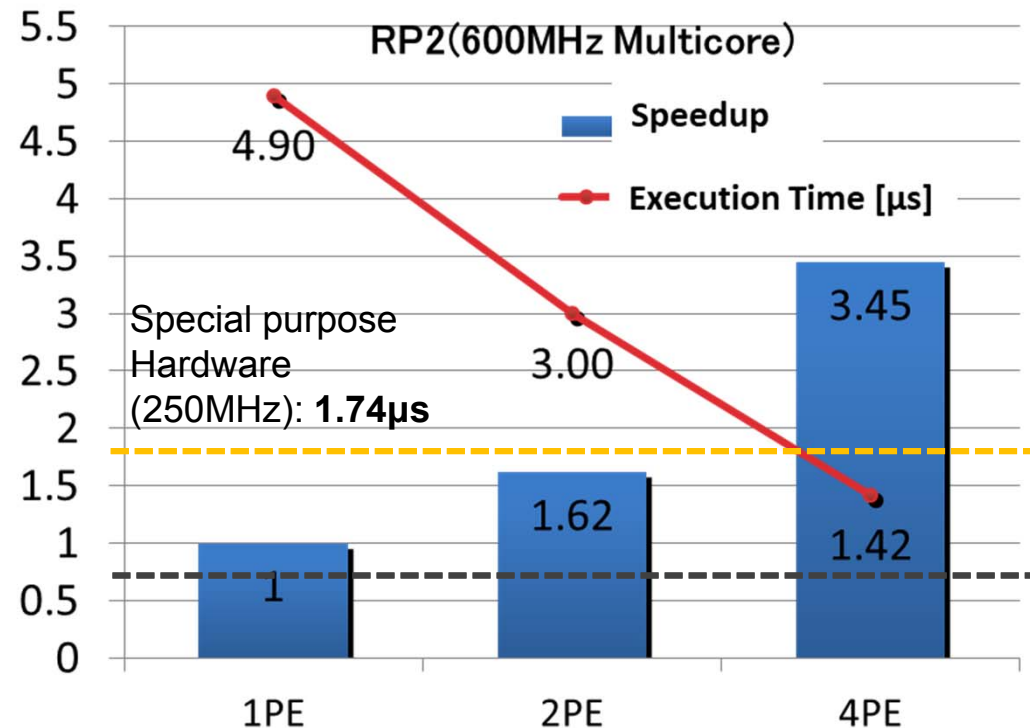
Generated Parallel Machine Code for Near Fine Grain Parallel Processing



【W-CDMA Base Band Communication】

Near Fine Grain Parallel Processing of EAICH Detection Program on RP2 Multicore with 4 SH4A cores

- Hadamard transform often used in the signal processing
- Parallel Processing Method
 - Near fine grain parallel processing among statements
 - Static Scheduling

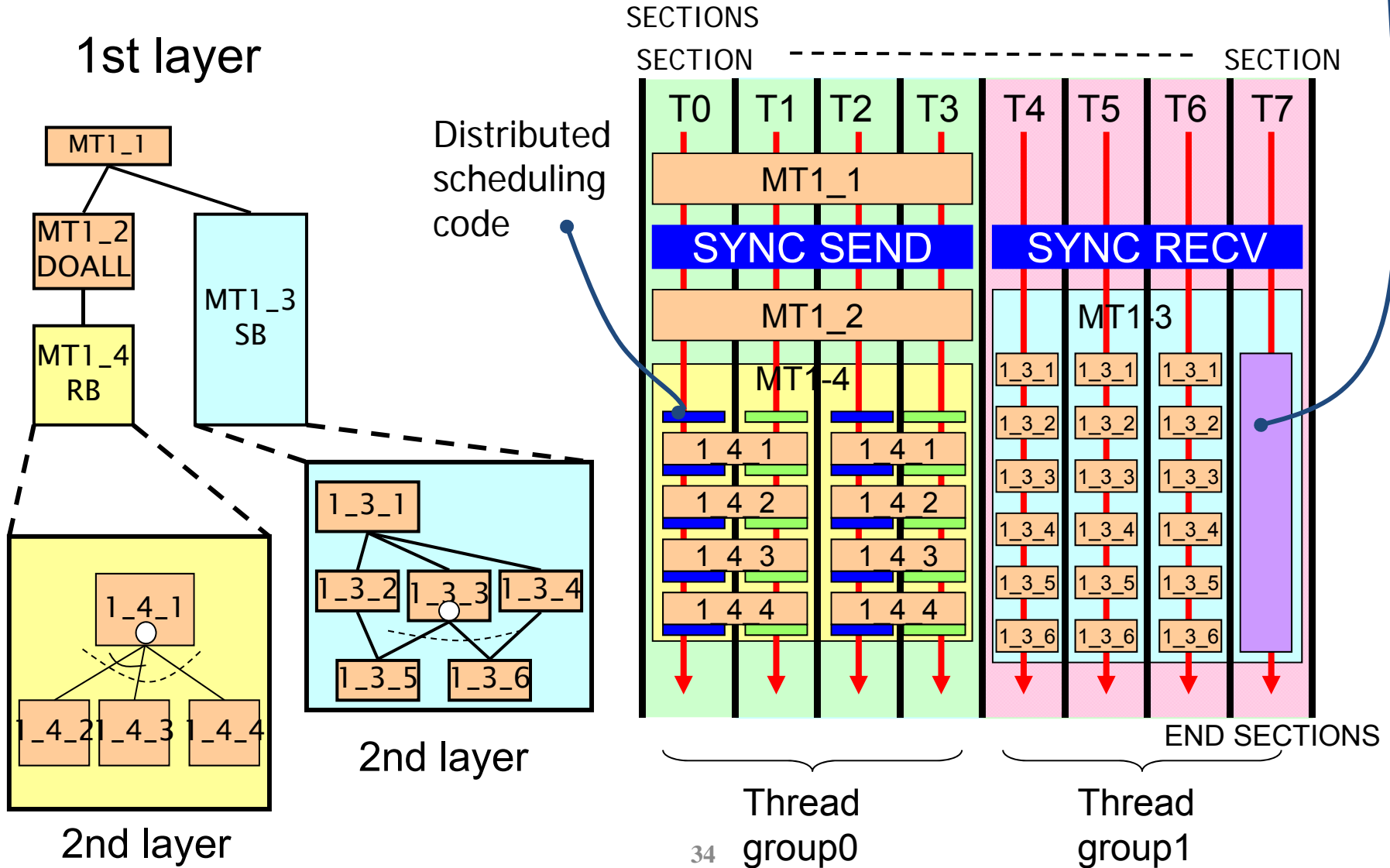


1.62 times speedup for 2cores, 3.45 times speedup for 4 cores for EAICH on RP2.

Generated Multigrain Parallelized Code

(The nested coarse grain task parallelization is realized by only OpenMP “section”, “Flush” and “Critical” directives.)

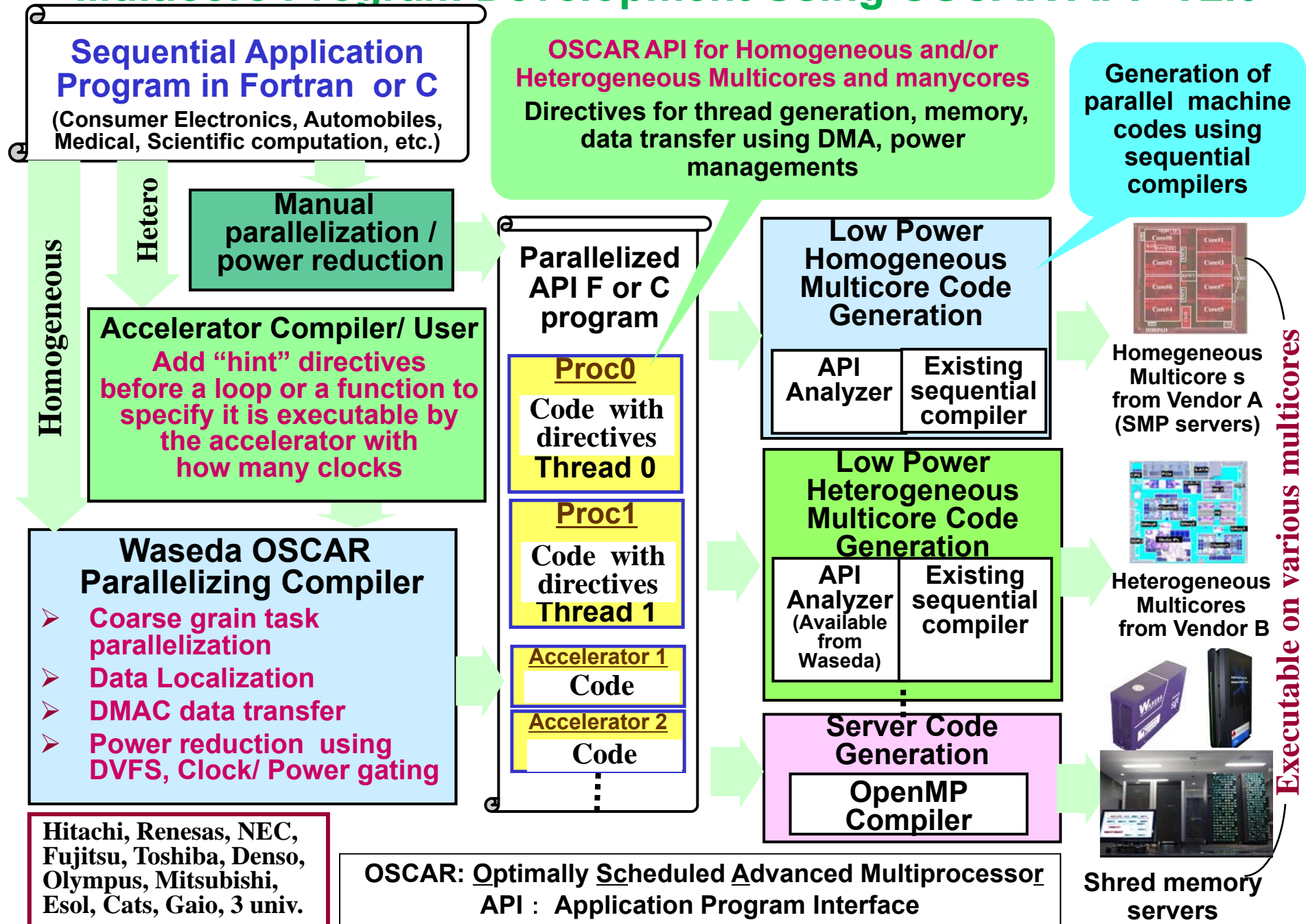
Centralized
scheduling
code



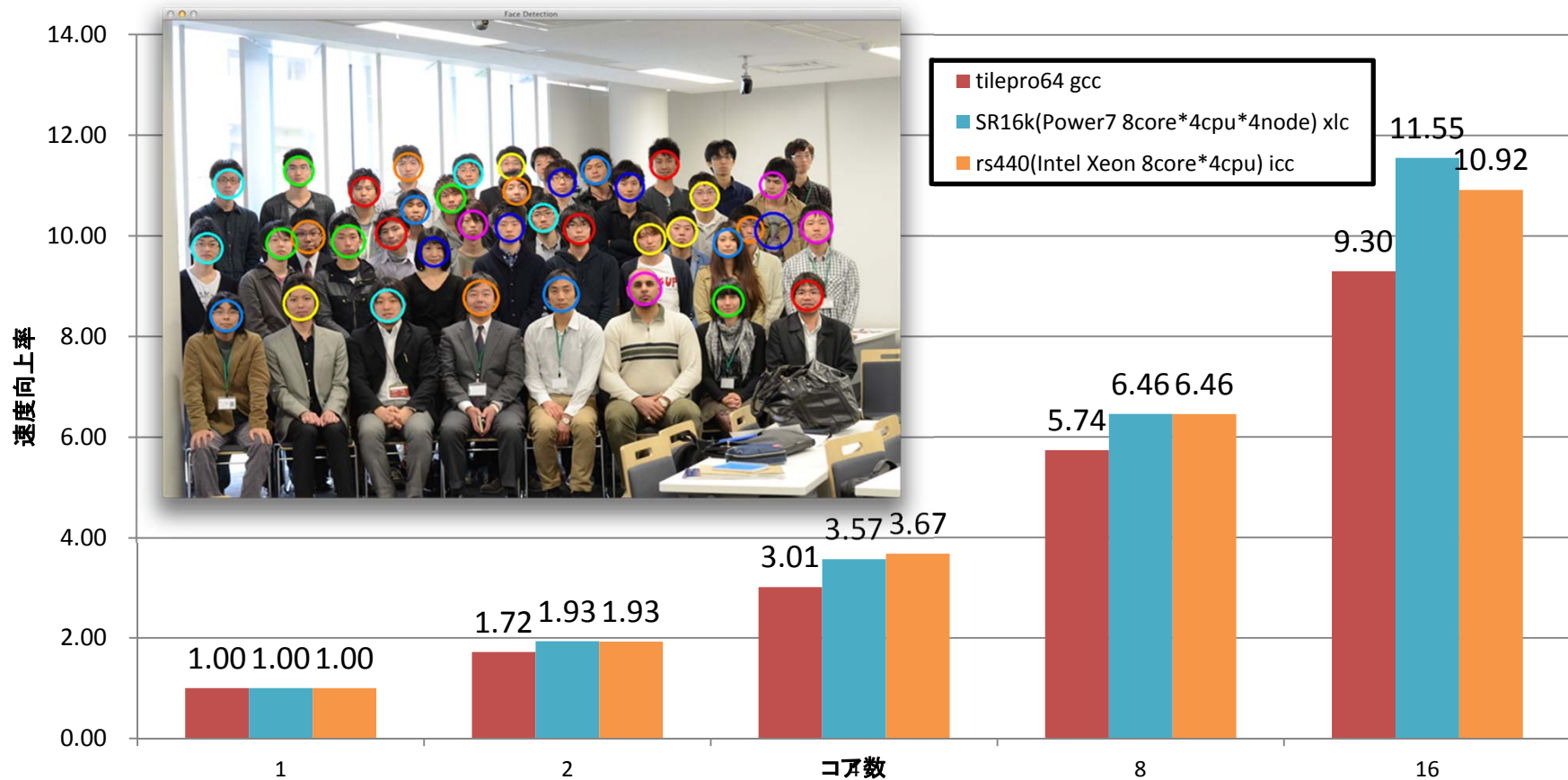
Code Generation Using OpenMP

- Compiler generates a parallelized program using **OpenMP API**
- **One time single level thread generation**
 - Threads are forked only once at the beginning of a program by OpenMP “PARALLEL SECTIONS” directive
 - Forked threads join only once at the end of program
- Compiler generates codes for each threads **using static or dynamic scheduling schemes**
- Extension of OpenMP for hierarchical processing is not required

Multicore Program Development Using OSCAR API V2.0

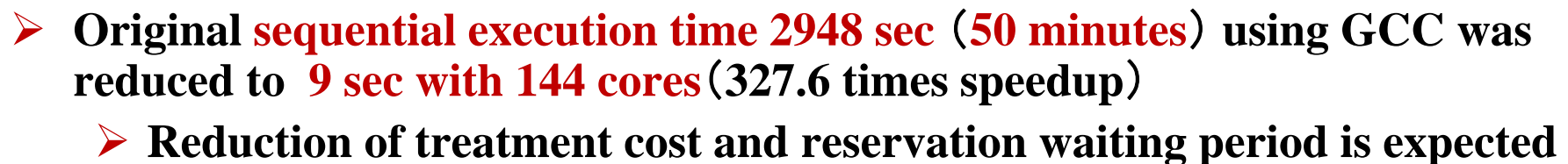
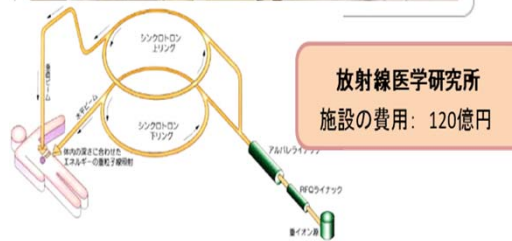


Parallel Processing of Face Detection on Manycore, Highend and PC Server

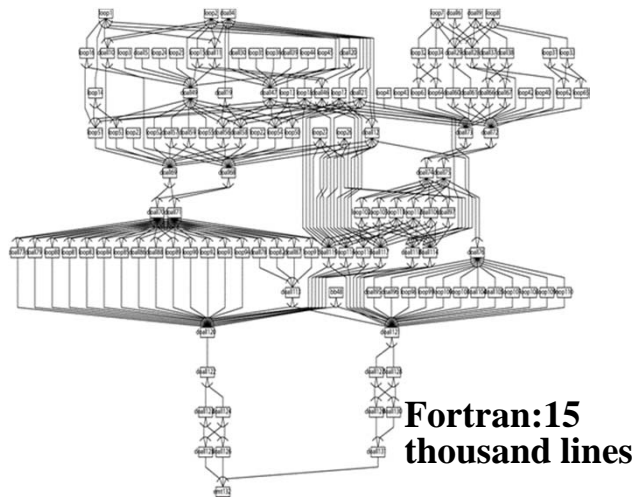
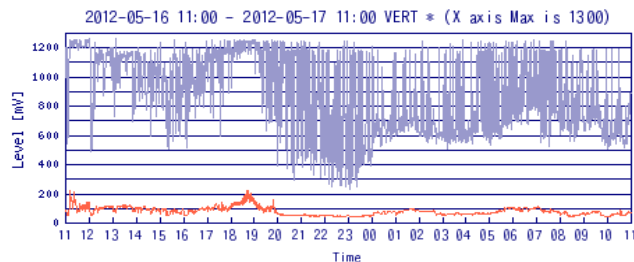


- OSCAR compiler gives us **11.55 times** speedup for 16 cores against 1 core on SR16000 Power7 highend server.

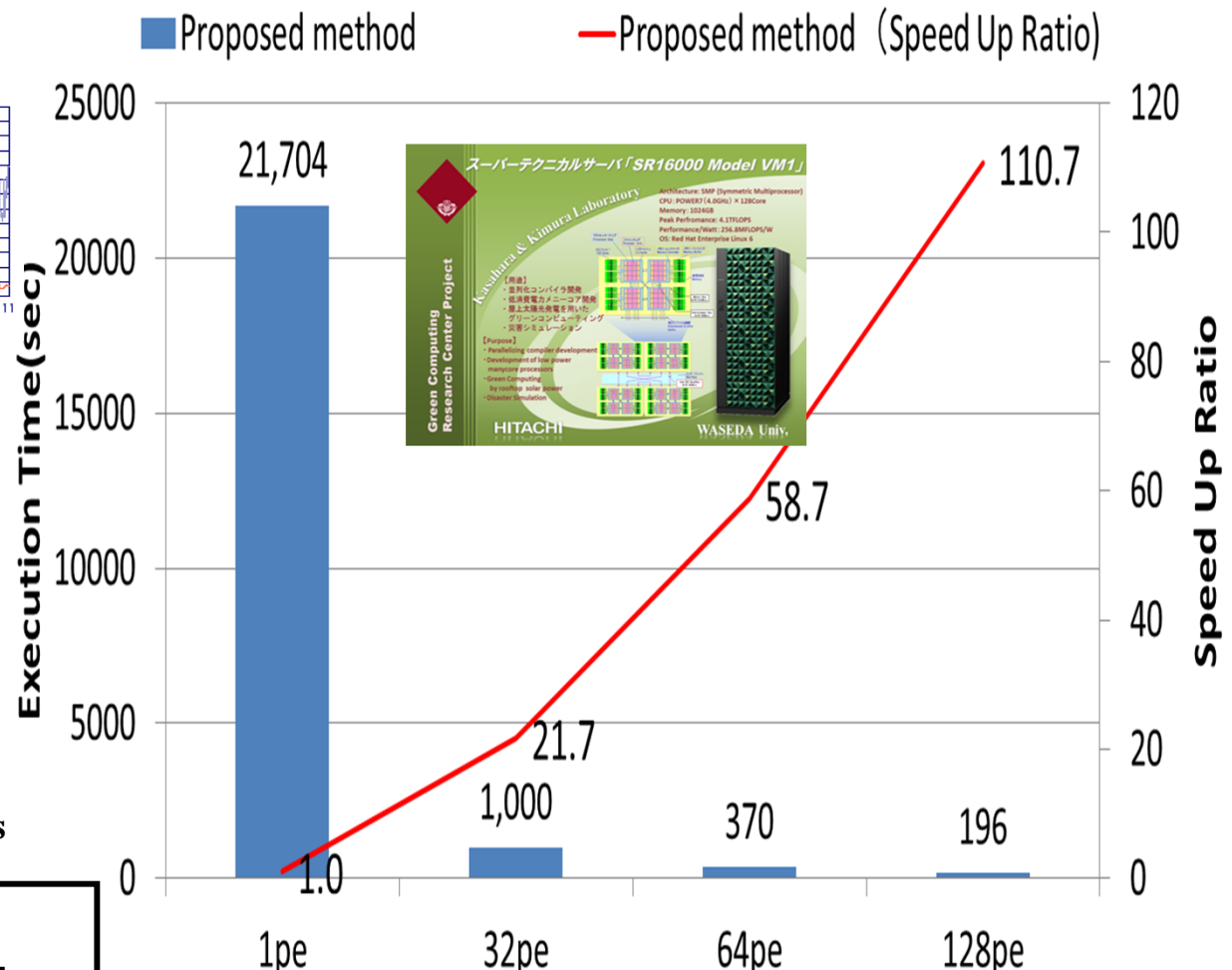
327 times speedup on 144 cores



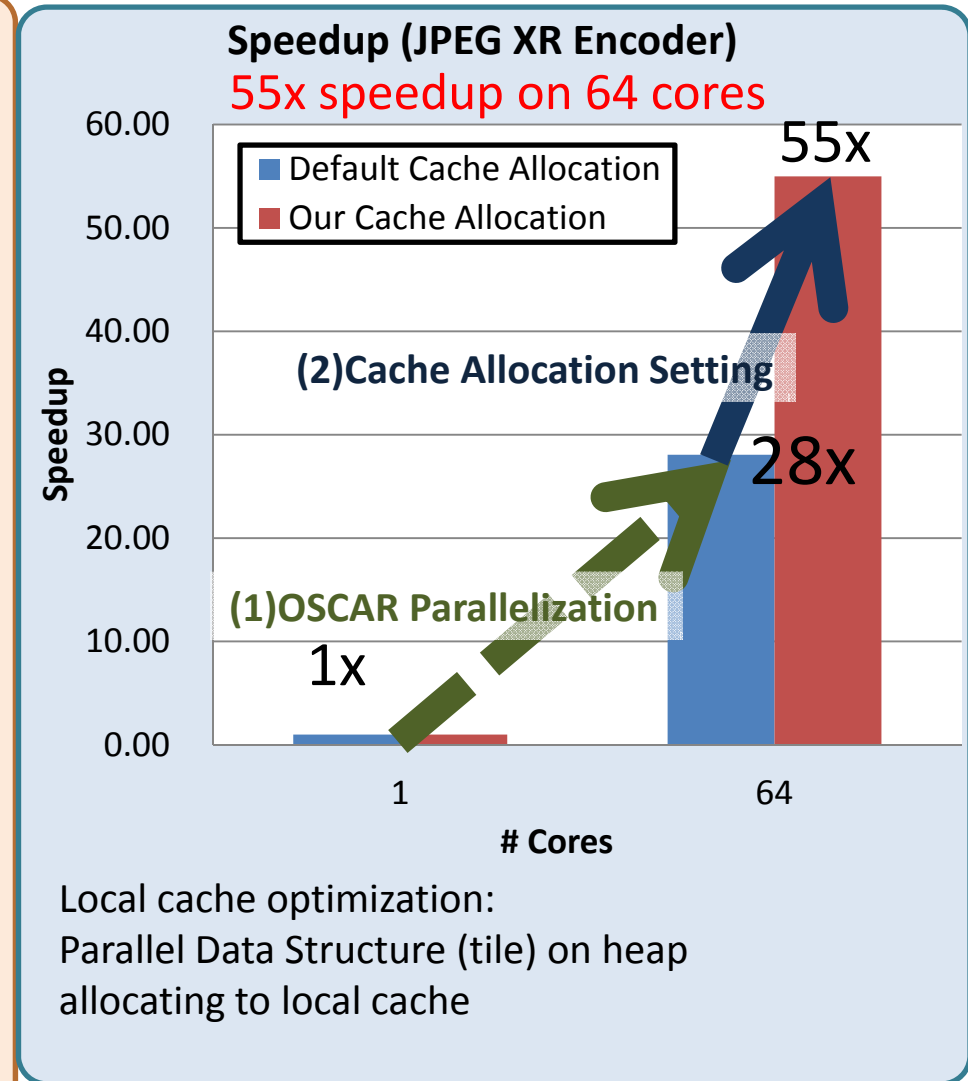
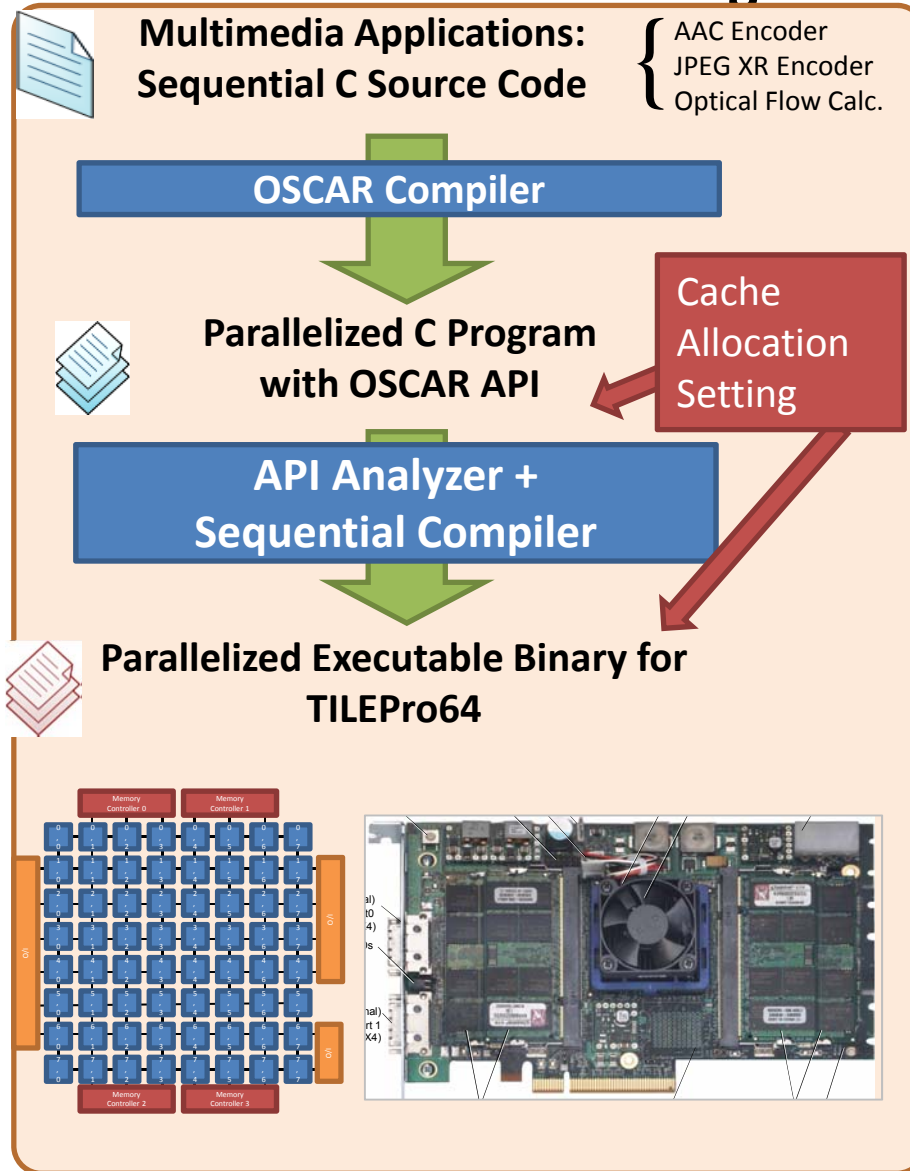
110 Times Speedup against the Sequential Processing for GMS Earthquake Wave Propagation Simulation on Hitachi SR16000 (Power7 Based 128 Core Linux SMP)



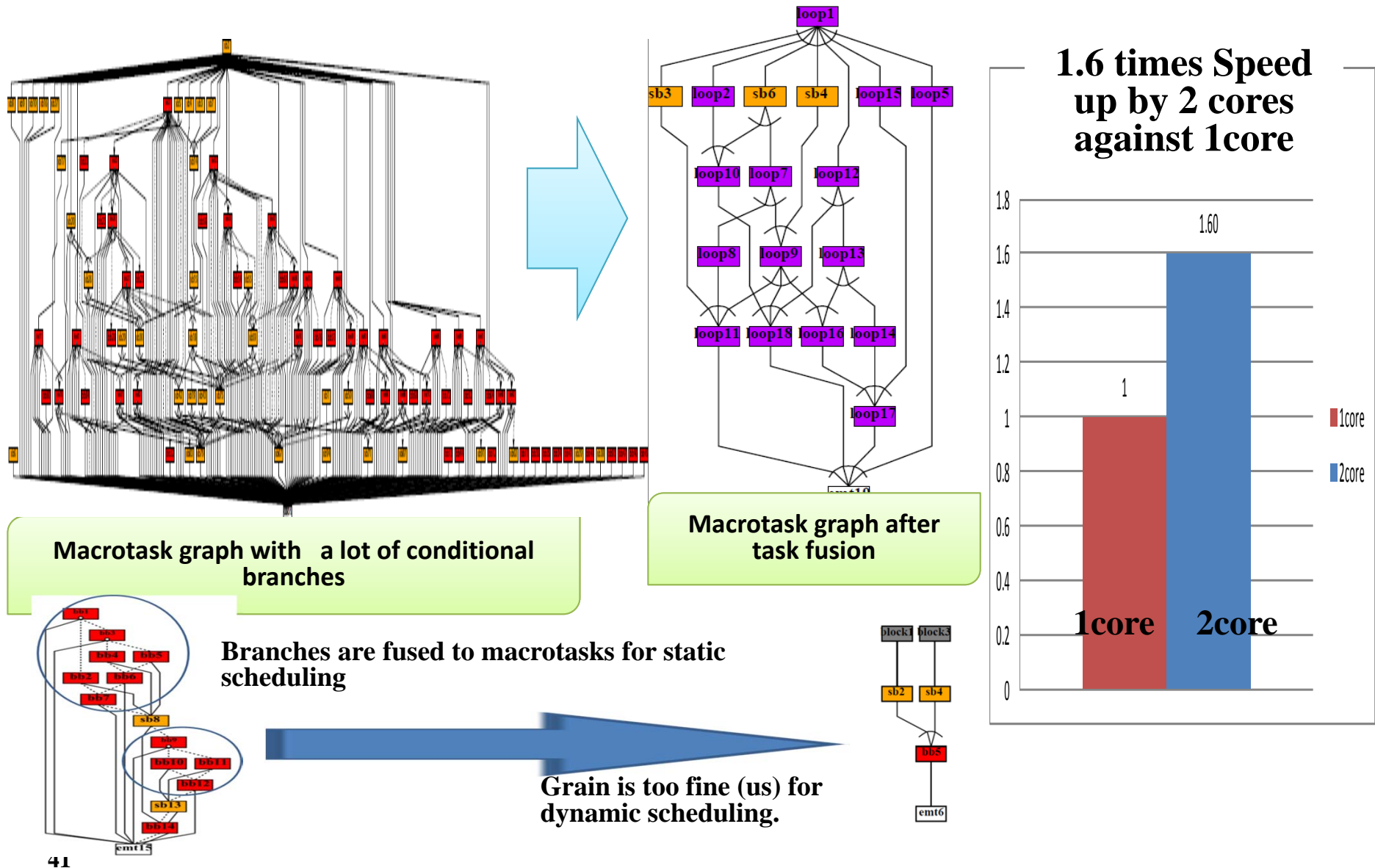
First touch for distributed shared memory and cache optimization over loops are important for scalable speedup



Parallel Processing of JPEG XR Encoder on TILEPro64



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

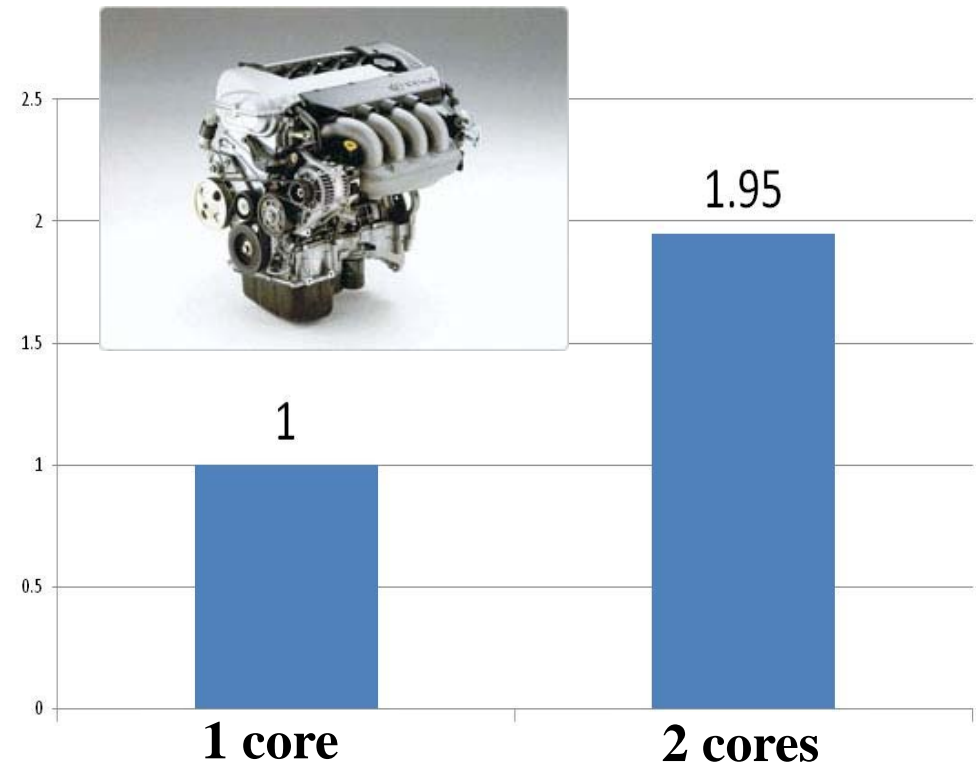




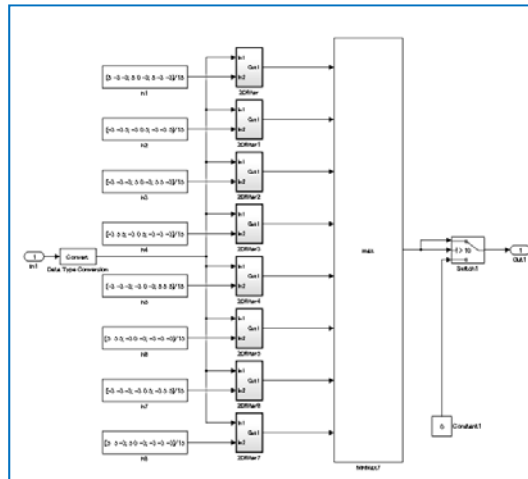
Hard real-time automobile engine control by multicore

エンジン

燃料タンク

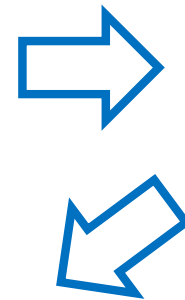


OSCAR Compile Flow for Simulink Applications



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:
     * Import: '<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);

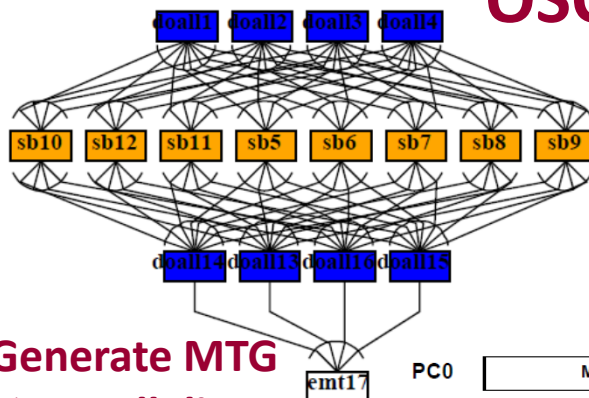
    /* End of Outputs for SubSystem: '<S1>/2Dfilter' */

    /* 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);
}
```

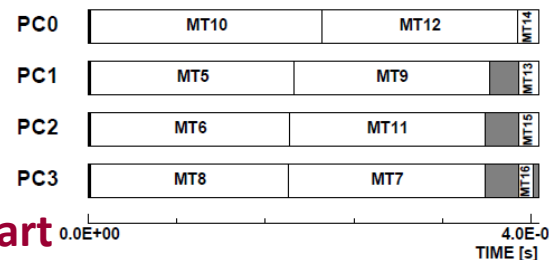
C code

OSCAR Compiler



(1) Generate MTG
→ Parallelism

(2) Generate gantt chart
→ Scheduling in a multicore



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

```
void VesselExtraction_step ( )
{
    int thr1 ;
    int thr2 ;
    int thr3 ;

    void thread_function_001 ( void )
    {
        VesselExtraction_step_PE1 ( ) ;
    }

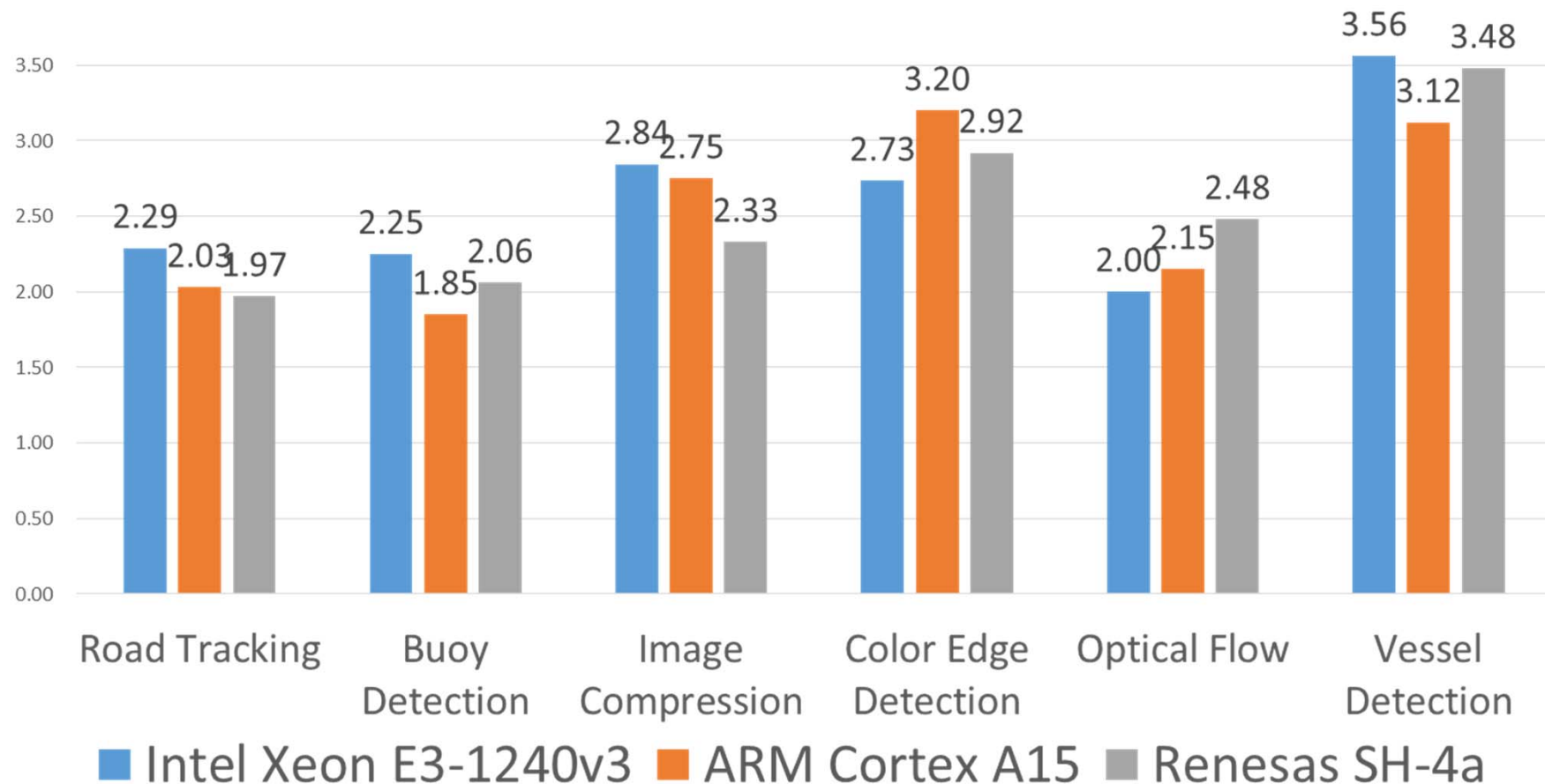
    oscar_thread_create ( & thr1 ,
        thread_function_001 , (void*)1 ) ;
    oscar_thread_create ( & thr2 ,
        thread_function_002 , (void*)2 ) ;
    oscar_thread_create ( & thr3 ,
        thread_function_003 , (void*)3 ) ;

    VesselExtraction_step_PEO ( ) ;

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

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>

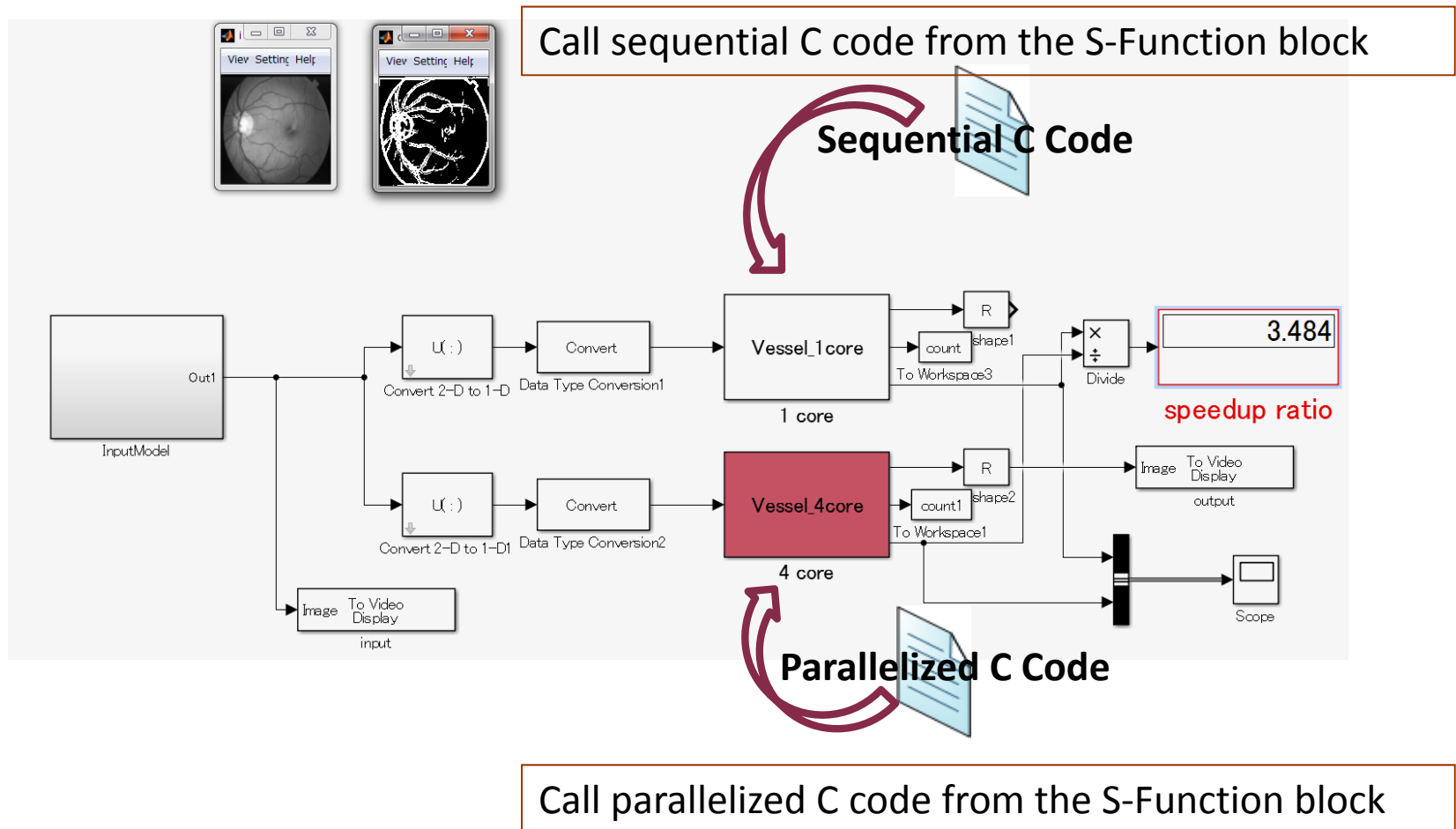
Buoy Detection : <http://www.mathworks.co.jp/matlabcentral/fileexchange/44706-buoy-detection-using-simulink>

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.



OSCAR API Ver. 2.0 for Homogeneous/Heterogeneous Multicores and Manycores

List of Directives (22 directives)

- ▶ **Parallel Execution API**
 - ▶ **parallel sections (*)**
 - ▶ **flush (*)**
 - ▶ **critical (*)**
 - ▶ execution
- ▶ **Memoay Mapping API**
 - ▶ **threadprivate (*)**
 - ▶ distributedshared
 - ▶ onchipshared
- ▶ **Synchronization API**
 - ▶ groupbarrier
- ▶ **Data Transfer API**
 - ▶ dma_transfer
 - ▶ dma_contiguous_parameter
 - ▶ dma_stride_parameter
 - ▶ dma_flag_check
 - ▶ dma_flag_send

(* from OpenMP)

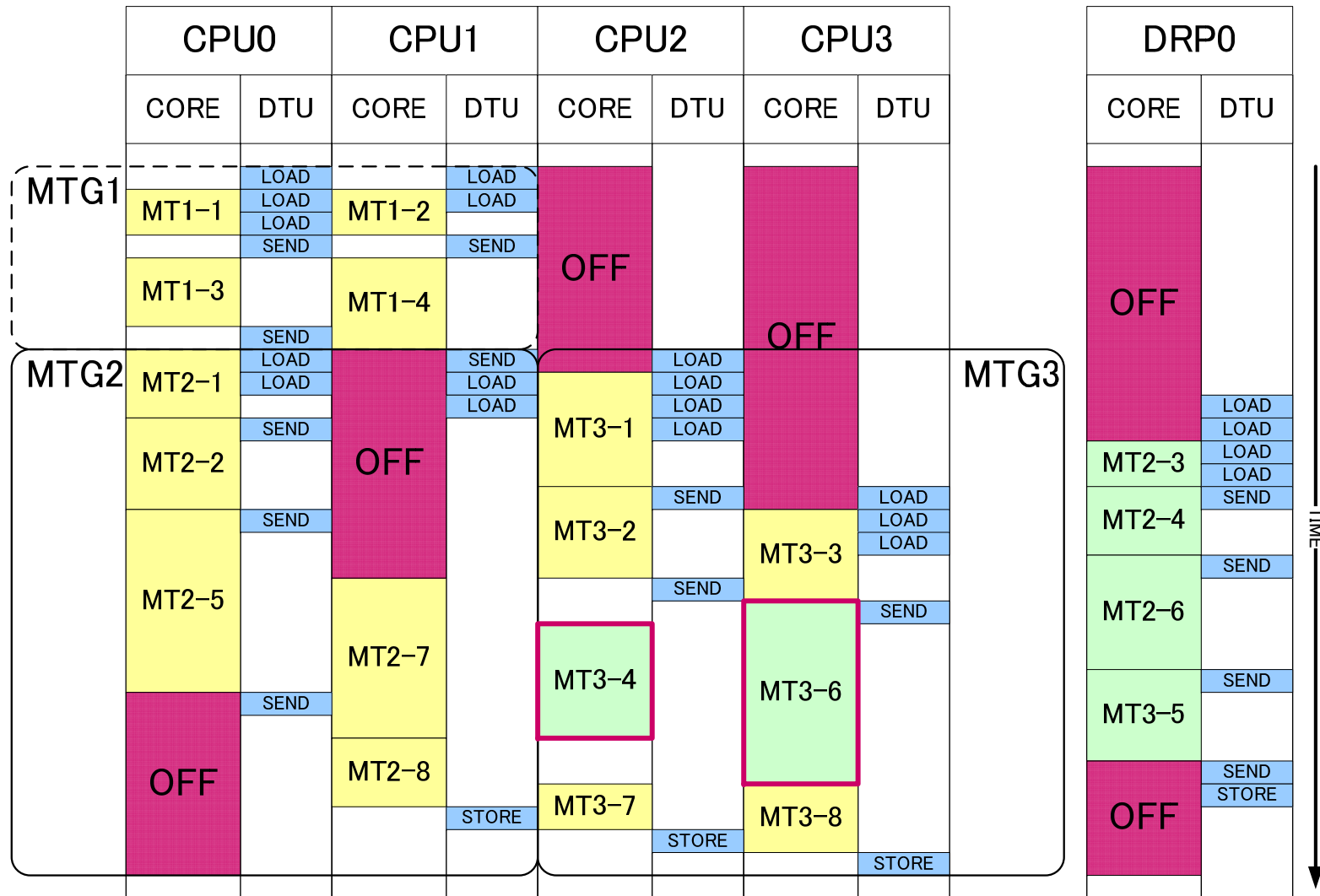
- ▶ **Power Control API**
 - ▶ fvcontrol
 - ▶ get_fvstatus
- ▶ **Timer API**
 - ▶ get_current_time
- ▶ **Accelerator**
 - ▶ accelerator_task_entry
- ▶ **Cache Control**
 - ▶ cache_writeback
 - ▶ cache_selfinvalidate
 - ▶ complete_memop
 - ▶ noncacheable
 - ▶ aligncache

2 hint directives for OSCAR compiler

- accelerator_task
- oscar_comment

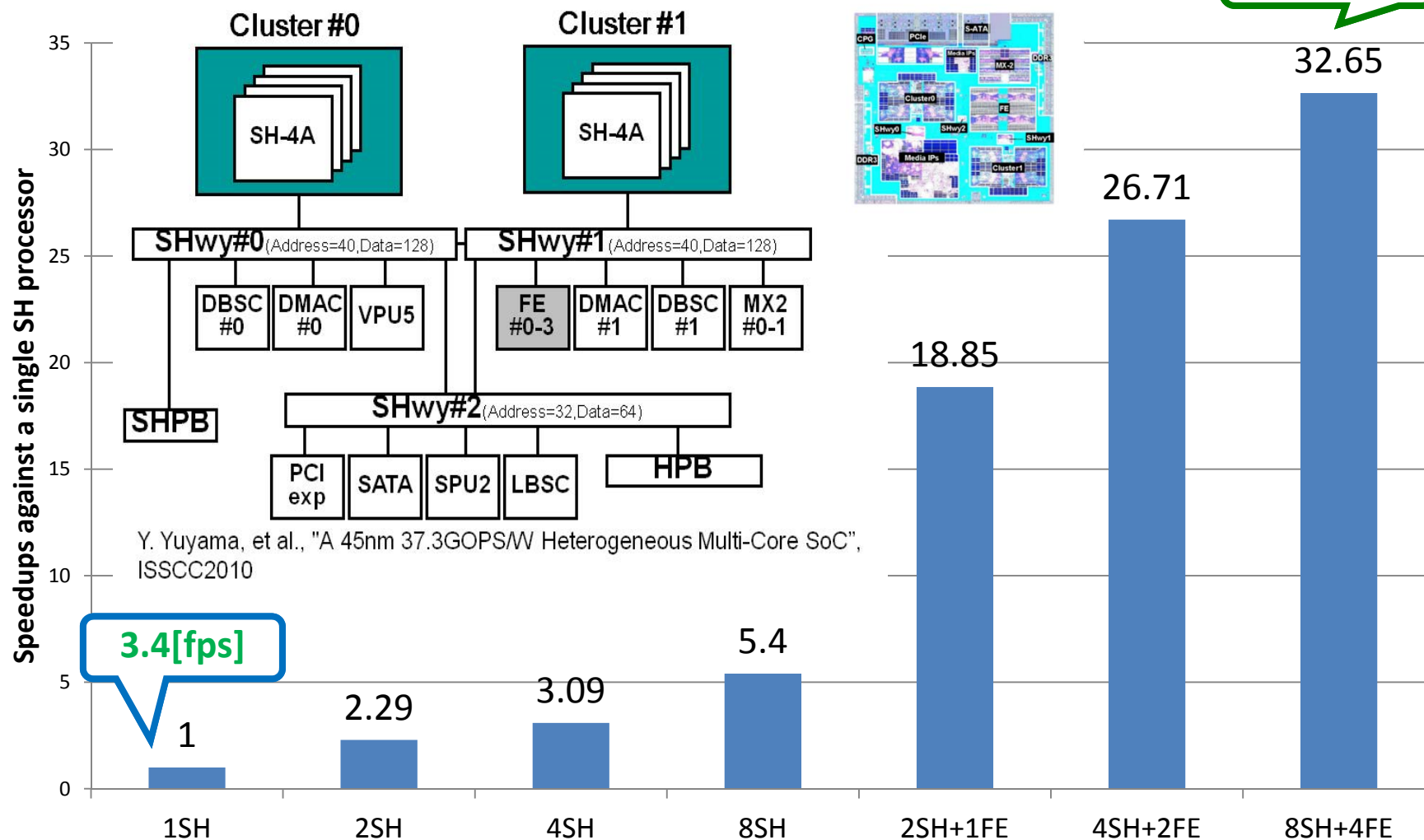
from V2.0

An Image of Static Schedule for Heterogeneous Multi-core with Data Transfer Overlapping and Power Control



33 Times Speedup Using OSCAR Compiler and OSCAR API on RP-X (Optical Flow with a hand-tuned library)

111[fps]



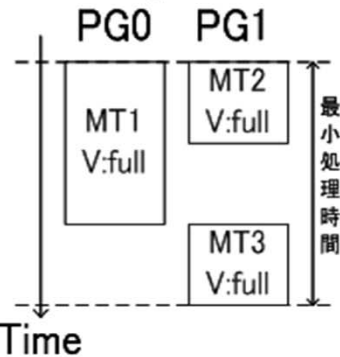
Power Reduction by Power Supply, Clock Frequency and Voltage

Control by OSCAR Compiler

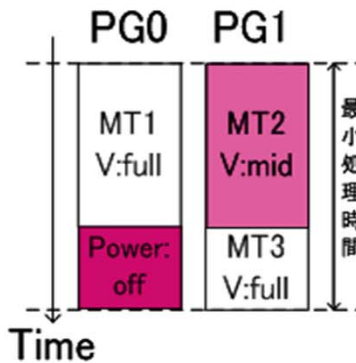
Frequency and Voltage (DVFS), Clock and Power gating of each cores are scheduled considering the task schedule since the dynamic power proportional to the cube of F (F^3) and the leakage power (the static power) can be reduced by the power gating (power off).

- Shortest execution time mode

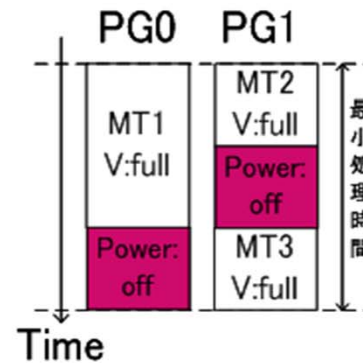
Ordinary scheduled results



FV control



Power control

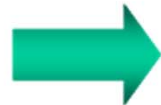
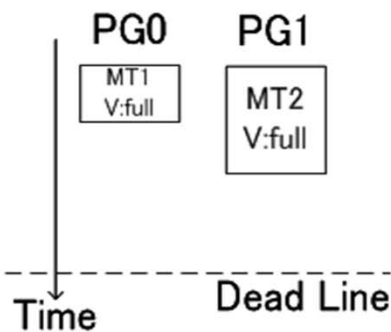


In this Fig.
Frequency
Full, Mid,
Low

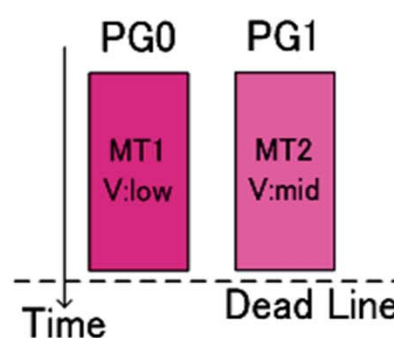
Power OFF:
Power
Gating

- Realtime processing mode with dead line constraints

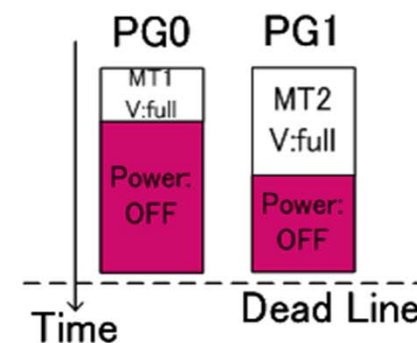
Ordinary scheduled results



FV control



Power control



An Example of Machine Parameters for the Power Saving Scheme

- **Functions of the multiprocessor**
 - Frequency of each proc. is changed to several levels
 - Voltage is changed together with frequency
 - Each proc. can be powered on/off

state	FULL	MID	LOW	OFF
frequency	1	1 / 2	1 / 4	0
voltage	1	0.87	0.71	0
dynamic energy	1	3 / 4	1 / 2	0
static power	1	1	1	0

• State transition overhead

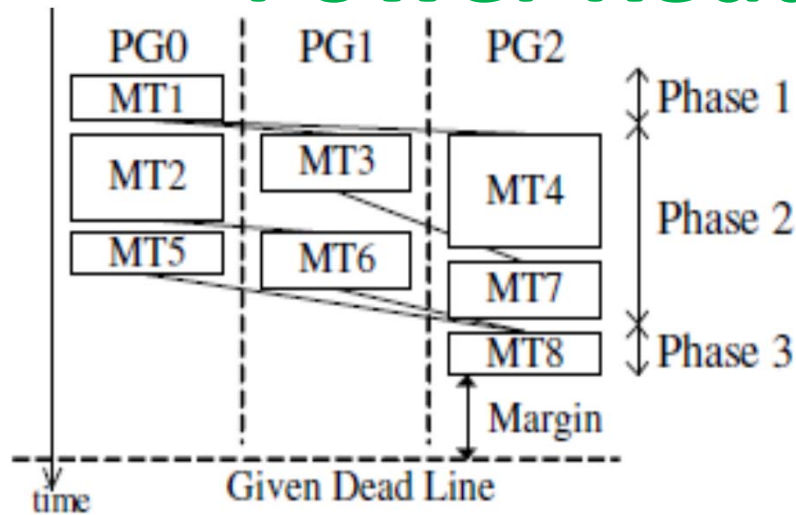
state	FULL	MID	LOW	OFF
FULL	0	40k	40k	80k
MID	40k	0	40k	80k
LOW	40k	40k	0	80k
OFF	80k	80k	80k	0

delay time [u.t.]

state	FULL	MID	LOW	OFF
FULL	0	20	20	40
MID	20	0	20	40
LOW	20	20	0	40
OFF	40	40	40	0

energy overhead [μ J]

Power Reduction Scheduling



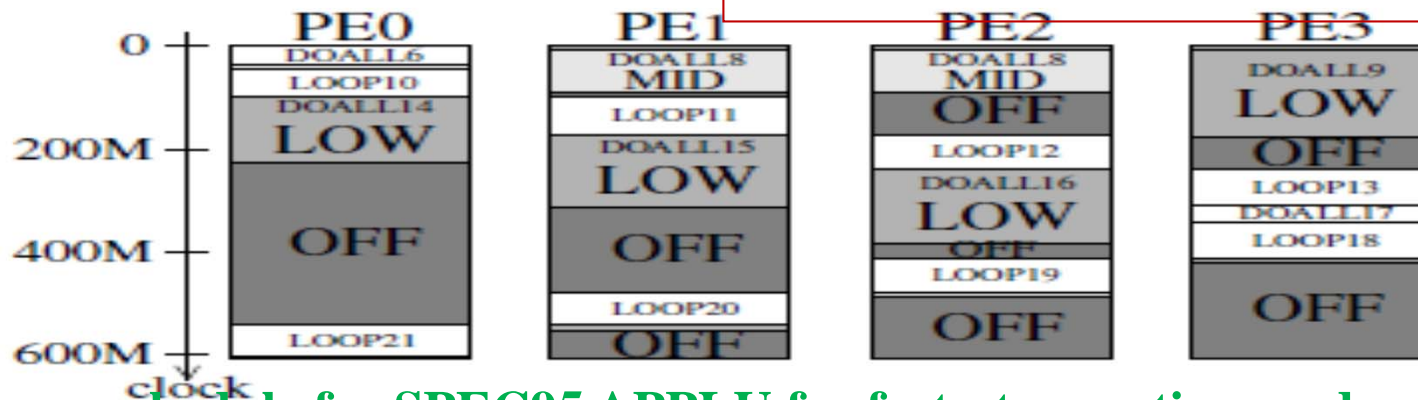
A macrotask graph assigned to 3 cores

Realtime scheduling mode

MTs 1,4,7,8 are on Critical Path (CP)

A power schedule for fastest execution mode

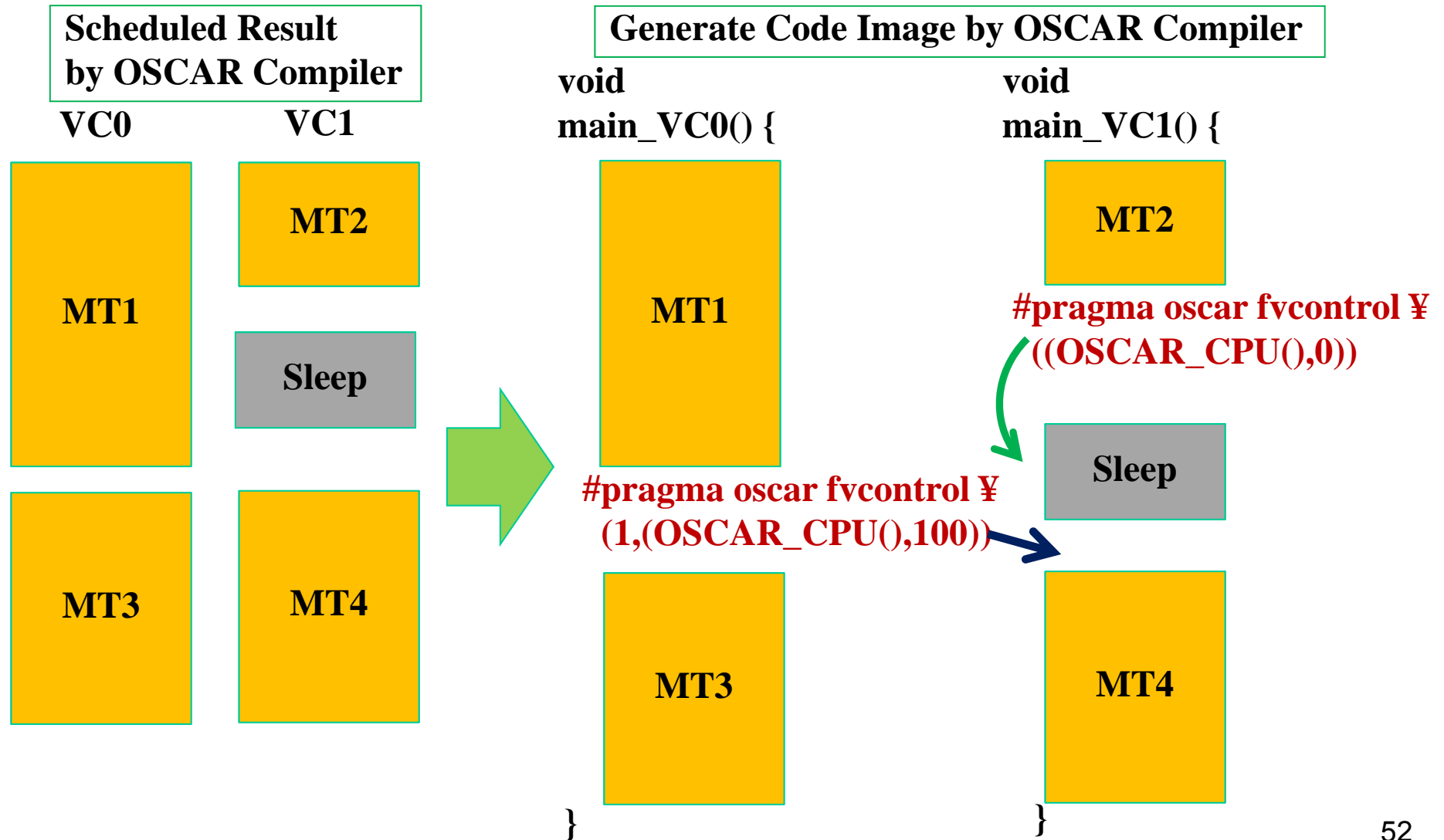
- 1) Reduce frequencies (Fs) of MTs on CP considering dead line.
- 2) Reduce Fs of MTs not on CP. Idle: Clock or Power Gating considering overheads.



A power schedule for SPEC95 APPLU for fastest execution mode

Doall6, Loop 10,11,12,13, Doall 17, Loop 18,19, 20, 21 are on CP

Low-Power Optimization with OSCAR API



Power Reduction in a real-time execution controlled by OSCAR Compiler and OSCAR API on RP-X (Optical Flow with a hand-tuned library)

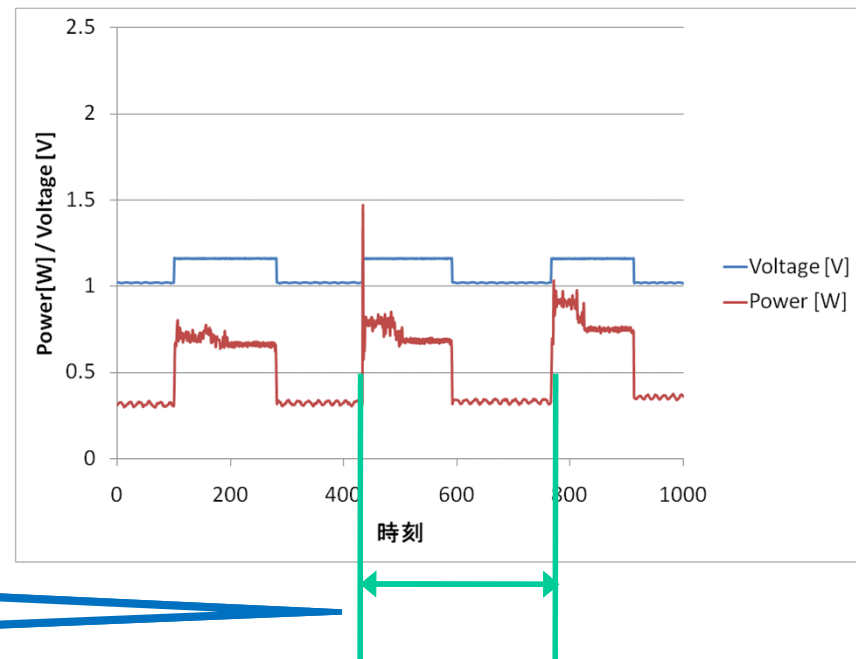
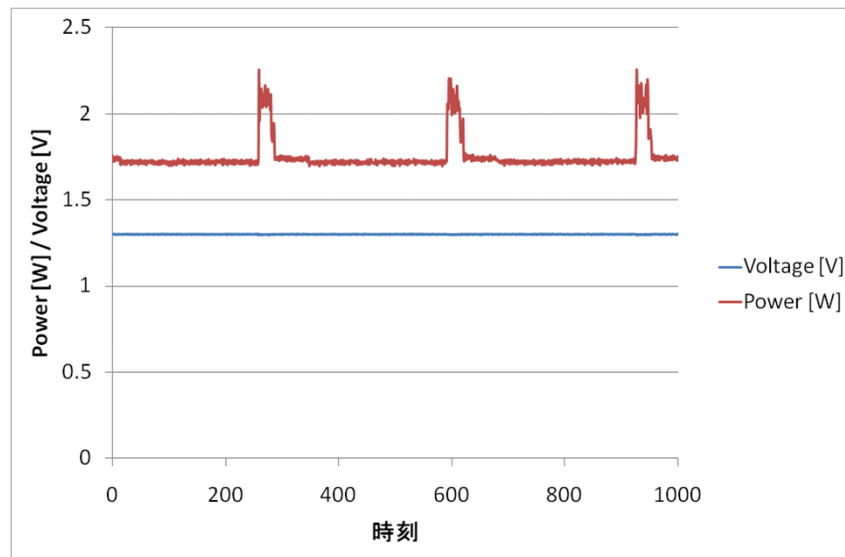
Without Power Reduction

70% of power reduction

**With Power Reduction
by OSCAR Compiler**

Average: 1.76[W]

Average: 0.54[W]

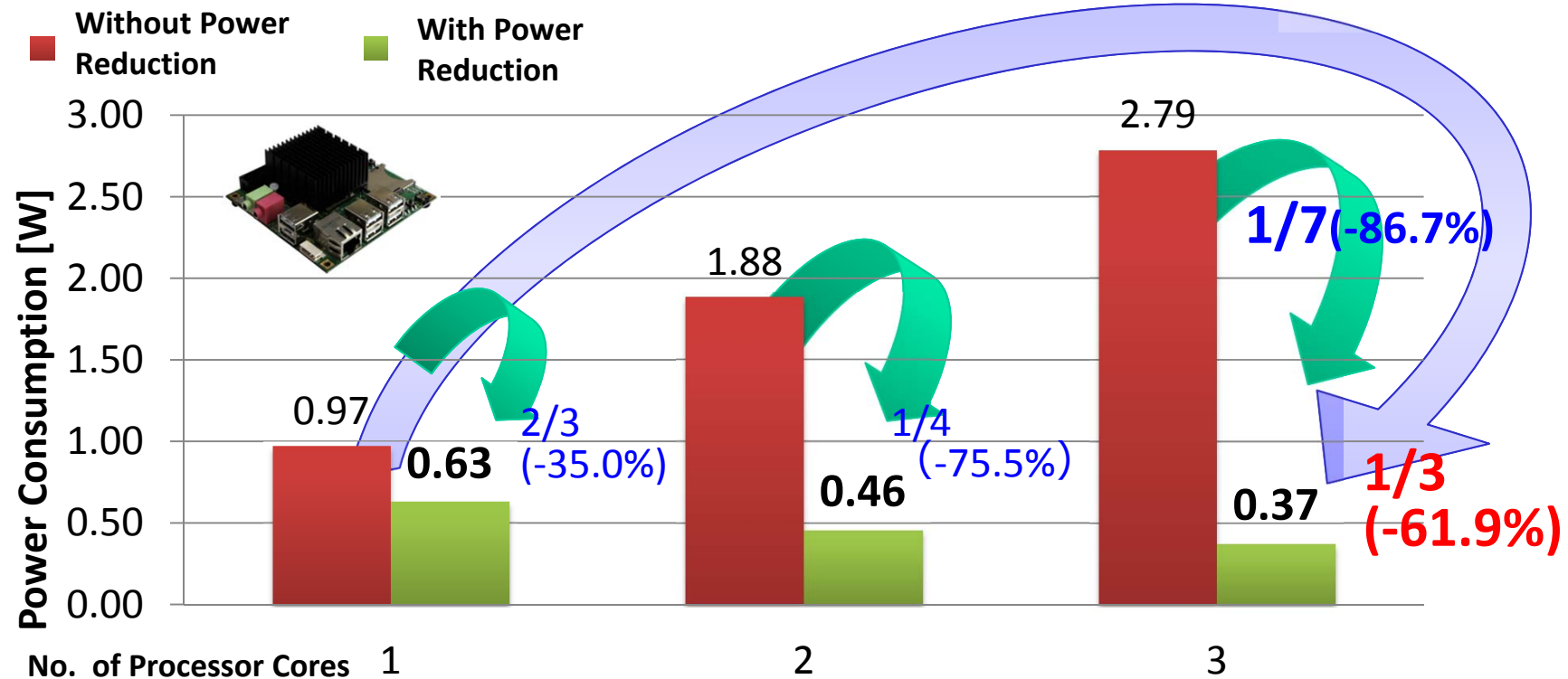


**1cycle : 33[ms]
→30[fps]**

Automatic Power Reduction for MPEG2 Decode on Android Multicore

ODROID X2 ARM Cortex-A9 4 cores

http://www.youtube.com/channel/UCS43INYEIkC8i_KIgFZYQBQ

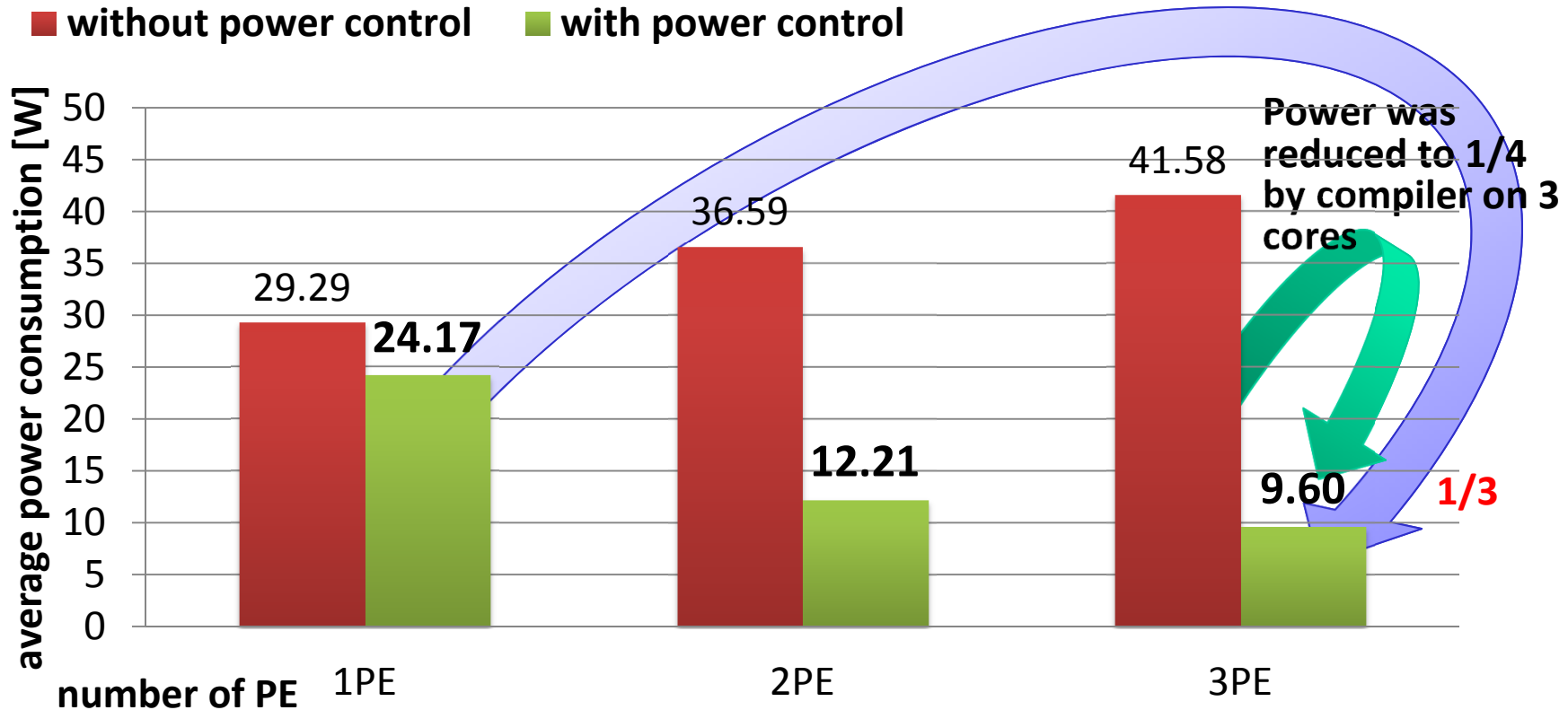
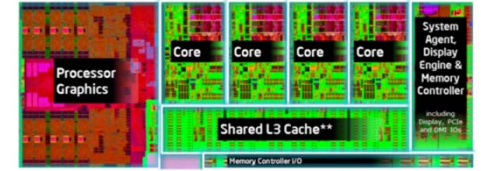


- On 3 cores, Automatic Power Reduction control successfully reduced power to 1/7 against without Power Reduction control.
- 3 cores with the compiler power reduction control reduced power to 1/3 against ordinary 1 core execution.

Power Reduction on Intel Haswell for Real-time Optical Flow

Intel CPU Core i7 4770K

For HD 720p(1280x720) moving pictures
15fps (Deadline66.6[ms/frame])



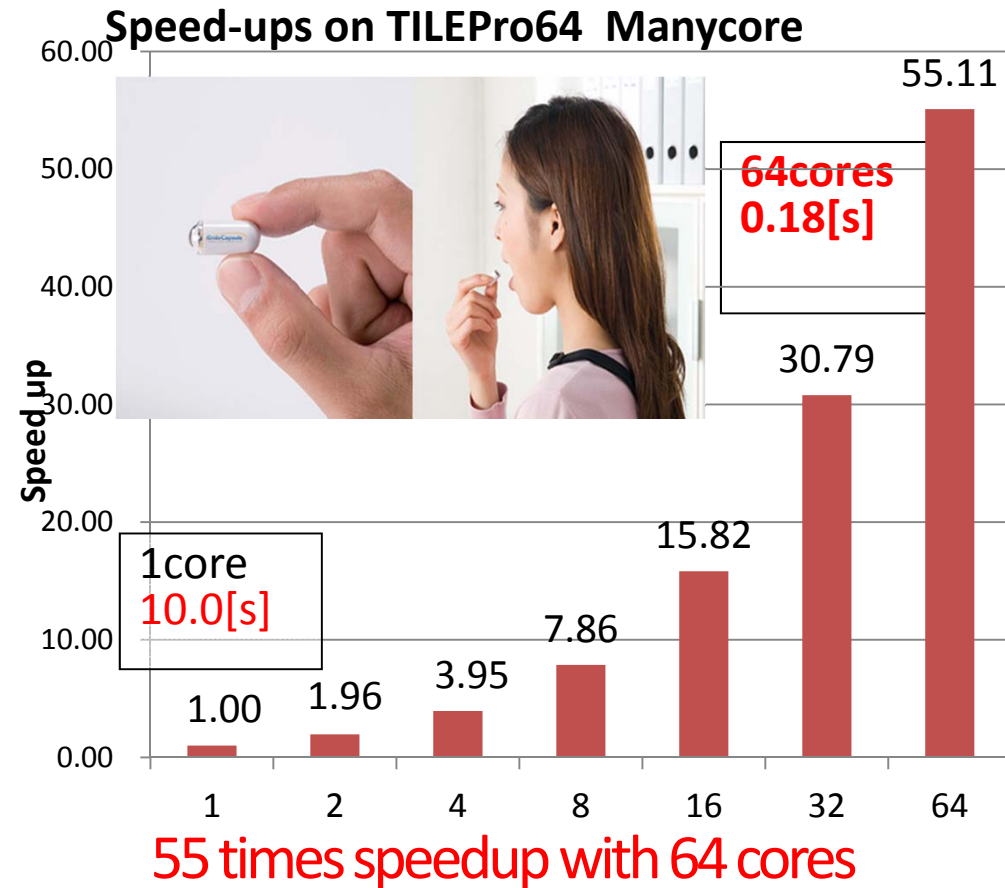
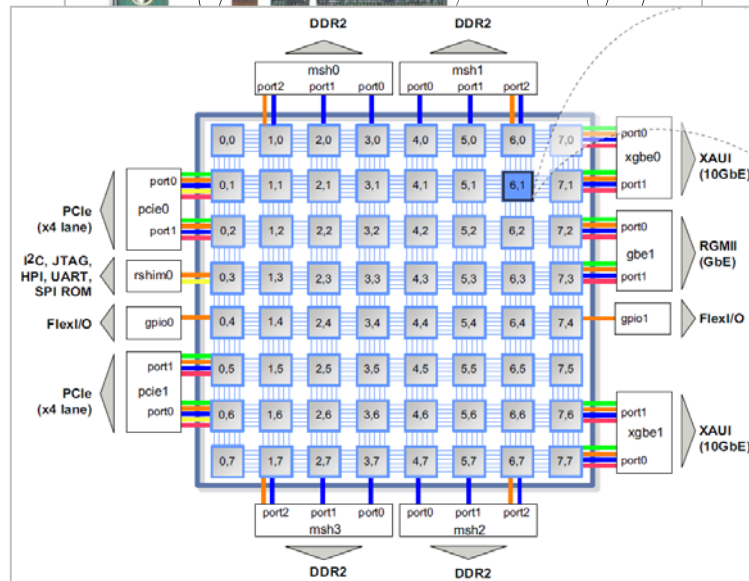
Power was reduced to **1/4 (9.6W)** by the compiler power optimization **on the same 3 cores (41.6W)**.

Power with 3 core was reduced to **1/3 (9.6W)** against **1 core (29.3W)**.

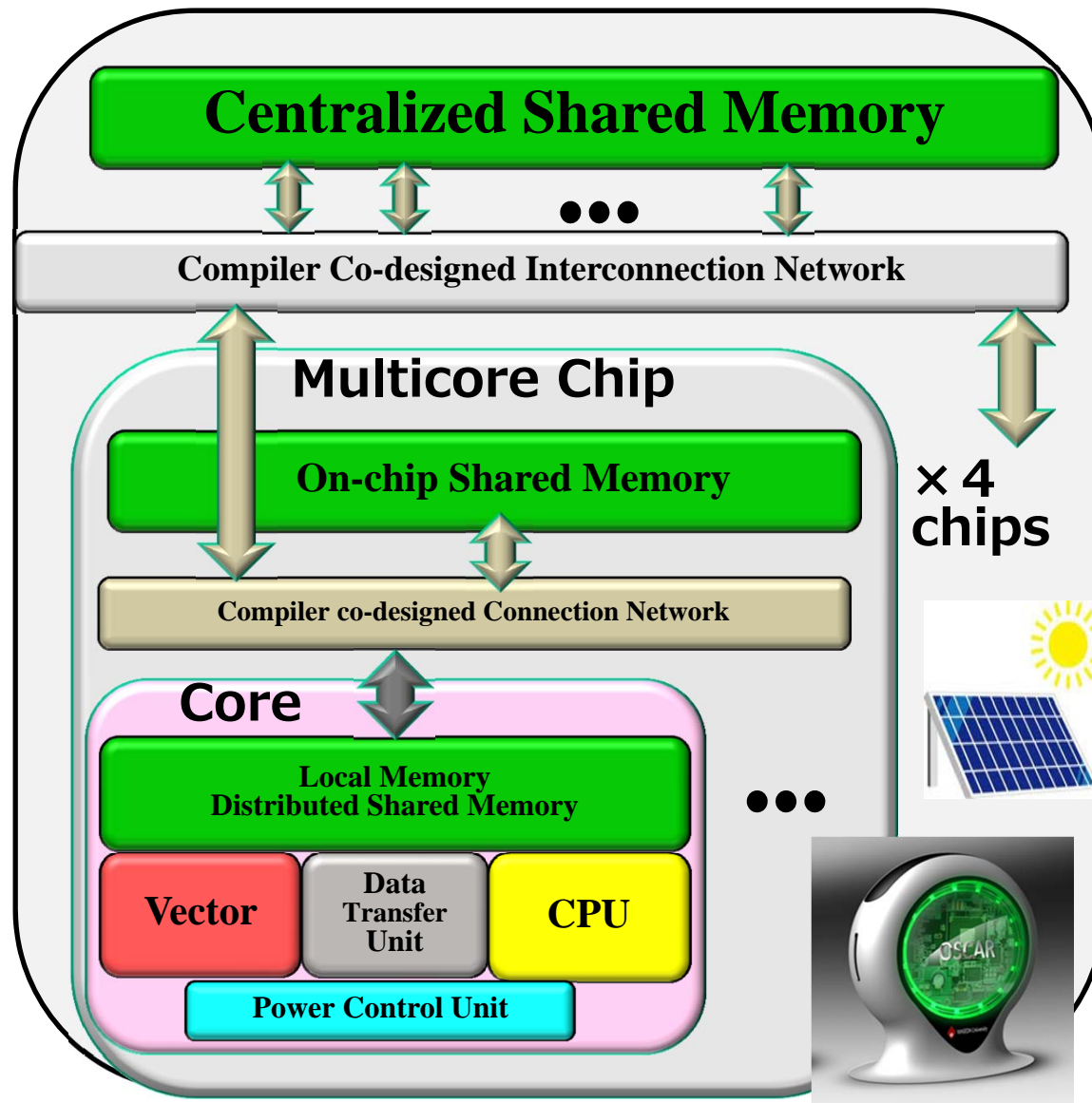
Automatic Parallelization of JPEG-XR for Drinkable Inner Camera (Endo Capsule)

10 times more speedup needed after parallelization for 128 cores of
Power 7. Less than 35mW power consumption is required.

- TILEPro64



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



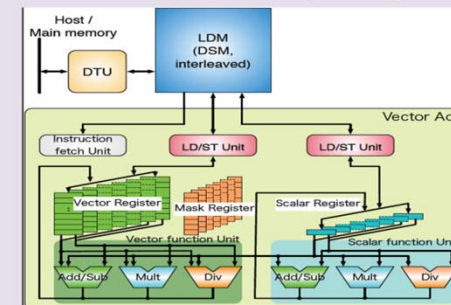
Target:

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

Vector Accelerator

Features

- Attachable for any CPUs (Intel, ARM, IBM)
- Data driven initiation by sync flags



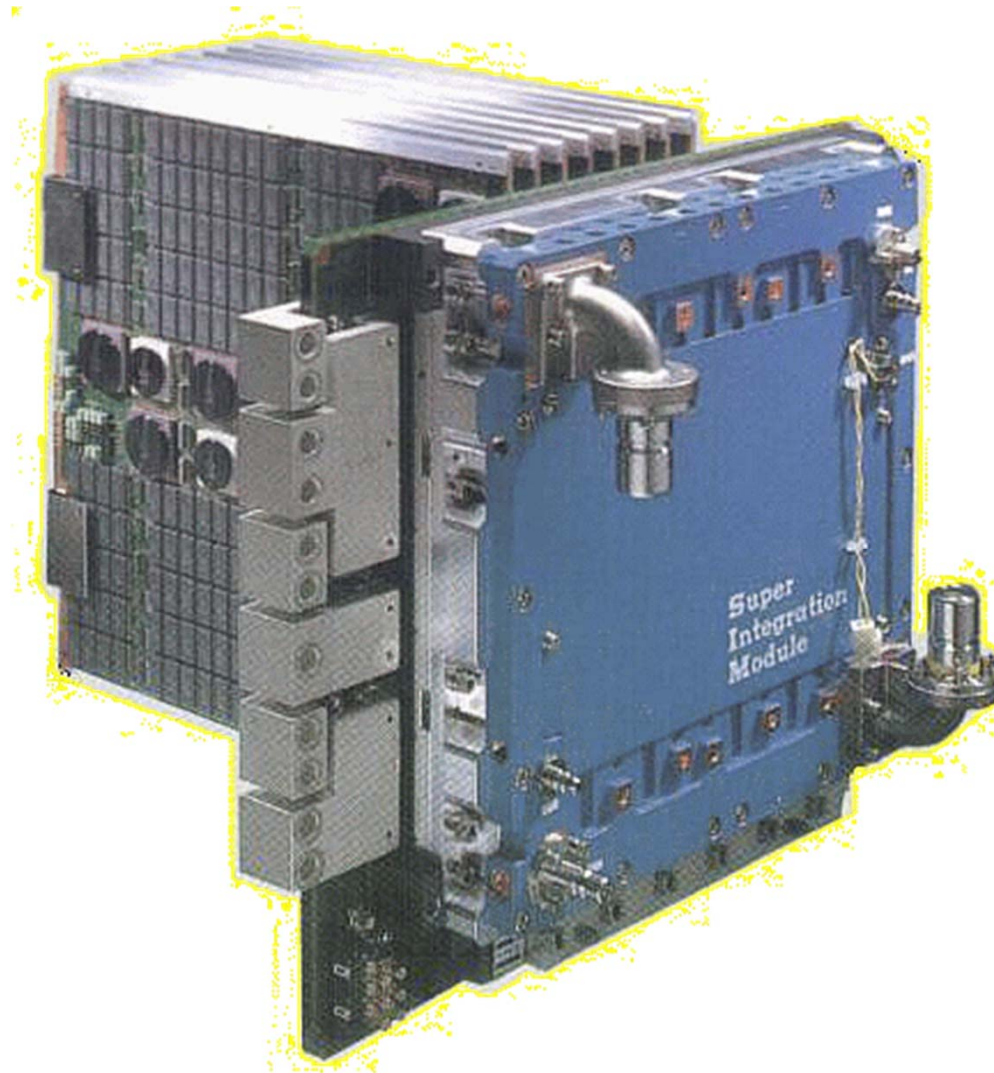
Function Units [tentative]

- **Vector Function Unit**
 - 8 double precision ops/clock
 - 64 characters ops/clock
 - Variable vector register length
 - Chaining LD/ST & Vector pipes
- **Scalar Function Unit**

Registers[tentative]

- **Vector Register** 256Bytes/entry, 32entry
- **Scalar Register** 8Bytes/entry
- **Floating Point Register** 8Bytes/entry
- **Mask Register** 32Bytes/entry

Fujitsu VPP500/NWT: PE Unit

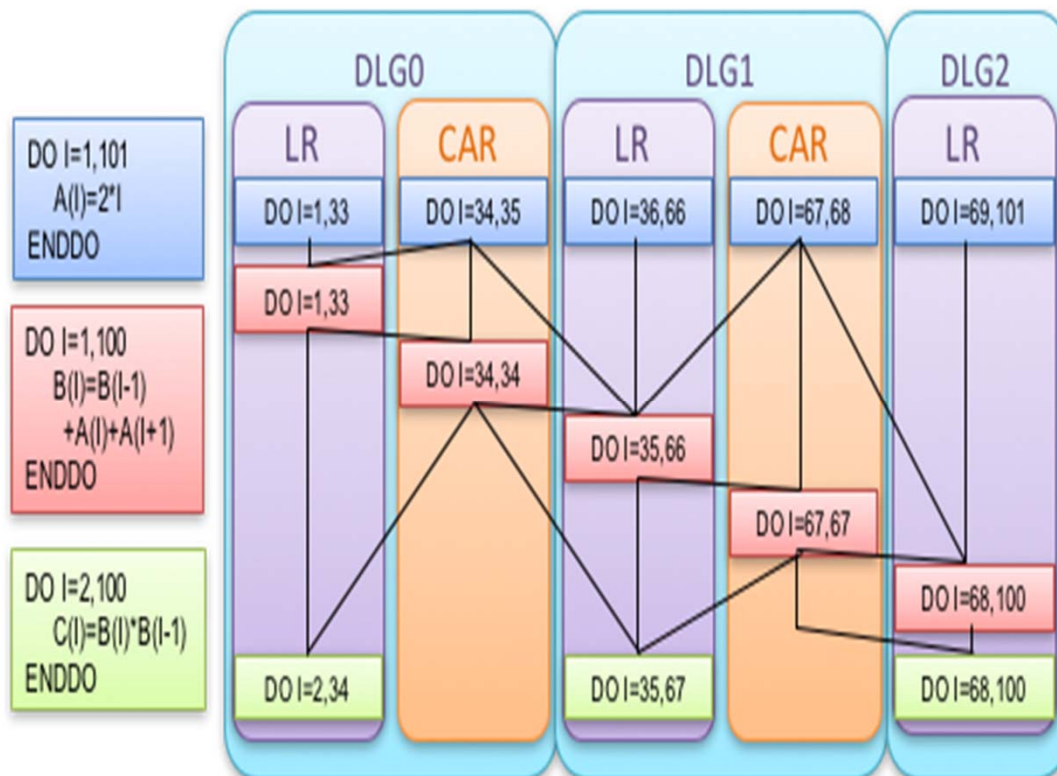


Automatic Local Memory Management

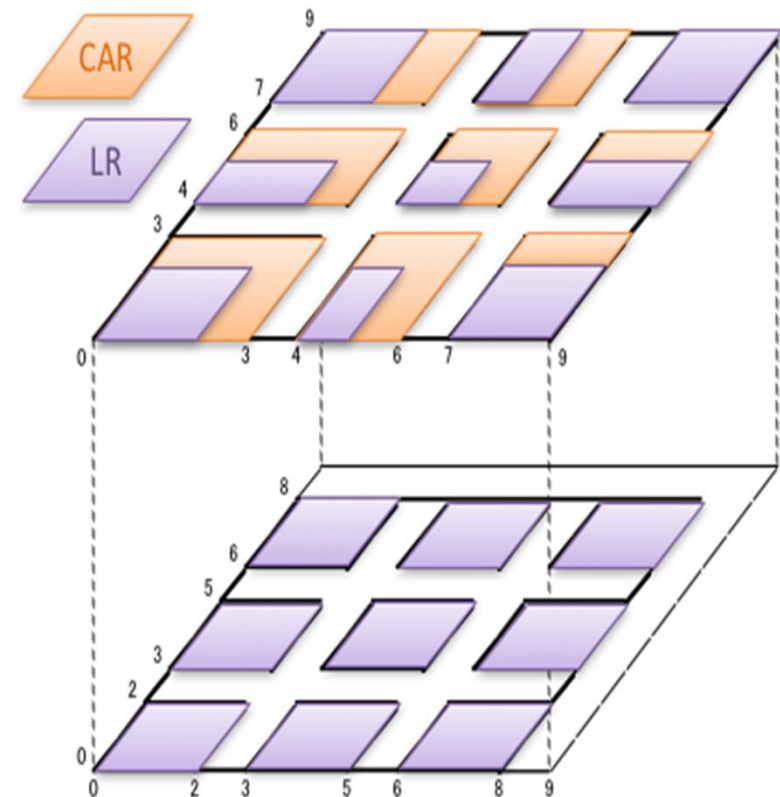
Data Localization: Loop Aligned Decomposition

- Decomposed loop into LRs and CARs
 - LR (Localizable Region): Data can be passed through LDM
 - CAR (Commonly Accessed Region): Data transfers are required among processors

Single dimension Decomposition

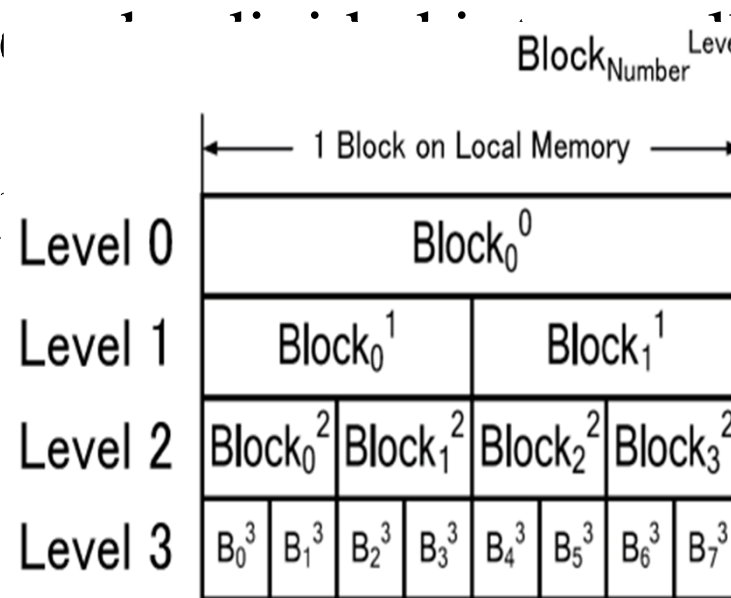


Multi-dimension Decomposition



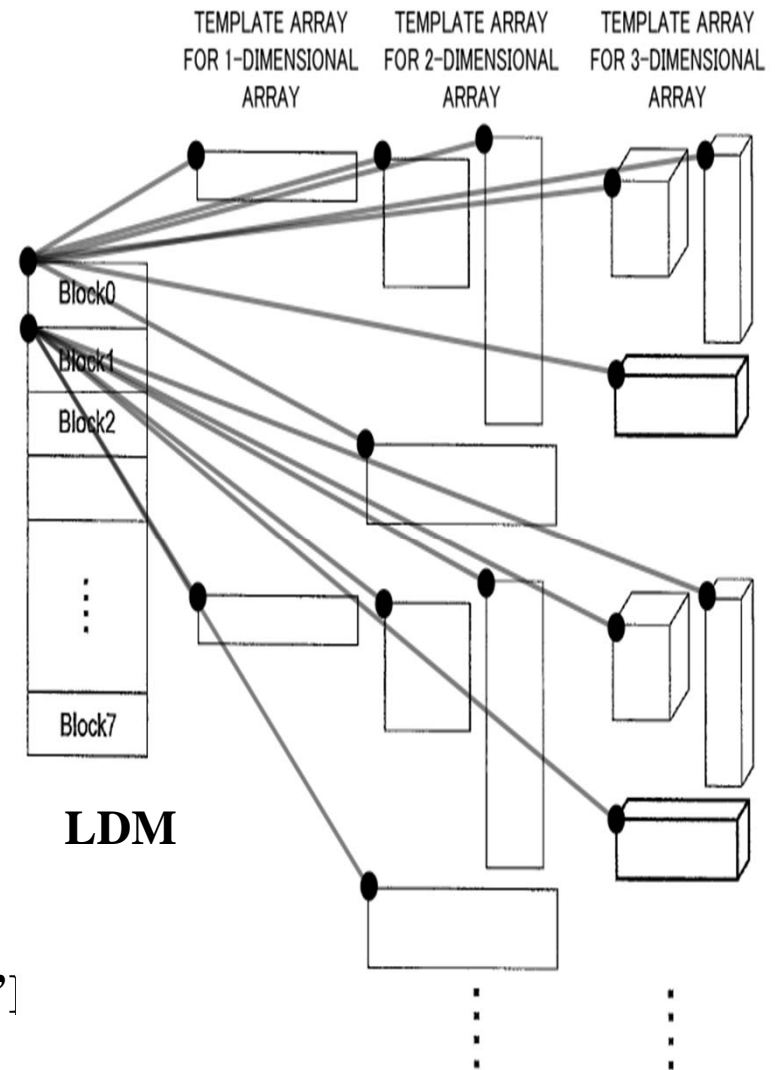
Adjustable Blocks

- Handling a suitable block size for each application
 - different from a fixed block size in cache
 - each block is a power of 2 or blocks with integer division of scalar variables

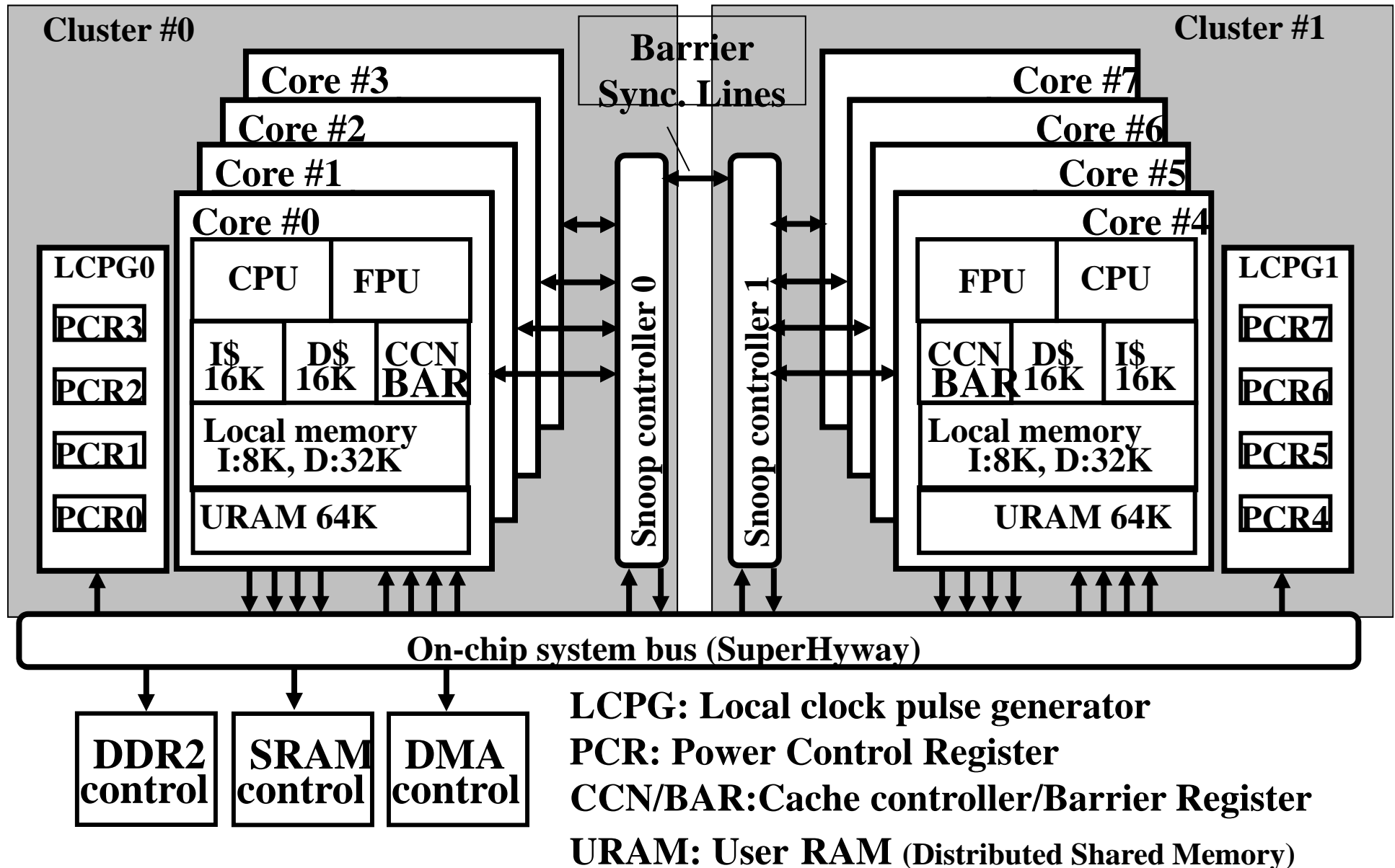


Multi-dimensional Template Arrays for Improving Readability

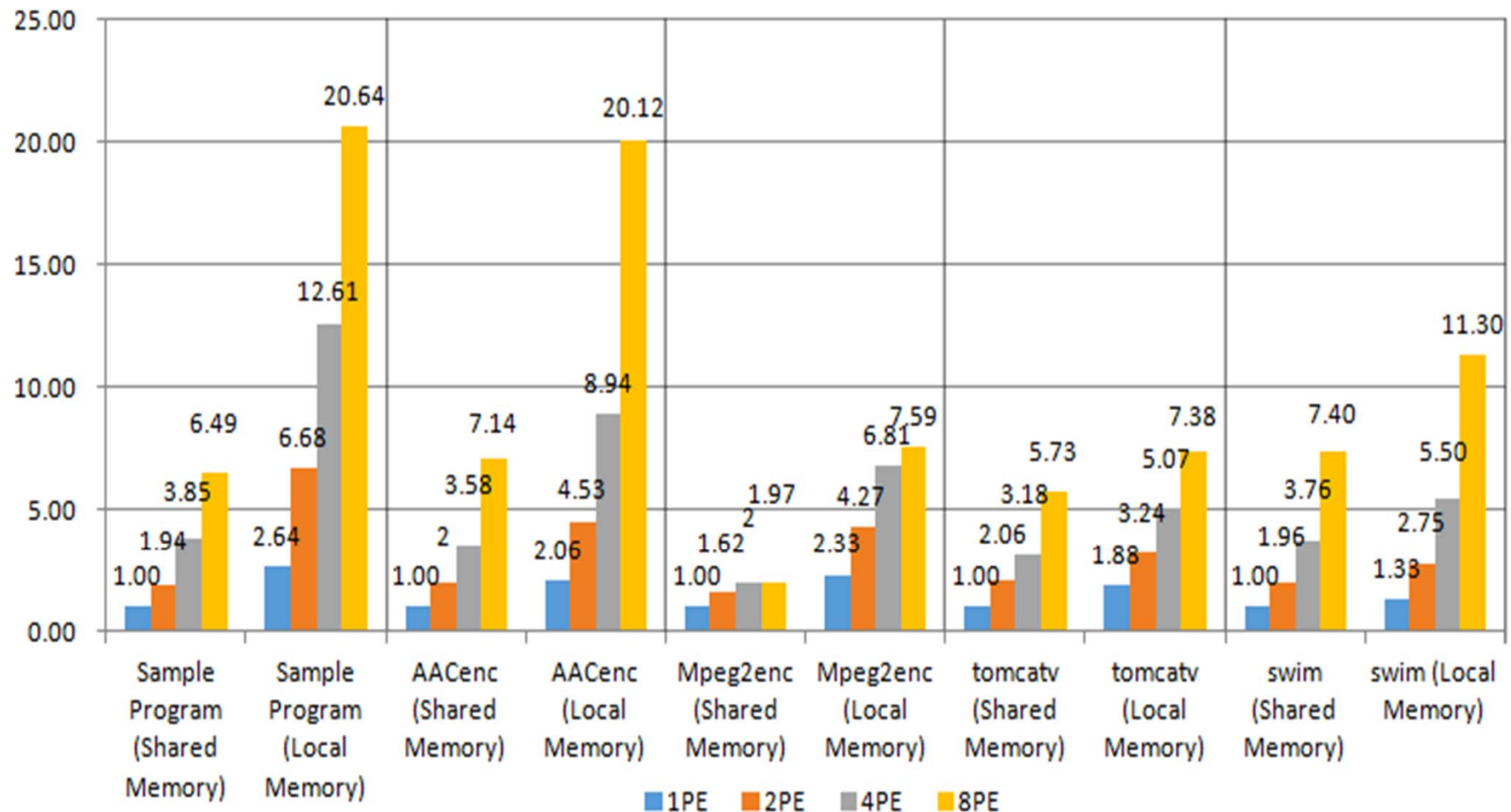
- a mapping technique for arrays with varying dimensions
 - each block on LDM corresponds to multiple empty arrays with varying dimensions
 - these arrays have an additional dimension to store the corresponding block number
 - $TA[Block\#][\]$ for single dimension
 - $TA[Block\#][\][\]$ for double dimension
 - $TA[Block\#][\][\][\]$ for triple dimension
 - ...
- LDM are represented as a one dimensional array
 - without Template Arrays, multi-dimensional arrays have complex index calculations
 - $A[i][j][k] \rightarrow TA[offset + i' * L + j' * M + k']$
 - Template Arrays provide readability
 - $A[i][j][k] \rightarrow TA[Block\#][i'][j'][k']$



8 Core RP2 Chip Block Diagram



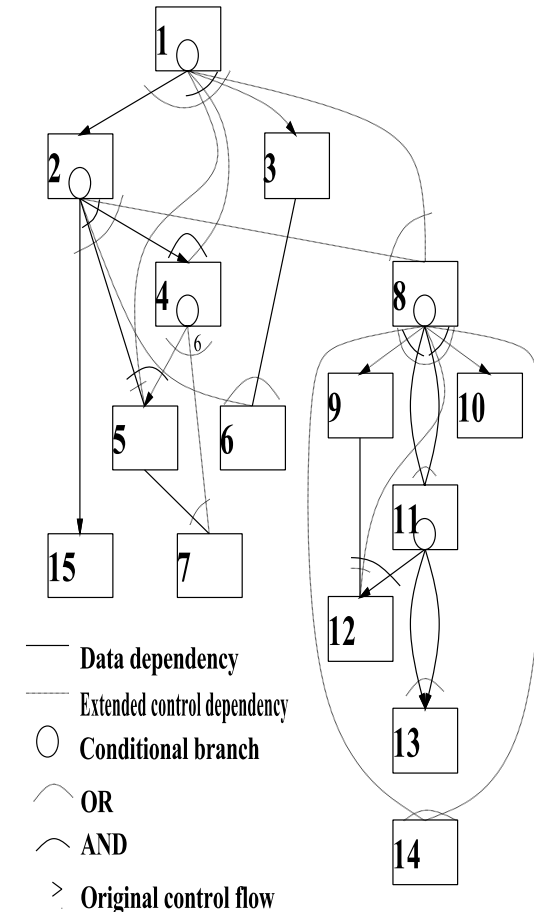
Speedups by the Local Memory Management Compared with Utilizing Shared Memory on Benchmarks Application using RP2



20.12 times speedup for 8cores execution using local memory against sequential execution using off-chip shared memory of RP2 for the AACenc

Software Coherence Control Method on OSCAR Parallelizing Compiler

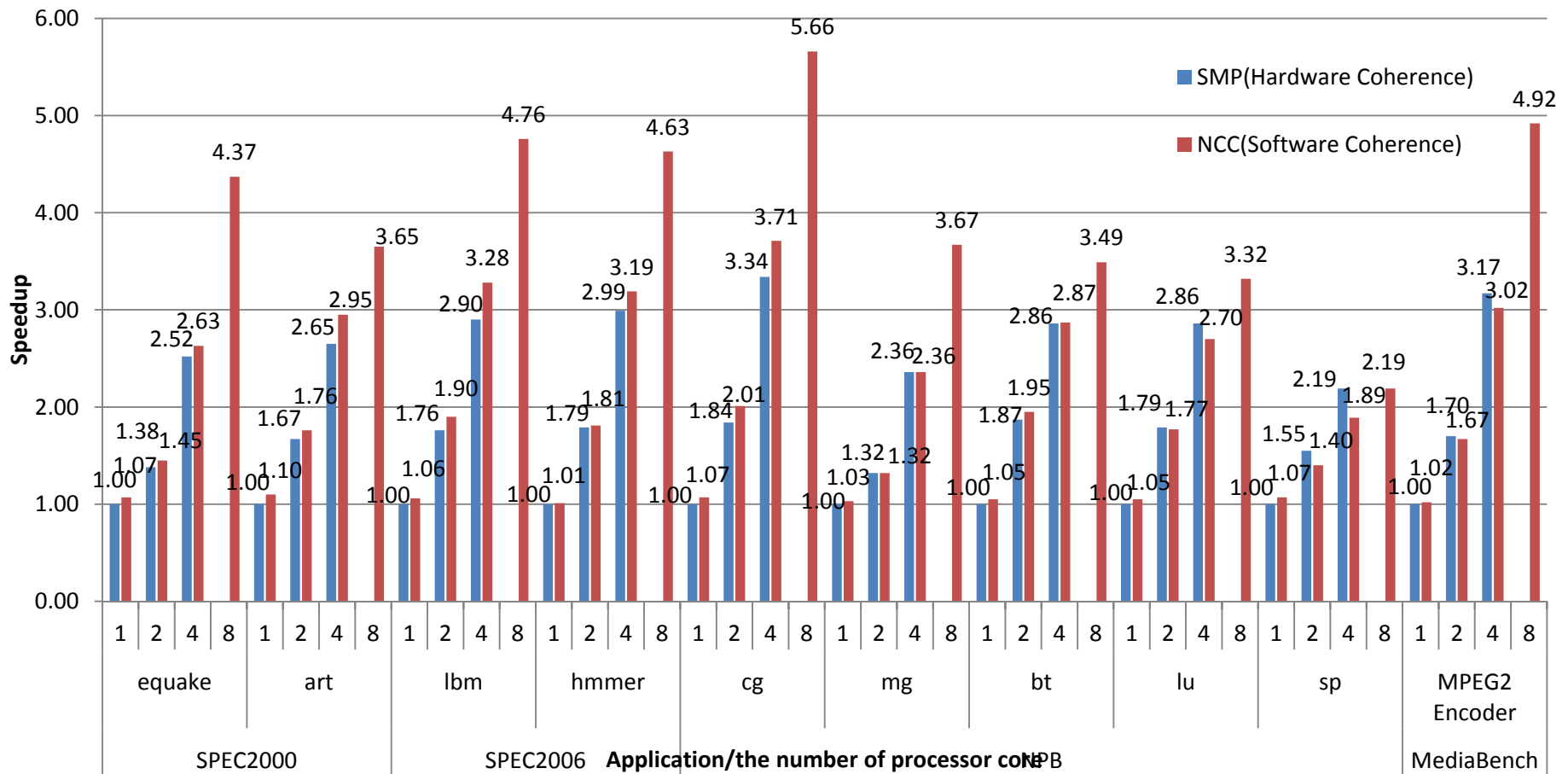
- Coarse grain task parallelization with **earliest condition analysis** (control and data dependency analysis to detect parallelism among coarse grain tasks).
- OSCAR compiler automatically controls coherence using following simple program restructuring methods:
 - To cope with stale data problems:
 - ◆ **Data synchronization by compilers**
 - To cope with false sharing problem:
 - ◆ **Data Alignment**
 - ◆ **Array Padding**
 - ◆ **Non-cacheable Buffer**

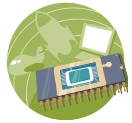


**MTG generated by
earliest executable
condition analysis**

Automatic Software Coherent Control for Manycores

Performance of Software Coherence Control by OSCAR Compiler on 8-core RP2





Future Multicore Products



Next Generation Automobiles

- Safer, more comfortable, energy efficient, environment friendly
- Cameras, radar, car2car communication, internet information integrated brake, steering, engine, motor control

Smart phones



- From everyday recharging to less than once a week
- Solar powered operation in emergency condition
- Keep health

Advanced medical systems



Cancer treatment, Drinkable inner camera

- Emergency solar powered
- No cooling fan, No dust, clean usable inside OP room



Personal / Regional Supercomputers



Solar powered with more than 100 times power efficient : FLOPS/W

- Regional Disaster Simulators saving lives from tornadoes, localized heavy rain, fires with earthquakes

Summary

- To get speedup and power reduction on homogeneous and heterogeneous multicore systems, collaboration of architecture and compiler will be more important.
- Automatic Parallelizing and Power Reducing Compiler has succeeded speedup and/or power reduction of scientific applications including “Earthquake Wave Propagation”, medical applications including “Cancer Treatment Using Carbon Ion”, and “Drinkable Inner Camera”, industry application including “Automobile Engine Control”, and “Wireless communication Base Band Processing” on various multicores.
 - For example, the automatic parallelization gave us 110 times speedup for “Earthquake Wave Propagation Simulation” on 128 cores of IBM Power 7 against 1 core, 327 times speedup for “Heavy Particle Radiotherapy Cancer Treatment” on 144cores Hitachi Blade Server using Intel Xeon E7-8890 , 1.95 times for “Automobile Engine Control” on Renesas 2 cores using SH4A or V850, 55 times for “JPEG-XR Encoding for Capsule Inner Cameras” on Tiler 64 cores Tile64 manycore.
- In automatic power reduction, consumed powers for real-time multi-media applications like Human face detection, H.264, mpeg2 and optical flow were reduced to 1/2 or 1/3 using 3 cores of ARM Cortex A9 and Intel Haswell and 1/4 using Renesas SH4A 8 cores against ordinary single core execution.
- For more speedup and power reduction, we have been developing a new architecture/compiler co-designed multicore with vector accelerator based on vector pipelining with vector registers, chaining, load-store pipeline, advanced DMA controller without need of modification of CPU instruction set.